Rapport projet WEB

CHINAL Guillaume – LÉVÊQUE Adrien

Introduction

Ce rapport s'accompagne de l'archive comprenant le code du mini-jeu *SuperFire*. Le jeu contenant des images pouvant faire l'objet de droits d'auteur, sa diffusion est restreinte au cadre de l'Université Paris Sud

Présentation du projet

SuperFire est un mini-jeu de plateforme dans lequel le joueur contrôle un personnage devant éviter toutes sortes d'obstacles afin d'atteindre le panneau de sortie, menant ainsi à la victoire.

La carte de jeu, stockée dans un fichier JSON, est décomposée en carrés (tiles) possédants chacun des propriétés. Certains objets sont létaux et pourront ainsi blesser voire tuer le personnage. Certains confèrent, en revanche, un bonus à celui-ci.

Le rendu est effectué sur une page web. Du fait de l'utilisation d'Ajax, un serveur Apache peut être nécessaire afin de permettre le bon fonctionnement du mini-jeu.

Rendu graphique

Structure HTML

La structure de la page HTML est relativement simple. Elle contient principalement les éléments suivants :

- canvas_container

Ce bloc est le conteneur principal et peut être assimilé à la caméra. Sa taille est dynamiquement calculée et s'adapte donc à la taille de l'écran du joueur, mais ne peut pas dépasser celle de la carte du jeu. La plus petite valeur des deux est donc attribuée (grâce aux propriétés CSS max-width et max-height). L'overflow des éléments contenus dans ce bloc est également caché pour des raisons qui seront présentées dans le suite de ce document.

- game_container

Ce premier canvas contient les éléments de la carte du jeu est n'est complété qu'au chargement de celui-ci. Il possède une taille dépendante du JSON présent dans /conf/map.json pouvant parfaitement excéder celle du conteneur principal.

- player_container

Ce second canvas possède la même taille que le précédent, mais ne contient que le joueur. Celui-ci est en revanche redessiné de nombreuses fois (environ 60 fois par seoncdes).

- visible_area

Bloc contenant les canvas et possédant la même taille que ceux-ci. Il permet de simuler les mouvements de caméra en se déplaçant.

- hit_div

Fond rouge ayant une légère transparence affichée lors d'une blessure.

- win_div / dead_div

Ecran de victoire / mort.

Chargement de la carte

Le chargement de la carte est effectué dynamiquement grâce aux deux fichiers de configuration présents dans le dossier /conf. Le fichier map.json contient la représentation de la carte du jeu tandis que le fichier sprites.json contient la définition des « tiles », du joueur et d'autres paramètres du jeu.

Ce chargement est une méthode de la classe Map permettant d'instancier sa variable membre *tiles*, un tableau contenant des instances de la classe *Tile*. La méthode draw, quant à elle, se contente d'appeler la fonction draw de chaque élément contenu dans sa variable *tiles*. Une promesse est renvoyée et ne sera résolue que lorsque que l'ensemble des images auront été chargées et affichées.

Chargement du joueur

L'étape de chargement du joueur est relativement similaire à celle de la carte du jeu. Deux images, indiquées dans le fichier de configuration, sont tout d'abord chargées, correspondants aux deux positions que peut prendre le personnage (droite et gauche). Celui-ci est ensuite dessiné sur son canvas.

Déplacement de la caméra

Le déplacement de la caméra se traduit par un déplacement de la div visible_area. Son comportement par défault est de centrer le personnage au milieu de ce bloc et s'en traduit par une modification des attributs left et top afin de faire overflow le contenu à masquer et de centrer le contenu à afficher. Lorsque les extrémités de la carte sont atteintes, la caméra ne défile pas.

Déroulement du jeu

Une fois les décors chargés, le jeu est démarré via la méthode *start* de la classe Game. Celle-ci contient trois fonctions anonymes exécutées à des intervalles de temps différents. L'une d'elle affiche le nombre de *fps* toutes les secondes. Les deux autres affiche le joueur et mettent à jour les coordonnées du joueur tous les soixantièmes de secondes.

Mouvements

Des handlers sont attachés aux flèches du clavier. Lors de l'appui sur l'une d'elle, la valeur présente à l'index, égal au code de la touche, du tableau *keys* de l'objet jeu est passé à vrai. A contrario, lorsque la touche est relâchée, la valeur passe à faux.

La fonction keysActions, appelée tous les soixantièmes de seconde, consulte les valeurs du tableau keys et appellera les fonctions correspondantes (exemple : jump si ArrowUp est à vrai).

Les déplacements sont traduits par une augmentation de la vitesse x et/ou y.

La fonction move, également appelée tous les soixantièmes de seconde, met à jour les coordonnées x et y du joueur en fonction de sa vitesse. Elle modifie également la valeur de la vitesse y en lui appliquant une force gravitationnelle. Enfin, elle appelle l'algorithme des collisions.

Collisions

L'algorithme des collisions fonctionne de manière assez simpliste. En fonction du mouvement, 3 extrémités du personnage sont choisies. Pour chacun de ces trois points, le bloc en contact est consulté. En l'occurrence :

- Si c'est un bloc spécial, sa fonction est exécutée ;
- Si c'est un bloc de victoire, la fonction victoire est appelée ;
- Si c'est un bloc létal, la fonction die ou hit est appelée ;
- Si le bloc est cassable, la fonction break est appelée ;
- Enfin, si le bloc est traversable, la fonction renvoi vrai, sinon elle renvoi faux.

Si une collision survient, la vitesse du personnage est annulée et sa position est modifiée pour coïncider avec l'extrémité du bloc touché (afin d'éviter que le personnage s'enfonce dedans).

Bonus implémentés

Barre de vie et blessures

Un nombre de vie par défaut est attribué au personnage (variable _lifes). Cette valeur est affichée dans une zone de texte située en bas à droite de l'écran de rendu. Lorsque le joueur heurte un bloc létal, si celui-ci ne *one-shot* pas, la méthode hit est appelée. Celle-ci affiche un fond rouge informant le joueur qu'il a été blessé. Il devient invincible pendant un court laps de temps (2 secondes) et le compteur de vie est décrémenté.

Défilement de l'arrière-plan

Le bloc visible_area contient un arrière-plan répété horizontalement et verticalement. Lors des mouvements de caméra, le déplacement du bloc donne l'illusion d'un défilement de l'arrière-plan. Cet arrière-plan est défini dans le fichier main.css.

Éléments de reliefs

Il est possible de configurer les blocs, dans le fichier sprites_config, de façon à leur attribuer le flag *special*. Lorsque celui-ci est à vrai, un nom de méthode doit être défini dans le champ *function*. La méthode de la classe Tile portant ce nom sera appelée lorsque le joueur heurtera de ce type. Il est ainsi facile de créer des bonus type vie, ou alors un objet type trampoline.

Vitesse de course

Lors d'un appui sur la touche espace, le coefficient d'accélération horizontal est multiplié par deux créant ainsi un effet de course.

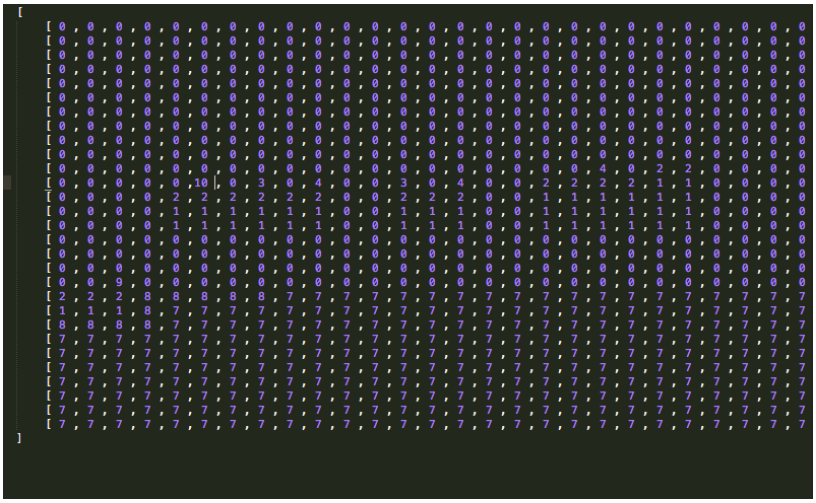
Fichiers de configuration

Sprites.json

- *Folder* : dossier contenant les images des tiles
- *Tiles* : configuration des tiles (le bloc 0 doit représenter l'air)
    - *Img* : image du tile (string)
    - *Lethal* : le bloc peut blesser (bool)
    - *Oneshot* : le bloc tue du premier coup – lethal only – (bool)
    - *Fixed* : le bloc ne disparaît pas au contact (bool)
    - *Crossable* : le bloc est traversable (bool)
    - *Win* : le bloc permet de remporter la partie (bool)
    - *Special* : toucher le bloc exécute une action particulière (bool)
    - *Function* : nom de la fonction à exécuter en cas de contact – specials only – (bool)
- *Width* : longueur d'un bloc
- *Height* : largeur d'un bloc
- *Player* : configuration du personnage
    - *Img_r* : image du personnage vers la droite
    - *Img_l* : image du personnage vers la gauche
    - *Width* : longueur du personnage
    - *Height* : largeur du personnage

Map.json

Ce fichier est un tableau à double dimension représentant la carte. L'inscription d'un chiffre conduira à l'affichage d'un tile ayant ce chiffre pour index dans le fichier précédemment défini. Exemple, pour l'air, l'index correspondant dans le fichier sprites est 0.



Répartition des tâches

La répartition des tâches fût relativement simple :

- Une personne s'occupant de la gestion des déplacements, collisions et bonus de course
- L'autre des points du projet.

Documentation et code réutilisé

Une très petite partie du code a été inspirée de discussions trouvées sur stack overflow :

- Le déplacement de la caméra a été inspiré du code disponible à l'adresse suivante : http://jsfiddle.net/yfqyq9a9/2/
- La résolution d'un tableau de promesse : http://stackoverflow.com/questions/4878887/how-do-you-work-with-an-array-of-jquery-deferreds
- .getJSON : http://api.jquery.com/jquery.getjson/
- .css : http://api.jquery.com/css/