LEGER
Romain

LP CDED

Nouveaux modes de développement
Rapport n°6 du 21/10/2015

1. Modules, tests unitaires et documentation (2 heures)

Le but de cette séance était de programmer les tests unitaires par rapport aux fonctions créées tout au long de ce module. J'ai mis du temps à en comprendre le mécanisme. En effet, je pensais qu'il était possible de tester toutes les fonctions présentes dans le projet actuellement. Mais on ne pouvait que tester les fonctions qui sont côté serveur, car pour validr ces dernières, nous avons besoin de Node.js. Or, côté client, nous ne sommes pas directement sur Node.js, et certaines fonctionnalités telles que Paper ne peuvent pas être implémentés sur Node.js. Donc j'ai décidé de faire un fichier de test sur le côté serveur.

Avant de commencer à faire des fonctions de tests, j'ai pensé à restructurer le code. Effectivement, toutes les fonctions étaient concaténées dans le fichier server.js, et donc plein d'instructions à la suite étaient présentes, sans lien particulier. Alors j'ai laissé le code obligatoire dans le fichier server.js, puis j'ai déplacé toutes les fonctions dans un autre fichier, appelé gameState.js. Alors nous pouvions créer un module de ce fichier, ce qui signifie qu'on pourra exporter ce fichier dans d'autres. Le module créé, j'ai ajouté l'inclusion de ce fichier dans server.js, et dans test_Server.js, le futur fichier de test des fonctions.

Enfin, je n'avais pas encore fait la Javadoc, et certaines parties n'étaient pas commentées. Alors j'ai créé cette Javadoc, en expliquant ce que la fonction devait faire, et agrémentée de commentaires dans cette fonction pour une meilleur lisibilité de code. Tout était dorénavant prêt pour créer les tests unitaires.

2. Tests unitaires serveur (2 heures)

Le nouveau fichier test_Server.js contiendra donc toutes les fonctions de tests du fichier gameState.js. Il fallait donc créer, pour chacune des fonctions de gameState.js, une autre fonction qui la testera. Pour plus de compréhension du fichier, toutes les fonctions de gameState ont une correspondance dans le nouveau fichier avec le nom "test_", plus le nom de la fonction testée.

Pour les tests, j'ai créé des variables semblables à l'environnement du serveur lors d'une partie de snake. Le fichier contient donc 2 snakes. 2 car ils seront surtout utiles pour les collisions. Puis, j'essaie de prédire, pour chaque fonction, son résultat, et je teste avec le vrai résultat. Je n'ai pas eu de nombreux problèmes pour cette partie, il fallait simplement bien regarder ce qui avait changé, et bien prédire le bon résultat dès le départ. J'ai eu une seule difficulté, en effet, les valeurs des variables sont stockées grâce à des pointeurs, et donc une affectation d'une variable à l'autre permet un pointage d'une zone mémoire identique. Au début, je n'avais pas compris cela, et donc tous mes résultats étaient faussés. Mais après avoir vu ce mécanisme, j'ai réussi à créer des fonctions de tests pour toutes les fonctions, hormis pour réinitialisation, car elle nous donne des résultats volontairement aléatoires, et donc nous ne pouvons tester ces résultats.

3. Obstacles (1 heure)

Je voulais maintenant améliorer la difficulté du jeu, car pour l'instant, le seul danger pour 1 joueur est les autres joueurs, car j'ai décidé que les bords ne soient pas pénalisant pour le snake, il réapparaît à l'opposé du jeu (en x ou en y). Donc il fallait un nouvel outil pour que le jeu ne soit pas dépendant que des autres joueurs. Alors j'ai eu l'idée de créer des obstacles, qui apparaissent aléatoirement dans le jeu, au hasard. Le principe est une nouvelle fois basée sur l'aléatoire. Toutes les frames (20 ms), un nombre est tiré aléatoirement, ce qui peut déclencher deux événements :

l'apparition d'un obstacle, ou la suppression d'un obstacle. Si le 1er événement est appelé, alors un obstacle de taille et de position aléatoire est créé sur le plateau. Dans le cas du second, le 1er obstacle apparu chronologiquement est supprimé. Puis, j'ai modifié les vues client pour afficher ces obstacles à l'écran.

4. Mort et réinitialisation (1 heure)

Un problème venait s'ajouter à cette création d'obstacle. En effet, le joueur, lorsqu'il perdait une vie, était réinitialisé aléatoirement sur le jeu. Mais parfois, il pouvait donc tomber directement sur un obstacle ou un autre joueur, et perdait directement une vie. Donc le joueur pouvait perdre un grand nombre de vie d'un coup, et cela juste en touchant 1 fois un obstacle ou un autre joueur. Donc j'ai créé une variable noKill, qui est utile lors de la perte d'une vie. En effet, dans ce cas, la valeur de cette variable est de 300. Celle-ci diminuera toutes les frames (20 ms), et, tant qu'elle n'est pas à 0, rien ne peut lui arriver, ni le gain de point, ni la perte de vie. Ce qui permet que, lorsqu'un réapparaît directement sur un obstacle, il ait le temps de se dégager de cet obstacle, pour reprendre le jeu dans les meilleures conditions possibles. De plus, lorsqu'un joueur est donc "invincible", un message est affiché à l'écran, pour l'informer de son statut. Si ce message n'est plus présent, alors il est de nouveau un joueur pouvant intéragir avec son environnement, et ses conséquences.

5. Prochaine étape

Pour la prochaine fois, il faut que le jeu soit terminé, que la phase de résolution de bug soit bien avancée. Il faudra aussi que je prépare un scénario de jeu pour le présenter devant la classe. Des dernières retouches seront apportées si besoin durant les vacances, ou pendant la première partie du cours de mercredi prochain.