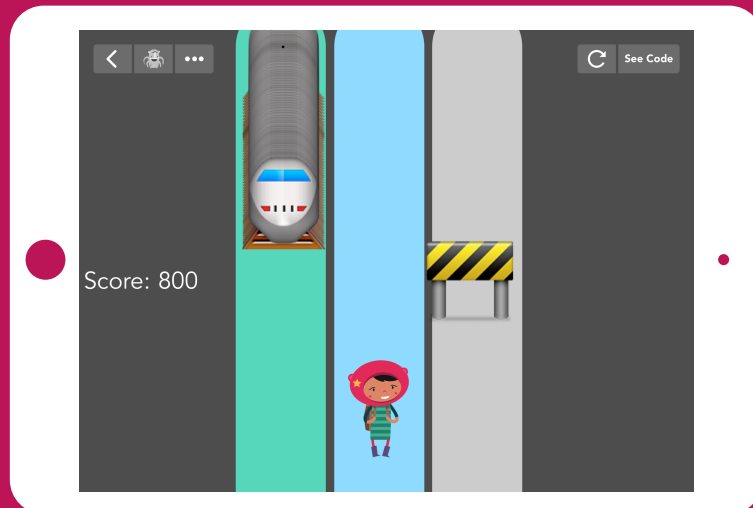


LESSON 5

SUBWAY SURFERS



An action game that has multiple possible implementations.

TIME

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

BIG IDEA

There is often more than one solution to a problem, and some solutions are better than others. There may be another way!

SKILL FOCUS

-
- Collaboration
 - CCSS.MATH.PRACTICE.MP3 Construct viable arguments and critique the reasoning of others
 - NGSS Practice 7 Engaging in argument from evidence
 - NGSS Practice 8 Obtaining, evaluating, and communicating information

KEY VOCABULARY

Pair programming: A technique in which two people work together on one device

TRANSFER GOALS

-
1. Students will try pair programming.
 2. Students will use appropriate vocabulary to communicate with their partner.
 3. Students will compromise and agree on solutions.
 4. Students will formulate strategies for dealing with disagreement, and compare solutions based on implementing and evaluating, without taking it personally.

MATERIALS

– 1 iPad per student, or 1 iPad per 2 students, for pair programming
– Video available on YouTube:
<http://hop.sc/SubwaySurferVideo>
– Complete project available:
<http://hop.sc/subwayproject>

Subway Surfers is an arcade-style game where the player controls a character running in three lanes, dodging trains and other obstacles by swiping to change lanes. We will build a stripped-down version of this game, focusing on the interaction between the hero and the trains. In extra time, students may be inspired to add the other elements of Subway Surfers, like collecting coins and bonuses, and jumping and/or rolling.

This lesson is a great chance to introduce **pair programming**, a technique in which two people work together on one device.

This lesson is an opportunity to work together, using relevant vocabulary, explaining ideas and comparing possible solutions. Assign pairs in whatever way you prefer. Where there is an odd student, create a group of three. The key to working together is understanding that there may be more than one solution to a problem, and the team decides together which solution is best based on how well it works—not on whose idea it was!

The crux of this lesson is deriving the formula for the train choosing one of three lanes to appear in. Be sure to familiarize yourself with it in advance of teaching the class.

LESSON

0. Discussion (5 minutes)

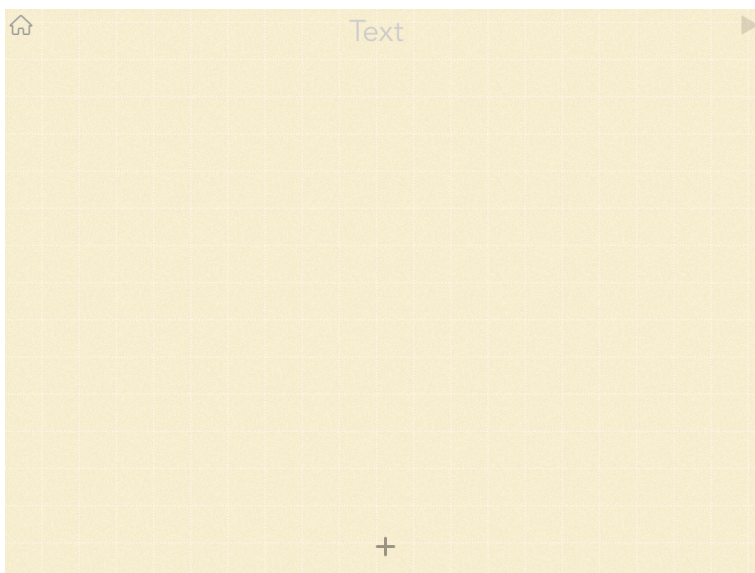
In **Pair Programming**, two programmers take turns coding. The driver holds the iPad and does the coding, and the navigator watches for mistakes, helps come up with solutions, and keeps the pair on task. Every few minutes, they switch roles. Both jobs are equally important! This is especially helpful when making complex games, because there are lots of details to attend to, and an extra pair of eyes can help spot errors before they become bugs. In professional environments, this is often considered a best-practice (and we do it frequently at Hopscotch).

What things would you need to keep in mind while pair programming? Ask your students to make a list of things that they should and should not do while pair programming (e.g. they should ask questions about why the programmer chose a certain block; they should not call each other names.)

1. Draw lanes (S) (10 minutes)

Show your students a completed version of the game. The first step is to create the three colored tracks that run down the middle of the screen. See if your students can draw three random-color lanes by reusing as much code as possible (abilities are helpful for this!). One solution is to make one drawing object, and repeat setting the position and drawing the line. Another possibility is to make three separate drawing objects that each use the same "Draw Lane" ability.

1.1 Add an invisible text object to the top middle of the screen

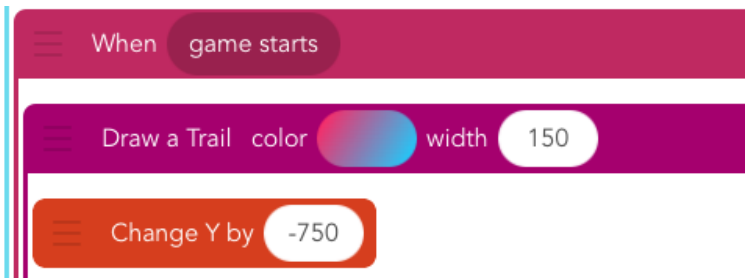


Remember that a shortcut for an invisible object is to make a text object, then tap "X" on the upper left corner when it asks you to set text.

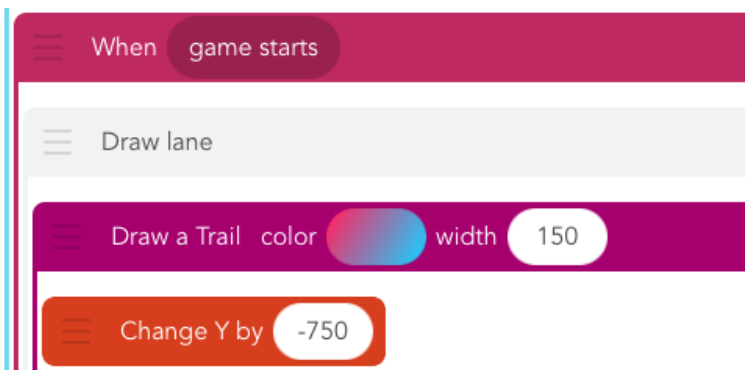
How would your code change if you put the text object at the bottom of the screen? It would have to move by a positive amount.

LESSON

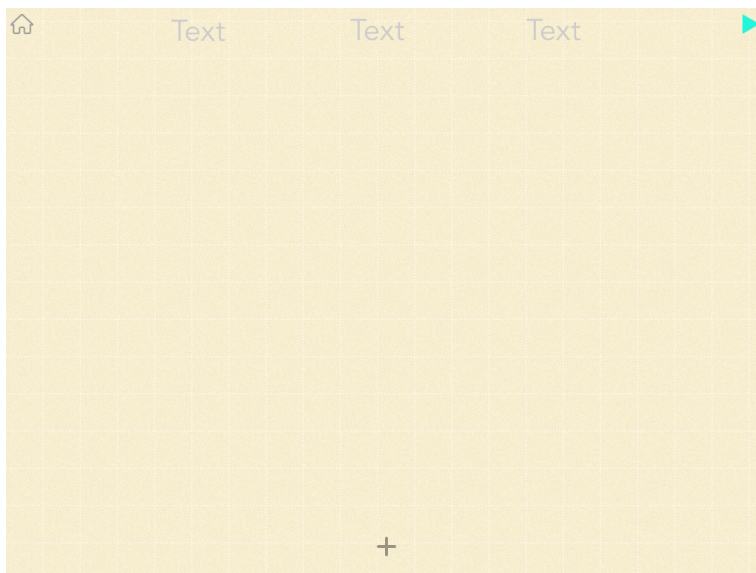
1.2 Add code to text object: Draw lane



1.3 Make it a custom block



1.4 Add two more text objects at appropriate places, give them the "Draw Lane" block



2. Control hero (ELCV) (10 minutes)

As your students saw when playing the sample game, in Subway Surfers the player controls the hero by swiping right and left on the iPad, jumping across lanes to avoid the oncoming objects. In students' versions, they will make the player swipe the hero, rather than the iPad.

LESSON

The rules for animating and controlling the hero are a chance to practice the concepts of events and loops learned in previous lessons. You can discuss the general requirements of this action as a class and then give your students the chance to try coding these rules on their own. The sample code below is just one possible solution.

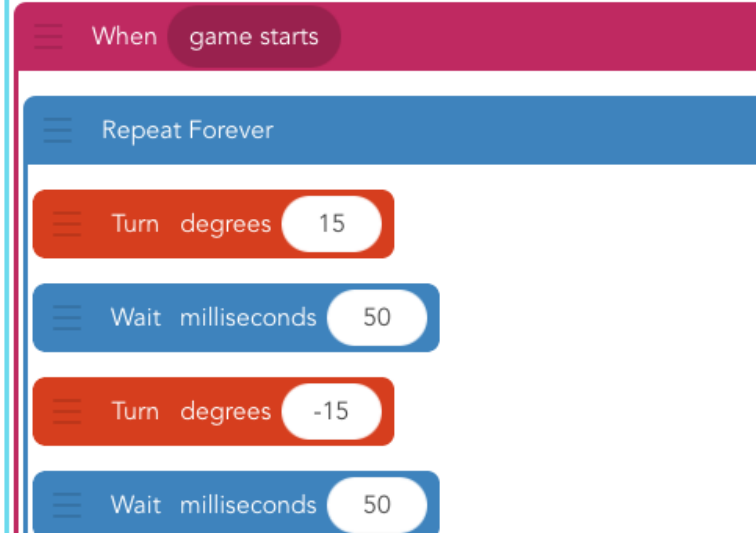
Check in to compare the results and make sure everyone's is working and, if your students have not brought it up, then raise the issue that nothing is stopping the player from swiping their hero out of the lanes and "off the board". In order to establish boundaries, we can introduce logic that only allows the character to move within a certain zone.

In our game, the character can move in one case but not another; we will tell the computer to check and see if a condition is true before moving on to execute the command. We do this with a **conditional** that checks the character's position on the X axis.

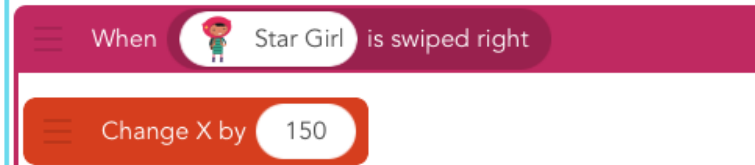
As a class, decide the range of X positions for which the character may be moved right and left, respectively. Derive the inequalities from these numbers (e.g. If character's X position is greater than [some number] you cannot move it further to the right). Code these rules as a class, in pairs, or as individuals. The numbers in the sample code are a possible solution.

2.1 Add hero object

2.2 Add new code to hero: animate hero object to run forever



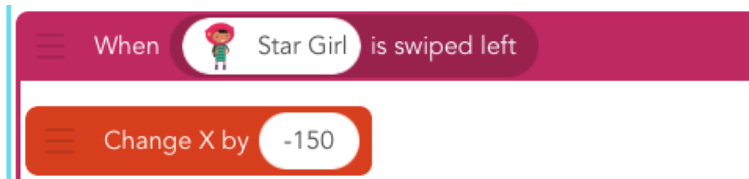
The image shows a Scratch script starting with a 'When game starts' event block. This is followed by a 'Repeat Forever' loop containing four blocks: 'Turn degrees 15', 'Wait milliseconds 50', 'Turn degrees -15', and 'Wait milliseconds 50'. A vertical line is drawn to the right of the loop, and the text 'swiped right' is written next to it, indicating the condition for the next script.



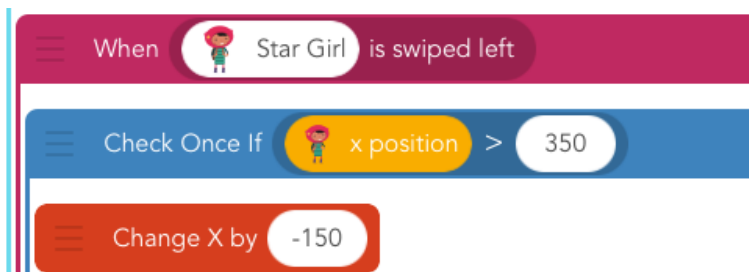
The image shows a Scratch script starting with a 'When Star Girl is swiped right' event block. This is followed by a 'Change X by 150' block.

LESSON

2.4 Add new code to hero: Change X by -150 when swiped left



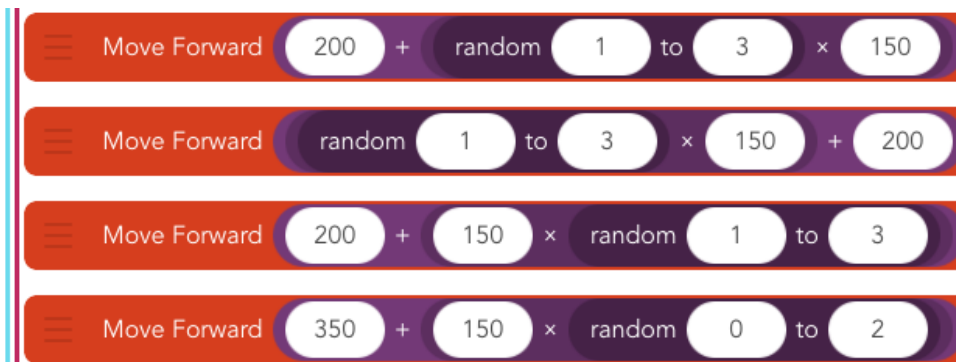
2.5 Edit swipe code to check for hero's X position



3. Add trains (SV) (10 minutes)

The most challenging aspect of this game is setting the trains to appear as challenging obstacles. Students should start, in pairs, by discussing how to program one train to move along one lane, starting at the top. If they forget the height of the screen, they can use the built-in value, "iPad's height".

Once they have a train running in one lane, they will need to program it to randomly appear in different lanes. This is accomplished by choosing a range of X-positions in which it may appear, using one of these formulas for the X position:



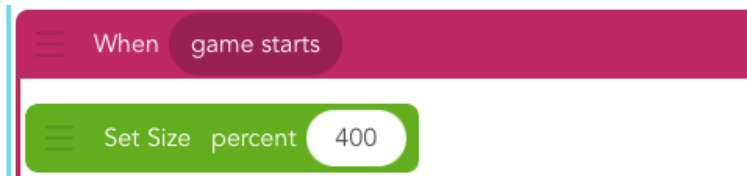
Note: If you're on an iPhone, you will need to adjust the width of the lanes to fit the smaller screen.

LESSON

If these formulas are confusing, check out the code below. For older students, work out this formula in pairs. Younger students may need to be shown or led to discover the formula. In either case, test the formula out with the possible values on paper before trying it in Hopscotch. Have students code it when they think they have it right, then evaluate the formula by testing its effect in the game.

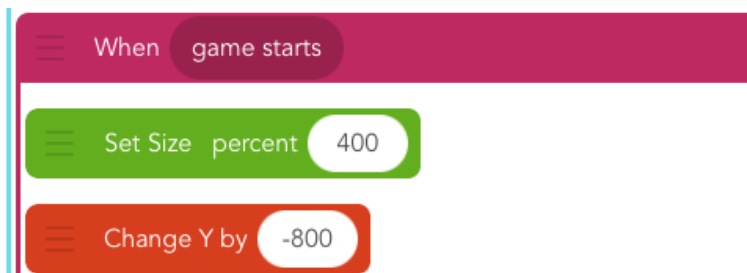
You can also ask the class how they would change the code if their lanes were wider. (change the X-position by 150 to accommodate lane width, make 200 lower). How would they change it if they had more than 3 lanes? (Change second number range to number of lanes, make 200 lower) For older students: Consider that these changes affect more than just one number; can you think of a general-purpose equation using variables for "NumberOfLanes" and "LaneWidth"?

3.1 Add train object and make it bigger



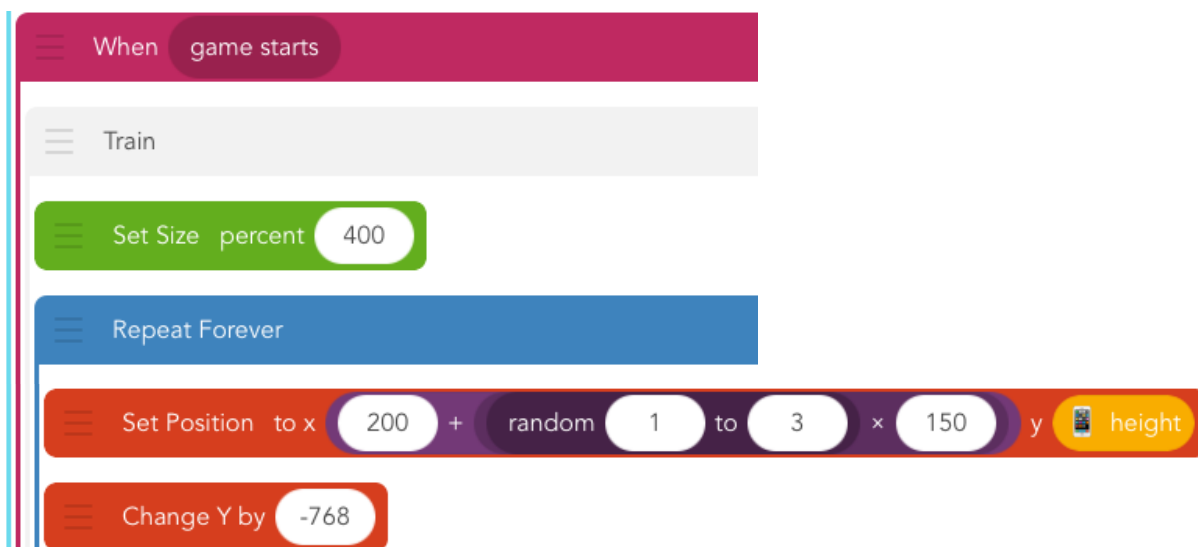
Name train object "Train".

3.2 Add new code to train: Make it run along one track



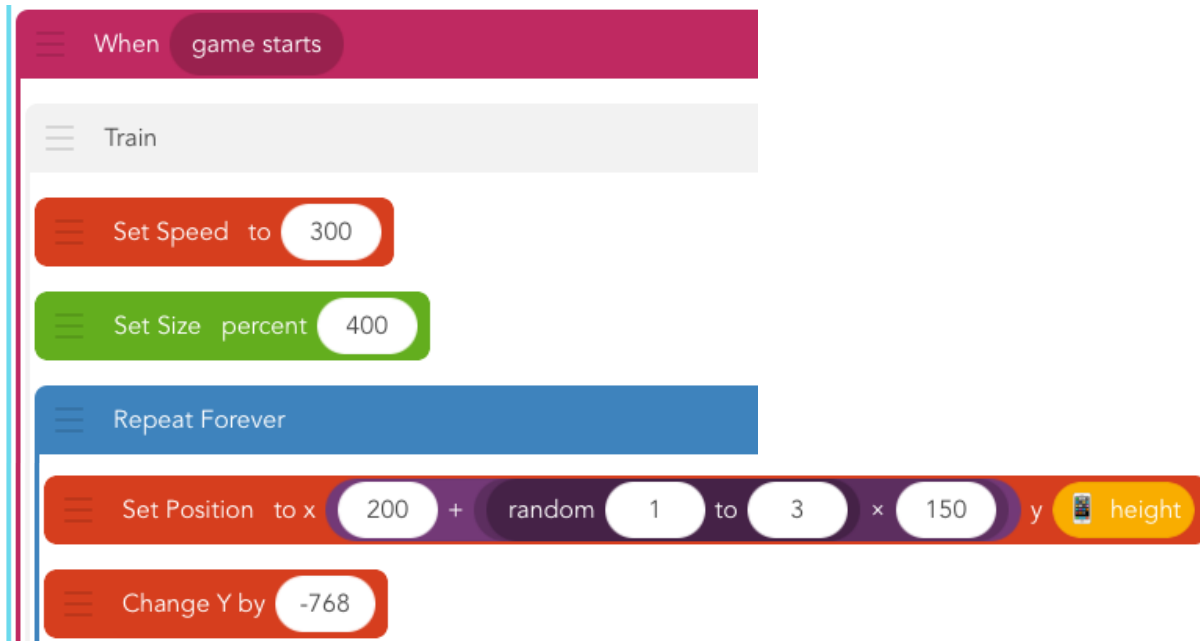
*Or, you can
Change Y by $(-1 * \text{iPad's Height})$.*

3.3 Edit code to choose random lane



LESSON

3.4 Adjust speed



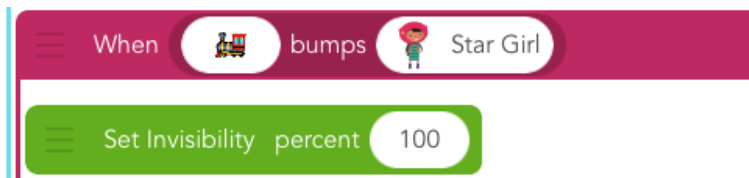
The image shows a Scratch script for a 'Train' object. The script starts with a 'When game starts' event block. Below it is a 'Train' object block. The script then contains the following blocks in order:

- 'Set Speed to 300'
- 'Set Size percent 400'
- 'Repeat Forever' loop containing:
 - 'Set Position to x 200 + random 1 to 3 x 150 y height'
 - 'Change Y by -768'

4. Collisions (ES) (5 minutes)

Ending the game is an opportunity for creativity. How do students want to indicate to the player that a collision has occurred? If you want, for example, to grow and spin at the same time, that's concurrency, and should be implemented with two different rules with the same event. Ask students to come up with a good indication that a collision has occurred, code it, and show it to a partner. The partner can help debug if necessary.

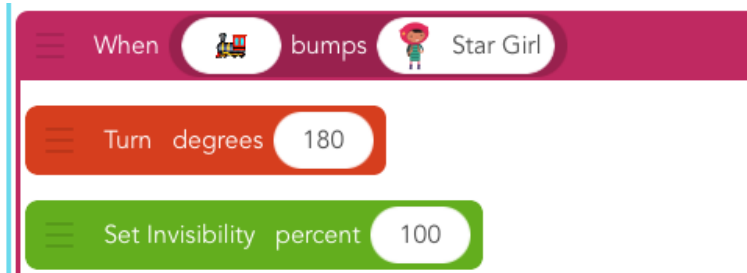
4.1 Add new code to hero: Disappear when you collide with a train



The image shows a Scratch script for a 'Star Girl' object. The script starts with a 'When bumps Star Girl' event block. Below it is a 'Set Invisibility percent 100' block.

LESSON

4.2 Edit hero's code: Animate collision



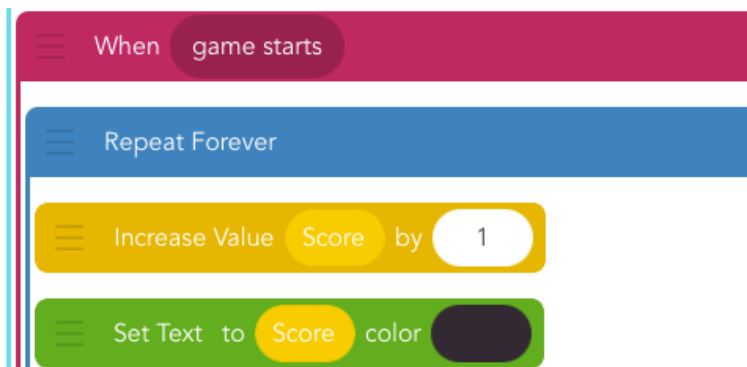
5. Keep score (ELV) (5 minutes)

As students might recall from earlier lessons, we use values to store numbers that may be changed. To create a score, students should introduce a text object that will keep track of and display the score. In this case, though, the score shouldn't keep increasing forever; it should stop when the hero collides with a train. You can ask students to discuss a solution to this problem. One possible way to implement the score is to change the event that triggers this rule, so instead of happening forever, it only happens when the hero is visible on the screen (e.g. has not collided with the train).

5.1 Add a score object

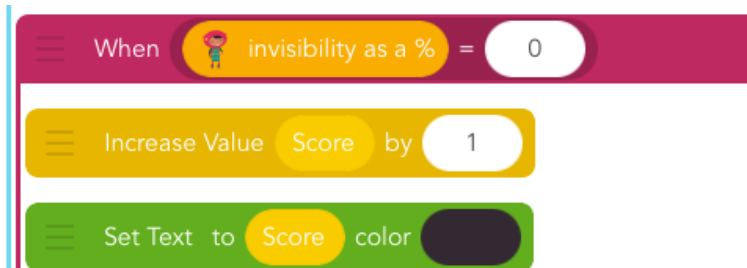
Name score object "Score".

5.2 Add new code to score: Make it count up by 1 forever and display score



The score is increasing as fast as the computer can go. Is that too fast? Add a wait block to slow it down. Find a good value for wait.

5.3 Change the event and delete the loop



The reason you can delete the loop is that this event is already repeating!

5.4 Publish your game!

DIFFERENTIATION

(15 minutes, optional)

In Subway Surfers, you have the option of jumping over or rolling under barriers, in addition to the option of avoiding them. Can you add this functionality to your game? Devise a strategy in pairs, either on paper, or in code. Share your ideas with the class and see how they are similar. Can we come up with a better solution by combining our ideas?

- Add hurdle object
- Make train's behavior a function, then give hurdle the same train rule
- Give hero a swipe down rule to roll - make "rolling value"
- When hero bumps hurdle, check if rolling
- Can you jump over the hurdles by swiping up?
- Can you make the trains speed up as the game goes along?

REFLECTION

(5 minutes, optional)

- What is pair programming? Why is it useful?
- Can you see a pattern in the order we use to build a game? Why did we choose that order? Is it the best, or can you think of improvements?
- Why are some values a different color than others? (Some are built-in and you change them with dedicated blocks, others are the ones you made and you have to change them by using set value or increase value.)