Dance Party

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

At a Glance

Overview and Purpose

Coders use the start on green flag block to create a silly dance party using motion blocks. The purpose of this project is to introduce young coders to adding sprites in code and triggering algorithms with the green flag in ScratchJr.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will learn how to use events to trigger an algorithm. **Question:** ● How can we use events to trigger an algorithm?	**Statement:** ● I will use the start on green flag block to trigger a silly dance party using motion blocks. **Question:** ● How can we use the start on green flag block to trigger a silly dance party using motion blocks?
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-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 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) ● **P5.2.** 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) ● **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)	**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** - 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

Supporting blocks	Control, 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)
Code	• Any set of instructions expressed in a programming language. (source) • Written computer instructions. The term code is somewhat colloquial. For example, a programmer might say: "I wrote a lot of code this morning" or "There's one piece of code that doesn't work." Code can appear in a variety of forms. The code that a programmer writes is called source code. After it has been compiled, it is called object code. Code that is ready to run is called executable code or machine code. (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)
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:** Physical education **Example(s):** This project could integrate with physical education classes if coders embodied the dance moves by physically mimicking a sprite's algorithm. Note this process may get a little silly in the best way possible.
Vocations	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.
 	
(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 (7+ minutes)

Suggested sequence	Resources, suggestions, and connections
1. Review and demonstration (5+ 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. Review how to open Scratch and create a new project. Review using motion and control blocks to make Scratch Cat dance. Explain to the class that we are going to add some more sprites (characters) to dance with Scratch Cat. Demonstrate pressing the plus sign on the left and finding another sprite; think out loud that you need to click the checkmark after you've selected your sprite. Repeat this process again with another sprite, then think out loud that you've changed your mind and you're going to delete a sprite. Demonstrate deleting a sprite by pressing and holding on a sprite (without wiggling your finger), then pressing the red delete button that appears. Ask the class what the steps are for deleting a sprite to quickly review.	**Practices reinforced:** • Communicating about computing **Video:** Project Preview (0:36) **Video:** Adding and deleting sprites (1:04) **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? **Demonstration suggestion:** When selecting a sprite, you could give a couple of options and have the class quietly vote with thumbs up or thumbs down. **Note:** Younger coders might have difficulty deleting a sprite as they tend to slide their finger once they tap down. This may simply take some practice and patience.
2. Quick review (2+ minutes): Have coders quickly review with a neighbor how to add in sprites and how to delete a sprite.	**Practices reinforced:** • Communicating about computing

After the discussion, coders will begin adding sprites to their project as a class, in small groups, or at their own pace.	**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:** ● Where do you press to add another sprite? ● How do you delete a sprite? ● Can you delete a sprite if your finger wiggles while you press and hold?

Project Work (40+ minutes; 1+ classes)	
Suggested sequence	**Resources, suggestions, and connections**
3. Adding in sprites (5+ minutes): Set an amount of time for coders to look through the different sprites and add at least a few sprites into their project. Facilitate by walking around and asking questions and encouraging exploration.	**Facilitation suggestion:** Narrate out loud some of the themes that emerge as you walk around and look at projects. Give some ideas of what kind of dance party coders might create; for example, a dance party in space, a spooky dance party, an old person dance party, etc.
4. Creating our dance party with trigger blocks (30+ minutes): ***8+ minute demonstration and discussion*** Once coders have at least a couple sprites added, bring everyone back together as a group. Tell the class we're going to have a three second dance party when you tap your hand on your head, and then everyone is going to freeze when you raise your hands in the air; practice this one or two times. Ask what event caused them to dance (hand on head) and what event caused them to freeze (hands in the air). Tell the class we're going to change the algorithm so that when you put your hand on your head everyone is going to jump three times and then stop; practice this a couple of times. Select a sprite and demonstrate how to make it jump three times when you press the green flag (using the start on green flag block); practice having the class jump with the sprite. Ask everyone to think back to when we first danced and recall if everyone had the same dance or if some of the dances were different. Demonstrate how we can switch to another sprite and give them their own code to dance to. Think out loud how you want to make sure to include a start on green flag block; demonstrate pressing the green flag so multiple sprites dance. ***22+ minute coding time and 1-on-1 facilitating*** Ask coders to create a dance party where sprites use repeat blocks and coders change the parameters in the motion blocks. Facilitate by walking around and asking questions and	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Communicating about computing ● Developing and using abstractions ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Video:** Trigger blocks (1:28)

encouraging coders to make it so each sprite dances in their project when the green flag is pressed.	
5. Where are they dancing? (5+ minutes): ***2+ minute demonstration*** Bring the class together again and demonstrate how to add a background. ***3+ minute coding time and 1-on-1 facilitating*** Give them time to add their own backgrounds and continue to work on their dances as long as time allows. Facilitate by walking around and asking questions about where sprites might dance and encouraging exploration of new algorithms.	**Practices reinforced:** ● Creating computational artifacts

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 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 change 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:** ● 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.

parameters in the motion blocks and can they predict how each sprite will dance? ● Did coders add repeats to their algorithms and can they explain how each sprite will dance? ● Did coders create algorithms to make at least ## sprites dance? ○ 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
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 ____? ● Where else might your sprites dance? ● What other sprites can you add to your project?
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, a dance to a song).

Debugging Exercise (1-5+ minutes)

Debugging exercises	Resources and suggestions
Debugging example code ScratchJr Debugging List	**Standards reinforced:** • **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 a dance where one of the sprites is not dancing, but all of the other sprites are when we tap the green flag. Ask the class: Why is the sprite not dancing? How can we fix it? Think out loud which sprite isn't dancing, and click on the sprite to look at the code. Explain that mistakes in code are called bugs. To fix the bugs, coders need to find the bug and get rid of it. This is called debugging. Include in the code motion blocks that do not have a start on green flag block. Ask the class to talk with a neighbor about how to fix the code so the sprite starts dancing again. Test out ideas until the bug is found and fixed in the code.

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. The big event
    a. Events are a great way to add variety to a pre-written algorithm. Sometimes you want your program to be able to respond to the user exactly when the user wants it to. That is what events are for.
2. Building a foundation
    a. New and unsolved problems are often pretty hard. If we want to have any chance of making something creative, useful, and clever, then we need to be willing to attack hard problems. This lesson teaches that failure is not the end of a journey, but a hint for how to succeed.

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?
- What should you do when your code doesn't work the way you expected it to?
- What are some events you think your ipad/laptop might use to trigger code?
- What are some other events you could use in ScratchJr besides the green flag?
    - What do you think they do?
- 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.