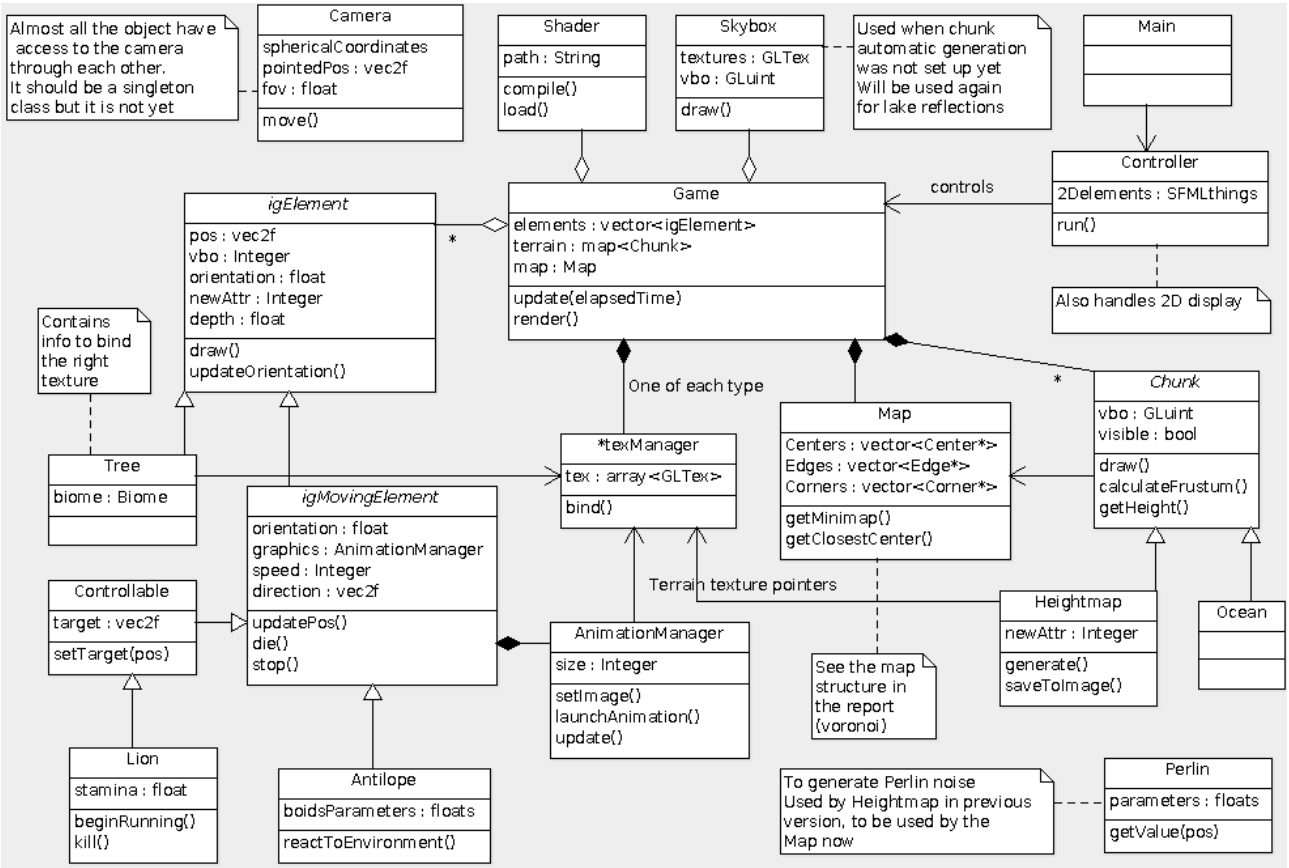
Rapport de projet

Moteur de jeu hybride 2D-3D généré procéduralement

Le projet a été réalisé avec OpenGL 2.1 dans un contexte SFML.

La structure UML du projet est la suivante : (disponible en .png ou .uml sur le dépôt git)



I/ Génération procédurale du terrain

1. *Première solution : bruit de Perlin*

J'ai basé ma génération de terrains sur la génération de heightmaps. Dans un premier temps pour obtenir du relief, j'ai utilisé une génération par bruit de Perlin, qui est une forme de bruit cohérent par superposition d'harmoniques. Je n'ai pas utilisé le vrai bruit de Perlin en tant que tel car je n'ai pas fait de génération de vecteurs mais une génération scalaire, et on utilise une interpolation cubique pour obtenir les différentes harmoniques après. Pour cela je me suis basé sur le tutoriel de developpez.com :

http://cochoy-jeremy.developpez.com/tutoriels/2d/introduction-bruit-perlin/

2. *Une structure modulaire*

Le problème est qu'il n'était pas envisageable de générer de la sorte l'ensemble du terrain, qui devait être un minimum grand. J'ai donc du modulariser mon modèle, en utilisant des « chunks », petits carrés de terrain dont la hauteur était justement générée par bruit de Perlin. Cependant, n'ayant pas réussi à générer un bruit de Perlin en fonction des chunks voisins, j'ai choisi de générer d'abord la carte de hauteurs puis de l'adapter aux chunks voisins en faisant un lissage linéaire.

Le premier avantage de cette méthode est que les chunks peuvent être générés au fur et à mesure que l'utilisateur se déplace plutôt que de tout générer dès le début. L'autre avantage important de cette structure modulaire est qu'elle me permet d'utiliser plus facilement le frustum culling, qui consiste à afficher uniquement les chunks qui sont dans le champ de la caméra : il suffit de comparer les 4 coins du chunk à la pyramide de vision de la caméra (je n'ai pas pris en compte le far plane). A noter que dans le cas d'un chunk non plat, si on compare uniquement avec les coins on risque de ne pas afficher le chunk s'il est dans le champ de vision (si un bord est toujours plus haut que les coins par exemple. J'ai donc choisi de comparer aussi avec les points les plus hauts et les plus bas de chaque côté. Cela n'a pas de garantie mathématique mais marche bien en pratique.

3. *La logique globale*

Une telle structure constituée uniquement de blocs similaires présente peu d'intérêt à être vaste, c'est pourquoi j'ai cherché à avoir une logique globale pour ma génération de terrain. Je voulais générer une grande carte avec différentes hauteurs globales, et des biomes pour adapter mes textures et mes paramètres de bruit de Perlin.

J'ai pour cela utilisé un article de Stanford qui mettait à disposition une génération d'île à base de diagrammes de Voronoi, qui proposait un générateur de map avec les informations à télécharger dans un fichier xml :

http://www-cs-students.stanford.edu/~amitp/game-programming/polygon-map-generation/

A chaque cellule de Voronoi correspond un biome différent, et les hauteurs sont contenues dans les centres et dans les coins qui correspondent aux extrémités de chaque segment entourant une cellule de Voronoi. J'ai donc implémenté une minimap afin d'avoir une vision globale du monde, et pour pouvoir se déplacer plus vite en cliquant dessus. Je n'ai cependant pas eu le temps d'implémenter les rivières. J'ai donc programmé une classe qui permettait d'interpréter le fichier XML et j'ai pu générer mes hauteurs à partir de ces

informations : pour chaque point de la heightmap, il faut trouver le centre dont il est le plus proche (pour trouver à quelle cellule il appartient), puis parcourir l'ensemble des triangles constituant la cellule (définis par le centre et un bord), déterminer les coordonnées barycentriques, et si celles-ci sont dans [0,1], c'est que le point est dans le bon triangle. On peut alors effectuer une moyenne des hauteurs pour avoir une sorte d'interpolation linéaire.

C'est ici que j'ai eu mon plus gros blocage (qui n'est toujours pas résolu à l'heure actuelle) : lorsque l'on cherche dans quel triangle se situe le point, il arrive régulièrement de ne pas en trouver (ce qui est bizarre car à ce moment le point serait dans une cellule adjacente et on aurait donc trouvé un autre sommet plus proche. Pour éviter que le problème se perçoive trop, j'ai choisi de prendre comme base le triangle qui donnait les coordonnées barycentrique les plus plausibles, mais on observe quand même des artéfacts.

3. *Optimisation*

Une autre problématique est que cette approche pousse à effectuer de très nombreux appels à cette fonction donnant le centre le plus proche d'un point, qui est donc un point central pour les performances du moteur. Dans un premier temps, je faisais un parcours linéaire de tous les centres pour trouver le minimum, ce qui donnait des performances non acceptables. J'ai donc choisi d'utiliser un kd-tree pour avoir une recherche en O(ln n) qui était beaucoup plus intéressant, j'ai utilisé pour cela la bibliothèque FLANN, et les résultats sont maintenant plus probants.

II/ Intégration des graphismes 2D

1. *Les sprites 2D*

J'ai conçu mon moteur afin qu'il affiche des sprites 2D dans un environnement en 3D. Pour garder un aspect un minimum réaliste, j'ai choisi d'utiliser les sprites d'Age of Empires (I et II), que j'ai extraits des jeux grâce au logiciel de modding Turtle Pack (j'ai fait de même pour les textures). Le principal intérêt de ces sprites est qu'ils sont définis pour 8 orientations différentes, ce qui permet d'avoir cet aspect 3D : lorsque l'on tourne autour des objets, le moteur va choisir l'orientation qui est la plus adaptée. J'ai ainsi généré des planches pour les animations à partir de ces graphismes, dont les informations nécessaires à leur interprétation sont contenues dans des fichiers XML. Le culling pour ces objets se fait pour l'instant simplement en fonction de celui des chunks, c'est-à-dire qu'on gère l'affichage et la mise à jour des sommets de l'objet seulement si le chunk auquel il

appartient est affiché. Il faudrait aussi prendre en compte une sorte de far plane pour optimiser le rendu.

2. *La flore*

La flore est ici générée en fonction des biomes, avec encore un fichier XML qui permet de donner les informations de densité de chaque biome. Le jeu va donc, pour chaque cellule de Voronoi, générer nombre d'objet 2D représentant des arbres dont la texture sera choisie au hasard parmi celles qui correspondent au biome de la cellule. Cependant, pour l'instant, tous les arbres sont générés d'un coup au début du jeu (ce qui explique pourquoi le programme est lent à se lancer). On pourrait aussi générer les arbres uniquement au moment où le chunk est généré.

3. *La faune, IA par algorithme de Boids*

Pour ce qui est de la faune, il n'y a pour l'instant que des antilopes et des lions (la faune variée et générée en fonction des biomes n'était pas une priorité). Un troupeau de 20 antilopes est généré à l'endroit où la caméra apparaît. On peut faire apparaître des lions avec un clic droit, et ils se contrôlent à la manière d'un jeu de stratégie en temps réel. Le principal intérêt ici était de travailler sur l'intelligence artificielle des gazelles, qui devaient fuir les lions mais aussi se comporter en troupeau. Je voulais implémenter une sorte d'algorithme de Boids pour gérer leur mouvement. Je me suis pour cela basé uniquement sur la page Wikipédia qui en explique les grandes lignes, et j'ai ensuite adapté l'algorithme à mes besoins pour le projet.

Les antilopes ont donc trois états : IDLE, RECOVERING et FLEEING . Lorsqu'elles sont en IDLE, elle se déplacent aléatoirement et restent brouter aux alentours, tout en évitant de trop s'éloigner les unes des autres. En FLEEING, elles fuient les lions lorsqu'elles les voient en priorité tout en cherchant à ne pas trop s'éloigner les unes des autres, et en RECOVERING elles marchent pour ne pas arrêter brutalement leur course, en cherchant à se regrouper.

III/ Perspectives

Il me reste encore pas mal de travail à effectuer cet été :

1. *Pour peaufiner*

Un des premiers problèmes qui se voit est le problème des textures, il n'y a pas de fondu entre les différents biomes. Il faut que j'utilise le multi-texturing pour pouvoir dessiner des zones de transition aux bordures des biomes. Je suis aussi assez déçu du mipmapping par défaut d'OpenGL (malheureusement nécessaire pour éviter les scintillements), qui floute à mon sens trop vite les textures, on perd ainsi le « grain » propre à Age of Empires, il va falloir que je règle ceci.

Il faut bien sûr que je finisse d'intégrer mon travail à la génération de terrain : les rivières, mais surtout l'ajout du bruit de Perlin qui reste cohérent avec le relief global de mon île.

2. *Optimisations*

Le rendu peut encore être optimiser davantage : lorsque l'on est loin d'un chunk, il est inutile d'afficher tous ses sommets, et il faut donc que j'utilise du clipmapping pour avoir, au même titre qu'une mipmap, plusieurs niveaux de précision sur la carte de hauteurs. Ca permettrait tout d'abord d'avoir un rendu beaucoup plus rapide, mais surtout d'assurer une gestion plus dynamique de la mémoire, et donc de décharger les sommets qui sont loin de la caméra. En effet, pour l'instant si on se balade sur l'intégralité de la carte, on finit par dépasser la mémoire vidéo (tout du moins sur mon ordinateur).

La programmation multi-cœurs avec openMP est aussi quelque chose que je voulais aborder, et même si j'ai parallélisé quelques boucles pour gagner un peu en vitesse, je voudrais surtout permettre de faire la génération de terrain en arrière plan, pour ne pas avoir de chute de fps lorsque l'on doit générer de nouveaux chunks.

3. *Pour aller plus loin*

Je voulais profiter du projet pour me familiariser avec de nombreuses techniques de base d'OpenGL que je ne maitrise pas encore, par exemple les vertex buffer objects, que j'ai choisi d'utiliser partout. La maitrise de ce concept m'a pris du temps, et je pense que c'est vraiment un des principaux intérêts de ce projet. Il y a donc encore de nombreuses techniques que je veux utiliser pour les maitriser un peu mieux : le bump mapping, la

génération de reflets sur les lacs par rendu dans un frame buffer object (qui me permettrait aussi de rendre à nouveau utile ma skybox), et bien sûr le multi-texturing. Je dois aussi trouver un moyen efficace pour générer des ombres pour les modèles 2D (je pensais à choisir la bonne orientation du modèle, la griser, et la rendre sur un rectangle sur le sol).

Conclusion

Ce projet, que j'ai réalisé seul, m'a permis d'avoir une grande liberté que je ne pense pas pouvoir retrouver plus tard dans le monde scolaire ou professionnel. J'ai donc essayé d'en tirer le maximum personnellement et j'y ai investi beaucoup de temps. Le projet était cependant assez ambitieux et je n'ai pas réussi à atteindre mes objectifs à temps, même si je compte le poursuivre cet été, et que le chemin a été très instructif. Je n'avais en effet pas réalisé à quel point le développement prenait du temps, notamment la résolution de bugs. Je pense cependant avoir gagné énormément en vitesse de développement en faisant face à des problèmes classiques et en ayant maintenant mes propres modèles.

Mon projet n'avait pas non plus de cahier des charges précis, si bien qu'il a évolué au fil du temps, ce qui a causé plusieurs restructurations, qui auraient été très fastidieuses dans le cadre d'un projet de groupe. J'ai quand même essayé de garder un code clair et modulable, pour qu'il soit exploitable par la suite.