Rapport de projet : World IMaker

Clélie CHASSIGNET et Mélissande GAILLOT
7 janvier 2020

Contexte

Notre meilleur ami Toto aimerait beaucoup pouvoir créer un joli monde en 3D pour le présenter lors d'un entretien de recrutement pour la boîte de dessin-animés de ses rêves. Le problème, c'est qu'il pense qu'il n'a pas le temps d'apprendre à se servir d'un logiciel trop évolué comme Blender avant son entretien. Si c'était un IMAC 2, il aurait trouvé le temps d'apprendre quand même en sacrifiant ses heures de sommeil. Malheureusement Toto n'a pas la motivation d'un IMAC alors il aimerait beaucoup trouver un logiciel très simple d'utilisation dont il pourrait se servir sans avoir à regarder des dizaines de tutos sur le net. Comme c'est notre meilleur ami on lui propose de créer l'éditeur de scène le plus simple possible s'il nous paye une bière en échange. En trinquant, nous définissons donc le cahier des charges avec lui.

Cahier des charges

L'application à produire est un éditeur de terrains 3D, constitué uniquement de cubes. Le monde en 3D est donc un pavé de taille longueur*Largeur*Hauteur constitué de cubes unitaires. L'utilisateur doit pouvoir déplacer un curseur dans la scène afin de la modifier en changeant la couleur des cubes, en les créant ou en les supprimant. Pour cela, il faut imaginer un menu permettant de choisir les outils d'édition. Il faut également pouvoir ajouter des lumières à la scène, de type "lumière directionnelle" ou "point de lumière". Enfin, il faut pouvoir donner des exemples de scènes en proposant un système de génération procédurale de terrain.
Il est également demandé d'apporter deux fonctionnalités additionnelles parmi une sélection d'améliorations possibles. Nous avons choisi d'intégrer les fonctionnalités "sauvegarder" et "charger" afin de pouvoir réutiliser une scène déjà créée, et "discrétiser" pour permettre à Toto de modifier sa scène rapidement.

SOMMAIRE

1- Architecture

(Diagramme de classes + explications)

Au début du projet, nous avons décidé de ne pas utiliser la librairie “glimac” que nous utilisions en TP car nous ne comprenions pas toujours bien comment elle fonctionnait, ni même quand est-ce qu’on l’utilisait et à quelles autres librairies elle faisait appel.
Nous avons donc préféré abandonner la base que nous utilisions en TP et utiliser la base que nous a proposé Guillaume Haerinck, notre professeur de soutien. Ainsi, nous avons pu comprendre plus facilement quelles librairies nous utilisions et lui poser des questions sur leur fonctionnement.

2- Bilan de l’implémentation

	Implémentation		
Fonctionnalité	**Intégralement**	**Partiellement**	**Pas du tout**
Affichage d'une scène avec des cubes	x		
Edition des cubes	x		
Sculpture du terrain	x		
Génération procédurale		x (pas eu le temps de faire assez de tests pour avoir des beaux résultats)	
Ajout de lumières		x (on ne peut pas placer la lumière ponctuelle, ni régler leur intensité)	
Option 1 : Save/Open	x		
Option 2 : Niveau de discrétisation		x (pas encore terminé, pas utilisable en l'état)	

3- Les fonctionnalités requises

3.1- Affichage d'une scène avec des cubes

Nous avons décidé d'envisager notre monde comme un pavé de taille longueur*Largeur*Hauteur constitué de Cubes unitaires. Ces cubes peuvent être visibles ou non et de différentes couleurs. Nous avons donc commencé par créer une classe Cube qui contient notamment la position et la couleur du cube, ainsi qu'un booléen permettant de déterminer si le cube est visible ou non. Nous aurions dut mettre tous les attributs de la classe en privé et utiliser des méthodes "get" et "set" pour y accéder et les modifier mais lorsque nous nous en sommes rendues compte nous avions déjà trop avancé le projet pour tout modifier.
Nous avons stocké l'ensemble de tous les cubes de notre monde dans un tableau "stockCube" à une dimension de taille longueur*Largeur*Hauteur afin de pouvoir facilement y accéder et les modifier. Puisque notre monde est en trois dimensions et que notre tableau est en une seule dimension, il faut faire une conversion pour retrouver les coordonnées des cubes en fonction de leur indice dans le tableau :

- coordonnée x = indice%longueur
- coordonnée y = (indice/longueur)*Largeur
- coordonnée z = (indice/longueur)%Largeur

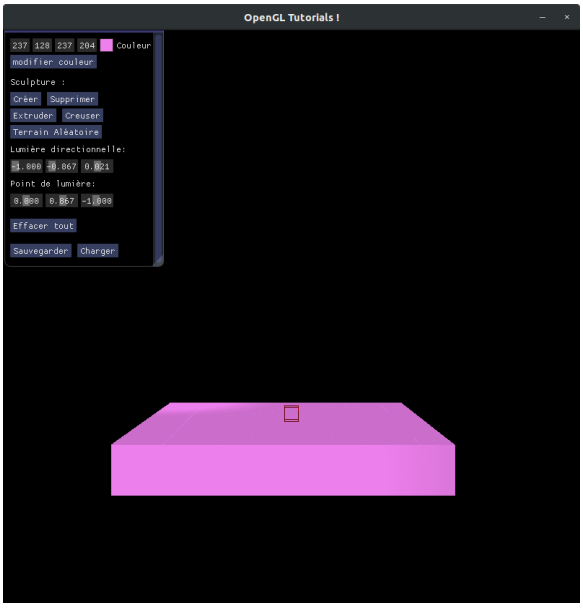
Initialement, le booléen de visibilité des cubes est à "false" car la scène est vide. On passe sa valeur à "true" uniquement pour les trois couches de cubes les plus basses qui constituent le sol de notre monde.
Enfin, dans la boucle de rendu, nous affichons tous les cubes dont le booléen de visibilité est à "true" à l'aide d'une méthode draw().
Nous avons également ajouté une caméra afin de pouvoir zoomer et dezoomer sur notre scène avec la molette de la souris, et tourner autour du centre de notre monde en maintenant la molette appuyée et en bougeant la souris, pour rendre l'édition de terrain plus facile en adoptant différents points de vue. Pour cela, nous nous sommes inspirés de la "TrackballCamera" vue en TP. Nous lui avons appliqué des matrices de translation afin de déterminer son emplacement initial lorsqu'on lance le programme.
Pour que l'affichage de la scène soit possible, nous avons créé le moteur de rendu et les shaders nécessaires.

3.2- Édition des cubes

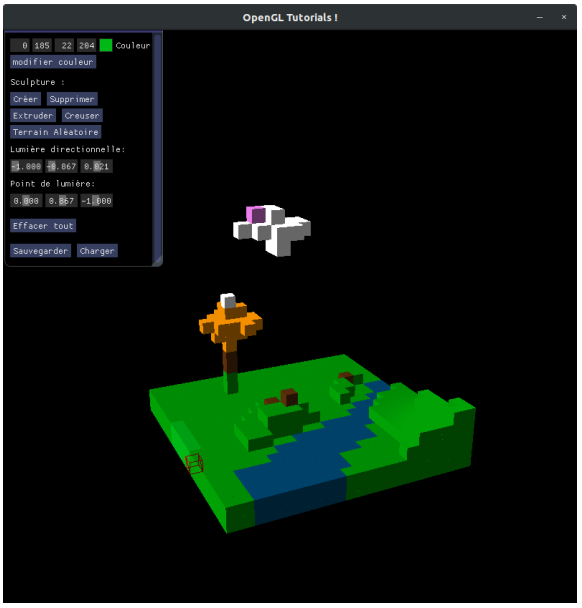
L’édition des cubes nécessite tout d’abord d’imaginer un système de curseur afin de pouvoir se déplacer dans la scène et sélectionner les cubes sur lesquels nous souhaitons agir.
La position de notre curseur est simplement déterminée par un entier qui correspond à l’indice du cube sur lequel il pointe dans notre tableau “stockCube” contenant tous les cubes du monde.
On peut diriger ce curseur à l’aide des flèches du clavier grâce à un événement dans la boucle de rendu qui change la valeur du curseur. Comme le curseur correspond à l’indice du cube dans le tableau à une dimension, il faut là aussi faire une conversion par rapport à notre monde en 3D. Par exemple, aller vers l’arrière dans le monde en 3D correspond en fait à ajouter la valeur “largeur” au curseur pour trouver l’indice du cube correspondant dans le tableau.
Afin de savoir où se trouve le curseur et pouvoir facilement se déplacer dans le monde 3D, nous avons décidé de le représenter visuellement en dessinant les arêtes du cube sur lequel il pointe.
Il est alors possible de changer la couleur des cubes “visibles”. Pour cela, nous avons créé une variable uniforme uColor dans notre fragment shader afin de pouvoir modifier sa valeur dans le main. Cette variable prendra alors la valeur que lui donne l’utilisateur. Pour cela il utilise un menu. Nous avons décidé d’utiliser la librairie “ImGui” qui permet d’éditer des interfaces homme-machine.
Cette librairie permet de proposer une palette de couleurs pour l’édition des cubes. La couleur sélectionnée par l'utilisateur est donc affectée à l’argument de couleur du cube sur lequel pointe le curseur si celui-ci est visible.

3.3- Sculpture du terrain

Pour la sculpture du terrain, nous avons ajouté des boutons à notre menu qui nous permettent soit de créer un cube à l’emplacement du curseur, c’est-à-dire le rendre visible en changeant la valeur du booléen de visibilité et lui donner la dernière couleur sélectionnée par l’utilisateur, soit de supprimer un cube, c’est-à-dire le rendre invisible.
Il y a également une fonction “extruder” qui permet de créer un nouveau cube au-dessus de la pile de cubes à laquelle appartient notre curseur. Pour cela il faut regarder un par un les cubes au-dessus du curseur jusqu’à trouver un cube “invisible” et le rendre visible (s’il n’est pas en-dehors du terrain).
De la même manière, la fonction “creuser” permet de supprimer les cubes situés en haut de la pile au-dessus du curseur. Il faut donc également regarder les cubes au-dessus du curseur jusqu’à en trouver un “invisible” et supprimer (rendre invisible) celui d’en-dessous qui correspond donc au dernier “visible”.



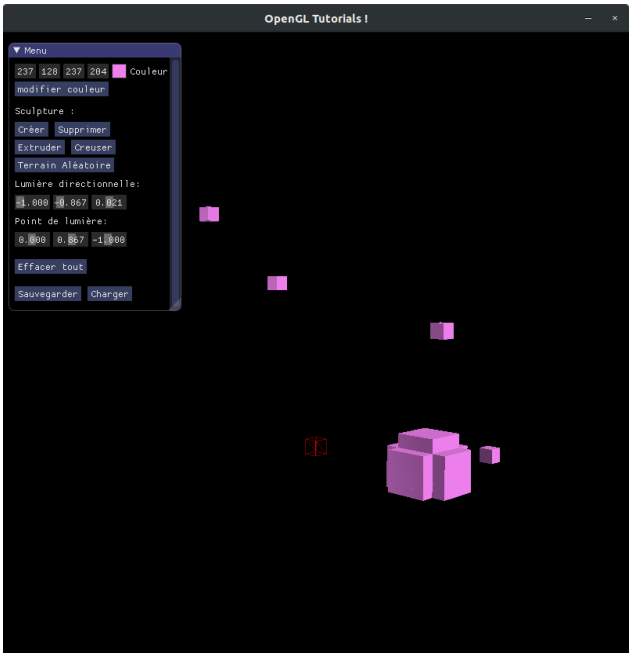
Initialisation de notre monde



Création d'un terrain

3.4- Génération procédurale

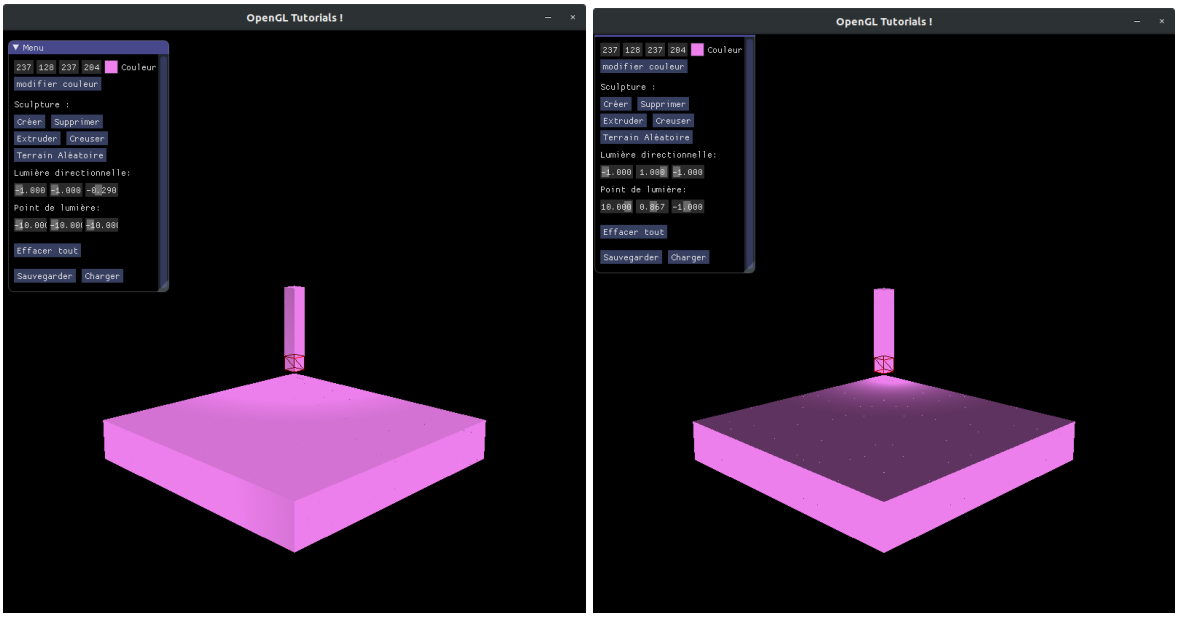
Afin de pouvoir proposer des exemples de terrains, nous proposons une génération procédurale de terrain à l'aide de radial basis function. Pour cela, nous tirons aléatoirement des points de contrôle dans notre monde, auquel nous affectons des poids, également tirés au hasard. Ces poids vont influencer la tendance de leurs voisins à être visibles ou non. Ainsi, plus un cube est proche d'un point de contrôle avec un fort poids, plus celui-ci va avoir tendance à être visible. A l'inverse, un cube proche d'un point de contrôle avec un poids très faible aura plutôt tendance à être invisible. Ainsi, à partir des points de contrôles tirés aléatoirement on peut calculer les "poids" de tous les autres cubes du monde afin de déterminer s'ils seront visibles ou non.
Ainsi, lorsque l'utilisateur clique sur le bouton "Terrain Aléatoire" dans le menu, le terrain précédent est effacé et laisse place à un nouveau terrain, généré de manière procédurale.



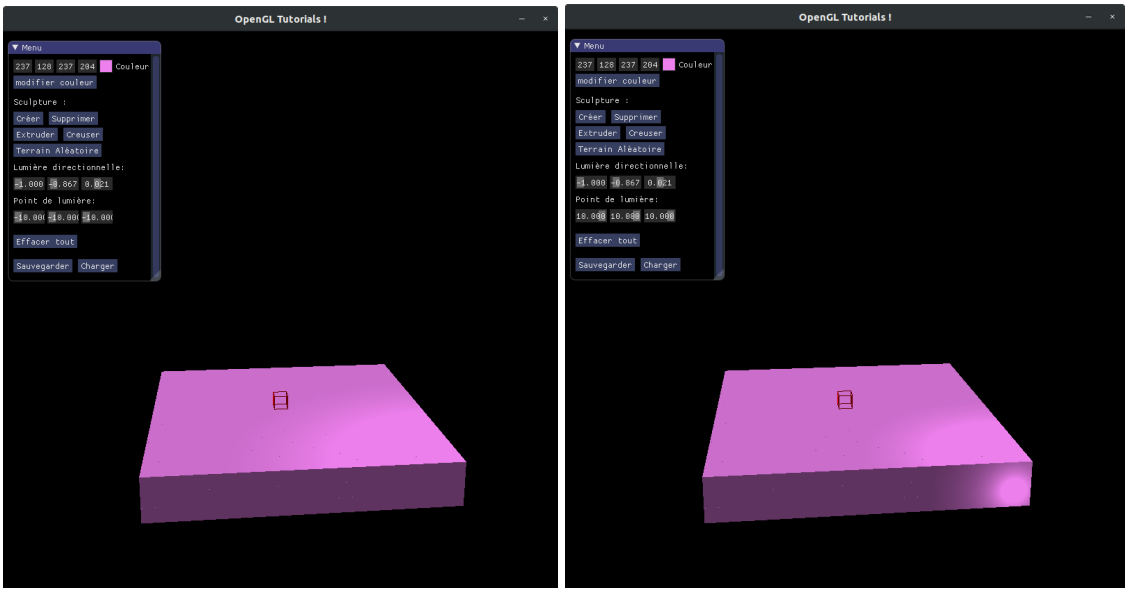
Rendu de terrain Aléatoire

3.5- Ajout de lumières

En ce qui concerne les lumières nous avons choisi de ne pas utiliser le modèle de Phong mais une méthode plus accessible et compréhensible. Pour cela, nous avons besoin des normales car elles indiquent la direction de chaque face et permettent de calculer la quantité de lumière que reçoit notre face en fonction de la source de lumière. Il faut ensuite, recevoir ces normales dans le vertex shader qui va les transmettre au fragment shader.
On crée enfin deux types de lumières. Pour commencer, la lumière directionnelle est décrite par une direction. Pour savoir à quel point la lumière nous éclaire, on regarde à quel point elle est alignée avec la normale. Pour cela on utilise le produit scalaire négatif car c'est à ce moment là que la lumière est orientée vers la face (la direction de la lumière oppose la direction de la normale). Puis on récupère le maximum entre le résultat du produit scalaire et une faible valeur (proche de 0) pour ne pas avoir des valeurs négatives de luminosité. Finalement, on multiplie cette valeur à la couleur.
Presque de la même façon on crée la lumière ponctuelle. La seule différence est qu'on passe une position à la lumière. On finit par les combiner en les additionnant.
L'utilisateur peut affecter différentes direction d'éclairage pour les lumières grâce au menu ImGui. De la même façon que pour les couleurs, cela modifie les variables uniformes uLumiereDirection et uPointLumiere, afin de leurs affecter leurs nouvelles valeurs.
Nous aurions pu améliorer les lumières en permettant à l'utilisateur de régler leur intensité et de choisir où placer les lumières ponctuelles.



Différentes directions sont affectées aux lumières directionnelles



Différentes directions sont affectées aux lumières ponctuelles

4- Les fonctionnalités additionnelles

4.1- Option 1 : Save/Open

Pour la première option, nous avons décidé de proposer de sauvegarder un terrain pour ensuite pouvoir l'ouvrir à nouveau à un autre moment, éventuellement le modifier et le sauvegarder à nouveau. De cette manière Toto pourra être sûr d'avoir

un super terrain à montrer, même s'il doit le faire en plusieurs fois à différents moments, et sans risquer de le perdre s'il fait une fausse manipulation.
Pour cela, nous avons identifié les paramètres importants à prendre en compte. Nous avons simplement besoin de savoir les indices dans le tableau "stockCube" des cubes visibles, ainsi que les composantes (RGB) de leurs couleurs.
Ainsi, lorsque l'utilisateur clique sur le bouton "Sauvegarder" du menu, il commence par choisir l'emplacement et le nom du fichier de sauvegarde en console. Ensuite, nous ouvrons un document à cet emplacement qui porte le nom choisi. S'il n'existe pas encore, il est créé, et s'il existe déjà les données sont écrasées pour pouvoir écrire les nouvelles données à la place.
Dans ce fichier, nous écrivons ensuite les données de la manière suivante : chaque ligne correspond à un cube "visible", les différents paramètres du cube sont écrits à la suite, séparés par un espace, dans l'ordre suivant : l'indice du cube dans le tableau "stockCube", suivi de la valeur de la composante rouge, puis verte, et enfin bleu.
Ce sont les seuls éléments que nous avons besoin de connaître pour pouvoir reconstruire la scène.
Enfin, il est possible d'ouvrir un fichier précédemment sauvegardé en cliquant sur le bouton "charger" du menu. Il faut ensuite taper en console l'emplacement et le nom du fichier que l'on souhaite ouvrir. Le programme va donc lire le fichier correspondant s'il existe de la manière suivante : la première composante de chaque ligne correspond à l'indice du cube dans le tableau, il faut donc aller dans cette case du tableau et le rendre visible. Il récupère alors les composantes de couleur du cube et lui affectent. Ainsi, dans la boucle de rendu, tous les cubes visibles sont dessinés avec la bonne couleur.

4.2- Option 2 : Discrétisation

Pour la deuxième option, nous avons tout d'abord essayé d'appliquer des textures sur nos cubes car cela nous semblait être la deuxième option la plus utile pour Toto afin qu'il puisse réaliser des scènes exceptionnelles. Comme nous l'avions déjà fait en TP, nous pensions pouvoir nous en inspirer. Seulement, comme nous n'avons pas récupéré la même base que dans les TP et que nous n'utilisons pas la librairie "glimac" nous avons eu du mal à mettre en oeuvre l'application d'une texture. Après des heures d'essais sans succès nous avons finalement décidé de choisir une autre option.
Nous avons donc choisi d'essayer d'implémenter l'outil permettant de changer le niveau de discrétisation du monde. Notre idée était de parcourir notre tableau "stockCube" avec un pas tel que nous ne regardons que les cubes situés en bas à gauche de chaque regroupement de 8 cubes (2x2x2). Pour chacun de ces regroupements de 8 cubes, nous regardons combien parmi eux sont visibles. S'il y en

a moins de la moitié, nous les rendons tous invisibles. S'il y en a plus de la moitié, nous rendons visibles ceux qui ne le sont pas encore. De plus, nous faisons la moyenne de chaque composante de couleur et nous affectons cette nouvelle couleur aux 8 cubes.
Seulement, en pratique, comme notre tableau "stockCube" est à une seule dimension il était difficile de trouver le pas tel que nous regardons seulement le cube en bas à gauche de chaque regroupement de huit cubes. Prises par le temps, nous n'avons malheureusement pas eu le temps de finir l'implémentation de cette fonction.

5- Difficultés rencontrées

L'une des difficultés rencontrées découle du fait que nous avons codé de manière trop linéaire en implémentant les fonctionnalités dans l'ordre du sujet sans vraiment se soucier de savoir si les décisions que nous prenions sur le moment n'allaient pas nous bloquer ou nous limiter pour implémenter les fonctionnalités suivantes. Parfois nous avons pu rattraper cette erreur par la suite mais quelques fois il aurait été trop compliqué de tout modifier et cela nous a posé problème.
Nous aurions également probablement eu besoin d'un cours supplémentaire sur les "cmake" car nous avons parfois eu des difficultés à ajouter des librairies par exemple.
De plus, avant de pouvoir commencer le projet il fallait être déjà bien avancé dans les TP d'openGL qui étaient assez longs à faire, ce qui nous laissait moins de temps pour l'implémentation du projet. Nous n'avons donc pas eu le temps de tout finir comme nous l'aurions aimé et surtout d'améliorer la propreté du code.

6- Retour d'expérience

6.1- Retour de Clélie

Je suis satisfaite du résultat que nous avons fourni. Tout d'abord parce que comparé au projet de l'année dernière, je trouve notre code plus propre et plus compréhensible. Bien sûr j'ai conscience que ce dernier reste à améliorer, mais en prenant compte de mes bases en C++ et en openGL je sens une nette évolution dans ma façon d'appréhender le code en général.

6.2- Retour de Mélissande

Mon retour d'expérience est assez positif dans l'ensemble. Certes, nous aurions pu faire mieux car il est toujours possible de s'améliorer mais nous n'étions même pas certaines d'arriver jusque là et nous sommes déjà fières du résultat.
Ce n'est pas le premier projet que je fais avec Clélie, nous connaissons notre manière de travailler et ce qui fonctionne pour nous. Nos niveaux sont relativement similaires. C'est pourquoi nous avons décidé de tout coder ensemble du début à la fin. Cela pourrait être vu comme une perte de temps pour d'autres qui préfèrent se partager les tâches, mais pour nous c'est réellement un gain de temps. Cela nous permet de vérifier que l'autre ne fait pas d'erreur et si l'une a compris quelque chose que l'autre n'a pas compris elle peut lui expliquer directement. De plus, cela nous permet de vraiment comprendre tout le fonctionnement du programme et de ne pas perdre de temps à essayer de comprendre le code de l'autre s'il a fait des modifications sans nous et tout ce que ça pourrait impliquer ou modifier. C'est une méthode qui nous correspond et qui nous permet d'apprendre et de progresser ensemble.
Tout au long du projet, il était assez satisfaisant de constater nos progrès par rapport au projet du semestre 2. Notre code est loin d'être aussi propre et optimisé qu'on le souhaiterait mais il l'est déjà beaucoup plus que l'année dernière ce qui est très encourageant et nous donne envie de progresser encore plus. Je pense que nous avons appris beaucoup de choses en faisant ce projet et qu'il nous reste encore plus de choses à apprendre pour atteindre le niveau qu'on vise.

7- Conclusion