LEGER
Romain

LP CDED

Nouveaux modes de développement
Rapport n°4 du 07/10/2015

1. *Cairo*, *Paper* et dépendances (1 heure)

Tout d'abord, j'ai voulu essayer de régler le problème de l'installation de *Paper* grâce à NPM. J'ai réussi après de nombreuses tentatives infructueuses l'installation de *Canvas* sur mon ordinateur portable Windows 10. Après cette étape, j'ai essayé d'installer *Paper*, ce que j'ai réussi, mais uniquement en global. Malheureusement, je n'ai toujours pas réussi à l'utiliser, car *Node.js* n'arrive pas à le reconnaître, même en global. Dans un dernier essai, j'ai copié-collé le dossier global dans le dossier local du projet, mais des erreurs sont encore survenues. Il fallait dorénavant penser autrement, car du temps était perdu pour avancer dans le projet. Une alternative était obligatoire.

2. Premières modifications côté client (1 heure)

Le script coté client devait être largement revu. En effet, c'était lui qui gérait totalement l'animation (hormis les messages envoyés vers le serveur), et donc il fallait revoir tout cela. De plus, lors de la connexion, les serpents étaient créés directement sous le format de *Paper*. Mais, lors de l'envoi au serveur, il était très difficile de les modifier, car le format avait été modifié par JSON, et non réutilisable en *Paper*.js sous le serveur, mais en plus, *Paper*.js n'étant pas installé sur le serveur, la tâche devenait impossible. Alors, au lieu de créer les disques (représentant le corps du serpent) avec cette bibliothèque, j'envoie seulement au serveur une coordonnée x, y et une couleur pour chaque partie du corps du snake. J'ai donc créé une nouvelle fonction de création du serpent. J'ai gardé l'ancienne, car elle nous sera utile. De plus, j'ai modifié la fonction *onMouseDown*, car elle renvoyait un point, qui est aussi utilisable par *Paper* uniquement. Alors j'extrais ses paramètres importants (x et y) par ses accesseurs, puis j'envoie seulement une variable contenant ces informations. Maintenant que toutes les variables envoyées au serveur ne nécessitent plus *Paper*, il faut reprendre le coté serveur.

3. Modification du serveur (1 heure)

Le serveur reçoit donc une information exploitable. J'ai introduit une nouvelle variable (globale), qui va stocker tous les snakes, créés par la fonction du client, donc en format (x,y,color). Pour chaque connexion, un serpent sera ajouté ici.

Les fonctions de calculs ont été entièrement revues, car elle dépendaient de *Paper*, et les formats n'étaient plus corrects. Donc j'ai modifié la fonction de changement de direction (grâce au clic), et celle de déplacement automatique des serpents dans la bonne direction. Les calculs sont plus simples grâce au nouveau format.

Ensuite, il fallait gérer les différentes méthodes d'envoi de données au client. Pour cela, une méthode de création de snake intervient après le message de création par le client. Ceci enverra à tous les clients le nouveau snake créé, et le nouveau client aura en plus tous les snakes avec leur état présent (leur position actuelle). Puis, il reste la fonction de mise à jour, qui doit intervenir périodiquement (tous les 20 ms pour mon cas), et envoyer l'état de tous les snakes à tous les clients, pour que chacun mette à jour sa vue. Enfin, lorsqu'une déconnexion d'un client intervient, il est supprimé de la liste des clients, son snake est aussi supprimé du tableau (mis à null sans déplacement des autres), et un message est envoyé aux clients pour supprimer l'affichage du snake dans leur vue, et le supprimer de leur propre tableau de serpents.

4. 2e partie de la modification du client (1 heure)

Maintenant que le serveur et les clients peuvent s'envoyer des messages, et que le contenu des messages est utilisable par les deux parties, il faut gérer la vue utilisateur, et donc cette fois-ci, les fonctions de *Paper*. D'abord, il faut réceptionner les messages du serveur. Les 3 messages vus précédemment peuvent être reçus.

Lors de la réception de la connexion par le client, le nouveau snake est créé avec *Paper*, et son contenu ajouté au tableau de tous les serpents. Pour la mise à jour, les propriétés des cercles créés sous *Paper* sont mis à jour par les coordonnées reçues par le serveur. Enfin, pour la déconnexion, les cercles sont détruits de la vue, et le snake est supprimé du tableau total des serpents.

Dorénavant, plusieurs connexions sont possibles, le clic modifiant bien le serpent associé au client, les déconnexions étant gérées, et les serpents ajoutés ou supprimés de la vue. Hormis les problèmes de performances en cas de connexions multiples, le jeu est jouable.

5. JSLint avec Brackets (1 heure 30 minutes)

Tout fonctionnant correctement, il faut maintenant se conformer avec les normes du JavaScript. J'ai choisi JSLint, car il est directement donné par l'éditeur Brackets (JSHint étant aussi installé), et donc nous pouvons voir les modifications en temps réel. Cela m'a pris du temps, car ma façon de coder ne correspondait pas du tout à leur façon (notamment pour les for, les if, et l'alignement des crochets). Donc une grande majorité du code était à revoir, dans les deux fichiers serveur et client. De plus, certaines erreurs n'étaient pas explicites, donc il fallait chercher sur la documentation de JSLint pour savoir comment résoudre ce problème. A ce moment, tout le code est conforme, je n'ai plus aucun warning sur les deux fichiers.

6. Gestion des collisions (30 minutes)

Avant de terminer la séance, j'ai eu le temps de commencer la gestion des collisions. Pour l'instant, un message sur le serveur est affiché lors de la collision de 2 snakes ou plus, avec le numéro des clients qui ont fait la collision. Pour cela, je prends les serpents 1 à 1, puis leur disque (corps) 1 à 1, et je les compare aux disques des autres serpents (excepté ceux déjà testés). Par exemple, si nous avons 3 serpents, tous les disques du serpent 1 sont comparés à ceux du serpent 2, puis au serpent 3. Ensuite, tous les disques du serpent 2 sont comparés au serpent 3, et cela s'arrête là.

7. Prochaine étape

La semaine prochaine, je vais gérer les événements de la collisions. J'ai plusieurs idées pour cela, soit un blocage temporaire du serpent qui a touché l'autre (ceci pourrait amener des problèmes, je vais y réfléchir) soit une remise en jeu au point de départ du serpent qui a effectué la collision, soit un système de point, ou le serpent qui touche aura une perte de point, et une fin de jeu est envisageable en cas de points négatifs trop importants (ou mise à 0 des points).

De plus, je vais réfléchir aux différentes améliorations possibles. Un système de musique de fond est déjà implémenté (non effective actuellement), et je vais rechercher d'autres idées à mettre en place pour la prochaine fois.

Enfin, je vais créer des tests unitaires et globaux pour tester les fonctions clients et serveurs. Je ne sais pas dans quel ordre j'effectuerai ces deux dernières tâches, cela dépendra du nombre d'idées et de la complexité de celles-ci que j'aurais lors de la prochaine séance.