CRÉEZ VOTRE JEU DE PLATEFORME 2D AVEC JAVASCRIPT

Créez votre propre jeu de plateforme 2D facilement intégrable à un site web.

Coding Club Epitech Toulouse

Édition Winter Camp 2016

Sommaire

1 – Introduction

Bonjour à tous, lors des différents Coding Clubs vous avez pu réaliser des sites internet à l'aide du langage HTML mais également un bot en JavaScript. Aujourd'hui nous allons amplifier ces connaissances en créant un jeu vidéo de plateforme 2D en JavaScript !

Si vous n'avez pas participé aux précédents Coding Clubs ne vous en faîtes pas, les Cobras sont là pour vous aider !

Le but de la session est de vous expliquer comment fonctionne un jeu vidéo, ce qui se passe derrière, tout en réalisant votre propre jeu que vous pourrez mettre en ligne facilement à l'aide des technologies web que nous utilisons !

Commençons !

2 – Prérequis

Pour ce tutoriel, voici ce dont vous aurez besoin :

- Un navigateur internet (Chrome, Firefox, Edge, Internet Explorer, ...)
- Un éditeur de texte, de préférence un éditeur de code (Visual Studio Code, Sublime Text, Brackets, ...)
- Le fichier phaser.min.js : https://github.com/photonstorm/phaser/releases/download/v2.6.2/phaser.min.js
- Les assets : https://epitechfr-my.sharepoint.com/personal/nicolas_gascon_epitech_eu/_layouts/15/guestaccess.aspx?guestaccesstoken=MN0A%2fHlE9z2OOwvsLOskEeRSu3iJZA44LbHO%2bwDDCcU%3d&docid=00c89e2b700b14ad5a53a14c080206797&rev=1

Phaser est un framework (un ensemble de librairies) permettant d'afficher des images sur une page HTML, jouer du son, récupérer les inputs (touches du clavier entrées par l'ordinateur), utiliser de la physique, des animations et plein d'autres choses !

3 – Création de la page

Tout d'abord nous allons commencer par rapidement créer une page internet qui importera un fichier JavaScript de Phaser ainsi que le fichier JavaScript que nous allons créer.

Commençons par créer un fichier « `index.html` », initialisons-le comme un fichier basique en déclarant le `head` et le `body`, puis dans le `head` mettez un titre. Si vous voulez des rappels sur le HTML je vous invite à regarder ce lien :

https://epitechfr-my.sharepoint.com/personal/nicolas_gascon_epitech_eu/_layouts/15/guestaccess.aspx?guestaccesstoken=m8oyxmFWyp9ao%2f1EXyiUvENIPPsCEr3GhgO1XzPctcc%3d&docid=066ed681f34ee4070980a7bb3af793e2c&rev=1

(Si vous voulez que votre page supporte les accents rajoutez : « `<meta charset="utf-8" />` » entre les balises `head`).

Voici à quoi devrait ressembler la page :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>Platformer</title>
    </head>
    <body>
    </body>
</html>
```

Maintenant nous allons voir un nouvel attribut HTML qui se nomme « `script` ». Celui-ci a plusieurs particularités. Nous pouvons l'utiliser afin d'écrire directement du code JavaScript qui sera interprété. Dans notre cas, il nous servira à importer un fichier JavaScript.

La balise contient donc plusieurs paramètres à entrer :

- `type` : Le type du fichier.
- `src` : Le chemin source du fichier.

Dans notre cas, nous allons dans un premier temps importer Phaser.

Pour cela, à côté de votre fichier `index.html`, faites un dossier « `js` » et ajoutez-y `phaser.min.js` à l'intérieur.

Ensuite, en dessous de `title` dans le fichier `index.html` avec le `</head>`, ajoutez-y la balise `script` afin d'importer Phaser comme ceci :

```
<script type="text/javascript" src="js/phaser.min.js"></script>
```

Ici, nous pouvons donc voir que le type est un texte en JavaScript et qu'il importe le fichier dans le dossier `js` (tout ce qui se trouve avant le dernier « `/` » sont des dossiers depuis le fichier HTML) `phaser.min.js`.

Si vous vous demandez pourquoi l'extension du fichier est `.min.js` et non `.js` c'est juste parce que dans cette version ils ont enlevés les espaces afin d'accélérer la lecture du code par l'ordinateur.

Juste en dessous, en suivant la première balise `script`, importez le fichier `main.js` qui se trouve dans le dossier `js`. Le chemin est donc « `js/main.js` »

Une fois que vous avez réussi à importer ceci, direction le `main.js` pour commencer à créer notre jeu !

4 – Initialisation

4.1 – Les variables

Commençons par initialiser notre jeu !

On va commencer par voir plusieurs notions des langages informatique, la première ? Les variables !

Qu'est-ce qu'une variable ? Une variable sert à stocker une donnée, un nombre, un texte (une chaîne de caractères) ou plein d'autres types comme des nombres à virgules etc...

Concrètement à quoi ça sert ? Nous pouvons tout simplement dire que les positions du joueur en X sont dans une variable « `x` », grâce à cela nous pouvons accéder à ses positions simplement en écrivant `x`, mais également modifier cette variable ! Si le joueur se déplace de 4 cases par exemple, nous pouvons dire que « `x = x + 4` ». Cela nous donne donc la bonne valeur en utilisant toujours la variable nommée `x` et cela est bien utile !

Pour créer une variable en JavaScript c'est très simple, il suffit de l'écrire comme ceci :

« `var positionX = 0;` »

Ici nous disons à l'ordinateur que nous faisons une variable (`var`) qui s'appelle `positionX` et qui vaut `0` à son initialisation (le moment où elle est créée).

Ensuite si nous voulons ajouter 4 à cette variable nous pouvons l'écrire de cette façon :

« `positionX = positionX + 4;` »

Cela veut dire que notre variable `positionX` va prendre la valeur (=) égale à `positionX` (anciennement `0`) + 4.

Notre variable vaudra donc `4` après cette ligne.

Ensuite, pour afficher du texte ou une variable, il faut appeler une fonction.

4.2 – Les fonctions

Une fonction est un bout de code qui est entouré d'accolades `{` et `}`, un peu comme les balises ouvrantes et fermantes en HTML, et tout ce qui se trouve à l'intérieur sera exécuté lorsque vous l'appellerez.

Cela sert à plusieurs choses : premièrement, à éviter de copier du code qui fait la même chose, mais également à rendre le code plus lisible ou plus générique (que nous pouvons utiliser dans plusieurs cas).

Nous allons donc commencer simplement en allant sur ce site : http://math.chapman.edu/~jipsen/js/

Dans le cadre à gauche, effacez tout et ajoutez-y les deux lignes que nous avons pu voir plus haut :

« `var positionX = 0;` »

« `positionX = positionX + 4;` »

Maintenant nous allons donc afficher sur le cadre à droite sa valeur.

Pour cela il y a une fonction qui s'appelle « `writeln` » pour écrire une valeur et effectuer un retour à la ligne (et non pas continuer à la suite la prochaine écriture lorsqu'il y en a plusieurs).

Pour appeler cette fonction il suffit d'écrire son nom suivi de parenthèses ouvrantes et fermantes et du fameux « `;` » qui sert à dire que nous finissons une instruction (déclaration de variable, appel de fonction, addition, etc...) .

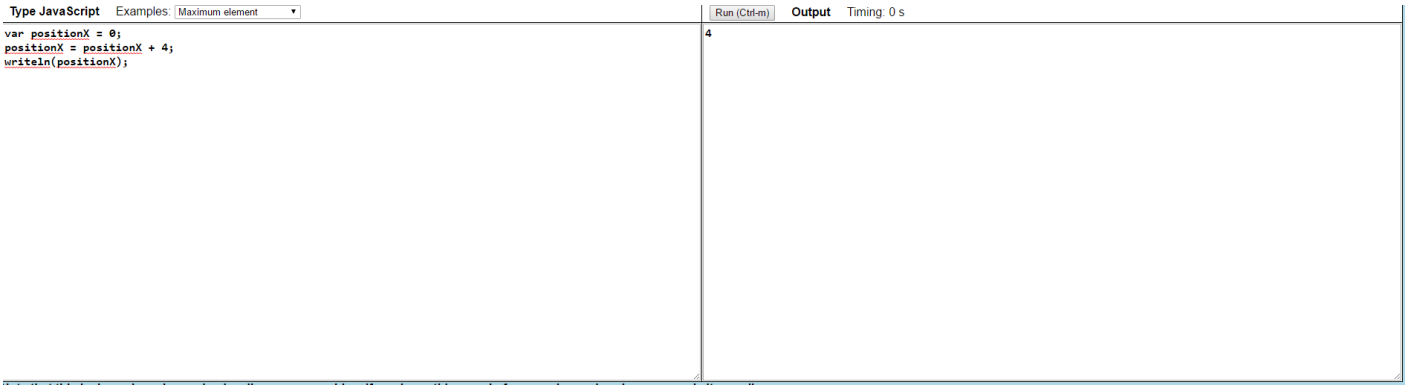
Entre ces parenthèses il va nous falloir un paramètre. Un paramètre c'est une variable dont la fonction a besoin afin d'effectuer son travail, cela peut être un chiffre, une chaîne de caractère (un texte), tout dépend du but de la fonction et de ce qu'elle doit effectuer.

Ici « `writeln` » prend tout ce que l'on veut en paramètre et l'écrit dans le cadre à droite.

Nous pouvons donc lui passer directement `positionX` tel que ceci :

« `writeln(positionX);` »

Si vous appuyez sur Run au-dessus du deuxième cadre, vous devriez avoir `4` qui s'affiche, ce qui veut dire que nous avons bien modifié `positionX` en lui ajoutant 4 !



Vous pouvez essayer en lui ajoutant plus ou moins que 4 ou bien en modifiant sa valeur par défaut qui est ici `0`.

Nous allons également devoir créer des fonctions pour réaliser notre jeu, pour cela JavaScript a une syntaxe un peu particulière, elle va stocker la fonction dans une variable.

On va donc faire une fonction qui va prendre deux nombres en paramètres, les additionner puis les afficher.

Pour créer une fonction la syntaxe est la suivante :

```
var nom = function(param1, param2) {
}
```

Les paramètres sont facultatifs, vous pouvez ne pas en mettre si votre fonction n'en a pas besoin.

Ici nous voyons donc que pour écrire la fonction nous allons devoir écrire entre les accolades `{}` qui délimitent la fonction.

Commençons par faire une variable `result` qui est égale à « `param1 + param2` » puis afficher `result`.

Voici à quoi devrait ressembler la fonction :

```
var add = function(param1, param2) {
    var result = param1 + param2;
    writeln(result);
}
```

Une fois que vous avez ceci, en dessous vous pouvez appeler cette fonction sans forcément préciser une variable mais sa valeur directement tel que :

« `add(1, 2);` »

Une fois lancé cela devrait nous afficher `3`.

Vous avez donc réussi à créer une fonction et l'appeler, c'est un très bon début !

Plus que trois petites choses à voir avant de commencer et nous serons prêt !

4.3 – Les listes

Nous allons commencer par les listes !

Une liste qu'est-ce que c'est ?

Une liste est déclarée de cette manière :

« `var liste = []` »

« `var liste = [1, 2, 3, 4]` »

Ici la liste possède deux versions, une sans valeur et une avec plusieurs valeurs. La liste permet donc de contenir plusieurs variables dans un seul objet, d'y accéder à l'aide des crochets `[]` en lui mettant l'index de la valeur que nous voulons (le n du n-ième élément - 1, une liste commence toujours à 0 donc si nous voulons le deuxième élément, à savoir le `2` de la deuxième liste, il nous faudra écrire `liste[1]` »

Nous pouvons également lui ajouter plusieurs variables dans la code au fur et à mesure que nous avançons. Cela nous aidera lorsque nous lirons la carte pour ajouter toutes les pièces, les murs, etc...

4.4 – Les chaînes de caractères

Une chaîne de caractères, ou bien un texte, est définie comme ceci :

« `var texte = 'Ceci est un texte'` »

Cela nous permet de stocker un texte que nous avons écrit, mais ce que je voulais vous préciser, c'est que la chaîne de caractères est simplement une liste de caractères. Un caractère est donc une lettre, ou bien un caractère spécial comme : `"*"`, `";"`, `","`, `"!"` ou encore `"$"`.

Ils sont tous des caractères que nous pouvons mettre dans la liste et nous pouvons donc accéder à un caractère en particulier à l'aide des crochets. Si nous voulons changer le « `i` » dans le texte qui est le 4ème caractère par un « `o` » il nous suffit donc d'écrire :

« `texte[3] = 'o';` »

Pensez bien que pour écrire un caractère il faut mettre les « `'` » entre sinon il cherchera une variable.

Nous pouvons également afficher un texte ou une ligne à l'aide de `writeln`.

4.5 – Les objets

Les objets sont la chose la plus complexe que vous allez voir en JavaScript. Ils sont définis également par des accolades, et sont des variables qui... possèdent des variables.

Comment ça ?

Ben si j'ai une variable d'un objet qui s'appelle `Player`, je peux avoir une variable `X` dans `Player` qui correspond à sa position en X ou la même chose pour Y.

Les variables à l'intérieur de l'objet sont séparées par des ','.

Voici un exemple avec `Player` qui contient `X` et `Y` qui valent `0` par défaut.

```
var Player = {
    X: 0,
    Y: 0
}
```

Faites très attention à la syntaxe !

A l'intérieur de l'objet nous avons donc une variable `X` qui vaut `0` et une variable `Y` qui vaut également `0`.

La déclaration de la variable dans un objet se fait différemment. Il n'y a pas le mot-clé « `var` » ni le signe égal, seulement le « `nom : valeur` ».

La valeur peut également être une fonction.

Ensuite si nous voulons accéder au `X` du `Player` il nous suffit d'écrire :

« `Player.X` »

On peut donc attribuer sa valeur par la suite en faisant :

« `Player.X = Player.X + 4;` »

Ce qui l'augmentera de 4.

Nous avons donc vu les plus grandes parties concernant JavaScript que nous allons utiliser !

Il restera des petites choses à voir tel que les conditions ou bien les boucles mais nous verrons ça plus tard :)

5 – Commencement

A côté de votre fichier « `index.html` » vous pouvez créer un dossier `assets` et copier à l'intérieur les fichiers téléchargeables ici :

https://epitechfr-my.sharepoint.com/personal/nicolas_gascon_epitech_eu/_layouts/15/guestaccess.aspx?guestaccesstoken=MN0A%2fHlE9z2OOwvsLOskEeRSu3iJZA44LbHO%2bwDDCcU%3d&docid=00c89e2b700b14ad5a53a14c080206797&rev=1

Ensuite faîtes un fichier `main.js` dans le dossier `js` et nous allons commencer le jeu !

5.1 – L'état de jeu

Notre première étape sera de créer un objet qui s'appellera « `mainState` » et qui contiendra 3 fonctions nommées `preLoad`, `create` et `update`.

Souvenez-vous, les fonctions ne sont pas définies de la même manière dans un objet (`nom : function() {}, nom2 : function(){}`)

Essayez de le faire de votre côté avant de regarder la solution en page suivante.

```
var mainState = {
    preload: function() {
    },

    create: function() {
    },

    update: function() {
    }
};
```

Voilà notre objet, ce sera « l'état » de note jeu.

La fonction `preload` sera appelée au chargement du jeu, la fonction `create` sera appelée à la création du jeu/de la carte (après le chargement) et la fonction `update` sera appelée à chaque génération d'image (il faut savoir qu'un jeu tourne souvent à 30, 60 ou 120 images par seconde qui sont générées en fonction des positions des objets dans la carte).

Ensuite il va falloir initialiser le jeu.

Pour cela nous allons faire une variable « `game` » qui contiendra le jeu :

« `var game = new Phaser.Game(500, 200);` »

Cette ligne permet d'appeler la fonction `Game` dans le fichier `phaser` et en paramètre nous pouvons lui dire les dimensions de la fenêtre que nous voulons (ici 500 pixels de largeur pour 200 pixels de hauteur).

Ensuite nous allons ajouter notre variable `mainState` au jeu :

« `game.state.add('main', mainState);` »

Ici nous ajoutons un état à la variable `game` (qui est un objet). L'état se nomme « `main` » , ce qui servira à le lancer et correspond à la variable « `mainState` » créée plus haut.

Pour finir l'initialisation du jeu nous allons lancer l'état à l'aide de la fonction `start` dans `game.state` :

« `game.state.start('main');` »

Nous pouvons voir que l'on réutilise donc le nom pour lancer l'état.

Cette ligne nous servira également à relancer le jeu lorsque le joueur mourra.

Voici le code que vous devriez avoir à présent :

```
var mainState = {
    preload: function() {
    },

    create: function() {
    },

    update: function() {
    }
};

var game = new Phaser.Game(500, 200);
game.state.add('main', mainState);
game.state.start('main');
```

Le reste de notre jeu se passera donc dans les fonctions `preload`, `create` et `update` !

5.2 – Le chargement

Nous allons donc commencer par la fonction `preload`, elle nous servira à charger les images de notre jeu.

Premièrement mettez le dossier téléchargé dans l'archive `assets.zip` à côté du fichier `index.html`, cela nous servira à avoir les images à afficher.

Retournons ensuite dans le fichier `main.js`.

Dans la fonction `preload` nous allons appeler plusieurs fois une fonction pour charger chacune des textures :

« `game.load.image('player', 'assets/player.png');` »

Nous devons donc appeler cette fonction plusieurs fois pour :

« `'wall', 'assets/wall.png'` »

« `'coin.png', 'assets/coin.png'` »

« `'enemy', 'assets/enemy.png'` »

Nous pouvons donc comprendre qu'il s'agit de paramètres, en premier le nom qui nous servira à utiliser la texture et en deuxième le chemin vers la texture.

Normalement maintenant nous devrions donc avoir chargé toutes nos textures !

```
preload: function() {
    game.load.image('player', 'assets/player.png');
    game.load.image('wall', 'assets/wall.png');
    game.load.image('coin', 'assets/coin.png');
    game.load.image('enemy', 'assets/enemy.png');
},
```

Maintenant que nous chargeons nos textures nous pouvons donc commencer à coder la fonction `create` !

6 – Create

La fonction `create` va nous servir à initialiser notre jeu.

La première chose que nous allons faire dedans c'est modifier la couleur du background.

Pour cela il suffit juste de changer la valeur « `backgroundColor` » dans « `game.stage` ».

```
game.stage.backgroundColor = '#3598db';
```

Ici nous lui donnons la valeur '`#3598db`'.

Cette valeur est une couleur en RGB (les composantes rouge, verte et bleue). Chacune des composantes possède une valeur en 0 et 255 représentée en hexadécimal (de 0 à FF).

Ensuite nous allons pourvoir démarrer la physique du jeu.

La physique sert à se déplacer mais également à utiliser les collisions (bloquer le joueur ou détecter une collision).

```
game.physics.startSystem(Phaser.Physics.ARCADE);
```

Ici nous démarrons donc le système physique du jeu avec une physique de type ARCADE (Phaser possède plusieurs types de physiques suivant ce que vous voulez faire).

Il faut ensuite dire que nous allons utiliser la physique sur tous les objets :

```
game.world.enableBody = true;
```

Maintenant nous allons initialiser plusieurs variables dont nous aurons besoin.

La première est simplement le clavier afin de savoir si le joueur appuie ou non sur une touche !

« `this.cursor = game.input.keyboard.createCursorKeys();` »

Ici nous appelons la fonction `createCursorKeys()` qui est dans `game.input.keyboard` et nous stockons son résultat dans `this.cursor`.

Le `this`, lorsque nous sommes dans un objet nous sert à définir une nouvelle variable à l'intérieur. Nous pourrons donc y accéder depuis les autres fonctions de l'objet notamment la fonction « `update` ».

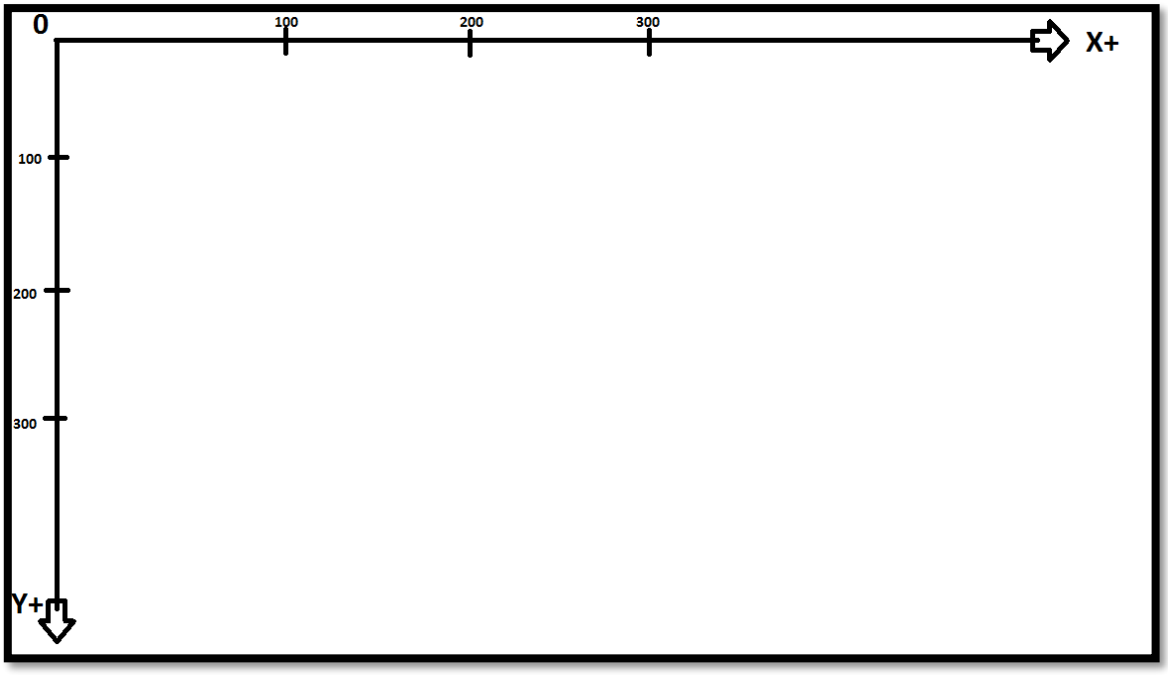
Ensuite nous allons créer notre premier sprite, qui sera le joueur, et le stocker dans une variable joueur dans l'objet. Pour cela nous devons faire :

```
this.player = game.add.sprite(70, 100, 'player');
```

`game.add.sprite` nous sert à ajouter un sprite au jeu. Le sprite défini est '`player`' (le nom utilisé lors du chargement) et qui sera en position 70, 100.

Les positions correspondent aux positions par rapport à l'écran que nous avons défini plus haut.

Sachant que les positions sont définies de cette manière :



Donc plus la valeur en X (largeur) est haute, plus nous serons à droite et plus la valeur en Y (hauteur) est haute, plus nous serons en bas.

En regardant l'image des positions au-dessus, vous pouvez vous imaginer où seront placés vos objets pour des positions données.

Ensuite nous allons modifier une valeur dans notre variable « `player` », la valeur que nous allons modifier est la gravité afin de descendre si il n'y a pas d'objets qui nous bloquent et donc tomber lors d'un saut par exemple.

Pour cela nous devons modifier la variable « `gravity.y` » dans « `this.player.body` ».

```
this.player.body.gravity.y = 600;
```

Ici nous modifions la gravité afin de la mettre à 600, cela nous donnera une gravité d'un jeu de plateforme qui reste un minimum en l'air lors d'un saut.

Venons-en à la partie compliquée de la fonction !

Cette partie consiste à créer la carte et la faire apparaître à l'écran.

Tout d'abord nous allons initialiser trois « listes ». Ce ne sont pas de réelles listes, elles sont faîtes spécialement pour Phaser afin de l'utiliser dans l'affichage ainsi que la physique.

```
this.walls = game.add.group();
this.coins = game.add.group();
this.enemies = game.add.group();
```

Ici nous initialisons donc 3 listes différentes : `walls` qui contiendra les murs de la carte afin de bloquer le joueur, `coins` qui contiendra les pièces que nous pouvons ramasser ainsi que `enemies` qui contiendra des cases qui tueront le joueur !

Maintenant vous voulez sans doute savoir comment nous allons initialiser la carte ainsi que remplir nos listes.

Pour cela nous allons réaliser une liste de chaînes de caractères pour avoir une carte avec du texte et où les caractères correspondront à des éléments différents.

Comme caractères nous aurons :

- '`x`' : Les murs
- '`!`' : Les blocs ennemis
- '`o`' : Les pièces
- ' ' (espace) : Vide

Essayez de réaliser la liste à l'aide des explications plus haut concernant la liste avant de regarder la correction puis assurez-vous d'avoir compris après ! Si vous avez besoin d'aide, appelez un Cobra !

```
var level = [
    'xxxxxxxxxxxxxxxxxxxxxx',
    '!           !        x',
    '!                 o x',
    '!          o       x',
    '!                  x',
    '!      o   !   x    x',
    'xxxxxxxxxxxxxxxx!!!!!x'
]
```

Et voici comment est donc notre carte. Si vous êtes motivés plus tard, vous pourrez ajouter une lecture de la carte dans un fichier, plusieurs niveaux, etc... !

Ensuite nous allons ajouter les sprites à la carte en fonction du texte.

Je vais ici vous donner tout le code et vous l'expliquer par la suite.

```
for (var i = 0; i < level.length; i++) {
    for (var j = 0; j < level[i].length; j++) {
        if (level[i][j] == 'x') {
            var wall = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'wall');
            this.walls.add(wall);
            wall.body.immovable = true;
        }
        else if (level[i][j] == 'o') {
            var coin = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'coin');
            this.coins.add(coin);
        }
        else if (level[i][j] == '!') {
            var enemy = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'enemy');
            this.enemies.add(enemy);
        }
    }
}
```

La première chose que nous faisons est une boucle qui sert à parcourir tous les éléments de la liste. Ensuite nous avons une autre boucle afin de parcourir tous les caractères qui nous servirons à savoir quoi afficher !

Pour accéder à un caractère ou une liste nous devons utiliser les '`[]`' qui contiennent un index `[0]`, `[1]` pour accéder au (n + 1)ième élément. (D'où le fait que nous commencions à 0).

Ensuite nous avons une condition : le `if` !

A quoi sert un `if` ?

Le `if` nous sert à vérifier une condition, ici si `level[i][j]`, où `i` correspond au numéro de la ligne et `j` au numéro du caractère, est égal à '`x`'.

Lors d'une condition pour vérifier une égalité il faut écrire '==' et non '='. Si vous utilisez le simple égal cela voudrait dire que le caractère en position `x` et `j` vaut '`x`' et donc vous ne demandez pas s'il est égal à `x`.

Les autres conditions `else if` veulent seulement dire sinon si, ce sont des conditions qui ne seront pas vérifiées si une des conditions au-delà est correcte.

Ensuite pour chaque caractère que nous utilisons, nous ajoutons le sprite à des positions calculées en fonction de leurs positions dans la liste pour chaque type, puis ajoutons le sprite à une des 3 listes que nous avons déclarées.

Dans le cas du mur il y a un ajout qui est « `wall.body.immovable = true;` », cela sert à nous dire que le mur ne bougera jamais alors que les pièces disparaitront et les ennemis pourront se déplacer si vous le leur faîtes faire.

Et voilà nous avons fini la fonction `create` !

Si tout se passe bien, si vous ouvrez `index.html` vous devriez avoir la carte dessinée sur la page !

Et voici en page suivante le code complet de la fonction :

```
create: function() {
    game.stage.backgroundColor = '#3598db';
    game.physics.startSystem(Phaser.Physics.ARCADE);
    game.world.enableBody = true;

    this.cursor = game.input.keyboard.createCursorKeys();
    this.player = game.add.sprite(70, 100, 'player');
    this.player.body.gravity.y = 600;

    this.walls = game.add.group();
    this.coins = game.add.group();
    this.enemies = game.add.group();

    var level = [
        'xxxxxxxxxxxxxxxxxxxxx',
        '!          !         x',
        '!                  o x',
        '!          o         x',
        '!                    x',
        '!      o   !   x     x',
        'xxxxxxxxxxxxxxxx!!!!!x'
    ]

    for (var i = 0; i < level.length; i++) {
        for (var j = 0; j < level[i].length; j++) {
            if (level[i][j] == 'x') {
                var wall = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'wall');
                this.walls.add(wall);
                wall.body.immovable = true;
            }
            else if (level[i][j] == 'o') {
                var coin = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'coin');
                this.coins.add(coin);
            }
            else if (level[i][j] == '!') {
                var enemy = game.add.sprite(30 + 20 * j, 30 + 20 * i, 'enemy');
                this.enemies.add(enemy);
            }
        }
    }
},
```

7 – Update

Faisons maintenant la fonction la plus intéressante et celle qui fera votre jeu !

Cette fonction sera bien plus courte que la dernière ne vous en faites pas, et vous pourrez après vous amuser !

Allons donc dans `update` et commençons.

La première chose que nous allons ajouter est la capacité à être bloqué par le mur et la capacité à détecter lorsque nous allons sur une pièce ou un ennemi.

```
game.physics.arcade.collide(this.player, this.walls);
game.physics.arcade.overlap(this.player, this.coins, this.takeCoin, null, this);
game.physics.arcade.overlap(this.player, this.enemies, this.restart, null, this);
```

Pour cela il nous faut appeler des fonctions de `game.physics.arcade`.

`arcade` puisque nous avons choisi le système d'arcade pour la physique du jeu.

La fonction `collide` sert à nous bloquer lorsque nous rentrons en collision avec un des objets ajoutés. Ici nous ajoutons donc le joueur dont nous voulons vérifier les collisions avec les différents sprites du mur.

La fonction `overlap` sert à appeler une fonction lors d'une collision.

Ici nous l'utilisons avec les pièces et les ennemis auxquels cas nous appelons `takeCoin` ou `restart` en cas de collision. Nous ajouterons les fonctions plus tard, elles font chacune une ligne.

Ensuite venons-en au déplacement !

Nous allons devoir utiliser les conditions « `if` » pour savoir si le joueur a appuyé ou non sur une touche.

```
if (this.cursor.left.isDown)
    this.player.body.velocity.x = -200
else if (this.cursor.right.isDown)
    this.player.body.velocity.x = 200;
else
    this.player.body.velocity.x = 0;
```

Rappelez-vous, `cursor` a été défini dans `create` et qui équivaut au clavier. Ensuite nous vérifions si `left` et `right` sont appuyés. « `isDown` » renvoie une valeur vrai ou faux afin de savoir si la touche est appuyée.

Dans le cas où une touche (ici les flèches droite et gauche) est appuyée, nous mettons la vélocité du joueur à 200 ou -200, cela correspond à une vitesse. Si la vitesse est positive le personnage ira vers la droite, si c'est négatif vers la gauche.

Ensuite il y a un `else` tout seul, cela veut dire que si aucune des conditions ci-dessus n'est vérifiée, nous mettons sa vitesse à 0 afin d'éviter de glisser.

Maintenant il reste un petit cas à vérifier. Le saut !

Nous allons vérifier que la touche haut est appuyée comme tout à l'heure mais également si le sprite touche le sol. Pour vérifier deux conditions nous les séparons par « `&&` » donc « `if (condition1 && condition2)` ».

Si le cas est vérifié nous mettons la vélocité en y à -250.

Et oui bien en négatif afin d'aller vers le haut de l'écran !

Sans attendre voici les conditions à vérifier :

```
if (this.cursor.up.isDown && this.player.body.touching.down)
    this.player.body.velocity.y = -250;
```

Et voilà pour la fonction update, maintenant nous n'avons plus qu'à rajouter les fonctions `takeCoin` et `restart` puis ce sera à votre tour d'ajouter du contenu !

```
update: function() {
    game.physics.arcade.collide(this.player, this.walls);
    game.physics.arcade.overlap(this.player, this.coins, this.takeCoin, null, this);
    game.physics.arcade.overlap(this.player, this.enemies, this.restart, null, this);

    if (this.cursor.left.isDown)
        this.player.body.velocity.x = -200
    else if (this.cursor.right.isDown)
        this.player.body.velocity.x = 200;
    else
        this.player.body.velocity.x = 0;

    if (this.cursor.up.isDown && this.player.body.touching.down)
        this.player.body.velocity.y = -250;
},
```

8 – takeCoin et restart

Nous allons donc créer deux fonctions à la manière d'`update`, `create` et `preload` avec les noms spécifiés ci-dessus.

La seule différence est que la fonction `takeCoin` prendre deux paramètres `(player, coin)`.

```
takeCoin: function(player, coin) {
    coin.kill();
},

restart: function() {
    game.state.start('main');
}
```

Ensuite les cas à détruire, pour la pièce c'est simplement de la détruire en appelant la fonction `kill()` dans `coin`. Vous pourrez ici vous amusez à créer un compteur de pièce par exemple ou autres...

Dans le cas de `restart` nous allons simplement lancer l'état `main` comme lors du lancement du jeu !

Fin !

Je vous fournis le code du jeu ici afin que vous puissiez vérifier votre code au cas où vous auriez un problème.

Maintenant j'aimerais que vous vous entraidiez, et lorsque plusieurs personnes auront terminé, nous ferons un brainstorming afin d'imaginer des ajouts que nous pourrons coder au jeu !

N'hésitez pas à communiquer entre vous ! :)

Bravo, vous êtes arrivés au bout de ce tutoriel ! Vous avez maintenant une base solide d'un jeu que vous pouvez facilement améliorer et intégrer à un site web. Laissez libre cours à votre imagination, et n'hésitez pas à demander aux Cobras de vous aider !

Écriture tutoriel et code source fourni : Nicolas Gascon

Mise en page et correction linguistique : Hugo Perez