

GRADE 6

GRADE 7

GRADE 8

Efficient Code

Subprograms and
Defined Count

Analysis of Data



GRADE 7 TEACHERS' GUIDE

Subprograms and defined count

Welcome to NII Explore's *Coding in the Classroom* program for Grade 7 students. During the 4-week program, you and your class will complete:

- 3 online lessons
- 3 offline activities
- A final coding project

This teacher guide includes everything you need to get started!

THE GRADE 7 CODING CURRICULUM

As of 2020, Ontario's math curriculum includes coding expectations. Put simply, coding is when we write instructions, or "code", for a computer to follow. There are two core expectations that run through every grade level of the coding curriculum.

1. Writing and executing code

2. Reading and altering existing code

Each grade level introduces students to a new coding skill. Students can practice this new skill while also using the skills learned in previous grades. In Grade 6, students practice making their code more efficient (i.e., solving problems with the fewest lines of code).

SCHEDULE AT A GLANCE

WEEK 1

- **Online Lesson 1**
Intro to Scratch
- **Offline Activity 1**
Defined Count Scavenger Hunt

WEEK 2

- **Online Lesson 2**
Make a Maze Game
- **Offline Activity 2**
Songs with Subprograms

WEEK 3

- **Online Lesson 3**
Single-Player Pong
- **Offline Activity 3**
Video Game Design

WEEK 4

- **Final Project**
Make Your Own Game

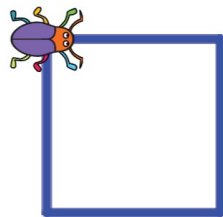
GRADE 7 introduces students to two new ideas that will make their code even more efficient: subprograms and defined count.

A **subprogram**, also called a function, is a section of code that performs a specific task. We use subprograms when there is a task that we know we will need to do multiple times during our program. Instead of writing the same instructions each time, we can put those instructions in a subprogram and call on it when we need it.

For example, we could write a subprogram that draws a single square.

```

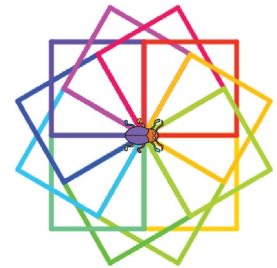
define draw square
  pen down
  repeat 4
    move 100 steps
    turn 90 degrees
  
```



We can then use that subprogram in a larger program to create an image with many squares.

```

when clicked
  repeat 12
    change pen color by 10
    draw square
    turn 30 degrees
  
```

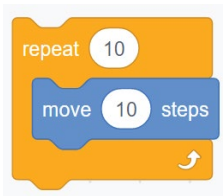


Coding with subprograms. The main program on the left calls the “draw square” subprogram 12 times to create the image on the right.

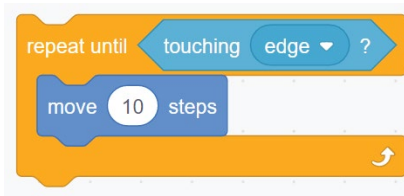
In Scratch, subprograms are called “My Blocks” and they are shown in **PINK**. Once a My Block has been defined, it can be used just like any other block.

Students also learn how to use a **defined count** in Grade 7. We use defined counts in coding to tell a loop how many times it should repeat. There are two ways we can do this. We can have our loop repeat *for* a specific number of times (often called a “for” loop) or we can ask our code to repeat *until* a certain condition is met (an “until” loop).

FOR LOOP



UNTIL LOOP



Two ways to define count. *In the first example, our sprite will move forward 10 steps and repeat that action exactly 10 times. In the second case, the sprite will keep moving forward until it touches the edge of the screen.*

We can also code our programs to **wait** for a specific amount of time or until something happens. These are all examples of control structures, and they are found in the **Control** tab in Scratch.

Students have probably used the idea of a defined count in their everyday lives, even if they were not aware of it.



Bake cookies **FOR** 12 minutes

OR

Bake cookies **UNTIL** golden brown



Play soccer **FOR** another 5 minutes

OR

Play **UNTIL** someone scores

Defined counts are everywhere! *We use defined counts whenever we need to decide how long or how often we will do something.*

PROGRAM SCHEDULE

The *Coding in the Classroom* program will last four weeks. Here is a detailed guide of what you will be doing each week.

BEFORE WEEK 1

Read through this teacher guide, including the instructions for the three online lessons. If you have time, you may want to try the online activities for yourself.

Make sure your class has access to devices (laptops or tablets) for each of the online lessons. Your class will also need devices for the final project.

WEEK 1

Online Lesson 1 – Intro to Scratch

This lesson introduces students to Scratch, an online coding platform. Students will use a subprogram and a defined count to make their own “clicker” game.

PREP Log onto computers and open Scratch. Ask students to “Join Scratch” and make an account using their school email address.

POST Complete “Offline Activity 1 – Defined Count Scavenger Hunt” before next online session.

See [Page 7.6](#) for lesson instructions and [Page 7.11](#) for a quick reference guide.

Offline Activity 1 – Defined Count Scavenger Hunt

Your class will practice using two types of defined count in this friendly competition.

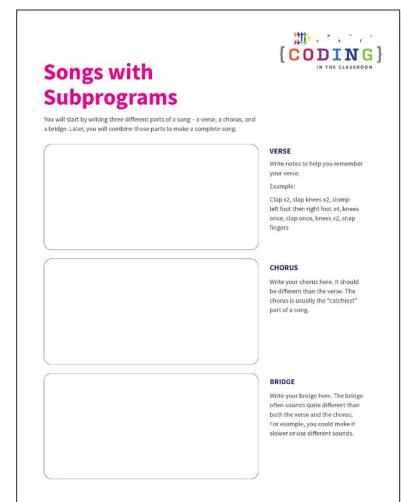
See [Page 7.15](#) for activity instructions.

WEEK 2

Online Lesson 2 – Make a Maze Game

In this activity, students will edit an incomplete Scratch file to make a working game. Students will practice using a defined count and apply transformations on a Cartesian plane.

PREP Have students log into their Scratch accounts and open the activity link.



Week 2 Songs with Subprograms

POST Complete “Offline Activity 2 – Songs with Subprograms” before next online session.

See Page **7.18** for lesson instructions and Page **7.23** for a quick reference guide.

Offline Activity 2 – Songs with Subprograms

Your students will quickly compose simple songs using the power of subprograms.

See **Page 7.27** for activity instructions.

WEEK 3

Online Lesson 3 – Single-Player Pong

Students will turn back the clock to make a single-player version of *Pong*. The lesson will explore xy-coordinates and angles in addition to coding concepts.

PREP Have students log into their Scratch accounts and open the activity link.

POST Complete “Offline Activity 3 – Video Game Design” and the Final Project.

See **Page 7.30** for lesson instructions and **Page 7.35** for a quick reference guide.

Offline Activity 3 – Video Game Design

Students will prepare for their final coding project by completing a Game Design Document.

See **Page 7.40** for activity instructions.

WEEK 4

Final Project – Make Your Own Game

Students will apply their learning to make their own game in Scratch. When they’re finished, they will share their projects for you to evaluate.

See **Page 7.44** for project instructions.

AFTER WEEK 4

Keep the coding going with the additional resources on **Page 7.49!**



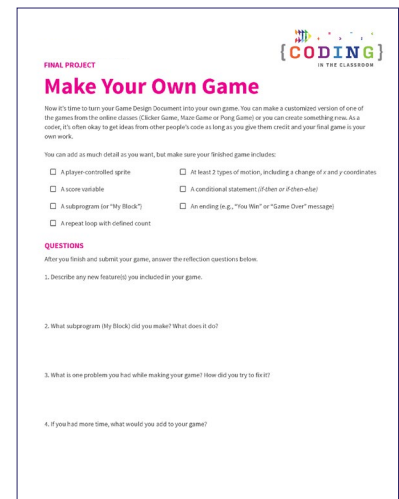
Game Design Document

Use this sheet to help plan your video game. You can use one of our existing games as a starting point or you can make a different game. Later, you will make your game in Scratch.

NAME _____

GAME TITLE	
DESCRIPTION What's the game about?	GAMEPLAY Explain what happens in the game. How does the player control their object? Score points? Win the game?
CHARACTERS Sketch and describe them here.	
SETTING Where does the game happen?	EXTRAS

Week 3 Game Design Document



FINAL PROJECT

Make Your Own Game

Now it's time to turn your Game Design Document into your own game. You can make a customized version of one of the games from the earlier classes (Clicker Game, Maze Game or Thing Game) or you can create something new. As a teacher, it's often okay to get ideas from other people's code as long as you give them credit and your final game is your own work.

You can add as much detail as you want, but make sure your finished game includes:

<input type="checkbox"/> A player-controlled sprite	<input type="checkbox"/> At least 2 types of motion, including a change of x and y coordinates
<input type="checkbox"/> A score variable	<input type="checkbox"/> A conditional statement (if/then or if/then-else)
<input type="checkbox"/> A subprogram (or "My Block")	<input type="checkbox"/> An ending (e.g., "You Win" or "Game Over" message)
<input type="checkbox"/> A repeat loop with defined count	

QUESTIONS

After you finish and submit your game, answer the reflection questions below.

- Describe any new feature(s) you included in your game.
- What subprogram (My Block) did you make? What does it do?
- What is one problem you faced while making your game? How did you try to fix it?
- If you had more time, what would you add to your game?

Week 4 Final Project

ONLINE LESSON 1

Intro to Scratch

(Timed clicker game)

60 MINUTES

The three online lessons and final project all use Scratch. If you are new to Scratch, you may want to check out their “**Getting Started**” tutorial.

In this lesson, you will introduce students to Scratch and have them make their own accounts. The goal is to familiarize students with Scratch and some of its basic commands. Along the way, students will use defined count and a subprogram to make a timed clicker game.

Subprograms (known as “functions” in most programming languages) are sections of code that perform a specific task. In Scratch, we can make subprograms by creating “My Blocks”. Check out [this video tutorial](#) to learn more about My Blocks.

The lesson will also introduce defined count. “**Defined count**” is not a commonly used coding term outside of the Ontario curriculum. Within the curriculum, however, count means the number of times that a loop repeats, and it can be defined in two ways. We can specify an exact number of repeats using a for loop or have it repeat until a certain condition is met (an until loop).

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves events influenced by a defined count and/or subprogram and other control structures
- **Math C3.2** – read and alter existing code, including code that involves events influenced by a defined count and/or subprogram and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code

OPERATIONS

- **Math B2.4** – use objects, diagrams, and equations to represent, describe, and solve situations involving addition and subtraction of integers

QUICK LINKS

Student Activity Link
scratch.mit.edu

Finished Example
scratch.mit.edu/projects/818521004

PowerPoint
[Grade 7 – Week 1 – Intro to Coding](#)

Quick Reference
[Intro to Scratch \(Timed Clicker Game\) – Quick Reference](#)

LESSON BREAKDOWN

SLIDE 1 - SET UP AND INTRODUCTION

Open the PowerPoint slides and Scratch links on your own computer. Project for the students to see. Open or print the **Intro to Scratch (Timed Clicker Game) - Quick Reference** for your own use during the lesson.

SLIDE 2 - WHAT IS CODING?

Check if your students have coding experience and if they've used Scratch before. Ask them what coding means (coding is when we give instructions to a computer).

SLIDES 3 AND 4 - WHAT TO EXPECT

Your class will complete three online lessons (using Scratch), three offline activities, and a final project (make their own game).

If your students make a game that they are particularly proud of, please share it with us at explore@nii.ca. NII Explore periodically awards prizes to some of our favourite coding projects.

SLIDE 5 - READY TO START

Have students open the student activity link on their devices. It should take them to the Scratch home page.

MAKING SCRATCH ACCOUNTS

Start the first week by having all the students make Scratch accounts. This will let them save their projects and access them at home or on another day. It may take a few minutes but will be worthwhile in the long run.

Click “**Join Scratch**” in the top right. Create a username and password. Have students choose a username that will be easy for them to remember.

For their password, they should choose something that is hard to guess. As an added measure, encourage them to write down their login credentials in a safe place.

It is not the best practice from a security standpoint, but for simplicity, they could use the same password that they use to log into their computers.

If their username is taken, have them add numbers at the end of it.

They do not need to give out their personal details other than an email address. Students can use their school email address.

If students already have Scratch accounts, they can log into them to start.

DEMO

Show the finished game to give students an idea of what they are working towards. You can make your own game or use this one:

scratch.mit.edu/projects/818521004

Click the four arrows icon to make the game full screen then click the green flag to start. See how many times you can click on the bat before the timer runs out.

TOUR OF SCRATCH

NOTE: If students have used Scratch before, you can speed through this part.

Have students create a new project then give them a tour of Scratch. Show them where the coding window and preview window are, and where they can access blocks for their code. **See Page 7.12 for more details.**

CHOOSING A SPRITE AND BACKDROP

Have students choose a backdrop (bottom right corner).

Have students delete the default cat sprite (garbage can beside the sprite icon in the bottom right) and pick a new sprite (blue cat button in bottom right). Ask students to share which backdrop and sprite they picked so you know when they're ready to move on.

CREATE SCORE VARIABLE

We want to be able to keep score in our game. The score is something that can change or “vary” which is why we call it a variable. We use variables to store single pieces of data.

Go to Variables and then “Make a Variable”. Name it “Score”.

You can also delete the default “my variable” by right-clicking on it and choosing “delete”.

INITIALIZE SCORE - STEP 1 IN QUICK REFERENCE

Ask students “What should the score be when we start a new game?” (Zero).

Add “**when green flag clicked**” to begin the program.

Then add “**set Score to 0**” right after the green flag block.

Now the score will be reset to 0 whenever you click the green flag to start a new game.

SCORING POINTS - STEP 2 IN QUICK REFERENCE

From Control, add “**when this sprite clicked**” then “**change Score by 1**”.

The score will go up by 1 every time the target sprite is clicked.

TEST THE GAME

Have the students test the game – does the score go up? Does it reset to 0 when you start a new game?

When everyone is ready, run a friendly competition with the students. See who can get the most clicks in 10 seconds. Ask them to share their scores.

PULSE SUBPROGRAM - STEP 3 IN QUICK REFERENCE

Now we will create a subprogram to make the sprite pulse (grow then shrink) when clicked.

Go to My Blocks then Make a Block called **pulse**. A **define pulse** pink block will appear in the coding area. Under define pulse **add change size by 10 → wait 0.05 seconds → change size by -10** [Step 3a]. Click the newly created block to see what it does.

Finally, add the new **pulse** block to your program under “when this sprite is clicked” [Step 3b].

MOVING SPRITE - STEP 4 IN QUICK REFERENCE

To make the game harder, let's make the sprite move around the screen.

Under the green flag section, add a “**forever**” loop (found in Control) then put “**go to random position**” (found in Motions) inside the loop [Step 4].

Have students test the game and see what happens. The sprite moves way too fast, but students find it funny.

SLOWING DOWN - STEP 4 CONTINUED

ASK: How can we make it go slower?

HINT: Check out the options in Control.

Answer: Add “**wait 1 seconds**” after “go to random position”.

TEST AGAIN

Have students test their games again. They can tweak the wait time between sprite movements to suit their tastes. For example, they could set the wait time to something like 0.8 seconds if they want the game to be a bit harder.

Testing the games regularly helps keep the students engaged, but it's also good practice to test code often to spot mistakes early.

SETTING TIMER - STEP 5 IN QUICK REFERENCE

Students will now add a timer, so their games don't go on forever.

Create a new variable called "Timer". The timer variable should now appear in the top left corner of the game window. You can leave it there or you can drag the timer readout to the top right corner and change it to "large readout". You can do this by double clicking or right clicking on the timer display.

Next, add the following code under a new "**when green flag clicked**" block. Ask students what they think should come next as you build out this timer code. For example, ask them when you should repeat until or what we should change the timer by.

when green flag clicked → set Timer to 10 → repeat until "Timer = 0" → wait 1 second → change Timer by -1 [Steps 5a and 5b]

They can now test that the timer does indeed count down.

ENDING GAME - STEP 6 IN QUICK REFERENCE

Finally, we will want the sprite to disappear and stop moving once the timer expires. Underneath the timer code, add **hide → stop "other scripts in sprite"**.

You will also need to add a **show** block at the very start of your program. You can either tell students that or have them figure it out on their own.

TEST

Have students play a few games and have them report their scores back to you.

ASK: Why do you think we set our timer to 10 for testing? (So that testing doesn't take too long. We can increase the timer once we know that our game works).

OPTIONAL - ADD A SECOND SPRITE TO REPORT THE SCORE - STEPS 7 AND 8 IN QUICK REFERENCE

If time allows, you can add a second sprite to report the score at the end of the game.

Create and initialize the sprite

Have students pick a new sprite, set its size, and drag it to an appropriate position on the screen. To its code, add **when green flag clicked → hide** (This sprite should not be visible until the timer runs out) [Step 7].

Add a broadcast message

Switch back to the main sprite's code. You will add a new broadcast message called "game over". **Add broadcast (game over)** after the part of the code where the timer runs out [Step 8a].

Switch back to your "reporter" sprite and add **when I receive game over** to its code.

Reporting score

After "when I receive game over", add **wait 1 seconds → show → say "Time's up!" for 2 seconds → say (Score variable)** [Step 8b].

FINAL TEST

Give students time to play through their games, try each other's games, or share with you and the rest of the class.

OPTIONAL ADD-ONS

Suggest some extra things students could add to their games.

- **Change time:** How long should the game last?
- **Change sprite size:** Sprite gets smaller each time it's clicked or just starts and stays smaller
- **Change speed:** Make the wait time shorter or longer
- **Add extra sprites:** Copy the movement and scoring code over to a new sprite. Perhaps this sprite is smaller or moves faster and is therefore worth more points.

SLIDE 7 - RECAP

With about 5 minutes left in the class, switch back to the PowerPoint slides.

Ask students to recall what they learned. What is another name for a My Block? (Subprogram or Function). Why did we use a "repeat until" loop instead of a simple repeat? (Timer needs to count down until we reach 0 no matter what it started at).

SLIDE 8 - POEM OF THE DAY

Each slideshow ends with a Poem of the Day to recap the lesson. Introduce the concept of the Poem of the Day then read the poem together.

SLIDE 9 - WHAT'S NEXT?

Let students know when you will be coding again. We recommend alternating between the online lessons and the offline activities. It requires less screen time for your class and will give students more time to absorb the new information.

Quickly preview what you will be doing next week. You will be making a game where students can control their sprite's movement.

QUICK REFERENCE

Intro to Scratch

(Timed clicker game)

After making Scratch accounts, you will give students a tour of Scratch and introduce them to some basic commands. The goal is to familiarize students with Scratch and block-based coding. By the end of the class, each student will have made a simple clicker game with a timer.

SET UP

- Have students open Scratch and either create a new account or log into an existing one
- Open Scratch on your own computer and create a blank project. Open the finished version of the game as a demo

QUICK LINKS

Student Activity Link
scratch.mit.edu

Finished Example
[scratch.mit.edu/
projects/818521004](https://scratch.mit.edu/projects/818521004)

SCRATCH TOUR

The image shows the Scratch IDE interface with several components labeled:

- Switch between code and costumes:** Points to the 'Code' and 'Costumes' tabs at the top left.
- Rename project:** Points to the 'Untitled' text in the top toolbar.
- Which sprite am I editing?:** Points to the small sprite icon in the top right of the workspace.
- Start/stop program:** Points to the green flag and red stop buttons in the top right of the workspace.
- Preview window:** Points to the large stage area on the right.
- Show project full screen:** Points to the full-screen icon in the top right of the stage area.
- Coding Block categories:** Points to the vertical list of colored categories on the left.
- Choose a coding block:** Points to a 'move 10 steps' block in the 'Motion' category.
- Coding window:** Points to the main workspace area.
- List of sprites:** Points to the 'Sprite1' icon in the bottom left of the stage area.
- Sprite properties:** Points to the 'Sprite1' properties panel in the bottom right.
- Choose a sprite:** Points to the 'Choose a sprite' button in the bottom right.
- Choose a backdrop:** Points to the 'Choose a backdrop' button in the bottom right.

THE CODE

This document shows the finished code with annotations explaining what each section does and a suggested order for the lesson. Check out the lesson plan on [Pages 7.7 to 7.10](#) for a more detailed breakdown of the lesson.

SPRITE #1 – MOVING SPRITE

This sprite moves randomly around the screen and the player clicks on it to score points.

Step 1. Initialization – At the start of the program (green flag), sprite appears, and score is reset to 0.

Step 4. Sprite is forever moving to a random position, waiting 1 second, then moving again. This loop won't end until it receives a "stop" command.

Students can change the wait time to suit their taste.

Step 2. Player scores a point every time this sprite is clicked.

Step 3b. Add new pulse block into main program after defining it.

Step 3a. Definition for the "pulse" My Block (subprogram)

Sprite grows a bit, waits a moment, then shrinks back to starting size.



```

when green flag clicked
  set Timer to 10
  repeat until Timer = 0
    wait 1 seconds
    change Timer by -1
  hide
  stop other scripts in sprite
  broadcast game over
  
```

Step 5a. Set the starting time for the timer. Students can change this later, but we use 10 seconds for testing purposes.

Step 5b. Wait 1 second then decrease the timer by 1. Repeat this process until the timer reaches 0.


Step 6. Hides the sprite and stops its movement after the timer reaches 0.

OPTIONAL

Step 8a. Send a “game over” message to trigger the “reporting” sprite.

OPTIONAL SPRITE #2 – REPORTING SPRITE

This sprite pops up at the end of the game to tell the player their score. It is an optional extension if you have time at the end of the lesson in lieu of a simple “Game Over” message.



```

when green flag clicked
  hide
  
```

Step 7. Initialization – Sprite should hide when the game begins.



```

when I receive game over
  wait 1 seconds
  show
  say Time's up! for 2 seconds
  say join Your score was Score
  
```

Step 8b. When this sprite receives the “game over” broadcast message (sent when the timer expires), it will wait for 1 second (optional) then appear.

The sprite then announces that time is up (or something similar) using a say block then reports the player’s score.

OFFLINE ACTIVITY 1

Defined count scavenger hunt

30–45 MINUTES

Students will explore two types of defined count – **FOR loops** and **UNTIL loops** – in this friendly competition.

LEARNING OBJECTIVES

- Understand the difference between the two types of defined count
- Demonstrate movement skills

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)

SET-UP

This activity requires some space. Consider moving to the gym or outside, or push desks out of the way. Collect required materials.

INSTRUCTIONS

SUMMARY

Teams of two will be competing against each other to find bean bags (or similar items). Each team will have a blindfolded “robot” collecting the items and a “coder” telling them where to search. The team that collects the most items wins the round. Some rounds will require the coder to use FOR loops, others will use UNTIL loops.

1. Ask students to recall what they learned in the online coding classes.
ASK “What is coding?” (Giving instructions to a computer) “What are the two types of loop we used? How are they different?” (**FOR loops** specify an exact number of repeats, **UNTIL loops** repeat until a certain condition is met)
2. Explain the premise of the activity. “Today we will be coding, but instead of giving instructions to a computer, you will tell each other what to do.”

MATERIALS

The class will need:

- Small objects like bean bags
- A large, open space
- Optional – Blindfolds

3. Move to a large, open space (recommended) or rearrange your classroom to create space.
4. Divide students into partners. Have the students choose who will be the Coder (person giving instructions) and who will be the Robot (person receiving instructions) for the first round. Explain that the Robot will have their eyes closed (or be blindfolded). You will place bean bags or similar items around the playing area. The Coder's job is to tell their partner where to go.
5. **ROUND 1** Have the teams spread out around the edge of the playing area. For this first round, explain that the Coders can only give instructions using FOR loops. That is, their instructions must include specific numbers. For example, they might say "Take 3 steps forward" or "Turn right 90 degrees". Try a few examples with the whole class.
6. The Robots won't be able to see during this activity so it's important to do a safety briefing before you begin. Remind students to walk slowly and with their arms extended. You will also need a keyword that gets everyone to freeze. In some coding languages (Python, for example), the command "Quit" is used to end a program. Tell the Coders that they should yell "QUIT" if they see a collision about to happen. The Robots should all freeze in place until you tell them to "RESUME" again. Practice this a few times with the whole class.
7. Ask the Robots to close their eyes (or blindfold them) then scatter the bean bags around the playing area. When you're ready, start the game. Let the students play until most of the bean bags have been collected. Whichever team collects the most bean bags wins the round.
8. **ROUND 2** Reset the game and have students switch roles (i.e., Coder becomes Robot and vice versa). Explain that Coders can now only use UNTIL loops – they may not use specific numbers. For example, they could say "Walk forward *until* I say stop" or "Lean over *until* you touch the ground."
9. Start the second round. At the end, ask students to reflect on their experience so far with the following discussion questions.
 - Did you prefer being the Robot or the Coder? Why?
 - Which type of instructions did you find easier to give? Easier to follow?
10. **ROUND 3** For the third round, let students decide who will do which role. This time, the Coder can mix and match their instructions. They can use both FOR loops and UNTIL loops.
11. You can play additional rounds if time allows. For example, students may switch partners or you could introduce additional twists like multiple Coders controlling the same Robot.
12. When you're done, debrief the activity using the following discussion questions as a guide.

NOTE If you have a large class or a smaller space, you may need to play this in multiple heats (i.e., not every team plays at the same time).

DISCUSSION QUESTIONS

What are the two types of loop we used? What's the difference?

Answer: FOR loops specify an exact number of repeats, UNTIL loops repeat until a certain condition is met.

Which round did you find the easiest? What was your winning strategy?

Do you prefer giving instructions to a computer or to a person? Why?

*Possible answers: Computers because they can execute tasks faster and will do exactly what you tell them.
Humans because they can interpret what you mean even if your instructions aren't perfect.*

Games are a good example of the two loop types. Can you think of a game that is played for a specific amount of time? What about a game that's played until a certain condition is met?

Examples: Sports like hockey, soccer, and basketball are played for a set amount of time. Volleyball, tennis, and badminton are played until a certain score is reached. Some video games have a time limit, while others (e.g., tetris) go until the player loses. Jenga is played until the tower falls. Monopoly is played until there is one player left.

Recipes are another good example of everyday instructions. What would a recipe with FOR loops sound like? What about UNTIL loops?

Possible answers: Cook onions for 5 minutes vs Cook onions until they start to brown.

ONLINE LESSON 2

Make a maze game

60 MINUTES

Students will modify (“remix”) an existing Scratch file to create their own maze game called “The Walls are Made of Lava”. The maze is designed so that students will have to add new code to complete each level. Students will use conditional statements, subprograms, and defined count, while also learning about three types of transformation (translations, rotations, and dilations).

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves events influenced by a defined count and/or subprogram and other control structures
- **Math C3.2** – read and alter existing code, including code that involves events influenced by a defined count and/or subprogram and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code

GEOMETRIC AND SPATIAL REASONING

- **Math E1.3** – perform dilations and describe the similarity between the image and the original shape
- **Math E1.4** – describe and perform translations, reflections, and rotations on a Cartesian plane, and predict the results of these transformations a Cartesian plane, and predict the results of these transformations

QUICK LINKS

Student Activity Link
[scratch.mit.edu/
projects/760367848](https://scratch.mit.edu/projects/760367848)

Finished Example
[scratch.mit.edu/
projects/818518940](https://scratch.mit.edu/projects/818518940)

PowerPoint
[Grade 7 – Week 2 – Events](#)

Quick Reference
[Make a Maze Game \(Lava\)](#)
[Quick Reference](#)

LESSON BREAKDOWN

SET UP

Open the PowerPoint slides and the Scratch links on your own computer. Project for the students to see. Open or print the **Make a Maze Game (Lava) - Quick Reference** for your own use during the lesson.

Get the students logged onto their computers with the activity link open.

SLIDES 1 AND 2 - WEEK 1 RECAP

Ask students to remember what they learned about in the first coding lesson. “We made this clicker game using a subprogram (pulse) and a defined count for our timer (repeat until timer = 0). It was a good beginner game, but our sprite moved randomly. In most video games, the player gets to control how their character moves.”

SLIDES 3 TO 5 - HISTORY MINUTE

Use the slides to explain “events” in coding with video games as an example. See slide notes for talking points.

DEMO

Quickly show students what the completed maze game looks like to give them an idea of what they are working towards (We recommend using Level 3 as the demo. You can switch the level by going to “Looks”, choosing “switch backdrop to Level 3” then clicking the block).

Link to finished game:
scratch.mit.edu/projects/818518940

SLIDE 6 - LOG INTO SCRATCH

Have students log into their Scratch accounts and open the activity link.

Have them click “**Remix**” to make their own version of the project.

When they open the project, it should be set to Level 1 (maze is a straight line).

LEVEL 1 - PICK A TARGET SPRITE

Have students create a new sprite to be the target. In our finished example, the target is “Cheesy Puffs”.

Ask students to share what they picked so you know they’re ready.

INITIALIZE THE TARGET SPRITE

Change size of target sprite to make it fit inside the maze. “100 is the default size, set the size to a smaller number to make it smaller.” You can edit the size in the sprite properties area.

Drag the target sprite to the end of the maze.

NOTE: We don’t write any code for the target sprite.

DEFINE COUNT - STEP 1 IN QUICK REFERENCE

Now we will use defined count to build our main code. In the code for the rocket ship, we have pre-set a “repeat until” loop. Ask students when the game should repeat until. In other words, what is the goal?

Answer: In the code for the rocket ship, go to Sensing and add “**touching (target sprite)**” to the “**repeat until**” block [Step 1]. Explain: “We want the game to keep running until our spaceship reaches the target.”

ENABLE LEFT/RIGHT TRANSLATIONS - STEP 2 IN QUICK REFERENCE

Now it's time to make our sprite move. We want our player to be able to control the sprite just like the earliest video games.

Point out the **define enable translations** block. Ask students if they remember what translations are from math class (translations are when we shift a shape left/right/up/down).

Under “define enable translations” we have pre-populated two conditional statements for the right/left arrows. It's up to the students to figure out what they should put inside the if-then blocks. Give them some time to do this.

Answer: if key (right arrow) pressed? then → change x by 4 and if key (left arrow) pressed? then → change x by -4 [Step 2].

The default is to change x by 10, but we recommend using 3 or 4 to allow finer control when navigating the maze. The spaceship will move too quickly if you use bigger values.

PRELIMINARY TESTING

Let students test their left and right arrows by completing the first level. After they reach the target sprite, a broadcast message called “level complete” is sent. A text sprite is pre-programmed to appear when it receives this message.

LEVEL 2

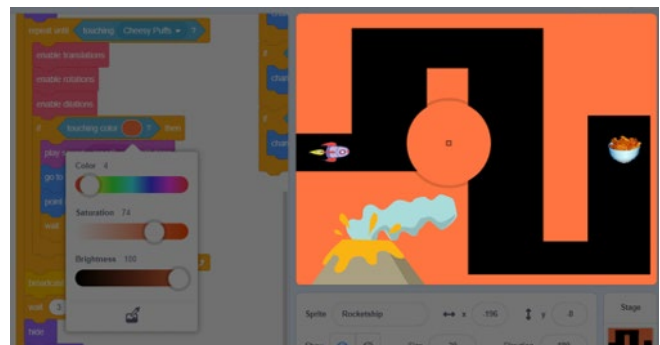
Have students change the backdrop to Level 2. They can do this by going to Looks and then clicking “next backdrop” or they can click on “Backdrops” in the “Stage” area and change to the Level 2 backdrop.

ACTIVATE THE LAVA - STEP 3 IN QUICK REFERENCE

This game is called “The Walls are Made of Lava” but so far the lava doesn't actually do anything. Show the students that you can fly right through the barriers.

To activate the lava, we will need to use a conditional statement. Inside the “repeat until” loop and after “enable translations”, add **if touching color (orange)? then.**

To make sure students select the correct shade of orange, click on the colour in the light blue sensing block then use the eyedropper tool to set it to the same colour as the walls.



This conditional will sense if the rocket ship is touching the orange lava. Next, we need to tell the rocket what to do when this happens. We are going to send our ship back to the start.

Add **go to x: (-196) y: (-8) → point in direction (180) → wait 0.5 seconds** [Step 3] into the conditional statement. This code will send the ship back to the starting position and wait momentarily so the ship doesn't immediately start moving again.

Students could also choose to add a sound effect as an audio cue that they've hit the wall (e.g., **play laser2 until done**).

Have students test by intentionally flying into the wall. Does their ship get sent back to the start? If not, the most common issue is that their shade of orange doesn't exactly match the walls. Remind them to use the eyedropper tool when setting the colour.

ENABLE UP/DOWN TRANSLATIONS - STEP 4 IN QUICK REFERENCE

To beat Level 2, students will need to make their ship move up and down. Tell them that they will need to add two new if-then statements to their **define enable translations** block. Give them some time to figure it out on their own before going over the answer as a class.

Answer: if up arrow pressed? then → change y by 4 and if down arrow pressed? then → change y by -4 [Step 4].

Have students test their new code by trying to beat this level.

NOTE: The most common mistake is that they've mixed up their x's and y's. Remind students that x is the horizontal axis and y is the vertical axis.

LEVEL 3 – ROTATIONS

Have students change the backdrop to Level 3. They can do this by going to Looks and then clicking “next backdrop” or they can click on “Backdrops” in the “Stage” area and change to the Level 3 backdrop.

ENABLE ROTATIONS - STEP 5 IN QUICK REFERENCE

To beat Level 3, students will need to use a different kind of transformation. Ask students what kind of transformation they could use (rotation).

We already have a My Block (subprogram) for translations; now we will create one for rotations. Go to “My Blocks” then “Make a Block” and call it **enable rotations**. A block called **define enable rotations** will appear in your coding area.

Give students a chance to try figuring out what code they could add.

HINT: It will be very similar to the code we used for translations, just with different keys and a different blue movement block.

Answer: Add two if-then statements to define the enable rotations block. Choose two keys (e.g., Z and C because they can be controlled with the left hand easily) to represent counterclockwise and clockwise rotation. Add the appropriate “turn ___ degrees” blocks.

NOTE: A smaller number of degrees will be easier to control.

Example Solution: if key (z) pressed? then → turn ↺ 4 degrees and if key (c) pressed? then → turn ↻ 4 degrees [Step 5a]

IMPORTANT: Students also need to add their new pink “enable rotations” block into the main program under **enable translations** [Step 5b]. It can be found in My Blocks.

TEST AGAIN

Have the students test their latest controls (translations and rotations) by completing Level 3. Challenge them to see how fast they can go as you wait for the rest of the class to catch up.

LEVEL 4 – DILATIONS

Have students change the backdrop to Level 4. They can do this by going to Looks and then clicking “next backdrop” or they can click on “Backdrops” in the “Stage” area and change to the Level 4 backdrop.

ENABLE DILATIONS - STEP 6 IN QUICK REFERENCE

Ask students to identify the problem (Our rocket ship is too big).

What can we do to solve the problem? (Make it smaller). In math and geometry, what's it called when we make a shape larger or smaller? (Dilation).

Like before, go to My Blocks and create one called **enable dilations**. Again, give students a chance to try figuring out the code for themselves.

HINT: They will need to pick keys to use and then use a certain purple Looks block.

Answer: if key (d) pressed? then → change size by -1 and if key (a) pressed? then → change size by 1 [Step 6a].

NOTE: Students can choose any keys that have not yet been used. Encourage them to write down which key does what somewhere.

IMPORTANT: Students also need to add their new pink “**enable dilations**” block into the main program under **enable rotations** [Step 6b]. It can be found in My Blocks.

TEST AGAIN

Have the students test all their controls (translations, rotations and dilations) by completing Level 4. Challenge them to see how fast they can go as you wait for the rest of the class to catch up.

BONUS - CUSTOM LEVELS

At this point, the game should be fully functional and you can definitely stop here if you are running low on time. If you only have a few minutes left before you start the wrap up, encourage students to test out each other's games or tinker with their own controls.

If time permits, it's fun for students to create their own level.

In backdrops, you'll find **Level 5**. This level has a starting area for the spaceship, but it doesn't have a path. Students can create their own path using either the rectangle tool or the paintbrush tool. The colour should be pre-set to black, but any colour will work as long as it's not the colour of the orange lava.

Give students a chance to share their games with each other.

Finally, they could make their game automatically cycle through all the levels by putting all of their main program inside a **repeat (number of levels)** loop with a **next backdrop** block at the end. See Steps 7a and 7b in Quick Reference.

SLIDE 6 - RECAP

With about 5 minutes left in the class, switch back to the PowerPoint slides.

Ask students to recall what they learned, ask if anyone can remember the three types of transformation we used (translations, rotations, dilations).

Ask what the My Blocks are called in coding (subprograms or functions).

What is an event? (Some input like pressing a button or joystick that affects our program).

SLIDE 7 - POEM OF THE DAY

Share this week's poem as a recap.

SLIDE 8 - WHAT'S NEXT?

Quickly preview what you will be doing next week. The students will be turning back the clock to make an arcade classic.

Try **Offline Activity 2 - Songs with Subprograms** before the next online lesson.

QUICK REFERENCE

Make a maze game

Students will modify (“remix”) an existing Scratch file to create their own maze game called “The Walls are Made of Lava”. The maze is designed so that students will have to add new code to complete each level. Students will use conditional statements, subprograms, and defined count, while also learning about three types of transformation (translations, rotations, and dilations).

SET UP

- Have students log into their Scratch accounts and open the student link
- Open both the student version and the finished game on your computer. You will demo the finished version then work from the student version.

THE CODE

Here is the final code for each sprite. Areas in **GREY** are pre-coded for the students. Steps marked in **PINK** need to be added during the lesson. See the lesson plan on [Pages 7.19](#) to [7.22](#) for detailed instructions

NOTE: We will not be writing any code for the target sprite.

QUICK LINKS

Student Activity Link
[scratch.mit.edu/
projects/760367848](https://scratch.mit.edu/projects/760367848)

Finished Example
[scratch.mit.edu/
projects/818518940](https://scratch.mit.edu/projects/818518940)

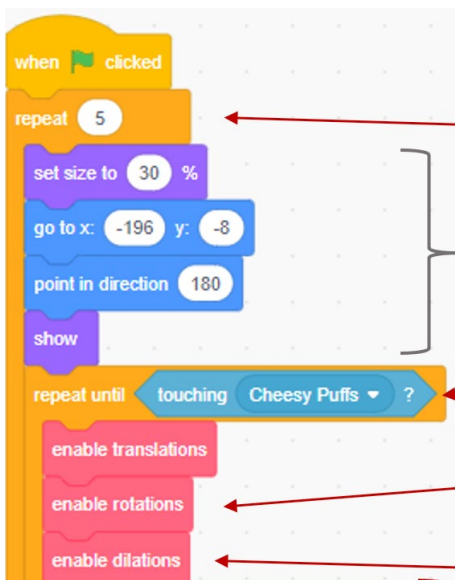
LEVEL COMPLETE SPRITE



This sprite is pre-made and coded for the students.

Sprite will hide when game begins and only appear once the player has reached the target sprite and the corresponding broadcast message has been sent.

SPACESHIP SPRITE - MAIN PROGRAM



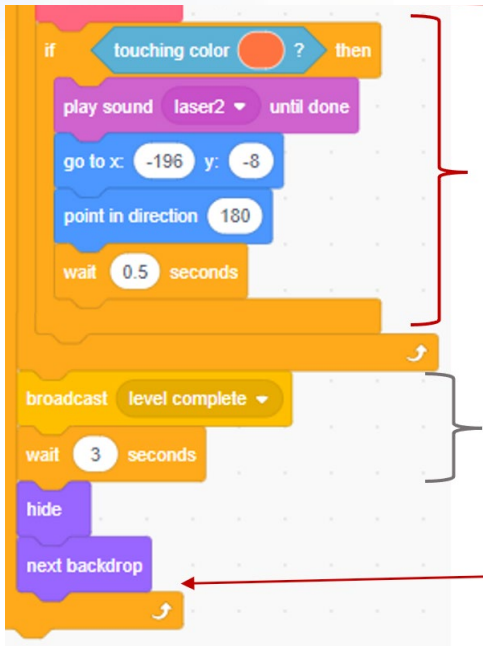
Step 7a. (Optional) Advance through the levels (backdrops) after each one finishes. Number in loop should equal the total number of backdrops.

Initializes sprite size, position, direction, and visibility.

Step 1. Define count by setting this end condition. Loop will repeat until reaching target sprite.

Step 5b. Remember to add this to the main program after writing the rotations subprogram.

Step 6b. Remember to add this to the main program after writing the dilations subprogram.



```

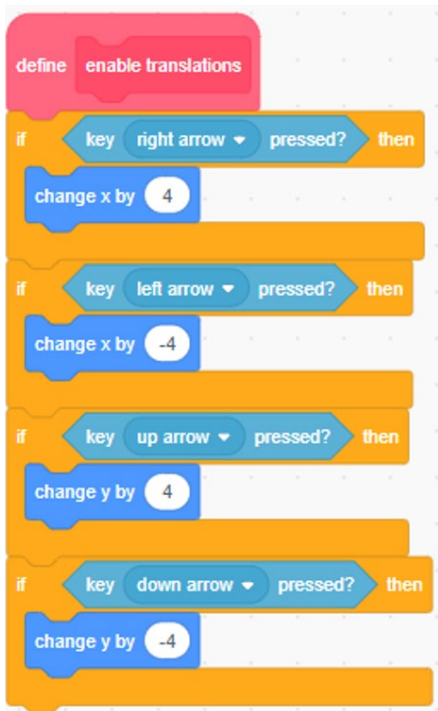
if touching color orange ? then
  play sound laser2 until done
  go to x: -196 y: -8
  point in direction 180
  wait 0.5 seconds
broadcast level complete
wait 3 seconds
hide
next backdrop
  
```

Step 3. Send the spaceship back to its starting position if it touches the orange “lava” walls at any point.

Sends a signal to the “Level Complete” sprite after the target is reached.

Step 7b. (Optional) Advance through the levels (backdrops) after each one finishes.

SUBPROGRAM 1 - TRANSLATIONS



```

define enable translations
  if key right arrow pressed? then
    change x by 4
  if key left arrow pressed? then
    change x by -4
  if key up arrow pressed? then
    change y by 4
  if key down arrow pressed? then
    change y by -4
  
```

Step 2. Fill in these two conditional statements to enable left/right translations.

Step 4. Add these two conditional statements to enable up/down translations.

SUBPROGRAM 2 - ROTATIONS

```

define enable rotations
if key z pressed? then
  turn 4 degrees
if key c pressed? then
  turn 4 degrees
  
```



Step 5a. Allows the sprite to rotate clockwise and counterclockwise.

SUBPROGRAM 3 - DILATIONS

```

define enable dilations
if key d pressed? then
  change size by -1
if key a pressed? then
  change size by 1
  
```



Step 6a. Allows the sprite to grow and shrink.

OFFLINE ACTIVITY 2

Songs with subprograms

45 MINUTES

Quickly compose simple songs using subprograms!

LEARNING OBJECTIVES

- Understand how to write and use subprograms
- Experiment with song structures

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)
- Arts C1 (Creating and Performing Music)

SET-UP

Instruments are optional but not required. If you don't use instruments, students can make percussion songs using claps, stomps, snaps, etc. Print and hand out worksheets.

INSTRUCTIONS

SUMMARY

The parts of a song (verse, chorus, bridge, etc.) are an everyday example of subprograms. Students will compose these musical subprograms before assembling them into a larger song.

1. Ask students to recall what they learned in the online coding classes.
ASK "What is a subprogram?" (A section of code that completes a specific task) "Why do we use subprograms?" (It makes our code shorter and faster to write because we don't need to write the same instructions multiple times)
2. Explain the premise of the activity. **SAY** "In the first part of this activity, you and your group will write three different parts of a song – a verse, a chorus and a bridge. These parts are like subprograms in coding. In the second half of the activity, we will put the parts (i.e., subprograms) together to make an entire song (i.e., main program)."

MATERIALS

Each student will need:

- 1 worksheet
- Optional - Small musical instruments like shakers or kazoos

- Discuss the three song parts.

The **verse** is usually how the song starts. It sets the tone for the song. The **chorus** is the main idea of the song. It often repeats several times and it's the catchiest part of the song. A **bridge** is used to connect other song parts. It sounds different to the other parts and adds variety to the song.

Part 1 – Writing subprograms

- Demonstrate what a song part (i.e., subprogram) might sound like by clapping out a pattern. For example, the song “We Will Rock You” has a simple and easily recognizable clapping pattern.
- Split students into groups of 2 or 3. Each group will compose a verse, a chorus and a bridge. They will use their worksheets to keep track of their ideas. For simplicity, their songs do not need any lyrics; just percussion is fine. Give students time to write their three song sections.

Part 2 – Making main programs

- You will now give students a song structure (i.e., main program). They will plug their song elements into your structure to make a complete song. Begin with this common song structure:

Verse
Chorus
Verse
Chorus

- Circle the room and help students as needed. Have them perform their songs for you.
- Write some more song structures on the board for the students to use. Other common structures include:

Verse
Verse
Chorus
Verse

Verse
Chorus
Verse
Chorus
Bridge
Chorus

- If time allows, have students perform their songs for the class. They may also choose to come up with their own song structure.
- Debrief the activity using the discussion questions as a guide.

CROSS-CURRICULAR CONNECTION

As an extension to the music curriculum, you could ask students to identify the parts (verse, chorus, bridge, etc.) of their favourite songs.

DISCUSSION QUESTIONS

What do the song sections (verse, chorus, bridge) represent in coding?

Answer: subprograms.

What does our song structure represent?

Answer: the main program.

Why did we write our song parts first? How did that help us?

Possible answers: It made our songwriting more efficient because we could make many different songs using the same parts. We didn't need to write out the same chorus multiple times.

Lots of popular songs use the same song structure. Why do you think that is? Do you think coders borrow from each other's programs?

Possible answers: Certain structures are known to work (i.e., it sounds good). Coders borrow things like subprograms (functions) that are known to work well.

ACTIVITY 2 WORKSHEET

Songs with subprograms

You will start by writing three different parts of a song – a verse, a chorus, and a bridge. Later, you will combine those parts to make a complete song.

VERSE

Write notes to help you remember your verse.

Example:

Clap x2, slap knees x2, stomp left foot then right foot x4, knees once, clap once, knees x2, snap fingers.

CHORUS

Write your chorus here. It should be different than the verse. The chorus is usually the “catchiest” part of a song.

BRIDGE

Write your bridge here. The bridge often sounds quite different than both the verse and the chorus. For example, you could make it slower or use different sounds.

ONLINE LESSON 3

Single-player Pong

60 MINUTES

Students will turn back the clock to make their own version of *Pong*. In this single-player game, players will keep a bouncing ball up for as long as possible. Students will write a “bounce” subprogram, build control structures with loops and conditional statements, and apply some important math concepts.

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing efficient code, including code that involves events influenced by a defined count and/or subprogram and other control structures
- **Math C3.2** – read and alter existing code, including code that involves events influenced by a defined count and/or subprogram and other control structures, and describe how changes to the code affect the outcomes and the efficiency of the code

OPERATIONS

- **Math B2.1** – use the properties and order of operations, and the relationships between operations, to solve problems involving whole numbers, decimal numbers, fractions, ratios, rates, and percents, including those requiring multiple steps or multiple operations

GEOMETRIC AND SPATIAL REASONING

- **Math E1.4** - describe and perform translations, reflections, and rotations on a Cartesian plane, and predict the results of these transformations

QUICK LINKS

Student Activity Link
scratch.mit.edu/projects/762298370

Finished Example
scratch.mit.edu/projects/818524819

PowerPoint
[Grade 7 – Week 3 – Video Game History](#)

Quick Reference
[Single-Player Pong – Quick Reference](#)

LESSON BREAKDOWN

SLIDE 1 - WEEKS 1 AND 2 REVIEW

Open the PowerPoint slides and the Scratch links on your own computer. Project for the students to see. Open or print the **Single-Player Pong - Quick Reference** for your own use during the lesson.

Ask students if they remember what a subprogram is (A section of code that performs a specific task) Why do we use them? (To make our code cleaner to read and to save us from having to write the same code many times).

SLIDES 2 TO 7 - HISTORY MINUTE

Use slides to discuss the history of video games, specifically *Pong*. See the slide notes for talking points.

DEMO

Quickly show students what the completed game looks like to give them an idea of what they are working towards.

Link to finished game:

scratch.mit.edu/projects/818524819

SLIDE 8 - LOG INTO SCRATCH

Have students log into their Scratch accounts and open the activity link.

Have them click “Remix” to make their own version of the project.

They can also rename their projects to something of their choosing.

MOVE THE PADDLE BACK AND FORTH - STEP 1 IN QUICK REFERENCE

Four sprites are pre-made for the students including the paddle sprite. To make the paddle move left and right students will need to fill in the conditional statements. This should be review for them because they did something very similar in Week 2 (Lava Game).

Answer: if key (right arrow) pressed? then → change x by 10 and if key (left arrow) pressed? then → change x by -10.

CHOOSE AND INITIALIZE BOUNCING SPRITE - STEP 2 IN QUICK REFERENCE

Let students choose a sprite to be their bouncing sprite (e.g., ball). Resize the sprite to something appropriate in the sprite properties then add the following code:

when green flag clicked → go to x: 0 y:0 → point in direction 30 [Step 2]

For the direction, 30 is our suggestion, but other angles would work well too (except 0, 90, 180, -90).

MAKE THE SPRITE MOVE - STEP 3 IN QUICK REFERENCE

Now it’s time to make our ball sprite move. To do this add, a **forever** loop under the initialization code and inside it put **move 10 steps → if on edge, bounce** [Step 3].

Have students hit the green flag and test what happens. At this point, their paddle should be able to move left and right and their ball should be bouncing around the screen. If the ball isn’t bouncing in a nice direction, have them change the starting direction.

MAKE A “BOUNCE” SUBPROGRAM - STEP 4 IN QUICK REFERENCE

So far, the ball bounces when it hits the edge, but we also want it to bounce when it hits our paddle. To do this, let’s write a “bounce” subprogram.

Go to My Blocks and create a new block called **bounce**. Under **define bounce**, we will add **if touching paddle? then → turn 180 degrees → move 15 steps → wait 0.02 seconds** [Step 4a].

Now when the ball hits the paddle, it will turn around and start heading back the way it came. Note that this isn’t technically the same way that “if on edge, bounce” works but this is simpler to code.

IMPORTANT: Remember to add the new **bounce** block into the main program’s forever loop, right after “if on edge, bounce” [Step 4b].

SCORING POINTS - STEP 5 IN QUICK REFERENCE

We can make this a true game by introducing a score variable.

Go to variables and then “Make a Variable” called **Score**. At the start of the program (under the “when green flag clicked”) add **set Score to 0** [Step 5a] then add **change Score by 1** into your “bounce” subprogram under “if touching paddle” [Step 5b].

PLAY TESTING

Give the students a few minutes to test out their games and offer assistance to anyone who needs it. At this point, players should be able to score points every time they hit the ball sprite with their paddle. Students can also play around with the speed of the ball sprite if they’d like (e.g. **move 15 steps** instead of move 10 steps).

ADD END GAME - STEP 6 IN QUICK REFERENCE

Now we will make the game end when the player misses the ball. To do this, we have added a red “Line” sprite along the bottom of the screen.

Inside the forever loop of the ball’s main program (under the “bounce” command), add **if touching Line then → broadcast (game over) → hide** [Step 6].

The “Game Over” sprite is pre-programmed to appear and stop all scripts when it receives the “game over” broadcast message.

NOTE: Students will also need to add a **show** command at the start of their main program so that their ball will re-appear at the start of the next game.

MORE PLAY TESTING

The students’ games should now be fully playable and you can give them some time here to test them out. Feel free to stop the lesson here if you are running low on time, or you can proceed to the final steps: adding more levels and changing the game speed.

OPTIONAL - ADD MORE LEVELS - STEP 7 IN QUICK REFERENCE

If time allows, we can refine our game by adding more levels. For each new level, we will change the backdrop and increase the speed of both our ball sprite and paddle sprite.

To add more levels, we will start by creating a new subprogram to check the score. This function will test if a certain target score is reached and change the level accordingly.

Go to My Blocks and “Make a Block” called **check score**. Under **define check score** add **if (score mod 5) = 0 then → broadcast (next level)** [Step 7a]. Next, add your new **check score** block into the **bounce** subprogram right after “change score by 1” [Step 7b].

Show students how to add the code first, then take some time to explain what “mod” does.

EXPLANATION: The operation “mod” (short for modulo) tells us the remainder when we divide by a certain number. For example, 7 mod 2 means “what is the remainder when you divide 7 by 2?” In this example, the answer is 1. In our code, we are taking “score mod 5” – that is, we are dividing the score by 5 and checking what the remainder is. If the remainder = 0, then our score must be a multiple of 5. If you want the level to change every 8 points, for example, you could make it mod 8 instead of mod 5.

ADD A SPEED VARIABLE - STEP 8 IN QUICK REFERENCE

The final thing we will do is add a speed variable so that the game gets faster each level.

First, create a new variable called “**speed**” and make it hidden.

At the start of the program, add “**set speed to 10**” [Step 8a]. This is a fairly slow starting speed, but they could make it even slower if they’d like.

Inside your “check score” subprogram, add “**change speed by 2**” into the conditional statement [Step 8b]. Now the speed variable will increase by 2 each level.

Finally, we need to use our **speed** variable within our program. In the main program, replace “move 10 steps” with **move (speed) steps** [Step 8c] and in the paddle code replace “change x by 10” with **change x by (speed)** and replace “change x by -10” with **change x by -1* (speed)** [Step 8d].

Note that you need to use a multiplication block to make the speed negative for the left arrow key.

BONUS - CUSTOM LEVELS

Here are some suggestions if you still have time for students to experiment.

Add more levels

We added 6 backdrops to start, but students are free to add more. We created costumes for the “Levels” sprite up to Level 10.

Tinker with speeds

Students can increase the speed by more or less each level. They can start at a slower or faster speed.

Sound effects

It can quickly get annoying, but students may want to add a sound effect when they level up or when they hit the ball.

SLIDE 9 - RECAP

With about 5 minutes left in the class, switch back to the PowerPoint slides.

Ask students to recall what they learned during the program. Possible topics: repeat until vs repeat ___ times, subprograms/functions, conditional statements.

SLIDES 10 AND 11 - FINAL PROJECT PREVIEW

Use the slides to explain the final project.

“You will be applying everything we’ve talked about to make your own game. You can keep going with one of the ones we made together and make it even better, or you can create something new.”

Remind students about the three games you made together and where they can find their saved projects in Scratch.

When students are done, they will share their work for you to evaluate.

As a bonus, NII Explore is always accepting submissions to our “Scratchathon” contest. Email your students’ projects to explore@nii.ca - we hand out prizes periodically for some of our favourite games!

SLIDE 12 - POEM OF THE DAY

Share the final Poem of the Day.

SLIDE 13 - WHAT’S NEXT?

Complete the final project in the next couple weeks while this lesson is still fresh. If you haven’t already, try out the offline activities with your class. Happy coding!

QUICK REFERENCE

Single-player Pong

Students will turn back the clock to make their own version of *Pong*. In this single-player game, players will keep a bouncing ball up for as long as possible. Students will write a “bounce” subprogram, build control structures with loops and conditional statements, and apply some important math concepts.

SET UP

- Have students log into their Scratch accounts and open student link
- Open both the student version and the finished game on your computer. You will demo the finished version then work from the student version.

THE CODE

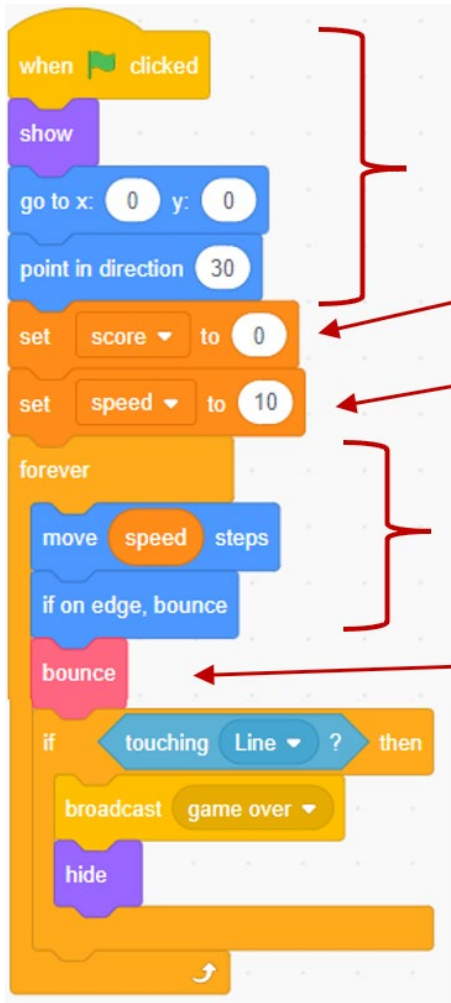
Here is the final code for all sprites. Parts annotated in **GREY** will be pre-populated for the students. Steps marked in **PINK** indicate code that you will be adding and the numbers correspond with the suggested order of steps from the from the lesson plan on **Pages 7.31 to 7.34**.

QUICK LINKS

Student Activity Link
[scratch.mit.edu/
projects/762298370](https://scratch.mit.edu/projects/762298370)

Finished Example
[scratch.mit.edu/
projects/818524819](https://scratch.mit.edu/projects/818524819)

SPRITE 1 – BOUNCING SPRITE



```

when clicked
  show
  go to x: 0 y: 0
  point in direction 30
  set score to 0
  set speed to 10
  forever
    move speed steps
    if on edge, bounce
    bounce
    if touching Line ? then
      broadcast game over
      hide
  
```

Step 2b – Start the sprite in a random direction.

Step 5a – Set the score to 0 when game starts.

Step 8a – Choose a starting speed.

Step 3 – Makes the sprite move and then bounce when it hits the edge (**Note:** It will say move “10” steps when you first write this code).

Step 8c – Replace the 10 with “speed”. Now the ball will speed up each level.

Step 4b – Add the new bounce subprogram into the main program.

Step 6 – Trigger the end game message and hide the ball if it hits the line at the bottom of the screen.

Step 4a – Write this subprogram to make the ball bounce when it hits the paddle. The two blocks about the score will be added later.

```

define bounce
  if touching Paddle ? then
    change score by 1
    check score
    turn 180 degrees
    move 15 steps
    wait 0.02 seconds
  
```

Step 5b – Increase score whenever ball hits the paddle.

Step 7b – Add the check score subprogram such that it checks immediately after scoring a point.

```

define check score
  if score mod 5 = 0 then
    change speed by 2
    broadcast next level
  
```

OPTIONAL

Step 7a – Check to see if the score is divisible by 5 (or 10) and then change the speed and level when it is.

Step 8b – Increase the speed variable each time a new level is reached.

SPRITE 2 – PADDLE

```

when green flag clicked
  forever loop
    if key right arrow pressed? then
      change x by speed
    if key left arrow pressed? then
      change x by -1 * speed
  
```

Controls the motion of the paddle.

Step 1 – When you first add code here, you will put in “10” for the right arrow and “-10” for the left arrow.

Step 8d – Replace the 10 and -10 with speed and -1*speed, respectively. This will allow the paddle to move faster each level to keep up with the faster ball.

SPRITE 3 – LINE

```

when green flag clicked
  go to x: 0 y: -180
  set drag mode to not draggable
  
```

Sets sprite location (bottom of screen), prevents sprite from being moved.

SPRITE 4 – GAME OVER

```

when clicked
  go to x: 0 y: -10
  hide

when I receive game over
  show
  stop all
  
```

Initializes sprite position and makes it hidden at the start of the game.

Game Over message appears when the “game over” broadcast message is received. Stops all scripts in the entire project.

SPRITE 5 – LEVELS

```

when clicked
  switch costume to Level 1
  go to x: -177 y: 153
  set drag mode not draggable

when I receive next level
  next costume
  
```

Initializes the sprite’s position and costume (Level 1) and can’t be moved.

Switches costumes when the level changes.

OFFLINE ACTIVITY 3

Video game design

30–45 MINUTES

Students will prepare for their final coding project by creating a Game Design Document.

LEARNING OBJECTIVES

- Create a plan for a coding project
- Communicate ideas through writing and concept art

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)
- Language – Writing, Overall Expectation 1
- Arts D1 (Visual Arts)

SET-UP

Print the worksheets for the Game Design Document and Final Project on the front and back of a single page. Students will use the front for this activity and the back for their final projects.

INSTRUCTIONS

SUMMARY

Video game developers often create a Game Design Document (GDD) when making a new game. The document serves as a guiding vision for all teams working on the project. In this activity, students will complete a simple GDD. In the final project, they will use their GDD as a guide to create their own game.

1. Explain the premise of the activity. **SAY** “We are going to make our own games like we practiced in our coding classes. Today, you are going to create something called a ‘Game Design Document’. The next time we have the computers, you will be coding your game in Scratch.”
2. Ask students to recall what they learned in the online coding classes. **ASK** “What kind of games did we make?” (Clicker game, Maze game, Pong game) “What kind of games do you like to play?”

MATERIALS

Each student will need:

- 1 worksheet
- Something to write with

3. Hand out worksheets and go through each section of the Game Design Document. The students can either customize and improve one of the games they made during the online classes (recommended) or create a new game from scratch (pun intended).
4. Give students time to fill in the document with their ideas. You can use the guiding questions below to answer questions about the document or help students when they get stuck.

Title

What is your game called? Does it sound like something you'd want to play?

Description

What is the game about? How would you explain it to a friend?

Characters

What characters (sprites) are in your game?
What do they look like? What do they do?

Setting

Where does the game take place? What does the backdrop look like? Does the backdrop ever change?

Gameplay

What is the goal of the game? Which sprite(s) does the player control? How do they control it? How many players are there? How do you score points? How does the game end?

Extras

What extra features will you add to your game?
Examples: sound effects, animations, different levels, more characters, leaderboard, "You Win" message

5. When students are finished their GDDs, collect the worksheets. You will hand them back when you have computer time for the final project.

ACTIVITY 3 WORKSHEET

Game Design Document

Use this sheet to help plan your own video game. You can use one of your existing games as a starting point or you can make a different game. Later, you will make your game in Scratch.

NAME _____

GAME TITLE

DESCRIPTION What's the game about?

GAMEPLAY Explain what happens in the game. How does the player control their sprite? Score points? Win the game?

CHARACTERS Sketch and describe them here.

SETTING Where does the game happen?

EXTRAS

FINAL PROJECT WORKSHEET

Make your own game

Now it's time to turn your Game Design Document into your own game. You can make a customized version of one of the games from the online classes (Clicker Game, Maze Game or Pong Game) or you can create something new. As a coder, it's often okay to get ideas from other people's code as long as you give them credit and your final game is your own work.

You can add as much detail as you want, but make sure your finished game includes:

- A player-controlled sprite
- At least 2 types of motion, including a change of x and y coordinates
- A score variable
- A conditional statement (if-then or if-then-else)
- A subprogram (or "My Block")
- An ending (e.g., "You Win" or "Game Over" message)
- A repeat loop with defined count

QUESTIONS

After you finish and submit your game, answer the reflection questions below.

1. Describe any new feature(s) you included in your game.

2. What subprogram (My Block) did you make? What does it do?

3. What is one problem you had while making your game? How did you try to fix it?

4. If you had more time, what would you add to your game?

FINAL PROJECT

Make your own game

60 MINUTES +

Students will apply their coding knowledge to make their own custom game. It's a fun opportunity to demonstrate what they've learned. Classes can submit their finished games to NII Explore at explore@nii.ca!

LEARNING OBJECTIVES

- Build, test, and improve a game in Scratch
- Write and edit code that includes the analysis of data
- Write a game description and instructions for other users

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)
- Media Literacy 3.4

SET-UP

Before class, check out [this video explanation](#) of the project. Have students log onto computers and open their Scratch accounts. Hand back the Game Design Documents from Activity 3.

INSTRUCTIONS

SUMMARY

Your students will be using the Game Design Documents they completed in Activity 3 to make their own game in Scratch. They can make a custom version of one of the games you made together (recommended) or make their own game.

1. Explain challenge to students. "You will be using the Game Design Documents you made last time to create your games in Scratch."
2. Review the final project instructions on the back of the student worksheets (Final Project Worksheet). It outlines the goal of the project and what elements should be included in their code.

MATERIALS

Each student will need:

- Laptop or tablet
- Scratch account
- Game Design Document from Activity 3

3. Students can open their previous games by clicking on the file folder in Scratch. If students want to view NII Explore’s version of the games on their computers, they can search “*NIIExplore2*” on Scratch and access our shared files. Remind students that it is okay to look at code from another game, but they shouldn’t copy someone else’s game exactly. They need to make it their own.
4. Give students lots of time to work on their games. Remind them to rename their projects and save their work often. Scratch will save their work automatically as long as they are logged into their account. If they lose their work at any point, they can try CTRL+Z to undo the most recent action, or go to “Edit” then “Restore”.
5. If students need more time, consider giving them another computer period.
6. **PROJECT SUBMISSION** When students are finished, they will share their projects with you. Be sure to save about 15 minutes for this process. First, they will click the orange “Share” button. Next, have them fill in the “Instructions” box that appears. Finally, they will click “Copy Link” and send the link to you via email or your online classroom (e.g., Brightspace or Google Classroom).
7. After submitting their Scratch projects, ask students to complete the written questions on their worksheet and hand them in. You may use the **ASSESSMENT FRAMEWORK** on the following pages to evaluate your students’ work.
9. If you haven’t already, try the offline activities with your class. They are meant as a fun way to reinforce coding ideas without needing computers.

FINAL PROJECT

Assessment and evaluation

When students have sent you their Scratch projects, you will be able to try their game by clicking the green flag. Next, you can click “See Inside” to view their code. Check that the students used the following elements in their game as outlined on their worksheets:

- A player-controlled sprite**
As the game’s player, is there a sprite that you can control? How do you control the sprite?
- A score variable**
Click on the “Variables” tab on the left. Do they have a score variable? Does it appear during the game?
- A subprogram (or “My Block”)**
Click on the “My Blocks” tab on the left. Have they created a pink My Block? What does it do? Where does it appear in their main program?
- A repeat loop with defined count**
Did they use a light orange repeat loop? Does it repeat for a specific number of times or until a condition is met?
- At least 2 types of motion, including a change of x and y coordinates**
Does a sprite undergo different types of transformation (e.g., rotation, reflection, translations, growing/shrinking)? Does the student show an understanding of xy-coordinates on the Cartesian plane?
- A conditional statement** (*if-then or if-else-then*)
Has the student included a light orange conditional statement? What does it control?
- An ending** (e.g., “You Win” or “Game Over” message)
How does the game reach an end? What happens when the end is reached?

Students will probably use more elements than the ones listed above, but these are the ones specified on their worksheets.

Read through both their code and their worksheet responses, then use the **ASSESSMENT FRAMEWORK** on the next page to evaluate their work.

Assessment framework

This chart will help you assess your students' work during the Final Project and the *Coding in the Classroom* program as a whole. It is based on the Ontario Mathematics (2020) curriculum.

KNOWLEDGE AND UNDERSTANDING

	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
Knowledge of content	Correctly uses few required elements in final code Does not answer questions during online classes, even with assistance	Correctly uses most required elements in final code Answers questions during online classes with some assistance	Correctly uses all required elements in final code Answers some questions during online classes	Correctly uses all required elements and some others in final code Answers many questions during online classes
Understanding of content	Rarely uses subprograms when appropriate Rarely uses defined count when appropriate	Sometimes uses subprograms when appropriate Sometimes uses one type of defined count (FOR or UNTIL) when appropriate	Often uses subprograms when appropriate Often uses defined count when appropriate	Always uses subprograms when appropriate Uses both types of defined count (FOR and UNTIL) when appropriate

THINKING

	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
Use of planning skills	Creates Game Design Document with few of the required elements	Creates Game Design Document with some of the required elements	Creates Game Design Document with most of the required elements	Creates Game Design Document with all of the required elements
Use of processing skills	Uses code to convert Game Design Document into finished game with limited effectiveness	Uses code to convert Game Design Document into finished game with some effectiveness	Uses code to convert Game Design Document into finished game with considerable effectiveness	Uses code to convert Game Design Document into finished game with high degree of effectiveness
Use of critical/creative thinking processes	Troubleshoots and "debugs" code with much assistance Re-creates one of the example games	Troubleshoots and "debugs" code with assistance Creates new game by modifying existing features	Troubleshoots and "debugs" code with some assistance Creates game with a new feature	Troubleshoots and "debugs" code with little assistance Creates game with several new features

COMMUNICATION

	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
Expression and organization of ideas and information in oral, visual, and/or written forms	Uses some concept art or written descriptions Game Design Document is not clearly organized	Uses concept art or written descriptions to create somewhat organized Game Design Document	Uses concept art and written descriptions to create organized Game Design Document	Uses concept art and written descriptions to create highly organized Game Design Document
Communication for different audiences and purposes in oral, visual, and/or written forms	Explains code and gameplay, either orally or in writing, with limited effectiveness	Explains code and gameplay, either orally or in writing, with some effectiveness	Explains code and gameplay, either orally or in writing, with considerable effectiveness	Explains code and gameplay, either orally or in writing, with a high degree of effectiveness

APPLICATION

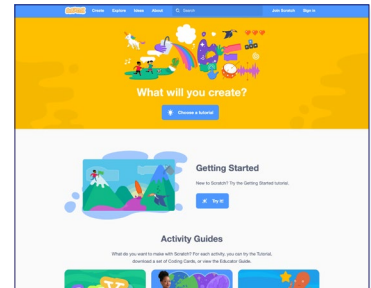
	LEVEL 1	LEVEL 2	LEVEL 3	LEVEL 4
Application of knowledge and skills in familiar contexts	Follows coding lessons with much assistance	Follows coding lessons with assistance	Follows coding lessons with some assistance	Follows coding lessons with little or no assistance
Application of knowledge and skills to new contexts	Applies coding knowledge to make custom game with much assistance	Applies coding knowledge to make custom game with assistance	Applies coding knowledge to make custom game with some assistance	Applies coding knowledge to make custom game with little or no assistance
Making connections within and between various contexts	Rarely participates in offline coding activities Rarely makes connections between coding concepts and everyday life	Participates somewhat in offline coding activities Sometimes makes connections between coding concepts and everyday life	Participates in offline coding activities Makes connections between coding concepts and everyday life	Participates fully in offline coding activities Often makes connections between coding concepts and everyday life

Additional resources

SCRATCH

Scratch has a series of activity guides under the “Ideas” tab. There are also countless tutorials available on YouTube.

scratch.mit.edu/ideas

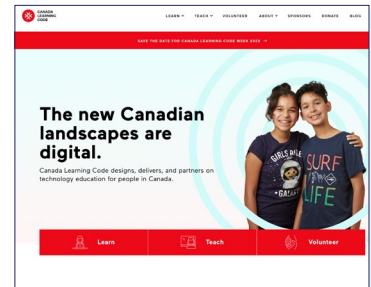


Scratch

CANADA LEARNING CODE

From lesson plans to professional development, this website has a wealth of resources for teaching coding.

canadalearningcode.ca

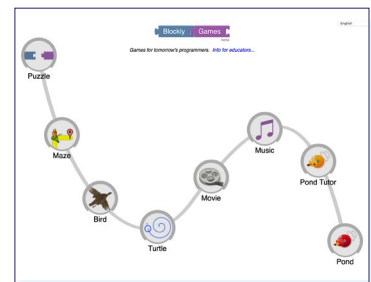


Canada Learning Code

BLOCKLY GAMES

These coding games cover a range of topics. Blockly offers a mix of block-based and text-based coding. The later levels of some lessons are quite tricky, but the first few levels should be accessible for Grade 7 students.

blockly.games



Blockly Games

*Coding screenshots are sourced from scratch.mit.edu/