Don't Catch Mine!

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

At a Glance

Overview and Purpose

Coders create a competitive game that keeps track of each player's score. The purpose of this project is to collaboratively reinforce variables to keep track of each player's score. ***Note:*** *this project builds off the understandings introduced in "Food Catcher."*

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will learn/review how to keep multiple scores with different variables. ● I will learn how to collaborate with others to create a coding project. **Question:** ● How can we keep multiple scores with different variables? ● How can we collaborate with others to create a coding project?	**Statement:** ● I will collaborate with others to create a multiplayer catching game that keeps track of multiple scores. **Question:** ● How can we collaborate with others to create a multiplayer catching game that keeps track of multiple scores?
Main standard(s):	**Reinforced standard(s):**
1B-AP-10 Create programs that include sequences, events, loops, and conditionals ● 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-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. ● Different algorithms can achieve the same result, though sometimes one algorithm might be most appropriate for a specific situation. Students should be able to look at different ways to solve the same task and decide which would be the best solution. For example, students could use a map and plan multiple algorithms to get from one point to another. They could look at routes suggested by mapping software and change the route to something that would be better, based on which route is shortest or fastest or would avoid a problem. Students might compare algorithms that describe how to get ready for school. Another example might be to write different algorithms to draw a regular polygon and determine which algorithm would be the easiest to modify or repurpose to draw a different polygon. (source)

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-11 Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process.

- 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)

1B-AP-16 Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development.

- Collaborative computing is the process of performing a computational task by working in pairs or on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Students should take turns in different roles during program development, such as note taker, facilitator, program tester, or “driver” of the computer. (source)

1B-AP-09 Create programs that use variables to store and modify data.

- Variables are used to store and modify data. At this level, understanding how to use variables is sufficient. For example, students may use mathematical operations to add to the score of a game or subtract from the number of lives available in a game. The use of a variable as a countdown timer is another example. (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)

1B-AP-15 Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended.

- 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)

1B-AP-17 Describe choices made during program development using code comments, presentations, and demonstrations.

- 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 2: Collaborating around computing ● "Collaborative computing is the process of performing a computational task by working in pairs and on teams. Because it involves asking for the contributions and feedback of others, effective collaboration can lead to better outcomes than working independently. Collaboration requires individuals to navigate and incorporate diverse perspectives, conflicting ideas, disparate skills, and distinct personalities. Students should use collaborative tools to effectively work together and to create complex artifacts." (p. 75)	**Practice 1: Fostering an inclusive computing culture** ● "Building an inclusive and diverse computing culture requires strategies for incorporating perspectives from people of different genders, ethnicities, and abilities. Incorporating these perspectives involves understanding the personal, ethical, social, economic, and cultural contexts in which people operate. Considering the needs of diverse users during the design process is essential to producing inclusive computational products." (p. 74) ● **P1.1.** Include the unique perspectives of others and reflect on one’s own perspectives when designing and developing computational products. (p. 74)

- **P2.1.** Cultivate working relationships with individuals possessing diverse perspectives, skills, and **personalities. (p. 75)**
- **P2.2.** Create team norms, expectations, and equitable workloads to increase efficiency and effectiveness (p. 76)

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)

- **P1.2.** Address the needs of diverse end users during the design process to produce artifacts with broad accessibility and usability. (p. 74)

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)
- **P7.3.** Articulate ideas responsibly by observing intellectual property rights and giving appropriate attribution. (p. 83)

Main concept(s):	Reinforced concept(s):

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 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)

Variables

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 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

• "Computer programs store and manipulate data using variables. In early grades, students learn that different types of data, such as words, numbers, or pictures, can be used in different ways. As they progress, students learn about variables and ways to organize large collections of data into data structures of increasing complexity." (p. 91) • **Grade 5 -** "Programming languages provide variables, which are used to store and modify data. The data type determines the values and operations that can be performed on that data." (p. 103)	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, Events, Motion, My Blocks, Variables
Supporting blocks	Looks, Operators, Sound

Vocabulary	
Conditional	• A feature of a programming language that performs different computations or actions depending on whether a programmer-specified Boolean condition evaluates to true or false. (A conditional could refer to a conditional statement, conditional expression, or conditional construct.) (source) • Referring to an action that takes place only if a specific condition is met. Conditional expressions are one of the most important components of programming languages because they enable a program to act differently each time it is executed, depending on the input. Most programming languages use the word if for conditional expressions. For example, the conditional statement: "if x equals 1 exit" directs the program to exit if the variable x is equal to 1. (source) • The computational concept of making decisions based on conditions (e.g., current variable values). (source)
Iterative	• Involving the repeating of a process with the aim of approaching a desired goal, target, or result (source) • 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) • 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)
Variable	• A symbolic name that is used to keep track of a value that can change while a program is running. Variables are not just used for numbers; they can also hold text, including whole sentences (strings) or logical values (true or false). A variable has a data type and is associated

	with a data storage location; its value is normally changed during the course of program execution. (source) • Variables play an important role in computer programming because they enable programmers to write flexible programs. Rather than entering data directly into a program, a programmer can use variables to represent the data. Then, when the program is executed, the variables are replaced with real data. This makes it possible for the same program to process different sets of data. (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:** Health science, math, media arts, physical education, science **Example(s):** This project could be paired with health, physical education, and science lessons that explore how certain types of food are healthy or unhealthy for different animals. For example, pairing with lessons on healthy snacks vs unhealthy snacks where a player earns points for collecting healthy snacks and loses points for collecting unhealthy snacks. This project could also connect with math standards because of the connections with X and Y coordinates on a coordinate plane, as well as the use of variables. 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	There are a wide range of careers in game development that involve coding. For example, coding character movement, player controls, particle and game physics, random world or object generators, sound synthesis, game engines and tools, localization, performance and server optimization, etc. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

- Example project
- Remix project
- Video walkthroughs
- Project files

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 sprites, challenges, and algorithms.
 	
(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 Scratch Tips
	  ○ Videos and tips on Scratch from our YouTube channel
	• BootUp Facilitation Tips
	  ○ Videos and tips on facilitating coding classes from our YouTube channel
	• Scratch Starter Cards
	  ○ Printable cards with some sample starter code designed for beginners
	• ScratchEd

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.	○ A Scratch community designed specifically for educators interested in sharing resources and discussing Scratch in education ● Scratch Help ○ This includes examples of basic projects and resources to get started ● Scratch Videos ○ Introductory videos and tips designed by the makers of Scratch ● Scratch Wiki ○ This wiki includes a variety of explanations and tutorials

Getting Started (6-10+ 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. Explain that today we are going to create a multiplayer catching game where each player scores points if you collect your own sprites, but lose points if you collect the other player's sprites. Display and demonstrate the sample project (or your own remixed version).	**Practices reinforced:** ● Communicating about computing **Video:** Project Preview (1:34) **Video:** Lesson pacing (1:48) 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. **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 divide into small groups (preferably in pairs) and begin working on their project.	**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 Scratch? ● 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 in a food catching game?
	  ○ What kind of code might they have?
	● When should you score points and when should you lose points in a game like this?
	  ○ What happens if you lose too many points?
	  ○ When might the game end of will it just keep getting harder and harder?

3. Remix the original project (1-5+ minutes): If not yet comfortable with logging in, review how to log into Scratch and remix this project. 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. **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. **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 (70-140+ minutes; 2+ classes)	
Suggested sequence	**Resources, suggestions, and connections**
4. Review or introduce the Food Catcher project (10-80+ minutes) If coders have not completed the Food Catcher project, spend a couple of classes going through the project work and project extensions to learn how to use variables to create a catching game. Otherwise, have coders open their previous catching game with a partner and give them about ten minutes to read through their prior comments to review what they created and compare their code with their partner's/group's project.	**Standards reinforced:** ● **1B-AP-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-16** Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development. **Practices reinforced:** ● Communicating about computing ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity

	Suggested questions: ● How did your comments from your previous project help remind you what your code does? ○ What could you change to improve your comments? ● How do you and your partner's/group's projects differ? ○ How are they similar?
5. Create a multiplayer catching game (40+ minutes, or the majority of the class): Ask partners/groups to take what they learned in the Food Catcher project and turn it into a multiplayer catching game. Facilitate by walking around and asking questions and encouraging coders to try out new project extensions and review what they previously learned about variables. **Facilitation Suggestion:** Make sure one coder isn't in control of the mouse the entire time. You can set a timer to alternate who "drives" the mouse and who "navigates" the program development, or you could encourage coders to make all adjustments on their own sprites (i.e., one coder adds/changes all of the code in the Player 1 and Ball 1 sprites, and another coder adds/changes all of the code in the Player 2 and Ball 2 sprites).	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **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. ● **1B-AP-16** Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development. **Practices reinforced:** ● Communicating about computing ● Creating computational artifacts ● Fostering an inclusive computing culture ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Video**: Challenge demonstration (1:19) **Suggested questions:** ● If we change the colors of the sprites will everyone be able to distinguish between them? a. What about people who are color blind? ● When can we use the same variable for both players (e.g., speed) and when do we need separate variables for each player (e.g., score)? ● How can you create controls for two players? ● How can you keep score for two different players? **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.
6. Play testing (20+ minutes, or an entire class) ***5+ minute play testing*** Give groups a few minutes to take turns trying out each other's games and discussing how they used code and variables in their game. Encourage groups to compare their code and discuss similarities and differences with the different functions they created. In addition, it's important to encourage coders to consider whether their project reaches a diverse set of end users (e.g., people who are color blind, people who are deaf, people with limited mobility, etc.). ***5+ minutes to revise their project and 1-on-1 facilitating*** Give groups five or so minutes to revise their projects based on feedback and ideas they gathered from their peers. Encourage peer-to-peer assistance and facilitate 1-on-1 as needed. ***I recommend repeating this process several more times to encourage sharing ideas and getting peer feedback***	**Standards reinforced:** ● **1B-AP-08** Compare and refine multiple algorithms for the same task and determine which is the most appropriate. ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **1B-AP-13** Use an iterative process to plan the development of a program by including others' perspectives and considering user preferences ● **1B-AP-16** Take on varying roles, with teacher guidance, when collaborating with peers during the design, implementation, and review stages of program development **Practices reinforced:** ● Collaborating around computing ● Communicating about computing ● Creating computational artifacts ● Fostering an inclusive computing culture ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Facilitation tip:** It may help to model the kind of feedback one might give to a game like this. To practice this, display the game I created for this lesson or one of your own games. Ask coders what's something they like about the project, what they might be curious about, and what suggestions they might have for improving the project(s).
7. Add in comments (*the amount of time depends on typing speed and amount of code*): ***1 minute demonstration*** 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. Quickly review how to add in comments. ***Commenting time*** 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:** ● **1B-AP-17** Describe choices made during program development using code comments, presentations, and demonstrations **Practices reinforced:** ● Communicating about computing **Concepts reinforced:** ● Algorithms **Video:** Add in comments (1:45) **Quick reference guide:** Click here **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 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 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. ● Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes? ● Did coders include descriptive comments for each event in all of their sprites? ● Can coders explain how they used broadcast blocks or My Blocks as functions to make their code more organized and easier to read (modularity)? ● Can coders explain what the different variables blocks do and how they could potentially use them in a project? ○ Can coders explain when they might need	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:** ● 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.	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 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.

to create a variable? ● Did coders create a catching game with at least ## different players and scores for each player? ○ Choose a number appropriate for the coders you work with and the amount of time available.		

Extended Learning

Project Extensions

Suggested extensions	Resources, suggestions, and connections
Use the example project as a guide (as needed) At some point, coders might get stuck or run out of ideas. Rather than explaining to them how to do something, ask them to open the example project, read the comments inside the various sprites and then look at the code to see if they can figure out how to solve their problem. Although this is a very open-ended approach, this models a common coding practice that helps coders become independent learners.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **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 **Practices reinforced:** ● Creating computational artifacts ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control ● Modularity **Resource:** Example project **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.
Add even more (30+ minutes, or at least one class): 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). While facilitating this process, monitor to make sure coders don't stick with one feature for too	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals **Practices reinforced:** ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **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

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.	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. **Suggested questions:** ● What else can you do with Scratch? ● What do you think the other blocks do? a. Can you make your project do ____? ● What other sprites can you add to your project? ● What have you learned in other projects that you could use in this project? ● Can you add more user control than demonstrated? ● What other variables blocks might you use in your project? ● Could you add in other sprites as enemies or power ups? ● Could you add even more players to your game?
Similar projects: 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). Watch this video (3:20) if you are unsure how to use a project studio.	**Standards reinforced:** ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals ● **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 **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," "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." **Generic questions:** ● What are some ways you can expand this project beyond what it can already do? ● 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 add player controls to this project? ● How might you use variables blocks in this project? ● If the project is not a game, could you turn this project into a game? ● If the project is a game, could you turn it into a different kind of game?
micro:bit extensions: **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	**Standards reinforced:** ● **1B-AP-09** Create programs that use variables to store and modify data ● **1B-AP-10** Create programs that include sequences, events, loops, and conditionals

before encouraging coders to use the micro:bit blocks.
 	
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 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.
	• **1B-AP-11** Decompose (break down) problems into smaller, manageable subproblems to facilitate the program development process
	• **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended
	Practices reinforced:
	• Recognizing and defining computational problems
	• Creating computational artifacts
	• Developing and using abstractions
	• Fostering an inclusive computing culture
	• Testing and refining computational artifacts
	Concepts reinforced:
	• Algorithms
	• Control
	• Modularity
	• Program Development
	• Variables
	
	Folder with all micro:bit quick reference guides: Click here
	Additional Resources:
	• Printable micro:bit cards
	  ○ Cards made by micro:bit
	  ○ Cards made by Scratch
	• Micro:bit's Scratch account with example projects
	
	Generic questions:
	• How can you use a micro:bit to add news forms of user interaction?
	• What do the different micro:bit event blocks do and how could you use them in a project?
	• How could you use the LED display for your project?
	• What do the tilt blocks do and how could you use them in your project?
	• How could you use the buttons to add user/player controls?
	• How might you use a micro:bit to make your project more accessible?

Differentiation

Less experienced coders	More experienced coders
Demonstrate the example remix project or your own version, and walk through how to experiment changing various parameters or blocks to see what they do. Give some time for them to change the blocks around. When it appears a coder might need some guidance or has completed an idea, encourage them to add more to the project or begin following the steps for creating the project on their own (or with BootUp resources). Continue to facilitate one-on-one using questioning techniques to encourage tinkering and trying new combinations of code. If you are working with other coders and want to get less experienced coders started with remixing, have those who are	Demonstrate the project without showing the code used to create the project. Challenge coders to figure out how to recreate a similar project without looking at the code of the original project. If coders get stuck reverse engineering, use guiding questions to encourage them to uncover various pieces of the project. Alternatively, if you are unable to work with someone one-on-one at a time of need, they can access the quick reference guides and video walkthroughs above to learn how each part of this project works. If you are working with other coders and want to get more experienced coders started with reverse engineering, have

interested in remixing a project watch this video (2:42) to learn how to remix a project.	those who are interested watch this video (2:30) to learn how to reverse engineer a project.

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
Why doesn't Player 1's score reset each time the game restarts? ● We need to set the score to 0 in our "Setup" function ○ This is a process known as initializing a variable ■ We are declaring there's a variable called "Player 2 Score" and are initializing (setting) that variable to 0 Why does Ball 1 stay at the bottom of the screen instead of disappearing like Ball 2? ● Our "Make the clone fall to the ground" function is repeating until the x position is less than -160; however, we want it to repeat until the y position is less than -160 (because it's only going to move down, not left to right) Why does Ball 2 stay at the bottom of the screen instead of disappearing like Ball 1? ● We need to delete this clone at the end of each chunk of code that starts with when I start as a clone, otherwise the clone will remain on the screen when it finishes running its code ***micro:bit required*** Why is the LED timer taking more than one second before switching? ● The "display text" block will not move to the next block until it finishes displaying the entire text, so taking a couple of seconds to scroll and then wait one second before switching to the next value ○ We can fix this using a variety of methods; however, the easiest is to send a message (not an "and wait" message) to change the LED value Even more debugging exercises	**Standards reinforced:** ● **1B-AP-15** Test and debug (identify and fix errors) a program or algorithm to ensure it runs as intended **Practices reinforced:** ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Suggested guiding questions:** ● What should have happened but didn't? ● Which sprite(s) do you think the problem is located in? ● What code is working and what code has the bug? ● Can you walk me through the algorithm (steps) and point out where it's not working? ● Are there any blocks missing or out of place? ● How would you code this if you were coding this algorithm from Scratch? ● 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. **Reflective questions when solved:** ● What was wrong with this code and how did you fix it? ● Is there another way to fix this bug using different code or tools? ● *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

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?
- 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 did someone else's game differ from your group project?
- What did you learn by working with another coder?

Sharing suggestions

Standards reinforced:

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

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.

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

• How else could you use variables blocks in other projects? • How could you add variables blocks to previously created projects? • More sample prompts	