The Helpful/Naughty Wizard

Minimum experience: Grades 1+, 2nd year using ScratchJr, 1st quarter or later

At a Glance

Overview and Purpose

Coders use a variety of blocks and sprites to create a story about a wizard who either helps people or plays pranks on them. The purpose of this project is to reinforce understanding of message blocks to give the appearance of switching costumes.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will use message blocks to make sprites look like they are switching costumes **Question:** ● How can we use message blocks to make sprites look like they are switching costumes?	**Statement:** ● I will create a story that uses message blocks to make a wizard appear to use magic. **Question:** ● How can we create a story that uses message blocks to make a wizard appear to use magic?
Main standard(s):	**Reinforced standard(s):**
1A-AP-10 Develop programs with sequences and simple loops, to express ideas or address a problem. ● Programming is used as a tool to create products that reflect a wide range of interests. Control structures specify the order in which instructions are executed within a program. Sequences are the order of instructions in a program. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. If the commands to program a robot are not in the correct order, the robot will not complete the task desired. Loops allow for the repetition of a sequence of code multiple times. For example, in a program to show the life cycle of a butterfly, a loop could be combined with move commands to allow continual but controlled movement of the character. (source) **1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes. ● Creating a plan for what a program will do clarifies the steps that will be needed to create a program and can be used to check if a program is correct. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer,	**1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks. ● Composition is the combination of smaller tasks into more complex tasks. Students could create and follow algorithms for making simple foods, brushing their teeth, getting ready for school, participating in clean-up time. (source) **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. ● Decomposition is the act of breaking down tasks into simpler tasks. Students could break down the steps needed to make a peanut butter and jelly sandwich, to brush their teeth, to draw a shape, to move a character across the screen, or to solve a level of a coding app. (source) **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. ● Algorithms or programs may not always work correctly. Students should be able to use various strategies, such as changing the sequence of the steps, following the algorithm in a step-by-step manner, or trial and error to fix problems in algorithms and programs. (source) **1A-AP-15** Using correct terminology, describe steps taken and

to illustrate what their program will do. Students at this stage may complete the planning process with help from their teachers. (source)	choices made during the iterative process of program development. ● At this stage, students should be able to talk or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. This could be done using coding journals, discussions with a teacher, class presentations, or blogs. (source)

Practices and Concepts

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

Main practice(s):	Reinforced practice(s):
Practice 4: Developing and Using Abstractions ● "Abstractions are formed by identifying patterns and extracting common features from specific examples to create generalizations. Using generalized solutions and parts of solutions designed for broad reuse simplifies the development process by managing complexity." (p. 78) ● **P4.4.** Model phenomena and processes and simulate systems to understand and evaluate potential outcomes. (p. 79) **Practice 5: Creating computational artifacts** ● "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) ● **P5.1.** Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations. (p. 80) ● **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80)	**Practice 6: Testing and refining computational artifacts** ● "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) ● **P6.1.** Systematically test computational artifacts by considering all scenarios and using test cases." (p. 81) ● **P6.2.** Identify and fix errors using a systematic process. (p. 81) **Practice 7: Communicating about computing** ● "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) ● **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):**
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)	**Algorithms** ● "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. As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms." (p. 91)

• **Grade 2 -** "Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions." (p. 96) **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 2 -** "People develop programs collaboratively and for a purpose, such as expressing ideas or addressing problems." (p. 97)	• **Grade 2** - People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow." (p. 96) **Modularity** • "Modularity involves breaking down tasks into simpler tasks and combining simple tasks to create something more complex. In early grades, students 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 2 -** "Complex tasks can be broken down into simpler instructions, some of which can be broken down even further. Likewise, instructions can be combined to accomplish complex tasks." (p. 97)

ScratchJr Blocks	
Primary blocks	Control, Looks, Motion, Triggering
Supporting blocks	End, Sound

Vocabulary	
Event (trigger)	• An action or occurrence detected by a program. Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory. Most modern applications, particularly those that run in Macintosh and Windows environments, are said to be event-driven,because they are designed to respond to events. (source) • The computational concept of one thing causing another thing to happen. (source) • Any identifiable occurrence that has significance for system hardware or software. User-generated events include keystrokes and mouse clicks; system-generated events include program loading and errors. (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)
Sprite	• A media object that performs actions on the stage in a Scratch project. (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, social studies **Example(s):** Instead of creating a project about a nameless wizard, this project could recreate or

	imagine how fictional characters might use magic to help (or play pranks on) others. If this project is modified to focus on a real person who helps (or plays pranks on) others, this project could integrate with history or social studies lessons.
Vocations	Authors, marketers, and media artists are often asked to create a story to sell a product or create a narrative. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

- Project files (both projects)
  - **Video:** Downloading project files (1:04)
- Sample project images

Project Sequence

Preparation (30+ minutes)

Suggested preparation	Resources for learning more

Ensure all devices are plugged in for charging over night.
 	
Customizing this project for your class (20+ minutes): Remix one or both of the project examples to include your own stories about helpful and naughty wizards.
 	
(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.
	• BootUp ScratchJr Tips
	  ○ Videos and tips on ScratchJr from our YouTube channel
	• BootUp Facilitation Tips
	  ○ Videos and tips on facilitating coding classes from our YouTube channel
	• Block Descriptions
	  ○ A document that describes each of the blocks used in ScratchJr
	• Interface Guide
	  ○ A reference guide that introduces the ScratchJr interface
	• Paint Editor Guide
	  ○ A reference guide that introduces features in the paint editor
	• Tips and Hints
	  ○ Learn even more tips and hints by the creators of the app
	• Coding as another language (CAL)
	  ○ A set of curriculum units for K-2 using both ScratchJr and KIBO robotics
	• ScratchJr in Scratch
	  ○ If you're using ScratchJr in Scratch, this playlist provides helpful tips and resources

Getting Started (5+ minutes)

Suggested sequence	Resources, suggestions, and connections
1. Review and demonstration (2+ minutes): 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.	**Practices reinforced:** • Communicating about computing **Video:** Project Preview (3:25) **Video:** Lesson pacing (1:48)

Explain that today we are going to create a story about a wizard who uses magic to either help others or to play pranks. Display and demonstrate one or both of the sample projects (or your own remixed versions). **A note on two project variations:** Because there are two versions of the project available to download or display, you can skip the "naughty" version of the project if you are unsure about whether the project is considered appropriate, and instead only show the "helpful" version of this project. However, the "naughty" version of the project could lead to a discussion on how to not treat other people by asking questions like "how do you think that person feels after they had a prank played on them?" This could also lead to a final page that coders create where the naughty wizard learns a lesson about treating people with respect and using magic to help others. Either select one of the two project variations for the classes you work with or give them the option to choose which project they will work on.	This can include a full class demonstration or guided exploration in small groups or individually. For small group and individual explorations, it might help to set a time limit for exploration before discussing the project. **A note on say blocks:** If you are displaying a sample project with say blocks, it might help to read the text out loud using various voices for each sprite as it is displayed. This strategy might help early/pre-readers, as well as young coders who are new to learning English. In addition, when young coders begin working on their own project, you can encourage them to use speech-to-text or emojis in their own say blocks instead of typing out words (or use recorded sound blocks). **Example review discussion questions:** ● What's something new you learned last time you coded? ○ Is there a new block or word you learned? ● What's something you want to know more about? ● What's something you could add or change to your previous project? ● What's something that was easy/difficult about your previous project?
2. Discuss (3+ minutes): Have coders talk with each other about how they might create a project like the one demonstrated. If coders are unsure, and the discussion questions aren't helping, you can model thought processes: "I noticed the sprite moved around, so I think they used a motion block. What motion block(s) might be in the code? What else did you notice?" Another approach might be to wonder out loud by thinking aloud different algorithms and testing them out, next asking coders "what do you wonder about or want to try?" After the discussion, coders will begin working on their project as a class, in small groups, or at their own pace.	**Practices reinforced:** ● Communicating about computing **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. **Example discussion questions:** ● What would we need to know to make something like this in ScratchJr? ● What kind of blocks might we use? ● What else could you add or change in a project like this? ● What code from our previous projects might we use in a project like this? ● What do you think the code is that makes the magic cloud appear to change how a sprite looks? ● How could a wizard use magic to help other sprites? ○ How could we use code to simulate magic? ○ How will the other sprites respond to the wizard helping them? ● How could a wizard use magic to play pranks on other sprites? ○ How can we use code to simulate magic? ○ How will the other sprites respond to the wizard playing pranks on them?

Project Work (75+ minutes; 3+ classes)

Suggested sequence	Resources, suggestions, and connections
3. Switch costumes (15+ minutes) ***5+ demonstration or review*** Demonstrate or review how to use show and hide blocks with message blocks to make it so when the magic cloud appears that one sprite hides behind it as another sprite shows behind it to make it appear as though the sprite is changing costumes with magic. Emphasize the importance of making sure the sprites are in the same location and that the magic cloud is on top of both sprites. Another option demonstrated in the video and in the example projects involves changing costumes by switching pages to the same backdrop. Although this option works, you have to make sure every sprite is perfectly lined up to do the transition (which may require some iteration to get the timing correct if the sprite is moving). Each method can work really well depending on the project and purpose. ***10+ minute application and exploration*** Encourage coders to try and experiment with using one of the two methods for making a sprite appear to change costumes using a magic cloud. Facilitate by walking around and asking questions about their processes.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem ● **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Video:** Switching costumes (3:20) **Sample questions:** ● When might you switch costumes in your projects? ○ What kind of blocks might we use when switching costumes? ● How will you use show and hide blocks with trigger message blocks in your project?
4. Create a storyboard (15+ minutes): Either hand out paper with at least four different quadrants (one for each page in ScratchJr), use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create. It may help to model this process with a separate set of random ideas. Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them. 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 step (creating); otherwise they can continue to think through and work on their storyboard. **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:** ● **1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes **Practices reinforced:** ● Creating computational artifacts **Concept reinforced:** ● Program development **Resource:** Example storyboard templates **Suggested storyboard questions:** ● What will happen in your story? ○ Will your wizard be helpful or naughty? ■ What will the wizard learn by the end of the story? ○ How will the wizard use magic? ■ What will happen to sprites and backdrops the wizard uses magic on? ● How many pages will you have? ○ What backdrop will you use for each page? ○ What sprites will we see on each page? ○ When will you go to the next page? ● What will the sprites do on each page? ○ What kind of code might we use to do that? ● What are all of the ways we can interact with the story? ○ Which pages will have user interaction?

	○ In each of these ways we can interact with the story, how might we use code to create that interaction? **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. **Note:** This process may take significantly longer than 15 minutes if storyboarding many different sprites in each page. In general, it is best to keep things simple when first creating a project, then adding more complexity if time permits.
4. Create a story about a helpful or naughty wizard (45+ minutes): Ask coders to create a project based on the storyboards they created. Facilitate by walking around and asking questions and encouraging coders to try out new blocks. **Note:** The more pages, the more time you can spend on each project. This project could take several days.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts ● Developing and using abstractions **Concepts reinforced:** ● Algorithms ● Control **Suggested questions:** ● What sounds might we hear in each backdrop? ● What do you like about your project? a. What do you want to change? ● How could we add even more to your story than what's in your storyboard?

Assessment

Standards reinforced:

- **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development

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 Assessment *of* Learning	**Formative** Assessment *for* Learning	**Ipsative** 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.	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	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.

For example, ask the following after a project:

- Can coders debug the debugging exercises?
- Did coders create a project similar to one of the project previews?
    - **Note:** The project preview and sample projects are not representative of what all grade levels should seek to emulate. They are meant to generate ideas, but expectations should be scaled to match the experience levels of the coders you are working with.
- Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes?
- Can coders explain how their project is similar to their storyboard?
- Can coders explain how they used show and hide blocks with trigger message blocks in their project?
    - Can coders explain how the magic cloud hides the "costume change?"
- Did coders create a story about a wizard using magic with at least ## different sprites that change costume?
    - Choose a number appropriate for the coders you work with and the amount of time available.
- Did coders use at least ## pages in their project?
    - Choose a number appropriate for the coders you work with and the amount of time available.
    - Can coders explain when/how the project will switch pages?

assessment.

For example, ask the following while coders are working on a project:

- What are three different ways you could change that sprite's algorithm?
- What happens if we change the order of these blocks?
- What could you add or change to this code and what do you think would happen?
- How might you use code like this in everyday life?
- See the suggested questions throughout the lesson and the assessment examples for more questions.

For example, ask the following after a project:

- How is this project similar or different from previous projects?
- What new code or tools were you able to add to this project that you haven't used before?
- How can you use what you learned today in future projects?
- What questions do you have about coding that you could explore next time?
- See the reflection questions at the end for more suggestions.

Extended Learning

Project Extensions

Suggested extensions	Resources, suggestions, and connections
Creating a transition overlay (Advanced) (15+ minutes) ***5+ minute intro demonstration*** Demonstrate how other projects (e.g., A Day at the Beach and The Story of the Stunky Robot) use a transition overlay in the project. Use the video to learn the following process: 1. Demonstrate how to create a blank sprite that is all one color. 2. Next, demonstrate creating another blank sprite of the same color, but add your text on one side of the sprite (e.g., something like "The End" or "2 Hours Later"). 3. Use grow blocks to make the sprites as large as possible. 4. Drag the blank sprite over to one side of the screen, then drag the sprite with the text on the other side of the screen (the text should now appear centered). 5. Use show and hide blocks in an algorithm attached to either the start on green flag or start on message blocks to make both overlays show or hide. ***10+ minute application and and 1-on-1 facilitating*** Quickly review each of the steps for creating a transition overlay (1 - create a blank sprite; 2 - create a sprite with text; 3 - make the sprites big; 4 - position the sprites on the screen; and 5 - trigger the transitions with code). Leave your example code on the screen and give coders time to replicate the process for creating a transition overlay. Facilitate 1-on-1 as needed, but encourage coders to work together.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem ● **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Video:** Creating a transition overlay (3:38) **Resource:** Paint Editor Guide **Facilitation suggestion:** This process might be a little complicated for younger coders; however, you can group coders together in small groups to collaboratively create their own transition screen. If coders complete their stinky robot early, encourage them to begin working on their storyboard.
Advanced reverse engineering even more ideas (15+ minutes each page): ***1 minute intro demonstration*** Demonstrate one of the following example pages on the board without displaying the code: **Helpful wizard** **Page 1** ● Ball, Colton, Dog, Helpful wizard, Magic cloud, and Stick **Page 2**	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem ● **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. ● **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. **Practices reinforced:** ● Communicating about computing ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:**

- Cold food, Father, Food, Helpful wizard, and Magic cloud

Page 3
- Helpful wizard, Magic cloud, Snow cat, and Teen

Page 4
- Cat, Helpful wizard, Magic cloud, and Teen

Naughty wizard

Page 1
- Colt, Colton, Dog, Magic cloud, and Naughty wizard

Page 2
- Crab, Father, Food, Magic cloud, and Naughty wizard

Page 3
- Cat, Magic cloud, Naughty wizard, and Teen

Page 4
- Magic cloud, Naughty wizard, Snow cat, and Teen

9+ minute reverse engineering and peer-to-peer coaching

Ask coders to see if they can figure out how to use their code blocks to create algorithms that recreates the entire page. Facilitate by walking around and asking guiding questions.

1 minute explanation demonstration

If coders figured out how to get their sprites to do something similar, have them document in their journal, share with a partner, or have a volunteer show the class their code and thought processes that led to the code. Otherwise, reveal the code, walk through each step of the algorithm, and explain any new blocks.

4+ minute application and exploration

Encourage coders to try something similar, and leave your code up on display while they work. Facilitate by walking around and asking questions about how coders might change their code so it's not the same as yours.

- Algorithms
- Control

Video: Suggestions for reverse engineering (4:25)

Helpful tip: You could add code to loop the page coders are trying to reverse engineer. To do this, add code on the next page that immediately switches back to this page (i.e., on the next page attach a go to page block to a start on green flag block to cycle back to the previous page).

Note: It is not recommended to show more than one page at a time, but to show one page, give time for application and exploration, show another page, give time for application and exploration, etc. This process could take multiple classes. Also, some of these examples may be difficult for young coders, so go slow and encourage copying and modifying code as it's good practice.

Alternative suggestion: If reverse engineering is too difficult for the coders you work with, you could display the source code and have coders predict what will happen.

Suggested guiding questions:
- What kind of blocks do you think you might need to do something like that?
- Do you see a pattern where we might use a repeat?
- What trigger blocks do you think I used for that sprite?
- Did I use one trigger block or more than one?
    - What makes you think that?

Potential discussion: There is not always one way to recreate something with code, so coders may come up with alternative solutions to your own code. When this occurs, it can open up an interesting discussion or journal reflection on the affordances and constraints of such code.

Suggested application and exploration questions:
- What other code blocks could you use?
- What other sprites might use similar code?

Adding even more (5+ minutes):

If time permits, encourage coders to explore what else they can create in ScratchJr. Although future lessons will explore different features and blocks, early experimentation should be encouraged.

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

Standards reinforced:
- **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem

Practices reinforced:
- Testing and refining computational artifacts
- Creating computational artifacts

Concepts reinforced:
- Algorithms
- Control

taking photos. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.	**Suggested questions:** ● What else can you do with ScratchJr? ● What do you think the other blocks do? a. Can you make your sprites do ____? ● What other sprites might we use in a project with magic? ● What other sounds might we hear when a wizard uses magic? ● Can you customize how your sprites look?
Similar projects: Have coders explore the sample projects built into ScratchJr (or projects from other coders), and ask them to find code similar to what they worked on today.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms **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," or "find three projects that each do one thing you would like to add to your project." **Generic questions:** ● How is this project similar (or different) to something you worked on today? ● What blocks did they use that you didn't use? a. What do you think those blocks do? ● What's something you like about their project that you could add to your project? ● How might we change the backdrop of this project? ● What other sound or looks blocks might we use in this project? ● Can you turn this project into a short story? ● Could we add a magical wizard to this project?

Differentiation

Less experienced coders	More experienced coders
ScratchJr is simple enough that it can be picked up relatively quickly by less experienced coders. However, for those who need additional assistance, pair them with another coder who feels comfortable working cooperatively on a project. Once coders appear to get the hang of using ScratchJr, they can begin to work independently.	Because ScratchJr is not inherently difficult, experienced coders might get bored with simple projects. To help prevent boredom, ask if they would like to be a "peer helper" and have them help out their peers when they have a question. If someone asks for your help, guide them to a peer helper in order to encourage collaborative learning. Another approach is to encourage experienced coders to experiment with their code or give them an individual challenge or quest to complete within a timeframe.

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
Helpful wizard	**Standards reinforced:**

Why doesn't the dog bark and say "Woof!" before the wizard leaves?

- We need to change the order of the messages so that the dog jumps and barks before the wizard leaves, otherwise the code will never run

Why doesn't the wizard leave after warming up the food?

- Because we need to send the red message to run the code that makes the wizard leave

Why doesn't anything happen after the wizard makes the weather warmer (i.e., when it goes to page four)?

- We need the Teen's code to start on green flag rather than on orange message

Naughty wizard

Why doesn't Colton turn into a Colt?

- Our message colors are reversed, so we don't see Colton appear before it switches pages (i.e., the blue message should be red and the red message should be blue)

Why does the Father sprite run away before the food turns into a crab?

- We need to send an orange message before the father sprite runs away

Why does the wizard say "Not for long" and then the story stops on page three?

- The app crashes when the blue message is used to call the blue message again (a process called "recursion"). To fix this, we need to make sure we have the correct message color (purple) to call the big magic cloud

ScratchJr Debugging List

- **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops

Practices reinforced:

- Testing and refining computational artifacts

Concepts reinforced:

- Algorithms
- Control

Display one of the debugging exercises and ask the class what they think we need to fix in our code to get our project to work correctly. Think out loud what might be wrong (e.g., did I use the wrong trigger block, did I forget to repeat something, did I put a block in the wrong place, am I missing blocks, etc.). Ask the class to talk with a neighbor how we might fix the code. Have a volunteer come up to try and debug the code (or demonstrate how). Repeat with each debugging exercise.

Unplugged Lessons and Resources

Standards reinforced:

- **1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks

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

Reflection and Sharing

Reflection 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.

Sample reflection questions or journal prompts:

- How did you use computational thinking when creating your project?
- 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?
- How did you help other coders with their projects?
    - What did you learn from other coders today?
- What's a fun algorithm you created today?
- What's something you could create next time?
- What questions do you have about coding?
    - What was challenging today?
- How did your storyboard help you plan out your project?
    - What did you end up adding that you didn't originally plan for in your storyboard?
- What did you learn about making a sprite appear to switch costumes in ScratchJr?
- How might you add message blocks to projects you already created?
    - How could you use message blocks to trigger more than one sprite at the same time?
    - How could you send and receive message blocks in the same sprite?
    - What questions do you have about message blocks?
- More sample prompts (may need adapting for younger coders)

Sharing suggestions

Standards reinforced:

- **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development

Practices reinforced:

- Communicating about computing
- Fostering an inclusive culture

Concepts reinforced:

- Algorithms
- Control
- Modularity
- 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.