Florentin PERRIER  
Victor BRETON

JavaScript – **Mario Snake**

**Compte rendu**

I – La game loop, Le contrôleur

La première chose que nous avons mis en place dans ce projet est le Contrôleur, permettant à tous les éléments de notre jeu de pouvoir communiquer ensemble.

Dans cet optique, nous avons commencé par créer une classe contrôleur permettant d’instancier le jeu : celui-ci initialisant le modèle et la vue lors de l’appel de la fonction init() : Voir « gController.js ».

L’élément le plus important de notre contrôleur est la boucle de jeu : Celle-ci permet de lancer la fonction de notre modèle calculant le « plateau » suivant, c’est-à-dire, tour suivant :

* Cette fonction est appelé update() dans notre modèle.

Cette boucle de jeu est invoquée par un intervalle de 50ms, qui lui-même incrémente l’horloge du jeu (tick -> gController). Tous les 8 « ticks », c’est le tour suivant.

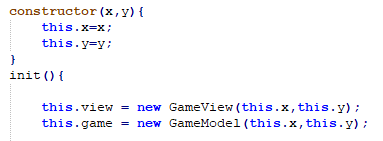
Ainsi, l’horloge du jeu est capable de se recaler, si le processus du navigateur devait ne pas répondre pendant un certain temps (de quelques millièmes de secondes).

La game loop est lancé par la fonction start() de notre contrôleur, et arrêté par la fonction stop() de celui-ci.

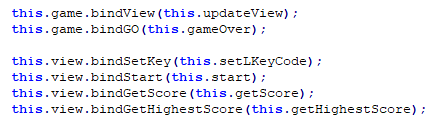
* La fonction start() est lancé par notre vue lors de l’appuie sur le bouton Play.
* La fonction stop() est lancé par notre modèle, lorsque le jeu est perdu.

Nous avons décidé, pour cette partie, de pouvoir générer le jeu en fonction d’un nombre de case x, y, à définir dans le constructeur de notre contrôleur. Ces valeurs sont également demandées par le constructeur de notre vue ainsi que celui de notre modèle.

* Ainsi, le modèle pourra générer la matrice avec une taille adéquate pour que notre vue puisse l’interpréter.



Nous avons également un certain nombre de fonctions bindé à notre vue et à notre modèle, pour permettre par exemple à notre modèle de rafraichir la vue avec la nouvelle matrice, etc…



II – Le modèle

# Initialisation :

Cette étape assez basique initialise notre matrice de jeu : Les murs à -1, le terrain à 0.

Elle initialise notre serpent, qui sera inséré dans notre matrice à chaque « tour », par appel de la fonction setMatrice() appelé elle-même par la fonction update().

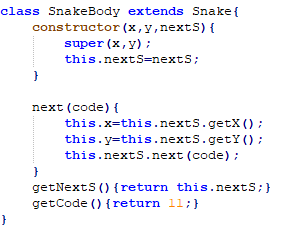
Elle génère un tableau de fruit.

# Le serpent :

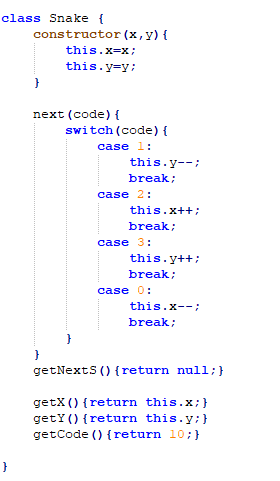
Nous avons choisi de créer deux classes pour la gestion du serpent. L’une correspondant à la tête et l’autre, classe fille de la tête, à chaque partie du corps du serpent.

Cette stratégie a été utilisé dans l’optique de faciliter les déplacements du serpent.

Chaque partie du corps du serpent possède un pointeur vers la partie suivante, jusqu’à l’avant dernière qui pointe vers la tête.



En appelant la fonction next() sur la queue du serpent, nous pouvons récupérer les coordonnées de la partie d’après et en appelant les méthodes getX() et getY() sur cette partie, nous poubons ainsi définir les nouveaux coordonnées X et Y de cette première. Nous appelons ensuite la fonction next() sur la partie suivante, et ainsi de suite.



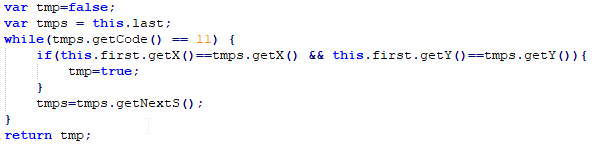
Quand nous appelons la fonction next() sur la tête, nous récupérons le code en arguments, permettant de définir vers où la tête doit se déplacer en fonction de la touche sur laquelle on a appuyé (code). Nous pouvons donc incrémenter ou décrémenter X ou Y en fonction de la valeur.

Maintenant que notre serpent est actualisé, nous pouvons vérifier s’il n’est pas sur un mur, s’il n’a pas mangé de fruit, ou son propre corps, et actualiser la matrice : Ces vérifications sont effectués via la fonction checkRules(), appelé par update() (=Loop).

Enfin, nous affichons la matrice si le jeu continu.

# Les vérifications :

Grace à notre serpent un peu particulier, les vérifications sont relativement simples à faire :



Par exemple : Pour vérifier si la tête de notre serpent ne mange pas une partie de son corps, nous stockons sa dernière partie (queue) dans une variable tmps, et tant que celle-ci renvoie le code 11 (11=> corps, 10=> tête), nous vérifions que la tête, stocké par notre modèle (this.first), n’a pas les mêmes coordonnées. Nous prenons ensuite la partie du corps suivante et ainsi de suite. Si nous trouvons des coordonnés identiques, nous renvoyons true, et c’est la fin du jeu.

Les fruits :

La génération de fruit, appelé à chaque fois qu’un fruit est mangé, génère un couple de coordonnées aléatoire, et tant que ces coordonnées ne correspondent pas à une case vide (0) de la matrice, régénère un couple de coordonnées, pour une fois sortie de la boucle générer le fruit.

La vérification se fait de façon basique avec une boucle for qui compare les coordonnées de la tête avec ceux de la liste de fruits.

Si un fruit est mangé, on vient rajouter un élément derrière le serpent, avec comme élément suivant, l’ancien dernier élément du serpent.

# Le score :

Le score maximal est stocké dans un cookie. Quelques réglages en lien avec la sécurité des navigateurs, font que notre application, stocké sur un disque local, ne permet pas aux navigateurs de mémoriser le cookie correctement.