GRADE 6 — Efficient Code
GRADE 7 — Subprograms and Defined Count
GRADE 8 — Analysis of Data

{CODING} IN THE CLASSROOM

GRADE 8 TEACHERS' GUIDE

Analysis of data

Welcome to NII Explore's *Coding in the Classroom* program for Grade 8 students. During the 4-week program, you and your class will complete:

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

This teacher guide includes everything you need to get started!

SCHEDULE AT A GLANCE

WEEK 1
- **Online Lesson 1**
  Intro to Scratch
- **Classroom Activity 1**
  Fun with Machine Learning

WEEK 2
- **Online Lesson 2**
  Make a Racing Game
- **Classroom Activity 2**
  Random Rollers

WEEK 3
- **Online Lesson 3**
  Two-Player Pong
- **Classroom Activity 3**
  Video Game Design

WEEK 4
- **Final Project**
  Make Your Own Game

THE GRADE 8 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 Grades 5 to 7, students learn to use conditional statements, loops, and subprograms.

In **GRADE 8,** students begin using code for the **analysis of data**. Coding is often used to analyze data because computers can store and manipulate large amounts of information easily.

We are living in the Information Age with many careers in coding and data science. Even if students don't work with data directly, a basic understanding of coding and data analysis will help them navigate an increasingly data-driven world.

Data comes in many forms, but we will focus on three types: numbers, strings, and Boolean data. **Numbers** are what we usually think of when we hear "data". Numerical data can be stored as integers (i.e., positive or negative whole numbers) or as decimals (called floats or doubles).

Strings are sequences of letters or other characters. Your computer password is a good example of a string.

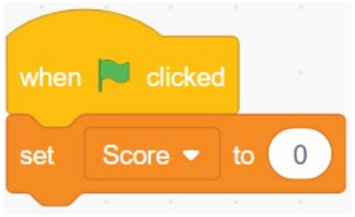
Finally, **Boolean data** is data that only has two possible values. For example, if we ask students a question like "Do you like ice cream?", their answers – either yes or no – would represent Boolean data.

DATA TYPE EXAMPLES

Numbers	Strings	Boolean
1 -5 12.27	**"abcdef"**	**True**
	"Coding 4 Ever"	**False**

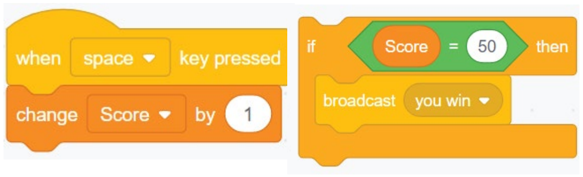
In Scratch, the platform we use for *Coding in the Classroom*, data can be stored as variables or in structures called lists.

Variables are single pieces of data that we tell our computer to remember. For example, we might create a variable called "Score" and set it to 0 at the start of our program.



```
when [flag] clicked
set Score ▾ to 0
```

Later, we can update the value of our variable or call on it for some purpose.

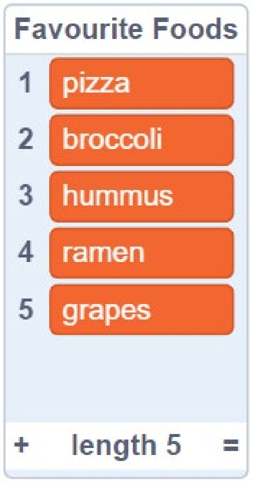


```
when space ▾ key pressed
change Score ▾ by 1
```

```
if Score = 50 then
    broadcast you win ▾
```

Coding with variables. *In the first example, the Score variable is increased by 1 whenever the space bar is pressed. In the second example, the program calls the Score variable and checks if it is equal to 50.*

Lists are multiple pieces of data stored in a particular order. Each piece of data in the list is assigned an item number as seen in the following example:

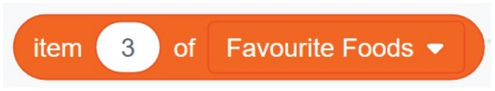


Favourite Foods

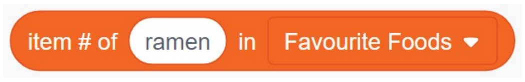
1	pizza
2	broccoli
3	hummus
4	ramen
5	grapes

+ length 5 =

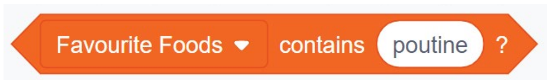
Once our data is stored, we can retrieve information about the list. Can you figure out what value would be returned from each of these commands? See bottom of page for answers.

1) 

```
item 3 of Favourite Foods ▾
```

2) 

```
item # of ramen in Favourite Foods ▾
```

3) 

```
Favourite Foods ▾ contains poutine ?
```

Although Scratch isn't typically used for data analysis, it is an excellent learning tool for new coders. The concepts of variables and lists are transferable to other software and coding languages.

Answers. *(1) hummus; (2) 4; (3) false – the list does not contain "poutine".*

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 variables and lists 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 "Classroom Activity 1 – *Fun with Machine Learning*" before next online session.

See **Page 8.5** for lesson instructions and **Page 8.10** for a quick reference guide.

Classroom Activity 1 – Fun with Machine Learning

Students will explore the power of machine learning by training a model to correctly classify their drawings.

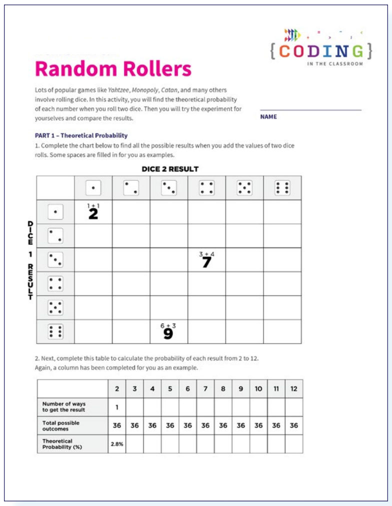
See **Page 8.15** for activity instructions.

WEEK 2

Online Lesson 2 – Make a Racing Game

In this activity, students will edit an incomplete Scratch file to make a working game. Students will use conditional statements and variables to control their racing sprites.

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



Random Rollers

Lots of popular games like *Yahtzee*, *Monopoly*, *Catan*, and many others involve rolling dice. In this activity, you will find the theoretical probability of each number when you roll two dice. Then you will try the experiment for yourselves and compare the results.

NAME

PART 1 – Theoretical Probability

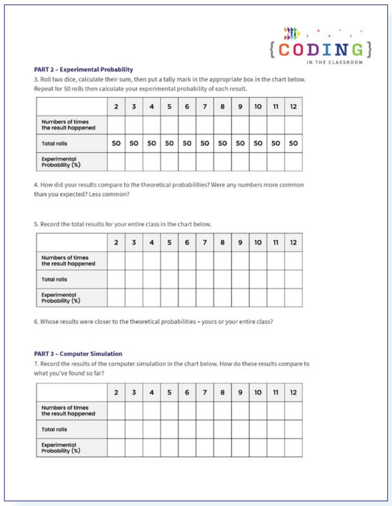
1. Complete the chart below to find all the possible results when you add the values of two dice rolls. Some spaces are filled in for you as examples.

DICE 2 RESULT

DICE 1 RESULT	⚀	⚁	⚂	⚃	⚄	⚅
⚀	1 + 1 2					
⚁						
⚂				3 + 4 7		
⚃						
⚄						
⚅			6 + 3 9			

2. Next, complete this table to calculate the probability of each result from 2 to 12. Again, a column has been completed for you as an example.

	2	3	4	5	6	7	8	9	10	11	12
Number of ways to get the result	1										
Total possible outcomes	36	36	36	36	36	36	36	36	36	36	36
Theoretical Probability (%)	2.8%										



PART 2 – Experimental Probability

3. Roll two dice, calculate their sum, then put a tally mark in the appropriate box in the chart below. Repeat for 50 rolls then calculate your experimental probability of each result.

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls	50	50	50	50	50	50	50	50	50	50	50
Experimental Probability (%)											

4. How did your results compare to the theoretical probabilities? Were any numbers more common than you expected? Less common?

5. Record the total results for your entire class in the chart below.

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls											
Experimental Probability (%)											

6. Whose results were closer to the theoretical probabilities – yours or your entire class?

PART 3 – Computer Simulation

7. Record the results of the computer simulation in the chart below. How do these results compare to what you've found so far?

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls											
Experimental Probability (%)											

***Week 2** Random Rollers*

POST Complete “Classroom Activity 2 – *Random Rollers*” before next online session.

See **Page 8.18** for lesson instructions and **Page 8.23** for a quick reference guide.

Classroom Activity 2 – Random Rollers

Your class will compare theoretical and experimental probabilities in dice rolling games and use a dice roll simulator.

See **Page 8.26** for activity instructions.

WEEK 3

Online Lesson 3 – Two-Player Pong

Students will turn back the clock to make their own version of ***Pong.*** The lesson will explore variable types and apply transformations on a Cartesian plane.

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

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

See **Page 8.31** for lesson instructions and **Page 8.36** for a quick reference guide.

Classroom Activity 3 – Video Game Design

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

See **Page 8.40** for activity instructions.

WEEK 4

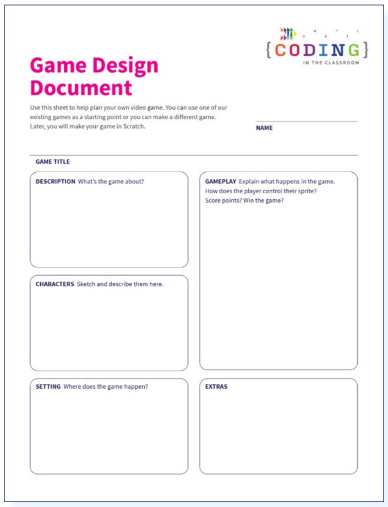
Final Project – Make Your Own Game

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 8.44** for project instructions.

AFTER WEEK 4

Keep the coding going with the additional resources on **Page 8.49**!


{CODING}
IN THE CLASSROOM

Game Design Document

Use this sheet to help plan your own 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?

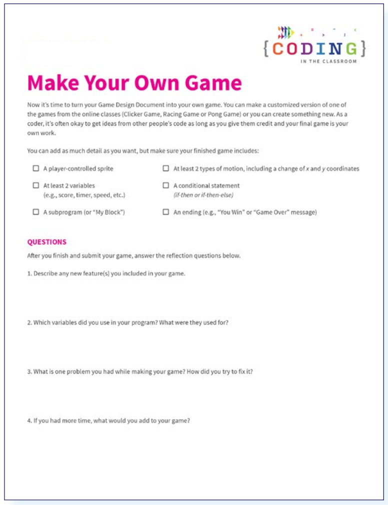
GAMEPLAY Explain what happens in the game. How does the player control their sprite? Score points? Win the game?

CHARACTERS Sketch and describe them here.

SETTING Where does the game happen?

EXTRAS

Week 3 *Game Design Document*


{CODING}
IN THE CLASSROOM

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, Racing 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 variables (e.g., score, timer, speed, etc.)
- ☐ A subprogram (or “My Block”)
- ☐ At least 2 types of motion, including a change of *x* and *y* coordinates
- ☐ A conditional statement *(if-then or if-then-else)*
- ☐ An ending (e.g., “You Win” or “Game Over” message)

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. Which variables did you use in your program? What were they used for?

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?

Week 4 *Make Your Own Game*

ONLINE LESSON 1

Intro to Scratch

(Clicker game with leaderboard)

60 MINUTES

The three online lessons and final project all use Scratch. Though it isn't typically used for data analysis, Scratch does allow us to store data as either variables or lists. If you are new to Scratch, you can check out their "**Getting Started**" tutorial. You can also learn more about Scratch variables **here** and lists **here**.

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 variables and lists to create a clicker game with a leaderboard.

QUICK LINKS

Student Activity Link
scratch.mit.edu

Finished Example
scratch.mit.edu/projects/818520725

PowerPoint
Grade 8 – Week 1 Intro to Coding

Quick Reference
Intro to Scratch (Clicker Game with Leaderboard) - Quick Reference

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions
- **Math C3.2** – read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code

DATA LITERACY

- **Math D1.2** – collect continuous data to answer questions of interest involving two variables, and organize the data sets as appropriate in a table of values

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 (Clicker Game with Leaderboard) - 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. **Answer:** Coding is when we give instructions to a computer.

SLIDE 3 AND 4 - WHAT TO EXPECT

Your class will complete three online lessons (using Scratch), three classroom (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 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. Have them 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/818520725

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 unicorn before the timer runs out then enter your name to be added to the leaderboard.

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 8.11 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

Add "**when this sprite clicked**" then "**change Score by 1**".

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

OPTIONAL You can also have the sprite pulse when it's clicked by adding **change size by 10 → Wait 0.05 seconds → Change size by -10.**

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.

MOVING SPRITE - STEP 3 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 3a].

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

SLOWING DOWN - STEP 3 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" [Step 3b]

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 4 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 4a and 4b]

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

ENDING GAME - STEP 5 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)** [Step 5a].

You will also need to add a **show** block at the very start of your program [Step 5b]. 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?

Answer: We don't want our testing to take too long. We can increase the timer once we know that our game works.

CREATE LEADERBOARD - STEP 6 IN QUICK REFERENCE

The final step is to create a leaderboard of sorts to keep track of the results. At the end of the game, we will ask the player to input their name. Then we will add their name and score to two lists to create a rudimentary leaderboard.

Create two lists

Under "Variables" select "Make a List" and create a list called **Name** then a second list called **Score**. Reposition the lists so they pop up in the center of the preview window.

Adding to the lists

At the bottom of your code (under the stop command), add **Ask "Enter Name" and wait** [Step 6a]. Whatever the player enters there will be temporarily stored as a variable called "answer" – we want to add that variable to our list of names.

Put **add (answer) to (Name)** to add the player's name to the Name list then **add (Points) to (Score)** to save their score to the Score list [Step 6b].

Show and hide lists

Finally, you want the lists to show up on screen when the game ends and hide when the next game begins. At the end of your code, put **show list (Name)** and **show list (Score)** [Step 6c]. Then at the start of your code, add **hide list (Name)** and **hide list (Score)** [Step 6d].

NOTE: This isn't a true leaderboard because the results won't be sorted to show the highest score first.

It is quite possible to create a proper leaderboard like this, but it is beyond the scope of this lesson. If you're interested, see the bonus section of the Quick Reference document.

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.

- Pulse animation: Have the sprite grow then shrink when clicked on "**change size by 10 → wait 0.05 seconds → change size by -10**"
- 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
- Clear leaderboard: To reset the leaderboard, go to "Variables" and click delete all of (Name) then delete all of (Score)

SLIDE 6 - RECAP

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

Ask students to recall what they learned.
What are the two ways we can store data in Scratch?

Answer: Variables and lists.

SLIDE 7 - 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 8 - WHAT'S NEXT?

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

Quickly preview what you will be doing next week. You will be making a game that uses variables in a new way.

QUICK REFERENCE

Intro to Scratch

(Clicker game with leaderboard)

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 timed clicker game with a leaderboard.

SET UP

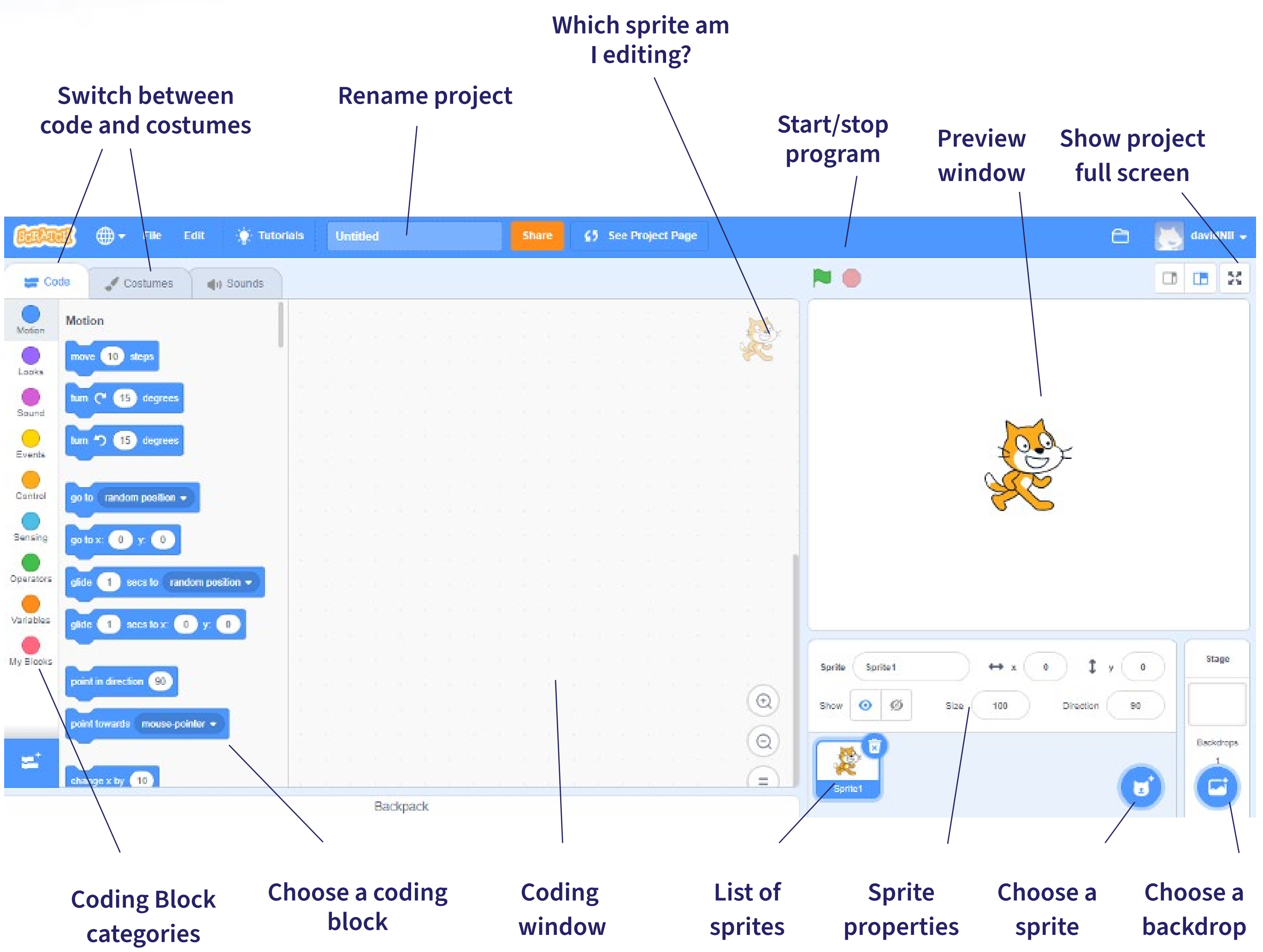
- Have students open Scratch and either create a new account or log into an existing one
- Open Scratch on your own computer and create a blank project. Open the finished version of the game as a demo

QUICK LINKS

Student Activity Link
scratch.mit.edu

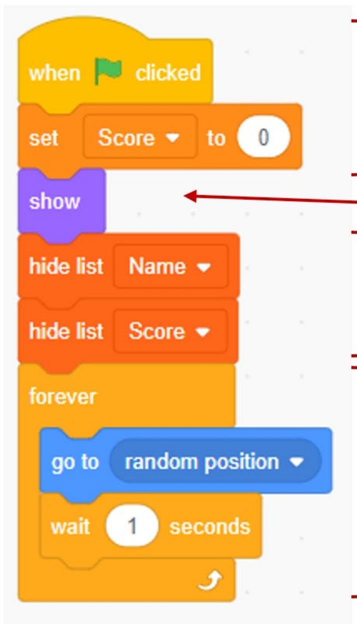
Finished Example
scratch.mit.edu/projects/818520725

SCRATCH TOUR



THE CODE

This document shows the finished code with annotations explaining what each section does and a suggested order. Check out the lesson plan on **Pages 8.6** to **8.9** for a more detailed breakdown of the lesson.



```
when [flag] clicked
set Score to 0
show
hide list Name
hide list Score
forever
    go to random position
    wait 1 seconds
```

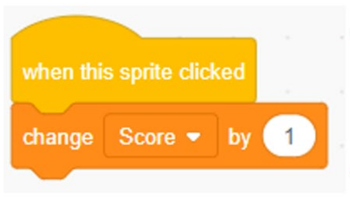
Step 1. Initialization – Score is reset to 0 when a new game starts.

Step 5b. Make the sprite reappear at the start of the next game.

Step 6d. Hides the leaderboard during gameplay.

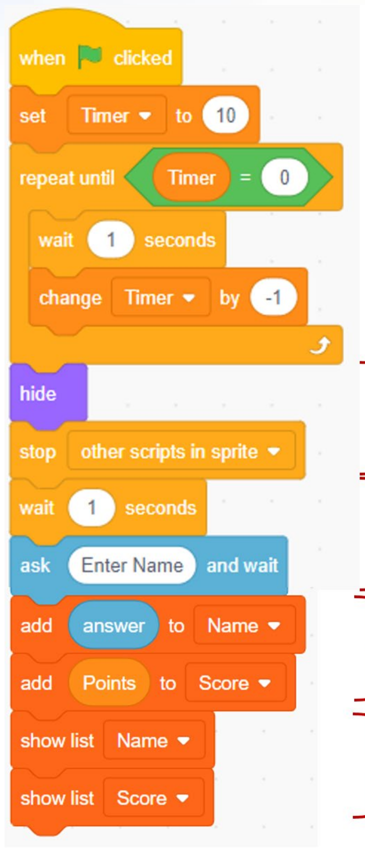
Step 3a. Sprite is forever moving to a random position, waiting 1 second, then moving again. This loop won't end until it receives a "stop" command.

Step 3b. Students can change the wait time to suit their taste.



```
when this sprite clicked
change Score by 1
```

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



```
when ⚑ clicked
set Timer to 10
repeat until <Timer = 0>
    wait 1 seconds
    change Timer by -1
hide
stop other scripts in sprite
wait 1 seconds
ask Enter Name and wait
add answer to Name
add Points to Score
show list Name
show list Score
```

Step 4a. Set the starting time for the timer. Students can change this later, but we use 10 seconds for testing purposes.

Step 4b. Wait 1 second then decrease the timer by 1. Repeat this process until the timer reaches 0.

Step 5a. Hides the sprite and stops its movement after the timer reaches 0.

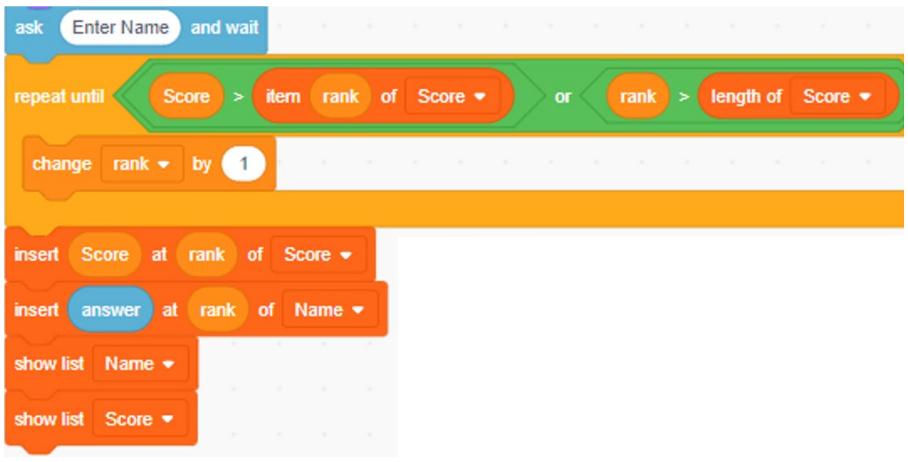
Step 6a. Ask the player to input their name.

Step 6b. Add their answer (i.e., their name) to the Name list and their score to the Score list.

Step 6c. Show both lists. These lists represent the “leaderboard” that will appear at the end of the game.

BONUS – SORTED LEADERBOARD

It probably isn't realistic that you will have time to properly explain and implement this during a 1-hour class, but here is the code for a sorted leaderboard. There are many ways to do this, but this method worked for us.



```
ask (Enter Name) and wait
repeat until <<(Score) > (item (rank) of [Score ▾])> or <(rank) > (length of [Score ▾])>>
    change [rank ▾] by (1)
end
insert (Score) at (rank) of [Score ▾]
insert (answer) at (rank) of [Name ▾]
show list [Name ▾]
show list [Score ▾]
```

How it works:

After the player enters their name, the program needs to sort through the existing list to find the correct place to slot in this new player and their score (often called "insertion sort"). To do that, we will use a temporary variable called "rank" and a repeat until loop. Beginning with the first item in the Score list (i.e., rank #1), the algorithm checks if the new score is bigger than the existing score. If it is, then the loop ends, and the player's name and score are inserted in the #1 position of their respective lists. If the new score is smaller, the process continues by comparing the score to second-place on the list (i.e., rank #2).

We can continue this process by increasing the rank variable by 1 each time. Finally, we also need to stop this process if the rank variable ever exceeds the length of the existing list (i.e., the new score is lower than all previous scores). In that situation, the player and their score are slotted in at the end of the list.

CLASSROOM ACTIVITY 1

Fun with machine learning

45 MINUTES

Students will collect data to train a machine learning model. They will then test the model's performance and make it better.

NOTE: No prior knowledge of machine learning is required.

ACTIVITY LINK

teachablemachine.withgoogle.com/train/image

LEARNING OBJECTIVES

- Analyze data using computers
- Understand the basics of machine learning

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)
- Math D1.2 & D1.6 (Data Literacy)

SET-UP

This activity works best if you have one computer for every 2-3 students, but it can also be done with a single teacher computer. Watch **this video tutorial** before the class. Open the activity link on your computer(s).

INSTRUCTIONS

SUMMARY

Students will be working with Google's Teachable Machine, a machine learning platform. The class will draw cats and trees as training data for the algorithm. They will then train the model, test how well it works, and make improvements.

1. Show students **this introductory video**.
2. Discuss machine learning as a class.

MATERIALS

The class will need:

- At least one computer with Internet connection and a webcam (e.g., the teacher's computer)
- Scrap paper and something to draw with

WHAT IS MACHINE LEARNING?

Machine learning is the process of teaching computers (machines) how to solve specific problems. We start by giving the computer lots of data and a desired task. The computer then uses that data to create a model to perform the task. We can then test the model with new data to see how well it works.

3. Explain today's activity. "We will be training a machine that can tell the difference between drawings of cats and trees. But first, let's see how good you are at this task."
4. Draw a cat on the board. Ask the students "Is this a cat or a tree?" Next, draw a tree on the board. Ask "Is this a cat or a tree?" Ask "How did you know? How could you tell the difference?"

PART 1 – DATA GENERATION AND INPUT

NOTE: These instructions are written as if you only have one computer (i.e., your teacher laptop). If you have more computers, you can divide students into groups such that every group has a computer.

5. The class will now generate training data for the model. Split the class into partners.
6. Within each pair, have one student draw a cat and the other draw a tree. As they are drawing, rename your two data classes as "Tree" and "Cat".
7. As students finish, ask them to bring their drawings to you. Using the webcam and the "Hold to Record" button, take several pictures of each student's drawing and add the images to the appropriate class.

PART 2 – TRAINING THE MODEL

8. Once the data is uploaded, it's time to train the model. Simply click the "Train Model" button and wait until it's ready. It should only take a minute.

   The Teachable Machine is using the images you gave it to create a model (algorithm) that can classify the images (i.e., decide what's a tree and what's a cat).

PART 3 – TESTING THE MODEL

9. When the model is ready, you can test how well it works. Pick a volunteer and have them hold their drawing up to the webcam. The preview window will show how confident it is about its classification. For example, it might be 97% sure the drawing is a cat.
10. The model should do very well at the first test because you are using images that were part of its training data. Next, you will test how well it works on pictures it has never seen before. Have each student draw a new picture. Those who drew trees will now draw a cat and vice versa. Encourage students to draw pictures that look different than the originals (e.g., different type of tree, more detailed cat, use colours).
11. One by one, have students bring their new drawings to the front and test it using the webcam. Make a pile of pictures that the model got right and ones that it has trouble with. Next, we can improve the model's performance by collecting the ones it got wrong and adding them as training data. You will then need to retrain the model.

12. For the final test, ask students to draw a picture that is halfway between a tree and a cat. For example, it could be a cat stuck in a tree or a tree with whiskers. Test the model again using these pictures and discuss the results. Challenge students to create an image that the model perceives as a 50/50 split.
13. Debrief the activity using the discussion questions as a guide.

DISCUSSION QUESTIONS

During the lesson

The machine needed lots of examples to use as training data, but you were able to identify trees and cats right away. Why do you think that is?

Answer: You've seen lots of trees and cats before, but humans are also just much better at this kind of thinking than machines are.

Which drawings did the model struggle to classify when we tested it? Why do think it was hard?

Possible answer: Machines have trouble with images that are too different from their training data. For example, they might recognize pointy trees but not rounded ones.

What happened when we intentionally tried to confuse the model? Why might a programmer do this?

Possible answer: Programmers can improve the model's performance by teaching it how to deal with tricky data.

After the lesson

What advantages do humans have over machines when it comes to classifying data? What are machines better at?

Possible answer: Humans are generally better at pattern recognition and making judgments. Machines can process information much faster. For example, humans might be better at sorting cats vs. trees in terms of accuracy, but a machine can sort through millions of images quickly.

Why do machines need to create their own models? Why can't a coder just program the model themselves?

Answer: There are some tasks that we can't explain to a computer in a language it will understand. We can't really write a program for computers to reliably tell the difference between pictures – the program is simply too complex for us to write.

ONLINE LESSON 2

Make a racing game

60 MINUTES

In this lesson, students will be editing an existing Scratch file to make a racing game. Students will use conditional statements and a "hidden" speed variable to control race car sprites as they drive around a custom track. As possible extensions, students can introduce new obstacles, add power-ups, add a second player, or create new racetracks.

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions
- **Math C3.2** – read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code

GEOMETRIC AND SPATIAL REASONING

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

OPERATIONS

- **Math B2.7** – multiply and divide integers, using appropriate strategies, in various contexts

QUICK LINKS

Student Activity Link
scratch.mit.edu/projects/761542706

Finished Example
scratch.mit.edu/projects/818521530

PowerPoint
Grade 8 – Week 2 – Video Game Variables

Quick Reference
Make a Racing Game – 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 Racing Game - Quick Reference** for your own use during the lesson.

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

SLIDE 1 - WEEK 1 RECAP

Ask students to recall what they learned about in Week 1. What are two ways we can store data in Scratch? (Variables and Lists) What's the difference? (Variables have a single data point, lists can hold many)

SLIDES 2 TO 5 - THE MATH OF VIDEO GAMES

Use slides to discuss video games as an application of coding and data analysis. See the slide notes for talking points.

DEMO

Quickly demonstrate a finished version of the game so students know what they are working towards.

Link to finished game:
scratch.mit.edu/projects/818521530

NOTE: Use the up arrow to drive forward, the down arrow to reverse, and the left/right arrows to turn. You may find it easier to use two hands.

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 and then change the title to something of their choosing. There should be a big, red car in front of a racetrack with three lakes.

Explain that their file is partially complete – the sprites (cars) and backdrops (tracks) were pre-made by NII, but they will have to write the code.

INITIALIZE SPRITE - STEP 1 IN QUICK REFERENCE

Have students set an appropriate size for their race car sprite and drag it to the start line.

After dragging the car behind the start line, students will initialize the sprite by adding **go to x: (value) y: (value) → point in direction 90 → set size to (value)%** underneath the first **when green flag clicked** block [Step 1].

Now the sprite will start in the correct position and direction when the game begins.

ENABLE FORWARD AND REVERSE MOVEMENT - STEP 2 IN QUICK REFERENCE

Now it's time to make the car sprite move. For this game, the sprite will be able to drive forward and reverse and turn to the left and right. How much help you give the students will depend on how quickly they seem to pick up on the patterns.

In the red car's code, you will see a second **when green flag clicked** block with a **forever** loop and four conditional statements inside it. To drive forward, students will need to figure out what to add inside the first conditional statement.

Answer: when green flag clicked → forever → if <key up arrow pressed?> then → move 5 steps [Step 2]. You can think of this like pressing the gas. Have the students test that their car can drive forward.

To reverse, students will need to fill in the second conditional statement (the one for the down arrow) **ASK:** What do we need to change to make our car reverse?

Answer: Change the positive to a negative i.e., move -5 steps [Step 2 continued].

Have students test again.

ENABLE TURNING - STEP 3 IN QUICK REFERENCE

To turn left and right, they will need to fill in the other two conditional statements.

Answer: To turn left, they will add **if <key left arrow pressed?> then → turn ↶ 5 degrees** (counterclockwise) and to turn right they will put **if <key right arrow pressed?> then → turn ↷ 5 degrees** (clockwise) into the third and fourth conditional statements [Step 3].

Students should now be able to practice driving their cars around the track using the four arrow keys.

CREATE A SPEED VARIABLE - STEP 4 IN QUICK REFERENCE

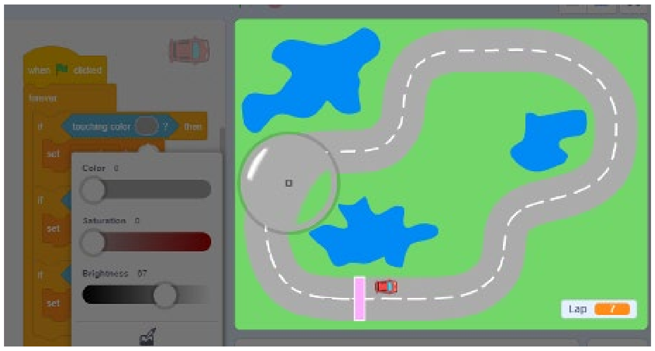
Remind students about variables from last week ("We used a variable to keep track of the score") and from today's intro ("Video games store a lot of data behind-the-scenes that the players never see").

Create a variable called "**speed**" and make it hidden (uncheck the blue box beside the variable name).

We are going to change the variable based on which surface (colour) the car is touching. The car should go fastest on the pavement (grey), slower on the grass (green) and slowest in the water (blue).

To do this, put **if <touching color (grey)? Then → set speed to 5** inside your existing forever loop (before the movement code). To make sure you select the correct shade of grey, click on the colour in the light blue sensing block then use the eyedropper tool to choose the same colour as the road [Step 4].

SET OTHER COLOUR CONDITIONS - STEP 4



CONTINUED

Get the students to repeat the same process with the other two colours – i.e., green and blue. We find that green = 3 and blue = 1 works well, but students can tinker with these values.

ADD THE SPEED VARIABLE INTO THE MOVEMENT CODE

We have now set the speed variable and changed its value based on the colour that the car is touching, but we still need to use that value in our code.

Have students replace the "5"s in all their movement blocks with "speed". For example, they should **move (speed) steps** instead of move 5 steps. Do the same thing for the turning code.

The trickiest part is reversing because we actually want it to be negative speed. Let students try to problem solve, offering some hints as needed.

Answer: move (speed * -1) steps

PLAY TEST

Have students test their game. They should be able to play through the whole game and have the speed update whenever they change surfaces.

Take the time now to help anyone who is having problems.

NOTE: If this is as far as you get, that is quite alright. The rest is fun, but not necessary.

ADD LAP COUNTER - STEP 5 IN QUICK REFERENCE

The game should be fully playable at this point, but there are some more elements we can add to make the game more "game-like". One of those additions is a lap counter. We will be working underneath the third and final (rightmost) **when green flag clicked** block for this last part. Some of the code is pre-populated.

Start by creating a new variable called "**red lap**" (we will specify the colour just in case we want to turn this into a 2-player game later on).

Immediately underneath the "when green flag clicked" add **set red lap to 0** [Step 5a].

Then inside the first if-then statement in the forever loop add **change red lap by 1**. Ask students what colour they should put inside the "**if touching (colour) then**" blocks. **Answer:** the pink colour of the start/finish line [Step 5b].

Use the eyedropper tool to ensure that the pink colour in both conditional statements exactly matches the colour of the start/finish line [Step 5c].

EXPLANATION OF CODE: We want to increase the lap counter whenever we pass the pink start line, but we only want it to increase once which is why we need the "wait until <not> touching color (pink)" part. Otherwise, the lap counter would keep increasing as long as the sprite is hovering over the start line. The wait 3 seconds part is to discourage cheating. In theory, you could just drive forward and reverse across the start/finish line a bunch of times. By forcing the player to wait 3 seconds, it encourages them to actually drive around the track.

PLAY TEST

Test the game again. Does the lap counter go up each time they cross the start/finish line?

The most common mistake is that their chosen pink colour is not an exact match (i.e., they didn't use the eyedropper tool).

SET END CONDITIONS - STEP 6 IN QUICK REFERENCE

We probably don't want our game to go on forever, so we'll need to set an end condition. To do that, we will add an if-statement into our lap counter code.

Find the code that says **if (blank) > 3 then → broadcast (red wins) → stop (all)** [Step 6].

Ask students what they should put into the blank.
Answer: **red lap** variable

Students can set a larger lap number if they'd like, but smaller numbers are better for testing.

The "Winner" sprite is pre-programmed to appear when it receives the "red wins" broadcast message.

OPTIONAL - ADD A SECOND PLAYER

There is a second, yellow car that the students can add if they want to make it a two-player game. Have them copy all their code from the red car over to the yellow car. They can do this by dragging the code over or by adding it to their backpack first.

Then they will need to make the following changes to their code:

1. Change starting positions so the cars start side by side.
2. Change up arrow to "w" key, down arrow to "s" key, right arrow to "d" key, left arrow to "a" key (or some other similar arrangement).
3. Add a "yellow lap" variable. Change "red lap" to "yellow lap" anywhere it appears. Make "yellow lap" visible.
4. Change "broadcast red wins" to "broadcast yellow wins".

PLAY TIME

The game is now fully complete. Give the students time to play and enjoy. They can race against each other.

BONUS - CUSTOMIZE THE GAME

There probably won't be time to explore these in class, but here are a few extra things students could try if they choose to keep working on their games either for fun or for their final projects.

Custom Maps

NII made a couple extra maps that the students can play. They can switch the backdrop to try these other courses. They could also try drawing their own courses! They'll just need to make sure that the colours in their backdrop match the ones in their code.

Obstacles or Power-ups

Students can play around with adding even more colours to their backdrops. For example, they could add a brown splotch (like mud) that has an even slower speed than the water. Alternatively, they could add a speed boost (say orange) on their track that temporarily increases the player's speed.

Even more players

It might get squishy around the keyboard, but in theory, students could add even more players to their game (3- or 4-player mode).

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 were two different ways we used variables in our code?" (Speed was a hidden variable, our laps were a simple counter variable) How could we use lists in our game? (Possible answer: the speeds for each surface type could be stored in a list).

See slide notes for more discussion questions.

SLIDE 8 - POEM OF THE DAY

Share this week's poem as a recap.

SLIDE 9 - WHAT'S NEXT?

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

Try the next classroom activity (Random Rollers) before next week.

QUICK REFERENCE

Make a racing game

Students will be editing an existing Scratch file to make a racing game. We will use conditional statements and a "hidden" speed variable to control race car sprites as they drive around a custom track. As possible extensions, students can introduce new obstacles, add power-ups, add a second player, or create new racetracks.

QUICK LINKS

Student Activity Link
scratch.mit.edu/projects/761542706

Finished Example
scratch.mit.edu/projects/818521530

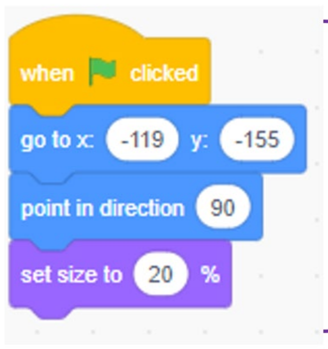
SET UP

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

THE CODE

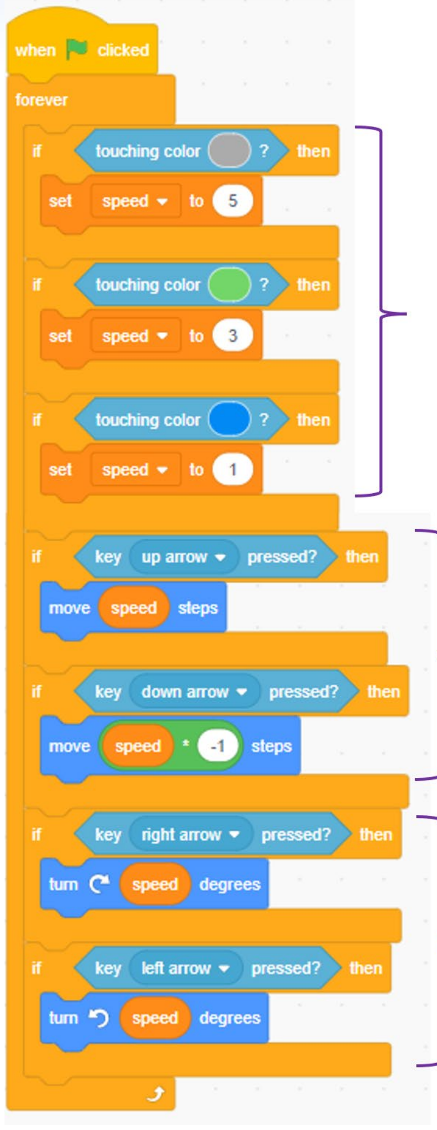
This is the final code with descriptions of what each part does and a suggested order. See the lesson plan on **Pages 8.19** to **8.22** for more detailed instructions. You will not necessarily get to all of this during a one-hour class.

SPRITE # 1 - RED CAR



```
when [flag] clicked
go to x: -119 y: -155
point in direction 90
set size to 20 %
```

Step 1. Initialization
Moves the car to the correct starting position, points it in the right direction, makes it an appropriate size.



```
when 🏁 clicked
forever
    if touching color ● ? then
        set speed ▾ to 5
    if touching color ● ? then
        set speed ▾ to 3
    if touching color ● ? then
        set speed ▾ to 1
    if key up arrow ▾ pressed? then
        move speed steps
    if key down arrow ▾ pressed? then
        move speed * -1 steps
    if key right arrow ▾ pressed? then
        turn ↻ speed degrees
    if key left arrow ▾ pressed? then
        turn ↺ speed degrees
```

Step 4. Set speed conditionally

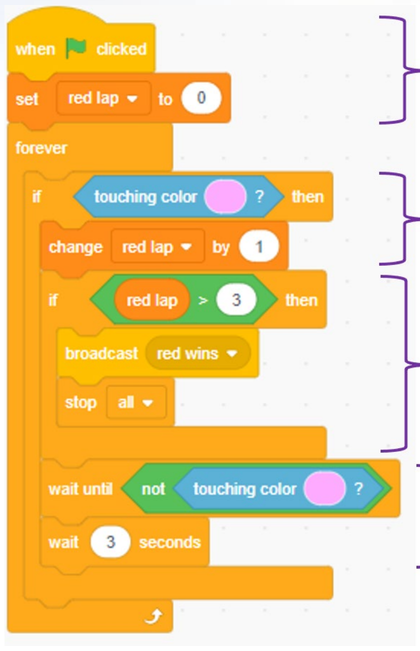
Changes the car's speed based on the surface (colour) that it is touching. Car goes fastest on the grey pavement and slowest through the blue water.

Step 2. Forward/reverse movement

Car moves forward at "speed" when up arrow is pressed and backwards (-speed) when the down arrow is pressed. **NOTE:** The *-1 is needed for the reverse arrow to work.

Step 3. Turning

Enables the right/left arrows to allow the sprite to turn. Note that the turning uses the same speed variable as the forward/reverse movement.



```
when [flag] clicked
set [red lap ▾] to (0)
forever
    if <touching color (pink)?> then
        change [red lap ▾] by (1)
        if <(red lap) > (3)> then
            broadcast (red wins ▾)
            stop [all ▾]
        end
    end
    wait until <not <touching color (pink)?>>
    wait (3) seconds
end
```

Step 5a. Set lap counter to 0 at start of game.

Step 5b. Increase lap counter each time player crosses the pink start/finish line.

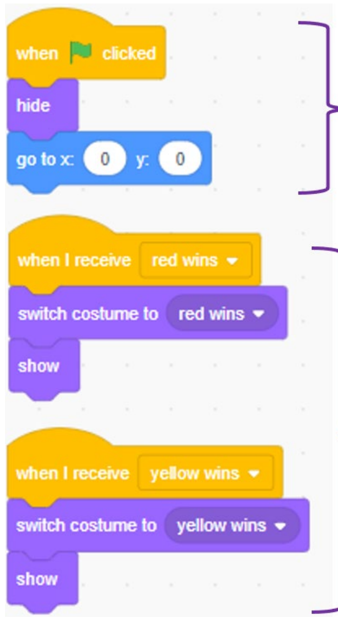
Step 6. Trigger the end game when the target lap number is reached/exceeded. In this case, the game will end after the red car finishes 3 laps.

Step 5c. Prevents the lap counter from increasing multiple times when touched. "Wait 3 seconds" prevents players from simply driving back and forth over start/finish line.

SPRITE # 2 – YELLOW CAR

The code for the yellow car is almost exactly the same as the red car. See the **Instructor Guide** or the finished Scratch file for the few differences.

SPRITE #3 - WINNER MESSAGE



```
when [flag] clicked
hide
go to x: (0) y: (0)

when I receive [red wins ▾]
switch costume to (red wins ▾)
show

when I receive [yellow wins ▾]
switch costume to (yellow wins ▾)
show
```

Initializes sprite position and visibility.

Appropriate winner message appears when the different broadcast messages are received.

CLASSROOM ACTIVITY 2

Random rollers

45 MINUTES

Students will compare theoretical and experimental probabilities, then see how their results stack up against a computer!

MATERIALS

Each pair of students will need:

- 1 worksheet
- 2 six-sided dice
- A calculator

The teacher will need:

- 1 device with Internet access (e.g., school laptop)

LEARNING OBJECTIVES

- Determine and compare the theoretical and experimental probabilities of independent events
- Use code to perform data analysis

CURRICULUM CONNECTIONS

- Math C3.1 & C3.2 (Coding)
- Math D1.2, D2.1 & D2.2 (Data)

SET-UP

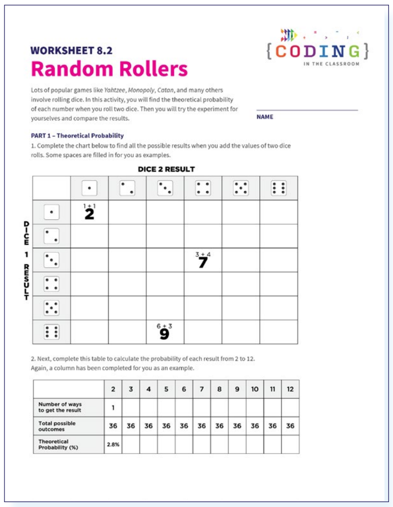
Print and hand out student worksheets. Gather required materials. Open the Scratch file below on your computer.

Dice Simulator: **scratch.mit.edu/projects/738168819**

INSTRUCTIONS

SUMMARY
Students will be comparing the theoretical and experimental results of rolling two dice. At the end, you will simulate thousands of dice rolls on Scratch to demonstrate the power of coding.

1. Ask students to recall what they learned in the online coding classes. **ASK** "What are two ways data can be stored in Scratch?" (Variables and lists) "What advantages do computers have when it comes to data analysis?" (They can store huge amounts of data and perform calculations very quickly)
2. Explain the motivation and premise of the activity. **SAY** "Lots of popular board games involve rolling dice – *Monopoly, Snakes and Ladders, Catan*, etc. Today, we will figure out the theoretical results of rolling two dice. Then we will try the experiment for ourselves and see how it compares."



WORKSHEET 8.2
Random Rollers

Lots of popular games like *Yahtzee*, *Monopoly*, *Catan*, and many others involve rolling dice. In this activity, you will find the theoretical probability of each number when you roll two dice. Then you will try the experiment for yourselves and compare the results.

NAME

PART 1 – Theoretical Probability
1. Complete the chart below to find all the possible results when you add the values of two dice rolls. Some spaces are filled in for you as examples.

DICE 2 RESULT

DICE 1 RESULT	⚀	⚁	⚂	⚃	⚄	⚅
⚀	1 + 1 2					
⚁						
⚂				3 + 4 7		
⚃						
⚄						
⚅			6 + 3 9			

2. Next, complete this table to calculate the probability of each result from 2 to 12. Again, a column has been completed for you as an example.

	2	3	4	5	6	7	8	9	10	11	12
Number of ways to get the result	1										
Total possible outcomes	36	36	36	36	36	36	36	36	36	36	36
Theoretical Probability (%)	2.8%										

***Week 2** Random Rollers Worksheet*

PART 1 – THEORETICAL PROBABILITY

3. Divide students into partners and have them complete Questions 1 and 2 on their worksheet. They will be filling in a grid to calculate the theoretical probability of each result when rolling two dice. Help students read and follow the instructions as needed.

PART 2 – EXPERIMENTAL PROBABILITY

4. Give two dice to each set of partners. Students will roll the dice 50 times and record the results in Question 3 of their worksheet. While they are collecting their data, draw a table like the one below on the board. When the students are done, have one student in each pair calculate the percent probability for each outcome and record it on their worksheet. Have the other student add their results in the first two rows of your table on the front board. **See Figure 1.**
5. Have students answer Question 4 on their worksheet and discuss their answers as a class.
6. You will now combine all the results from the class into a single data set. Before you begin, ask the students to predict: *If we combine all our data, will our results be closer or farther from the theoretical probabilities?*
7. As a class, add up the results in each section of your big chart. Divide by the total number of rolls to get the experimental probability for each outcome. Students will record these answers in Question 5 of their worksheet and compare these probabilities to their earlier results in Question 6.

PART 3 – COMPUTER SIMULATION

8. Open the Dice Simulator in Scratch and project it for the students to see. Click “Roll” a few times to show the students how it works. If time allows, you can also take a closer look at the code inside. After doing a few rolls, click “Reset” to clear the counters.
9. Point out the “x1000” button. **ASK** *How long do you think it will take the computer to simulate 1000 rolls? How long would it take you?*

   Click “x1000” and observe the results. Click it until you reach 10,000 total rolls (or another similarly large number).
10. Have students record the simulator’s results in Question 7 of their worksheet and calculate the probability of each outcome.

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls											
Experimental probability (%)											

Figure 1

(OPTIONAL) PART 4 – PLAY A DICE GAME

11. If time allows or on another day, let students play a game that involves dice rolling. They will use what they know about the theoretical and experimental outcomes to inform their strategy.

    Here is a simple 2-player game:

    Player 1 rolls two dice and says the sum. **Player 2** then predicts if their result will be higher, lower or the same as Player 1's before rolling the dice. If Player 2 is correct, they score a point. If Player 2 is incorrect, Player 1 gets a point. Players alternate turns until someone reaches 10 points.

12. Have students complete all the questions on their worksheet. You may also choose to debrief the activity as a class using the discussion questions below as a guide.

DISCUSSION QUESTIONS

How did increasing the number of trials affect the experimental probability?

Answer: As the sample size increases, the experimental probability gets closer to the theoretical probability. This is known as the law of large numbers.

What advantage does the computer have over humans? What advantage do we have?

Possible answer: The computer is much better at computing experimental probability because they can perform experiments with a large number of trials. Humans, however, are better at calculating theoretical probabilities. Sometimes math problems require creative thinking, a skill that is distinctly human.

Can you think of another situation that could be simulated on a computer?

Examples: flipping a coin, rock-paper-scissors games, lottery draws, entire board games (e.g., chess, checkers, Monopoly)

What was your strategy in the dice game? How could you apply what you learned in today's activity when playing board games?

Answers will vary.

ACTIVITY 2 WORKSHEET

Random rollers

Lots of popular games like *Yahtzee*, *Monopoly*, *Catan*, and many others involve rolling dice. In this activity, you will find the theoretical probability of each number when you roll two dice. Then you will try the experiment for yourselves and compare the results.

____
NAME

PART 1 – Theoretical probability

1. Complete the chart below to find all the possible results when you add the values of two dice rolls. Some spaces are filled in for you as examples.

DICE 2 RESULT

DICE 1 RESULT

	⚀	⚁	⚂	⚃	⚄	⚅
⚀	1 + 1 **2**					
⚁						
⚂				3 + 4 **7**		
⚃						
⚄						
⚅			6 + 3 **9**			

2. Next, complete this table to calculate the probability of each result from 2 to 12. Again, a column has been completed for you as an example.

	2	3	4	5	6	7	8	9	10	11	12
Number of ways to get the result	1										
Total possible outcomes	36	36	36	36	36	36	36	36	36	36	36
Theoretical probability (%)	2.8%										

EXPLORE

PART 2 – Experimental probability

3. Roll two dice, calculate their sum, then put a tally mark in the appropriate box in the chart below. Repeat for 50 rolls then calculate your experimental probability of each result.

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls	50	50	50	50	50	50	50	50	50	50	50
Experimental probability (%)											

4. How did your results compare to the theoretical probabilities? Were any numbers more common than you expected? Less common?

5. Record the total results for your entire class in the chart below.

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls											
Experimental probability (%)											

6. Whose results were closer to the theoretical probabilities – yours or your entire class?

PART 3 – Computer simulation

7. Record the results of the computer simulation in the chart below. How do these results compare to what you've found so far?

	2	3	4	5	6	7	8	9	10	11	12
Numbers of times the result happened											
Total rolls											
Experimental probability (%)											

ONLINE LESSON 3

Two-player Pong

60 MINUTES

In this lesson, students will make their own version of the arcade classic *Pong*. The lesson will draw on the students' prior coding knowledge, including conditional statements, loops, variables, and subprograms. Students will also practice applying transformations (translations and reflections) on a Cartesian plane.

CURRICULUM CONNECTIONS

CODING

- **Math C3.1** – solve problems and create computational representations of mathematical situations by writing and executing code, including code that involves the analysis of data in order to inform and communicate decisions
- **Math C3.2** – read and alter existing code involving the analysis of data in order to inform and communicate decisions, and describe how changes to the code affect the outcomes and the efficiency of the code

OPERATIONS

- **Math B2.7** – multiply and divide integers, using appropriate strategies, in various contexts

DATA LITERACY

- **Math D1.1** – identify situations involving one-variable data and situations involving two-variable data, and explain when each type of data is needed

GEOMETRIC AND SPATIAL REASONING

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

QUICK LINKS

Student Activity Link
https://scratch.mit.edu/projects/762906582

Finished Example
https://scratch.mit.edu/projects/818525405

PowerPoint
Grade 8 – Week 3 – Video Game History

Quick Reference
Two-Player Pong – Quick Reference

LESSON BREAKDOWN

SLIDE 1 - WELCOME

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

Ask students to recall what they learned in the previous two Scratch lessons.

SLIDES 2 TO 7 - HISTORY MINUTE

Use slides to discuss the history of computers and video games. See the slide notes for talking points.

DEMO

Quickly demonstrate a finished version of the game so students know what they are working towards.

Link to finished game:
scratch.mit.edu/projects/818525405

NOTE: Left (green) paddle is controlled by the W and S keys; right (red) paddle uses up and down arrows.

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 and then change the title.

Explain that their file is partially complete. They will have to finish the code to make a working game.

MAKE THE PADDLES MOVE - STEP 1 IN QUICK REFERENCE

The code is mostly pre-written for the paddle sprites. Students just need to fill in the conditional statements for both sprites. Give them time to try this on their own before sharing the answer.

Answers:

Red Paddle

if key (up arrow) pressed? then **change y by 10**

if key (down arrow) pressed? then **change y by -10**

Green Paddle

if key (w) pressed? then **change y by 10**

if key (s) pressed? then **change y by -10**

MAKE THE BALL MOVE AND BOUNCE - STEP 2 IN QUICK REFERENCE

Now that the paddles can move, it’s time to make the ball move. The ball’s size and starting position are pre-coded. The “wait 1 second” block gives the players time to get ready before the game starts.

Start movement

To get the sprite to move, add **forever → move 10 steps → if on edge, bounce** after the “wait 1 second” block [Step 2a].

To test, click the green flag. The ball should move back and forth horizontally across the screen, bouncing when it hits the edge.

Random direction

We don't want our sprite to always bounce at the same angle, so let's make it start at some random angle. Add **point in direction (pick random 50 to 80)** before the forever loop [Step 2b]. Have the players click the green flag and see what happens. The ball should bounce around the screen beginning at a random angle.

MAKE "BOUNCE" SUBPROGRAM - STEP 3 IN QUICK REFERENCE

We want the ball to bounce off the paddles the same way it bounces off the walls. To do this, we are going to use conditional statements and a subprogram. Let's start with the subprogram.

Have students click the green flag and watch the ball bounce. As it bounces, have them pay attention to the sprite's direction value in the sprite properties box (near bottom right of screen). **ASK:** What happens to the sprite's direction when it hits one of the side walls?

Answer: It changes its sign. For example, 75 changes to -75 and vice versa.

With that knowledge in mind, we will program a direction for our "bounce" block. The bounce block is pre-made, but we still need to define it. Under **define bounce** (you may have to scroll right to find it), students will see a **point in direction (0)**. Ask students – what should go in the blank? Give them time to think about it and put their math knowledge to the test. **HINT:** What can we multiply our direction by to change its sign? (-1)

Answer: point in direction (-1 * direction) [Step 3]

The "move 10 steps" and "wait 0.5 seconds" are pre-written and make it so the ball isn't constantly touching the paddle sprite.

PADDLES HIT BALL - STEP 4 IN QUICK REFERENCE

We've made our "bounce" subprogram – now we will tell our ball when to execute that command.

Drag out a new **when green flag clicked** block and add a **forever** loop below it. Inside that loop, add two **if-then statements.** Ask students if they can figure out what to put inside the two conditionals.

Answers: if <touching Red Paddle> then → bounce and **if <touching Green Paddle> then → bounce** [Step 4]

PLAY TEST

Have students test their games. At this point, the ball should bounce back and forth, and they should be able to hit it with their paddles.

Take the time now to help anyone who is having problems.

ADD SCORE VARIABLES - STEP 5A IN QUICK REFERENCE

Let's start keeping score. First, we will need to create two score variables. Go to Variables then "Make a Variable" called **green** or **green score** and then do the same for red. Drag the variable readouts to an appropriate position on the game screen (e.g., top corners).

Initialize the scores by adding **set green score to 0** and **set red score to 0** at the start of the program [Step 5a].

SCORE POINTS - STEP 5B IN QUICK REFERENCE

You may notice that there is a green line across one end of the screen and a red line at the other. These lines represent each player's "goal". Red should score a point if the ball hits green's goal and vice versa.

Add two more conditional statements under the ones you already have (the bounce ones). You could have students figure out what to put or you can do it together.

If (touching Red Goal) then → change green score by 1 and **if (touching Green Goal) then → change red score by 1** [Step 5b]

"RESET BALL" SUBPROGRAM - STEP 6 IN QUICK REFERENCE

After scoring a point, the ball should reset by going back to the middle of the screen. To do this, let's write a "reset ball" subprogram.

Go to My Blocks and "Make a Block" called **reset ball →** Under **define reset ball**, add **hide → wait 1 second, go to x: 0 y: 0 → point in direction (pick random -50 to -80) → show** [Step 6a].

Add the reset ball block into the appropriate conditional statements underneath "change green score by 1" and "change red score by 1" [Step 6b].

Now the ball will disappear and wait a second before restarting back at the middle of the screen.

PLAY TESTING

The game should now be fully playable. Give students a chance to play against each other and test each other's games. If the game is too easy, they can increase the ball/paddle speed or make the paddles smaller. If the game is too hard, they can do the opposite of those things.

OPTIONAL - ADD A TARGET SCORE

If there's lots of time left and students are feeling particularly ambitious, they could add in a target score. In the original *Pong*, games were first to 11.

Immediately after "change green score by 1" add **if green score = 11 then → broadcast (green wins).** After "change red score by 1" add **if red score = 11 → broadcast (red wins)**.

NII Explore has pre-programmed a victory sprite to appear when it receives either of those broadcast messages.

BONUS - CUSTOMIZE THE GAME

There probably won't be time for these during a 1-hour class, but here are a few extra things students could try. These could be good ideas for their final projects.

Change speed during game

The ball could get faster and faster as the game goes on. Otherwise, two good players might end up playing forever.

Experiment with bounce angles

In the original *Pong*, the ball bounces at a different angle depending on which part of the paddle it hits. Students could experiment with different bounce angles to give their games an added layer of fun/complexity.

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: variables, lists, types of variables, conditional statements, loops. See slide notes for suggested talking points.

SLIDE 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 (click on the file folder in the top right corner).

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 classroom activities with your class. Happy coding!

{CODING}
IN THE CLASSROOM

QUICK REFERENCE

Two-player Pong

In this lesson, students will make their own version of the arcade classic *Pong*. The lesson will draw on the students' prior coding knowledge, including conditional statements, loops, variables, and subprograms. Students will also practice applying transformations (translations and reflections) on a Cartesian plane.

QUICK LINKS

Student Activity Link
https://scratch.mit.edu/projects/762906582

Finished Example
https://scratch.mit.edu/projects/818525405

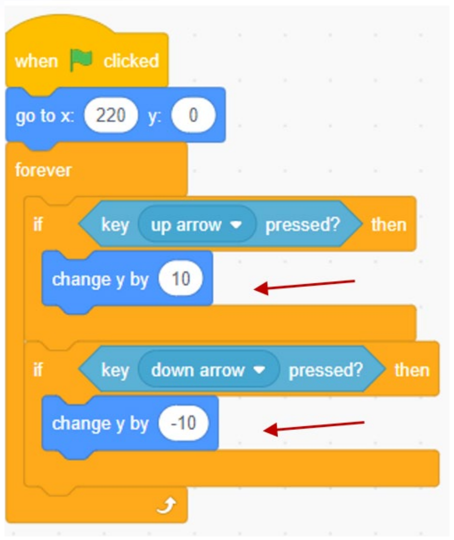
SET UP

- 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

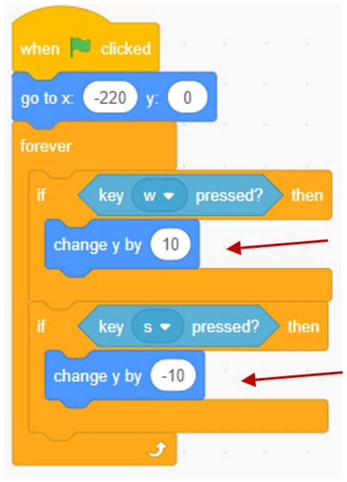
Here is the final code for all sprites. Parts annotated in **GREY** will be pre-populated for the students. Sections marked in **PINK** indicate code that you will be adding, and the numbers correspond with the suggested order of steps from the lesson plan on **Pages 8.32** to **8.35**.

RED PADDLE



```
when [flag] clicked
go to x: 220 y: 0
forever
    if key up arrow pressed? then
        change y by 10
    if key down arrow pressed? then
        change y by -10
```

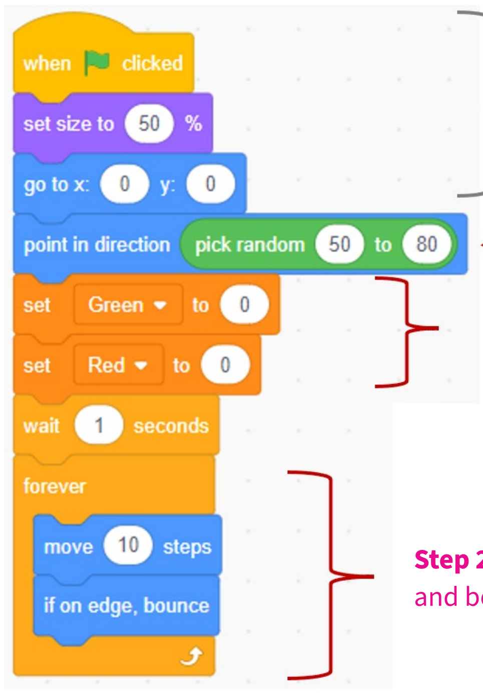
GREEN PADDLE



```
when [flag] clicked
go to x: -220 y: 0
forever
    if key w pressed? then
        change y by 10
    if key s pressed? then
        change y by -10
```

Step 1. Add the appropriate “change y by ___” blocks to complete this movement code.

BALL SPRITE



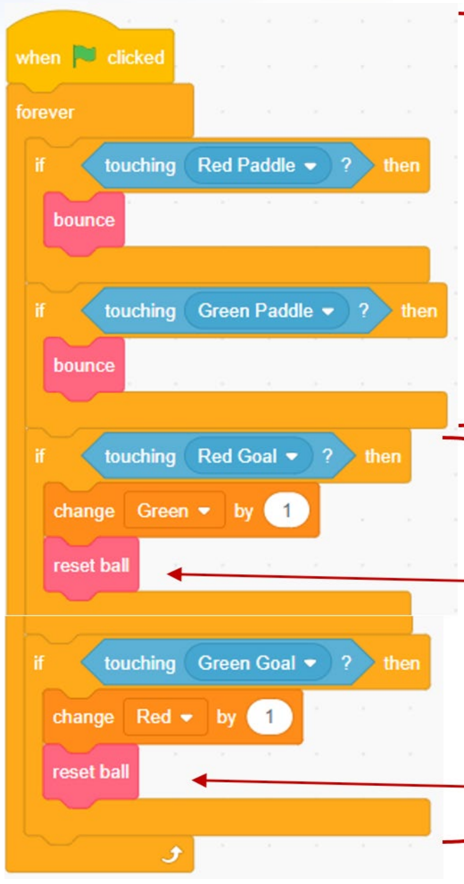
```
when [flag] clicked
set size to 50 %
go to x: 0 y: 0
point in direction pick random 50 to 80
set Green to 0
set Red to 0
wait 1 seconds
forever
    move 10 steps
    if on edge, bounce
```

The ball sprite’s size and starting position are already initialized.

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

Step 5a. Create score variables then initialize scores.

Step 2a. Make the sprite move forever and bounce when it hits the edge.

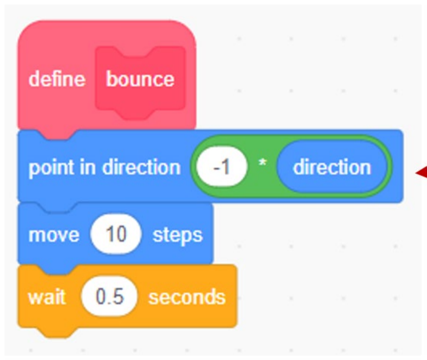


```
when 🏁 clicked
forever
    if touching Red Paddle ? then
        bounce
    if touching Green Paddle ? then
        bounce
    if touching Red Goal ? then
        change Green by 1
        reset ball
    if touching Green Goal ? then
        change Red by 1
        reset ball
```

Step 4. Ball should bounce when it hits either of the two paddles.

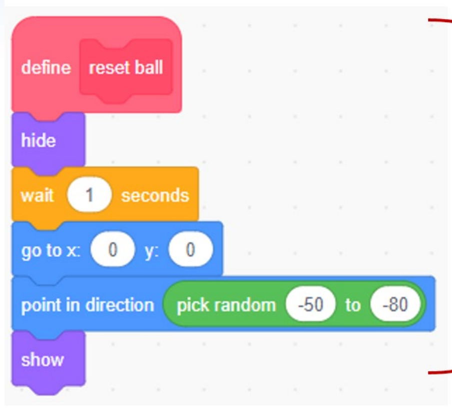
Step 5b. Change the appropriate score whenever the ball touches one of the goals.

Step 6b. Reset the ball back to the middle after a point is scored (i.e., after one of the goals is touched).



```
define bounce
point in direction (-1 * direction)
move 10 steps
wait 0.5 seconds
```

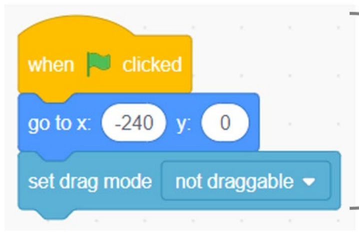
Step 3. Most of this subprogram is pre-written but the students need to figure out what goes inside this “point in direction” block.



```
define reset ball
hide
wait (1) seconds
go to x: (0) y: (0)
point in direction (pick random (-50) to (-80))
show
```

Step 6a. Create this My Block (subprogram) to send the ball back to the middle after each goal.

GREEN AND RED GOALS

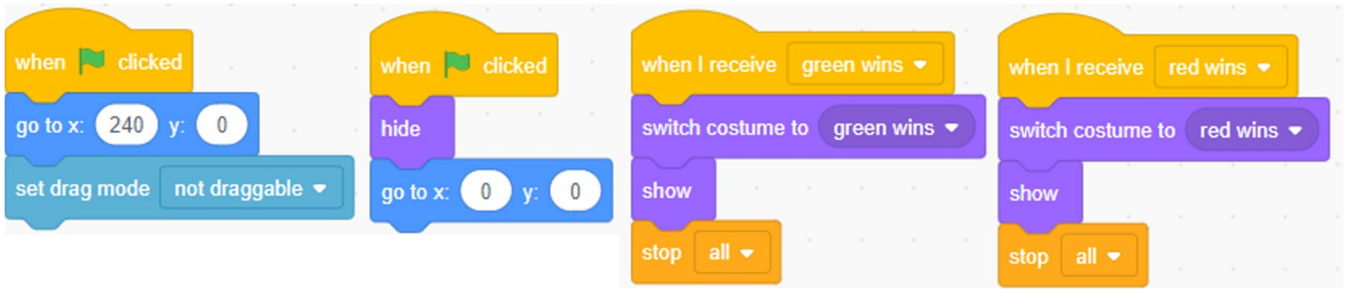


```
when green flag clicked
go to x: (-240) y: (0)
set drag mode [not draggable]
```

Initializes position and keeps it from moving by accident.

WINNER MESSAGE SPRITE

OPTIONAL This sprite remains hidden during the game. It is only programmed to appear when it receives one of the winner broadcast messages. It will end the game once it appears (**stop all** command).



```
when green flag clicked
go to x: (240) y: (0)
set drag mode [not draggable]
```

```
when green flag clicked
hide
go to x: (0) y: (0)
```

```
when I receive [green wins]
switch costume to [green wins]
show
stop [all]
```

```
when I receive [red wins]
switch costume to [red wins]
show
stop [all]
```

CLASSROOM 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. They will use the front for this activity and the back for their final projects.

INSTRUCTIONS

SUMMARY

Video game developers often complete 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 create 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, Racing 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 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 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 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 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, Racing 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
- ☐ At least 2 variables (e.g., score, timer, speed, etc.)
- ☐ A conditional statement (*if-then* or *if-then-else)*
- ☐ A subprogram (or "My Block")
- ☐ An ending (e.g., "You Win" or "Game Over" message)

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. Which variables did you use in your program? What were they used for?

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 send 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 **Page 8.47** to evaluate your students' work.
8. If you haven't already, try the classroom 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?

- ☐ **At least 2 variables** (e.g., score, timer, speed, etc.)
  Click on the "Variables" tab on the left. Do they have at least two orange variables? What are they?

- ☐ **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?

- ☐ **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 variables and data structures (e.g., lists) when appropriate Rarely looks for ways to make code more efficient	Sometimes uses variables and data structures (e.g., lists) when appropriate Sometimes looks for ways to make code more efficient	Often uses variables and data structures (e.g., lists) when appropriate Often looks for ways to make code more efficient	Always uses variables and data structures (e.g., lists) when appropriate Always looks for ways to make code more efficient

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 classroom coding activities Rarely makes connections between coding concepts and everyday life	Participates somewhat in classroom coding activities Sometimes makes connections between coding concepts and everyday life	Participates in classroom coding activities Makes connections between coding concepts and everyday life	Participates fully in classroom 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

CANADA LEARNING CODE

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

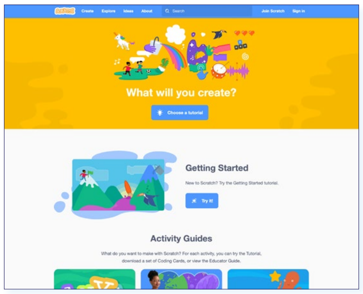
canadalearningcode.ca

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 8 students.

blockly.games

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


What will you create?

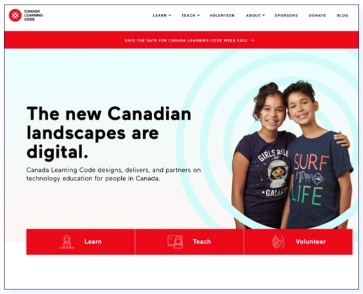
Getting Started

New to Scratch? Try the Getting Started tutorial.

Activity Guides

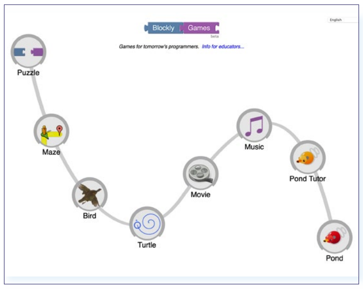
What do you want to make with Scratch? For each activity, you can try the Tutorial, download a set of Coding Cards, or view the Educator Guide.

Scratch


The new Canadian landscapes are digital.

Canada Learning Code designs, delivers, and partners on technology education for people in Canada.

Canada Learning Code


Games for tomorrow's programmers. Info for educators...

Blockly Games