Boomshine ICE B

Overview
So you've got one circle bouncing around - how about 2 or 10 or 60? (like we'll eventually need with Boomshine). The approach we're using so far (of hard-coding circle variables) is not great, so let's fix it.

Part I - Getting Started

1) Make a copy of the completed Boomshine folder from last time, and name it **Boomshine-B**

2) Delete the code you added in *Boomshine-A ICE*, Part II and Part III

3) Comment out the *Boomshine-A ICE*, Part IV code (don't delete it) - we'll refactor it in a little while.

4) Reload the HTML page and check the console. There should be no errors, and the FPS should still be visible.

5) Add the following as a properties of `app.main`:

```
NUM_CIRCLES_START: 5,
START_RADIUS : 8, // starting circle radius
MAX_SPEED : 80, // pixels-per-second
CIRCLE_STATE: { // fake enumeration, actually an object literal
    NORMAL : 0,
    EXPLODING : 1,
    MAX_SIZE : 2,
    IMPLODING : 3,
    DONE : 4
},
```

These are handy "constants" that our app will need.

Reference: https://stijndewitt.wordpress.com/2014/01/26/enums-in-javascript/

6) Add the following as a properties of `app.main`:

```
circles : [],
numCircles: this.NUM_CIRCLES_START,
```

The `[]` syntax declares a new array.

You also could have typed `circles : new Array(),`

Part II - Creating Circles

```
makeCircles: function(num){
    var array = [];
    debugger;
    for(var i=0; i<num;i++){
        // make a new object literal
        var c = {};

        // add .x and .y properties
        // .x and .y are somewhere on the canvas, with a minimum margin of START_RADIUS
        // getRandom() is from utilities.js
        c.x = getRandom(this.START_RADIUS * 2, this.WIDTH - this.START_RADIUS * 2);
        c.y = getRandom(this.START_RADIUS * 2, this.HEIGHT - this.START_RADIUS * 2);

        // add a radius property
        c.radius = this.START_RADIUS;

        // getRandomUnitVector() is from utilities.js
        var randomVector = getRandomUnitVector();
        c.xSpeed = randomVector.x;
        c.ySpeed = randomVector.y;

        // make more properties
        c.speed = this.MAX_SPEED;
        c.fillStyle = getRandomColor();
        c.state = this.CIRCLE_STATE.NORMAL;
        c.lifetime = 0;

        // no more properties can be added!
        Object.seal(c);
        array.push(c);
    }
    return array;
}
```

1) We're going to add a helper method to `app.main` to make circles for us - here it is:

Note our technique of creating an empty JS object literal, and then adding properties to it.

`Object.seal()` prevents any more properties from being added to each object - this prevents really hard to track down errors. We'll demo this in class.

Reference: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/seal

2) Add the following to `app.main.init()`, right before the call to `this.update()`:

```
this.numCircles = this.NUM_CIRCLES_START;
this.circles = this.makeCircles(this.numCircles);
console.log("this.circles = " + this.circles);
```

3) Reload the page and check the console, you should see the circle array log out 5 circles.

```
window.onload called
app.main.init() called
this.circles = [object Object],[object Object],[object Object],[object Object],[object Object]
```

Discussion: In this section, we learned how to create a JS Object literal to hold circle state and behavior, and to then add properties to it.

This is one way, but not the only way to do OOP, in JavaScript.

Part III - Drawing and moving our circles

1) To draw our circles on the screen, we're going to go ahead and create a `draw()` method for the circles to use. Add the following to the top of `app.main.makeCircles()`:

```
// a function that we will soon use as a "method"
var circleDraw = function(ctx){
    // draw circle
    ctx.save();
    ctx.beginPath();
    ctx.arc(this.x, this.y, this.radius, 0, Math.PI*2, false);
    ctx.closePath();
    ctx.fillStyle = this.fillStyle;
    ctx.fill();
    ctx.restore();
};
```

2) Add the following to the loop in `app.main.makeCircles()`, before we `Object.seal()` the `c` object:

```
c.draw = circleDraw;
```

Now our circle objects know how to draw themselves!

3) Now add a `drawCircles()` method to `app.main`:

```
drawCircles: function(ctx){
    for(var i=0;i<this.circles.length; i++){
        var c = this.circles[i];
        c.draw(ctx);
    }
},
```

4) Call it from `app.main.update()`

```
// ii) draw circles
this.drawCircles(this.ctx);
```

5) Reload the page, you should see the 5 circles of random colors, placed at random positions around the screen:

6) Now let's teach the circles how to move. Add the following to the top of `app.main.makeCircles()`:

```
// a function that we will soon use as a "method"
var circleMove = function(dt){
    this.x += this.xSpeed * this.speed * dt;
    this.y += this.ySpeed * this.speed * dt;
};
```

7) Add the following to the loop in `app.main.makeCircles()`, before we `Object.seal()` the `c` object:

```
c.move = circleMove;
```

8) Now add a `moveCircles()` method to `app.main`:

```
moveCircles: function(dt){
    for(var i=0;i<this.circles.length; i++){
        var c = this.circles[i];
        c.move(dt);
    }
},
```

9) Call it from `app.main.update()`

```
// 4) UPDATE
// move circles
this.moveCircles(dt);
```

10) Reload the page - you should 5 circles moving across on the screen. If not, check the console for syntax errors. You can also insert break points with the `debugger;` statement and then check the values of certain variables while your code is running.

11) To get the circles to bounce, make `app.main.moveCircles` look like this:

```
moveCircles: function(dt){
    for(var i=0;i<this.circles.length; i++){
        var c = this.circles[i];
        c.move(dt);

        // did circles leave screen?
        if(this.circleHitLeftRight(c)) c.xSpeed *= -1;
        if(this.circleHitTopBottom(c)) c.ySpeed *= -1;
    }
},
```

12) Uncomment these methods of `app.main` and make them look like this:

```
circleHitLeftRight: function (c){
    if (c.x <= c.radius || c.x >= this.WIDTH - c.radius){
        return true;
    }
},

circleHitTopBottom: function (c){
    if (c.y < c.radius || c.y > this.HEIGHT - c.radius){
        return true;
    }
}
```

13) Reload the page - the circles should now bounce.

Part IV - Creating a pause screen using *window·blur* and *window·focus*

Now we'll create a pausing screen that pauses the app whenever the user puts this window in the background, or creates a new browser tab.

1) Add the following to the top of `app.main`:

```
paused: true,
```

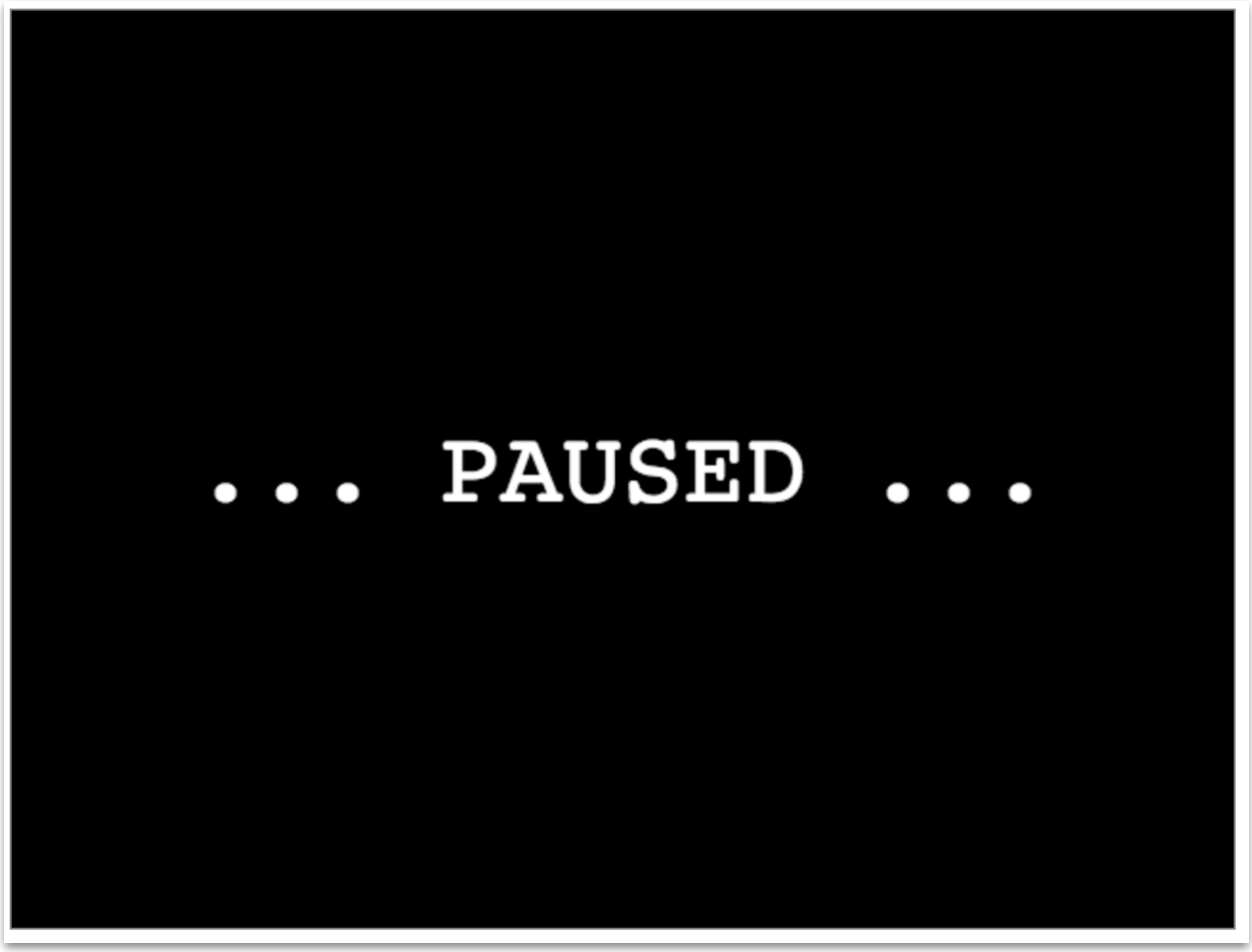
2) Add a `drawPauseScreen()` method to `app.main`:

```
drawPauseScreen: function(ctx){
    ctx.save();
    ctx.fillStyle = "black";
    ctx.fillRect(0,0,this.WIDTH,this.HEIGHT);
    ctx.textAlign = "center";
    ctx.textBaseline = "middle";
    this.fillText("... PAUSED ...", this.WIDTH/2, this.HEIGHT/2, "40pt courier", "white");
    ctx.restore();
}
```

3) In app.main.*`update()`*, add the following:

```
// 2) PAUSED?
// if so, bail out of loop
if (this.paused){
    this.drawPauseScreen(this.ctx);
    return;
}
```

4) Reload the page, you should see the the following:



4) To get pausing/resuming to work:

A) change the value of `app.main.paused` to `false`

B) add the following property to `app.main`:

```
animationID: 0,
```

C) Make this line:

```
requestAnimationFrame(this.update.bind(this));
```

Look like this:

```
// 1) LOOP
// schedule a call to update()
this.animationID = requestAnimationFrame(this.update.bind(this));
```

D) add these two event handlers to **loader.js**

```
window.onblur = function(){
      console.log("blur at " + Date());
      app.main.paused = true;

      // stop the animation loop
      cancelAnimationFrame(app.main.animationID);

      // call update() once so that our paused screen gets drawn
      app.main.update();
};

window.onfocus = function(){
      console.log("focus at " + Date());

      // stop the animation loop, just in case it's running
      cancelAnimationFrame(app.main.animationID);

      app.main.paused = false;

      // restart the loop
      app.main.update();
};
```

The app should now start in a running state, and then pause when the window loses focus.

`window.onblur` is fired when the window loses focus. You can test this by creating a new browser window and putting it in front of the window you are running the app in.

`window.onfocus` is fired when the window returns to the foreground.

5) Break this app!

A) Comment out the `cancelAnimationFrame()` calls and then pause/unpause the app several times. What happens? Why do you think this is happening.

Now uncomment the `cancelAnimationFrame()` calls to fix this.

B) Create a new property on your circles, accidentally:

i) Comment out this line in `app.main.makeCircles()`:

```
Object.seal(c);
```

ii) Change this line:

```
if(this.circleHitLeftRight(c)) c.xSpeed *= -1;
```

to this:

```
if(this.circleHitLeftRight(c)) c.xspeed *= -1; // lower case s
```

- Reload the page - right and left bouncing now fails!

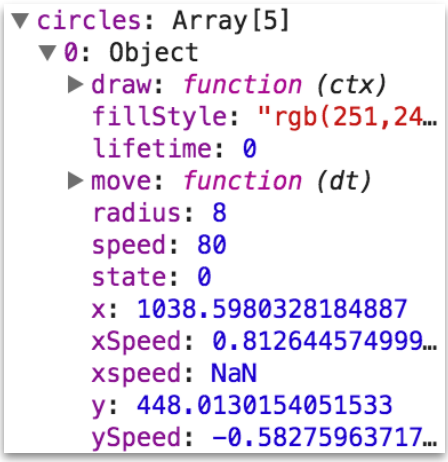
- Check the console - there are no errors!

- What happened? Your spelling mistake created a new property named `xspeed`

`this.xspeed` has an `undefined` value

- In the above code, when you multiply `this.xspeed` (which is `undefined`) by -1, you get an `undefined` result.

You can see the new `xspeed` property and its `NaN` value in the debugger screenshot ->



This ability to create properties on the fly leads to problems if you're not careful.

Go ahead and uncomment `Object.seal(c);` - now you should get a helpful error message in the console:

```
Uncaught TypeError: Can't add property xspeed, object is not extensible
```

iii) Go ahead and fix the code so it works properly.

Part V - Submission

That's enough for now - we'll build on this next time.

Boomshine B Rubric

DESCRIPTION	SCORE	VALUE %
Circles Draw – Many circles drawn correctly to the screen.		**20**
Circles Move – Circles all move around the screen correctly.		**20**
Circles Bounce – All of the circles bound correctly.		**20**
Pause State – Correctly pauses with onblur and cancelAnimationframe.		**20**
Resume State – Correctly resumes from pause with onfocus and does not break/speed up the animation.		**20**
Errors Thrown – Any errors thrown in the console.		**-20% (this time)**
Previous Work – Deductions for any parts from previous assignments that no longer work correctly. There are no set values for penalties. The more penalties you make the more points you will lose.		
Additional Penalties – These are point deductions for missing submission links, poorly written code, etc. There are no set values for penalties. The more penalties you make the more points you will lose.		
TOTAL		**100%**

ZIP the folder and post it to mycourses. **Include a link to the work on Banjo in the submission comments.**