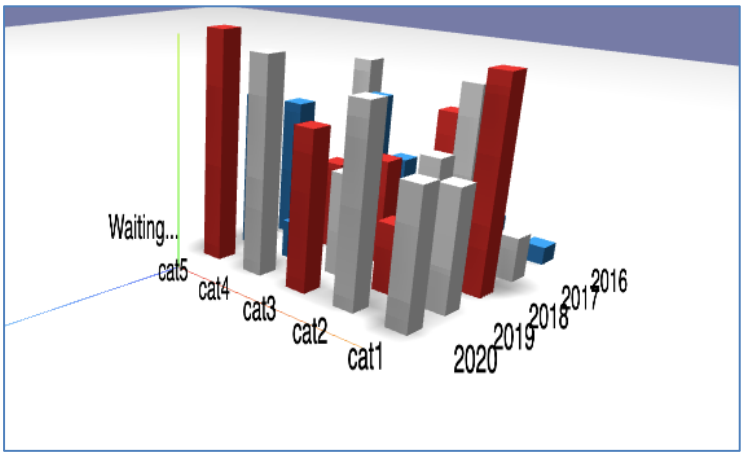
TP3 - THREE.JS

Dans ce TP vous allez réaliser une nouvelle application qui utilise les différentes notions vues pour visualiser des données.
Vous pouvez recopier vos fichiers html et js, en nettoyant la fonction createContent. Nous allons utiliser les bibliothèques suivantes :

```
<script src="./libs/three.min.js"></script>
<script src="./libs/OrbitControls.js"></script>
<script src="./libs/TweenMax.min.js"></script>
<script src="./libs/DAT.GUI.min.js"></script>
<script src="graph.js"></script>
```

Voici l'objectif :



Au lieu d'une skybox, vous pouvez utiliser une couleur pour le renderer grâce à l'instruction :
renderer.setClearColor(0x7777aa);

LE PLAN

Nous allons commencer pour créer le plan sur lequel les autres éléments seront placés. Pour cela, on s'en servira de la géométrie PlaneBufferGeometry et d'un matériau que vous pouvez évidemment changer à votre guise :

```
function createFloor() {
   var material = new THREE.MeshPhongMaterial({ color: 0xcccccc, shininess: 20  });
   material.side = THREE.DoubleSide;

  var planeGeo = new THREE.PlaneBufferGeometry(100,100,32);
  var planeMat = new THREE.MeshLambertMaterial(0xffffff)
  var plane = new THREE.Mesh(planeGeo,material);
  plane.rotation.x = -.5 * Math.PI;
  plane.receiveShadow = true;
    scene.add(plane);
  }
```

Testez votre script.

LES DONNEES

Avant de créer les barres nous devons avoir des données. Dans ce cas, elles seront simplement simulées, et leur hauteur sera calculée de manière aléatoire. Nous commençons par créer les points X, Y, Z, indiquant combien de barres seront placées dans chaque coordonnée Z ainsi que le début du placement et l'espace entre elles. Pour l'instant, la hauteur (coordonnée Y) reste à 1 pour tous les éléments.

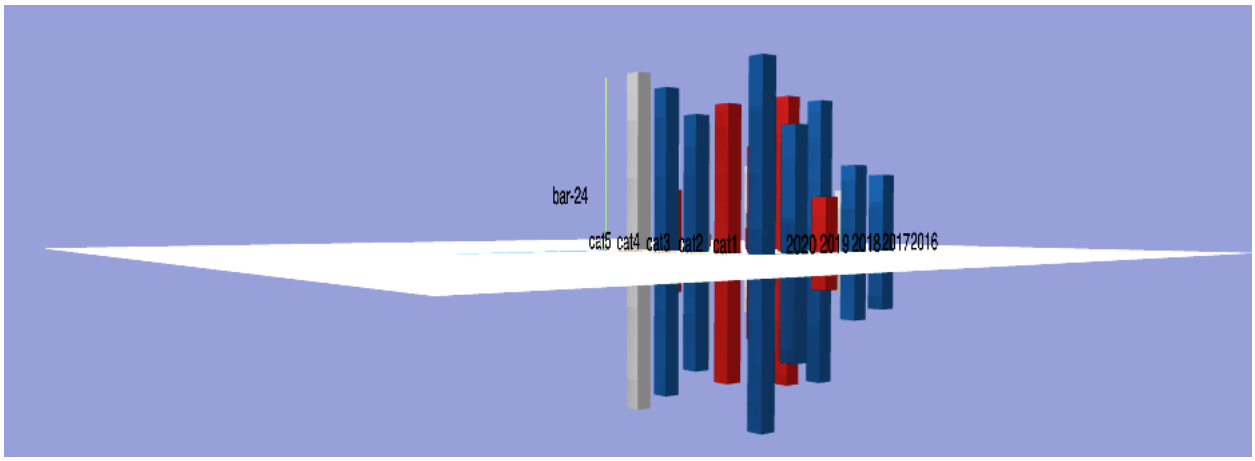
```
function initData() {
  var nbBarsX = 5;  // number of objets for each Z
  var startZ = -25; // position of first Z line
  var nbZ = 5;
  var espacmnt = 5;
  var z = startZ;

  for (var i = 0; i < nbZ; i += 1) {
    for (var j = 0; j < nbBarsX; j += 1) {
      data.push([j * espacmnt, 1, z]);
    }
    z += espacmnt;
  }
}
```

CONSTRUCTION DU DIAGRAMME

Une fois les données disponibles, nous pouvons passer à la construction du diagramme. Nous allons utiliser des boites, avec un matériau simple où la couleur est choisie aléatoirement parmi l'ensemble colors. Chaque barre a un nom qui correspond à sa position dans le vecteur de données. Dans la deuxième partie, on trouve aléatoirement une valeur pour la coordonnée Y entre 0 et 10 et on réalise une animation entre la valeur 1 donnée au début et la nouvelle valeur.

```
function createBars() {
  var geometry;
  var colors = [0xff0000, 0x1176c5,0xf9f9f9];
  var geometry = new THREE.BoxBufferGeometry(barSize, barSize, barSize);

  for (var i = 0; i < data.length; i++) {
   var material = new THREE.MeshPhongMaterial({ color: colors[0]});
   material.color.setHex(colors[Math.floor(Math.random() * 3)]);
   var obj = new THREE.Mesh(geometry, material);
   obj.position.set(data[i][0],data[i][1],data[i][2]);
   obj.name = "bar-" + i;
   obj.castShadow = true;
   obj.receiveShadow = true;
   scene.add(obj);
   bar.push(obj);
  }
  // Animate Y value
  for (var i = 0; i < bar.length; i++) {
   var tween = new TweenMax.to(bar[i].scale, 1, { ease: Elastic.easeOut.config(1, 1),  y: Math.random() * 10,
     delay: i * 0.25
   });
  }
 }
```

Testez votre script.



Corrigez le script pour éviter que les barres dépassent du sol.

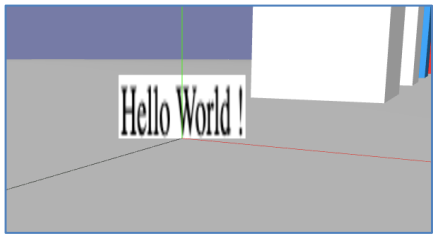
LE PANNEAU

Créez les variables globales suivantes :

```
// tooltip
var fontSizeInit = 40;
var sprite1;
var canvas1, ctx, texture1;
```

Elles nous permettront d'ajouter le panneau de texte principal (qui affiche le nom de l'objet pointé) et ensuite des étiquettes pour les axes. Les panneaux seront créés grâce à l'objet Sprite, qui permet d'avoir un objet qui est toujours tourné vers l'utilisateur (« Billboard »). Nous allons utiliser également un élément canvas pour réaliser le rendu du texte.

```
function createTooltip() {
    // draw text on canvas
    var fontface = "Helvetica";
    var fontsize = fontSizeInit;
    var message = "Hello World";

    canvas1 = document.createElement('canvas');
    ctx = canvas1.getContext('2d');
    ctx.font = fontsize + "px " + fontface;
    ctx.textBaseline = 'top';

    // text color
    ctx.fillStyle = 'black';
    ctx.fillText( message, 0, fontsize);

    // canvas contents will be used for a texture
    texture1 = new THREE.Texture(canvas1)
    texture1.minFilter = THREE.LinearFilter;
     texture1.needsUpdate = true;

    var spriteMaterial = new THREE.SpriteMaterial({ map: texture1});
    var sprite1 = new THREE.Sprite( spriteMaterial );
    sprite1.position.set( 5, 5, 5 );
    sprite1.scale.set(10,10,10);
    scene.add( sprite1 );
}
```



Testez votre script.

LES INTERSECTIONS

Dans un système interactif il est important de pouvoir sélectionner les objets. Pour cela, nous allons utiliser la technique du ray-casting, c'est-à-dire un rayon qui partira de l'écran (la souris) vers l'espace 3d et qui renverra tous les objets qui sont intersectés. Nous avons besoin donc, pour commencer, de récupérer la position de la souris :

```
// global variables for intersections
var mouse = { x: 0, y: 0 }, INTERSECTED ;

function onDocumentMouseMove( event ) {
    // the following line would stop any other event handler from firing
    //  (such as the mouse's TrackballControls)
    // event.preventDefault();

    // update the mouse variable
    mouse.x = ( event.clientX / window.innerWidth ) * 2 - 1;
    mouse.y = - ( event.clientY / window.innerHeight ) * 2 + 1;
    findIntersections();
}
```

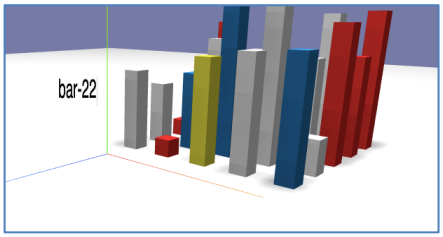
La fonction doit être déclarée dans le main, dans un listener :

```
    // when the mouse moves, call the given function
    renderer.domElement.addEventListener( 'mousemove', onDocumentMouseMove, false );
```

Le ray-casting est implémenté dans la function findIntersections(). Lorsqu'un objet est pointé, son nom apparaît dans le panneau central :

```
    function findIntersections() {

        // create a Ray with origin at the mouse position and direction into the scene (camera direction)
        var vector = new THREE.Vector3( mouse.x, mouse.y, 1 );
        vector.unproject(camera );
        var ray = new THREE.Raycaster( camera.position, vector.sub( camera.position ).normalize() );

        // create an array containing all objects in the scene with which the ray intersects
        var intersects = ray.intersectObjects( scene.children );

        // INTERSECTED = the object in the scene currently closest to the camera
        //and intersected by the Ray projected from the mouse position

        // if there is one (or more) intersections
        if ( intersects.length > 0 ) {
            // if the closest object intersected is not the currently stored intersection object
            if ( intersects[ 0 ].object != INTERSECTED )  {
                // restore previous intersection object (if it exists) to its original color
                if ( INTERSECTED ) {
                    INTERSECTED.material.color.setHex( INTERSECTED.currentHex );
                }

                // store reference to closest object as current intersection object
                INTERSECTED = intersects[ 0 ].object;
                // store color of closest object (for later restoration)
                INTERSECTED.currentHex = INTERSECTED.material.color.getHex();
                // set a new color for closest object
                INTERSECTED.material.color.setHex( 0xffff00 );

                var message = "Waiting...";
                ctx.clearRect(0, 0, canvas1.width, canvas1.height);
```

```
                // update text, if it has a "name" field.
                if ( intersects[0].object.name ) {
                    var intersectedPoint = intersects[0].point;
                    message = intersects[0].object.name;
                }
                ctx.fillStyle = 'black';
                ctx.fillText(message, 0, fontSizeInit);
                texture1.needsUpdate = true;
            }
        }
        else // there are no intersections
        {
            // restore previous intersection object (if it exists) to its original color
            if ( INTERSECTED)
                INTERSECTED.material.color.setHex( INTERSECTED.currentHex );

            // remove previous intersection object reference
            //     by setting current intersection object to "nothing"
            INTERSECTED = null;
        }
    }
```



Testez votre script. Il faut noter qu'il nécessaire que le canvas prenne la taille de la fenêtre pour que les calculs d'intersection fonctionnent bien.
Modifiez la fonction pour afficher dans le panneau principal la valeur de l'axe Y de l'objet pointé.

LES ETIQUETTES

Créez une variable global labels :

```
var labels = {
   x: ["cat1","cat2","cat3","cat4","cat5"],
   z: ["2016","2017","2018","2019","2020"]
};
```

La function labelAxis permet de réaliser les étiquettes d'un axe particulier. P est la position de départ, le separator est le décalage entre deux étiquettes et direction l'axe à travailler.

```
    function labelAxis(dataLbl, direction, separator, p) {
      // data from bars creation !
     var dobj = new THREE.Group();

     for ( var i = 0; i < dataLbl.length; i ++ ) {
                var label = makeTextSprite(dataLbl[i]);

                label.position.set(p.x,p.y,p.z);
        label.scale.set(10,10,10);
                dobj.add( label );
        p[direction]+=separator;
     }
     return dobj;
    }
```

La fonction makeTextSprite utilise un canvas 2d comme précédemment pour créer les étiquettes en utilisant des sprites :

```
function makeTextSprite( message) {
    var fontface = "Helvetica";
    var fontsize = fontSizeInit;
    var canvas2 = document.createElement('canvas');
    var context = canvas2.getContext('2d');
```

```
    context.font = fontsize + "px " + fontface;
    context.textBaseline = 'top';

    // text color
    context.fillStyle = 'black';
    context.fillText( message, 0, fontsize);

    // canvas contents will be used for a texture
    var texture = new THREE.Texture(canvas2)
    texture.minFilter = THREE.LinearFilter;
    texture.needsUpdate = true;

    var spriteMaterial = new THREE.SpriteMaterial({ map: texture});
    var sprite = new THREE.Sprite( spriteMaterial );

    return sprite;
}
```

La fonction createLabels fait appel à la fonction labelAxis pour créer les étiquettes :

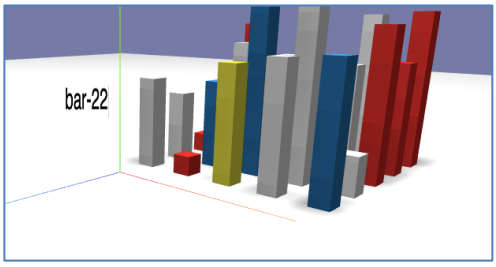
```
    function createLabels() {
      var pz = {x:30,y:0,z:-25};
      var labelsZ = labelAxis(labels.z, "z", 5, pz);
      scene.add(labelsZ);
    }
```

Testez votre script. Modifiez la fonction createLabels pour ajouter les étiquettes sur l'axe X.

LA SELECTION

Nous souhaitons maintenant pouvoir sélectionner un objet spécifique. Pour cela, nous allons ajouter un nouveau listener de manière à identifier un clic (mouseDown) et laisser affiché en jaune l'objet sélectionné. La variable globale selected permet d'enregistrer l'objet cliqué lors du premier clic.

```
    function onDocumentMouseDown( event) {
       inSelectionMode = !inSelectionMode;
       if (inSelectionMode) {
         selected = INTERSECTED;
         if (selected) {
           selected.material.color.setHex( 0xffff00 );
          }
       }
       else
          selected = null;
    }
```



La variable globale inSelectionMode (initialisée à false) permet d'arrêter/redemarrer le ray-tracing à chaque clic. Il est donc nécessaire de modifier également la fonction onDocumentMouseMove :

```
  if (!inSelectionMode)
    findIntersections();
```

L'INTERFACE GRAPHIQUE – GUI

La sélection va nous permettre de modifier la couleur de l'objet sélectionné grâce à une interface graphique. Nous allons la créer en utilisant la bibliothèque dat.gui.min.js, qui propose divers widgets pour modifier vos variables.

Ajoutez une nouvelle variable globale colorChanged=false, puis la fonction createGUI()

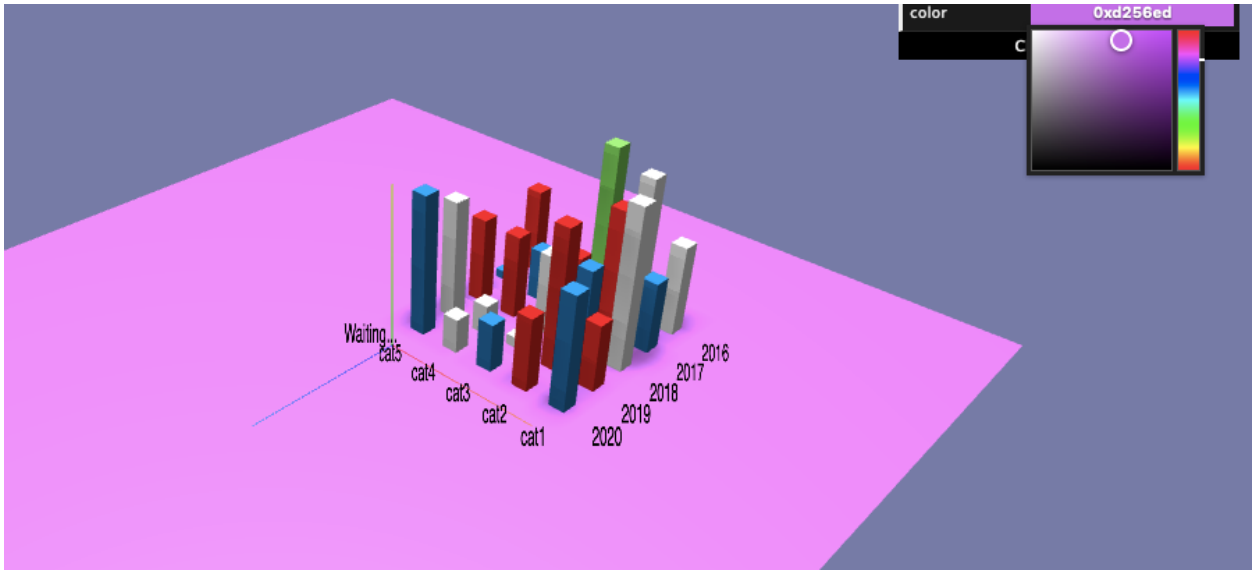
```
function createGUI() {
   var controls = { color: 0xffffff }
   var gui = new dat.GUI();
   gui.addColor(controls, 'color').onChange(function(value) {
        if (selected) {
           selected.material.color = new THREE.Color(value);
           colorChanged = true;
         }
   } );
}
```

gui.addColor nous permet d'ajouter un color picker dont la couleur sera affectée à la couleur de l'objet sélectionné. Nous allons également indiquer qu'un changement de couleur a été effectué grâce à la variable colorChanged.
Il est donc nécessaire de modifier la fonction findIntersections pour éviter que la couleur soit perdue lorsque le ray-tracing rédemarre :

```
// restore previous intersection object (if it exists) to its original color
if ( INTERSECTED && !colorChanged) { … }
```

et puis :

```
texture1.needsUpdate = true;
colorChanged = false;
```



Déposez votre document et votre code sur Moodle.

REFERENCES

Librement adapté de
https://codepen.io/DapperDirewolf/pen/mevjKp
http://stemkoski.github.io/Three.js/Sprite-Text-Labels.html