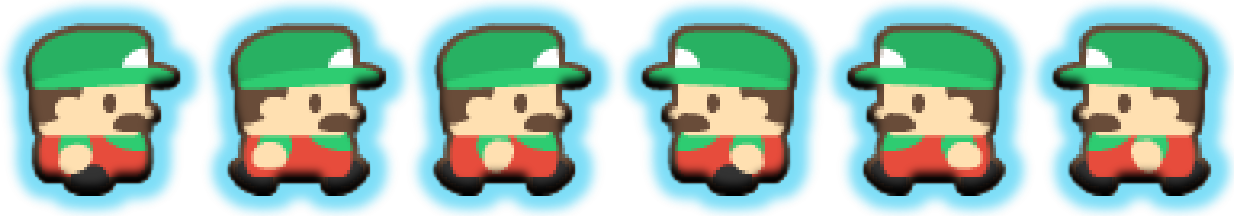
MASTER 1 –PROGRAMMATION WEB AVANCE

Projet Web

Jeu de plate-forme

Cousin Gaëtan & Rosaz Lucas

12/04/2017



L'objectif de ce projet est de concevoir un jeu de plates-formes similaire à Mario. Dans un tel jeu, un personnage, contrôlé par le joueur au moyen du clavier évolue dans un monde en deux dimensions, vu de profil. Le but du jeu est de parcourir un niveau en évitant une suite d'obstacles en arrivant à la sortie

Table des matières

Introduction

Concevoir un jeu de plate-forme est très intéressant, cela permet de comprendre le mécanisme de rendu et le fonctionnement de ces derniers. C'était un projet ambitieux pour notre binôme car le JavaScript n'était pas vraiment un langage que nous maitrisions. Peu de temps après avoir commencé à travailler sur le projet, nous nous sommes rendu compte que nous étions un peu perdus, nous avons donc décidé d'effectuer quelques recherches et de suivre plusieurs tutoriels concernant la conception de jeu de plate-forme en JavaScript avant de nous relancer pleinement dans le projet. Suite à ça, nous étions beaucoup plus sereins et nous progressions plus rapidement même si quelques aspects du projet dont nous parlerons plus tard nous ont quelque peu ralentis !

Répartition du travail

Concernant la répartition du travail sur le projet, nous avons décidé de réaliser le projet ensemble. Comme nous étions tous les deux tout aussi novice l'un que l'autre concernant le JavaScript il était compliqué pour nous de partager des tâches ne sachant pas vraiment comment aborder le projet. Nous avons donc décidé de travailler ensemble sur des sessions de travail en commun afin que chacun de nous sache où en est l'avancée du projet. De ce fait chacun notre tour nous nous renseignons sur l'étape suivante à implémenter et faisions un récapitulatif avant chaque session de travail afin d'être efficace.

Implémentation

Notre map est organisé dans un canvas et les personnages (Joueur et ennemies) dans un autre pour éviter de recharger la map. Nous avons choisis d'utiliser 3 tilesets, un qui sera à l'origine du chargement de la map, un autre pour le joueur, et un dernier pour les ennemis.

Voici les différents Tilesets :

- Joueur :

- Map :

- Ennemies :

Les ennemis n'ont pas été implémentés avec l'animation, donc le tileset comporte une seule tile.

Documentation utilisé

- Un site expliquant les différentes techniques pour réaliser un jeu de plate-forme : http://higherorderfun.com/blog/2012/05/20/the-guide-to-implementing-2d-platformers/

- Un site contenant de graphiques réutilisables librement pour les blocs et les personnages : https://opengameart.org/

- Un tutoriel sur la creation d'un jeu de plateform : https://openclassrooms.com/courses/creer-un-mini-rpg-en-javascript-avec-canvas/mise-en-place-du-terrain

- Un article expliquant la création d'un jeu de plate-forme : (Celui-ci nous a essentiellement servi pour la collision/gravité) http://codeincomplete.com/posts/tiny-platformer/

- Un site permettant de tester la validité des fichiers JSON : https://jsonformatter.curiousconcept.com/

Code réutilisé et modifications

http://codeincomplete.com/posts/tiny-platformer/

→ Article de Jakes Gordon

- **Gestion des forces extérieures :**

```
player.ddx = 0;
player.ddy = GRAVITY;

if (player.left)
  player.ddx = player.ddx - ACCEL;     // player wants to go left
else if (wasleft)
  player.ddx = player.ddx + FRICTION;  // player was going left, but
not any more

if (player.right)
  player.ddx = player.ddx + ACCEL;     // player wants to go right
else if (wasright)
  player.ddx = player.ddx - FRICTION;  // player was going right,
but not any more

if (player.jump && !player.jumping && !falling) {
  player.ddy = player.ddy - JUMP;     // apply an instantaneous
(large) vertical impulse
  player.jumping = true;
}
```

- **Gestion de la collision :**

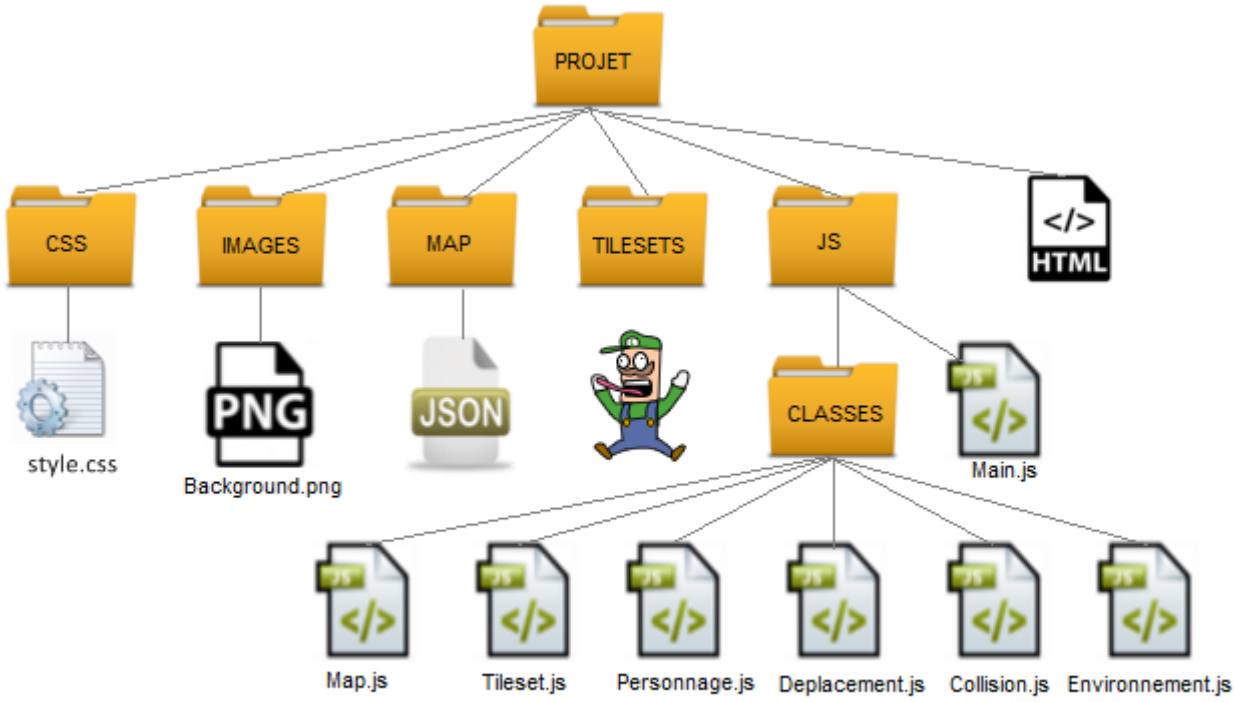
```
var tx        = p2t(player.x),
    ty        = p2t(player.y),
    nx        = player.x%TILE,         // true if player overlaps
right
    ny        = player.y%TILE,         // true if player overlaps
below
    cell      = tcell(tx,     ty),
    cellright = tcell(tx + 1, ty),
    celldown  = tcell(tx,     ty + 1),
    celldiag  = tcell(tx + 1, ty + 1);

if (player.dy > 0) {
  if ((celldown && !cell) ||
      (celldiag && !cellright && nx)) {
    player.y = t2p(ty);       // clamp the y position to avoid
falling into platform below
    player.dy = 0;            // stop downward velocity
    player.falling = false;   // no longer falling
    player.jumping = false;   // (or jumping)
    ny = 0;                   // - no longer overlaps the cells
below
  }
}
else if (player.dy < 0) {
  if ((cell      && !celldown) ||
      (cellright && !celldiag && nx)) {
    player.y = t2p(ty + 1);   // clamp the y position to avoid
jumping into platform above
    player.dy = 0;            // stop upward velocity
    cell      = celldown;     // player is no longer really in that
cell, we clamped them to the cell below
```

```
    cellright = celldiag;     // (ditto)
    ny        = 0;            // player no longer overlaps the cells
below
  }
}
if (player.dx > 0) {
  if ((cellright && !cell) ||
      (celldiag  && !celldown && ny)) {
    player.x = t2p(tx);       // clamp the x position to avoid
moving into the platform we just hit
    player.dx = 0;            // stop horizontal velocity
  }
}
else if (player.dx < 0) {
  if ((cell     && !cellright) ||
      (celldown && !celldiag && ny)) {
    player.x = t2p(tx + 1);  // clamp the x position to avoid moving
into the platform we just hit
    player.dx = 0;           // stop horizontal velocity
  }
}
```

Dans l'exemple de son article, il n'utilise pas de classe, nous avons implémenté son code dans notre classe collision et dans notre classe personnage, les variables qu'il utilise pour le personnage sont dans notre cas initialisé dans le constructeur de personnage. Le constructeur de notre classe collision contient uniquement les données de map extraite du JSON. La classe collision reprend la base de son code et nous utilisons cette base pour ajouter des implémentations supplémentaires comme les tiles létales par exemple.

Organisation du code

Voici la hiérarchie de notre code :



Nous avons commencé par implémenter l'affichage de la map, de ce fait nous avons mis en place deux objets :

- Un objet Map
- Un objet Tileset

Après avoir récupéré une image pour générer les tiles, nous nous somme attelé à mettre en place un découpage des tiles pour pouvoir les récupérer une à une. Elles sont bien entendu récupérées dans un ordre précis pour permettre de les reconnaitre plus tard. En ce qui concerne la Map, il contient le context 2D de notre canvas et nous découpons aussi ce canvas en une grille permettant de recevoir les tiles. Il fallait aussi une methode pour récupérer les données de la map qui se trouvent dans un format JSON.

C'est ici que nous avons rencontré le plus de difficulté ! Nous avions un problème dans le chargement de nos objets, comme le JavaScript est un langage asynchrone, les objets ne sont pas forcément chargés dans l'ordre dans lequel l'appel est effectué. Nous avons donc

commencé à utiliser des méthodes de callback pour essayer de résoudre ce soucis, ce qui à fonctionné en partit car notre map chargeait et se dessinait correctement excepté qu'une fois sur dix, le chargement plantait, un soucis d'asynchronisme subsistait donc toujours. Nous n'arrivions pas à corriger ce problème jusqu'à ce qu'on apprennent l'existence des promesses qui permettait de gérer ce genre de soucis. Une fois le mécanisme des promesses maitriser nous avons finalement corrigé tous nos problèmes et enfin avancer dans le projet !

Une fois notre map dessiné dans un premier canvas, il nous fallait implémenter le personnage. Pour éviter de recharger la map à chaque action de notre personnage, le personnage a été implémenté dans un 2ème canvas. Donc nous avons créé une nouvelle classe, la classe personnage. Cette classe permet de gerer toutes les caractéristiques de notre personnage et ces fonctionnalités, nous réutilisons notre classe Tileset pour charger notre personnage.

Notre personnage apparaissait donc sur notre canvas mais il était comme un stickers qu'on aurait placé là en décoration de map. Nous avons donc mis en place la gravité pour qu'il puisse avoir les pieds sur terre. Notre personnage était en mouvement ! mais uniquement un mouvement verticale bas et infini, il sortait de notre canvas en traversant le sol de notre map. Nous avions donc besoin de gérer la collision.
Nous avons donc mis en place une classe Collision qui permettrait de gérer la collision de notre personnage, pour cette classe, nous nous sommes aidé d'un article de Jakes Gordon (cf Bibliographie). Nous avons donc implémenté la collision avec l'aide du code de Jake Gordon pour qu'il fonctionne avec nos classes et nos variables. Une fois ceci fait, notre personnage s'arrêtait de tomber sur le sol de notre map. Bon et maintenant… Il faut qu'il puisse se déplacer notre personnage !

Vient à présent la classe déplacement, cette classe gère le déplacement de notre personnage grâce au gestionnaire d'événements, c'est un classe sans constructeur qui gère uniquement les « eventHandler » avec une fonction statique. A cet instant, on se dit que notre classe main dans laquelle on effectue tous nos appelles est un peu trop désordonné, on décide donc de créer une classe Environnement dans laquelle on effectue tous les appels, les instances d'objets et où on définit les fonctions utiles. Notre classe Main est donc

désencombrer et contient uniquement l'instance de notre Environnement et l'ajout des EventListener.

Le projet est bien avancé, on décide ensuite de gérer les tiles létales, on s'occupe de ça dans la classe colission ou on spécifie certaine tiles comme létales et d'autre qui blesse uniquement. On s'occupe donc aussi de la barre de vie du personnage, on affiche aussi texte suivant au-dessus de la barre pour bien que l'on comprenne que c'est une barre de vie : « Santé : this.vie % ». On spécifie aussi des tiles pour la victoire, pour permettre de finir le niveau autrement que par le décès du personnage. On s'occupe donc sur notre image de tiles d'ajouter deux tiles piège : la lave et les piques. Deux trois retouches avec Photoshop feront l'affaire. On ajoute aussi la fonction de course sur la touche Shift dans la classe Déplacement.

Il ne reste plus qu'une fonctionnalité avancée à ajouter, on choisit d'implémenter des ennemies mobile. Pour cela, on ajoute des attributs particuliers dans la classe personnage pour différencier une instance de personnage humain et une instance de personnage ennemie. On initialise l'ennemie avec un mouvement horizontale et avec une simple fonction on le fait se déplacer de gauche à droite sur une plate-forme. On ajoute ensuite les blessures occasionné par les ennemies à leurs contacts.

Il était difficile de gérer les animations avec l'image de tiles d'origine du personnage, on décide de retoucher le tileset et de gérer les animations de déplacement gauche et droite dans la classe Personnage lors du rendu, en fonction de l'action en cours on affiche une tile correspondante au mouvement.

Conclusion

Ce projet a été très intéressant, il nous a permis de progresser en JavaScript, mais aussi de comprendre les mécaniques utilisées derrière les jeux de plateforme. Dans ce que l'on pourrait améliorer dans notre projet, il y a le gestionnaire de niveau à ajouter et pourquoi éditer plus de tiles différente et d'ennemis afin de diversifier les niveaux. C'était un projet long pour des débutants comme nous, mais pour lequel nous avons beaucoup aimé travailler, chaque amélioration que l'on apportait était une réelle satisfaction.