Web Animation

Week 3

Plan of Action

- WEEK1
  - Intro to animation and canvas: Setup our constructors and other game functions
- WEEK2
  - Set up our Bird character and gravity
- WEEK3
  - Obstacles and Collision
- WEEK4
  - Score and End Screen
- WEEK5
  - Wrap up

What is an object

- Object Oriented Programming (OOP) refers to using self-contained pieces of code to develop applications.
- Objects usually represent things and can contains a collections of values
- We use objects as building blocks for our applications.

Var vs object

What are the benefits of using OOP?

- Building applications with objects allows us to reuse code and adopt some valuable techniques like:
  - **Inheritance** = objects can inherit features from other objects
  - **Encapsulation** = each object is responsible for specific tasks

Object Literal vs Constructor

- The Object Literal and Constructor function methods are similar, but fundamentally different.
- An Object Literal creates an object that can be immediately used without first having to use the new keyword.
- However, an object literal cannot implement the basic OOP principles of encapsulation and inheritance.
- By using a constructor function, we are able to implement object-oriented design in JavaScript. A constructor function is essentially a class.

```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue",
    sayHello: function(){
        alert("Hello");
    }
};
```

How do we access the object values/methods?

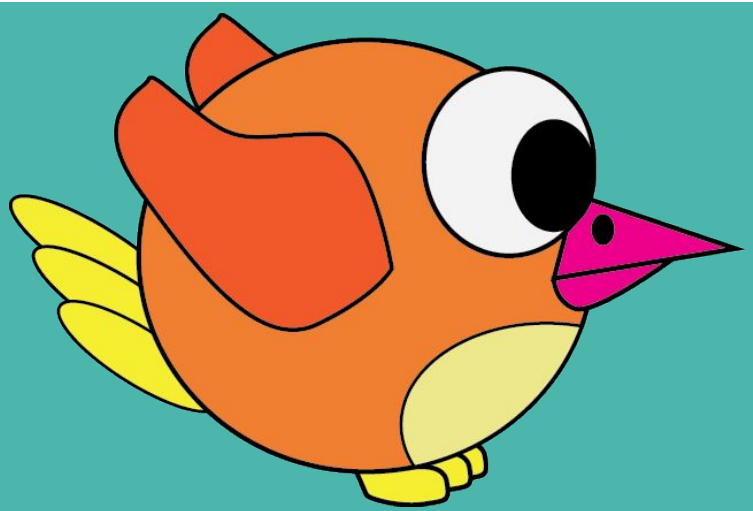
Syntax:

```
objectName["propertyName"]
objectName.propertyName
objectName.methodName()
```

Our person object example:

```
person.firstName
person.["firstName"]
person.sayHello()
```

Where did we use our Component in last weeks code?



FlyingCharacter Object

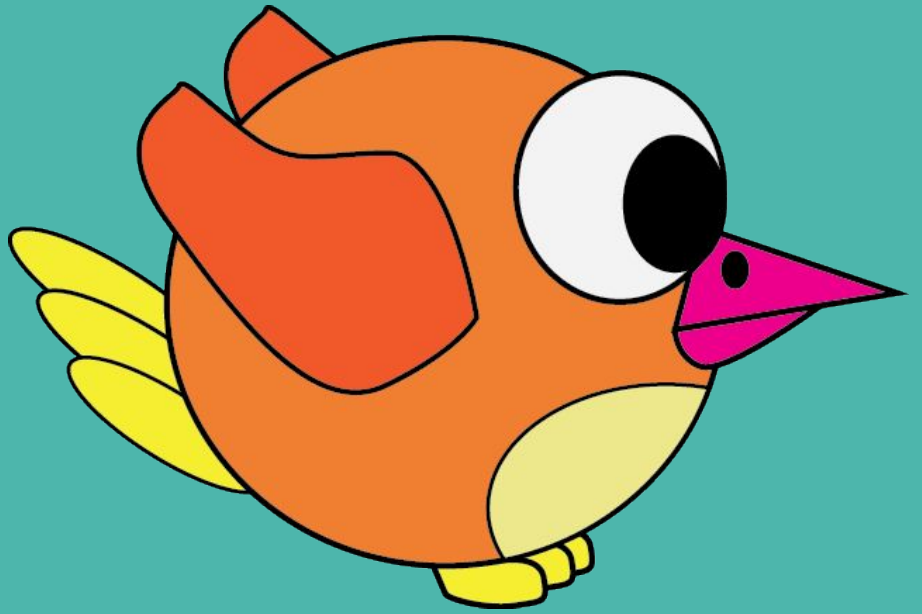
- Our character is created using the constructor we wrote
- We gave it a width, height, look, x, y, and type as arguments

```
flyingCharacter = new component(50, 30, "images/bird2.png", 10, 120, "image");
```

New Methods for our Flying Character

What kinds of methods does the character use now?

Hint: There's 3



Flip Method

- Changes how our character looks
- Looks different when it's flying or falling
- Note how we write the function: `this.flip`

```
this.flip = function(key){
    if(key == "up") { flyingCharacter.image.src = "images/bird1.png";}
    else if(key == "down") { flyingCharacter.image.src = "images/bird2.png";}
}
```

New Position Method

- Changes the position of our character
- Increases how fast our character is flying or falling
- Calls the hitBottom() function
  - What's this for?

```
this.newPos = function() {
    this.gravitySpeed += this.gravity;
    this.y += this.gravitySpeed;
    this.hitBottom();
}
```

```
function updateGameArea() {

    myGameArea.clear();
    myGameArea.frameNo += 1;
    myBackGround.update();
    flyingCharacter.newPos();
    flyingCharacter.update();

}
```

Hit Bottom Method

- Checks if our character has hit the bottom of the canvas
- If it gets there, stops our character from falling outside the canvas

```
this.hitBottom = function() {
    var rockbottom = myGameArea.canvas.height - this.height;
    if (this.y > rockbottom) {
        this.y = rockbottom;
        this.gravitySpeed = 0;
    }
}
```

Button

- We created a button in our index.html
- This button flipped the bird
- And changed the gravity of the bird

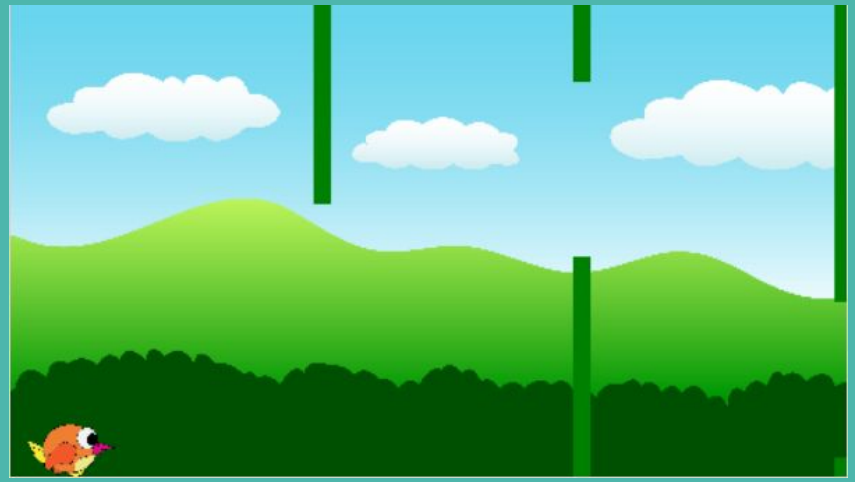
```
<button onmousedown="accelerate(-0.2); flyingCharacter.flip('up');"
onmouseup="accelerate(0.05); flyingCharacter.flip('down');">FLY BIRDY</button>
```

This week

Create Obstacles at a set time

Update

Check for collision



When do we want the obstacles to be drawn?

Need to draw an obstacle every 3 seconds

To do this we will use the frameNo variable which counts every frame

FPS = 50

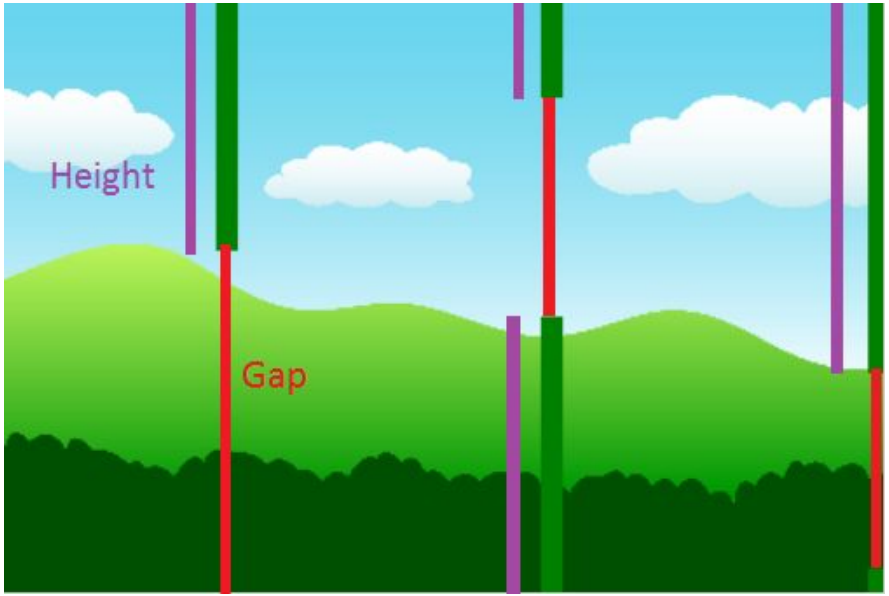
So if frameNo = 150 it has been 3 secs

```
function everyinterval(n) {
    // draw every n secs
    if ((myGameArea.frameNo / n) % 1 == 0) {
        console.log("make obstacle");
        return true
        ;}
    return false;
}
```

How to make obstacles random heights?

We will create some variables for the height and gap

We also want the min max height and gap so our obstacles aren't all the same




```
var myObstacles = [];
```

```
var x, height, gap, minHeight, maxHeight, minGap, maxGap;
```

Obstacle function

We will draw 2 obstacles each time: 1 at the top and 1 at the bottom

A box is drawn from the upper left corner, so we will draw second obstacle at a lower y position

```
// making the obstacles ========================================
 if (myGameArea.frameNo == 1 || everyinterval(150)) {
    x = myGameArea.canvas.width;
    minHeight = 20;
    maxHeight = 200;
    height = Math.floor(Math.random()*(maxHeight-minHeight+1)+minHeight);
    minGap = 50;
    maxGap = 200;
    gap = Math.floor(Math.random()*(maxGap-minGap+1)+minGap);
    myObstacles.push(new component(10, height, "green", x, 0));
    myObstacles.push(new component(10, x - height - gap, "green", x, height + gap));
}
```

JavaScript Functions

push() -> adds a new item to the end of an array

Math.random() -> returns a value between 0 and 1

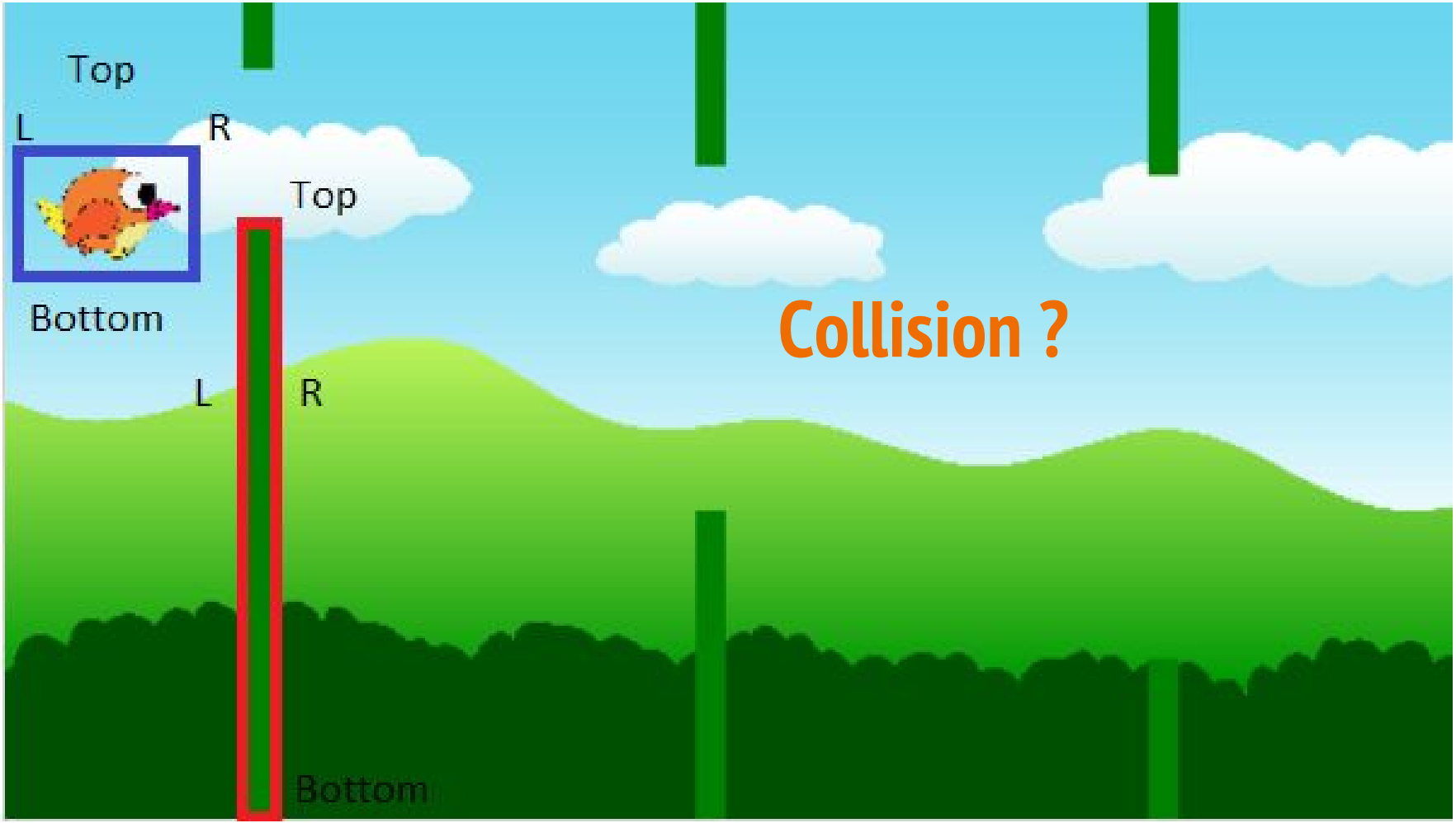
Math.floor() -> rounds down to the nearest int

How to update an array?

Remember how we update the other components

We will the same update method but we must traverse through the array to update each obstacle

```
for (i = 0; i < myObstacles.length; i += 1) {
    myObstacles[i].x += -1;
    myObstacles[i].update();
}
```


Top
L
R
Top
Bottom
L
R
Bottom
Collision ?

Collision

Create vars for all sides of the 2 objects we want to check are colliding

There are lots of instances where the 2 objects can be colliding

Hence we will only check for instances we are certain they are not colliding

```
//make collision function
this.crashWith = function(otherObj){
    // setup some vars to reference sides to obj1
    var myleft = this.x;
    var myright = this.x + this.width;
    var mytop = this.y;
    var mybottom = this.y + this.height;
    // setup some vars to reference sides to obj2
    var otherleft = otherObj.x;
    var otherright = otherObj.x + otherObj.width;
    var othertop = otherObj.y;
    var otherbottom = otherObj.y + otherObj.height;
    // set crash to be true by default
    var crash = true;
    // check to see if its not colliding and set to be false
    if((mybottom < othertop) || (mytop > otherbottom)
        || (myright < otherleft) || (myleft >otherright)){
        crash = false;
    }
    return crash;

}
```