TP2

Refactorisation et Classe World

Au TP1, vous vous êtes concentrés sur les fonctionnalités mises en place et pas sur la qualité du code produit
L'application est simple mais monobloc.
L'objectif est maintenant de refactoriser c'est-à-dire reconstruire les fonctions en organisant tout cela sous forme de petit modules simples qui vont gérer une petite partie de la complexité globale.

1. Copiez/collez le projet `/tp1` et renommez le `/tp2`.

Tous les scripts seront placés dans le dossier `/src`.

2. Faites et une copie et videz le contenu du fichier `src/main.js` pour tout reconstruire.

L'idée de base est que l'application web que vous allez construire le doit pas connaitre l'existence de **Three.js**.
Pour l'instant ce n'était pas le cas puisque le code de Three.js figurait directement dans le fichier `main.js`.
Il faut ajouter une couche d'abstraction.

3. Créer les nouveaux dossier `/src/World3D`

Tout ce qui concerne Three.js se trouvera dans cet espace et pas ailleurs.
Vous allez créer une classe nommée **World** (votre petit monde en 3D) qui encapsulera tout ce qui doit être fait pour travailler avec Three.js
Pour rappel, dans l'approche "classique" il fallait mettre en place 6 étapes :

1. Configuration initiale
2. Créer la scène
3. Créer/placer la caméra
4. Ajouter un/plusieurs objets dans la scène
5. Créer le rendu
6. Injecter le rendu dans la scène

Mais, avec la nouvelle classe **World**, les choses doivent se simplifier et devenir :

1. Créer une instance de la classe World3D
2. Initialiser les objets de la scène
3. Rendre la scène

Le code de la classe **World3D** va globalement ressembler à `/src/World3D/World3D.js`

```
import { createScene } from "./components/scene.js";
…

class World3D {
  constructor(target) {}

  /**
   * Initialiser les éléments dans la scène.
   */
  init() {
    …
  }

  /**
   * Démarrer la boucle d'animation
   */
  start() {
    this.renderer.setAnimationLoop(this.animate.bind(this));
  }

  /**
   * La boucle d'animation.
   */
  animate() {
    …
  }
}

export { World3D };
```

Voilà, la classe **World3D** ne sera pas plus compliquée.

4. Complétez maintenant l'écriture de `main.js` qui doit instancier la classe :

```
import { World3D } from './World3D/World3D';

// fonction principale
function main() {
  // cibler le conteneur du canvas
  const container = document.querySelector('#scene-container');

  // 1. Instancier la classe World
  const world = new World3D(container);

  // 2. Déclencher l'actualisation de la scène
  world.init();
  world.start();
}

// appeler main pour démarrer l'application
main();
```

Remarque : la fonction main peut tout à fait intégrer en plus du code qui n'a pas de lien direct avec votre petit monde en 3D.
Les modules que vous allez créer seront répartis en 2 sous-ensembles : les **composants** et les **systèmes**.
Les composants représentent tout ce qui peut être placé dans la scène, comme le cube, la caméra et la scène elle-même.
Les systèmes sont des éléments complémentaires (comme les helpers).
Voici ce que vous allez ajouter :

```
src/World3D/components/cameras.js
src/World3D/components/lights.js
src/World3D/components/scene.js
src/World3D/components/renderer.js

src/World3D/components/scene/cube.js

src/World/system/helpers.js
```

Remarque : le dossier `/World3D/components/scene` va contenir les objets qui seront placés dans la scène (dans cet exemple simple, c'est le cube).

5. Ajoutez le code du fichier `src/World3D/components/scene.js` :

```
import { Color, Scene } from "three";

function createScene() {
  const scene = new Scene();
  scene.background = new Color("cornsilk");

  return scene;
}

export { createScene };
```

Remarques : dans l'import, on ne vient prendre que les classes utilisées, cela permet de mieux repérer les éléments utiles au script.
Systématiquement, une fonction `createXXX` sera mise en place et exportée.
Ici, la fonction retourne l'objet `scene`.
Cet objet sera mémorisé dans le script Word.js (développé plus loin).

6. Ajoutez le code du fichier `src/World/components/renderer.js` :

```
import { WebGLRenderer } from "three";

function createRenderer(target) {
  const renderer = new WebGLRenderer();
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setSize(target.offsetWidth, target.offsetHeight);
  target.appendChild(renderer.domElement);

  return renderer;
}
```

```
export { createRenderer };
```

Remarque : la fonction `createRenderer` reçoit comme paramètre la variable `container` qui représente le canvas.

7. Ajoutez le code du fichier `src/World/components/cameras.js` :

```
import { PerspectiveCamera } from "three";

function createCameras(target) {
  const camera = new PerspectiveCamera(
    75, // FOV
    target.offsetWidth / target.offsetHeight, // ratio
    0.1, // zone proche
    100 // zone lointaine
  );
  camera.position.set(2, 2, 15);

  return camera;
}

export { createCameras };
```

Remarque : si plusieurs caméras étaient mises en place et donc retournées par la fonction alors il faudrait écrire :

```
  return {camera1, camera2, camera3};
```

8. Ajoutez le code du fichier `src/World/components/lights.js` :

```
import { AmbientLight, DirectionalLight } from "three";

function createLights() {
  const ambientLight = new AmbientLight(0xcccccc, 1);
  const directionalLight = new DirectionalLight(0xffffff, 1);

  return { ambientLight, directionalLight };
}

export { createLights };
```

Remarque : ici 2 lumières sont retournées. Elles seront ultérieurement ajoutées dans la scène dans le script `World.js`

9. Ajoutez le code du fichier `src/World/system/helpers.js` :

```
import { AxesHelper, GridHelper } from "three";
import { OrbitControls } from "three/addons/controls/OrbitControls.js";

function createHelpers(camera, renderer) {
  // les 3 axes
  const axesHelper = new AxesHelper(12);
  // la grille (dimension 20 divisée en 20)
  const gridHelper = new GridHelper(20, 20);
  // contrôle avec la souris
  const orbitControls = new OrbitControls(camera, renderer.domElement);

  return { orbitControls, axesHelper, gridHelper };
}

export { createHelpers };
```

10. Ajoutez enfin le code du fichier `src/World/components/scene/cubes.js` :

```
import { BoxGeometry, MeshBasicMaterial, Mesh } from "three";

function createCubes() {
  const geometry = new BoxGeometry(6, 6, 6);
  const material = new MeshBasicMaterial({ color: "blue", wireframe: false });
  const cube1 = new Mesh(geometry, material);

  return { cube1 };
}

export { createCubes };
```

Remarque : la fonction retourne une collection de cubes ... même si pour le moment, il n'y a qu'un seul cube dans cette collection.

Vous allez maintenant complétez le code de la classe principale World3D.

11. Voici les imports :

```
import { createScene } from "./components/scene.js";
import { createRenderer } from "./components/renderer.js";
import { createCameras } from "./components/cameras.js";
import { createLights } from "./components/lights.js";
import { createHelpers } from "./system/helpers.js";
import { createCubes } from "./components/scene/cubes.js";
```

12. Voici ce que met en place le constructeur :

```
  constructor(target) {
    this.target = target;
    // 1. Ajouter la scène
    this.scene = createScene();
    // 2. Mettre en place le rendu dans le canvas
    this.renderer = createRenderer(this.target);
    // 3. Ajouter les caméras
    this.camera = createCameras(this.target);
    this.scene.add(this.camera);
    // 4. Ajouter les lumières
    const { ambientLight, directionalLight } = createLights();
    this.scene.add(ambientLight, directionalLight);
    // les outils (optionnel)
      const { controls, axesHelper, gridHelper } = createHelpers(this.camera,
this.renderer);
    this.scene.add(axesHelper, gridHelper);
    this.controls = controls;
  }
```

Les ajouts des éléments ne doivent se faire que dans la classe World3D.

13. Voici les actions d'initialisation :

```
  init() {
    const { cube1 } = createCubes();
    this.cube1 = cube1;
    this.scene.add(this.cube1);
  }
```

14. Voici le traitement de la boucle d'animation :

```
  animate() {
    this.cube1.rotation.y += 0.01;
    this.renderer.render(this.scene, this.camera);
  }
```

A FAIRE : complétez le code pour animer les 3 cubes comme dans le TP1.
La fonction `createCubes` doit maintenant retourner la collection des 3 cubes.
Ils seront simplement mis en rotation.

A FAIRE : mettez en commentaire le processus d'affichage des cubes.
Consultez la page qui présente un exemple simple pour afficher des segments :
https://threejs.org/docs/index.html#manual/en/introduction/Drawing-lines
Vous devez mettre en place un script `src/World/components/scene/lines.js` qui retourne les segments.
Le but est de présenter une flèche pointant sur l'axe des Z.