



RÉMI RUCOJEVIC

Creative web developer

EDITION

2018 -2019

SOMMAIRE

MES PROJETS :

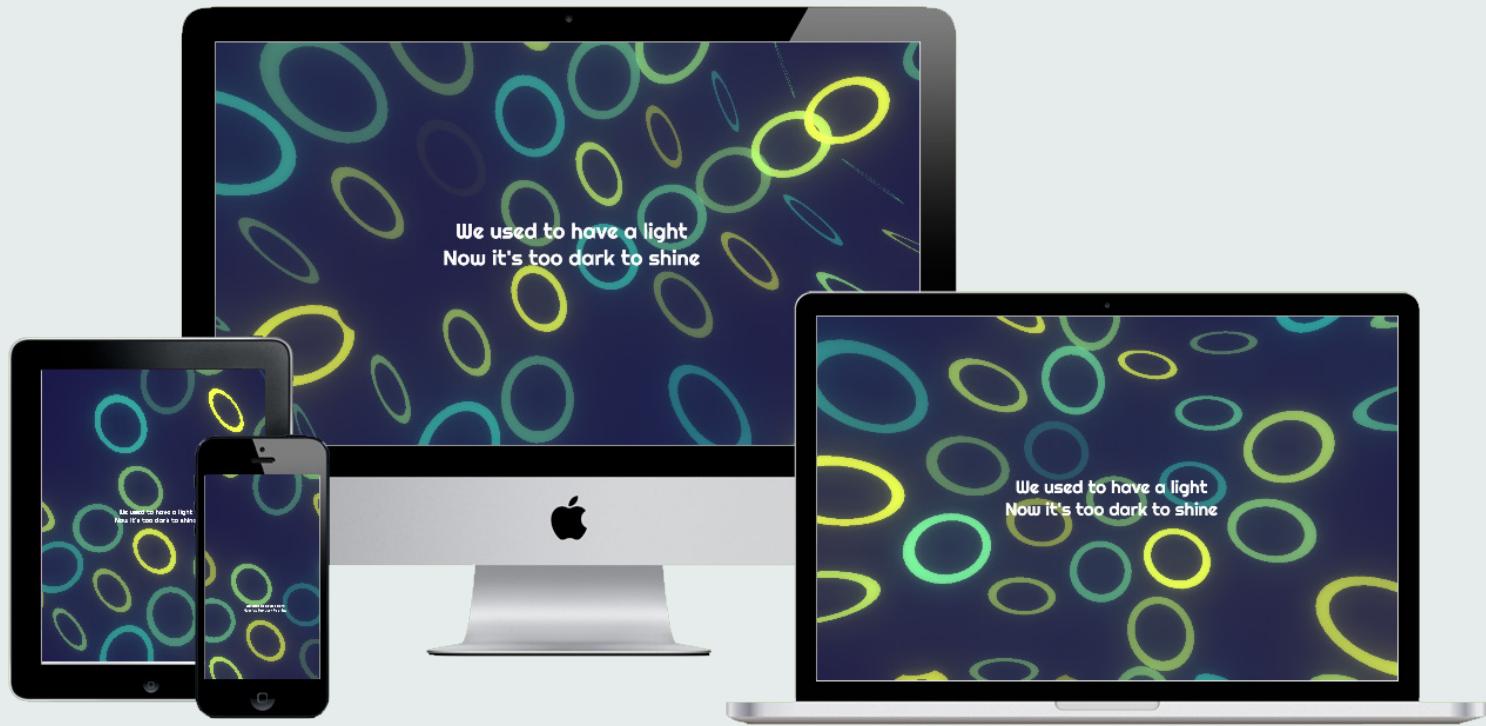
- ◆ Chasing Ghost **WebGL expérience** 4
- ◆ Yamanote Line **Simulateur de train**..... 8
- ◆ Co2 Planète **Dataviz WebGL** 12
- ◆ JapanNi **Site web vitrine** 16
- ◆ Otium **Application web** 20

PORTFOLIO EN LIGNE :

[HTTPS://REMIRUC.COM](https://remiruc.com)

CHASING GHOST

WEBGL EXPÉRIENCE



Fiche technique

Novembre 2018

Chasing ghost est une expérience auditive et visuelle 3d réalisée en 2 jours en utilisant la librairie Three.js. Basée sur la musique «Chasing Ghost» du groupe Against The Current, les visuels sont entièrement créés à l'aide de Three.js et réagissent à la musique (fréquence, tempo...)

Les objectifs principaux de ce projet étaient de pratiquer le **WebGL**, en utilisant **Three.js**, ainsi que d'utiliser l'**Audio API** native de Javascript. Ce projet m'a aussi permis de m'initier à **l'animation** sur navigateur.

Outils utilisés



HTML 5



CSS 3



Javascript



Three.js



Audio API



Un peu de 3d...

Toute l'expérience tourne autour d'une techno principale : Three.js. Cette techno permet d'utiliser plus simplement le WebGL, permettant lui-même de faire de la 3d sur navigateur. La totalité des visuels sont créés à l'aide des fonctions fournies par Three.js (Sphere, Cercle...). Aucun assets n'a donc été créé sur un logiciel externe.

Une partie de la Class «SpotLight» qui automatise la création de chaque objet 3D. Un SpotLight correspond à un anneau + un triangle

```
//init
this.render = new THREE.Object3D()
//Ring
var geometry = new THREE.RingGeometry( 1, 1.3, 32, 1, 0 )
var material = new THREE.MeshBasicMaterial( {color: "rgb("+this.ringColor.r+", "+this.ringColor.g+", "+this.ringColor.b+")" } )
this.render.add(new THREE.Mesh( geometry, material ))
//Arrow
var geometry = new THREE.CircleGeometry( 1, 1, this.thetaStart );
var material = new THREE.MeshBasicMaterial( {color: "rgb("+this.arrowColor.r+", "+this.arrowColor.g+", "+this.arrowColor.b+")" } )
this.render.add(new THREE.Mesh( geometry, material ))
this.render.position.set(this.x, this.y, this.z)
this.render.lookAt(new THREE.Vector3( 0, 0, this.z ))]
```



...liée à de la musique

Pour apporter plus de diversité, les «SpotLights» réagissent tous en fonction de la musique. A l'aide de l'audio API de Javascript, il est possible de récupérer les fréquences du son qui est actuellement joué. Il suffit ensuite de les lier comme on le souhaite aux «SpotLights». Dans notre cas, chaque «SpotLights» est lié à une fréquence, et c'est son opacité et sa taille qui sont affectées selon la puissance de sa fréquence.

```
analyser.getByteFrequencyData(frequencyData)

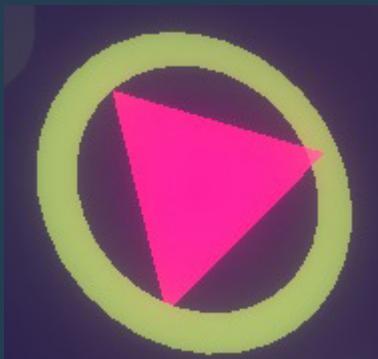
spotArr.forEach(el=>{
    let newOpacity = map(frequencyData[el.frequencyLink], 0, 255, 0,1)
    let newRayon = map(frequencyData[el.frequencyLink], 0, 255, 12,8)
    el.update(newOpacity, newRayon)
})
```

Une Class pour éviter la duplication de code

Comme chaque «SpotLight» se ressemble mais sont chacun indépendant les uns des autres, la création d'une Class permettant d'automatiser leur création puis leur réaction était une solution qui évitait notamment la duplication de code ainsi et facilitait la mise à jour en temps réel de chaque «SpotLight»

```
constructor(x, y, z, thetaStart, rayon, frequencyLink, column, row, arrowColor, ringColor){  
    this.x=x  
    this.y=y  
    this.z=z  
    this.column = column  
    this.row = row  
    this.arrowColor = arrowColor  
    this.ringColor = ringColor  
    this.thetaStart = thetaStart  
    this.rayon = rayon  
    this.frequencyLink = frequencyLink  
    this.opacity = 1
```

Une partie de la Class «SpotLight»



Résultat d'un Objet de la Class «SpotLight»

```
var wall = new THREE.Object3D()  
var spotArr = []  
let gradient = 0  
for (let column = 0; column < numberColumn; column++) {  
    for (let row = 0; row < numberRow; row++) {  
        let x = Math.cos(-row * Math.PI/(numberRow/2)) * (circleRayon);  
        let y = Math.sin(-row * Math.PI/(numberRow/2)) * (circleRayon);  
        let thetaStart  
        if (row%2==0) {  
            thetaStart = 0  
        } else {  
            thetaStart = 3  
        }  
        if (column ==0) {  
            console.log(thetaStart)  
        }  
        let arrowColor = {  
            r: Math.floor(map(gradient,0, numberRow*5,246,232)),  
            g: Math.floor(map(gradient,0, numberRow*5,0,0)),  
            b: Math.floor(map(gradient,0, numberRow*5,3,247)),  
        }  
        let ringColor = {  
            r: Math.floor(map(gradient,0, numberRow*5,192,13)),  
            g: Math.floor(map(gradient,0, numberRow*5,192,126)),  
            b: Math.floor(map(gradient,0, numberRow*5,19,106)),  
        }  
        if (gradient>=numberRow*5) {  
            gradient = 0  
        } else {  
            gradient++  
        }  
        let aleatoire = rand(10,800)  
        let geometry = new SpotLight(x,y,-column*3, thetaStart, circleRayon  
            ,aleatoire, column, row, arrowColor, ringColor)  
        spotArr.push(geometry)  
        wall.add(geometry.render)  
    }  
}
```

La boucle qui permet la création du mur de «SpotLight» entier.



Shader et Post-processing en WebGL

Pour créer cet effet fluorescent, il a fallu appliquer sur l'ensemble de la scène 3D une sorte de filtre sur l'image rendu. Cet effet a été réalisé à l'aide des possibilités de Post-processing offertes par Three.js

```
//Post processing
var renderScene = new THREE.RenderPass( scene, camera );
var bloomPass = new THREE.UnrealBloomPass( new THREE.Vector2( window.innerWidth, window.innerHeight ), 1.5, 0.4, 0.85 );
bloomPass.renderToScreen = true;
bloomPass.threshold = 0;
bloomPass.strength = 0.8;
bloomPass.radius = 1;

composer = new THREE.EffectComposer( renderer );
composer.setSize( window.innerWidth, window.innerHeight );
composer.addPass( renderScene );
composer.addPass( bloomPass );
```

Mise en place du post-processing



Résultat sans Post-Processing



Résultat avec Post-Processing



YAMANOTE LINE

SIMULATEUR DE TRAIN



Fiche technique

Janvier 2019

Yamanote Line est une application web utilisable sur ordinateur et mobile. C'est un simulateur de métro, la «Yamanote Line», une des lignes de métro de la ville de Tokyo. Il est possible à travers cette application d'embarquer dans le train depuis l'une des 29 stations et de se laisser bercer par les sons atypiques de cette ligne de métro.

Les objectifs principaux de ce projet étaient de créer une application web interactive et responsive ainsi que de créer une expérience qui dépend du temps écoulé / passé sur l'application.

Outils utilisés



HTML 5



CSS 3



Javascript

Une application web en Javascript natif

Dans une simple envie de progresser en Javascript, j'ai décidé pour ce projet de ne pas utiliser de framework front, mais d'utiliser du Javascript pur. Le but est simple : Il y a un train, et celui-ci se balade à travers les 29 (bientôt 30) stations de la ligne. Le son et l'affichage du site agit en fonction d'où se trouve le train : Entre 2 stations, à une station particulière....



1 Class -> 29 stations

La Yamanote Line est composée de 29 stations (bientôt 30). Ayant chacune le même comportement avec des informations différentes (Nom, distance, sons en gare...), j'ai créé une Class «Station» permettant d'automatiser plusieurs actions.

```
class Station{  
    constructor(names, numberStation, position, distanceToLastStation, distanceToNextStation, melody){  
        this.names = names  
        this.numberStation = numberStation  
        this.position = position  
  
        this.melodyName = melody  
  
        this.annoucement = new Audio("/sound/announcement/"+names[0]+".mp3")  
        this.melody = [new Audio("/sound/melodies/"+melody[0]+".mp3"), new Audio("/sound/melodies/"+melody[1]+".mp3")]  
        this.melody.forEach(el=>{  
            el.volume = 0.6  
        })  
  
        this.distanceToStation = [distanceToLastStation, distanceToNextStation]  
    }  
}
```

Constructeur de la Class «Station»

```

playAnnoucement(){
    this.annoucement.play()
}

playMelody(direction){
    this.melody[direction].play()
}

stopSounds(){
    this.annoucement.pause()
    this.annoucement.currentTime = 0
    this.melody.forEach(el => {
        el.pause()
        el.currentTime = 0
    })
}
}

```

Méthodes de la Class «Station»

```

new Station(["Tokyo","東京","とうきょう","TYO"], 1, 27.7, 1.3, 0.8,["JR SH-3", "JR SH-3"])
new Station(["Kanda","神田","かんだ","KND"], 2, 26.4, 0.7, 1.3,["JR Babble", "JR Babble"])

```

Création des stations Tokyo et Kanda

1 Class -> 1 train

Si les Class sont utiles pour automatiser des actions répétitives (ex ici : création de stations), les Class le sont tout autant lorsqu'une seule instance d'elle-même est créée. Il y a beau n'y avoir qu'un seul train par session d'utilisateur, celui-ci, pour fonctionner correctement, a besoin de calculer de nombreuses informations différentes afin de savoir où il est, quelles informations il doit afficher, où doit-il se rendre... Une seule instance de Train va donc être créée, mais celle-ci va contenir de multiples données et méthodes. Le regroupement en un objet unique aide beaucoup à son utilisation.

```

class Train{
    constructor(stations){
        this.speed = 0.0
        this.speedDisplayed = 0.0
        this.velocity = 0.020
        this.position = 0.0
        this.maxSpeed = 90

        this.stations = stations
        this.nameToScreen = 1

        this.positionBetweenStation = 0

        this.lastStation
        this.actualStation
        this.nextStation
        this.thenStation
        this.thenThenStation
    }
}

```

Une infime partie de la Class Train. Celle-ci contient en réalité le triple de lignes dans son constructeur et contient au total 12 méthodes différentes. La totalité de la Class est consultable à l'adresse suivante : <https://yamanoteline.remiruc.com/js/train.js>



Un affichage géré en Javascript

Pour éviter de gérer des données de «jeu» en Javascript d'un côté et de gérer l'affichage en HTML de l'autre, j'ai décidé de gérer l'affichage des stations en Javascript. De ce fait, dès qu'un nouvel objet Station est créé ou supprimé, la station est automatiquement ajoutée ou supprimée du menu. Cette manière de faire permet aussi de gérer la lecture ou non des sons de chaque station.

```
for (let i = 0; i < stations.length; i++) {
    let li = document.createElement("li")

    let div = document.createElement("div")
    div.classList.add("station")

    div.dataset.index = i

    let numberInformation = document.createElement("div")
    numberInformation.classList.add("number-information")

    let shortName
    if (stations[i].names[3] === undefined) {
        shortName = ""
    } else {
        shortName = stations[i].names[3]
        numberInformation.innerHTML =
            `<div class="black-container">
                <p>`+shortName+`</p>
            </div>`
    }

    let number
    if (stations[i].numberStation < 10) {
        number = "0" + stations[i].numberStation
    } else {
        number = stations[i].numberStation
    }
    numberInformation.innerHTML +=

    `<div class="white-container">
        <p class="jy">JY</p>
        <p class="number">`+number+`</p>
    </div>
    `

    div.appendChild(numberInformation)

    let p = document.createElement("p")
    p.innerHTML = "<span>" + stations[i].names[1] + "é...</span><br>" + stations[i].names[0] + " Sta."
}
```

Une partie de la boucle qui crée l'affichage pour chaque station à l'aide de multiples utilisations de la méthode createElement(). Cette boucle agit un peu comme un template.

Les objets Stations affichées dynamiquement en HTML grâce à la boucle du dessus



CO2 PLANÈTE

DATAVIZ WEBGL



Fiche technique

Septembre 2018

Co2 Planète est une dataviz (visualisation de données) interactive en 3d permettant de visualiser à différentes périodes le taux d'émission de Co2 de certains pays.

L'objectif principal de ce projet est de s'initier à la Dataviz et de pratiquer le WebGL à l'aide de Three.js.

Outils utilisés



HTML 5



CSS 3



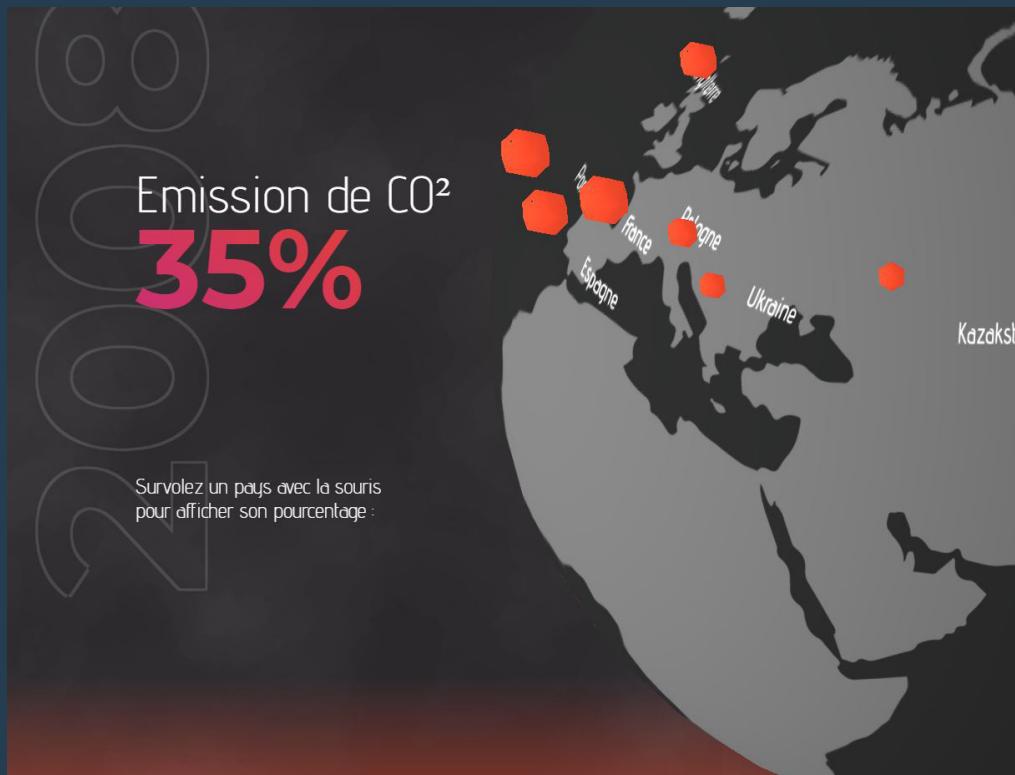
Javascript



Three.js

Un affichage 3D en fonction des données

Le thème de la dataviz, les émissions de CO₂, nous a orienté vers la solution suivante : Créer une représentation de la Terre en 3D, et apposer dessus différents points sur chaque pays représentant l'émission de CO₂ par pays. Les différents points du globe changent de tailles selon la valeur des données récoltées.



Les points rouges sont de tailles différentes selon la valeurs des données de chaque pays.

Partie du code qui permet le redimensionnement des points rouges

```
actualData.forEach(element => {
    for (let i = 0; i < sphereTab.length; i++) {
        if(sphereTab[i].name === element[1]){
            var index = i;
        }
    }
    var maxA = 0;
    var minA = 0;
    var maxB = 0.5;
    var minB = 0.2;
    for (let i = 0; i < actualData.length; i++) {
        if (actualData[i][0] < minA) {
            minA = actualData[i][0];
        }
        if (actualData[i][0] > maxA) {
            maxA = actualData[i][0];
        }
    }
    if (typeof(sphereTab[index]) !== "undefined") {
        var pourc = ((element[0] - minA) / (maxA - minA)) * (maxB - minB) + minB;
        sphereTab[index].scale.set(pourc,pourc,pourc);
    }
});
```

Une map interactive

Dans un souci de praticité, la map en 3d est manipulable à l'aide de la souris. On peut la faire défiler en cliquant et bougeant la souris ou encore surligner les informations qui nous intéressent sur la map pour voir la valeur numérique s'afficher à côté.

```
// update the picking ray with the camera and mouse position
raycaster.setFromCamera( mouse, camera );

// calculate objects intersecting the picking ray
var intersects = raycaster.intersectObjects( object.children );
if (intersects.length>0) {
    if (intersects[0].object.name !== "") {
        console.log(intersects[0].object.name);
        document.body.style.cursor = "pointer";
        dataToUse.forEach(element => {
            if (element.CountryCode === intersects[0].object.name) {
                console.log(element);
                var num = parseInt(element[actualYear]);
                document.getElementById("countryCo2Pourcentage").innerHTML = num + "%";
                document.getElementById("countryName").innerHTML = element["CountryName"];
            }
        });
    } else {
        moveRoation[1] += 0.002;
        object.rotation.x += (moveRoation[0] - object.rotation.x)*0.1;
        object.rotation.y += (moveRoation[1] - object.rotation.y)*0.1;
        document.body.style.cursor = "auto";
    }
} else {
    moveRoation[1] += 0.002;
    object.rotation.x += (moveRoation[0] - object.rotation.x)*0.1;
    object.rotation.y += (moveRoation[1] - object.rotation.y)*0.1;
    document.body.style.cursor = "auto";
    document.getElementById("countryCo2Pourcentage").innerHTML = "";
    document.getElementById("countryName").innerHTML = "";
}
```

Dans une requestAnimationFrame et après avoir récupéré les coordonnées de la souris, on agit sur la rotation de la map : Soit il n'y a aucune action de la part de l'utilisateur et elle tourne toute seule, soit l'utilisateur drag and drop sur la map et alors son mouvement de souris fait tourner la terre comme il le souhaite.

Couvrant 28,1 % des besoins énergétiques mondiaux en 2015, le charbon est la seconde ressource énergétique de l'humanité, derrière le pétrole...

Continuer

Un storytelling et une musique de fond a également été ajouté à l'expérience afin d'immerger encore plus l'utilisateur à travers l'expérience.

JAPAN NI

SITE WEB VITRINE



Fiche technique

Février 2019

JapanNi est un site vitrine perso qui regroupe mes photos prises lors de mes voyages au Japon à travers différentes villes. On peut sélectionner une ville en particulier sur une carte et y voir plusieurs photos.

Les objectifs principaux de ce projet étaient de s'initier au framework **Vue.JS** et de tester l'interactivité sur des images de format **SVG**.

Outils utilisés



HTML 5



CSS 3



Javascript



Vue.JS



Une application web avec VueJS

VueJS est un des framework front les plus utilisés du moment. Il facilite de nombreuses choses au niveau du développement web : Facilité à créer des Progressiv Web App (PWA), création de components réutilisables, gestion des views dynamiques...

```
<template>
  <div id="app">
    <TheHeader></TheHeader>
    <transition name="fade" appear>
      <router-view/>
    </transition>
  </div>
</template>
```

Un simple HTML liant components et router permettant de gérer dynamiquement l'application.

Un router et des components dynamiques

Vue permet de créer des routes dynamique. Il peut lire l'URL et afficher un contenu différent selon l'URL utilisé. Dans notre cas, nous avons plusieurs villes que nous souhaitons afficher. Les pages ont le même comportement, la même structure, mais les informations affichées dessus sont différentes selon la ville. C'est là que la création d'un component intervient.

```
// Affichage des données
<template>
  <div class="view">
    <Background :bg="this.cities[$route.params.city]['path']"></Background>
    <div class="content">
      <h2 class="title">{{$route.params.city}}<span>{{this.cities[$route.params.city]['kanji']}}</span></h2>
      <div class="text">
        <p v-for="content in this.cities[$route.params.city]['content']" :key="content"> {{content}} </p>
      </div>
      <div class="photos">
        <a target="_blank" v-for="i in this.cities[$route.params.city]['photosNum']"
          :key="i" :href="'/static/'+$route.params.city+'/img ('+i+').JPG'">
          
        </a>
      </div>
    </div>
  </div>
</template>

// Récupération des données des données
<script>
export default {
  name: "City",
  computed: {
    cities: () => dataCities.cities,
  }
};
</script>
```

Le component City, qui permet l'affichage de la page d'une ville selon celle qui a été sélectionnée



Une carte en SVG interactive

L'avantage d'une image SVG est le fait qu'elle est codée en XML, un langage de balise comme le HTML. Elle est donc facilement réutilisable de la même manière qu'une balise HTML. Dans notre cas, j'ai créé et intégré une carte du Japon en SVG. Chaque point en rouge, qui est de base une forme sur Adobe Illustrator, devient un lien cliquable qui redirige vers la ville en question.



La carte SVG



Une des villes sur la carte est «hover» par la souris

```
<circle class="cls-1" cx="268.99" cy="522" r="5"/>
<circle class="cls-1" cx="279.99" cy="522" r="5"/>
<router-link exact :to="{name: 'City', params:{city:'kyoto'}}" id="kyotoGroup">
  <circle id="kyoto" class="cls-1 link" cx="290.99" cy="522" r="5"/>
</router-link>
<circle class="cls-1" cx="302.01" cy="522" r="5"/>
<circle class="cls-1" cx="313" cy="522" r="5"/>
<router-link exact :to="{name: 'City', params:{city:'nagoya'}}" id="nagoyaGroup">
  <circle id="nagoya" class="cls-1 link" cx="324" cy="522" r="5"/>
</router-link>
<circle class="cls-1" cx="335" cy="522" r="5"/>
<circle class="cls-1" cx="346" cy="522" r="5"/>
<circle class="cls-1" cx="159" cy="532.99" r="5"/>
<circle class="cls-1" cx="170" cy="532.99" r="5"/>
```

Une partie du SVG généré. Chaque balise `<circle>` correspond à un cercle sur la carte. Certains sont entourés par une balise `<router-link>`, ce sont les liens vers les différentes villes. Elles deviennent rouges sur la carte.



Japan Ni

TOSHIBA SHARP Parcourir À propos

NEC HITACHI SONY

SHIBUYA LABI SHINJUKU

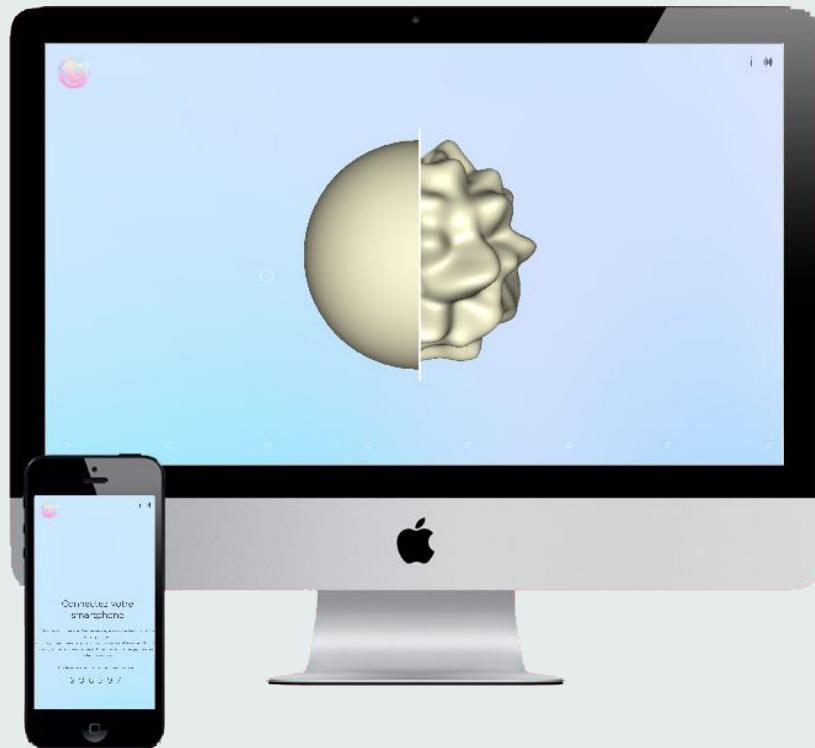
TOKYO

Tokyo est la capitale du Japon. Il y a tellement de choses à faire là-bas, chaque quartier est différent l'un de l'autre.

Une fois sélectionnée, l'URL change et la ville choisie est affichée. C'est le router qui lit la nouvelle URL et envoie au component «City» le nom de la ville. C'est ensuite le component qui se charge d'afficher les bonnes informations.

OTIUM V.1

APPLICATION WEB



Fiche technique

Janvier - Juin 2019

Otium est mon projet de fin d'année Gobelins 2019 : c'est un site ayant pour vocation d'aider ses utilisateurs à se relaxer. Muni de son smartphone (faisant office de manette) et de son ordinateur (faisant office d'écran), l'utilisateur est invité à répondre à 8 questions concernant son état général, et ce, de manière ludique, en utilisant les capacités de son smartphone (gyroscope...). A la suite de ce questionnaire, un univers en 3d est généré selon les réponses qui auront été données. L'utilisateur peut alors ensuite parcourir cet univers personnalisé dans le but de se relaxer.

Les objectifs principaux de ce projet étaient de créer une application cross-plateformes en liant ordinateur et smartphone en même temps à l'aide de **Socket.IO**.

Outils utilisés



HTML 5



CSS 3



Javascript



Three.js



Node.js

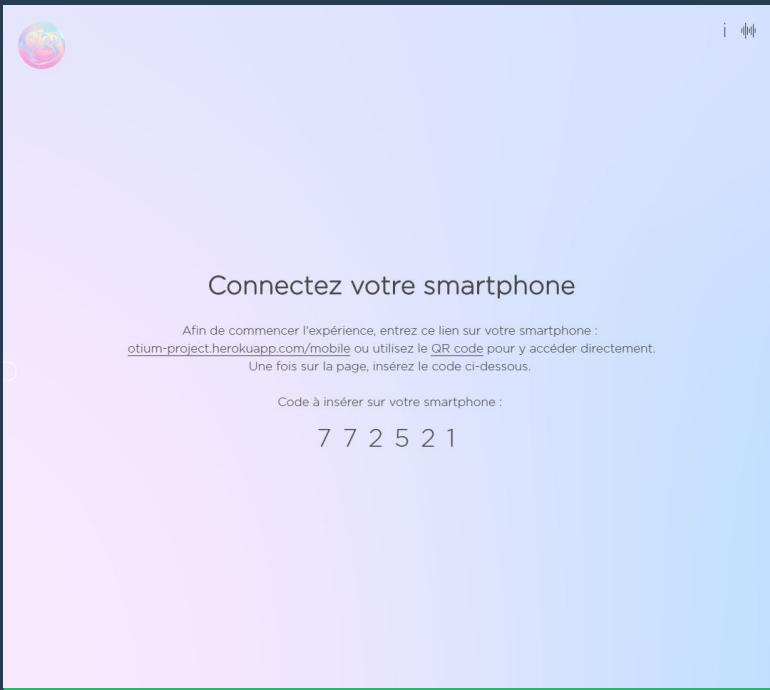


Socket.io

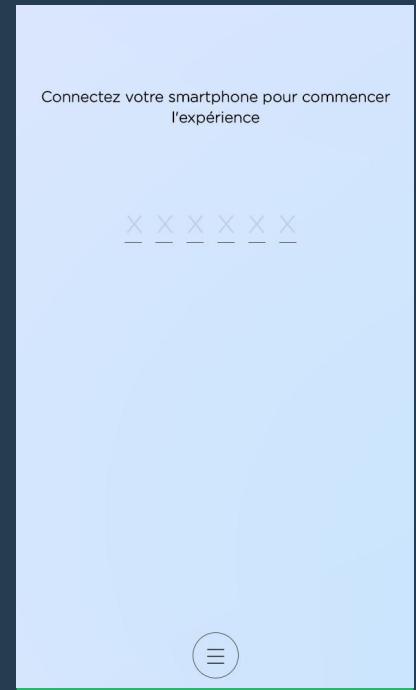


Une application cross-plateformes

La particularité d'Otium réside dans son côté cross-plateformes. Pour utiliser l'application, il faut connecter un ordinateur et un smartphone. Dans notre projet, nous le faisons à l'aide de Socket.IO, une technologie Javascript permettant de faire du temps réel à travers un serveur Node.js. Un ordinateur crée une «Room» et génère un code. Un smartphone peut entrer dans une «Room» en entrant le code. Une fois les deux périphériques connectés entre eux, la room est fermée et plus personne ne peut la rejoindre. L'ordinateur et le smartphone sont connectés.



L'ordinateur génère un code



Le smartphone entre le code pour se connecter avec l'ordinateur

```
// Connexion d'un ordinateur
socket.on('computerConnexion', function (id) {
  console.log(id)
  socket.serverId = id.id
  computers.push( socket.serverId.toString() )
  socket.join(socket.serverId.toString())
  socket.broadcast.emit('getRoom', {computers})
})

// Tentative de connexion d'un smartphone
socket.on('askMobileConnexion', (code) => {
  let i = computers.indexOf( code )
  if (i > -1) {
    if (io.sockets.adapter.rooms[code].length < 2) {
      console.log(io.sockets.adapter.rooms[code].length)
      socket.clientId = code
      socket.join(code)
      io.sockets.to(code).emit('mobileConnected')
      console.log('connected')
    }
  }
})
```

Une partie du code contenu dans le serveur Node permettant de gérer la connexion d'un ordinateur et d'un smartphone.



Des questions générées à l'aide de Class

Les 8 questions étant répétitives au niveau de leur fonctionnement, nous avons créé une Class «Page» permettant de gérer chaque question les unes après les autres. À chaque changement de page, un socket.emit est envoyé à l'autre périphérique connecté afin que les deux appareils soient tous les deux synchronisés et restent toujours à la même étape que l'autre périphérique.

How to use Page class

Rémi Rucojevic edited this page on 31 Mar · 4 revisions

Methods

PAGE_TO_DISPLAY.displayPage('DEVICE_NAME')

Use this method to display a page when there is nothing on the DOM (Use for example when the user open the site)

```
intro.intro.displayPage('desktop') // Display the intro.intro page
```

ACTUAL_PAGE.transitionTo('DEVICE_NAME', NEXT_PAGE)

Use this method to switch from a page to another (Use for example to switch from a question to another)

```
intro.intro.transitionTo('desktop', intro.connexion) // Switch from intro.intro to intro.conn
```

PAGE_TO_GO.comeBackTo('DEVICE_NAME')

Use this method to cancel all the navigation and go back to a defined page. (Use for example when the other device is disconnected, to go back to main menu)

```
intro.intro.comeBackTo('desktop') // Cancel everything and display the intro.intro page
```

Documentation sur GitHub résumant comment utiliser la Class Page dans notre projet.

```
// Le périphérique actuel change de page...
ValidationBtn.actualPage.transitionTo("mobile", ValidationBtn.nextPage)
// ... et notifie l'autre appareil pour qu'il en fasse de même
socket.emit("validationQuestion", {from: ValidationBtn.actualQ, to: ValidationBtn.nextQ})
```

Le périphérique actuel change de page et notifie l'autre appareil à l'aide d'un socket.emit pour qu'il en fasse de même

Contrôler l'ordinateur avec le smartphone

Une fois connectés entre eux, c'est le smartphone qui va devenir le contrôleur de l'ordinateur. Chaque question utilise une possibilité du smartphone (gyroscope, écran tactile, accéléromètre...) afin de répondre sur l'ordinateur à une question.



```

// Le desktop actualise son affichage lorsque le smartphone réagit
socket.on("q1", (eventData) => {

    if (eventData.tiltLR >= -50 && eventData.tiltLR <= 50) {
        document.getElementById("forme-net").style.width = 50 + eventData.tiltLR + "%";
        document.getElementById("forme-abstraite").style.width = 50 - eventData.tiltLR + "%";

        document.getElementById("forme-net").style.transition = '1.5s';
        document.getElementById("forme-abstraite").style.transition = '1.5s';

        window.resultats.setResult("q1", { res: eventData.tiltLR })
    }
}

// Le smartphone prévient l'ordinateur que l'utilisateur a fait une action
function deviceOrientationHandler(eventData) {
    if (ValidationBtn.touch === false && window.getComputedStyle(document.querySelector(".gifValidation")).getPropertyValue('opacity') == 0) {
        socket.emit("q1", { tiltFB: eventData.beta, tiltLR: eventData.gamma, dir: eventData.alpha });
    }
}

```

L'ordinateur attend un Socket.on pour actualiser son affichage. Celui-ci lui est envoyé par le smartphone dès que l'action correspondante à la question (ici le changement d'orientation du téléphone) est activée.

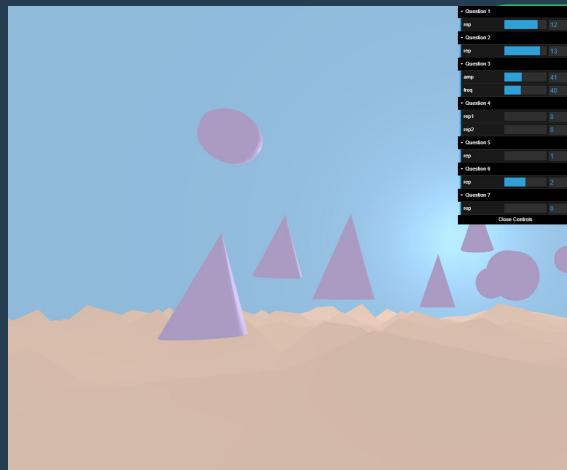
Des réponses qui génèrent un univers 3D

Toutes ces questions posées ont pour but final de générer un univers 3D. Chaque question va agir sur un élément du décors (par exemple le relief, le moment de la journée, les objets présents...).

```

let params = {
    q1: {
        rep: 2
    },
    q2: {
        rep: 2
    },
    q3: {
        amp: 15,
        freq: 60
    },
    q4: {
        rep1: 0,
        rep2: 0
    },
    q5: {
        rep: 1
    },
    q6: {
        rep: 2
    },
    q7: {
        rep: 0
    },
}

```



Les réponses aux questions sont stockées...

...puis utilisées dans le code pour générer l'univers (ici le relief au sol)

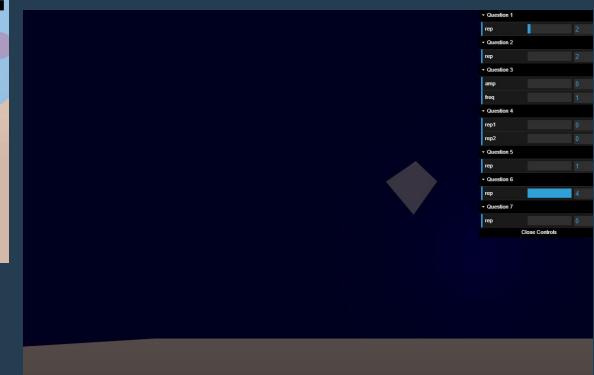
```

for (var i = 0; i < geometryPlane.vertices.length; i++) {
    geometryPlane.vertices[i].z += Math.random() * params.q3.amp - params.q3.amp;
    geometryPlane.vertices[i]._myZ = geometryPlane.vertices[i].z
    plane.geometry.verticesNeedUpdate = true;
}

```

Ex 1 : Un univers de jour avec un relief montagneux et beaucoup d'objets.

Ex 2 : Un univers de nuit avec un relief plat et peu d'objets.



ÇA NE S'ARRÊTE PAS LÀ ! POUR EN VOIR PLUS
N'HÉSITEZ PAS À VISITER MON PORTFOLIO EN
LIGNE OU À ME SUIVRE SUR LES RÉSEAUX
SOCIAUX

HTTPS://REMIRUC.COM



@remiruc



/in/rémi-rucojevic



@remiruc



RemiRuc

**EN VOUS REMERCIANT D'AVOIR PRIS LE
TEMPS DE REGARDER CE BOOK.**

[HTTPS://REMIRUC.COM](https://remiruc.com)

