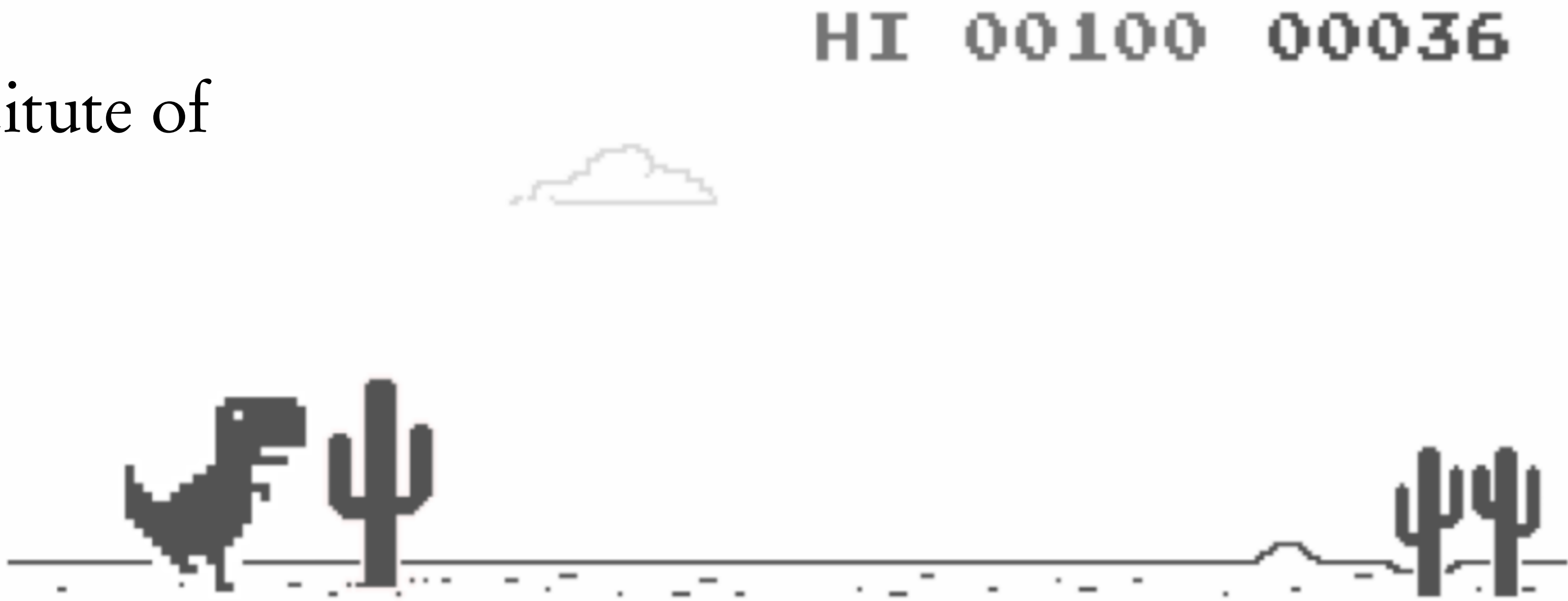
HackWITus Lecture
Chrome Dinosaur Game: p5.js Clone

Dr. Micah Schuster

Fall 2019

Wentworth Institute of Technology



First: Tools

- **Any IDE You Like:** I'll be using VSCode, but you can use any IDE you like that can handle JavaScript.
- **Chrome Web Browser:** We'll use Chrome to test our game.
- **Web Server for Chrome App:** Allows us to create a simple web server to avoid security issues when loading graphics.
    - **https://chrome.google.com/webstore/detail/web-server-for-chrome/ofhbbkphhbklhfoeikjpcbhemlocgigb?hl=en**
- **Starter Code (and Finished Code):**
    - https://github.com/mdschuster/chickenrun

Second: Setup

- **Chrome:** Go to chrome://apps after installing the web server for chrome extension.
    - This should set up the web server, as long as the app window is open, the server is running.
    - Select the folder were you downloaded the git repo
    - Go to the listed URL: http://127.0.0.1:8887
    - The javascript console can be viewed via View->Developer->JavaScript Console
- **Write Code:** Use your editor to follow along and edit the javascript files as we go though the lecture.

Quick Intro to JavaScript

- **Variable Creation:** Just Do It

```
c = new Chicken();
let value = 0;
```

No Type!

- **Control Flow:** Same as Other Languages

```
for(let i = 0; i < array.length; i++){
  //stuff
}
```

```
if(currentTime < 0){
  //stuff
}
```

WENTWORTH INSTITUTE OF TECHNOLOGY

Quick Intro to JavaScript

- Functions (Outside of Classes):

```
function keyPressed(){
  //stuff
}
```

No Return Type, but you can still return a value

- Classes (ES6+):

```
class Chicken{
  constructor(){
    this.x = 0;
  }

  jump(){
    //stuff
  }
}
```

Constructor:
Just `constructor()`

`this` is everywhere!

Functions start with the name only

p5.js

- Helps us easily use graphics in JavaScript.
- Includes ways to create a canvas and update the canvas at regular intervals (a time step).
- Can draw basic shapes and display images to the canvas.
- Free and open source.

p5.js

`www.p5js.org`

See **`index.html`** for the line that imports **`p5js`**

WENTWORTH INSTITUTE OF TECHNOLOGY

Chicken Constructor

- Contains a few important variables:
  - x and y position
  - y velocity
  - gravity
  - size of sprite and size of collision object
  - counter for animation

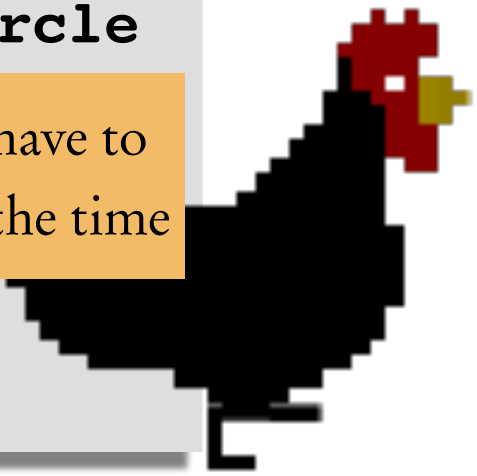
Declaring the variable with **`this`** creates the member variable in the class.

We'll start with three of these (x and y position, size of bounding circle) and add the others as we need more functionality

Chicken Constructor

```
class Chicken{
  constructor(){
    this.d = 100; //diameter of bounding circle
    this.r = this.d/2;
    this.x = 50 + this.r;
    this.y = height - this.r;
  }
}
```

Radius, so we don't have to write **`this.d/2`** all the time



Canvas x axis:

Left = **`0`**

Right = **`width`**

Canvas y axis:

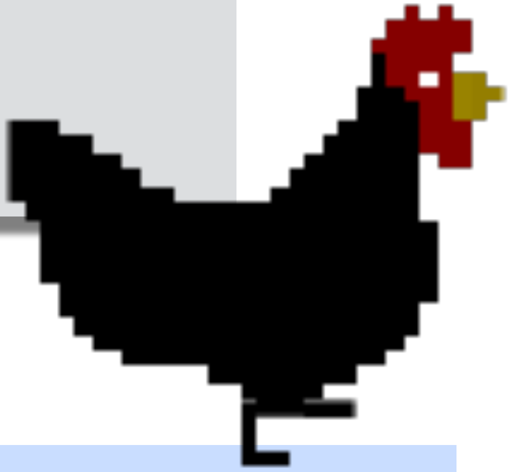
Top = **`0`**

Bottom = **`height`**

Why the shift in **`x`** and **`y`** (**`50`** or **`height`**) in the above code?

Draw a Circle (Still in the Chicken Class)

```
show(){
  fill(255);
  ellipseMode(CENTER);
  ellipse(this.x, this.y, this.d, this.d);
}
```



`Fill:`
Sets color of future fill.
`(255=white)`

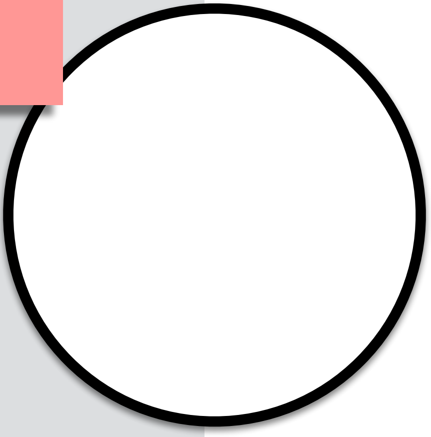
`EllipseMode:`
`Center` defines the ellipse location at the center of the shape

`Ellipse:`
Draws ellipse at x,y with height/ width = **`this.d`**

Now, Draw to the Screen

```
function setup(){
  c = new Chicken();
}

function draw(){
  background(255);
  strokeWeight(2);
  c.show();
}
```

I've already started these functions for you in **`sketch.js`**



`strokeWeight` sets the thickness of the lines

`draw()` runs every time the screen refreshes (every "frame")

Although this only draws the circle for now, it will eventually represent the collision zone.

WENTWORTH INSTITUTE OF TECHNOLOGY

Jump!

```
function jump(){
  if(this.y == height - this.r){
    this.vy = -35;
  }
}
```

Back to the Chicken class

There are two pieces to **`jump()`**:

- Check if we're on the ground
- Increase y velocity (negative is up!)

Currently, the y velocity does nothing. So we need another function that handles the actual motion.

Physics Lesson

$$\frac{dx}{dt} = v$$

A change in position over time is velocity

$$\frac{dv}{dt} = a$$

A change in velocity over time is acceleration

Since the chicken only jumps, the acceleration is gravity!

Physics Lesson

Using the definition of the derivative:

$$\frac{df(t)}{dt} = \frac{f(t+dt) - f(t)}{dt}$$

Function at the next time

Function at the current time

So,

$$\frac{dx}{dt} = v$$

$$\frac{x(t+dt) - x(t)}{dt} = v$$

$$x(t+dt) = v \cdot dt + x(t)$$

What is *dt*?

WENTWORTH INSTITUTE OF TECHNOLOGY

Physics Lesson

What is *dt*?

Change in time between frames

$\mathcal{O}(16ms)$

Change in Position

$$x(t + dt) = v \cdot dt + x(t)$$

Change in Velocity

$$v(t + dt) = a \cdot dt + v(t)$$

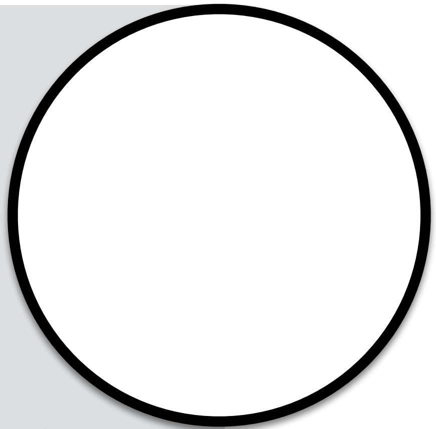
Ultimately, all we need to know is the previous position/velocity!

To simplify our equations, we're just going to supply a constant for these values.

This effectively packages the value of **`dt`** into a number that feels good in the game.

Back to the Chicken!

```
move(){
  //updates position based on velocity
  this.y+=this.vy;
  //updates velocity based on gravity
  this.vy+=this.gravity;
  //constrains y between 0 and height-radius
  this.y=constrain(this.y,0,height-this.r);
}
```



`this.gravity` is a fixed value set in the constructor: **`this.gravity=-35`**

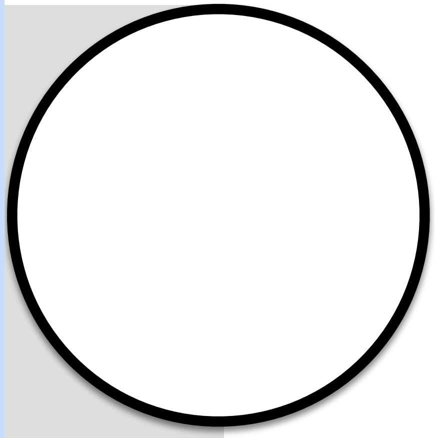
`this.vy` is set when the player jumps (see the Jump! slide)

`constrain` is a p5js function that keeps **`this.y`** between the final two function arguments

Jumping Chicken (back to **`sketch.js`**)

```
function draw(){
  background(255);
  strokeWeight(2);
  c.move();
  c.show();
}

function keyPressed(){
  if(key==' '){
    c.jump();
  }
}
```

Add **`c.move()`** to the draw function (updated every frame)



`keyPressed()` is a built-in p5js function to detect keyboard presses.

When the user presses space, we apply a y velocity to the chicken

Something to Jump Over (**Egg**)

```
class Egg{
  constructor(){
    this.d=40;
    this.r=this.d/2;
    this.x=width+this.d;
    this.y=height-this.r;
  }
  move(){
    this.x-=16;
  }
  show(){
    fill(255);
    ellipseMode(CENTER);
    ellipse(this.x, this.y, this.d, this.d);
  }
}
```

Setup

This is the full **Egg** class (without sprite graphics)

It has the same structure as **Chicken**, but simpler. It only has to move in the negative x direction at a constant rate (-16)

WENTWORTH INSTITUTE OF TECHNOLOGY

Something to Jump Over (**`Egg`**)

```
class Egg{
  constructor(){
    this.d=40;
    this.r=this.d/2;
    this.x=width+this.d;
    this.y=height-this.r;
  }
  move(){
    this.x-=16;
  }
  show(){
    fill(255);
    ellipseMode(CENTER);
    ellipse(this.x, this.y, this.d, this.d);
  }
}
```

Simple Movement

This is the full **`Egg`** class (without sprite graphics)

It has the same structure as **`Chicken`**, but simpler. It only has to move in the negative x direction at a constant rate (-16)

WENTWORTH INSTITUTE OF TECHNOLOGY

Something to Jump Over (**Egg**)

```
class Egg{
  constructor(){
    this.d=40;
    this.r=this.d/2;
    this.x=width+this.d;
    this.y=height-this.r;
  }
  move(){
    this.x-=16;
  }
  show(){
    fill(255);
    ellipseMode(CENTER);
    ellipse(this.x, this.y, this.d, this.d);
  }
}
```

This is the full **`Egg`** class (without sprite graphics)

It has the same structure as **`Chicken`**, but simpler. It only has to move in the negative x direction at a constant rate (-16)

Draw Ellipse

Lay Some Eggs in `sketch.js`

```
let eggs=[];

function spawnEgg(){
   eggs.push(new Egg());
}
```

Create an array (outside of the function) and add to the array (**`push`**) in the **`spawnEgg()`** function.

```
//—inside of Draw()—

//spawn based on random number
if(random(1)<0.005){
   spawnEgg();
}
```

Call **`spawnEgg()`** when the random number is small (called every frame).

```
for(let i=0; i<eggs.length; i++){
   e=eggs[i];
   e.move;
   e.show;
}
```

Loop through **`eggs`** array, **`move()`** and **`show()`** each egg.

WENTWORTH
INSTITUTE OF TECHNOLOGY

Where are we?

At this point, we almost have a working game:

- Interactable player character
- Obstacles that the player must avoid

What we still need:

- Collision
- Actual graphics

Tweakable:

- Egg spawning
- Game “feel” (adjusting constants)

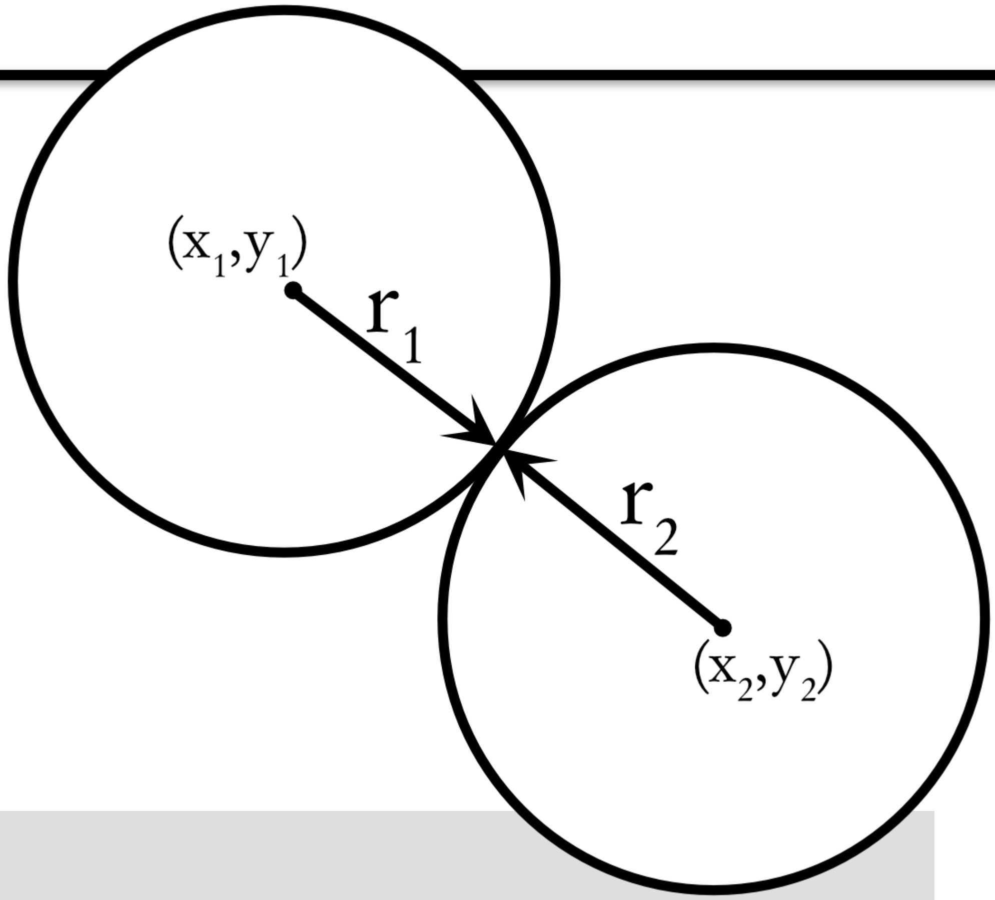
Ultimately, tweak the game to make it play the way you want. Dev studios spend years before and after release tweaking their game.

WENTWORTH INSTITUTE OF TECHNOLOGY

Colliding Circles

What we need to know:

- Center of both circles
- Radii of both circles
- Distance between the centers of both circles



```
hits(egg){
  let distance=dist(this.x,this.y,egg.x,egg.y);
  if(distance<(this.r+egg.r){
    return true;
  } else {
    return false;
  }
}
```

This is in the **`Chicken`** class

When Chicken hits Egg

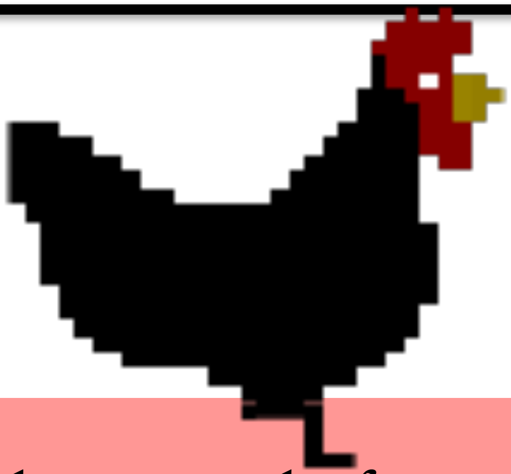
```
//–inside of Draw()–
let hit=false;
for(let i=0; i<eggs.length; i++){
   e=eggs[i];
   e.move;
   e.show;
   if(c.hits(e)){
      hit=true;
   }
}

if(hit==true){
   console.log("Game Over");
   noLoop();
}
```

We need to check for a collision every frame after the chicken and egg move.

If there's a hit, stop the game.

Load Simple Graphics (in `sketch.js`)

I've included some simple sprites (made by me) for you to use.



preload() happens before start and is generally used to load external resources.

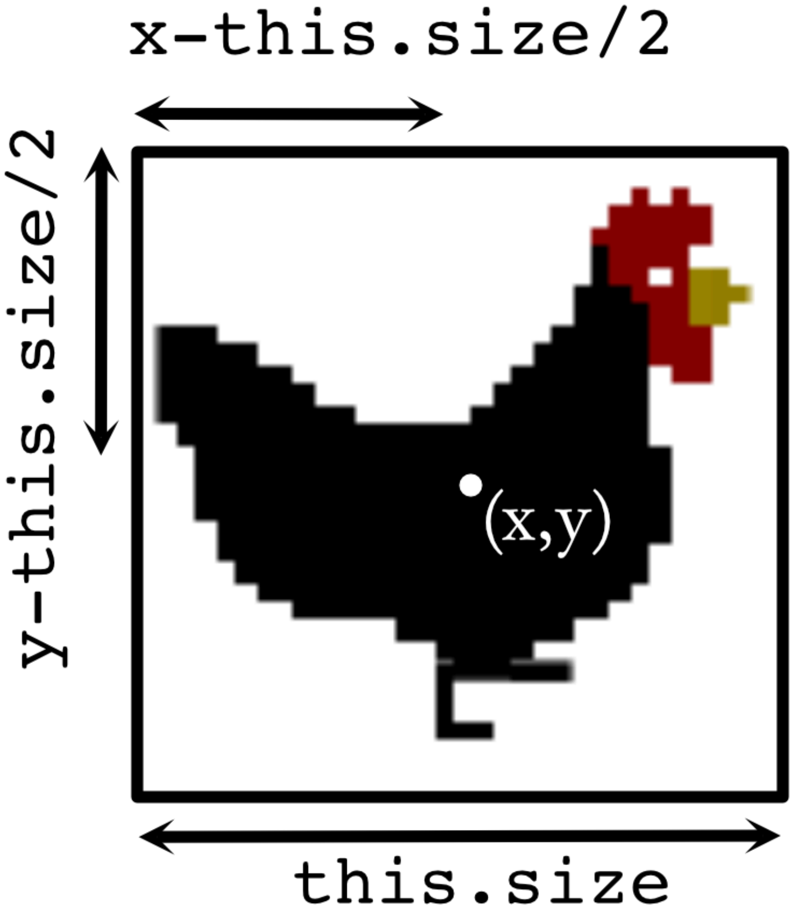
```
let C1image;
let C2image;
let Eimage;
function preload(){
  C1image=loadImage("graphics/chicken1.png");
  C2image=loadImage("graphics/chicken2.png");
  Eimage=loadImage("graphics/egg.png");
}
```

WENTWORTH INSTITUTE OF TECHNOLOGY

Load Simple Graphics (in `sketch.js`)

`p5js` has a simple function that can display an image.

```
//assuming a square image
image(sprite,x,y,size,size);
```



But, it's not that easy…
`x` an **`y`** represent the upper right corner of the image

So, we have to convert from our center based system to the upper right.

WENTWORTH INSTITUTE OF TECHNOLOGY

Display the `Egg`

Showing a single image is easy

But, we need a few new variables

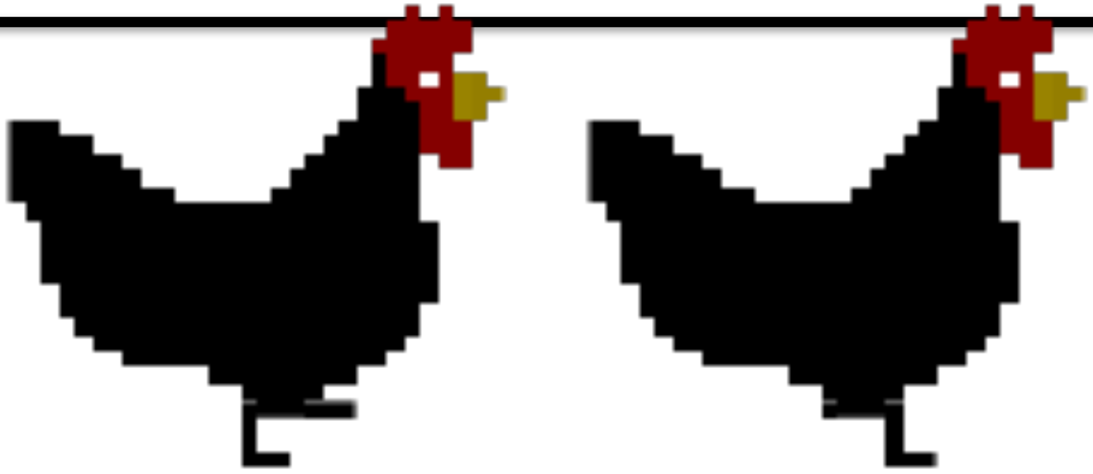
```
this.size=this.d+20;
this.imx=this.x-this.size/2;
this.imy=this.y-this.size/2;
```

`size` is how big we want the sprite on the screen (a little bit bigger than our collision circle)

`imx` and **`imy`** are the upper right corner of the sprite location.

```
//Egg class in show()
image(Eimage,this.imx,this.imy,this.size,this.size);
```

Chicken Animation



We have two chicken images so that we can have a walking animation.

Our simple animation will swap the image that is displayed every 10 frames.

imx, **imy**, and **size** are needed as well.

```
//chicken class in show()
if(this.counter<=10){
  image(C1image,this.imx,this.imy,this.size,this.size)
} else {
  image(C2image,this.imx,this.imy,this.size,this.size)
}
counter++;
if(this.counter==20) this.counter=1;
```

Create a **counter** field for the class and set it to 1

It works!

We could stop here and feel good about our game, tweaking the parameters until we're ready to share it.

After player for a while though, the frame rate will drop…
Why?

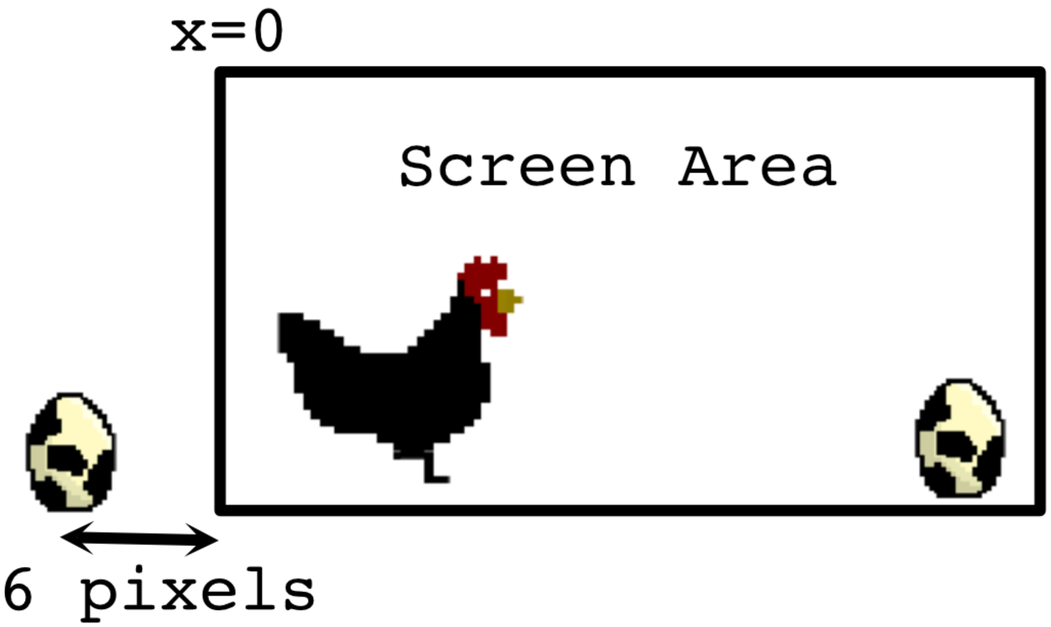
Every **`Egg`** we created still exists and is moving in the negative x direction.

We then check every egg, every frame, to see if it hits the chicken!

Egg Clean Up

If the egg leaves the screen area (**x<0**) then we remove it from the **eggs** array.

The garbage collector takes care of the rest.



```
for(let i=0; i<eggs.length; i++){
  //other egg stuff here...
  if(e.x < -16){
    count++;
  }
}
for(let i=0; i<count; i++)
  eggs.shift();
```

array.shift() removes elements from the beginning of the array.

These will be the oldest, which should be removed first.

The Sky is the Limit

You now have the basic layout of the game.

For the next steps:

- Add background graphics (clouds, trees, etc.) that move at different speeds to simulate parallax.
- Add different spawn patterns (see my final version) based on a random number generator.
- Adjust the gravity, jump velocity, and egg speed so that the game feels good to you.
- Add a score counter and death screen.