Compte Rendu

Elise Massa, Emily-Rose Strich

Projet les Ptimacs : Elmer_The_Square

Synthèse d’image I, Programmation-Algorithmique Avancée

I. Notre Projet

Notre jeu de plateformes Elmer_The_Square reprend le principe du jeu “Thomas was Alone”. Il Constitué de 3 niveaux de difficulté croissante. Le niveau se joue avec un joueur, le niveau 2 avec 2 joueurs et le niveau 3 avec 3 joueurs différents. Pour passer d’un niveau à l’autre, il suffit que chaque joueur rejoigne son “portail” (un carré de même taille et dimension que le joueur).

L’univers graphique est inspiré de la série littéraire ***Elmer the Patchwork Elephant*** (souvent abrégé en *Elmer*) de David McKee.
C’est une œuvre que nous apprécions toutes les deux particulièrement étant enfant et qui se prêtions bien à une réinterprétation dans un jeu vidéo constitué exclusivement de rectangle. Ainsi, le décor et les joueurs sont des couleurs de l’éléphant : jaune, orange, rouge, rose, violet, bleu, vert, noir et blanc.

II. Notre Méthode de Travail

Nous avons toutes les deux codé sur Visual Studio Code en C++ et avons créé un Git pour le projet.

- *Semaine 0* : nous avons d'abord réfléchie à la structure du projet ensemble avant de nous lancer et réaliser le code de base qui permet d'ouvrir une fenêtre noire à partir du Makefile que nous avons réalisé à deux.

Puis nous nous repartissions le travail semaine par semaine en fonction de l'emploi du temps de chacune et des difficultés rencontrées :

- *Semaine 1* : Emily Rose a commencé par faire le code de dessin du joueur en OpenGL et celui des déplacements de base avec SDL. Elise a récupéré ce code pour l'implémenter au projet uniquement via la création de classe et d'objet.
- *Semaine 2* : Ensuite Emily-Rose a travaillé de son côté sur la réalisation du QuadTree tandis qu'Elise "préparait le terrain" pour l'ajout de ce dernier dans le code en organisant l'héritage des différents fichiers entre eux.
- *Semaine 3* : Emily-Rose ajoute au projet les interfaces de début, de jeu et de fin pendant qu'Elise s'occupe de créer un fichier .txt dans lequel les coordonnées des différents carrées de la Map peuvent être stockées, ces valeurs sont ensuite récupérées par une fonction qui initialise la carte et Elise a ainsi pu dessiner la Map (à partir de dessin sur Figma). Au même moment Emily-Rose, commence à coder les interfaces pour passer d'un niveau à l'autre et d'un joueur à un autre.
- *Semaine 4* : Nous avons travaillé ensemble sur le début des collisions afin que chacune saisisse bien les parties codées par l'autre, puis Emily-Rose a travaillé sur la réalisation et le fonctionnement final de ces collisions pendant qu'Elise a avancé sur le rapport du projet et les graphismes. (réalisation des Maps des 2 autres niveaux, changement de couleur de la map…)
- *Semaine 5* : Finalisation du rapport et rendu du projet.

III. Détails Techniques

Pour démarrer, nous sommes partis du fichier minimal.c et du Makefile de Monsieur Dumazet, pour avoir une initialisation de la SDL et d'OpenGL fonctionnelle, et l'ouverture d'une fenêtre. Puis nous l'avons séparé dans nos fichiers main.cpp et game.cpp en classes, et nous avons donc poursuivi en C++.

- **Makefile**

Pour lancer le jeu, il suffit d'exécuter les commandes *Make Game* puis *bin/main* dans le terminal. Le Makefile compile ainsi le fichier main.cpp qui lui fait appel aux fichiers Game, StartingInterface, GamingInterface, EndingInterface qui permettent au joueur d'interagir avec le jeu (et de s'éclater ;)).

- **Programmation en objet, avec des classes et de l'héritage**

Notre code a été découpé en différents fichiers cpp chacun relié à un fichier hpp auquel appartient une classe : Game, Map, Player, QuadTree, Square, etc.
Les classes permettent ainsi au code d'avoir une structure la moins lourde possible et laissent la possibilité d'implémenter de nouvelles fonctionnalités au code en ciblant directement l'objet et sa classe associée. De plus, il était aussi plus facile pour nous de nous répartir le code à faire séparément et de l'associer ensuite avec des classes.

- **Interfaces**

Nous avons créé 3 interfaces différentes, de sorte à ce qu'en lançant le jeu, on arrive sur un écran d'accueil. Puis en appuyant sur Entrée, on arrive sur l'interface de jeu. Elle se divise en deux niveaux. Enfin, lorsque le joueur a passé tous les niveaux, grâce à un booléen qui devient vrai, il arrive directement sur un écran de fin. Les interfaces sont gérées dans la classe Game. Chacune s'initialise grâce à son propre constructeur, puis il y aura une gestion des évènement (handleEvents()), une actualisation (update()), et un rendu (render()), en boucle. Les interfaces de début et de fin utilisent des textures.

- **Passer les niveaux**

Sur la Map, il y a le Player, que l'on peut bouger, ainsi que son alter-égo, le Winner. Celui est placé à l'arrivée de la Map, et est formé seulement d'un rectangle vide au contour blanc. C'est quand le Player arrive aux coordonnées du Winner que le jeu passe au niveau suivant. Au niveau 2, comme il faut déplacer deux Players, lorsqu'il y en a un qui arrive au niveau de son Winner, il devient blanc. À ce moment, on change

automatiquement de Player, et il n’est plus possible de déplacer celui qui est arrivé à destination. Il faudra alors déplacer le 2e Player jusqu’à son Winner.

- **Gestion de la caméra**

La caméra a été créée à l’aide de translations. Le Player que l’on bouge doit toujours être positionné au centre de l’écran, et quand on le déplace, c’est en fait toute la Map que l’on translate. Elle est utilisée dans la GamingInterface, et est appelée dans l’actualisation du jeu, après la gestion des événements (donc des déplacements) et avant de redessiner le jeu.

- **QuadTree**

Le QuadTree est une autre classe. Il y en a deux, un pour chaque niveau. À chaque QuadTree, on insère toutes les boxes de la Map. Puis celui-ci se divise pour qu’à chaque position du joueur, on ne prenne en compte que 4 boxes maximum. Il est composé de plusieurs fonctions, qui l’initialisent, déterminent s’il est une feuille, insèrent des boxes, et cherchent des boxes en fonction d’une position.

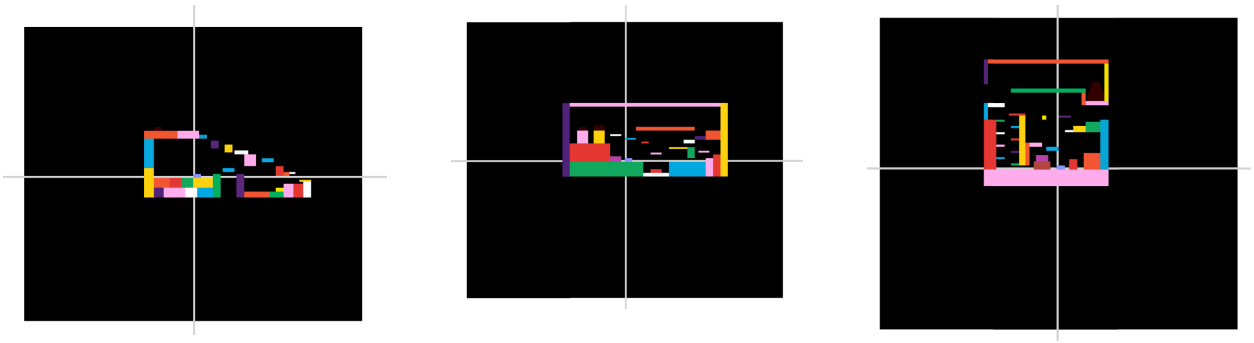
Capture d’écran d’un test sur la division de la Map dans le QuadTree.

- **Collisions**

Pour tester les collisions, nous avons une fonction associée au Quadtree, qui boucle sur chaque boxes du QuadTree lié à la position du joueur (soit maximum 4 boxes), et qui détermine s’il y a collision ou non. Pour cela, nous avons considéré que si le Player est complètement à droite de la boxe du décor, il n’y a pas de collision, et de même s’il est complètement au-dessus, en dessous, et à gauche. Si ce n’est pas le cas, il y en a une. Dans la gestion des événements, si le joueur déplace le rectangle, ses coordonnées changent, et s’il y a une collision avec les nouvelles coordonnées, on les fait revenir en arrière. Ensuite, on dessine le tout.

- **Coordonnées de la Map stockées à part dans un fichier texte**

Nous avons fait le choix de stocker les informations (position x, position y, hauteur, largeur) de chaque bloque de la Map dans une ligne du fichier map.txt dans le dossier *data* sous la forme : "x., y., h., w., ". Ces informations sont récupérées par la fonction initDecor() de la classe Map. La fonction lit ligne par ligne le fichier en récupérant chaque informations comme un string avant de le convertir en float et de la stocker dans le tableau correspondant : X[] pour x, Y[] pour y, etc. Une fois les tableaux initialisés, la Map est dessinée dans le constructeur de la classe éponyme et chaque bloc devient un AABB du QuadTree et rentre ainsi dans l'algorithme de collision.



Capture d'écran des maps dessinées sur Figma des niveaux 1, 2 et 3 (de gauche à droite)

(Les niveaux 1 et 2 sont jouables dès le lancement du jeu, le niveau 3 sera disponible dans le DLC qui sortira quelques mois après le lancement du jeu ;))

- **Passage d'un Player à un autre**

Lorsque le joueur passe un niveau, il rencontre un nouveau Player, un cube d'une taille différente. Pour prendre le contrôle du Player, il doit appuyer sur I, de même s'il veut reprendre contrôle du Player initial. Les niveaux sont conçus pour être réalisés en alternant entre les Players puisque certaines plateformes ne sont atteignables que par certains cubes ou grâce à certains cubes. En changeant de Player, celui qui est sélectionné est automatiquement positionné au centre de l'écran grâce à la caméra.

- **Jump**

Nous avons rencontré des difficultés à coder des sauts pour lesquels la gravité fonctionnait avec fluidité. Nous avons donc fait le choix de fixer une certaine hauteur maximale du cube par rapport au sol avant que celui-ci ne retombe par terre. De plus,

chaque cube possède une hauteur par saut différente, plus le cube est gros plus il peut sauter haut. Sans cette fonctionnalité, nos niveaux ne seraient pas réalisables.

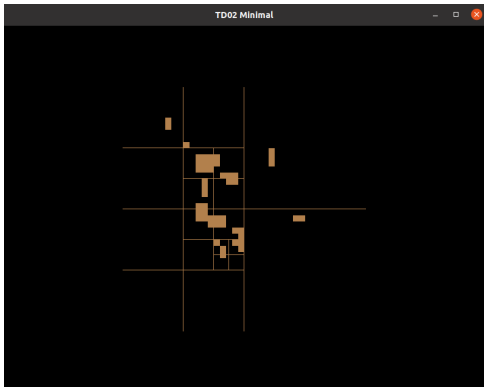
- **Couleurs**

Comme nous voulions garder un univers coloré, nous sommes parties sur une Map qui change de couleur. Les couleurs sont générées de façon aléatoire à partir d'une gamme de couleur prédéfinie, et sont appelées lorsque l'on dessine le décor. Cela rajoute une animation à notre jeu.

IV. Difficultés rencontrées

- **Faire le QuadTree**

Pour créer le QuadTree, il a fallu le faire sur un fichier à part puisque cela nous posait trop de difficultés de le faire en même temps que la réalisation du jeu. Nous avons mis une dizaine de jours à obtenir un QuadTree fonctionnel. Il insère bien les boxes et se divise en 4 récursivement.



Capture d'écran de la première fois où le QuadTree a fonctionné, avec des boxes aléatoires.

- **Les collisions**

Bien que dans le fichier test du QuadTree, à part du jeu, nous parvenons à déterminer la présence de collisions, lorsque nous sommes dans le jeu, la gestion des collisions n'est pas fonctionnelle. Elles ne sont pas toutes reconnues.

- **Manipuler les classes, comprendre les constructeurs**

C’était la première fois pour toutes les deux que nous devions réaliser un code aussi important en cpp. Emily-Rose avait eu l’occasion de se familiariser avec la poo et les classes lors du Pokimac, mais c’était la première fois qu’Elise devait gérer ce type de structure. C’est pourquoi elle s’est occupée de l’organisation des fichiers et la construction des principaux objets : le joueur et la Map. C’était un véritable défi que de s’attaquer aux classes sans n’en avoir jamais fait avant et cela a posé problèmes au début mais on a assez vite pu s’adapter et réussit à comprendre la logique nécessaire à cette structure et donc d’avoir des bases solides pour l’implémentation du reste du code.

- **Organisation des différents éléments**

Dans la continuité de la difficulté précédente, nous avons eu du mal à assembler nos codes avec les classes. Nous avons travaillé dans des fichiers indépendants à ceux du git lorsque cela était possible afin de limiter au plus les erreurs. Puis lorsque le code marchait indépendamment, on l’ajoutait au jeu (comme ça a été le cas pour la caméra ou les codes de dessin par exemple).
C’est souvent là que nous avions eu du mal à bien distinguer quelle fonction appartenait à quelle classe et à quel moment précis du code il fallait faire appel à celle-ci.

- **Les Sauts**

Comme expliqué dans la partie précédente, nous n’avons pas réussi à coder les sauts comme souhaités au départ, ils sont basés sur un système de maximum de hauteur par rapport au sol avant que le cube ne retombe. Il cesse de retomber lorsqu’il y a une

collision avec le sol, ce système marche bien en théorie, mais en pratique vu que certaines collisions ne sont pas détectées, les sauts ne sont pas toujours fonctionnels.

V. Améliorations Possibles

- Régler les problèmes de collision et les problèmes de saut qui en découlent.
- Ajouter un système de gravité et d'accélération vers le bas
- Gérer les déplacements et la gestion de la caméra en ajoutant un paramètre de vitesse, pour les rendre plus fluides.
- Bonus : Lorsque le joueur passe un niveau il rencontre un nouveau cube et à la fin de chaque niveau avec ce cube il fusionne avec et ainsi de suite jusque construire Elmer l'éléphant

Finalement, ce projet a été pour nous un vrai défi. Même si nous ne sommes pas parvenues à créer un jeu aussi fonctionnel que Thomas Was Alone, nous avons appris énormément en codant Elmer_The_Square. Nous sommes contentes d'avoir réussi à maîtriser un code aussi important par nous mêmes. C'était un projet super enrichissant et intéressant, nous avons aimé travailler dessus.