Boomshine C

Overview:
This time we'll be adding some more properties to `app.main`, we'll be adding circle hit testing when we click in the window, and we will see how `Object.freeze()` works.

Part I - Creating Object Literals in a function

1) We're going to need quite a few constants in the next 2 parts of this exercise, so let's just get them typed in now and be done with them. These are properties of `app.main`.

Go ahead and replace `START_RADIUS : 8`, `MAX_SPEED : 80`, and `NUM_CIRCLES_START: 5` with this `CIRCLE` object:
(see **boomshine-C-codez.txt** for some copy-paste help)

```
//  properties
WIDTH : 640,
HEIGHT: 480,
CIRCLE: {
     NUM_CIRCLES_START: 5,
     NUM_CIRCLES_END : 20,
     START_RADIUS : 8,
     MAX_RADIUS : 45,
     MIN_RADIUS : 2,
     MAX_LIFETIME : 2.5,
     MAX_SPEED : 80,
     EXPLOSION_SPEED : 60,
     IMPLOSION_SPEED : 84,
},
…
```

Reload the page. You've got a logic error from renaming your properties.

i) Change this line:
```
this.numCircles = this.NUM_CIRCLES_START;
```

to this:

```
this.numCircles = this.CIRCLE.NUM_CIRCLES_START;
```

ii) and change all 5 of the references of `this.START_RADIUS`

to

```
this.CIRCLE.START_RADIUS
```

iii) and change this line:

```
c.speed = this.MAX_SPEED;
```

to this:

```
c.speed = this.CIRCLE.MAX_SPEED;
```

Reload the page - if all is good you should see the bouncing circles.

2) Here are some more properties for `app.main` we'll need soon:

```
GAME_STATE: { // another fake enumeration
     BEGIN : 0,
     DEFAULT : 1,
     EXPLODING : 2,
     ROUND_OVER : 3,
     REPEAT_LEVEL : 4,
     END : 5
},
```

Part II - Where is the mouse clicking?

Getting the correct mouse coordinates relative to the canvas takes a little work.

1) Add the following event handle to `main.init()`:

```
// hook up events
this.canvas.onmousedown = this.doMousedown;
```

2) The first implementation of `main.doMousedown` looks like this:

```
doMousedown: function(e){
	console.log("e=" + e);
	console.log("e.target=" + e.target);
	console.log("this=" + this);
}
```

3) Save and reload the page - then click on the canvas. Your logs should look like this:

```
e=[object MouseEvent]
main.js:240 e.target=[object HTMLCanvasElement]
main.js:241 this=[object HTMLCanvasElement]
```

`e` is the `Event` object
`e.target` refers to the object that sent the event (the `<canvas>` element)
here `this` also refers to the `<canvas>` element

4) Add these two lines to the end of `doMousedown`:

```
	console.log("e.pageX=" + e.pageX);
	console.log("e.pageY=" + e.pageY);
```

Then reload the page and click on the canvas as close to the upper-left corner (0,0) as you can and check the logs:

Huh? Why not *(0,0)* or *(1,1)*? The reason that we got larger numbers is that `pageX` and `pageY` are the `x` and `y` coordinates of the mouse click in the *window*, not in the *canvas*. So we need to adjust for the “margin and padding left” and “margin and padding top” of the canvas to find out where the user clicked in “canvas coordinates”.

5) Time for another helper function - `getMouse()` is already in your **utilities.js** file.

```
// returns mouse position in local coordinate system of element
function getMouse(e){
	var mouse = {} // make an object
	mouse.x = e.pageX - e.target.offsetLeft;
	mouse.y = e.pageY - e.target.offsetTop;
	return mouse;
}
```

6) Add the following to the end of `doMousedown`:

```
var mouse = getMouse(e);
console.log("(mouse.x,mouse.y)=" + mouse.x + "," + mouse.y);
```

Then reload the page and try to click at the `(0,0)` of the canvas again - then check the console - you can see that the `getMouse()` function did the coordinate adjustment for us.

Part III - Clicking Circles?

Now we’ll make the circles stop moving and turn red when they are clicked (later we’ll make them “explode”):

1) We need another helper function for this - add it to **utilities.js**:

```
// http://stackoverflow.com/questions/2212604/javascript-check-
mouse-clicked-inside-the-circle-or-polygon

// using 'distance squared' here, why?
// I is for "Instance"
function pointInsideCircle(x, y, I) {
     var dx = x - I.x;
     var dy = y - I.y;
     return dx * dx + dy * dy <= I.radius * I.radius;
}
```

2) You have to write `main.circleClicked()` now:

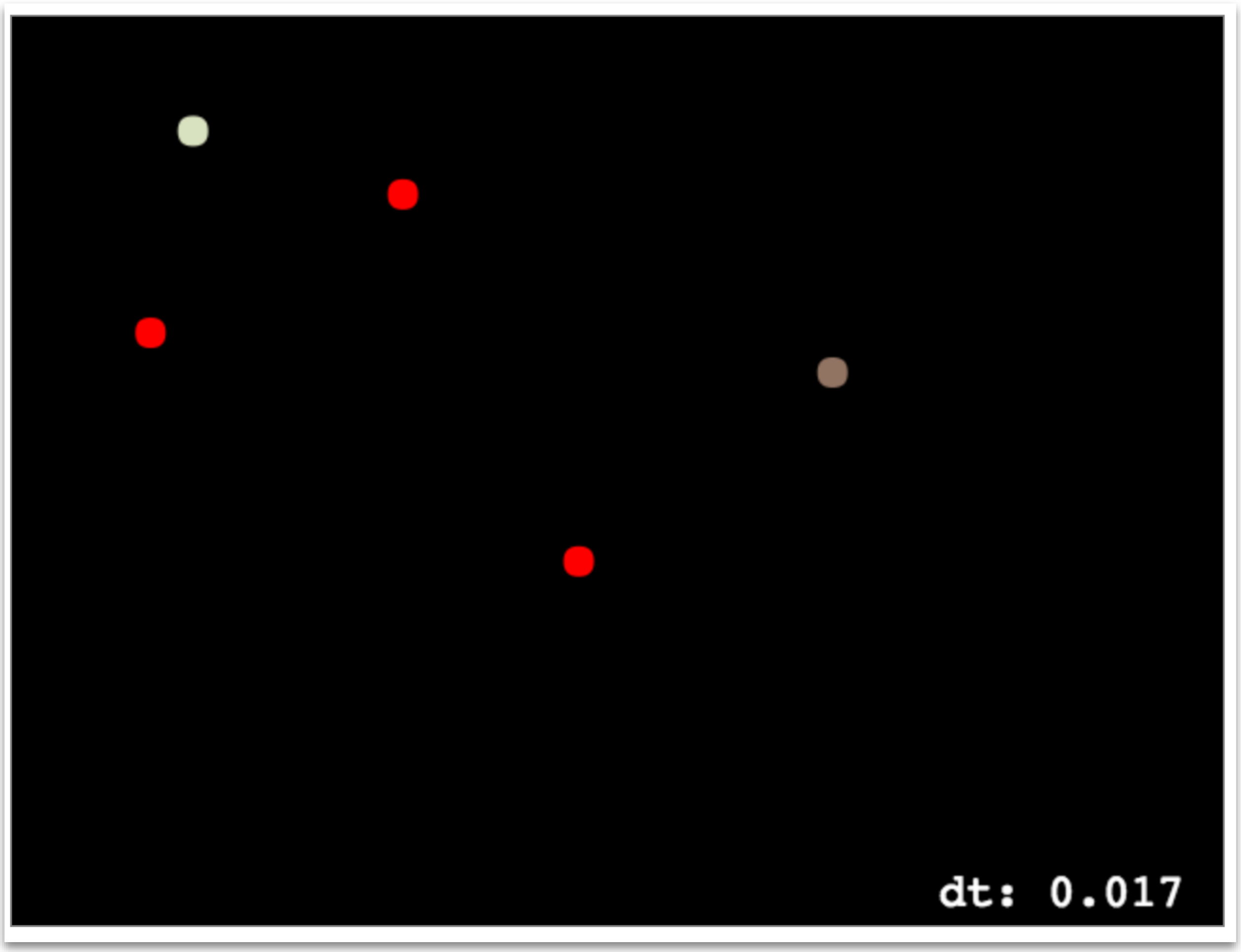
```
checkCircleClicked: function(mouse){
     // looping through circle array backwards, why?
     for(var i = this.circles.length -1; i>=0; i--){
          var c = this.circles[i];
          if (pointInsideCircle(mouse.x, mouse.y, c)){
               c.fillStyle = "red";
               c.xSpeed = c.ySpeed = 0;
               break; // we want to click only one circle
          }
     }

}
```

3) Now make `main.doMousedown()` look like this:

```
var mouse = getMouse(e);
// have to call through app.main because this = canvas
// can you come up with a better way?
app.main.checkCircleClicked(mouse);
```

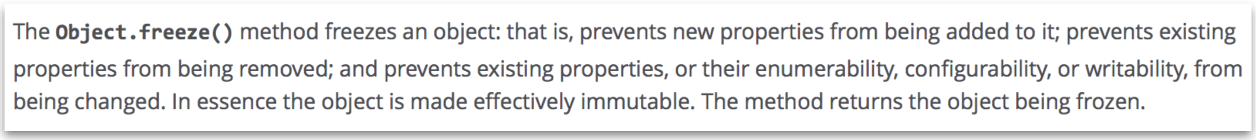
Save and reload the page and click some circles to see them turn red and stop moving. You may want to want to make `CIRCLE.START_RADIUS` a larger value so that it is easier to see and click on the circles.



Part IV - Object.freeze()

One last thing - let's use `Object.freeze()` on our 3 "constants" objects to make them truly immutable.

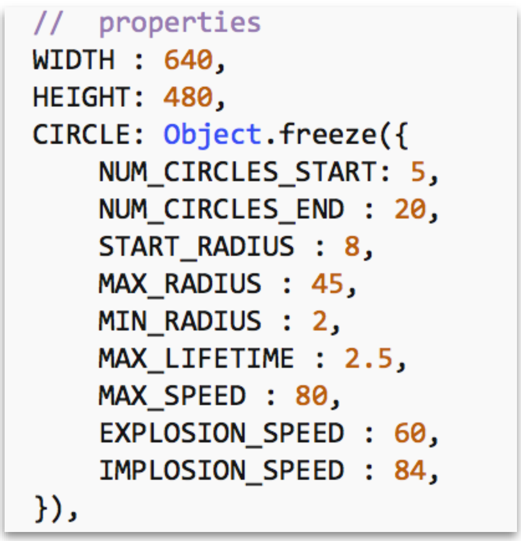
What does `Object.freeze()` do?



The **`Object.freeze()`** method freezes an object: that is, prevents new properties from being added to it; prevents existing properties from being removed; and prevents existing properties, or their enumerability, configurability, or writability, from being changed. In essence the object is made effectively immutable. The method returns the object being frozen.

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

This is how you apply `Object.freeze()` to `CIRCLE` - go ahead and do this to `CIRCLE_STATE` and `GAME_STATE` as well.



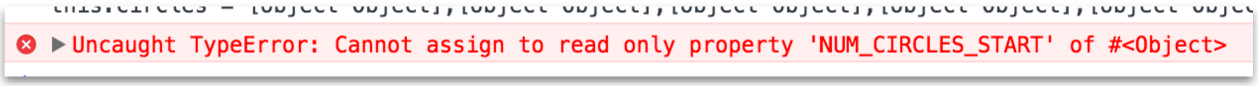
```
//  properties
WIDTH : 640,
HEIGHT: 480,
CIRCLE: Object.freeze({
    NUM_CIRCLES_START: 5,
    NUM_CIRCLES_END : 20,
    START_RADIUS : 8,
    MAX_RADIUS : 45,
    MIN_RADIUS : 2,
    MAX_LIFETIME : 2.5,
    MAX_SPEED : 80,
    EXPLOSION_SPEED : 60,
    IMPLOSION_SPEED : 84,
}),
```

Now if the code we write mistakenly changes one of these values, we'll get an error to the console.

Part V - Break the code!

1) One last thing - add the following to `app.main.init():`

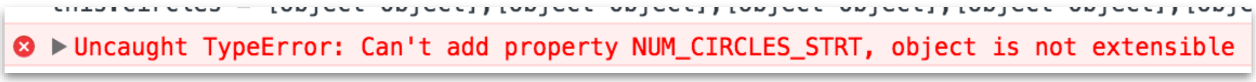
```
this.CIRCLE.NUM_CIRCLES_START = 100
```



```
▶Uncaught TypeError: Cannot assign to read only property 'NUM_CIRCLES_START' of #<Object>
```

Reload the page, you'll get this error because we "froze" `app.main.CIRCLE`:

Type: `this.CIRCLE.NUM_CIRCLES_STRT = 100`



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

Misspelling "START" with "STRT" also gives an error:

Go ahead and fix the errors.

In the next part this will begin to look more like a game!

Boomshine C Rubric

DESCRIPTION	SCORE	VALUE %
Circles Large Enough – Circle are large enough to click easily.		**20**
Clicking Circle – Clicking circles works correctly.		**40**
Circles Stop – Clicking a circle stops it and turns it red.		**40**
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.**