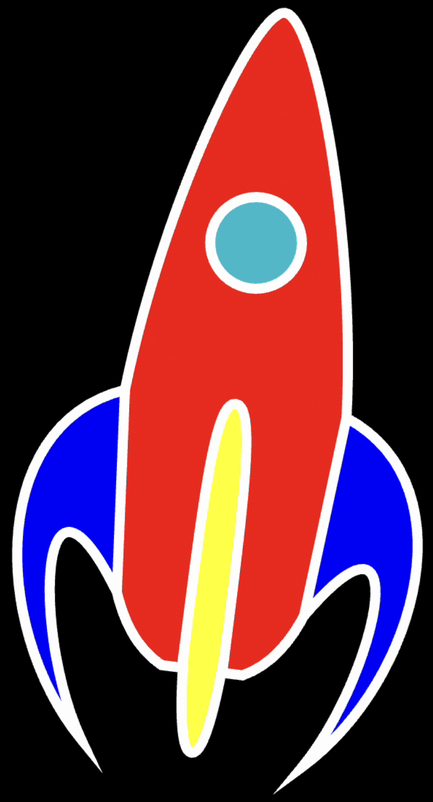
# Animation & Interaction

## CF1

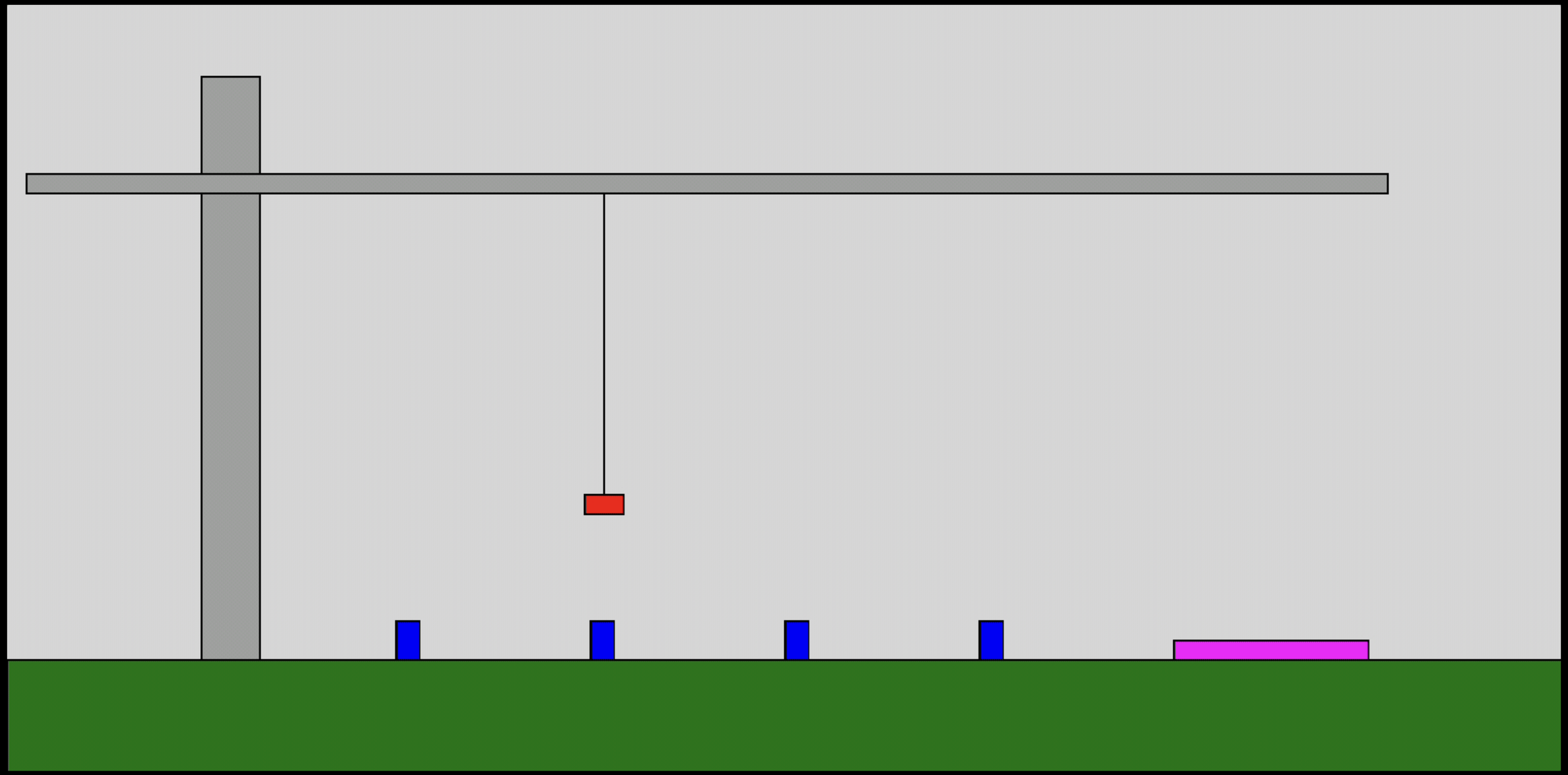
Gabe Johnson | Sept 3 2024



This is a complicated interactive & animated rocket. We haven’t covered all the things needed to code this yet, but over the course of the front half of the course we will get there.

https://editor.p5js.org/johnsogg/sketches/O-hdnAMNJ

We’re going to code this weird little game live in class



https://editor.p5js.org/johnsogg/sketches/J9VHe2u_a

# Setup & Draw Functions

## Once and Forever

- The Processing environment will call setup() one time at the beginning of (its conception of) time
- … and it will call draw() 60 times per second, forEVER
  - You can change the frame rate with the frameRate() function

# Variables

## Give names to your favorite things

- A variable is a symbol that has a value, like:
  - mouseX: the horizontal coordinate of the mouse
- Functions are also symbols, but generally we use them by invoking them:
  - const x = someFunction(someArgument);
- Here we invoke someFunction using someArgument as input, and capture the result into the variable that we call x
- ‘const’ means “it should never change”. If you want to put different values into the variable, declare it with ‘let’

# Variables

## Give names to your favorite things

- ‘const’ means “it should never change”.
- If you want to put different values into the variable, declare it with ‘let’:
  - let x = someFunction(someArgument)

- In some older sources on the web might use ‘var’. Don’t use var. It is a holdover from 1995, and we don’t want any further reminders of the 90s.

# Declare -> Initialize -> Use

## And when do we do these things?

- Declare a variable with let (or const), followed by the name:

```
let x; // I do declare, there is a variable named x
```

# Declare -> Initialize -> Use

## And when do we do these things?

- Initialize a variable by putting a value into it for the first time.
- Typically this is done on the same line that you declare it.

```
let x = 2 * 4; // There is a variable named x and its value is 8
```

# Declare -> Initialize -> Use

## And when do we do these things?

- Use the variable *after* you've declared and initialized it
- If you use it before it is declared, you'll get errors
- If you use it before it is initialized, it will have a default value that is probably not what you want.

```
circle(x, 10, 20); // draws a circle at coordinate (x, 10)
```

- In this case, x is 8, so we put the circle at (8, 10) with a radius of 20.

# Assignment and Equality

## = and == and === oh myyyy

- = is the assignment variable. It is used to put a value into a variable.
- Example:

```
myRadius = mouseX + mouseY;
```

- It evaluates the stuff on the right (the ‘r-val’) first, and *then* stuffs it into a variable called myRadius. (Note that it must have already been declared!)

# Assignment and Equality

## = and == and === oh myyyy

- == (double equals) is used to determine if two things are kinda equivalent
  - For now: don’t use double equals. We’ll get back to this subject.
- === (triple equals) is used to test if two things are the same. Use this.
- Example: if (mouseX === 0) { /* do something */ }
- Testing a condition like this is called a boolean test
  - ‘Boolean’ means ‘binary’ - true or false, named after George Boole

# Extra Canvas!

## As seen on Coding Train

- You can create a second (and third, fourth, etc) graphics context that you can draw on:
  - let nightSky = createGraphics(w, h);
  - nightSky.circle(10, 10, 4); // draw tiny star on the nightSky buffer
- Then render that image onto your regular canvas at a point of your choosing:
  - image(nightSky, 0, 0;

# What’s the dot about

## nightSky.circle (??)

- This example uses something that is common in many languages:
- Dot notation like *nightSky.circle* - this means “reach into the *nightSky* symbol, and pull out something named *circle*.”
- In this case, *circle* is a function, so we can invoke it like any other function!
- The neat thing is that “members” of an object can have their own internal state that is separate from everything else.
- We’ll cover objects late in the course

# Animation and Interaction

## Make it move

- Various ways to make things move:
  - Every time you draw something, update a variable to change its geometry
  - Or, use the frameCount variable to determine its geometry
- You can get fancy with frameCount if you want to do things in a cyclic way.
  - let spot = frameCount % width;
  - This uses the *modulo* operator

# Modulo what?

## Remember long division?

- What's 18 divided by 4? A fifth grader would probably tell you: 4, remainder 2.
- In code, *integer division like 18 / 4 actually equals 4*
- We can use the modulo operator to get at the remainder.
- It is the percent sign, and looks like this:
- 18 % 4
- So if we do x = 18 % 4, then x will then have the value 2, because the remainder is 1.

# Animation and Interaction

## Make it move

- Back to this:
- let spot = frameCount % width;
- Here the frameCount increments 60 times per second, so it could be a big number like 273482.
- And *width* is the width of your canvas (at the moment), something like 600.
- So 273482 % 600 will give results between 0 and 599 inclusive.
- In this case, the result is 482.

# Animation and Interaction

## Make it listen to your whims

- Mouse and Keyboard events, for now
- Three main ways of handling input:
  - 1. Checking built-in variables that processing maintains for you. Things like:
    - mouseX, mouseY, mouseIsPressed, mouseButton
  - 2. Checking state by calling built-in functions to see what's up:

```
if (keyIsDown(DOWN_ARROW) === true) { handY += 1; }
```

  - 3. Writing (overriding, actually) a function that receives events. Use this to capture specific mouse or keyboard events.

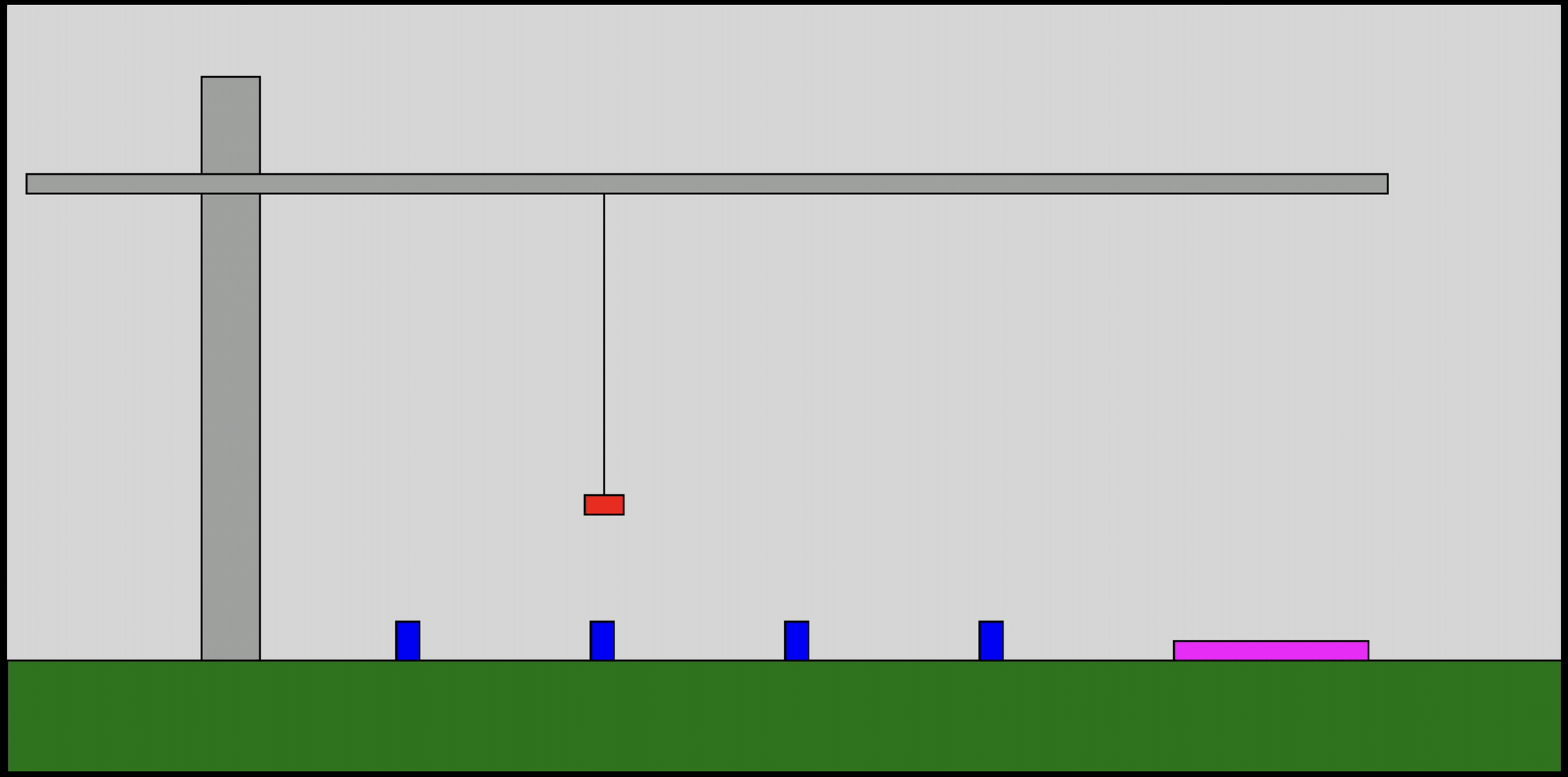
# Animation and Interaction

## Make it listen to your whims

- In the crane game, we'll use the keyboard arrow keys to move the little 'hand' part up and down, left and right.
- When we get a specific event, we can adjust variables. This one switches *multimode* from true to false (and back to true, etc) when the 'm' key is pressed.

```
function keyPressed() {

  if (key == "m") {

    multimode = !multimode;

  }

}
```

Now let’s make as much of this game as we can for the rest of class



https://editor.p5js.org/johnsogg/sketches/J9VHe2u_a

# Misc Things To Mention


WHAMMY
HARMONY
WHAMMY
CLASSIC
CHORDS
SHALLOW
DETUNE
DEEP
DIVE BOMB
Bass Whammy
CHECK
MODE
OUTPUT
INPUT
DIRECT OUT
Digital Delay
DD-3
BOSS
Flanger
BF-2
GUITARX
BLEND
LEVEL
DRIVE
LOW
HI MIDS
OUTPUT
INPUT
GROUND LIFT
DIRECT OUTPUT
PARALLEL OUTPUT
MICROTUBES B7K
Analog Bass PreAmp

# Canvas Size

- You can use windowWidth, windowHeight to get the size of the web page

```
// do this is setup()
createCanvas(windowWidth, windowHeight);
```

- You can override the windowResized function to respond to size changes

```
function windowResized() {
  resizeCanvas(windowWidth, windowHeight);
}
```

# Set Frame Rate

## Might help with debugging

- If you use console.log to output text to the console it will fly by super fast
- You can always press the Stop button
- Alternately, you can slow it way down by turning the frame rate down:
  - Do this in the setup() function
  - frameRate(1)
  - That will only draw one time per second.