Création de jeux avec BabylonJS

Support des cours

Document mis à jour le 25/01/2024 13:37

Version de brouillon

Historique des versions

Date	Version	Commentaires
22/01/2024	V1	Première version
25/01/2024	V1.1	Correction sur jsconfig.json
25/01/2024	V1.2	Suppression des commentaires dans le code et de l'entête Ceci pour faciliter le Copier/Coller en version pdf.
25/01/2024	V1.3	Corrections sur jsconfig.json (en prévision du TypeScript)

Liste de diffusion

Arguimbau Olivier	1ex

Auteur : **Olivier Arguimbau**

Table des matières

Support 2 – Partir de zéro

NPM, Git, Test et déploiement

Remarque : Nous utilisons ici le langage JAVASCRIPT pour commencer, après réflexion le passage en TypeScript sera adopté lors du support 4, la complexité sera abordée tout en douceur.

Introduction

Vous allez devoir distribuer un jeu, un jeu codé en JavaScript et comportant un certain nombre de dépendances :

- BabylonJS
- Textures
- Son
- Modèles 3D

A l'intérieur de votre code JavaScript vous pourriez être tenté de référencer les dépendances avec leurs URLs externes, cela peut fonctionner un moment mais il n'y a aucune garantie de pérennité, et de toute façon c'est une mauvaise habitude autant ne pas la prendre.

La solution consiste donc à distribuer votre jeu puis à l'héberger de manière autonome, le site contiendra donc toutes les dépendances localement et n'aura pas besoin de ressources externes.

La solution technique dépend de l'architecture de développement choisie ; Pour ma part j'ai choisi de développer dans un environnement packagé et géré par le Node Package Manager dans un environnement Javascript moderne (ES6) ou TypeScript suivant les projets. Le packaging final du site et son déploiement se fera par l'intermédiaire de WebPack.

Pour résumer l'environnement technique choisi sera :

- Visual Studio Code
- Node Package Manager
- WebPack
- BabylonJS
- Ecma Script 6
- Codage en POO
- Blender pour les modèles 3D
- Gimp pour les besoin de retouches d'images
- ChatGPT4 ou autre générateur d'images par IA pour les logos et autres images annexes.

Installations des logiciels

1. Installer Visual Studio Code

Depuis le site officiel choisir la plateforme et installer le logiciel :
https://code.visualstudio.com/download

Il est également fortement recommandé d'installer des extensions suivantes (depuis VScode) :

- LiveServer
- ESlint

2. Installer NPM

Depuis le site officiel choisir la plateforme et installer le logiciel : https://nodejs.org/en

Choisir la version LTS de préférence.

3. Installer les logiciels de création 2D/3D

C'est optionnel mais cela peut servir SI vous avez le temps et la patience de vous former à Blender, pour GIMP c'est un peu plus facile.

- Blender : https://www.blender.org/download/
- Gimp : https://www.gimp.org/downloads/

Mise en place de l'environnement à partir de zéro

1. Création du dépôt sur github

Depuis votre compte github créez un nouveau dépôt :

- Remplir les informations de base (nom et description)
- Choisissez le mode public (vous n'avez rien à cacher et pour le concours ce sera plus pratique).
- Ajouter un README.md
- Ne pas choisir de .gitignore par défaut
- Editez votre README.md
- Copier la ligne de clonage depuis le dépôt (Bouton Code / Texte contenant l'url du dépôt ex : https://github.com/pigmin/testBJS.git)

2. Edition du projet sous VScode

- Ouvrir un terminal dans le répertoire de vos sources
- Lancer le clonage du dépôt
- Rentrer dans le répertoire et ouvrir VScode (code .)
- Enregistrer votre workspace

3. Configuration du projet pour NPM, ES6 et WebPack

Packages

- Depuis VScode ouvrir un terminal
- Lancer la commande « npm init »
- Répondre aux questions et confirmer
- Installer les packages webpack

```
npm install webpack webpack-cli webpack-merge webpack-dev-server -D
```

```
npm install clean-webpack-plugin -D
```

Infos :

- *webpack-cli nous permet d'utiliser webpack depuis la ligne de commande*
- *webpack-merge permet d'utiliser des fichiers de configurations modulaires*
- *webpack-dev-server permet de lancer notre application directement pendant la phase de dev*

- Installer les loaders et utilitaires webpack :

```
npm install file-loader js-loader source-map-loader url-loader -D
```

```
npm install html-webpack-plugin -D
```

Infos :

- *les loaders permettent de traiter les ressources dans notre package webpack*
- *le plugin html permet de gérer l'inclusion de l'appli dans le html et la gestion du favicon*

- Installer les autres packages pour gérer le déploiement sur « git pages »

```
npm install gh-pages
```

- Installer eslint :

```
npm init @eslint/config -D
```

  - Répondre « check and find problems »
  - « JavaScript modules »
  - « None of these »
  - « No » (pour TypeScript, conseillé pour le moment)
  - « Browser »
  - « JavaScript »
  - Répondre « Yes » puis « npm »

- Installer les plugin eslint

```
npm install eslint-plugin-import
```

Configuration du .gitignore

Créer le fichier `.gitignore` à la racine et ajouter les lignes suivantes :

```
node_modules
dist
```

Ceci demande à git de ne pas prendre en compte les répertoires des modules et de distribution.
Enregistrer.

Configuration webpack

Editer le fichier `package.json` et ajouter les lignes :

```
  "scripts": {
    "start": "npx webpack serve --config webpack.dev.js",
    "start:prod": "webpack serve --config webpack.prod.js",
    "build:dev": "npx webpack --config webpack.dev.js",
    "build": "npx webpack --config webpack.prod.js",
    "lint": "npx eslint . --ext .js",
    "predeploy": "npm run build",
    "deploy": "gh-pages -d dist"
  }
```

Les scripts seront utilisés pour tester, packager et éventuellement déployer notre application.

Créer un fichier `webpack.common.js` et ajouter :

```
const path = require("path");
const fs = require("fs");
const HtmlWebpackPlugin = require("html-webpack-plugin");
const { CleanWebpackPlugin } = require("clean-webpack-plugin");

const appDirectory = fs.realpathSync(process.cwd());

module.exports = {
    entry: path.resolve(appDirectory, "src/index.js"),
    output: {
        filename: "js/babylonBundle.js",
        path: path.resolve("./dist/"),
    },
    resolve: {
        extensions: [".ts", ".js"],
        fallback: {
            fs: false,
            path: false,
        },
    },
    module: {
```

```
        rules: [
            {
                test: /\.m?js/,
            },
            {
                test: /\.(js|mjs|jsx|ts|tsx)$/,
                loader: "source-map-loader",
                enforce: "pre",
            },
            {
                test: /\.tsx?$/,
                loader: "ts-loader",
            },
            {
                test: /\.(glsl|vs|fs)$/,
                loader: "ts-shader-loader",
                exclude: /node_modules/,
            },
            {
                test: /\.(mp3|wav|ogg|mp4|glb|hdr)$/i,
                use: [
                    {
                        loader: "file-loader",
                    },
                ],
            },
            {
                test: /\.(png|jpg|gif|env|gltf|stl|dds|json)$/i,
                use: [
                    {
                        loader: "url-loader",
                        options: {
                            limit: 4096,
                        },
                    },
                ],
                type: 'javascript/auto'
            },
        ],
    },
    plugins: [
        new CleanWebpackPlugin(),
        new HtmlWebpackPlugin({
            inject: true,
            favicon: "public/favicon.ico",
            template: path.resolve(appDirectory, "public/index.html"),
        }),
    ],
};
```

Créer un fichier `webpack.dev.js` et y ajouter :

```
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');
const path = require('path');
const fs = require('fs');

// App directory
const appDirectory = fs.realpathSync(process.cwd());

module.exports = merge(common, {
    mode: 'development',
    devtool: 'inline-source-map',
    devServer: {
        static: path.resolve(appDirectory, "public"),
        compress: true,
        hot: true,

        open: true,
    },
});
```

Créer un fichier `webpack.prod.js` et y ajouter :

```
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
  mode: 'production',
  devtool: 'source-map'
});
```

Le détails de ces fichiers est assez long mais leur lecture est aussi assez simple, n'hésitez pas.

Finalisation de la configuration

Dans votre projet créer les répertoires « `src` », « `assets` », « `public` » et « `dist` » à la racine.
Dans le répertoire « `public` » créer le fichier « `index.html` » et y ajouter :

```
<!DOCTYPE html>
<html lang="fr" xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width,initial-scale=1.0">
    <title>BabylonJS Game</title>
    <link rel="icon" type="image/x-icon" href="./favicon.ico">
    <style>
        html, body {
          overflow: hidden;
          width: 100%;
          height: 100%;
          margin: 0;
          padding: 0;
        }
        #renderCanvas {
          width: 100%;
          height: 100%;
          touch-action: none;
        }
        #fps {
                  position: absolute;
                  background-color: black;
                  border: 2px solid red;
                  text-align: center;
                  font-size: 16px;
                  color: white;
                  top: 15px;
                  right: 10px;
                  width: 60px;
                  height: 20px;
              }
      </style>
  </head>
  <body>
    <canvas id="renderCanvas" touch-action="none"></canvas>
    <div id="fps">0</div>
  </body>
</html>
```

Vous devez également ajouter un fichier « `favicon.ico` » à côté du fichier `index.html`

Ici : https://raw.githubusercontent.com/BabylonJS/Controls/master/www/assets/favicon.ico
Enfin dans le répertoire « `src` » créer le fichier « `index.js` » et y ajouter :

```
window.onload = () => {    console.log('Hello World!'); };
```

A ce stade on peut essayer notre configuration (ce n'est pas terminé encore un peu de patience).

Configuration de eslint

Editer le fichier « `.eslintrc.js` » et remplacer le contenu par :

```
module.exports = {
  root: true,
  env: {
    browser: true,
    es2021: true
  },
  settings: {
    "import/resolver": {
      "node": {
        "extensions": [
          ".js",
          ".jsx"
        ]
      }
    }
  },
  plugins: [
    'import',
  ],
  extends: [
    'eslint:recommended',
    'plugin:import/errors',
    'plugin:import/warnings',
  ],
  overrides: [
    {
      env: {
        node: true
      },
      files: [
        '.eslintrc.{js,cjs}'
      ],
      parserOptions: {
        sourceType: 'script'
      }
    }
  ],
  parserOptions: {
    ecmaVersion: 'latest',
    sourceType: 'module'
  },
  rules: {
    "no-unused-vars" : "warn",
```

```
    "no-empty" : "warn",
  }
}
```

Créer le fichier « `.eslintignore` » à la racine, y ajouter :

```
node_modules
dist
webpack.*
```

Enfin dans le répertoire racine créer le fichier « `jsconfig.json` »

```
{
    "compilerOptions": {
    "module": "ES6",
    "target": "ES6",
    "moduleResolution": "node",
    "strict": true,
    "noImplicitAny": true,
    "noImplicitReturns": true,
    "noImplicitThis": true,
    "strictNullChecks": true,
    "strictFunctionTypes": true,
    "preserveConstEnums":true,
    "sourceMap": true,
    "rootDir": "./src",
    "esModuleInterop": true,
    "allowJs": true,
    "forceConsistentCasingInFileNames" : true,
    },
    "exclude": ["node_modules/**/*", "dist/**/*", "webpack*.*"]
  }
```

Ce fichier indique à VScode que notre projet est en Javascript et lui précise nos préférences pour ce projet : Norme JS , type d'import des modules , etc.

LA CONFIGURATION EST TERMINEE

Dans package.json vous avez dans la section « scripts » les scripts disponibles, mais voici les deux principaux (depuis le terminal de VScode dans le répertoire de votre projet) :

`npm start` => execute votre projet et lance un navigateur au besoin (ctrl + C pour l'arrêter)

`npm build` => création du répertoire de distribution dans « dist »

`npm deploy` => déploiement sur votre « git pages » (nécessite une petite configuration)

4. Préparation du projet pour BabylonJS

Suivant votre projet vous aurez besoin de plus ou moins de packages babylonJS, mais commençons pas les plus évidents, depuis le terminal installons :

```
npm install @babylonjs/core @babylonjs/gui @babylonjs/inspector
@babylonjs/loaders @babylonjs/materials -D
```

Pour le moteur physique (nous utiliserons HAVOK) :

```
npm install @babylonjs/havok -D
```

5. Premier test en 3D

Ouvrir le fichier index.js et le modifier un peu :

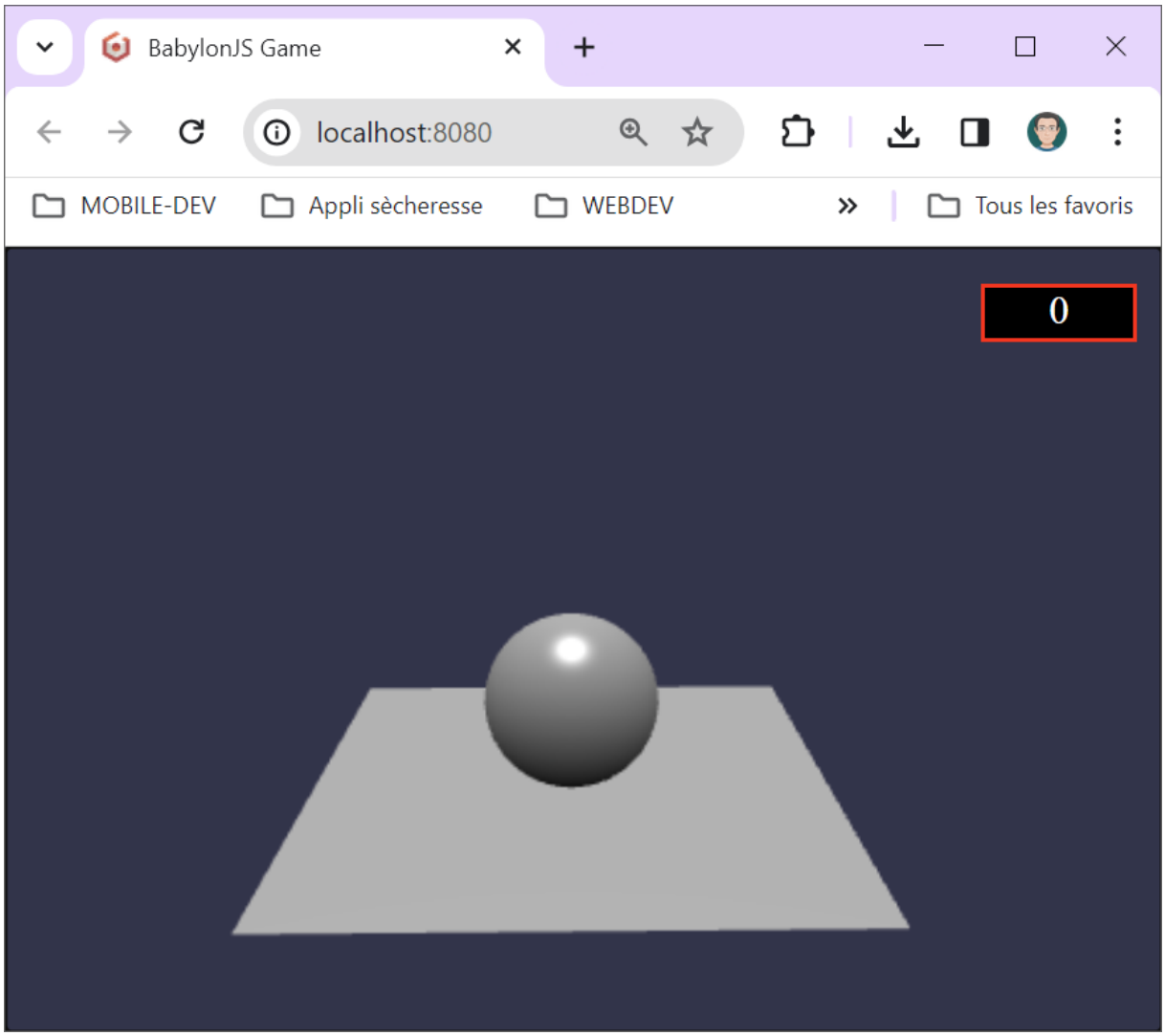
```
import { Engine, FreeCamera, HemisphericLight, MeshBuilder, Scene, Vector3 }
from "@babylonjs/core";

window.onload = () => {
    console.log('Hello World!');

    const canvas = document.getElementById("renderCanvas");
    let engine = new Engine(canvas, true);

    const createScene = function () {
        const scene = new Scene(engine);
        const camera = new FreeCamera("camera1",
            new Vector3(0, 5, -10), scene);
        camera.setTarget(Vector3.Zero());
        camera.attachControl(canvas, true);
        const light = new HemisphericLight("light",
            new Vector3(0, 1, 0), scene);
        light.intensity = 0.7;
        const sphere = MeshBuilder.CreateSphere("sphere",
            {diameter: 2, segments: 32}, scene);
        sphere.position.y = 1;
        const ground = MeshBuilder.CreateGround("ground",
            {width: 6, height: 6}, scene);
        return scene;
    };
    const scene = createScene();
    engine.runRenderLoop(function () {
            scene.render();
    });
    window.addEventListener("resize", function () {
            engine.resize();
    });
};
```

Essayez et vous devriez voir :



Mise en place de l'environnement à partir du template git

Un dépôt est actif sur mon github : https://github.com/pigmin/BabylonJS-Game

Il suffit de le cloner en tant que « Template » pour avoir une base de départ plus ou moins stable.