Animal House

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

At a Glance

Overview and Purpose

Coders use a variety of blocks and sprites to create their own interactive diorama about animals on a farm. The purpose of this project is to introduce coders to the sound blocks.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will learn how to trigger sounds and other algorithms when sprites are tapped. **Question:** ● How can we trigger sounds and other algorithms when sprites are tapped?	**Statement:** ● I will create an interactive diorama that responds to a user. **Question:** ● How can we create an interactive diorama that responds to a user?
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-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

	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 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) ● **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80) ● **P5.3.** Modify an existing artifact to improve or customize it. (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) ● **Grade 2 -** "Computers follow precise sequences of instructions that automate tasks. Program execution	**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)

can also be nonsequential by repeating patterns of instructions and using events to initiate instructions." (p. 96)	• **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)

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

Vocabulary	
Algorithm	• A step-by-step process to complete a task. (source) • A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point. (source)
Debugging	• The process of finding and correcting errors (bugs) in programs. (source) • To find and remove errors (bugs) from a software program. Bugs occur in programs when a line of code or an instruction conflicts with other elements of the code. (source)
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)
Simulation	• Imitation of the operation of a real-world process or system. (source) • The process of imitating a real phenomenon with a set of mathematical formulas. Advanced computer programs can simulate weather conditions, chemical reactions, atomic reactions, even biological processes. In theory, any phenomena that can be reduced to mathematical data and equations can be simulated on a computer. In practice, however, simulation is extremely difficult because most natural phenomena are subject to an almost infinite number of influences. One of the tricks to developing useful simulations, therefore, is to determine which are the most important factors. (source)
Sprite	• A media object that performs actions on the stage in a Scratch project. (source)
More vocabulary words from CSTA	• Click here for more vocabulary words and definitions created by the Computer Science Teachers Association

Connections	
Integration	**Potential subjects:** Media arts, science **Example(s):** This project could connect with science classes as it models the motions and sounds of creatures within an environment or system. Rather than making this project about the farm, coders could explore other creatures within other environments.
Vocations	Scientists and researchers often create models or simulations of environments in order to better understand the processes and systems at play. Click here to visit a website dedicated to exploring

	potential careers through coding.

Resources

- Sample project file
  - **Video:** Downloading project files (1:04)
- Sample project images

Project Sequence

Preparation (At least one day prior)

Suggested preparation	Resources for learning more

Ensure all devices are plugged in for charging over night and prepare headphones for each device.
 	
(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 (10+ minutes)

Suggested sequence	Resources, suggestions, and connections
1. Review and demonstration (8+ 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. Explain to the class we are going to create a project about animals on a farm that make sounds when we press them. Demonstrate the sample project or your own farm diorama, but don't display the code. Review how to open ScratchJr, create a new project, delete Scratch Cat, and select an outside background.	**Practices reinforced:** • Communicating about computing **Video:** Project Preview (0:50) **Video:** Lesson pacing (1:48) **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?

Review how to add in a sprite by looking at the different animals and objects we might find, then select and add in an animal. Demonstrate how to record a sound and then trigger a recorded sound block using a start on tap trigger.	
2. Discuss (2+ 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 kind of sprites might we see on a farm? ○ What kind of code might they have?

Project Work (30+ minutes; 2+ classes)	
Suggested sequence	**Resources, suggestions, and connections**
3. Create a farm sound project (20+ minutes): Ask coders to create a project about farm sounds and animals. Facilitate by walking around and asking questions and encouraging coders to try out new blocks.	**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
4. Reverse engineering even more ideas (10+ minutes each): ***1 minute intro demonstration*** Demonstrate one of the following example sprites on the board without displaying the code (they are in order of complexity): ● Pig breaking out of a fence ● Barn sounds ● Horse sounds ● Chicken dance ● Pig rolling in mud ***4+ 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 an algorithm that makes a sprite do	**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:**

something similar to what was demonstrated. Facilitate by walking around and asking guiding questions.

1 minute explanation demonstration
If coders figured out how to get their sprite 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. If coders haven't figured it out, demonstrate how to record a sound and add it to their project.

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)

Note: It is not recommended to show each of these ideas at once, but to show one idea, give time for application and exploration, show another idea, 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 encouraging 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?

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

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. **For example, ask the following after a project:** • Can coders debug the	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. **For example, ask the following while coders are working on a project:**	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:** • How is this project similar or

debugging exercises? ● Did coders create a project similar to the project preview? ○ **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. ● Can coders explain how to record a sound in ScratchJr? ● Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes? ● Did coders create a farm simulation with at least ## different sprites with different algorithms that include recorded sounds? ○ Choose a number appropriate for the coders you work with and the amount of time available.	● 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.	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
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 taking photos. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.	**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 **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 can you add to your project?

	• What else might we find on a farm? • What other sounds might we hear when the green flag is pressed? • What other simulated spaces might we create? a. How might we create simulations related to what we are learning in other classes?
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?

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 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, and remind them the helper is "hands off" and does not take over working on another person's project. 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 (e.g., a reverse engineering challenge).

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
Why does the barn get bigger and not bigger, then smaller? • The second grow block should be a shrink block	**Standards reinforced:** • **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops **Practices reinforced:**

Why does the chicken play a pop sound and not the recorded sound? ● The pop block sound be a play recorded sound block Why does the horse play the same recording twice instead of two different recordings? ● The second play recorded sound block should have a number 2 and not a number 1 ScratchJr Debugging List	● 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.). Explain that mistakes in code are called bugs. To fix the bugs students need to find the bug and get rid of it. This is called debugging. Ask the class to talk with a neighbor about 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.

Suggested unplugged lessons:

1. Human crane
    a. A lesson where kids create and test crane algorithms that move blocks from one bowl to another.
2. How to train your robot
    a. The game works as follows: every kid is turned into a “robot master” and their mom or dad becomes their “robot”. I give each kid a “Robot Language Dictionary” and explain to them that this is the language their robot understands. The dictionary has symbols for “move left leg forward”, “turn left”, “grab”, “drop” etc.

List of 100+ unplugged lessons and resources

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. **Sample reflection questions or journal prompts:** ● How did you use computational thinking when creating your project?	**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

● 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 using sounds make your project more interesting? ● Should everything have a sound? ○ Why or why not? ● How can you use sound to make your project more natural for other users? ● More sample prompts (may need adapting for younger coders)	**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.