Boomshine D

Overview:
Up to now we've spent quite a bit of time building out a skeleton of a game, but it doesn't do much yet. This time we'll add levels, explosions, circle collisions, a HUD, and game states to our *Boomshine* clone.

We'll also talk about the academic definitions that try to answer the question "What is a game", as well as get you to think about what **Boomshine ICE** needs to becoming something playable that is compelling to at least some people.

Part I - Getting ready for levels and explosions

Make a copy of the **Boomshine-C folder** and name it **Boomshine-D**

1) Add three new properties to `app.main`:

```
gameState : undefined,
roundScore : 0,
totalScore : 0,
```

2) Initialize `main.gameState` in `main.init()`

```
this.gameState = this.GAME_STATE.BEGIN;
```

3) Add a new method named `reset() to app.main()` - put it right after `init()`:

```
// creates a new level of circles
reset: function(){
     this.numCircles += 5;
     this.roundScore = 0;
     this.circles = this.makeCircles(this.numCircles);
},
```

4) Now call `this.reset()` in init() - it should now appear as follows:

```
// methods
init : function() {
    console.log("app.main.init() called");
    // initialize properties
    this.canvas = document.querySelector('canvas');
    this.canvas.width = this.WIDTH;
    this.canvas.height = this.HEIGHT;
    this.ctx = this.canvas.getContext('2d');

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

    this.gameState = this.GAME_STATE.BEGIN;

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

    // load level
    this.reset();

    // start the game loop
    this.update();

},
```

5) Reload the page - it should function identically as before (except there are now 5 more circles at the start.

Basically, init() is only going to be called once, and reset() will be called whenever it's time to load a new level.

6) In `app.main`, create an empty implementation of checkForCollisions()

```
checkForCollisions: function(){
    // TODO
}
```

7) Call it in `app.main.update()`, right after the “move circles” call:

```
// CHECK FOR COLLISIONS
this.checkForCollisions();
```

8) Make a few additions to `app.main.drawCircles()` so that it looks like this:

```
drawCircles: function(ctx){
    if(this.gameState == this.GAME_STATE.ROUND_OVER) this.ctx.globalAlpha = 0.25;
    for(var i = 0; i < this.circles.length; i++){
        var c = this.circles[i];
        if(c.state === this.CIRCLE_STATE.DONE) continue;
        c.draw(ctx);
    }
},
```

Reload the page - it should work as before.

Part II - HUD!

1) Let's lower the opacity of the circles we're drawing so they are semi-transparent:

```
// ii) draw circles
this.ctx.globalAlpha = 0.9; // NEW
this.drawCircles(this.ctx);
```

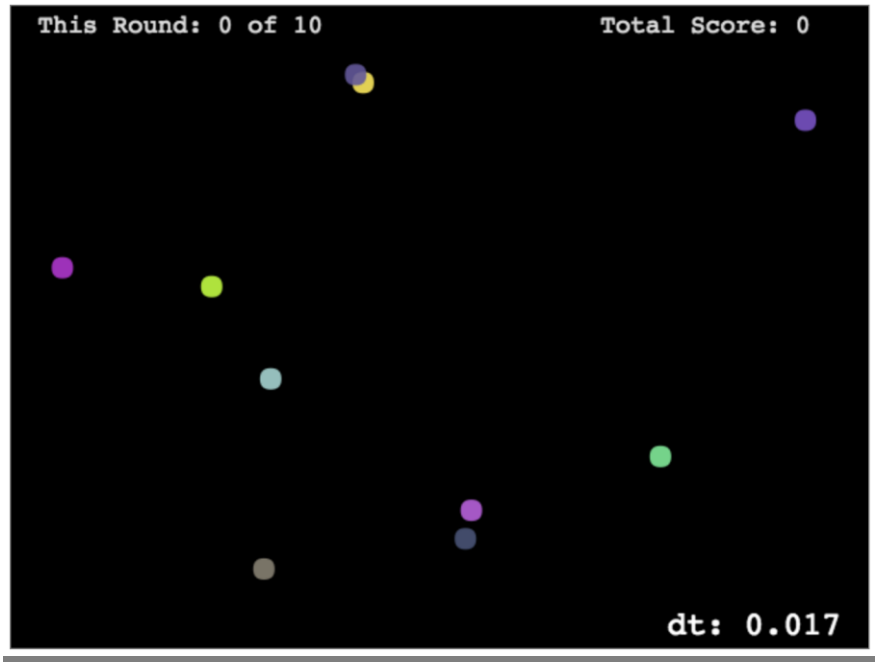
2) In app.main, implement an initial version of drawHUD():

```
drawHUD: function(ctx){
    // draw score
    // fillText(string, x, y, css, color)
    this.fillText("This Round: " + this.roundScore + " of " + this.numCircles, 20, 20, "14pt courier", "#ddd");
    this.fillText("Total Score: " + this.totalScore, this.WIDTH - 200, 20, "14pt courier", "#ddd");

}
```

3) Add the following to `app.main.update()`, right after the `drawCircles()` call:

```
// iii) draw HUD
this.ctx.globalAlpha = 1.0;
this.drawHUD(this.ctx);
```

4) Reload the page, it should work as before, and now show scoring.



5) Your `app.main.update()` method should now appear as follows:

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

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

    // 3) HOW MUCH TIME HAS GONE BY?
    var dt = this.calculateDeltaTime();

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

    // CHECK FOR COLLISIONS
    this.checkForCollisions();

    // 5) DRAW
    // i) draw background
    this.ctx.fillStyle = "black";
    this.ctx.fillRect(0,0,this.WIDTH,this.HEIGHT);

    // ii) draw circles
    this.ctx.globalAlpha = 0.9;
    this.drawCircles(this.ctx);


    // iii) draw HUD
    this.ctx.globalAlpha = 1.0;
    this.drawHUD();


    // iv) draw debug info
    if (this.debug){
        // draw dt in bottom right corner
        this.fillText("dt: " + dt.toFixed(3), this.WIDTH - 150, this.HEIGHT - 10, "18pt courier", "white");
    }

},
```

Part III - Exploding Circles!

1) Let's change the `.state` of the circle, the `.gameState` of the game, and the score whenever a circle is clicked. Modify `checkCircleClicked()` to appear as follows:

```
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;
            c.state = this.CIRCLE_STATE.EXPLODING;
            this.gameState = this.GAME_STATE.EXPLODING;
            this.roundScore ++;
            break; // exit the for, we just want to click one circle
        }
    }

},
```

Reload the page. Clicking circles makes them stop, turn red, and the round score goes up by 1.

2) To get circles exploding:

A) Now we need to modify `moveCircles()` to monitor and transition between the various states of the circles:

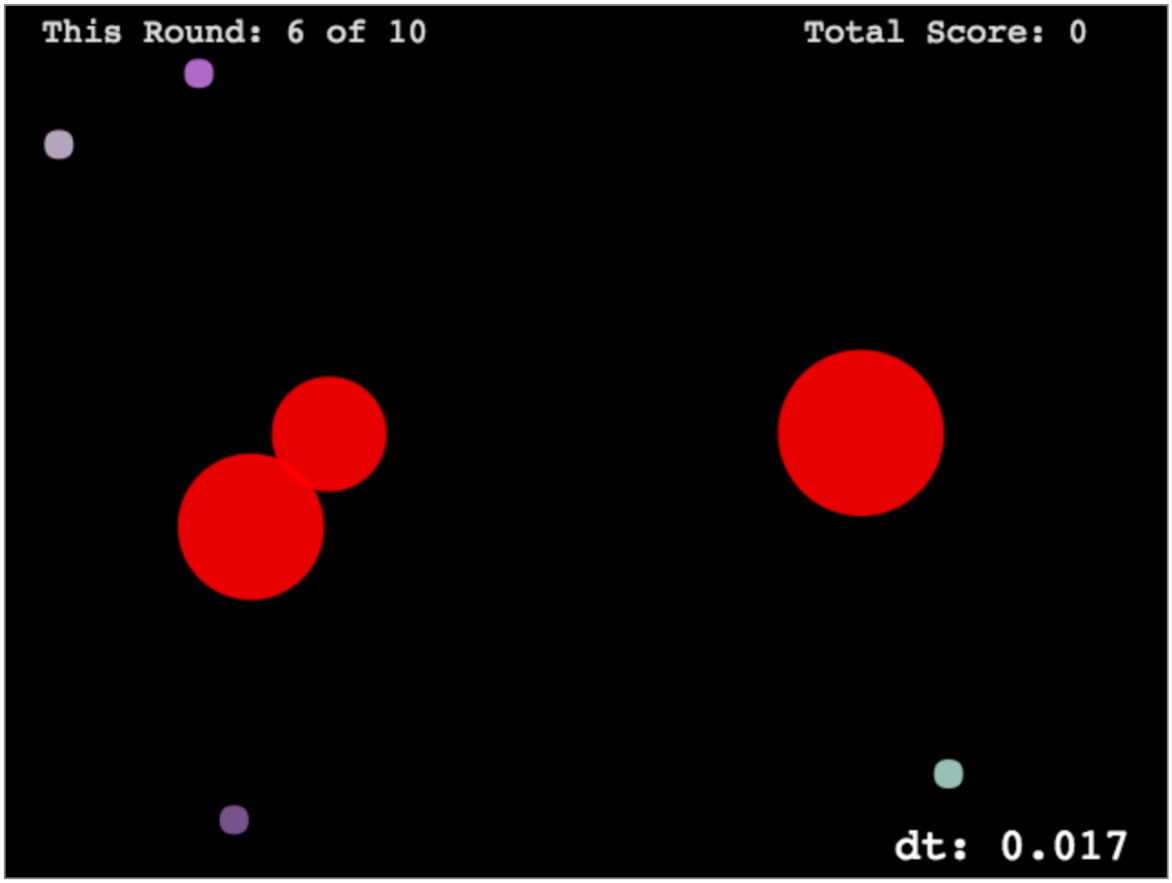
- `CIRCLE_STATE.NORMAL`
- `CIRCLE_STATE.EXPLODING`
- `CIRCLE_STATE.MAX_SIZE`
- `CIRCLE_STATE.IMPLODING`
- `CIRCLE_STATE.DONE`

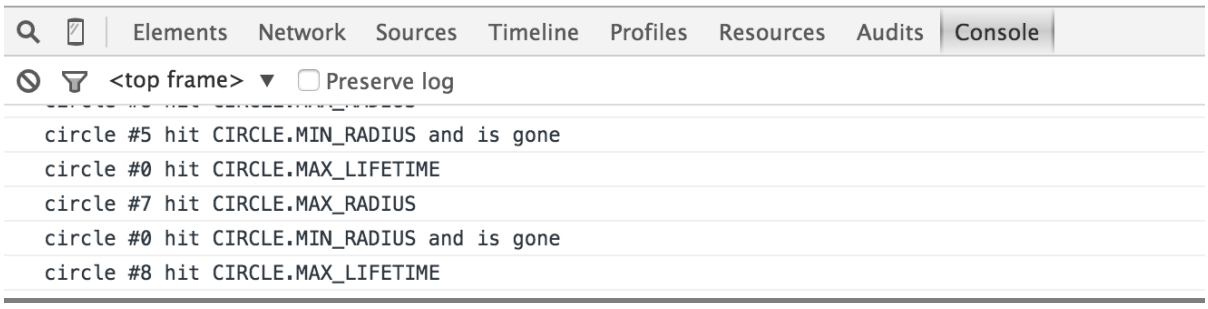
B) Modify `app.main.moveCircles()` to appear as it does below (see mycourses for code):

```
moveCircles: function(dt){
    for(var i=0;i<this.circles.length; i++){
        var c = this.circles[i];
        if(c.state === this.CIRCLE_STATE.DONE) continue;
        if(c.state === this.CIRCLE_STATE.EXPLODING){
            c.radius += this.CIRCLE.EXPLOSION_SPEED  * dt;
            if (c.radius >= this.CIRCLE.MAX_RADIUS){
                c.state = this.CIRCLE_STATE.MAX_SIZE;
                console.log("circle #" + i + " hit CIRCLE.MAX_RADIUS");
            }
            continue;
        }

        if(c.state === this.CIRCLE_STATE.MAX_SIZE){
            c.lifetime += dt; // lifetime is in seconds
            if (c.lifetime >= this.CIRCLE.MAX_LIFETIME){
                c.state = this.CIRCLE_STATE.IMPLODING;
                console.log("circle #" + i + " hit CIRCLE.MAX_LIFETIME");
            }
            continue;
        }

        if(c.state === this.CIRCLE_STATE.IMPLODING){
            c.radius -= this.CIRCLE.IMPLOSION_SPEED * dt;
            if (c.radius <= this.CIRCLE.MIN_RADIUS){
                console.log("circle #" + i + " hit CIRCLE.MIN_RADIUS and is gone");
                c.state = this.CIRCLE_STATE.DONE;
                continue;
            }

        }

        // move circles
        c.move(dt);

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

    } // end for loop
},
```

Reload the page and test it. Clicking the circles should cause them to balloon up to `CIRCLE_STATE.MAX_SIZE`, wait for `CIRCLE.MAX_LIFETIME` seconds, and then shrink down to `CIRCLE.MIN_RADIUS`. At this point, the circle's `.state` is transitioned to `CIRCLE_STATE.DONE`

`drawCircles()` already has code that will skip drawing a circle if its `.state` is `CIRCLE_STATE_DONE`.





3) To make it so we can only click one circle per level, and fix a possible pausing issue, add the following to the top of `app.main.doMouseDown()`**:**

```
// ugh - the 'this' issue - should have planned ahead better
var main = app.main;

// unpause on a click
// just to make sure we never get stuck in a paused state
if(main.paused){
	main.paused = false;
	main.update();
	return;
};

// you can only click one circle
if(main.gameState == main.GAME_STATE.EXPLODING) return;
```

Reload the page and test it. Now you can only click one circle.

Discussion:

Note that we never delete a circle out of the `circles` array when it's done exploding - we just leave it in the array, and don't draw it to the screen.

Part IV - Colliding Circles!

1) Now we want other circles to explode whenever they collide with a circle that is already exploding.

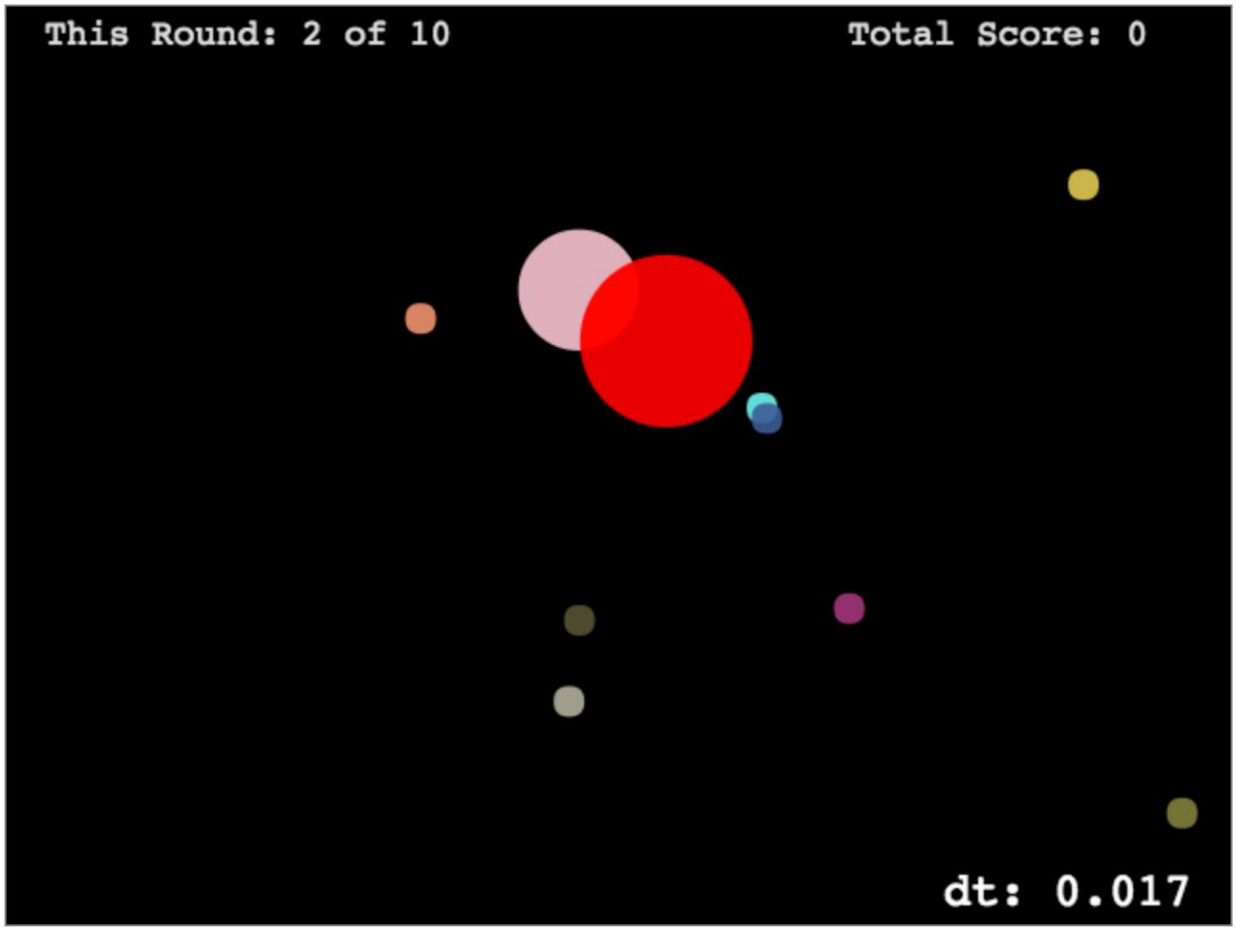
A) Add this helper method to your **utilities.js**:

```
function circlesIntersect(c1,c2){
      var dx = c2.x - c1.x;
      var dy = c2.y - c1.y;
      var distance =  Math.sqrt(dx*dx + dy*dy);
      return distance < c1.radius + c2.radius;
}
```

B) Now implement checkForCollisions() - and unless you're a typing junky, check mycourses for the copy-paste version:

```
checkForCollisions: function(){
    if(this.gameState == this.GAME_STATE.EXPLODING){
        // check for collisions between circles
        for(var i=0;i<this.circles.length; i++){
            var c1 = this.circles[i];
            // only check for collisions if c1 is exploding
            if (c1.state === this.CIRCLE_STATE.NORMAL) continue;
            if (c1.state === this.CIRCLE_STATE.DONE) continue;
            for(var j=0;j<this.circles.length; j++){
                var c2 = this.circles[j];
            // don't check for collisions if c2 is the same circle
                if (c1 === c2) continue;
            // don't check for collisions if c2 is already exploding
                if (c2.state != this.CIRCLE_STATE.NORMAL ) continue;
                if (c2.state === this.CIRCLE_STATE.DONE) continue;

                // Now you finally can check for a collision
                if(circlesIntersect(c1,c2) ){
                    c2.state = this.CIRCLE_STATE.EXPLODING;
                    c2.xSpeed = c2.ySpeed = 0;
                    this.roundScore ++;
                }
            }
        } // end for

        // round over?
        var isOver = true;
        for(var i=0;i<this.circles.length; i++){
            var c = this.circles[i];
            if(c.state != this.CIRCLE_STATE.NORMAL && c.state != this.CIRCLE_STATE.DONE){
             isOver = false;
             break;
            }
        } // end for

        if(isOver){
            this.gameState = this.GAME_STATE.ROUND_OVER;
            this.totalScore += this.roundScore;
         }

    } // end if GAME_STATE_EXPLODING
},
```

Test it. Collisions and explosions should now be working, but you're only getting 1 level.



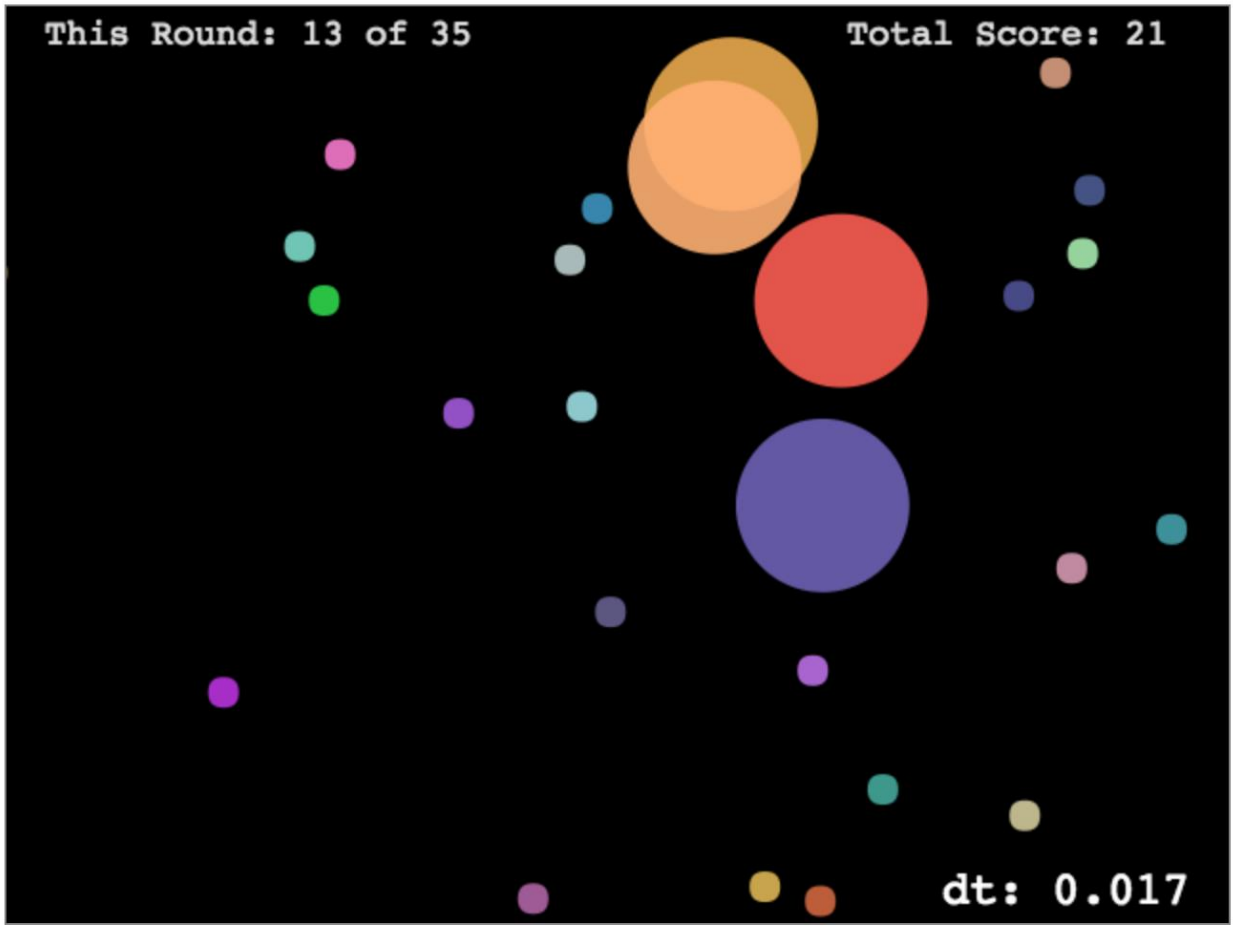
Part V - Levels and Game State Transitions!

1) To get levels working, we now need to add code that will handle transitioning between our various game states:

A) Modify `doMousedown()` to appear as follows:

```
doMousedown: function(e){
    // ugh - the 'this' issue - should have planned ahead better
    var main = app.main;

    // unpause on a click
    // just to make sure we never get stuck in a paused state
    if(main.paused){
        main.paused = false;
        main.update();
        return;
    };

    // you can only click one circle
    if(main.gameState == main.GAME_STATE.EXPLODING) return;

    // if the round is over, reset and add 5 more circles
    if(main.gameState == main.GAME_STATE.ROUND_OVER){
        main.gameState = main.GAME_STATE.DEFAULT;
        main.reset();
        return;
    }

    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);

},
```

Test the code. When the explosions are finished, you can click to load a new level. Note that the number of circles increments by 5, and that the scoring works properly.



Part VI - Screens Baby, Game Screens!

1) So we know are using a few different game states (the first 4 in our `GAME_STATE` "enumeration") - but how does the **user** know which state they are in? Mostly by guessing at this point. This is poor usability - which will make the end product a lot less fun to play - let's fix that with having the HUD give them more information.

A) Make `drawHUD()` look like this (check myCourses for code):

```
drawHUD: function(ctx){
    ctx.save(); // NEW
    // draw score
    // fillText(string, x, y, css, color)
    this.fillText("This Round: " + this.roundScore + " of " + this.numCircles, 20, 20, "14pt courier", "#ddd");
    this.fillText("Total Score: " + this.totalScore, this.WIDTH - 200, 20, "14pt courier", "#ddd");

    // NEW
    if(this.gameState == this.GAME_STATE.BEGIN){
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
        this.fillText("To begin, click a circle", this.WIDTH/2, this.HEIGHT/2, "30pt courier", "white");
    } // end if

    // NEW
    if(this.gameState == this.GAME_STATE.ROUND_OVER){
        ctx.save();
        ctx.textAlign = "center";
        ctx.textBaseline = "middle";
        this.fillText("Round Over", this.WIDTH/2, this.HEIGHT/2 - 40, "30pt courier", "red");
        this.fillText("Click to continue", this.WIDTH/2, this.HEIGHT/2, "30pt courier", "red");
        this.fillText("Next round there are " + (this.numCircles + 5) + " circles", this.WIDTH/2 , this.HEIGHT/2
    } // end if

    ctx.restore(); // NEW
}
```

Reload the page. Note that we have a starting screen, and a “level done” screen with any remaining circles faded out but still moving in the background. Nice!
(We also got to do this without making any buttons - you might not be so lucky in your game.)

Part VII - Are we done yet?

Now walk through our completed code - and be sure to attempt to understand what's going on. Please ask about anything you are not sure at our next meeting.

In Boomshine-E will do a little re-factoring, but that's it, no more features.

Are YOU done - is this a game?

Here's a comparative chart of "Elements of a game definition" according to various authors and game theoreticians:

Rules of Play: Game Design Fundamentals

By Katie Salen and Eric Zimmerman, Copyright Massachusetts Institute of Technology _ 2004, Publishe

Elements of a game definition	Parlett	Abt	Huizinga	Caillois	Suits	Crawford	Costikyan
Proceeds according to rules that limit players	√	√	√	√	√	√	
Conflict or contest	√					√	
Goal-oriented/outcome-oriented	√	√			√		√
Activity, process, or event		√			√		
Involves decision-making		√				√	√
Not serious and Absorbing			√				
Never associated with material gain			√	√			
Artificial/Safe/Outside ordinary life			√	√		√	
Creates special social groups			√				
Voluntary				√	√		
Uncertain				√			
Make-believe/Representational				√		√	
Inefficient					√		
System of parts/Resources and Tokens						√	√
A form of art							√

*A)When did **Boomshine ICE** become a game according to the third column (Huizinga) above?*

- Proceeds according to rules that limit players
- Not Serious and Absorbing
- Never associated with material gain
- Artificial/Safe/Outside normal life
- Creates Special Social Groups

Where is our ***Boomshine ICE*** falling short?

*B)When did **Boomshine ICE** become a game according to the second column (Abt) above?*

- Proceeds according to rules that limit players
- Goal Oriented/Outcome Oriented
- Activity, process, or event
- Involves decision making

Where is our ***Boomshine ICE*** falling short?

*C) Better yet, when did **Boomshine ICE** become a game a significant number of "someone's" might want to play? i.e.The non-academic definition of a game.*

Boomshine ICE A: one bouncing circle

Boomshine ICE B: multiple bouncing circles

Boomshine ICE C: multiple bouncing circles that stop and turn red when we click them

Boomshine ICE D:

- multiple bouncing circles
- clicking a circle starts a chain reaction
- scoring keeps track of how many circles exploded
- when the explosions stops, load a new level
- more circles appear on each level
- continues forever, or until the browser crashes

My Take:
Boomshine ICE D is still not really (yet) what we like to call a game - it rewards repetitive and addictive behaviors like FourSquare, Cookie Clicker, 3DS Streetpass, …

D) What does the original version of *Boomshine* add to the Boomshine ICE make it more of a game that people would actually like to play?

(Think about it, we'll discuss this in class)

E) For **Boomshine-E**, we're going to refactor some of the code, but we're not going to add any features.

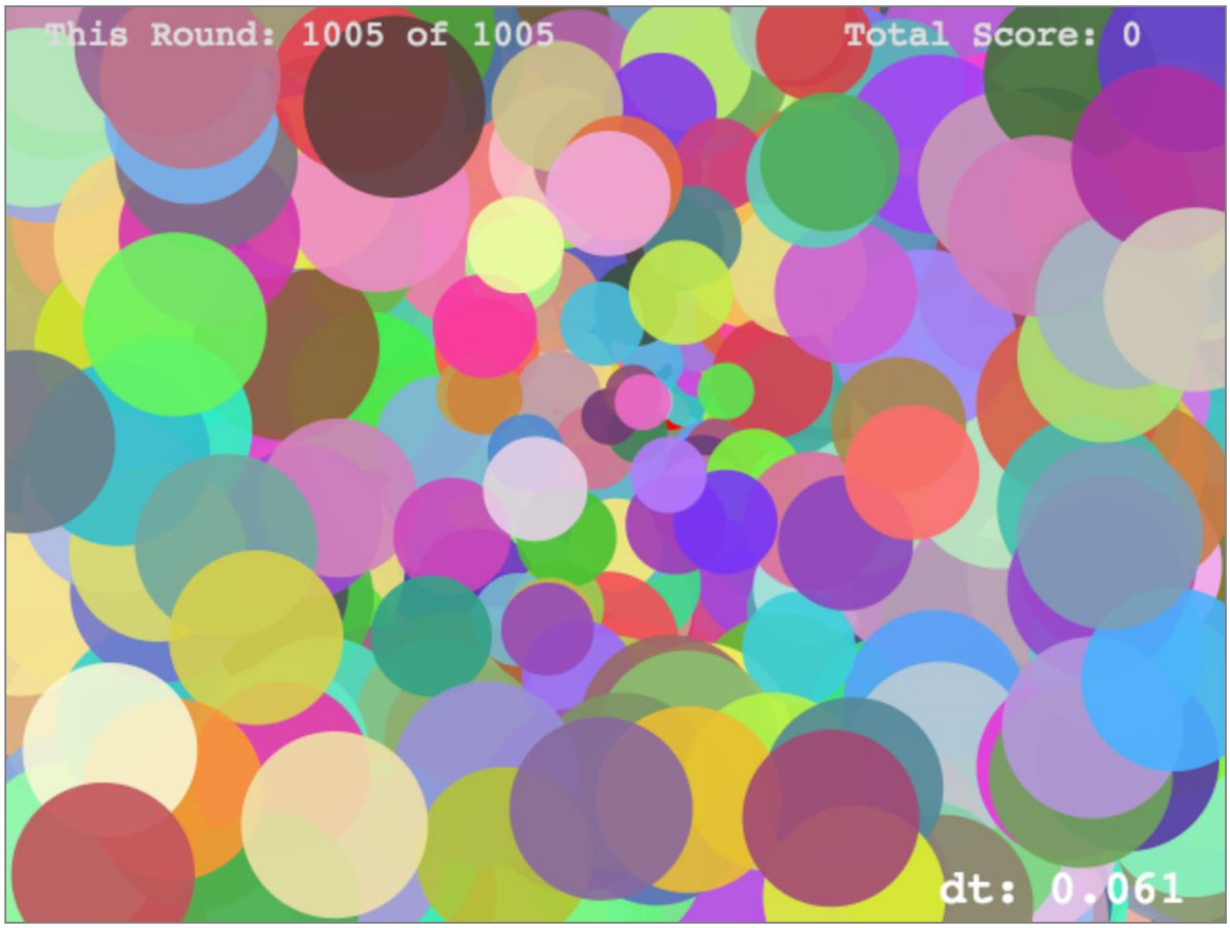
You will soon be asked to finish up this Boomshine, and add some of the features it needs to become a real game that at least some people want to play (hint: it actually won't take too much code more to do this)

Part VIII - Break this code

Just for fun, change NUM_CIRCLES_START to a large number, like 1000, 5000, or 10000 and reload the page.

Here we can see that the app still runs at 15 FPS with 10,000 circles (on a 2015 Mac laptop). Note that the relative speed of the circles remains the same even as the frame rate drops - this is because of our use of *delta time* (`dt`) to smooth out the circle motion.

(*But if you click, the browser will really start hurting. Use a number more like 1000 to test the explosion behavior!*)



That's pretty good performance for 2d canvas, but you could actually do even better, here are some ideas:

- Copying a bitmap is faster than creating shapes using the canvas API. So draw each circle only once (with the `ctx.arc()`… code) to an offscreen canvas, then save that bitmap and use `ctx.drawImage()` to *blit* the circle image to the screen.

- For super-fast iteration, store the circle data in byte and float arrays, or in a struct-like format. See the *Working with Complex Data Structures* section of this page: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Typed_arrays

- Use WebGL, which is video hardware accelerated, to draw your sprites. A library can help with this. The **pixi.js** library uses WebGL to draw sprites: http://www.pixijs.com

Part IX - Submission

Now walk through our completed code - and be sure to attempt to understand what's going on - if you don't ask!

ZIP and Post. **Include a link to your code on Banjo in the submission comments.**