Projet Casse-Tête

Jérémie Pagny N°Étudiant : 12311178
Kevin Havranek N°Etudiant : 12310320

Introduction

Et voici le rapport tant attendu de notre projet, dans le cadre de la validation du cours nous avons donc dû développer un casse-tête sur Android. Pour ce projet nous voulions pouvoir faire le faire en 3D, car cela nous permettrait de découvrir comment fonctionne le NDK pour pouvoir développer en C/C++ et ainsi utiliser les fonctionnalités de la bibliothèque externe OpenGL ES 2.0.

Le rapport sera constitué de 2 parties, la première concernera le moteur du jeu développé en JNI et la deuxième concernera le système du jeu qui est développé en Java. Nous avons séparé notre programme d'une telle façon, car nous voulions séparer toutes les tâches bas niveau concernant la 3D dans JNI pour avoir une puissance de calculs optimale, mais garder les avantages de Java pour pouvoir créer le système du jeu.

Le Moteur 3D en JNI

Où trouver les sources ?

Pour commencer si vous souhaitez accéder au code source du moteur 3D en JNI, il faut pour cela afficher les « Project Files » dans Android Studio et aller dans le dossier « app/src/main/jni ». Sinon les fichiers ne s'affichent pas, car nous n'utilisons pas la version expérimentale du compilateur JNI de Android Studio, en effet nous avons préféré modifier manuellement le fichier « build.gradle » pour que le moteur de compilation aille utiliser directement le NDK téléchargé séparément et ainsi permettre plus de possibilités comme l'utilisation de bibliothèques tiers (ce qui n'est pas possible pour le moment avec Android Studio).

Pourquoi utiliser JNI ?

Si nous avons choisi d'utiliser JNI, c'est parce que nous avions l'habitude d'utiliser la bibliothèque externe OpenGL. Cela signifie que nous connaissions déjà quelques fonctions et que nous voulions pouvoir nous en servir aussi dans ce projet. De plus cela permettrait de programmer en C/C++, qui est un langage qui a la particularité d'être performant et qui s'adapterait bien avec le moteur 3D.

L'objectif de ce moteur

Si nous avons crée ce moteur c'est parce que nous voulions avoir une bibliothèque facilitant l'utilisation de la 3D, en effet faire de la 3D implique souvent des calculs matriciels, de l'allocation de mémoire sur le GPU, etc. Cependant nous ne voulions pas mélanger ces calculs avec l'application en Java, car cela rendrait le code illisible et inutilisable. Ce moteur s'occupera donc de toutes ces tâches bas niveaux pour que nous n'ayons plus qu'a l'utiliser tout simplement pour le jeu via une interface. Nous avions d'ailleurs commencé le développement du moteur bien avant le début du projet, car celui-ci n'était pas lié au jeu directement.

Les bibliothèques externes utilisées

Le moteur pour être fonctionnel, a besoin de la SDL2 (pour utiliser SDL2_image), ainsi que de Assimp :

– Si nous utilisons la SDL2_image c'est pour pouvoir nous faciliter le chargement des images, en effet grâce à elle nous pouvons charger facilement des images avec différents formats (PNG, JPG, etc).

– La bibliothèque Assimp est une bibliothèque de chargement de modèles 3D, grâce à cette bibliothèque nous pouvons charger plusieurs formats de modèles (OBJ, 3DS, etc). Cependant pour ce projet nous ne chargeons pas n'utiliserons pas cette fonctionnalité, car nous avons préféré ne pas utiliser de modèles pour éviter d'avoir une application trop gourmande en taille.

Le mode de rendu

Le mode de rendu utilisé par le moteur est un rendu direct. Au départ nous voulions tout d'abord utiliser un rendu différé, cependant après quelques recherches nous nous sommes rendu compte que les smartphones n'étaient pour le moment pas assez puissant pour utiliser un tel rendu. Cela empêchera donc d'effectuer des opérations post-traitement sur l'image finale d'une scène.

Les Shaders en OpenGL ES 2.0

En OpenGL, il existe plusieurs versions des Shaders, les versions varient selon les périphériques, cependant nous voulions que le moteur puisse être utilisé sur le plus grand nombres de téléphones Android. Nous avons donc choisi d'utiliser la toute première version du langage GLSL à savoir la version 1.00.

L’application en JAVA

Interfaçage avec les classes JNI

Nous avons commencé le développement en Java, en créant les « Wrapper Class » associées aux classes du moteur 3D pour pouvoir les utiliser en Java directement par la suite dans le jeu.

Le système du jeu

Le système de jeu a été conçu pour que l’on puisse créer des niveaux facilement, et qu’il soit relativement ergonomique pour le joueur. En effet la 3D implique une certaine perspective pouvant entraîner quelques problèmes pour le joueur.

Les objets du type « Form »

Les objets du type « Form » sont les pièces du puzzle qui vont êtres utilisées pour les niveaux, il y a 5 formes de pièces différentes (ce qui explique les classes Form1, Form2…). Lorsque l’on instancie une pièce, il faut indiquer la couleur de celle-ci, sa position initiale, ses positions gagnantes ainsi que son orientation dans la scène.

Le choix de la couleur est extrêmement important et ne dois pas être la même qu’une autre pièce, car la couleur est utilisée pour pouvoir faire du « Color Picking ». Il s’agit d’une technique permettant de savoir ce que l’utilisateur sélectionne comme pièce simplement en regardant la couleur de la pièce sur laquelle il a cliqué. Ainsi si deux pièces ont la même couleur alors lorsque l’utilisateur sélectionnera l’une des deux pièces, ce sera toujours une des deux même pièce qui sera sélectionnée.

Concernant la position initiale il s’agit tout simplement de la position de la pièce au démarrage d’une partie. Ensuite une pièce peut avoir plusieurs positions gagnantes, c’est pour cela que le constructeur demande un tableau de positions gagnantes.

Les objets du type « Level »

Il s’agit tout simplement d’une classe qui gère un niveau entier, en effet lorsque l’on crée un « Level », on peut ensuite lui ajouter des « Form » pour ensuite le laisser gérer tout seul les interactions. C’est avec cette classe que l’on va savoir quand un niveau est terminé ou non.

Le gameplay

Concernant le gameplay il y a deux mécanismes que nous avons intégré pour en améliorer l'ergonomie :

– Tout d'abord les ombres, grâce aux ombres le joueur visualise mieux l'environnement et lui permet de mieux placer ces pièces, sans quoi il est beaucoup plus dur de s'y retrouver.

– Puis les ombres dites « gagnantes », il s'agit tout simplement des ombres des pièces qui sont dessinées en rouge sur leurs positions gagnantes. Cela a 2 avantages, le premier concerne le joueur, en effet ça lui permet de visualiser où est-ce qu'il doit résoudre le puzzle ainsi que sa forme, puis le deuxième concerne le développeur, car cela lui permet tout simplement de voir quel niveau il est en train de dessiner, de ne pas se préoccuper du placement de la zone gagnante et ainsi en faciliter le débogage.

Les préférences

Nous avons utilisé les préférences pour 2 types d'informations :

– La première utilisation concerne les settings, dans ce cas-là il s'agit seulement de l'activation de la musique. Ces préférences sont d'ailleurs utilisées avec une PreferenceActivity pour pouvoir les afficher plus facilement a l'écran.

– La deuxième utilisation concerne l'enregistrement des records des joueurs, cela permet de sauvegarder les scores même après la fermeture de l'application. Ces préférences ne sont d'ailleurs pas affichées directement, nous les utilisons surtout pour pouvoir enregistré des données d'une façon persistante. Les scores sont affichés manuellement.

La musique

Pour la musique nous avons donc utilisé une classe « MediaPlayer », au début nous ne savions pas très bien nous en servir et elle était nécessaire dans plusieurs objets différents, ce qui avait pour conséquence d'avoir un qui code n'était pas très lisible. Nous avons donc par la suite implémenté un singleton (« MyMusicManager ») accessible à tous, ce qui a permis de clarifier le code.

Organisation dans le travail

Mise en place d'un dépôt GIT

Pour pouvoir travailler facilement, nous avons tout d'abord mis en place un dépôt GIT (https://github.com/Mzartek/AndroidProject), cela nous a permis de nous partager et de travailler sur le code beaucoup plus facilement.

La répartition du travail

Concernant la répartition du travail celle-ci fut assez simple même si nous nous sommes beaucoup entraidés. L'un était donc chargé de travailler sur le moteur 3D tandis que l'autre était sur l'application Java.

Des rendez-vous hebdomadaires

Nous nous donnions souvent rendez-vous une journée de la semaine pour pouvoir faire un résumé du travail effectué, puis nous enchaînions sur une séance de programmation collective. C'est d'ailleurs dans ces moments-là que nous avancions le plus, car nous complétions nos idées et déboguions plus facilement.

Les difficultés rencontrées

Développement d’un moteur 3D long

Le développement du moteur fut extrêmement long, nous avions pourtant commencé à le faire assez tôt (début octobre), cependant la compilation des librairies tiers (pour chaque type de processeurs existant), les soucis de portabilités selon les périphériques (les shaders par exemple), et tous les petits problèmes avec le moteur de compilation « graddle » ont rendu le développement difficile.

Un système de jeu mal prévu

Nous avions conçu et implémenté un système de jeu qui avait un défaut a l’origine, en effet celui-ci était dans l’incapacité d’utiliser plusieurs fois une pièce de puzzle de la même forme et de même orientation pour une position gagnante. Une seule et unique position gagnante était donnée par pièce et une pièce en double avait donc plusieurs positions gagnantes. Nous nous en sommes rendu compte au moment de la conception des niveaux.

Diagrammes de classe

Généré avec Code Iris

Pour y voir plus claire nous vous conseillons de zoomer.

Diagramme de classe général

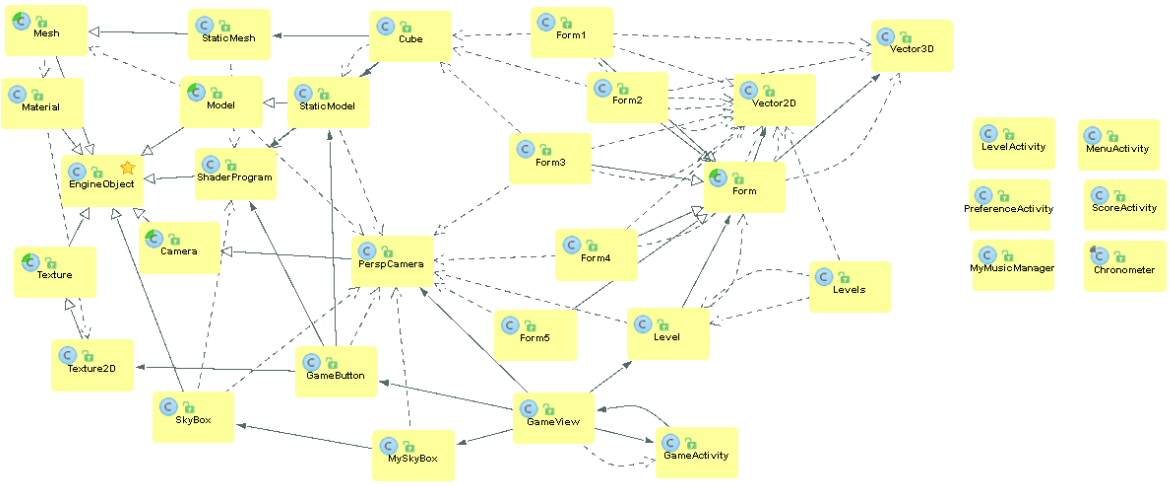


Diagramme de classe du package « engine »

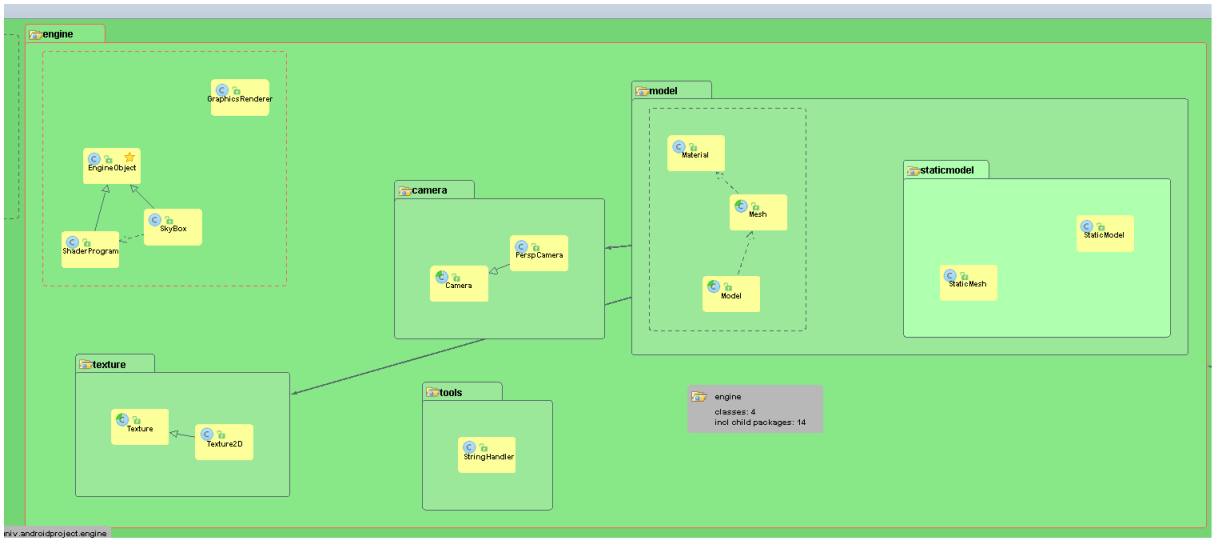
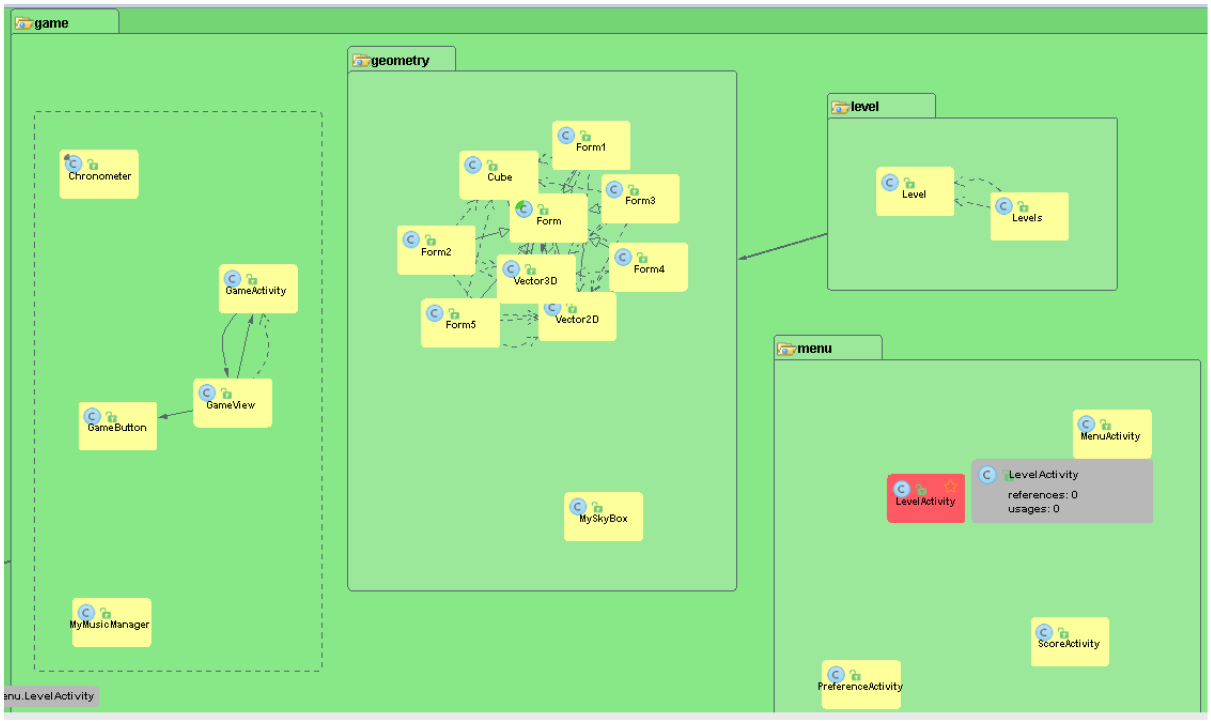


Diagramme de classe du package « game »



Conclusion

Notre but étant de faire ce casse-tête en 3D, le but est en partie atteint, cependant il reste quelques points qui pourraient être amélioré. Le premier concernait les contrôles, en effet même si ceux-ci sont relativement simples, nous avons remarqué qu'après avoir fait tester l'application à certains utilisateurs, que leurs premiers réflexes étaient d'effectuer du Drag And Drop. Le deuxième concerne la sauvegarde de l'état du jeu lorsque l'on quitte un niveau, ayant eu des problèmes avec le système de jeu, nous n'avons tout simplement pas eu le temps de nous intéresser à ce sujet.

Concernant les points positifs, nous avons une application qui fonctionne et qui est stable. La 3D, malgré sa simplicité, est tout a fait fonctionnel. Nous avons réussi à implémenter une interface de communication entre les classes C++ et Java avec JNI.