LEGER
Romain

LP CDED

Nouveaux modes de développement
Rapport n°3 du 30/09/2015

1. Installation de JSHint (10 minutes)

En début de cours, il nous a été conseillé d'installer JSHint pour optimiser le code. J'ai donc effectué cette installation. Cependant, je ne l'ai pas utilisé de suite, voulant déjà finir mon travail en cours, et sachant que mon code n'est pas définitif à ce moment de l'implémentation du jeu.

2. Web Socket et Connexion (2 heures)

Dès le départ, j'ai continué sur la création des Web Sockets, sujet déjà commencé la dernière fois. Je m'étais arrêté à l'envoi simple de message entre le serveur et le client.

Maintenant, le but était d'avoir plus d'options, et plus de communication entre le serveur et le client. Pour cela, j'ai décidé de créer un tableau de client, pour différencier les connexions au serveur. Il se remplit quand on se connecte, et supprime le client s'il se déconnecte. De plus, j'ai pensé à créer un snake lorsqu'un utilisateur se connecte, ce qui permet à chacun des utilisateurs d'avoir son propre serpent.

De plus, j'ai modifié la fonction *onFrame()* donnée par *Paper.js*, pour créer une fonction qui permet d'être appelée par le serveur. C'est donc le serveur qui s'occupe de lancer cette fonction, avec notamment la fonction *setTimeOut*, qui permet de faire une exécution toutes les *n* millisecondes. A partir de ce moment, le jeu et le temps entre les déplacements étaient gérés par le serveur. Maintenant, il fallait mettre en place une solution pour que chaque utilisateur qui se connecte gère son propre snake (déjà implémenté), mais surtout qu'il voit les autres joueurs.

3. Compréhension des rôles et modifications client (1 heure 30)

Le serveur et le client communiquant, il fallait dorénavant savoir quel était le rôle de chacun. Ceci est une notion qui à l'air simple, mais qui est difficile à mettre en place. En effet, du côté serveur, il faut prendre en charge les données de tous les joueurs, donc de tous les serpents sur l'espace de jeu. Donc du côté client, il ne faudra plus gérer seulement un serpent, mais le serpent de l'utilisateur courant, et le déplacement des autres serpents. J'avais mis en place des variables globales correspondant au serpent de l'utilisateur. Il fallait donc revoir toute cette partie là, pour d'abord s'occuper correctement d'un seul serpent. J'ai donc modifié toutes les fonctions et l'architecture du client pour mettre en place un système qui gère un seul serpent, avec des variables locales passées en paramètre. Cela m'a pris autant de temps à tout repenser dans le client, et à modifier les fonctions, sans passer par des variables globales.

4. Rôle serveur et fonctions associées (2 heures)

Dans cette partie j'ai décidé de faire tous les calculs coté serveur. En effet, maintenant, toutes les fonctions pour un seul serpent sont du coté client. Mais la modification doit se faire du coté serveur, et pour tous les serpents. Il fallait donc à nouveau modifier le code coté client, pour que les fonctions de base appellent les fonctions de calcul. Le rôle de client est maintenant juste d'afficher l'état de tous les serpents de la partie, et gérer le clic souris, qui enverra un message au serveur. Le coté client se vide donc au profit du coté serveur.

Le serveur sera maintenant le gestionnaire de tous les serpents, donc de l'"espace de jeu". C'est aussi lui qui fera les calculs pour chacun des serpents, et, tous les 20 millisecondes, enverra la

nouvelle version des serpents au client pour l'affichage. Toutes les fonctions de calcul de déplacement se trouvent dans le fichier serveur, en prenant appui sur les précédentes fonctions client. Il faut juste gérer plusieurs serpents.

Du coté client, j'ai donc laissé pour l'instant deux fonctions importantes : La création du serpent (et ses fonctions associées), et tout cela va être transmis au serveur pour qu'il fasse la mise à jour. La difficulté était le type des données car on ne peut pas envoyer directement un objet. Il faut passer par JSON, qui permet de transformer un tableau en une chaîne de caractère. Je rajoute devant le mot "creation" pour indiquer au serveur, lors de la réception, qu'il doit créer un serpent. La deuxième fonction est le clic souris, le client transmet donc le point cliqué (en JSON), avec le mot-clé "clic".

Du coté serveur, il faut maintenant gérer la réception des messages. Lorsque le premier mot clé est "creation", alors il retire cette partie, retransforme en objet, et l'ajoute en tant que nouveau serpent dans le jeu. Pour le deuxième cas, il retire le mot-clé "clic", créé un vecteur entre le serpent courant et le point cliqué, et modifie la nouvelle direction du serpent. Problème : le serveur ne connaît pas les fonctions "Paper.js"

5. Windows, Cairo et dépendances (20 minutes)

Il faut donc inclure Paper.js au fichier de serveur. Mais on ne peut pas directement l'inclure dans le fichier. Il aurait fallu installer Cairo pour le permettre. Le problème est qu'il est impossible de télécharger Cairo pour Windows. J'ai donc testé des solutions disponibles sur le Web, mais aucune ne fonctionne vraiment. La solution serait de réécrire les fonctions, vector.normalized() étant assez simple à recréer. Mais ceci est une solution temporaire, peut-être qu'une solution pour installer Cairo sous Windows existe.

Rapidement, j'ai testé le site https://github.com/Automattic/node-canvas/wiki/Installation---Windows, en ayant tout installé, cela ne fonctionne toujours pas.

6. Prochaine étape

La prochaine tâche à réaliser est de réécrire la fonction normalized(), puis modifier les fonctions d'affichage sur le fichier client, et enfin d'appeler ces fonctions coté serveur. Cela ne devrait pas prendre de temps (sauf erreur inattendue), la matinée prochaine. Je pourrais aussi, si je suis en avance, réessayer d'installer Cairo. Enfin, je commencerais à réaliser quelques améliorations dans l'après midi, après avoir optimisé mon code avec JSHint.