**Rapport final.**

Je n’ai malheureusement pas réussi à terminer le projet de node.js, à savoir faire fonctionner correctement le jeu en mode multijoueur.

Je n’ai pas eu beaucoup de temps pour y travailler depuis mercredi étant donné que nous avions un projet de gestion à rendre pour vendredi, ainsi qu’un autre projet d’UML a finir ce week-end pour la démonstration de lundi matin.

J’ai cependant réussi à obtenir quelque chose de mieux qu’à la démonstration de mercredi.

Voici comment fonctionne mon code, c’est-à-dire comment communiquent les clients et le serveur de jeu :

Le serveur possède son propre tableau contenant l’ensemble des snakes de tous les joueurs connectés. Il a également un tableau contenant toutes les sockets des clients connectés.

Les informations servant à modifier les positions sont stockés dans des tableaux côté serveur :

-dxs et dys contenant l’ensemble des dx et dy de chaque client,

- norms contient la norme calculée à partir des dx et dy pour chaque client.

Evidemment, toutes les informations du client i sont stockées aux indices i dans tous les tableaux du côté du serveur.

Lorsque le client charge la page (localhost :8000 étant donné que le jeu ne marche qu’en local), le serveur websocket envoie la commande ‘init’ au client.

Le client reçoit la commande init, appelant la fonction init(). Cette fonction va créer un cercle dont les coordonnées sont générées aléatoirement, ainsi que 9 copies de ce cercle.

Ces 10 cercles sont stockés dans le tableau « snake » du coté du client. Chaque client possède donc son propre tableau contenant uniquement son snake.

Ensuite, de son côté, le serveur possède une boucle infinie qui, toutes les 50 ms, broadcaste à tous les clients connectés le tableau contenant les positions de tous les cercles de chaque snake sous forme de string. Le string envoyé contient donc un ensemble de positions, séparées par des /. Chaque serpent est délimité par un |.

Avant d’envoyer le message, le serveur va tout d’abord, pour chaque serpent stocké dans son tableau de snakes, modifier les positions des cercles de chaque joueur en fonction de son vecteur (décomposé en dx et dy) via la fonction calculer.

Lors du broadcast, le serveur envoie la commande ‘upda’ pour update, ainsi que l’id du client, suivi du message.

Lorsque le client reçoit la commande update, il extrait les 5 premiers caractères du message et appelle la fonction update() en lui passant en paramètre le string contenant les positions, ainsi que son id.

La méthode update va ensuite effacer tous les cercles présents à l’écran et extraire le tableau de tous les snakes en utilisant string.split(« | « ) , puis chaque position en utilisant string.split(« / ») .

Il affiche ensuite tous les cercles en créant un point en utilisant les positions 2 à 2 (x et y), puis créée un cercle de radius 20 et de couleur rouge à partir de ce point.

L’id permet de savoir à quelle itération le client doit, en plus d’afficher les cercles, modifier les positions de ses propres cercles (stockés dans le tableau « snake) en utilisant les coordonnées des cercles.

A la fin de la fonction update, le client envoie au serveur l’ensemble du contenu de son tableau « snakes », à nouveau sous forme de string.

Lorsque le serveur reçoit le message, il va enregistrer les nouvelles positions des cercles du client son tableau des snakes en utilisant l’index de la socket dont il a reçu le message.

Dans le cas où le client bouge sa souris, via la fonction paperjs onMouseMove, le client envoie un message différent. Ce message contient les positions de ses 10 cercles, plus la position x et y du point qui a été cliqué à l’écran, que l’on récupère via event.point . Ces coordonnées permettent de créer le nouveau vecteur coté serveur pour ce client (décomposé en dx et dy).

Ainsi, lorsque le client ne bouge pas la souris, le serveur calcule continuellement les nouvelles positions et les envoie au client. S’il bouge la souris, changeant la position où le snake doit se diriger, le vecteur sera instantanément mis à jour côté serveur, permettant au prochain calcul de diriger le snake dans la nouvelle direction.

Actuellement, au niveau du rendu graphique en mode multijoueur, chaque joueur possède son propre snake, représenté par 10 cercles superposés. Etant donné que tous les cercles sont de couleur rouge, il n’est pas possible de les différencier par leur couleur.

Vu que je n’ai pas pu terminer le mode multijoueur, les 10 cercles superposés ne se déplacent que très légèrement lorsque l’utilisateur déplace la souris.

En revanche, chaque joueur « contrôle » effectivement son propre « snake ».

Pour résumer, je n’ai pas réussi à terminer le déplacement des snakes, ni l’affichage graphique complet du snake lors des déplacements. Je n’ai pas non plus eu le temps de gérer les collisions entre snake, étant donné que le reste n’était pas terminé.

Remarques sur le code :

- La fonction calculer() permet aussi de vérifier si un cercle ne touche pas un des bord de la zone de jeu. Si c’est le cas, le cercle sera envoyé à l’opposé (selon l’axe des x ou des y) de la zone de jeu.

- J’ai choisi d’envoyer les messages toutes les 50 ms de manière arbitraire. Il est possible de changer cet interval, sans pour autant changer la vitesse de déplacement du snake. En effet, la vitesse du snake dépend de la valeur par laquelle on divise dx et dy dans la fonction calculer. Actuellement, je divise dx et dy par norm / 5, norm étant la norme du vecteur. Si veut passer à 25ms par exemple, pour doubler les fps du joueur, il suffit de diviser norm par 10 au lieu de 5 afin que les snakes se déplacent toujours à la même vitesse.

- Au début, si le client ne déplace pas sa souris après s’être connecté, il se déplace dans une direction générée aléatoirement à l’aide de la position du premier cercle et d’un autre point également généré aléatoirement.

- Les 10 cercles de chaque joueur sont actuellement superposés, car je souhaitais tout d’abord faire fonctionner le déplacement d’un seul cercle en mode multijoueur avant de m’occuper du rendu graphique, en ayant 10 cercles qui se suivent les uns les autres. Il aurait fallu modifier la position de départ des 9 copies dans la fonction init, en les décalant dans le sens inverse du premier vecteur.

- Le code paper permettant de déplacer le serpent pour le mode monojoueur est contenu dans le fichier indexsolo.html, situé au même endroit que le fichier index.html. De même, le code du serveur qui contrôle cette animation est dans le fichier serverwssolo.js. Il suffit de renommer l’indexsolo.html en index.html et de lancer le serverwssolo au lieu du serverws pour afficher la page.

- Le code a été commenté de façon à correspondre à mes explications