Ninja Maze Challenge 2

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

At a Glance

Overview and Purpose

Coders will solve four different maze challenges that focus on diagonal movements, then create their own unique mazes that peers will solve. The purpose of this project is to reinforce understanding of predicting and sequencing a sprite's movement using the motion blocks running in parallel.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: ● I will use motion blocks running in parallel to guide a sprite through a maze without touching walls. **Question:** ● How can we use motion blocks running in parallel to guide a sprite through a maze without touching walls?	**Statement:** ● I will be able to explain how a pair of algorithms guides a sprite through a maze without touching walls. ● I will learn how to create mazes that a friend will solve. **Question:** ● How can a pair of algorithms guide a sprite through a maze without touching walls? ● How can we create mazes that a friend will solve?
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-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	**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

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)	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) ● **P5.1.** Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations. (p. 80) ● **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80) **Practice 6: Testing and refining computational artifacts** ● "Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts." (p. 81) ● **P6.1.** Systematically test computational artifacts by considering all scenarios and using test cases." (p. 81) ● **P6.2.** Identify and fix errors using a systematic process. (p. 81)	**Practice 7: Communicating about computing** ● "Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences." (p. 82) ● **P7.2.** Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. (p. 82)
Main concept(s):	**Reinforced concept(s):**
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)	**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** - 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)	• **Grade 2 -** "Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions." (p. 96) **Program Development** • "Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing." (p. 91) • **Grade 2 -** "People develop programs collaboratively and for a purpose, such as expressing ideas or addressing problems." (p. 97)

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

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)
Bug	• An error in a software program. It may cause a program to unexpectedly quit or behave in an unintended manner. (source) • The process of finding and correcting errors (bugs) is called debugging. (source) • An error or defect in software or hardware that causes a program to malfunction. Often a bug is caused by conflicts in software when applications try to run in tandem. According to folklore, the first computer bug was an actual bug. Discovered in 1945 at Harvard, a moth trapped between two electrical relays of the Mark II Aiken Relay Calculator caused the whole machine to shut down. (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)
Decompose	• To break down into components. (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)
More vocabulary words from CSTA	• Click here for more vocabulary words and definitions created by the Computer Science Teachers Association

Connections

Integration	**Potential subjects:** Math, media arts **Example(s):** This project could integrate with math lessons by using the grid and counting/adding how many steps a sprite should move to solve a maze.
Vocations	Media artists and designers are frequently asked to create levels or worlds for characters to exist within. In this project we are asking coders to not only create levels, but to determine the precise amount of steps to take to navigate within such a space. Such a process relates to basic math. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

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

Project Sequence

Preparation (20+ minutes)

Suggested preparation	Resources for learning more

Ensure all devices are plugged in for charging over night.
 	
Customizing this project for your class (10+ minutes): Remix the project example to include your own maze challenges that require coders to use at least two sets of motion blocks running in parallel. Make a copy of your custom project that doesn't include any motion blocks on the NinjaCat sprite (this copy will be distributed to coders to remix).
 	
(10+ minutes) Read through each part of this lesson plan and decide which sections the coders you work with might be interested in and capable of engaging with in the amount of time you have with them. If using projects with sound, individual headphones are very helpful.
	• BootUp ScratchJr Tips
	  ○ Videos and tips on ScratchJr from our YouTube channel
	• BootUp Facilitation Tips
	  ○ Videos and tips on facilitating coding classes from our YouTube channel
	• Block Descriptions
	  ○ A document that describes each of the blocks used in ScratchJr
	• Interface Guide
	  ○ A reference guide that introduces the ScratchJr interface
	• Paint Editor Guide
	  ○ A reference guide that introduces features in the paint editor
	• Tips and Hints
	  ○ Learn even more tips and hints by the creators of the app
	• Coding as another language (CAL)
	  ○ A set of curriculum units for K-2 using both ScratchJr and KIBO robotics
	• ScratchJr in Scratch
	  ○ If you're using ScratchJr in Scratch, this playlist provides helpful tips and resources

Getting Started (5+ minutes)

Suggested sequence	Resources, suggestions, and connections
1. Review and demonstration (2+ minutes): Begin by asking coders to talk with a neighbor for 30 seconds about something they learned last time; assess for general understanding of the practices and concepts from the previous project. Explain that today we are going to solve and create maze challenges using code. Display and demonstrate the sample project (or your own remixed version).	**Practices reinforced:** ● Communicating about computing **Video:** Project Preview (0:51) **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, it might help to set a time limit for exploration before discussing the project. **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 solve the maze displayed in the example project 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 will need to move around the walls to get to the goal. Where can the ninja move so it doesn't touch a wall? What motion block(s) might be in the code to make the sprite move in that direction? Where will the ninja need to go next?" 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 solve something like this in ScratchJr? ● What kind of blocks might we use? ○ How can we use two trigger blocks to make the ninja move diagonally? ● If you were creating your own mazes, 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 maze challenge? ○ What kind of code might they have?

Project Work (60+ minutes; 2+ classes)

Suggested sequence	Resources, suggestions, and connections
3. Solve the mazes (32+ minutes):	**Standards reinforced:**

Begin by sharing with everyone the sample project (or your own remixed version) that does not include any motion blocks that move the NinjaCat through the maze (use this guide if you are unsure how to share project files).

Repeat the following for each level.

1 minute prompt
Making sure the motion blocks are removed or hidden from the example project (or levels you created and shared), display one of the levels so everyone can see the maze:

- Level 1
  - Answer
- Level 2
  - Answer
- Level 3
  - Answer
- Level 4
  - Answer

5+ minute problem solving and peer-to-peer coaching
Ask coders to see if they can figure out how to make the sprite navigate through the maze without touching a wall. Facilitate by walking around and asking guiding questions.

2 minute explanation demonstration
If coders figured out how to navigate through the maze without touch a wall, 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 and walk through each step of the algorithm. Repeat this process with the remaining levels.

- **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
- Creating computational artifacts
- Testing and refining computational artifacts

Concepts reinforced:

- Algorithms
- Control

Resource: Sharing ScratchJr Projects

Suggested guiding questions:

- What kind of blocks do you think you might need to solve this maze?
- Do you see a pattern where we might use a repeat?
- Do we need to use one trigger block or more than one?
  - What makes you think that?
- How could you change the level to make it easier or harder?

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

4. Create even more mazes (20+ minutes):
Ask coders to create a new project with a small sprite, a goal, and at least one level sprite. Use looks blocks to shrink the sprite that will navigate the maze and to grow the level sprite(s) to create obstacles to navigate around. Facilitate by walking around and asking questions and encouraging coders to not only create at least one new level a peer will solve, but to test out their mazes to make sure the sprite can vertically, horizontally, or diagonally navigate through the maze without touching a wall or obstacle.

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:

- Creating computational artifacts
- Program development
- Testing and refining computational artifacts

Concepts reinforced:

- Algorithms
- Control

Suggested questions:

- How can we make a level easier or harder?
- How many sprites will you use for a wall?
  a. How will a player know if they've touched a

	wall? ● How many trigger blocks might we need to get code to run in parallel? ● Will we need to use any wait blocks?
5. Solve the new mazes (8+ minutes or until the end of class): *If time permits, repeat the following process until out of time.* ***1 minute device swapping*** Making sure the motion blocks are removed or hidden from the newly created levels have coders swap devices with a neighbor to try and solve the newly created mazes. ***5+ minute problem solving and peer-to-peer coaching*** Ask coders to see if they can figure out how to make the sprite navigate through the maze without touching a wall. Facilitate by walking around and asking guiding questions. ***1 minute explanation demonstration*** If coders figured out how to navigate through the maze without touch a wall, 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, have the creator of the project reveal the solution and walk through each step of the algorithm. If time permits, repeat this process by having coders remove the solutions (or partial solutions) and swap their device with another coder to try and solve another level.	**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 ● Creating computational artifacts ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Suggested guiding questions:** ● What kind of blocks do you think you might need to solve this maze? ● Do you see a pattern where we might use a repeat? ● Do we need to use one trigger block or more than one? ○ What makes you think that? ● How could you change the level to make it easier or harder? **Potential discussion:** There is not always one way to solve a problem with code, so coders may come up with alternative solutions to your own code. When this occurs, it can open up an interesting discussion or journal reflection on the affordances and constraints of such code.

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	The 1-on-1 facilitating during each project is a form of formative	The reflection and sharing section at the end of each lesson can be a form of

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 solve or 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? ● Can coders accurately predict how a sprite will move before running code? ● Did coders solve or create a maze game with at least ## different levels to navigate through? ○ Choose a number appropriate for the coders you work with and the amount of time available. ● Did coders use at least ## pages in their project? ○ Choose a number appropriate for the coders you work with and the amount of time available. ○ Can coders explain when/how the project will switch pages?	assessment 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.	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.

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 might we use in a maze game? ● What other sounds might we hear if a sprite touches a wall? a. What about if it touches the goal? ● Can you customize how your sprites look?
Similar projects: Have coders explore the sample projects built into ScratchJr (or projects from other coders), and ask them to find code similar to what they worked on today.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms **Note:** Coders may need a gentle reminder we are looking at other projects to get ideas for our own project, *not to simply play around*. For example, "look for five minutes," "look at no more than five other projects," or "find three projects that each do one thing you would like to add to your project." **Generic questions:** ● How is this project similar (or different) to something you worked on today? ● What blocks did they use that you didn't use? a. What do you think those blocks do? ● What's something you like about their project that you could add to your project? ● How might we change the backdrop of this project? ● What other sound or looks blocks might we use in this project? ● Can you turn this project into a maze game or challenge?

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	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

feels comfortable working cooperatively on a project. Once coders appear to get the hang of using ScratchJr, they can begin to work independently.	them help out their peers when they have a question. If someone asks for your help, guide them to a peer helper in order to encourage collaborative learning. Another approach is to encourage experienced coders to experiment with their code or give them an individual challenge or quest to complete within a timeframe.

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
For each of the following debugging exercises, display the code with the bug and ask coders to predict what the bug is and solve the bug without running the code. The purpose of this differentiation in debugging is to encourage coders to analyze and predict what an algorithm will do without running the code. What's the bug in this code and how can we fix it? ● We are not moving up far enough What's the bug in this code and how can we fix it? ● We need to move right before moving up What's the bug in this code and how can we fix it? ● We are moving too far to the right after moving left What are the bugs in this code and how can we fix them? ● We are not moving down far enough or up far enough 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 one of the debugging exercises and ask the class what they think we need to fix in our code to get our project to work correctly. Think out loud what might be wrong (e.g., did I use the wrong trigger block, did I forget to repeat something, did I put a block in the wrong place, am I missing blocks, etc.). Ask the class to talk with a neighbor how we might fix the code. Have a volunteer come up to try and debug the code (or demonstrate how). Repeat with each debugging exercise.

Unplugged Lessons and Resources

Standards reinforced:

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

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

List of 100+ unplugged lessons and resources

Reflection and Sharing

Reflection suggestions	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? ● 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 or solved today? ● What's something you could create or solve next time? ● What questions do you have about coding? ○ What was challenging today? ● More sample prompts (may need adapting for younger coders)	**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.