LABO 4: MODULES / LOOSE COUPLING

Hello NodeJS

Terminal

We gaan vanaf nu vaak aan de slag in Terminal. Lukt het nog niet om vlot te navigeren via de terminal, mappen en bestanden aan te maken / verplaatsen / kopieren / etc…?

Volg dan zeker deze tutorial: https://ashleynolan.co.uk/blog/getting-started-with-terminal

De tijd die investeert in vlot kunnen werken met Terminal win je vrij snel terug…

Node Version Manager

Webpack is een nodejs applicatie. Vooraleer je webpack dus kan gebruiken, moet je nodejs installeren op je computer.

We willen meerdere versies van node kunnen gebruiken op onze computer. Daarom zullen we nodejs niet installeren via de installer van nodejs zelf, maar via een open source tool "**node version manager**".

Hiervoor volgen we volgend stappenplan :

1. Installeer XCode via de App Store (hopelijk heb je dit reeds gedaan).
2. Start XCode op en accepteer de voorwaarden.
3. Open Terminal en maak een .profile bestand aan via volgende commando:

   ```
   touch ~/.profile
   ```

4. Installeer node version manager. het script dat je moet uitvoeren in de console kan je kopieren van https://github.com/creationix/nvm#install-script
5. Vervolgens kunnen we de laatste versie van node installeren via volgend commando:

   ```
   nvm install node
   ```

6. Daarna installeer je Homebrew via het commando dat je kan kopiëren van https://brew.sh/ Homebrew hebben we nodig om de yarn package manager te installeren.

7. Voer volgend commando uit om yarn te installeren:

```
brew install yarn --without-node
```

Oefening 1 - Particles

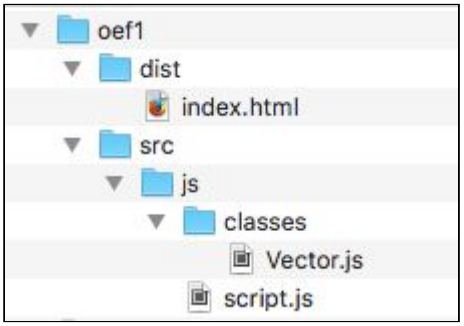
"A particle system is a collection of many many minute particles that together represent a fuzzy object. Over a period of time, particles are generated into a system, move and change from within the system, and die from the system."

—William Reeves, "Particle Systems—A Technique for Modeling a Class of Fuzzy Objects," ACM Transactions on Graphics 2:2 (April 1983), 92.

We zullen een herbruikbaar **particle systeem** bouwen voor onze games / canvas applicaties.

Initialiseren Project

Vertrek van het aangeleverde startproject. Je vindt er de volgende mappenstructuur terug:



Onze javascript broncode zullen we schrijven in de src/js map. Mbv **webpack** zullen we er voor zorgen dat deze broncode omgezet wordt naar een gebundelde script.js file in de dist map.

Open een Terminal, en navigeer naar de root map van dit project (oef 1).
Initialiseer daar het project door een package.json automatisch aan te maken mbv yarn:

```
$ yarn init
```

Na het doorlopen van een korte wizard, zal een package.json file automatisch aangemaakt worden, naast de dist en src mappen.

De **package.json** zou er BVB zo kunnen uitzien:

```
{
  "name": "w04_oef1",
  "version": "1.0.0",
  "description": "Particles oefening.",
  "main": "index.js",
  "author": "Koen De Weggheleire",
  "license": "MIT"
}
```

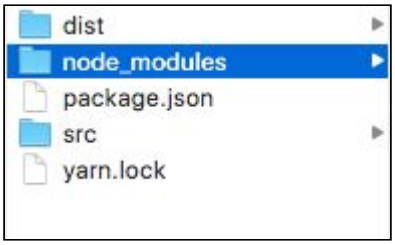
De volgende stap zal zijn om de nodige nodejs libraries te koppelen aan ons project. Wij zullen gebruik maken van **webpack** in combinatie met **babel**.
Voer volgend commando uit om de webpack en babel libraries toe te voegen:

```
$ yarn add webpack babel-core babel-loader --dev
```

Dit commando zal de nodige libraries met hun dependencies downloaden en toevoegen aan het package.json bestand. De package.json zal er nu BVB als volgt uitzien:

```
{
  "name": "w04_oef1",
  "version": "1.0.0",
  "description": "Particles oefening.",
  "main": "index.js",
  "author": "Koen De Weggheleire",
  "license": "MIT",
  "devDependencies": {
      "babel-core": "^6.26.0",
      "babel-loader": "^7.1.2",
      "webpack": "^3.7.1"
  }
}
```

Daarnaast zal je ook merken dat er een map **node_modules** is toegevoegd aan de projectmap



Open deze node_modules map in Finder.
Deze map bevat meer dan alleen onze 4 libraries: elk van deze libraries heeft namelijk een eigen package.json met daarin andere libraries als dependencies. Ook al deze libraries worden recursief gedownload en geïnstalleerd.

Je kan een overzichtje krijgen van deze dependency tree via yarn:

```
$ yarn list
```

Deze node_modules map is vrij omvangrijk.
Bij het delen van projecten, zal deze meestal niet mee verspreid worden: alle informatie om deze dependencies te koppelen zit namelijk in de package.json.

Deze map met dependencies kan dus automatisch aangemaakt worden.
Wis de map node_modules. Voer nu yarn uit, zonder parameters:

```
$ yarn
```

Je zal zien dat de node_modules map opnieuw verschijnt, en alle nodige dependencies gedownload zijn.

Bij het opzetten van een aangeleverd nodejs project zal je dit meestal moeten doen als eerste stap; yarn uitvoeren om de node_modules klaar te zetten.

Configuratie Webpack

De volgende stap zal zijn om webpack te configureren voor ons project. Maak in de root van het project een bestandje **webpack.config.js** aan, met de volgende inhoud:

```
module.exports = {
  entry: `./src/js/script.js`,
  output: {
      path: require('path').resolve(`./dist`),
      filename: `js/script.js`,
  },
  module: {
      rules: [
      {
      test: /\.(js)$/,
      exclude: /node_modules/,
      loader: `babel-loader`
      }
      ]
  }
};
```

We willen nu ook gebruik maken van de feature om volgens browser support te transpilen.
Hiervoor moeten we een nieuwe preset installeren (nieuwe dependency) van babel.
Voer volgend commando uit om de nieuwe dependency toe te voegen aan package.json:

```
$ yarn add babel-preset-env --dev
```

Als je package.json opent zal je zien dat de **nieuwe dependency** is toegevoegd:

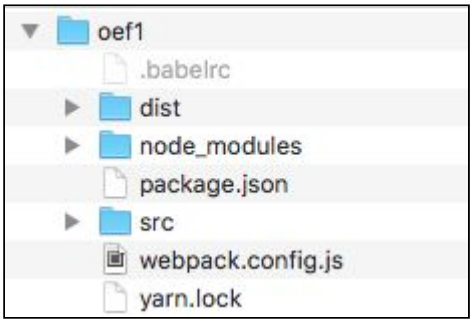
```
{
  "name": "w04_oef1",
  "version": "1.0.0",
  "description": "Particles oefening.",
  "main": "index.js",
  "author": "Koen De Weggheleire",
  "license": "MIT",
  "devDependencies": {
      "babel-core": "^6.26.0",
      "babel-loader": "^7.1.2",
      "babel-preset-env": "^1.6.0",
      "webpack": "^3.7.1"
  }
}
```

Maak nu een bestandje **.babelrc** aan (let op: bestandsnaam begint met een punt!) in de root, met de volgende inhoud:

```
{
  "presets": [
      ["env", {
      "targets": {
      "browsers": ["last 2 versions", "safari >= 7"]
      }
      }]
  ]
}
```

Deze preset zorgt ervoor dat enkel de polyfills voor de laatste 2 versies van elke browser en Safari 7 of groter worden toegevoegd. Op dit moment zijn we volledig klaar met de setup en staat alles klaar voor Webpack & Babel om de code te bundelen in de volgende stap.

Onze finale structuur ziet er als volgt uit:



Uitvoeren Webpack

Bepaalde node modules, zoals **webpack**, kunnen uitgevoerd worden als een terminal commando.
Deze scripts zijn terug te vinden in **node_modules/.bin/**

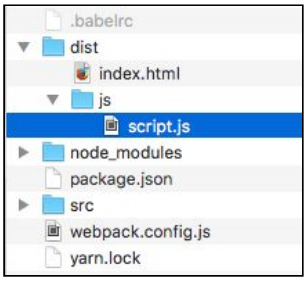
Geef het volgende commando in, om webpack uit te voeren in je project:

```
$ ./node_modules/.bin/webpack
```

Als alles correct is verlopen, krijg je onderstaand resultaat te zien:

```
Hash: 1a1b00f58ac0303938c2
Version: webpack 3.7.1
Time: 657ms
      Asset       Size  Chunks             Chunk Names
js/script.js  5.81 kB       0  [emitted]  main
   [0] ./src/js/script.js 719 bytes {0} [built]
   [1] ./src/js/classes/Vector.js 2.52 kB {0} [built]
```

In de dist folder is een js subfolder aangemaakt met daarin een script.js bestand.



Wanneer je nu index.html in de browser opent zou je de muis coördinaten in de console moeten zien staan. Onze verschillende source files (script.js en classes/Vector.js) zijn samengevoegd in dist/js/script.js.

Webpack Watch Mode

Je kan webpack ook automatisch laten uitvoeren bij het wijzigen van je sourcefiles. Op die manier hoef je niet telkens je Terminal op te roepen bij het maken van een wijzigen.

Voer onderstaande commando uit:

```
./node_modules/.bin/webpack --watch
```

Je zal zien dat webpack wordt uitgevoerd, en dat Terminal blijft runnen; je kan geen nieuwe commando's ingeven.
Laat je Terminal openstaan, en open src/js/script.js in je code editor (**let op dat je de script.js in de src map opent, en niet de dist/js/script.js**).

Verwijder de console.log uit de draw functie, en plaats een console.log('hello world') in je init functie.

Sla het bestand op & refresh je browser. Je merkt dat deze wijzigingen in je src bestanden doorgevoerd zijn in de gebundelde script.js!

Wanneer je het Terminal venster bekijkt, zal je merken dat webpack automatisch opnieuw een bundel gemaakt heeft.

Wanneer je een Terminal programma wil onderbreken, dan moet je gebruik maken van de sneltoets **CTRL + C**. Geef deze sneltoets in in je Terminal window om het webpack watch proces te stoppen.

Package.json scripts

Om te vermijden dat wel lange commando's moeten ingeven, die we toch vergeten, kunnen we gebruik maken van package.json scripts: dit zijn aliassen voor Terminal commando's die je kan definiëren in je package.json file voor een bepaald project.

Plaats een scripts onderdeel in je package.json:

```
"scripts": {
  "development": "webpack --watch"
},
```

We hoeven ./node_modules/.bin zelfs niet te vermelden; package.json scripts zullen automatisch kijken of een commando aanwezig is in deze map.

Voer nu het script uit, mbv yarn:

```
$ yarn run development
```

Je zal zien dat webpack je files watcht. Op deze manier kun je met een eenvoudig commando, complexere acties uitvoeren.

Je moet geen complexe commando's onthouden om een project te compileren: neem een kijkje in de package.json en je ziet meteen de mogelijke opties.

Hello Particle

Maak een Particle class aan in de map classes. De basis hiervan is vrij gelijkaardig aan onze andere classes die bewegen op een canvas: we hebben een location, acceleration en velocity, met een draw loop:

```
import Vector from './Vector.js';

export default class Particle {
  constructor(x, y) {
      this.location = new Vector(x, y);
      this.velocity = new Vector(0, 0);
      this.acceleration = new Vector(0, 0.05);
  }
  draw(ctx) {
      this.velocity.add(this.acceleration);
      this.location.add(this.velocity);
      ctx.fillStyle = `white`;
      ctx.fillRect(this.location.x-1, this.location.y-1, 2, 2);
  }
}
```

In je script.js hou een globale array bij van particles. In de draw functie maak je de canvas zwart en push je telkens een particle op een random x-coördinaat (en y–coördinaat 0) in de array.

Roep de draw functie van elke particle ook op:

Lifespan

Een particle heeft een lifespan: een tijd waarop deze actief is. Bij elke update verlaagt deze lifespan. Wanneer deze 0 wordt, dan is de particle niet meer actief, en wordt deze niet meer getoond.

Definieer deze property in de constructor:

```
this.lifespan = 100;
```

In de draw functie verlaag je deze lifespan:

```
this.lifespan--;
```

En vervolgens gebruik je de lifespan als opacity van de fillStyle. We maken hier gebruik van template strings: in plaats van strings te concateneren mbv quotes en + tekens, kun je gebruik maken van backticks (`) rond je string.
Binnen de backticks kun je javascript code / variabelen gebruiken door deze te wrappen in ${ en }:

```
ctx.fillStyle = `rgba(255, 255, 255, ${this.lifespan / 100})`;
```

De particles zouden nu moeten uitfaden naarmate ze “ouder” worden.
We moeten nu nog de “dode” particles nog verwijderen uit de array. Schrijf in de Particle class een getter die returnt of een particle nog leeft of niet:

```
get isAlive() {
  return this.lifespan > 0;
}
```

In je main draw loop, doe je nu een filter van de particles array om enkel de levende particles over te houden:

```
particles = particles.filter(particle => particle.isAlive);
```

Doe een console.log van de particles.length. Deze zou niet groter dan 100 mogen worden.

Random(min, max)

We hebben regelmatig een functie nodig die een random getal genereert tussen een minimum en een maximum waarde.

Maak een mapje src/js/functions aan met daarin een lib.js. In deze file exporteer je een improved random functie:

```
export const random = (min = 0, max = 1) => {
  return Math.random() * (max - min) + min;
}
```

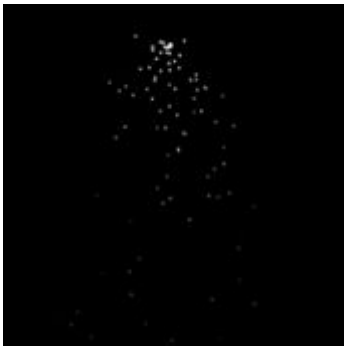
Importeer deze named export nu in je Particle class:

```
import {random} from '../functions/lib.js';
```

Geef elke particle een random velocity in de constructor:

```
this.velocity = new Vector(random(-1, 1), random(-1, 1));
```

Maak de particles tenslotte aan in op de muis coördinaten en test de applicatie.



Oefening 2- Smoke

Je kunt ook leuke particle effecten bereiken, door gebruik te maken van images in plaats van getekende cirkels of vierkanten. Dupliceer oefening 1 naar een nieuwe map en zorg dat webpack watch runt deze nieuwe map.

Plaats de smoke.png uit de bronbestanden in een mapje dist/images:

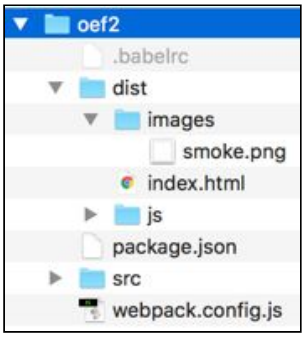


Image load promise

Zoals eerder gezien, kun je afbeeldingen maar renderen op de canvas nadat deze zijn ingeladen. Event handlers of callbacks voor asynchrone communicatie zijn niet zo handig; in plaats hiervan kunnen we gebruik maken van Promises.

Voeg aan je functions/lib.js een loadImage functie toe die een promise returned:

```
export const loadImage = url => {
  return new Promise((response, reject) => {
      //TODO: image load
  });
}
```

In de promise functie maak je een image element aan:

```
const image = new Image();
```

Koppel event listeners voor het load en het error event:

```
image.addEventListener('load', event => response(image));
image.addEventListener('error', event => reject(event));
```

Stel tenslotte het src attribuut in, om de image in te laden:

```
image.setAttribute('src', url);
```

Sommige browsers zijn vrij agressief als het op caching aankomt en zullen onmiddellijk laden (en het load event niet triggeren). Voorzie daarom - na het instellen van het src attribuut - nog een extra check op het complete property, zodat we daar ook de response callback oproepen:

```
if(image.complete) {
  response(image);
}
```

Load & Start

Definieer in de script.js een globale variabele smokeImg. Deze variabele zal de ingeladen smoke.png image bevatten.

Importeer de loadImage functie in je script.js. In de init functie roep je deze op, en zorg je dat de ingeladen image opgeslaan wordt in de globale variabele smokeImg:

```
loadImage(`images/smoke.png`)
  .then(img => smokeImg = img)
```

Koppel een extra then() aan deze promise chain, waarin je het mousemove event koppelt en de draw loop start.
In de draw loop test je of je de smokeImg op de canvas kunt plaatsen:

```
ctx.drawImage(smokeImg, 0, 0);
```

Smoke Particle

Maak nu nieuwe klasse SmokeParticle aan, die overerft van Particle. In deze SmokeParticle klasse komt de te gebruiken afbeelding binnen in de constructor:

```
export default class SmokeParticle extends Particle {
  constructor(x, y, image) {
      super(x, y);
      this.image = image;
  }
}
```

In de draw functie teken je deze image. Maak gebruik van globalAlpha om de opacity van de image te laten afhangen van de lifespan:

```
ctx.globalAlpha = this.lifespan / 100;
ctx.drawImage(this.image, this.location.x, this.location.y);
```

Gebruik nu deze SmokeParticle in plaats van je Particle class in je script.js. Je zou onderstaand resultaat moeten zien:

Update - draw

De draw functies in onze Particle class en de SmokeParticle class bevatten beide de volgende 3 lijnen code:

```
this.lifespan--;
this.velocity.add(this.acceleration);
this.location.add(this.velocity);
```

Deze code heeft niets met rendering te maken, enkel met berekeningen ivm de positionering van het element.

Deze logica zullen we opsplitsen; alles van logica die ook 60 keer per seconde moet uitgevoerd worden, maar niets van rendering doet, zullen we in een functie update() plaatsen. Definieer deze update functie in je Particle class:

```
update() {
  this.lifespan--;
  this.velocity.add(this.acceleration);
  this.location.add(this.velocity);
}
```

Wis deze 3 lijnen code uit de draw functies van zowel je Particle als SmokeParticle class. Voer tenslotte de update functie uit vanuit je script.js:

```
particles = particles.filter(particle => particle.isAlive);
particles.forEach(particle => particle.update());
particles.forEach(particle => particle.draw(ctx));
```

Zo'n update-draw loop zul je vaker tegenkomen. Zo kunnen bvb botsende elementen (bvb bij het raken van een vijand met een kogel) na de update van de display objects gedetecteerd worden en vermeden worden dat deze nog getekend zullen worden.

Particle Forces

We zullen de smoke wat realistischer maken door forces uit te voeren op de particles.
Stel in de particle class de initiële acceleration in op 0:

```
this.acceleration = new Vector(0, 0);
```

In de update functie doe je op het einde een reset van deze vector:

```
this.acceleration.mult(0);
```

Voeg een applyForce functie toe aan de Particle class, om een vector toe te voegen aan de acceleration:

```
applyForce(force) {
  this.acceleration.add(force);
}
```

In je main draw loop, plaat je nu de particles onderaan het scherm, en voer je een force uit op de particles:

```
let wind = new Vector(0, -0.05);
particles.forEach(particle => particle.applyForce(wind));
```

Kijk nu om de x-richting van de wind vector te mappen aan de muis-x-coordinaat, zodat de smoke weggeblazen wordt door de muiscursor:

Aangezien je regelmatig eens een waarde moet re-mappen, kun je hiervoor een functie toevoegen aan je lib.js:

```
export const map = (value, istart, istop, ostart, ostop) => {
  return ostart + (ostop - ostart) * ((value - istart) / (istop -
istart));
};
```

En deze gebruiken voor de berekening van de x-component van de vector:

```
wind.x = map(mouse.x, 0, canvas.width, 0.2, -0.2);
```