Nathan ROTH – M1 CMI IIRVIJ

Projet Arkanoïd – Rapport

Introduction

Roth Nathan :

- Architecture générale (dépendance des classes, héritage, etc....)
- Réalisation des fonction de « dessins » (assigner une texture à un objet, prendre une certaine partie des fichiers bmp pour rogner l'image, etc....)
- Gestion de l'exécution du jeu (transition, affichage, fenêtre win/lose, menu)

Schneberger Maxime :

- Mise en place du gameplay général :
  - Mouvement du vaus, de la balle
  - Gestion des collisions
  - Système de bonus
- Variables liées au joueur
- Réalisation des niveaux par fichiers textes

Affichage des éléments visuels

Pour afficher un objet à l'écran en SDL, il faut créer et remplir une SDL_Surface et l'appliquer sur la surface de la fenêtre. Un jeu 2D fonctionne généralement sur le principe d'une spritesheet : une seule image sur laquelle se trouve les différents objets du jeu (ou versions de l'objet dans le cas d'une animation). Cette image n'est chargée qu'une seule fois en mémoire, permettant de diminuer la charge de travail du CPU. Dans notre projet, une telle spritesheet est un objet Atlas. Initialisé en indiquant le chemin vers une spritesheet, l'objet Atlas abstrait un type SDL_Surface, rempli par l'image en argument.

Cependant, il faut découper la zone correspondant à un objet à chaque fois que l'on veut l'afficher. On fait alors appel à un objet Texture, l'objet de plus bas niveau pouvant être dessiné. Une Texture a la capacité de n'afficher qu'une partie de l'atlas à un endroit donnée sur une surface.

Maintenant que nous avons la capacité d'afficher une portion d'image, et donc les objets du jeu, il nous est possible de créer des Sprites. C'est un niveau d'abstraction un peu plus élevé que la Texture, permettant une utilisation plus aisée en tant qu'objet pour un jeu. Il est aussi possible de définir une animation. En effet, la fonction de dessin prend en argument un paramètre de temps, permettant de savoir quand changer de version. Lorsque le changement est requis, les coordonnées de la Texture abstraite par le Sprite sont modifiées pour atteindre la version suivante.

Chaque élément du jeu jeu faisant partie du gameplay (brique, vaisseau, balle etc.) dérive de Sprite.

Les Widget sont des éléments de l'interface utilisateur. Tous n'utilisent pas forcement un Atlas pour s'afficher, comme la surface, qui est une abstraction directe de la Surface SDL.

Le Text en revanche a besoin d'une table de caractère, un atlas donc, pour y puiser les symboles à afficher. Dans notre projet nous utilisons la table fournis par le professeur. Cette table compile les caractères dans l'ordre ASCII, permettant de faire correspondre les coordonnée du caractère sur l'Atlas directement avec les caractères du texte en lui-même.

Parsing des fichiers de niveau

Nos niveaux sont décris par des fichiers textes, dont l'architecture est décrite dans le rapport de Maxime Schneberger. Le parser a été écris à la main car il était assez simple. La lecture se fait ligne par ligne, puis chaque ligne est transformée en stringstream pour être analysée. Le parser génère en sortie un objet de type Level, contant les information du niveau parsée ainsi que la liste de brique.

Calcul vectoriel

Pour plus de confort de programmation, j'ai crée une classe de vecteur mathématique 2D. En effet, chaque position, taille, déplacement etc... a besoin d'être représenté et manipulé pour permettre une interaction entre les objets. La classe Vec2 répond à ce besoin. Elle est templaté et permet donc l'utilisation de plusieurs type de données (entiers, flotant ...). Elle implémente aussi la surcharge d'opérateur, permettant une programmation plus claire.

La classe Game

La classe principale Game contient l'initialisation du jeu ainsi que la boucle principale. L'initialisation lance d'abord la lecture des fichiers de niveau contenu dans le dossier donnée en argument. Les niveaux sont enregistrés dans une liste chaînée dont le premier élément est gardée à part en mémoire afin de pouvoir recommencer le jeu depuis le début. Puis il initialise les spritesheet des objets et du texte, eux aussi donnés en argument. Le jeu est ensuite lancée en boucle sur lui-même (voir main.cpp) dans l'attente d'un flag EXIT pour quitter complètement. Le Game a une fonction spécifique pour l'affichage du menu principale.

La boucle principale s'occupe, dans l'ordre :

- de la gestion des inputs
- des modifications et déplacements éventuels des objets
- du dessin de la fenêtre

Les menus de transition

Les menus de transition, ou SubWindow dans le code, sont les message affiché lorsque le joueur met le jeu en pause, perd, gagne, ou passe au niveau suivant. Un choix de deux options sont présentées à l'utilisateur et celui-ci doit cliquer pour continuer dans l'application.

Cette objet est en fait une mini classe Game, car elle possède une boucle de dessin principale qui ne s'arrête que lorsqu'elle reçoit une certaine entrée de l'utilisateur. Typiquement elle attend que l'un des deux « bouton » soit cliqué, chacun renvoyant un code spécifique pour le retour de la fonction show. La fonction show bloque l'exécution du bout de programme appelant, ce qui provoque la pause par exemple si elle est appelée depuis la gameLoop.

Un menu de transition contient un objet UICollection. C'est un objet qui contient des pointeurs vers les objets de l'interface à dessiner. Ces objets (pointeurs) se voient assignés un tag (label) et peuvent être récupérés grâce à ce label. Il possèdent aussi un z-index déterminant l'ordre dans lequel ils vont être dessinés.

Conclusion

Ce projet était assez intéressant et a posé de nombreux challenges. Avec plus de temps, l'architecture aurait pu être améliorée et harmonisée, notamment dans l'abstraction des objets dessinables où Widget et Sprite auraient pu être fusionnés. Le callback des évènements pourrait aussi être amélioré.