Mise en place du projet GitHub

Installation de node.js et des modules nécéssaires (express...)

Mise en place d'un serveur simple via node.js

Réorganisation du dossier GitHub, création de dossiers public/private

Passage du site en https, génération des clés et certificats, installation des modules fs, https et openssl à l'aide de cygwin et console.

_____ TP2 _____

Exclusion du dossier private (clés/certificats) de GitHub pour raisons de sécurité, tâche assez rapide.

Installation de paper.js dans le projet, consultation de tutoriels sur le site officiel de paper.js dans le but d'en comprendre le fonctionnement.

Correction d'un bug dans le code du serveur réussie, on obtient désormais une page avec un canvas paper.js.

Intégration de fonctions de "dessin" paper.js dans index.html.

Test non fructueux, recherche de solutions. Adaptation de l'élément canvas à toute la page pendant ce temps.

Dessin opérationnel, début du travail d'animation de déplacement vectoriel. Expérimentation de features mineures (changement de couleur en temps réel avec onFrame)

Déplacement fonctionnel, une partie assez ardue mais il est désormais possible de déplacer le snake à vitesse constante via clics de souris comme attendu. Le snake est composé de nombreux cercles copiés du cercle originel qui sont rafraîchis régulièrement.

Cleanup de fonctions obsolètes commentées pour éclaircir le code. Modifications du CSS pour définir un cadre autour du canvas.

Début de la mise en place du server WebSocket.

_____ TP3 _____

Cleanup rapide du code du fichier server.js via JSLint.

Revert de quelques modifications apportées (on déplacera le snake à l'aide de clics de souris et non pas via la position temps réel de la souris pour un souci de simplicité dans les échanges client/serveur)

Suppression du dossier private sur GitHub (il avait été exclu des commits lors du TP précédent

mais pas retiré)

Consultation de divers tutoriels WebSockets, questions sur StackOverflow... pour mieux comprendre son potentiel fonctionnement dans le projet.

Mise en place de la possibilité de communiquer entre le client et le serveur via WebSocket, réflexion sur l'implémentation de l'animation avec cette nouvelle méthode.

Ajouts de quelques petites fonctions pour une meilleure intégration multi-joueur (différenciation des différents snakes avec couleur aléatoire, départ à un point aléatoire du canvas), des difficultés à trouver la manière pour réussir à implanter correctement le mouvement de plusieurs joueurs côté serveur.

Ajout de nombreux commentaires dans le but d'éclaircir le code. Restructuration majeure des fonctions, mise en place d'un "échaufaudage" pour le code client/serveur encore non fonctionnel.

On peut désormais passer les données de position des serpents et sa direction entre les clients et le serveur, grâce à un broadcast timé via setInterval, qui envoie à tous les clients un tableau avec les données de chaque serpent.Un des problêmes que j'ai rencontré était le fait de ne pas pouvoir envoyer de tableau entre le client et le serveur, mais grâce aux fonctions JSON.stringify et JSON.parse, j'ai pu résoudre cette issue. Reste à résoudre le problême de l'affichage de ceux-ci car les objets, transformés en JSON et reformés semblent ne pas pouvoir être clonés, mais globalement le problême semble être facilement gérable avec plus de recherches. Cette partie s'est avérée difficile à matérialiser dans l'idée mais une fois le principe compris, la réalisation en trial&error était assez triviale bien que assez longue pour obtenir un fonctionnement sans problêmes.

TP4

Ccodage d'une fonction de déconnexion : Réalisation imparfaite mais empêchera le plantage dans la plupart des cas pour les tests.

Passage rapide sous JSLint pour corriger quelques imperfections, ajout de commentaires et amélioration générale du code.

Il semblerait finalement impossible de passer les éléments Paper.js depuis le client vers le serveur et inversement dans la mesure ou le passage en JSON empêche la reconstitution correcte des objets. Après une heure de recherche environ, on essaiera donc de trouver une autre solution. On notera que parmi les difficultés du projets, on peut peiner à trouver des solutions utiles en recherchant sur Internet car relativement peu de développeurs semblent utiliser paper.js et le problême est encore plus important pour trouver des solutions à des problêmes de modules tels WS.

On tente de donner un ID au client pour qu'il puisse identifier son snake, mais la réception et l'envoi de divers types de messages entre le client et le serveur provoque trop de potentiels problêmes pour continuer sur cette voie.

On essaie de passer des données simples au serveur : coordonnées x - y, couleur du snake et longueur. Ainsi, plus de problêmes de type et la majorité de l'interprétation graphique sera réalisée côté client.

Révision de toute l'algorithmique, on couche sur papier les différentes manières pour potentiellement procéder et on va poser des questions rhétoriques pour arriver à un résultat logique. En effet, arrivé à ce point dans le projet, il m'est très difficile de trouver un moyen d'avancer. En attendant de trouver un fonctionnement adéquat, quelques améliorations diverses pour respecter des normes JSLint, puis début de la séparation des fichiers pour rendre le code plus lisible, agréable et modulable.

TP4

Suite à des problêmes sur mon ordinateur personnel, j'ai dû formater et réinstaller mon système d'exploitation. Ainsi, le début du TP fût assez laborieux car malgré la conservation des modules, mise en place du git et installation de node.js, une erreur EMPTY_RESPONSE apparaît au lancement du serveur, et mes recherches liées à cette erreur s'avèrent infructueuses.

Dans l'impossibilité de tester mon code pour le moment, je réalise quelques améliorations dans le formatage de mon code.

Une fois le problême résolu, début de la planification du code pour avoir un plan plus précis : Mise en place d'un modèle se rapprochant plus d'une orientation objet, et réalisation d'un diagramme de séquence pour mettre au clair le fonctionnement de notre application.

On a donc un nouveau fichier "snake.js" qui représentera le modèle qu'on utilisera côté serveur, et potentiellement côté client si faire se peut. Pour s'assurer du bon fonctionnement de ce modèle, on réalise également un fichier de tests pour réduire la probabilité d'avoir des erreurs plus tard.

Fin de séance, par manque de temps pour finir le modèle, je nettoie le client/serveur de leurs méthodes et attributs obsolètes selon la nouvelle implémentation souhaitée.

Je prévois de tenter d'avancer sur ce projet pendant des heures libres pour rattraper le retard pris en début de journée.

TP5

L'objectif de la journée est de pouvoir terminer le modèle et par conséquent de pouvoir gérer un mode multijoueurs plus ou moins fonctionnel.

J'ai donc commencé par avancer dans l'élaboration du modèle, élaboration accompagnée de divers tests pour vérifier la bonne fonctionnalité des fonctions intégrées.

Ensuite, j'ai revu mon système d'exports pour pouvoir inclure le même fichier de modèle dans le serveur et le client sans conflit.

Une fois ces tâches réalisées, je commence à implémenter les échanges client/serveur pour pouvoir tester le fonctionnement concret de mon code et pouvoir donc débugger de manière plus adéquate.

Après un remaniement du système d'exports vu que les changements précédents étaient plus problématiques que d'autres alternatives plus simples, on s'attaque donc au multijoueur. En parrallèle, on met en place un nouveau système d'affichage des cercles du snake adapté à plusieurs utilisateurs parralèles.

On peut désormais afficher les cercles des divers joueurs, reste à supprimer les cercles obsolètes ne faisant plus partie du snake. J'ai essayé diverses méthodes, mais la solution pour obtenir le corps du snake suivant correctement la tête m'échappe.

Finalement, après des tests et une réflexion sur le code, l'animation gérée par le serveur est fonctionelle. Reste à gérer les collisions entre joueurs et murs !

Dernier TP

Avant ce TP, pendant la semaine de vacances, j'ai pu avancer par moi même sur l'application et ai réalisé la partie permettant de gérer les collisions, premièrement avec le bord du terrain puis entre les différents joueurs. La base du jeu était désormais fonctionelle.

J'avais donc pour but dans cette séance d'attaquer le dernier problême majeur de mon application : La déconnexion. En effet, j'avais d'abord utilisé une sorte de “détour” permettant de transférer le contrôle d'un snake déconnecté au dernier joueur arrivé sur le serveur. Il n'était pas imaginable d'utiliser une telle méthode dans le cadre d'une application complète, de ce fait j'ai tenté de transmettre divers signaux aux autres joueurs pour signaler les connexions et déconnexions. Le système tel quel fonctionnait globalement, mais posait problême quand un client arrivait sur un serveur sans joueurs actifs, mais ayant accueili des joueurs précédemment. Finalement, j'ai pu simplifier le code et résoudre le problême en proposant au client de ne pas afficher quoi que ce soit vis à vis des clients dont le snake était == null, méthode bien plus logique et fonctionelle. Le snake est donc actuellement assez fonctionnel, je ne trouve pas de

problême majeur (crash). Resterait potentiellement à améliorer le gameplay en ajoutant les pastilles permettant aux snakes de grandir, plutôt que d'utiliser un classement à base du nombre de morts, avec un peu de temps supplémentaire.