Kid.js

Simulating Dominos

Using JavaScript

Objective

JavaScript is the most popular programming language, and growing. It was created to power interactive websites, but has since grown beyond the web browser and is used to write applications large and small, on servers and on mobile devices.

This lesson is intended as an introduction to programming JavaScript for students with little or no prior knowledge as well as students familiar with block programming looking to make the leap to “real” coding. We hope to generate a few magical **ah-ha** moments along the way.

This lesson uses the Kid.js JavaScript framework. To get started with Kid.js, create an account at https://kidjs.app.

Grade Level

This lesson is recommended for Grades 3 - 6. Part 3B is optional for more advanced groups.

Key Topics

Topics covered include Functions, Variables, Properties, For Loops and Events.

Materials

A computer or Chromebook with Internet access. Tablets are not recommended.

Recommendations

Working in small groups allows students who may be move computer savvy to help their peers, however ensure that everyone gets keyboard time! Other tips are sprinkled throughout the lesson plan.

Part 1: Create a Domino

10 minutes

Functions

A computer is programmed by giving it a series of commands. Sometimes these commands are referred to as functions. To run a command or function in JavaScript, type the name of the function followed by parentheses.

Many functions need parameters. For example, if you were to command someone to walk, you may also need to tell them how many steps or which direction. The parameters go inside of the parentheses.

We'll use the `rect()` function to create a domino.

```
rect(200, 200, 50, 100)
```

The parameters are (in order) the x coordinate, y coordinate, width and height. Try different values and see the result.

Encourage students to experiment with different parameter values and see the results. While they are doing this, tour the class and ensure everyone was able to get a rectangle to appear. Common issues are a missing comma between parameters, missing parentheses or a mispelt function name.

Variables

Variables are used to keep track of things. Think of a variable as a box with a label you can put things in. There are many things we may need to keep track of in an app, such as high score or health. Take a moment to brainstorm other items you may need to keep track of.

Variables are also useful for assigning names to things. Let's give the domino a name. This will allow us to reference it later.

```
let domino = rect(200, 200, 50, 100)
```

Besides a handful of reserved words, variables can be named anything. Depending on the experience level, you may want to encourage students to choose their own variable names. Remember that whatever name is choosen, the same name must be used later when referring to it. (It also cannot contain spaces or start with a number.)

Properties

Properties describe something. You have properties, such as your age and height. Objects in JavaScript can have properties too.

Our domino is stuck floating in mid-air. For physics to apply, we'll set its "anchored" property to false.

```
let domino = rect(200, 200, 20, 100)
domino.anchored = false
```

Of course, our domino shouldn't be quite so bouncy. In fact, it shouldn't be bouncy at all. Set its "bounciness" property to 0.

```
let domino = rect(200, 200, 20, 100)
domino.anchored = false
domino.bounciness = 0
```

Encourage students to try different bounicess values. It's likely someone may ask about changing the color. This can be done by setting the color property. (e.g. domino.color = 'red')

Part 2: Line 'em Up

25 minutes

Instead of repeating this code for each domino, we'll create our own custom function. Functions are useful for organzing code into reusable blocks. Ideally functions should perform a single action, and its name should describe that action.

```
function placeDomino() {
  let domino = rect(200, 200, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}
```

Like variables, functions can be named almost anything. Encourage students to choose their own name. It's best practice that the name describe what the function does, and isn't too silly.

Where did our domino go? While we created a function to place a domino, we did not run (or execute) it. To do this, type the function name followed by parentheses.

```
function placeDomino() {
  let domino = rect(200, 200, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}

placeDomino()
```

Next, modify the function to accept x and y parameters. This will allow us to not just place a domino, but specify where we want to place it.

```
function placeDomino(x, y) {
  let domino = rect(x, y, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}
```

We can now run the function multiple times passing different x and y values to place more dominos.

```
function placeDomino(x, y) {
  let domino = rect(x, y, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}

placeDomino(200, 200)
placeDomino(260, 200)
placeDomino(320, 200)
```

This is a great time to pause and ensure all students are caught up to this point. Instruct students who are ahead to continue filling up their screen with dominos, or to help other students.

It's going to take a lot of lines of code to fill the screen with dominos. Luckily there is a better way. We'll use a piece of code called a for loop.

The For Loop

A for loop is used to repeat a block of code. It's broken into three parts. Code that runs before the loop starts; a condition to check if we should continue; and code that runs after each pass through the loop.

In the first part, we'll create a new variable, x, to keep track of the horizontal position of the domino. Let's start it at 200.

For the second part, we check to see if x is less than the width of the screen (minus 200 pixels.) If yes, then the loop continues.

For the third part, we add 60 pixels to x. This moves the position of the next domino to the right.

```
function placeDomino(x, y) {
  let domino = rect(x, y, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}

for (let x = 200; x < width - 200; x = x + 60) {
  placeDomino(x, 200)
}
```

For loops are not always intuitive for students and it often takes time to become familiar. Encourage students to play with the numbers and see the effect. If time permits, take a break from the activity and walk through a few example for loops step-by-step. If your students are familiar with block coding, make the comparison to Repeat and Repeat Until blocks.

Part 3A: Knock 'em Down

25 minutes

Now for the fun part! To knock the dominos down, we'll place a marble where the user clicks and launch it towards the dominos.

Events

Events are "things" that happen while our app is running, like a mouse click, or key press. We can write code that runs when that event occurs.

The first step is to write a function that is intended to run when the event occurs. These types of functions are referred to as Event Handlers.

Our function will create a new marble and give it a push. We'll use the `circle()` function to create a marble, and `push()` to set it in motion. The parameters for the circle function are the x and y position and diameter. The parameters for push are the amount of horiztonal and vertical force to apply. Kid.js provides two global variables, `mouseX` and `mouseY` that contain the current mouse coordinates.

> Global variables are variables that are accessible from anywhere in our code. See Part 3B for further discussion on scope.

```
function launchMarble() {
  let marble = circle(mouseX, mouseY, 50)
  marble.push(20, 0)
}
```

Then connect the function to the "click" event.

```
on('click', launchMarble)
```

And **voilà**, our domino simulation is complete.

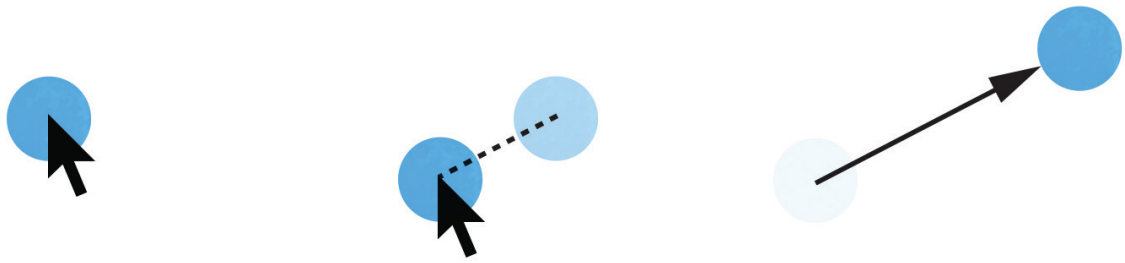
```
function placeDomino(x, y) {
  let domino = rect(x, y, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}

for (let x = 200; x < width - 200; x = x + 60) {
  placeDomino(x, 200)
}

function launchMarble() {
  let marble = circle(mouseX, mouseY, 50)
  marble.push(20, 0)
}

on('click', launchMarble)
```

Part 3B: Sling Shot

25 minutes (Optional)

Let's take it further. Instead of placing the marble on click, let's allow the user to click and drag to aim and "sling-shot" the marble towards the dominos. It will go something like this:

- **On mouse down,** save the mouse position in a varible and create the marble.
- **On mouse move** (while pressed) move the marble to the current mouse position.
- **On mouse up,** launch the marble towards where we first clicked.



Scope

Before we go much further, we need to talk about variable scope. Variables only live inside of the block of code they are defined in. If we define a variable inside of a function, it only exists inside that function. It cannot be accessed by code outside of the function.

We're going to need three variables to launch our marble. Two variables to store the x and y location where we started dragging and another variable for the marble itself. Several functions will need to use these variables, so we define them outside of any function.

```
let startX
let startY
let marble
```

Start

When the user presses down on the mouse button, we save the current mouse position in `startX` and `startY` and add a circle to the stage for the marble.

We'll write this code in a function and attach it to the "mousedown" event.

```
function start() {
  startX = mouseX
  startY = mouseY
  marble = circle(startX, startY, 50)
}

on('mousedown', start)
```

Drag

When the user moves the mouse, we'll update the position of the marble. We only do this while the mouse button is pressed, otherwise the marble will follow the mouse even after it is launched.

We'll write this code in a function and attach it to the "mousemove" event. We can check if the mouse button is pressed by looking at the global `mouseButton` variable.

```
function dragging() {
  if (mouseButton) {
    marble.x = mouseX
    marble.y = mouseY
  }
}

on('mousemove', dragging)
```

And Release!

Finally, when the mouse button is released, we launch our marble. We'll do a little math to figure out the direction we want to push the marble by subtracting the starting coordinates from the current mouse coordinates.

```
function release() {
  marble.push(startX - mouseX, startY - mouseY)
}

on('mouseup', release)
```

Putting It All Together

Let's put it all together. We use `placeDomino()` and a for loop to line up all the dominos, and can now click and drag to launch a marble to knock them down.

```
function placeDomino(x, y) {
  let domino = rect(x, y, 20, 100)
  domino.anchored = false
  domino.bounciness = 0
}

for (let x = 200; x < width - 200; x = x + 60) {
  placeDomino(x, 200)
}

let startX
let startY
let marble

function start() {
  startX = mouseX
```

```
  startY = mouseY
  marble = circle(startX, startY, 50)
}

function dragging() {
  if (mouseButton) {
    marble.x = mouseX
    marble.y = mouseY
  }
}

function release() {
  marble.push(startX - mouseX, startY - mouseY)
}

on('mousedown', start)
on('mousemove', dragging)
on('mouseup', release)
```

Taking it Further

- Add rectangles and other shapes to act an platforms and barriers. Line up even more dominos!
- Use the mouse click event to place dominos. Use a modifier key, such as shift, to change to launch mode.
- Play around with the various physics properties (gravity, bounciness, etc.)

Notes