ES6 Particle System

I decided to incorporate my presentation on ECMAscript 6 with one of the existing labs to showcase a working example of a an application written using the new features of ECMAscript.

Babel

ES6 is not currently in most browsers but thankfully a NPM module is available that will compile the ES6 source code into a browser compatible version.

Babel must be installed as a global node module

```
npm install babel -g
```

compile using the following command

```
babel ./dev/ --out-files ./build/app.js
```

Classes

In JavaScript inheritance is prototype-based, objects inherit through their prototype object. ES6 introduced the ability to define classes, while the syntax is very similar to other class based languages, inheritance is still prototype-based.

Particle

```
class Particle {
    constructor(rad = 40) {
        this.radius = rad;
        this.x = 0;
        this.y = 0;
        this.color = randomColor();
        this.xVelocity = 0;
        this.yVelocity = 0;
    }

    draw (context) {
        ...
    }

    update () {
        ...
    }

    kill (height, width, postition = 0, velocity = 10) {
       ...
    }
}
```

The Particle class sets the radius, x and y position and the color. One of the new features in ES6 is default parameters, this can be seen in the constructor. If a radius is not set as a parameter it will default to 40.

The draw method will draw the particle on the canvas.

BaseCanvas

```
class BaseCanvas {

    constructor(elem) {
        this.canvas = getElem(elem);
```

```
        this.context = this.canvas.getContext('2d');
        this.canvasWidth = this.canvas.width;
        this.canvasHeight = this.canvas.height;
        this.tick = 0;
    };

    clear() {
        this.context.clearRect(0, 0, this.canvasWidth, thi
    }
}
```

The BaseCanvas class encapuslates properties about the canvas. It also includes a function that will clear the canvas.

ParticleSystem

```
class ParticleSystem extends BaseCanvas {

    constructor(elem, amount = 10) {
        super(elem);
        this.particles = [];
        this.amount = amount;

        this.velocity = 5;

        this.animation = null

        this.init();
    }

    ....
}
```

The ParticleSystem extends the BaseCanvas class so it inherits all the canvas properties and methods. The super method is called to create an instance of the parent class. The ParticleSystem creates several Particles with random values and draws them to the canvas.

```
setListeners() {
    let range = getElem('velo'),
        amount = getElem('amount');

    range.addEventListener('change',() =>{
        this.velocity = range.value;
        this.reset();
    },false);

    amount.addEventListener('change', () => {
        this.amount = amount.value;
        this.reset();
    }, false);
}
```

The user can change the velocity and amount of particles using a slider. Change events are set on the range inputs. In the current versio of JavaScript a function is based as callback to be invoked when the event occurs. The context of 'this' in the callback function reflects the object that triggered the event, the context can be set using the bind method. ES6 provides arrow functions which act similarly to standard functions but the context of 'this' is preserved. The syntax is also different, the keyword function is dropped and the => symbols are used.

```
loop() {
    // clear the canvas
    super.clear();

    this.particles.forEach((particle) => {
        particle.update();

        particle.kill(this.canvasHeight, this.canvaWidth,

        particle.draw(this.context);

        this.createParticles();
```

```
        });

        window.requestAnimationFrame(this.loop.bind(this));
    }
```

The loop method will draw each particle to the canvas, and if it is out of the canvas boundaries will reset the starting position.

Conclusion

While researching the new features of ECMAscript 6 I was sceptical that they would make a difference to the way JavaScript is currently written. After developing the particle system using some of the features of ES6 I can safely say that in my opinion development is much easier and the old pitfalls of JavaScript no longer exist (mainly global scope).

While I can recommended developing small applications in ES6, I would refrain from doing so in big applications as the code executed by the browser (ES5) is very different to the code written in ES6. Debugging the code can be extremely hard and confusing. For the time being I would recommend sticking with ES5 but to keep a close eye on the development and specs of ES6 because it is clearing the future of JavaScript.