2015 HOPSCOTCH HOUR OF CODE

Teacher notes for student-led tutorial

TIME

45-60 minutes (+15 minutes of optional, free code time)

BIG IDEA

Computers can only do what you SAY because they are not smart enough to figure out what you MEAN. Be specific!

SKILL FOCUS

– Debugging
– Make sense of problems and persevere in solving (CCSS.MATH.PRACTICE.MP1)
– Look for and make use of structure (CCSS.MATH.PRACTICE.MP7)
– Designing solutions (NGSS Practice 6)

KEY VOCABULARY

Sequence: The order in which instructions are given to the computer
Event: When something happens
Rule: Instructions that tell your computer what to do (the command) and when to do it (the event)
Loop: Code that repeats
Random: a surprise
Range: the highest and lowest number for *random* to choose between

TRANSFER GOALS

1. Students will understand that coding requires giving a computer explicit directions
2. Students will become familiar with creating and editing rules
3. Students will practice testing their programs to find bugs.
4. Students will abstract a problem to design a solution.
5. Students will develop confidence and persistence.

MATERIALS

– 1 iPad per student, or 1 iPad per 2 students, for pair programming
– Video tutorial available in Hopscotch and on YouTube: http://hop.sc/HOC_Video
– Complete project available: http://hop.sc/HOC_project

Hi!

We're *really* excited that, this Hour of Code, you're programming with your students—both for them and for you. Kids have remarkable imaginations, and creating computer programs is an amazing way for them to express themselves. We've seen kids create astonishing things using our simple but powerful tool. We know you'll see the same when using Hopscotch, and hope you share what your students create.

Anyone, regardless of their experience in programming, can teach this Hour of Code lesson. Just as Hopscotch was built on the principle that anyone can become a great programmer, this lesson is designed on the premise that anyone can teach basic programming, including you!

In this lesson, students will build a game in which their character jumps over fast-moving obstacles. It's simple but fun, and very quickly allows them to experience the satisfaction of telling their computer what to do.

You can teach this Hour of Code in several ways:

1. Students independently complete their games, following along with the video tutorial in the app. The tutorial is designed to be used without any outside help (though we encourage kids to pause it as they're coding). The tutorial is available at http://hop.sc/HOC_Video.
2. Students indepently complete their games but you ask them to pause their own work for discussion and group work. We offer discussion ideas, as well as differentiation techniques and reflection questions, in the following pages.
3. You project the video to the class and use it as a supplement to your instruction. You lead discussion and group work, and adjust the directions for the project based on the following.

After a discussion of what needs to be built and, if desired, how it might be coded, students can start coding. Depending on how many iPads you have, you can have students work independently or in pairs. At Hopscotch, we do a lot of pair programming (two programmers share one computer) because it helps us write smarter, less-buggy code. We recommend trying it! All students should get into the habit of testing their code frequently by running (playing) it. It is much easier to find and solve mistakes when you're constantly testing.

Have fun and we can't wait to see what your students build. Share their projects on social media and tag us either with #madeonhopscotch or @hopscotch on Twitter and @gethopscotch on Instagram :)

Yours,
Jocelyn Leavitt
Co-founder and CEO, Hopscotch

ACTIVITY GUIDE

0. Introductory Discussion (5 minutes)
The first and most important lesson of computer science is that computers do what they are told, and only what they are told, in the order they are told to do it.

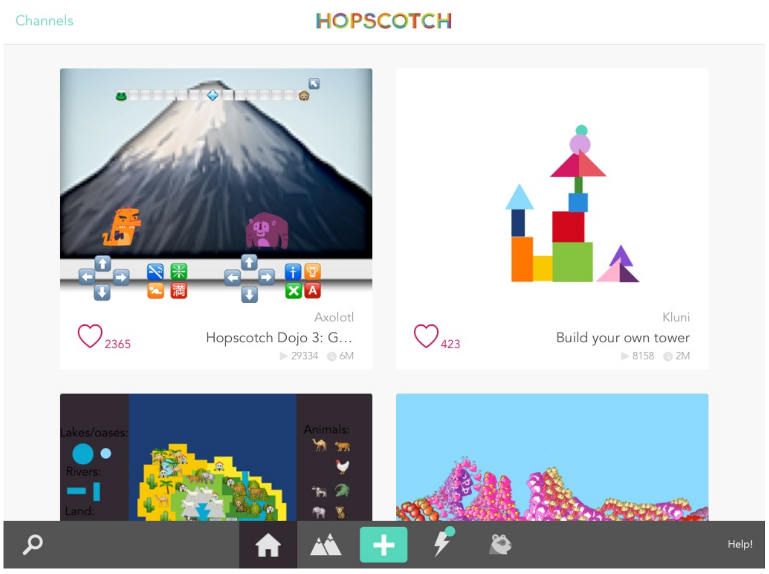
If you fully understand this concept and begin to think of everyday processes (making a sandwich, getting to school) as a set of instructions, you will begin to think like a programmer without trying very hard! A programmer is a person who codes, or writes computer programs. A program is a set of instructions a computer can understand. We refer to these instructions as a **sequence**. This term also refers to the idea that computers must follow the instructions in the order, or in the "sequence" in which they're given.

Ask your students to name some programs they use. Consider all their games and apps, but also the software a DJ uses to mix tracks, the database your doctor uses to keep track of your health, and the video games you play after school. All are programs and all were created by programmers.

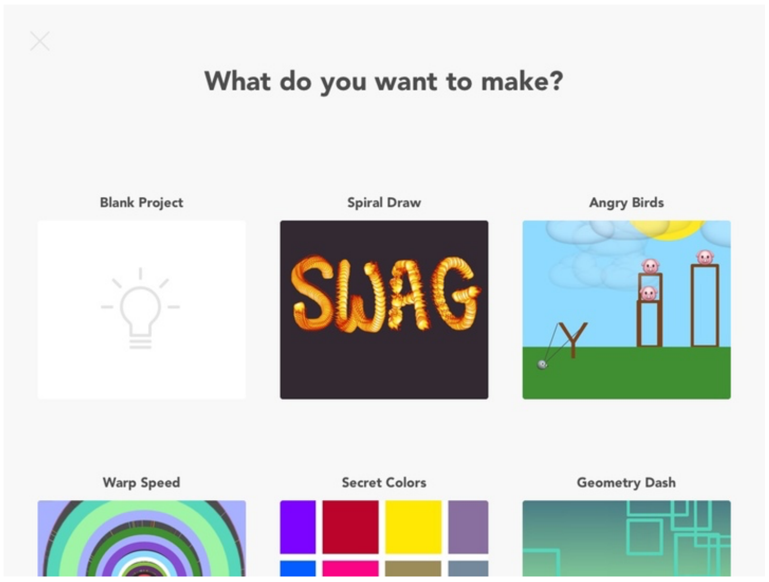
How many times a day do you interact with computers? Are there computers in surprising places? How about a car? How about a phone? If you can control these computers and write programs for them, you can make things that millions of people use every day!

1. Using Hopscotch (5 minutes)
Download and get your students acquainted with Hopscotch. (http://hop.sc/gethopscotch)

> **1.1 Finding the Hopscotch app on your iPad**
> **1.2 Signing into your account (students may need to create accounts)**
> **1.3 Making a new project: Tap on the Highlighted + on the bottom of the screen**



1.4 Choose Hour of Code Jumper Game



2. Getting started (10 minutes)

In this game, the player controls a little square that jumps over obstacles. It's a quick and fun way to experience the power of programming, and leaves lots of room for customization.

One of the most important lessons of this activity is learning that the programmer must not only put together all the components of the game (characters, background, etc.), but also explicitly tell the computer how they should work. For this, we need to create a rule, or code that tells the computer what to do and when to do it. **A rule** has two components: an event and commands (or action).

An event is a trigger that the computer recognizes and causes it to do some action. In Hopscotch, all events start with the word "When" and are the first thing you choose when you write a rule. Think of it as completing a "WHEN….., THEN….." sentence.

Events are deeply important for computer engineers because they tell the computer when it should do something. When you touch the phone icon on your home screen, then your phone brings up the interface to make calls. When an Angry Bird hits a block, then the block falls down.

Discuss some events (triggers) that happen in the classroom. Identify the trigger and resulting action: When I raise my hand (trigger), then stop talking (action), when the bell rings (trigger), then put down your pencil and turn in your test (action).

You can demonstrate some of these concepts by playing your blank project (wihout any code). Nothing happens. This is because we've yet to build our game! Until we tell the computer how the game works, we don't have a game. After a general discussion of rules and events, you can transition to talking about programming your game.

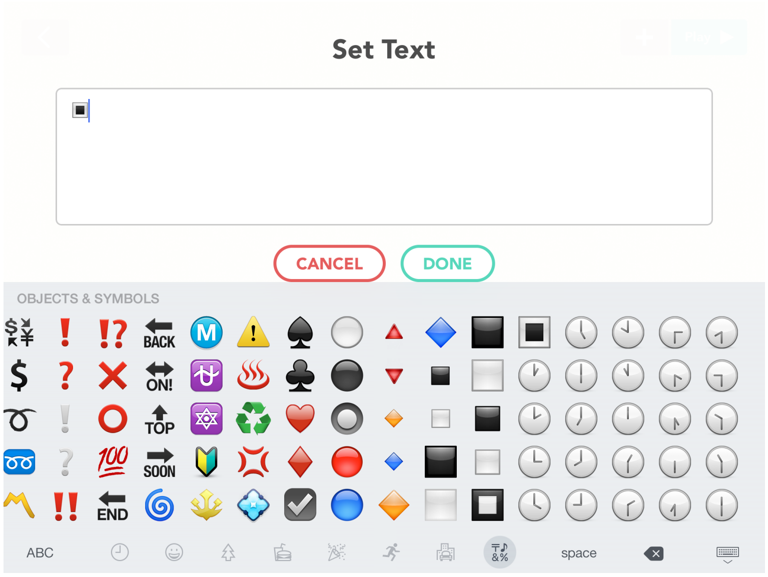
ACTIVITY GUIDE

We will need to program a hero (a square emoji) to jump up over moving obstacles when the iPad is tapped.

As a class, break down the components of the first rule: when the iPad is tapped, the square should jump. Get students to deconstruct the two steps of jumping (move up, then move down). Does this up and down movement occur along the X or Y axis? As a class, determine the code that will create this movement.

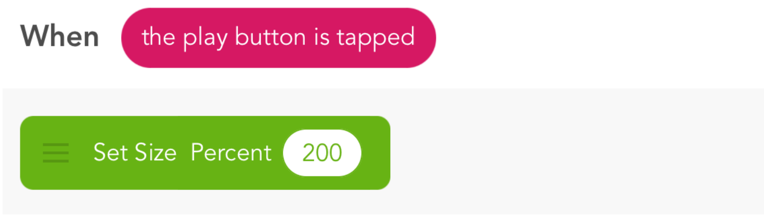
Then, ask your students to add their hero object (the square emoji) and tell it to jump when they tap their iPad. You should encourage students to test (play) their code frequently as they work. By playing it, they can see if it creates the desired effect and correct any mistakes they may have made. Once they have a working jumper, have them play it for a minute. Cool!

2.1 Add hero object (square emoji) and place it near the bottom left corner of screen



Tap the grey "+" button in the upper right corner of the screen to get your object. Choose a text object. When the keyboard opens, make sure the emoji keyboard is enabled, which you can do in your iPad's settings.

2.2 Add rule to hero to make it bigger

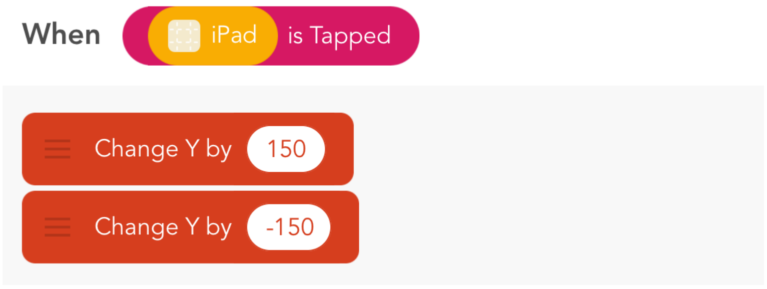


When the play button is tapped

Set Size Percent 200

If you choose a number other than 200, all of the other numbers we give will also have to change. This is an opportunity for debugging.

2.3 Add rule to hero to jump



When iPad is Tapped

Change Y by 150

Change Y by -150

3. Obstacles [10 minutes]
In games like Flappy Bird and Geometry Dash, it feels like the hero is moving forward through a stationary world but actually, the hero is stationary and the world is moving backward. Have you ever been sitting in a stationary car and another car next to you backs up – doesn't it feel, for just a moment, like you're moving forward? In this game, the hero is the car you're in, and the obstacles are the things moving backwards.

Take some time to talk about the movement of an obstacle from the right edge of the screen across to the left edge. See if you can come up with the sequence of obstacle's movement rule as a class. Hint: the iPad screen is 1024 pixels wide. So, to go from the right side to the left, the obstacle needs to move -1024 pixels (from x=1024 to x-0). What should trigger this rule?

What if we want to make it look like there are many obstacles but only use one object? This is another great design trick. See if your students can identify the technique to make this possible – putting the code inside a loop.

This is a good time to discuss **sequence** and **loops.**

Sequence is the order in which instructions are given to the computer. The idea of putting instructions in the correct sequence seems obvious and basic, but it's a vital concept in computer programming.

You can reference a real-life example: making sandwiches for their friends. Ask the class what process they would need to employ in order to make and wrap 10 tuna sandwiches. Does it matter if the process happens in same order for each sandwich? What if they added mayo after putting canned tuna on bread? Or what if you put the bread in the bag before opening the tuna? Silly, but order matters.

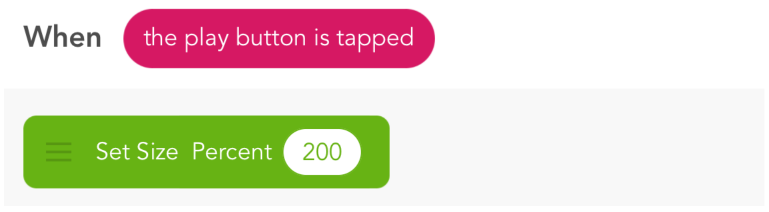
Computers have a finite set of kinds of tasks they can accomplish. But when these tasks are combined properly, amazing things can be built. In addition to running instructions sequentially, computers are very good at repeating sets of instructions. In computer science we call this a "**loop**", or code that repeats.

Consider using a loop to repeat the sandwich making process: For the number of sandwiches I need: open the tuna, add mayo, stir, put on bread, put in bag.

As a class, discuss the behavior of the obstacle and together make a list of the steps it takes. Ask students to consider the difference between using "Repeat 10 times" and "Repeat Forever". Which is appropriate for the sandwich? Which is appropriate for the obstacle's movement? Also, consider what happens if instructions are out of order.

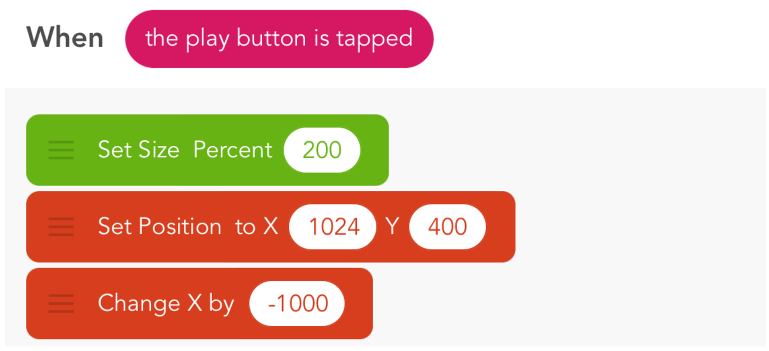
3.1 Add emoji object for obstacle (triangle)

3.2 Add rule to obstacle to make it bigger



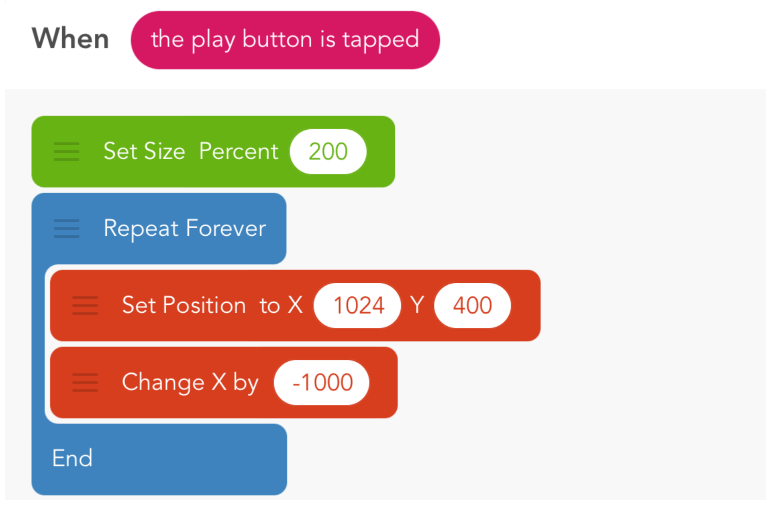
```
When the play button is tapped
    Set Size Percent 200
```

3.3 Edit obstacle's rule to move it from a fixed starting place on the right side of the screen across to the left



```
When the play button is tapped
    Set Size Percent 200
    Set Position to X 1024 Y 400
    Change X by -1000
```

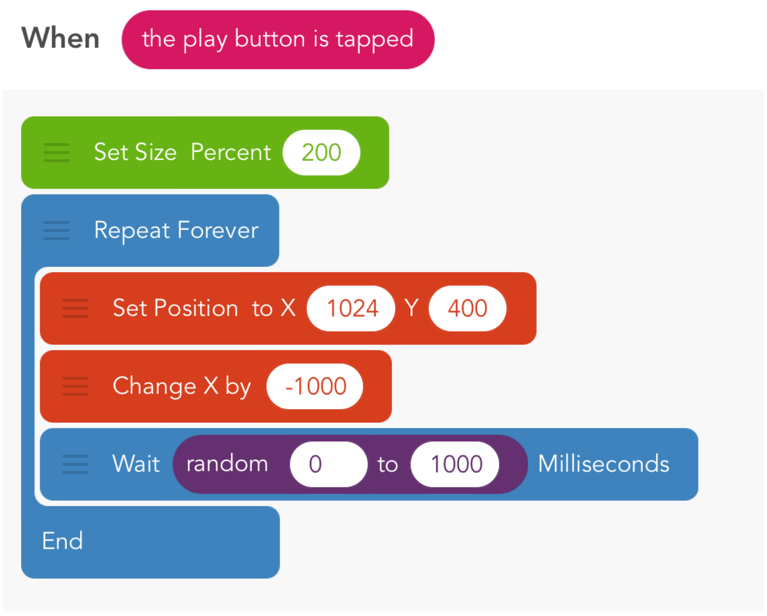
3.4 Edit obstacle's rule to make sequence repeat forever



```
When the play button is tapped
    Set Size Percent 200
    Repeat Forever
        Set Position to X 1024 Y 400
        Change X by -1000
    End
```

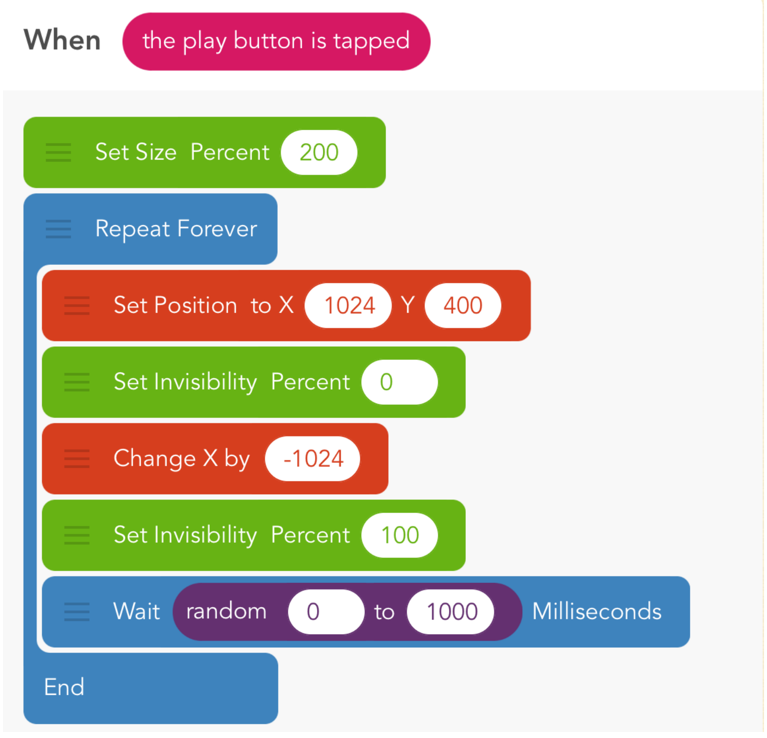
When moving code into the repeat block, make sure to not change the order. Students will probably make a mistake here—a good opportunity for debugging!

3.5 Edit obstacle’s rule to make the obstacle wait before moving



```
When the play button is tapped
Set Size Percent 200
Repeat Forever
    Set Position to X 1024 Y 400
    Change X by -1000
    Wait random 0 to 1000 Milliseconds
End
```

3.6 Edit obstacle’s rule to make the obstacle visible only when moving



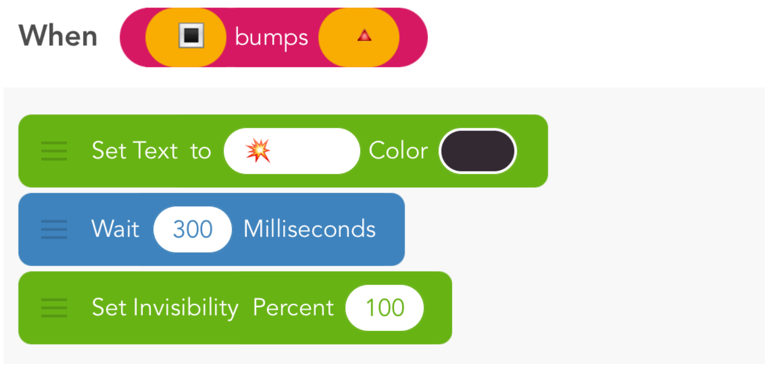
```
When the play button is tapped
Set Size Percent 200
Repeat Forever
    Set Position to X 1024 Y 400
    Set Invisibility Percent 0
    Change X by -1024
    Set Invisibility Percent 100
    Wait random 0 to 1000 Milliseconds
End
```

4. Collisions [10 minutes]

When two objects bump into one another, it is called a **collision**. A collision is a type of **event**, so we can decide what actions should happen when that event occurs. In Geometry Dash, when the hero collides with an obstacle, the game is over.

To designate "game over," upon a collision the hero will explode and then disappear. In Hopscotch, when an object is invisible, it can no longer collide with anything, be tapped, or swiped. Spend some time testing this sequence and getting the timing just right.

4.1 Add new collision rule to hero



```
When ■ bumps ▲
    Set Text to 💥 Color ●
    Wait 300 Milliseconds
    Set Invisibility Percent 100
```

If you cannot find the "bumps" event, tap "more" in the event menu. You can change the object into an explosion, make it spin around, or drop off the screen like Mario. Turning invisible is necessary, because it stops the game from being playable.

5. Add a background (10 minutes)

Drawing the background is a skill that you can apply to any game. Because drawing is just like any other code, you have to choose an object to be in charge of drawing. It is customary to make this object invisible, so you don't see the thing itself, only the picture it draws. For this reason, it doesn't really matter which object you choose.

In Hopscotch, we draw with a block called "Leave a trail" that sets the color and width of the line, then executes the code inside – typically "Move forward" – as if the object were dragging a marker behind it. It will make a dot if it just moves by 1. To color in the whole screen, make a huge dot (width 3000). To make a thick line, you have to set the position to where you want it to start, and then move along the desired path.

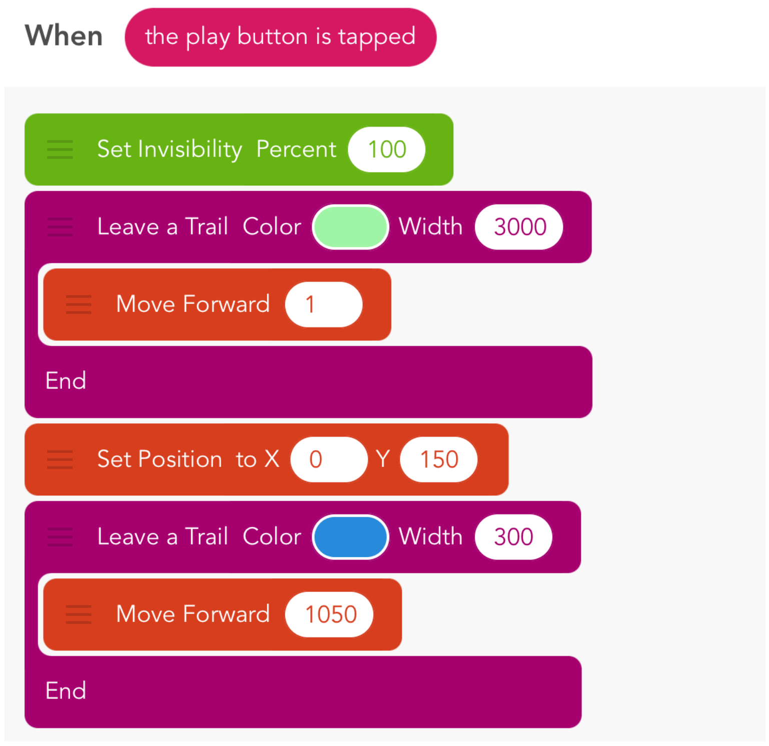
This is another opportunity for debugging. Have the students make a prediction about the following questions and then test out changing their code. What happens… if you don't put anything inside the drawing block? …if you forget to set the width? …if you set the color to white? …if you don't set the position before you start?

Then, have students attempt drawing their background on their own. They can change the artist's speed to draw the background faster.

5.1 Add drawing object (choose anything)

5.2 Add rule to drawing object to paint background

Set the invisibility to 100 so you can't see the painter.

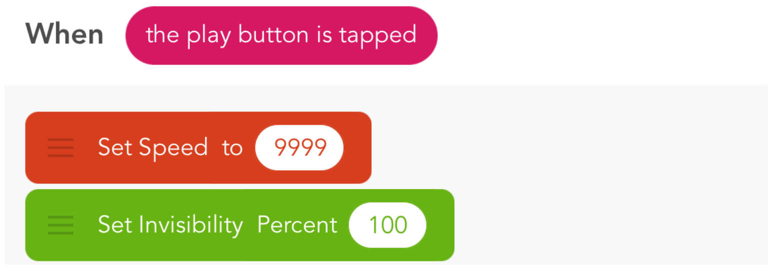


```
When  the play button is tapped

  Set Invisibility  Percent  100
  Leave a Trail  Color 🟩  Width  3000
    Move Forward  1
  End
  Set Position  to X  0  Y  150
  Leave a Trail  Color 🟦  Width  300
    Move Forward  1050
  End
```

5.3 Edit drawing object's rules to draw faster

The default speed is 400. 9999 is as high as you ever need to go; that speed is indistinguishable from 999999999...



```
When  the play button is tapped

  Set Speed  to  9999
  Set Invisibility  Percent  100
```

5.4 Publish your game for others to play and remix!

DIFFERENTIATION

(15 minutes, optional)

- Draw a better background
- Make the background colors random
- Add more obstacles (two or three emojis in a row is a possibility, make movement into an ability)
- Set the obstacle size to random each time; pick a good range!

REFLECTION

(5 minutes, optional)

- What are computers good at? What are they bad at?
- How does this compare to what humans are good and bad at?
- Is drawing with a computer easier or harder than drawing with pencil and paper? Why? If it is harder, why do we still do it?

GLOSSARY FOR YOUNGER STUDENTS

Ability: Code that can be reused

Algorithm: A recipe for a program

Coding: Telling computers what to do

Concurrence: Two things happening at the same time

Conditional: Statements of the form "IF (something is true) THEN (do an action)".

Debugging: Finding mistakes in your code and fixing them

Event: When something happens

Iteration: Having ideas and making mistakes, over and over

Logic: The process of making decisions

Loop: Code that repeats

Operator: A mathematical symbol that makes an equation

Program: A set of instructions a computer can understand

Programmer: A person who writes programs

Programming Language: A set of rules or blocks that can be used to write any program

Random: When there's no pattern

Range: The highest and lowest number random can choose between

Rule: Instructions that tell your computer what to do (the command) and when to do it (the event)

Sequence: The order in which instructions are given to the computer

Object: A character or text with its own rules

Value/Variable: A holder for a number

GLOSSARY FOR OLDER STUDENTS

Ability/Function/Procedure/Subroutine: A saved set of blocks. What we call abilities in Hopscotch are known as functions or subroutines in other programming languages. Easily replicable routines are a key concept in computer programming, and allow you to scale your code and create complex programs.

Algorithm: Algorithms are at the heart of computer science; they are the recipes that computers follow to solve problems.

Bug: An error that a programmer has made in their code

Coding: Writing the rules of behavior for a computer to follow automatically; programming

Concurrency: Two or more things happening at the same time, or triggered by the same event

Conditional: Statements of the form "IF (something is true) THEN (do an action)"

Debugging: Finding mistakes in your code (bugs) and fixing them

Event: A trigger that the computer recognizes and causes it to do some action. In Hopscotch, events include "When the iPad is tapped" or "When the play button is tapped"

Iteration: the repetition of a process

Logic: the science of the formal processes of thinking and reasoning

Loop: a repeating set of instructions

Operator: a mathematical symbol that produces a value

Program: a set of instructions a computer can understand

Programmer: a person who writes programs

Programming Language: a set of words, rules, blocks or instructions that can be used to write a program.

Random: Any number or item among a set. The lack of a pattern among items in a set.

Range: The highest and lowest number random can choose between

Rule: Rules tell your object what to do and when to do it. When you make an ability and pair it with an event, you create a rule.

Sequence: An ordered list of things (instructions, blocks, numbers, etc) which can be triggered by an event or repeated

Object: A character or text with its own rules on screen

Value: A holder for a number. Also known as a variable