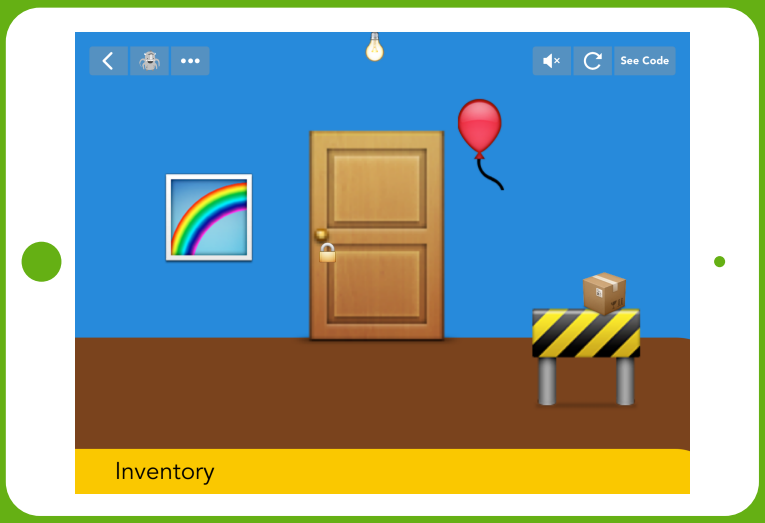
LESSON 6
CAN YOU ESCAPE?



An open-ended logic game, enabling students to make the connection between real-world logic and programming logic.

TIME

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

BIG IDEA

The way to make good programs is to have ideas and make mistakes, over and over. Stick to it!

SKILL FOCUS

- Iteration
- CCSS.MATH.PRACTICE.MP1 Make sense of problems and persevere in solving them
- CCSS.MATH.PRACTICE.MP6 Attend to precision
- Practice 1 Defining problems
- Practice 6 Designing solutions

KEY VOCABULARY

Logic: The process of making decisions
Iteration: Having ideas and making mistakes, over and over

TRANSFER GOALS

1. Students will generate ideas for game elements and design solutions.
2. Students will test solutions and design improvements.
3. Students will recognize that some ideas take more than an hour to implement.
4. Students will be able to define the 5 Core Coding Concepts and give examples of their uses.
5. Students will recognize the programming convention of 1=True, 0=False.

MATERIALS

– 1 iPad or iPhone per student, or 1 device per 2 students, for pair programming
– Video available on YouTube: http://hop.sc/EscapeTheRoomVideo
– Complete project available: http://hop.sc/escapetheroomproject

An Escape Game is an adventure game in which, surprise surprise, the player is locked in a room and must locate objects and solve puzzles to escape. Can You Escape? is a current example, but The Room is related, and Myst and Monkey Island are ancestors. This type of game relies heavily on logic, order of events, and attention to detail. It pulls together the programming concepts introduced in previous lessons and gives students a great opportunity to be creative. Students will create **values** to keep track of the state of the game, then use **conditionals** and **events** to change the behavior of the game, depending what state it is in.

The foundation of this game, and computing as a field, is **binary logic**. In Escape the Room, students will use binary logic to assess whether the player has completed the activities required to escape. While they have not, they cannot leave (the escape condition is untrue, or 0). Once they have completed the activities required, the escape condition becomes true (1) and the player can escape!

This activity is an excellent opportunity for customization and extra time. It can be worthwhile to start with a quick summary of these concepts and prepare students to implement them on their own. This process of making small, incremental changes to your code—**iteration**—is a key tenant of programming.

LESSON

0. Discussion of Escape genre (5 minutes)
Some of your students will be familiar with the escape game genre, and some will not. Take a few minutes to get everyone on the same page. Look at some examples on the projector, or elicit descriptions of good escape games from the class. Talk about why they are fun. A good escape game should be challenging, but not impossible, and have a few puzzles, but not too many. Are there other important elements, like an interesting setting, good animation, or sound effects?

Remind your students that the best programmers make lots of small changes to their code and test them frequently. This process, **iteration**, helps you identify bugs early and make sure that everything works as you intend. We, at Hopscotch, are always reminding ourselves to slow down and check our work. :)

1. Intoduction to basic logic (EVC) (10 minutes)
Discuss the basic premise of the game with your class: the player is trapped in a room that has a locked door and they, initially, do not have the key. The player's task is to find the key and escape, but this is complicated by series of puzzles that they have to solve to find the key.

Since the door cannot open unless the player has found the key, the game must know whether or not it's been found at all times. We will create and use a value "HasKey" to keep track of this condition with the help of one of computing's most important concepts: **binary logic.**

In binary logic, we use 0 to represent FALSE and 1 to represent TRUE. These are known as truth values. Imagine a lamp: When the switch is in the ON position, 1 light is on, so the sentence "a light is on" is TRUE. When it is in the OFF position, 0 lights are on, so the sentence "a light is on" is FALSE. Now, if you have two lamps, you can deal with them both: If the first switch is in the ON position (1) and the second is in the OFF position (0), the sentence "a light is on" (0+1) is TRUE! If you've ever heard jokes about computing being all 0s and 1s, this is why…

Start by having students independently establish the room by adding a key and a door that opens when swiped. Then, in pairs or as a class, discuss the logic required to only open the door when the key is found. Depending on your students' experience level, you might ask them to come up with and share out potential logic rules.

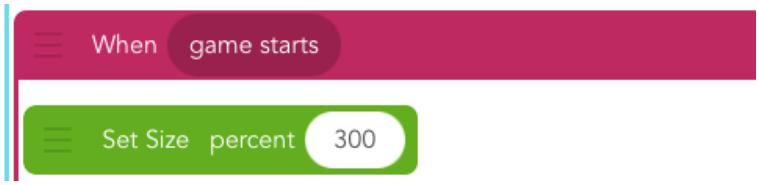
Make sure everyone is on the same page and then have students code the logic rules. In the escape game, when the key is found (tapped), "HasKey" will equal 1 (True), and when it has not, "HasKey" will equal 0 (False). The door should only open when "HasKey" equals 1. If "HasKey" does not equal one, the door should signal to the player that it cannot be opened. This is a good opportunity to discuss the difference between "Check Once If" and "Check Once If // Else".

LESSON

The door cannot be opened unless the player has the key ("HasKey" = 1). Students will use conditionals to create two states:

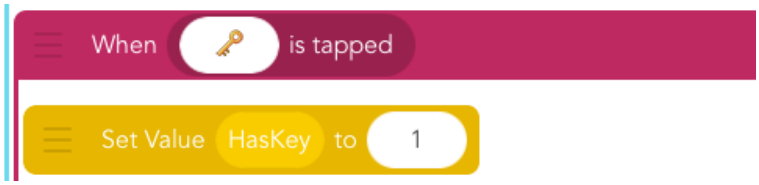
The door opens if the player has the key.
The door wiggles (but does not open) if the player does not have the key.

1.1 Add door & key emojis
1.2 Make the door bigger



1.3 Animate door opening

1.4 Set HasKey to 1 when you get the key



You will need to add the "HasKey" value. Remember that all values are 0 by default before you set them.

1.5 When door is swiped, check if "HasKey" = 1, then execute the opening code.

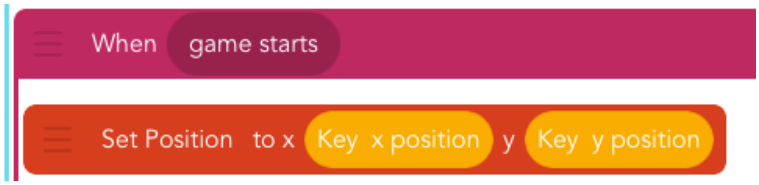
Check out that "Else" condition!

You will need to create your own Wiggle ability. Here is one way to do it.

2. Two-step logic (ES) (10 minutes)

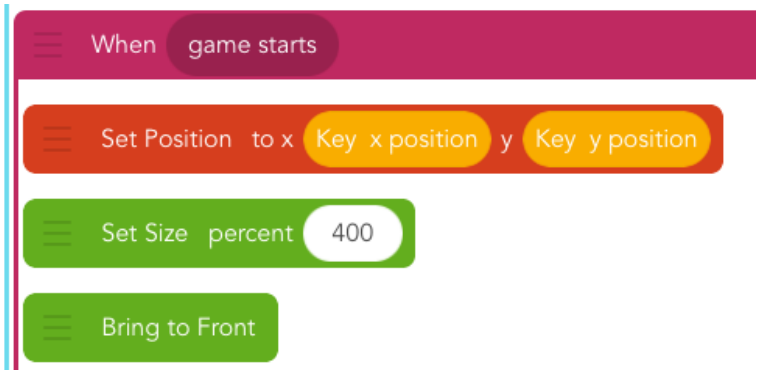
This game is too easy! Let's start adding things to make the key harder to get. One possible solution is to obscure objects by putting one in front of the other and revealing the second only when the first (outermost) has been moved. The below code shows how to do this. Note that objects show up in the order you added them. If you want one character to be in front of another, use "Bring to Front". There is also "Send to Back". What does that do? Discuss where in your code you would use these (generally as the first action).

2.1 Add portrait emoji at same position as key

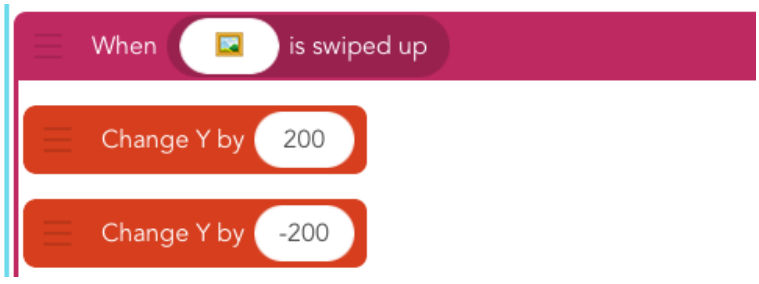


2.2 Make portrait bigger

2.3 Bring to Front



2.4 Make portrait slide up, then back down, when swiped



2.5 Test and adjust positions of objects

3. Inventory (ESV) (10 minutes)

An Inventory is a widely-used game element that helps the player keep track of the things they "have". There are multiple ways to implement an Inventory; one way is to create a designated storage place on the screen with copies of all Inventory items, and then to show or hide each item depending on whether it's been collected. We already keep track of this state using the "HasKey" value!

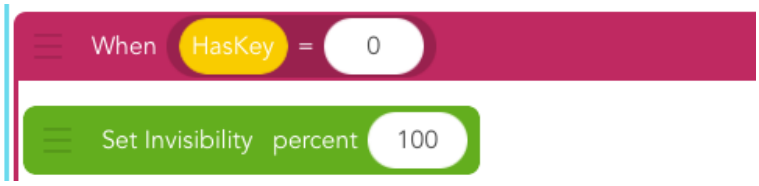
Students should start by drawing an Inventory box, and then add a new key emoji. In pairs, see if they can identify the two rules this key needs (hint: one for each state). As a class, make sure everyone has the right rules. Then, together, adjust the door's rule to change the "HasKey" value back to 0 when it has been used to open the door. Test the program and check whether the Inventory key is appearing and disappearing at the right times.

3.1 Add Inventory label (optional)

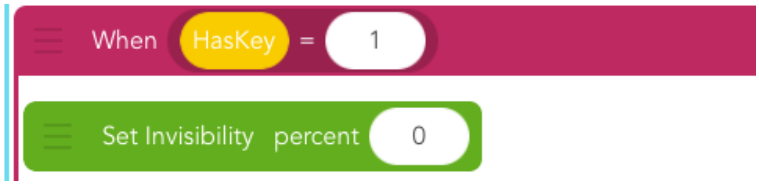
3.2 Draw Inventory box (optional)

3.3 Add a new key emoji to Inventory

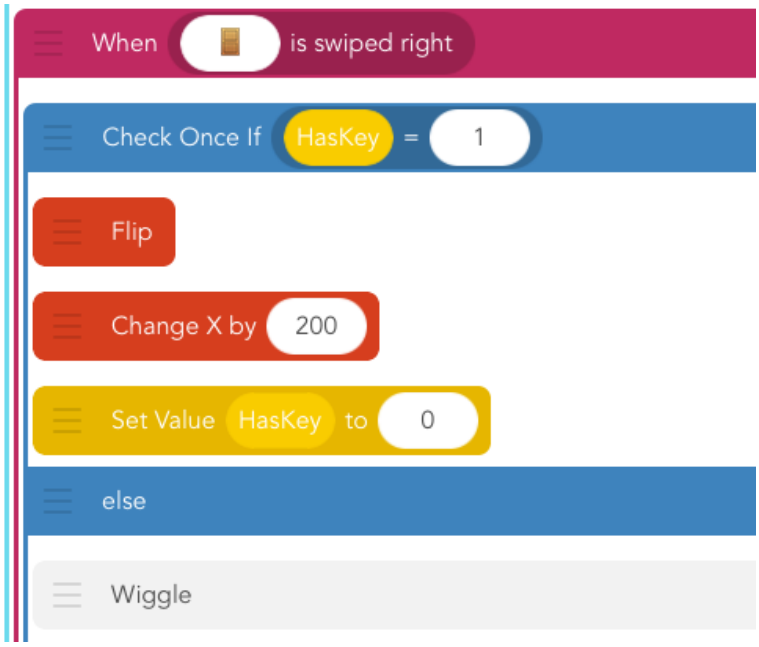
3.4 Add code: Do not display key in Inventory when condition is false



3.5 Add code: Display key in Inventory when condition is true



3.6 Edit the door's swipe code to set "HasKey" to 0 (when you've used it up)



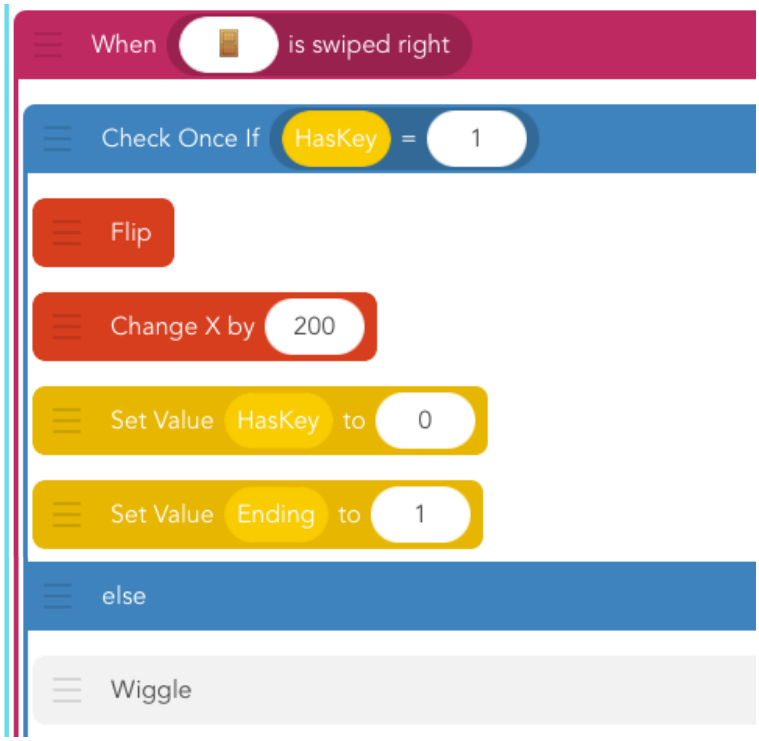
4. Win State (EV) (5 minutes)

It's important to tell the player when he or she has won. We'll keep track of whether the player has won using a new value, "Ending". When the game is finished, "Ending" will be 1. Ask your students to consider which event would trigger the value to increase from 0 (starting value) to 1?

As in previous games, ask students to create a "win state" text that will appear when the "Ending" value is 1.

As a class, discuss what would happen if there were another set of conditions that set "Ending" to 2 or 3 and caused different endings to happen.

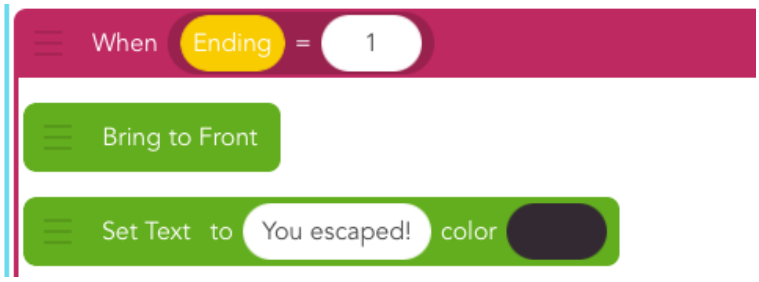
4.1 Edit door's code: Opening door sets "Ending" to 1 (remember all values are 0 by default when you make them)



Create a new value "Ending".

4.3 Add new text object, cancel text

4.4 Add new code to text object



4.5 Publish your game!

DIFFERENTIATION

(15 minutes, optional)

- Show how you could have two different endings
- Add appropriate sounds to each event
- Add more puzzles to the game (search for escape games in the Hopscotch Community for inspiration)
- User testing: Watch silently while other people play your game and observe how they play it
- Active feedback: Ask players what they thought of your game
- Make a real-life escape game in your classroom!

REFLECTION

(5 minutes, optional)

- What is logic? Is there a difference between coding logic and real-life logic?
- What's the difference between IF and IF...ELSE?
- How do you check if two things are both true? (Nested conditionals)
- How do you choose how to respond to feedback? Do you have to make all the changes someone suggests?