Splitting the game into several JavaScript files

INTRODUCTION

JSBin is a great tool for sharing code, for experimenting, etc. But as soon as the size of your project increases, this tool is not suited for developing a real project.

In order to keep working on this game framework, we recommend that you modularize the project and split the JavaScript code into different JavaScript files:

- Look at the different functions and isolate those that have no dependence with the framework. Obviously, the sprite utility functions, the collision detection functions, and the ball constructor function can be put outside of the game framework, and could easily be reused in other projects. Key and mouse listeners also can be isolated, gamepad code too...
- Look at what you could change to limit the dependencies: add a parameter in order to make some functions independent from global variables, for example.
- In the end, try to keep in the `game.js` file only the core of the game framework (`init` function, `mainloop`, game states, score, levels), and put the rest in separate files: `utils.js`, `sprites.js`, `collision.js`, `listeners.js`, etc.

LET'S DO THIS TOGETHER!

Start with a simple structure

First, create a `game.html` file that contains the actual HTML code:

`game.html`:

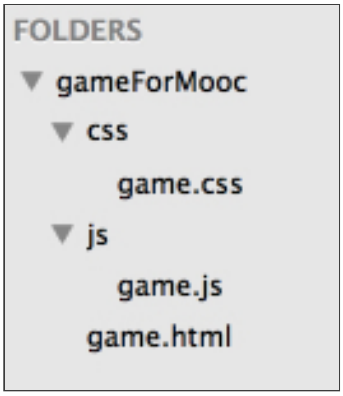
```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Nearly a real game</title>
 <!-- External JS libs -->
 <scriptsrc="https://cdnjs.cloudflare.com/ajax/libs/howler/1.1.25/howler.min.js">
</script>
 <!-- CSS files for your game -->
 <link rel="stylesheet" href="css/game.css">
 <!-- Include here all game JS files-->
 <script src="js/game.js"></script>
</head>
<body>
 <canvas id="myCanvas" width="400"height="400"></canvas>
```

```
</body>
</html>
```

Here is the `game.css` file (very simple):

```
canvas {
   border: 1px solid black;
}
```

All the JavaScript code from the last JSBin example is in a `game.js`file, located in a subdirectory `js` under the directory where the`game.html` file is located. The CSS file is in a `css` subdirectory:



Try the game: open the `game.html` file in your browser. If the game does not work, open the devtools, look at the console, fix the errors, try again, etc. You will do this several times when you split your file and encounter errors.

Isolate the ball function constructor

Put the `Ball` constructor function in a `js/ball.js` file, include it in the `game.html` file and try the game: it does not work! Let's open the console:

`Ball.js`:

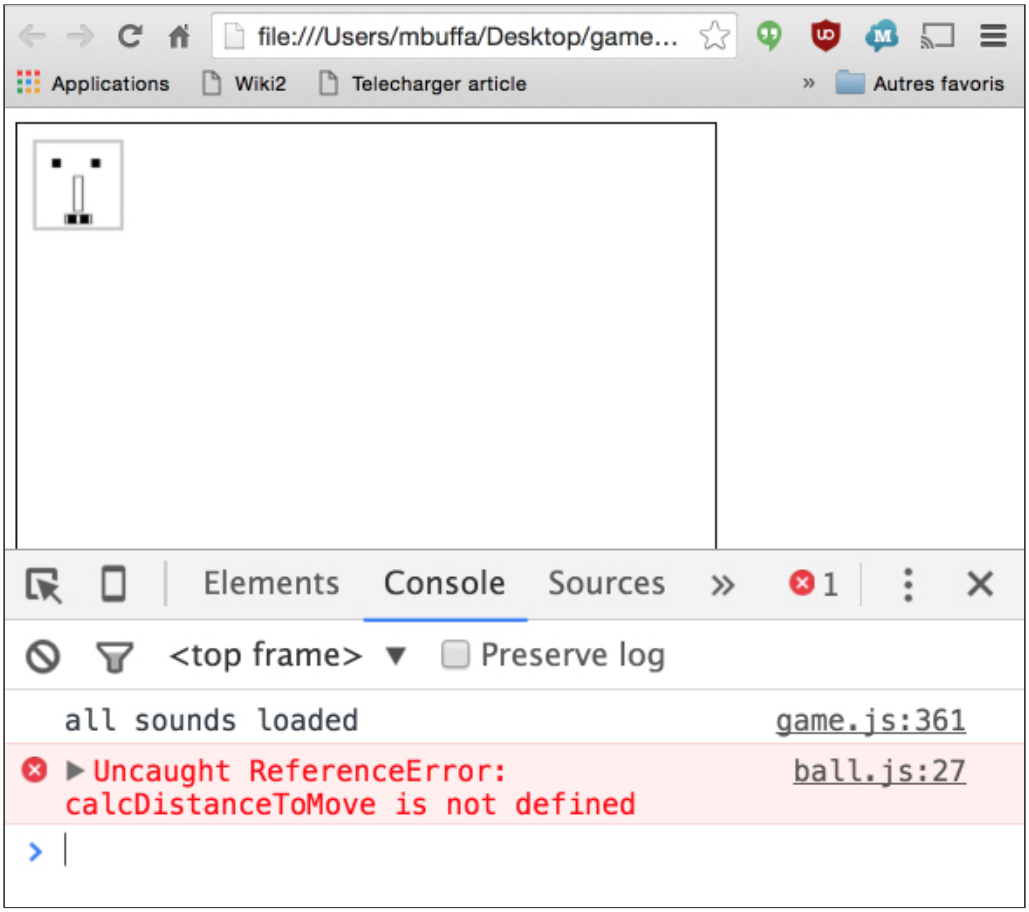
```
// constructor function for balls
 function Ball(x, y, angle, v, diameter) {
   ...
    this.draw = function () {
       ctx.save();
       ...
    };
    this.move = function () {
      ...
      this.x += calcDistanceToMove(delta,incX);
```

```
        this.y += calcDistanceToMove(delta,incY);
    };
}
```

We need to isolate the time based animation functions into a separate file

Hmmm... the `calcDistanceToMove` function is used here, but is defined in the `game.js` file, inside the `GF` object and will certainly raise an error... Also, the `ctx` variable should be added as a parameter to the `draw` method, otherwise it won't be recognized...

Just for fun, let's try the game without fixing this, and look at the devtools console:



Aha! The `calcDistanceToMove` function is indeed used by the Ball constructor in `ball.js` at line 27 (it moves the ball using time based animation). If you look carefully, you will see that it's also used for moving the monster, etc. In fact, there are parts in the game framework related to time-based animation. Let's move them into a`timeBasedAnim.js` file!!

Fix: extract the time based animation related utility functions and add a `ctx` parameter to the draw method of `ball.js` and don't forget to add it in `game.js` where `ball.draw()` is called. the call should be now `ball.draw(ctx);` instead of `ball.draw()` without any parameter.

`timeBasedAnim.js`:

```
var delta, oldTime = 0;

function timer(currentTime) {
   var delta = currentTime - oldTime;
   oldTime = currentTime;
   return delta;
}

var calcDistanceToMove = function (delta,speed) {
   //console.log("#delta = " + delta + " speed = " + speed);
   return (speed * delta) / 1000;
 };
```

Isolate the part that counts the number of frames per second

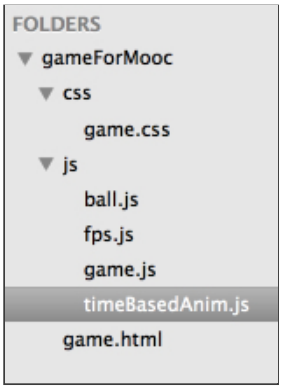
We needed to add a small `initFPS` function for creating the `<div>`that displays the FPS value... this function is now called from the`GF.start()` method. There was code in this `start` method that has been moved into the `initFPS` function we created and added into the `fps.js` file.

`fps.js`:

```
// vars for counting frames/s, used by the measureFPS function
var frameCount = 0;
var lastTime;
var fpsContainer;
var fps;

var initFPSCounter = function() {
  // adds a div for displaying the fps value
  fpsContainer =document.createElement('div');
  document.body.appendChild(fpsContainer);
}
var measureFPS = function (newTime) {

  // test for the very first invocation
  if (lastTime === undefined) {
     lastTime = newTime;
     return;
  }

  //calculate the difference between last & current frame
  var diffTime = newTime - lastTime;

  if (diffTime >= 1000) {
     fps = frameCount;
     frameCount = 0;
```

```
            lastTime = newTime;
        }

30.     //and display it in an element we appended to the
        // document in the start() function
        fpsContainer.innerHTML = 'FPS: ' + fps;
        frameCount++;
    };
```

At this stage, the structure is as follows:



Let's continue and isolate the event listeners

Now, we put all the code located in the `GF.start()` method, that creates the listeners, into a `listeners.js` file. We had to pass the canvas as an extra parameter (to resolve a dependency) and we also moved the `getMousePos` method in there.

`listeners.js`:

```
    function addListeners(inputStates, canvas) {
        //add the listener to the main, window object, and update the states
        window.addEventListener('keydown',function (event) {
            if (event.keyCode === 37) {
                inputStates.left = true;
            } else if (event.keyCode === 38) {
                inputStates.up = true;
            } ...
        }, false);

        //if the key is released, change the states object
        window.addEventListener('keyup', function(event) {
            ...
14.     }, false);

        // Mouse event listeners
        canvas.addEventListener('mousemove',function (evt) {
```

```
        inputStates.mousePos = getMousePos(evt,canvas);
    }, false);
...
}

function getMousePos(evt, canvas) {
    ...
}
```

Isolate the collision tests

Following the same method, let's put this in a `collisions.js` file:

```
// We can add the other collision functions seen in the
// course here...
// Collisions between rectangle and circle
function circRectsOverlap(x0, y0, w0, h0,cx, cy, r) {
    ...
}

function testCollisionWithWalls(ball, w, h){
    ...
}
```

We added the width and height of the canvas as parameters to the`testCollisionWithWalls` function to resolve depedencies. We could have put in this file the other collision functions (circle-circle and rectangle-rectangle) presented in the course.

FINAL DOWNLOADABLE VERSION AND CONCLUSION

At this stage, we have this structure:

Final `game.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
 <meta charset="utf-8">
 <title>Nearly a real game</title>
 <link rel="stylesheet" href="css/game.css">
 <scriptsrc="https://cdnjs.cloudflare.com/ajax/libs/howler/1.1.25/howler.min.js">
</script>
 <!-- Include here all JS files -->
 <script src="js/game.js"></script>
 <script src="js/ball.js"></script>
 <script src="js/timeBasedAnim.js"></script>
 <script src="js/fps.js"></script>
 <script src="js/listeners.js"></script>
 <script src="js/collisions.js"></script>
</head>
<body>
 <canvas id="myCanvas" width="400"height="400"></canvas>
</body>
</html>
```

We could go further by defining a `monster.js` file, turning all the code related to the monster/player into a well formed object with draw / move methods, etc. There are many potential improvementsyou could make. JavaScript experts are welcome to make a much fancier version of this little game :-)

Download the zip for this version, just open the game.html file in your browser!

The main purpose of this week course was to show you the main techniques/approaches for dealing with animation, interactions, collisions, dealing with game states, etc.

The quizzes for this week are not so important this time. We hope that you will write your own game - you can reuse some of the examples and modify them, improve the code structure, playability, add sounds, better graphics, more levels, etc.