Tutoriel Phaser

Préparation...

Avant toute chose, il vous faut mettre en place votre environnement de travail et de test.

Vous avez plusieurs possibilités :

- Installer une extension de Visual Studio Code permettant une prévisualisation passant par un serveur web local. Exemple : "Live Server Preview".
- Télécharger Brackets et vous servir de son outil "aperçu en direct" pour vos tests.

Dans tous les cas, sachez qu'il n'est pas possible de faire fonctionner un jeu exploitant la bibliothèque Phaser si vous ouvrez simplement votre navigateur en y indiquant une URL de type "C:\Mes Documents/etc". L'URL commencera forcément par 127.0.0.1 ou par *localhost*.

Vous allez avoir besoin de quelques images pour que cela fonctionne. Téléchargez ce pack d'images afin d'avoir de quoi faire vos premiers tests.

Mise en place du moteur graphique

Créez un fichier .html comme suit :

```
<!doctype html>
<html lang="en">
<head>
    <meta charset="UTF-8" /><title>Mon 1er jeu Phaser</title>
    <script
src="//cdn.jsdelivr.net/npm/phaser@3.11.0/dist/phaser.js"></script>
    <style type="text/css"> body { margin: 0; }</style>
</head>
```

```
<body>
<script type="text/javascript">
</script>
</body>
</html>
```

Entre les balises <script>, vous pouvez ajouter les instructions javascript suivantes :

```
var config = {
    type: Phaser.AUTO,
    width: 800, height: 600,
    scene: {preload: preload, create: create, update: update }
};

new Phaser.Game(config);

function preload(){
    this.load.image('sky', 'assets/sky.png');
    this.load.image('ground', 'assets/platform.png');
    this.load.image('star', 'assets/star.png');
    this.load.image('bomb', 'assets/bomb.png');
    this.load.spritesheet('perso','assets/perso.png',
        { frameWidth: 32, frameHeight: 48 });
}

function create(){
    this.add.image(400, 300, 'sky');
    this.add.image(400, 300, 'star');
}

function update(){}
```

Vous devez alors observer une étoile sur un fond bleu dégradé.

Décortiquons ensemble les différentes parties du code javascript.

- une variable config, dans laquelle on range différents paramètres : le type de rendu graphique (on utilise la valeur par défaut, `Phaser.AUTO`), la largeur et la hauteur de la zone de jeu (800x600)), ainsi que les fonctions qui vont définir notre scène.
- le lancement du moteur de jeu proprement dit, tenant compte de la configuration.
- on implémente la fonction de `preload()`. Cette fonction permet de précharger en mémoire les assets de notre jeu. C'est une étape nécessaire avant de les afficher.
- La fonction `create()` est celle qui nous permet de gérer les affichages initiaux.

Essayez d'inverser les lignes affichant le ciel et l'étoile. Que remarquez-vous ?

Mise en place du moteur physique

Dans la variable de configuration, entre les paramètres `height` et `scene`, on ajoute le paramètre suivant :

```
physics: {
        default: 'arcade',
        arcade: {
            gravity: { y: 300 },
            debug: false
        }},
```

Il nous permet d'initialiser aussi un moteur physique. Cela va nous autoriser l'ajout de plateformes dans notre jeu.

Juste avant la fonction create, nous allons pouvoir définir une variable *platforms*, qui va nous permettre de gérer toutes les plateformes de notre jeu. Nous ajoutons quelques instructions à la fonction create afin d'en initialiser quelques unes. Nous obtenons le résultat suivant :

```
var platforms;

function create (){
    this.add.image(400, 300, 'sky');

    platforms = this.physics.add.staticGroup();
    platforms.create(400, 568, 'ground').setScale(2).refreshBody();
    platforms.create(600, 400, 'ground');
    platforms.create(50, 250, 'ground');
    platforms.create(750, 220, 'ground');
}
```

Faites le test. Vous devriez obtenir plusieurs plateformes.
N'hésitez pas à les déplacer en changeant les paramètres des `platforms.create()` !

Vous pouvez constater qu'elles sont gérées en un groupe. Ouvrez une console javascript et observez le contenu de la variable *platforms*. Que constatez-vous ? Est-il possible d'accéder à une plateforme de façon indépendante ?

Ajout d'un personnage !

Déclarez une variable *player* au même endroit que vous avez déjà créée une variable platforms. Ensuite, ajoutez à create les instructions suivantes :

```
    player = this.physics.add.sprite(100, 450, 'perso');
    player.setBounce(0.2);
    player.setCollideWorldBounds(true);
    this.physics.add.collider(player, platforms);

    this.anims.create({
        key: 'left',
        frames: this.anims.generateFrameNumbers('perso', {start:0,end:3}),
        frameRate: 10,
        repeat: -1
    });

    this.anims.create({
        key: 'turn',
        frames: [ { key: 'perso', frame: 4 } ],
        frameRate: 20
    });

    this.anims.create({
        key: 'right',
        frames: this.anims.generateFrameNumbers('perso', {start:5,end:8}),
        frameRate: 10,
        repeat: -1
    });
```

La 1ère ligne permet de générer le sprite du personnage à partir de la spritesheet. La 2nde définit un rebond lors du contact avec le sol. La 3e et la 4e permettent au personnage de collisionner respectivement avec les bords du niveau et avec les plateformes..

Les 3 autres instructions correspondent à des créations d'animations : aller vers la gauche, faire face à la caméra, aller vers la droite. On peut remarquer au passage qu'une animation consiste en un nombre de frames que Phaser va pouvoir découper automatiquement dans la spritesheet, en fonction de la taille définie lors du préchargement de l'asset.

Gestion des contrôles

Déclarez une variable *cursors* au même endroit que les variables *player* et *platforms*.

Dans la fonction `create()`, ajoutez la ligne suivante. Elle permet, lors de la mise en place du jeu, de générer la prise en charge des touches directionnelles du clavier.

```
    cursors = this.input.keyboard.createCursorKeys();
```

Ensuite, la gestion des contrôles en elle-même se déroule dans la fonction `update()`. Cette fonction est assez particulière, et réellement centrale dans le fonctionnement du jeu : c'est la fonction qui sera exécutée lors de chaque frame du jeu. Autrement dit, 60 fois par seconde dans le cas d'un jeu en 60FPS.

```
    if (cursors.left.isDown){ //si la touche gauche est appuyée
        player.setVelocityX(-160); //alors vitesse négative en X
        player.anims.play('left', true); //et animation => gauche
    }
    else if (cursors.right.isDown){ //sinon si la touche droite est appuyée
        player.setVelocityX(160); //alors vitesse positive en X
        player.anims.play('right', true); //et animation => droite
    }
    else{ // sinon
        player.setVelocityX(0); //vitesse nulle
        player.anims.play('turn'); //animation fait face caméra
    }

    if (cursors.up.isDown && player.body.touching.down){
        //si touche haut appuyée ET que le perso touche le sol
        player.setVelocityY(-330); //alors vitesse verticale négative
        //(on saute)
    }
```

Vous pouvez essayer d'expérimenter différentes valeurs de vitesse.

Score et collectibles

Déclarez des variables *stars*, *score*, et *scoreText*, au même endroit que les autres variables. Initialisez *score* à 0.

Dans la fonction `create()`, on créé un groupe de 12 étoiles (une tous les 70 pixels).

```
    scoreText=this.add.text(16,16,'score: 0',{fontSize:'32px',fill:'#000'});
     //affiche un texte à l'écran, pour le score

    stars = this.physics.add.group({
        key: 'star', repeat: 11,
        setXY: { x: 12, y: 0, stepX: 70 }
    });

    stars.children.iterate(function (child) {
        child.setBounceY(Phaser.Math.FloatBetween(0.4, 0.8));
    }); //chaque étoile va rebondir un peu différemment
    this.physics.add.collider(stars, platforms);
                    //et collisionne avec les plateformes
    this.physics.add.overlap(player, stars, collectStar, null, this);
    //le contact perso/étoile ne génère pas de collision (overlap)
    //mais en revanche cela déclenche une fonction collectStar
```

On doit alors créer cette fonction collectStar, en dehors des autres fonctions.

```
function collectStar(player, star){
    star.disableBody(true, true); // l'étoile disparaît
    score += 10; //augmente le score de 10
    scoreText.setText('Score: ' + score); //met à jour l'affichage du score
}
```

Ennemis et *game over*

De la même façon que précédemment, déclarez une variable *bombs* et `gameOver = false;`
Implémentons les bombes dans `create()`, similairement aux étoiles.

```
    bombs = this.physics.add.group();
    this.physics.add.collider(bombs, platforms);
    this.physics.add.collider(player, bombs, hitBomb, null, this);
```

Nous devons donc créer la fonction hitBomb, en dehors des autres fonctions.

```
function hitBomb(player, bomb){
    this.physics.pause();
    player.setTint(0xff0000);
    player.anims.play('turn');
    gameOver = true;
}
```

On ajoute ce petit code à collectStar pour que quand on ramasse toutes les étoiles du niveau, ça génère de nouvelles étoiles, ainsi qu'une bombe qui rebondit dans tout le niveau :

```
    if (stars.countActive(true) === 0){// si toutes les étoiles sont prises
        stars.children.iterate(function (child) {
            child.enableBody(true, child.x, 0, true, true);
        }); // on les affiche toutes de nouveau
        var x = (player.x < 400) ? Phaser.Math.Between(400, 800) :
                                   Phaser.Math.Between(0, 400);
        // si le perso est à gauche de l'écran, on met une bombe à droite
        // si non, on la met à gauche de l'écran
        var bomb = bombs.create(x, 16, 'bomb');
        bomb.setBounce(1);
        bomb.setCollideWorldBounds(true);
        bomb.setVelocity(Phaser.Math.Between(-200, 200), 20);
        bomb.allowGravity = false; //elle n'est pas soumise à la gravité
    }
```

On peut dire au début de `update()` que le `gameOver` empêche de jouer :

```
    if (gameOver){return;}
```