15

CS in Algebra | Lesson 15

Sam the Bat

Lesson time: 30-60 Minutes

LESSON OVERVIEW

Using Boolean operators, students will write code that checks the location of a sprite to make sure it doesn't go off-screen.

LESSON OBJECTIVES

Students will:

- Use Boolean operators to compare values.
- Apply Boolean logic, such as AND, OR, and NOT, to compose complex Boolean comparisons.

ANCHOR STANDARD

Common Core Math Standards

- **6.NS.8**: Solve real-world and mathematical problems by graphing points in all four quadrants of the coordinate plane. Include use of coordinates and absolute value to find distances between points with the same first coordinate or the same second coordinate.

Additional standards alignment can be found at the end of this lesson

TEACHING SUMMARY

Getting Started

1) Introduction

Activity: Sam the Bat

2) Online Puzzles

Extension Activities

3) Safe Up and Down

TEACHING GUIDE

MATERIALS, RESOURCES, AND PREP

For the Student

- Safe-left? Design Recipe (in the student workbook)
- Safe-right? Design Recipe (in the student workbook)
- Onscreen? Design Recipe (in the student workbook)

GETTING STARTED

1) Introduction

This is Sam the Bat, and his mother tells him that he's free to play in the yard, but he don't set foot (or wing) outside the yard! Sam is safe as long as he is always entirely onscreen. The screen size is 400 pixels by 400 pixels, so how far can Sam go before he starts to leave the screen?

In this stage students write functions that will take in Sam the Bat's next x-coordinate and a return a boolean. That function should return *true* if part of Sam will still be visible, or *false* if he would go too far off-screen. If the function returns *false*, Sam isn't allowed to move.

Students will start by writing functions to check the left and right side of the screen independently, before combining those with a single onscreen? function that prevents Sam from leaving on both the left and right.

For each stage, make sure students try to get Sam to leave through the side they are checking. If Sam makes it all the way off-screen when he shouldn't, they'll get an error, but if he is successfully stopped they'll succeed and move to the next puzzle.

LESSON TIP

It's extremely valuable in this stage to have three students stand, and act out each of these three functions: - Ask each student to tell you their Name, Domain and Range. If they get stuck, remind them that all of this information is written in their Contract! - Practice calling each function, by saying their name and then giving them an x-coordinate. For example, "safe-left? fifty" means that the number 50 is being passed into `safe-left?`. That student should return "true", since the code currently returns `true` for all values of x. - Do this for all three functions, and have the class practice calling them with different values as well. Note: the volunteer for `onscreen?` should first call `safe-left?`, before replying with the value.

Why not write just one function?

Some students may wonder why they should write separate functions for `safe-left?` and `safe-right?` when `onscreen?` could just check the dimensions of the screen directly. There is more to being a writer than good spelling and grammar. There's more to being an architect or an artist than building a bridge or coloring in a canvas. All of these disciplines involved an element of *design*. Likewise, there is more to being a Programmer than just writing code.

Suppose you just built a car, but it's not working right. What would you do? Ideally, you'd like to test each part of the car (the engine, the transmission, etc) one at a time, to see which one was broken. The same is true for code! If you have a bug, it's much easier to find when every function is simple and easy to test, and the only complex functions are just built out of simpler ones. In this example, you can test your `safe-left?` and `safe-right?` functions independently, before stitching them together into onscreen?.

Another reason to define multiple, simple functions is the fact that it lets programmers be lazy. Suppose you have a few characters in a videogame, all of which need to be kept on the screen. Some of them might only need `safe-left?`, others might only need `safe-right?`, and only a few might need `onscreen?`. What happens if the game suddenly needs to run on computers with differently-sized monitors, where the width is 1000 pixels instead of 400? If you have simple and complex functions spread throughout your code, you'll need to change them all. If your complex functions just use the simpler ones, you'd only need to change them in one place!

Badly designed programs can work just fine, but they are hard to read, hard to test, and easy to screw up if things change. As you grow and develop as a programmer, you'll need to think beyond just "making code work". It's not good enough if it just works - as artists, we should care about whether or not code is *well designed*, too. This is what functions allow us to do! Everyone from

programmers to mathematicians uses functions to carve up complex problems into simpler pieces, which make it possible to design elegant solutions to difficult problems.

ACTIVITY: SAM THE BAT

2) Online Puzzles

Using Boolean logic, you're going to write functions to help make sure Sam the Bat doesn't leave his mom's yard. Head to CS in Algebra stage 15 in Code Studio to get started programming.

EXTENSION ACTIVITIES

3) Safe up and down

The final puzzle of this stage is a Free Play puzzle that will allow you amd your students to experiment with other ways to keep Sam in his yard. The basic activity only prevents Sam from leaving on the left and right, but what about the top and bottom of the screen?

If you add a second variable to the `onscreen?` function to take in Sam's y coordinate, then you can check Sam's position on each axis. As students pursue this extension, encourage them to think about how they wrote small component functions to check the left and right. Could you follow a similiar approach to deal with the top and bottom?

Derived from BOOTSTRAP www.bootstrapworld.org

This curriculum is available under a Creative Commons License (CC BY-NC-SA 4.0)

If you are interested in licensing Code.org materials for commercial purposes, contact us: https://code.org/contact