Option Réalité Virtuelle : TP WebGL

Intoduction :

WebGL est une spécification d'interface de programmation de 3D dynamique pour les pages et applications HTML5 créée par le Khronos Group. Elle permet d'utiliser le standard OpenGL ES au sein d'une page web.

WebGL permet d'afficher, de créer et de gérer dynamiquement des éléments graphiques en 3D dans la fenêtre d'un navigateur web. Il est actuellement implémenté dans la plupart des grands navigateurs modernes, cependant pour ce TP nous travaillerons sur Firefox car Chrome pose quelques problèmes d'affichage de textures.

Lorsqu'un élément graphique de type WebGL est inclus dans une page web, le navigateur exécute un programme en JavaScript utilisant l'interface WebGL. La bibliothèque WebGL appelle à son tour le pilote OpenGL ES du système d'exploitation qui se chargera de faire les calculs nécessaires à l'affichage sur l'écran, en exploitant si possible l'accélération matérielle du ou des processeurs graphiques du terminal.

Bibliothèques et Framework existants : Three.js et Babylon.js étant les plus répandues, il existe bon nombre d'autres bibliothèques comme Blend4Web (Blender), OSGJS (Open Scene Graph) et Unity (qui propose l'export en option depuis la version 5).

Logiciels : Firefox et Sublime Text.

Pas nécessaire d'apprendre Html ou CSS, les pages sont fournies.

Pour ce TP, nous utiliserons Babylon.JS : un moteur 3D temps réel sous forme de bibliothèque JavaScript. Nous l'utiliserons car en plus d'être simple à implémenter (une simple balise <canvas> suffit), il possède un moteur physique qui nous permettra de gérer les collisions très simplement.

Quelques notions de Javascript

JavaScript est un langage de programmation de scripts principalement employé dans les pages web interactives. C'est un langage orienté objet à prototype, c'est-à-dire que les bases du langage et ses principales interfaces sont fournies par des objets qui ne sont pas des instances de classes, mais qui sont chacun équipés de constructeurs permettant de créer leurs propriétés, et notamment une propriété de prototypage qui permet d'en créer des objets héritiers personnalisés.

Syntaxe :

Instructions séparées par des « ; ». Il n'y a pas de sensibilité aux espaces. Les commentaires s'écrivent comme en C++.

Déclaration d'une variable :

```
var maVariable ;
```

On peut déclarer plusieurs variables sur une même ligne :

```
var maVariable=4 ; maVariable2 ;
```

Le type de la variable est attribué dynamiquement en JavaScript. On peut dès lors réaliser l'opération :

```
maVariable = 5.5 ;
```

Pour déclarer une variable vous n'avez donc qu'à connaître le mot clé var ! Pour accéder au type d'un objet, il suffit d'utiliser la commande « typeof maVariable ».

Fonctions :

Une fonction se déclare de la façon suivante :

```
function myFunction(arguments) {
    // Le code que la fonction va devoir exécuter
}
```

Portée des variables : Faire attention au nommage car la déclaration des variables se faisant par le seul mot clé *var*, il peut y avoir conflits entre des variables globales et locales.

Forme d'un objet :

(Source : https://developer.mozilla.org/fr/docs/Web/JavaScript/Guide/Le_mod%C3%A8le_objet_JavaScript_en_d%C3%A9tails)

Ce qu'il faut bien comprendre avec l'orientation prototype de JavaScript, c'est qu'il n'y a plus de différence entre une classe et l'instance d'une classe :

- Une classe définit un certain nombre de propriétés caractérisant un ensemble d'objets. Elle est la représentation abstraite de cet ensemble d'objets.
- Une instance est un des membres de la classe, et elle possède exactement les mêmes propriétés.

En JavaScript, tout objet peut définir ses propriétés. On peut donc définir des objets prototypiques c'est-à-dire définissant des propriétés initiales. Cela n'empêchera pas un objet d'hériter du prototype et de posséder ses propres propriétés.

```
function Employé () {
  this.nom = "";
  this.branche = "commun";
}


function Manager () {
  this.rapports = [];
}
Manager.prototype = new Employé;

function Travailleur () {
  this.projets = [];
}
Travailleur.prototype = new Employé;
```

Ici les objets *Travailleur* et *Manager* héritent des propriétés de *Employé* mais possèdent aussi leurs propriétés.

On peut ensuite déclarer un Manager et accéder à ses propriétés :

```
var emilie = new Manager ;

emilie.nom = "Emilie" ;
```

Mais on peut ensuite ajouter des propriétés à emilie elle-même !

emilie.idees = ["vacances", "ProjetRSE"] ;

L'objet *emilie* est donc particulier. Ceci est pratique mais attention à ne pas perdre vos propriétés dans certains objets !

this et _this :

Source : https://javascriptplayground.com/blog/2012/04/javascript-variable-scope-this/

Revenons à la portée d'une variable : si on déclare directement

```
var x = 9;
```

Cela revient à écrire window.x =9 ;

Si en revanche on écrit

```
function myFunc() {
        var x = 5;
};
```

```
console.log(x); //undefined
```
Car x a été définit localement. Lorsqu'on n'utilise pas le mot clé *var* la variable est automatiquement globale ! Ceci est une TRES mauvaise idée pour la stabilité du code…

this peut-être géré de différentes façons :

```
function foo() {
        console.log(this); //global objet
};

myapp = {};
myapp.foo = function() {
        console.log(this); //pointe sur myapp objet
}

var link = document.getElementById("myId");
link.addEventListener("click", function() {
        console.log(this); //pointe sur link
}, false);
```

Lors de la création de fonctions (dans la portée globale), this fait donc référence à des variables/objets globaux. On utilise donc `var _this = this` pour stocker dans _this la valeur courante de this.

```
$("myLink").on("click", function() {
        console.log(this); //points to myLink (as expected)
        var _this = this;  //store reference
        $.ajax({
                //ajax set up
                success: function() {
                        console.log(this);  //points  to  the  global  object.
Hein?
                        console.log(_this); //better!
                }
        });
});
```

<u>window.addEventListener :</u>

Quand on ajoute un window.addEventListener dans une fonction, on va modifier directement le this. On vient gérer un évènement dans la fonction. On va pouvoir par exemple gérer l'appui de touches, ou bien le clic sur la fenêtre pour prendre la main sur le pointeur. Permet également un contrôle plus fin sur la phase d'activation de l'écouteur. Fonctionne sur tous les éléments DOM (Document Object Model) et pas seulement HTML.

-autres subtilités qui viendront au cours du tp

WebGL

Le projet se compose d'une page html qui va pouvoir être lue par notre navigateur, une page de style css, et enfin nos 3 pages javascript sur lesquelles nous allons travailler. Nous stockerons nos textures/vidéos/… dans le dossier assets.

Game.js, le fichier principal qui va établir les liens entre le joueur et l'arène.

Player.js qui va définir notre joueur, et notamment, ses déplacements.

Arena.js dans lequel nous allons créer notre arène.

Vous devrez utiliser la documentation de Babylon JS afin de mener à bien ce TP.

http://doc.babylonjs.com/

Partie I

1/Nous allons compléter le fichier Game.js en créant notre scène.

Avant toute chose, lisez attentivement le contenu de l'objet Game.

Ensuite, allez dans la fonction _initScene située dans le prototype de l'objet Game et complétez-la.

2/ Vous allez créer une scène de plateformes dans la page Arena.js en créant :

-des meshs contenant des matériaux et textures comme indiqués sur la page

-de l'audio

-des lumières

-des collisions

-une skybox (lien utile : http://3delyvisions.co/skf1.htm)

3/ Maintenant que tout est en place, vous allez animer la scène en faisant bouger nos meshs dans le prototype de Arena.js

Partie II

Nous allons nous concentrer sur la page Player.js, notamment sur le déplacement de la caméra.

Comme pour l'objet Game, nous vous conseillons de bien le lire avant de commencer à coder.

Allez dans la fonction _initCamera , supprimez l'attache au canvas et initialisez les axes de mouvements de la caméra à faux.

Allez maintenant dans votre objet Player. Nous allons maintenant voir comment gérer les axes de déplacement de la caméra quand les touches sont relachées ou enfoncées.

Nous allons donc utiliser la méthode addEventListener de l'objet **window**.

Petite aide : quand on détecte un certain type d'évènement (par exemple « touche relachée », on va alors appeler notre listener qui sera une fonction ayant pour argument notre événement « evt » et qui va dans notre cas modifier les axes de la caméra (vrai/faux) suivant la touche.

Pour la fonction addEventListener :
https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener

Types d'évènements : https://developer.mozilla.org/fr/docs/Web/Events

Keycodes : http://keycode.info/

Notre code est maintenant capable d'intercepter les touches qu'on utilise pour déterminer le mouvement de notre personnage. Nous verrons plus tard comment le déplacer à partir de ça.

Pour l'instant, de la même manière que pour les touches du clavier, nous allons nous occuper de gérer les mouvements de la souris.

Pour cela, nous allons utiliser un eventListener pour le mouvement de la souris.

Dans celui-ci, nous allons d'abord nous occuper de la rotation sur l'axe y. Il va nous suffir de l'incrémenter avec la propriété adéquate de notre événement, multiplié par une valeur très petite (sinon la caméra s'emballe).

PS: Les angles sont en radians !

Concernant la rotation sur x, il faudra faire en sorte avant d'effectuer la rotation, que le joueur ne regarde pas en dessous de ses pieds ou au-dessus de sa tête.

Nous allons enfin pouvoir déplacer notre personnage !

Rendons-nous dans notre fonction _checkMove où une vitesse relative au ratio FPS a été créée.

Pour chaque axe de mouvement de la caméra, s'il est sur vrai, calculer à partir du vecteur rotation de la caméra ses vecteurs de déplacement #cahsohtoa :)

Vous devrez utiliser 2 fois la fonction parseFloat() pour convertir en nombre flottant les valeurs retournées par l'objet Math (sin et cos) et les angles de rotation de notre camera.

Une fois le vecteur déplacement créé, déplacer votre caméra avec la fonction moveWithCollision().

Passons au saut !

On va d'abord créer un eventListener dans notre objet pour détecter le saut : Si on appuie sur la touche saut et que le perso peut sauter, on définit la hauteur de son saut (sur l'axe y) et on l'empêche de pouvoir ressauter.

La gestion du saut va se gérer en 2 étapes : on monte, puis on descend.

Etape1 :

De retour dans _checkMove(), si notre hauteur à atteindre existe, on va effectuer un lerp.

Le lerp est un mouvement adouci. C'est-à-dire qu'on calcule la distance qu'il nous reste à parcourir et on va la diviser (par 3,4,5... comme vous voulez).

On déplace notre joueur selon ce vecteur saut et on va regarder la hauteur de notre personnage +1 est supérieure à la hauteur de saut souhaitée. Si oui, on reset la hauteur souhaitée et on initialise notre accélération vers le bas (servira pour la chute): this.camera.acceleration = 0;

Etape 2 :

L'objectif est de déterminer la distance qu'il reste à parcourir pour atteindre le sol et accélérer tant que le joueur n'est pas arrivé au sol.

Pour cela, nous allons lancer un rayon depuis le joueur vers le bas.

On regarde quel est le premier objet qu'on touche en excluant le mesh qui appartient au joueur.

Si la distance avec le sol est inférieure ou égale à la hauteur du joueur -> On a touché le sol !

Du coup, le joueur peut de nouveau sauter, l'acceleration et la hauteur de saut sont réinitialisés .Sinon, l'acceleration augmente et on déplace le joueur vers le bas, avec l'acceleration multipliée par la vitesse relative et divisée par un multiple de 10 (à juger)

Bravo ! Vous avez fini !

Une dernière partie reste à faire maintenant que vous disposez d'une scène et du contrôle sur la caméra avec un saut : les interactions avec la scène.

Ceci peut-être simplement le saut sur des plate-formes mobiles, le clic sur un objet ou la collision avec celui-ci... Vous pouvez laisser libre cours à votre imagination !

Enfin pour ceux qui ont terminer et qui souhaitent aller plus loin, vous pouvez essayer de programmer une arme dans un script Weapon.js.