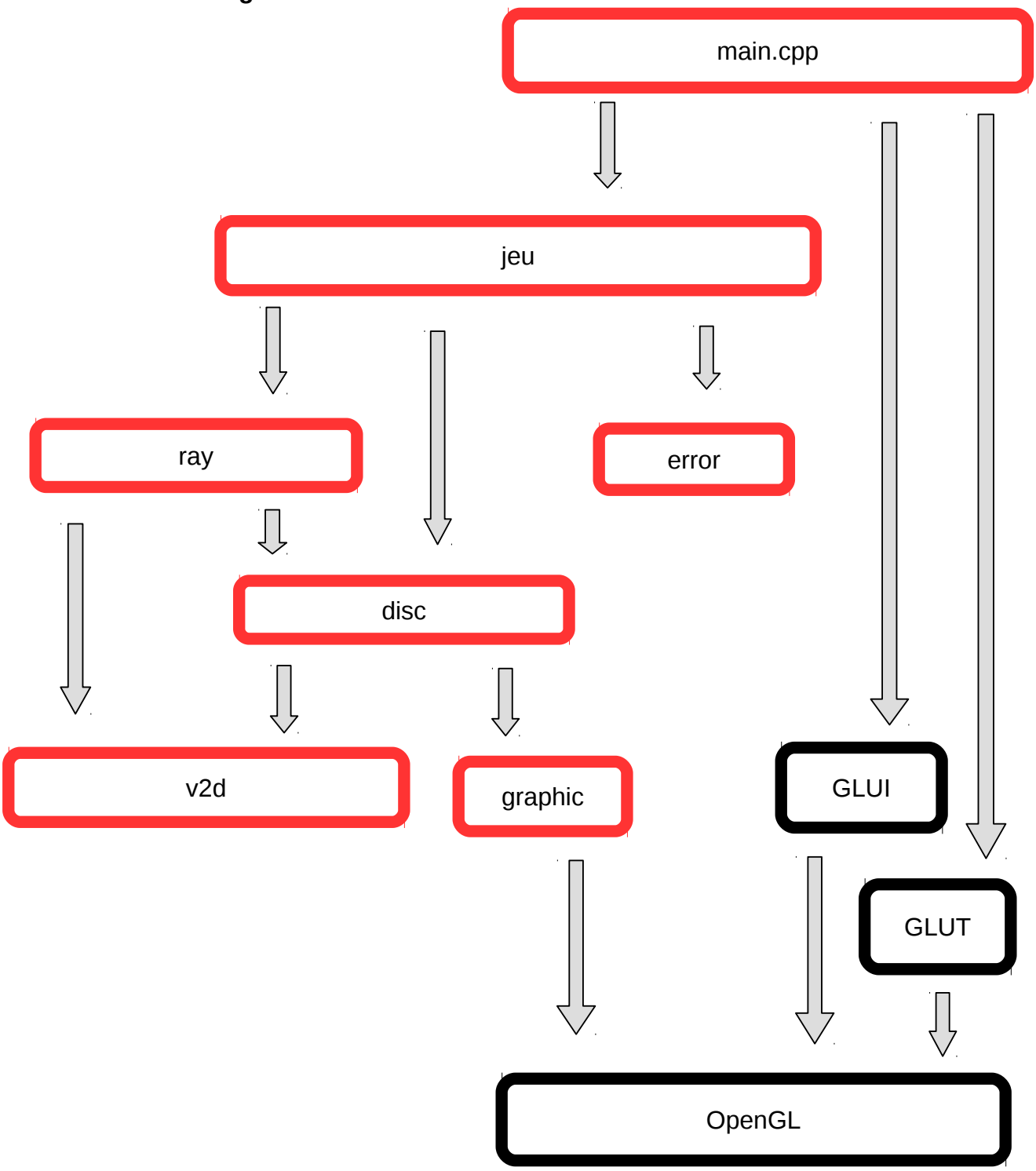
DUCCI Francesco Paolo **237672**
FELLEY Guillaume **234732**

Section MT, Bachelor semestre 2, semestre de printemps **2014**

Rapport Final

rendu 3 du 18 mai 2014

I. Architecture logicielle



Compléments et remises en question depuis le deuxième rendu

Après le deuxième rendu, les tâches de gestion du rayon et de destruction des disques, précédemment délégués aux modules **rendu1** et **rendu2**, ont été révisés pour s'adapter aux possibilités et aux limitations propres de la gestion dynamique de la mémoire. Ceci a impliqué avant tout la création d'un nouveau type de donnée : le type **RAY** décrit plus avant. En outre, un nouveau champ a été ajouté à la structure **DISQUE**, le champ **temp_destr**, qui indique si le disque a été temporairement détruit par le rayon, avant que la destruction de ce dernier soit validée, et définitivement éliminé de l'espace mémoire. La gestion du rayon ce fait par étapes : en premier dans le **main .cpp**, à l'aide de fonctions callbacks liée à des actions de la souris, puis des appelles des fonctions du module **jeu**. Le module **jeu** appelle des fonctions du module **ray** qui, elles-mêmes, appellent des fonctions du module **disc**.

Structuration finale des données

On a utilisé des listes chaînées pour les rayons et les disques vu leur nature fluide, a cause des fréquentes destructions / créations qu'ils subissent lors d'un match. Par contre, on a utilisé un tableau pour les joueurs, vu qu'il ne subissent pas de modifications, sauf dans leurs champs.

- le type **V2D** est un type de bas niveau qui gère les vecteurs dans le plan et peut être utilisée par programmes différents. C'est le seul type concret, et peut-être utilisée par les autres modules.
- Le type **PLAYER** contient les données des joueurs : le nom, le score, l'énergie et le nombre de disques. Le tableau qui contient les joueurs est déclarée comme variable globale dans jeu.c. C'est un type opaque propre au module jeu.
- Le type **DISC** gère les disques. Il contient les coordonnées du centre, la valeur (proportionnelle au rayon), le joueur auquel il appartient et son indice. Les deux dernières variables ont pour but de simplifier les tâches d'affichage et de detection des erreurs, en ayant seulement une liste chaînée. En outre il content un pointer de tipe *** DISC** et un champ qui indique qu'il est temporairement détruit par le rayon, et donc ignoré par la propagation suivante et affiché d'une couleur différente. C'est un type opaque, il est utilisé seulement par le module disc.
- le type **RAY** contient deux champs pour les coordonnées de départ et de fin d'un rayon de type **V2D** et un champ qui contient un pointer de type *** RAY,** pour créer une liste chaînée. C'est un type opaque.

Gestion des données et coût calcul

Lors de son appel, la fonction **ray_propagate**, exécute les actions suivantes :

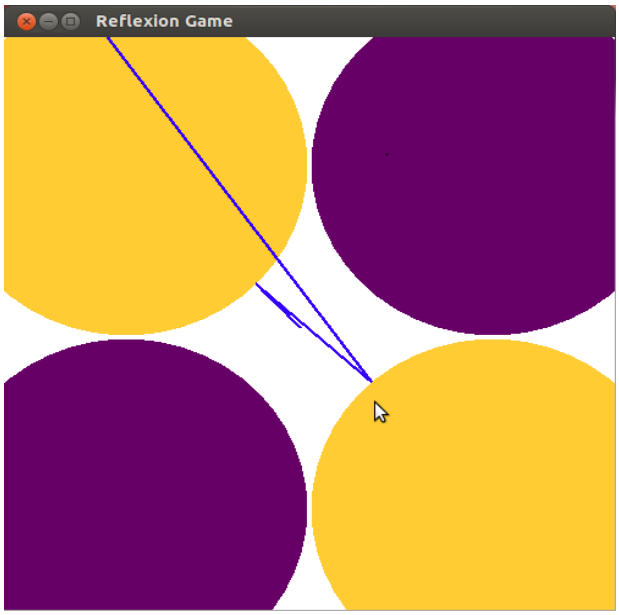
– Appel de la fonction **ray_delete**, qui détruit le rayon générée par son appel précédent (complexité R ou R = nombre de rebonds du rayon)
– Appel de la fonction **disc_state_reset**, qui parcourt la liste des disques pour réinitialiser leur **champ temp_destr** à zéro. (complexité N ou N = nombre de disques)
– Appel de la fonction **disc_intersect** qui trouve les coordonnées du disque le plus proche intersecté par le rayon en calculant les distances à l'aide de la fonction **disc_distance** et exécute les opérations nécessaires sur le disque aligné (s'il y en a un). (complexité N)
– Calcul du point d'intersection entre le rayon et le disque (complexité constante)

- Calcul du nouveau vecteur directeur généré par le rebond du rayon sur le bord du disque (complexité constante)
- Soustraction de l'énergie dépensée à l'énergie totale et disponible pour le tour (complexité constante)
- Génération d'un segment du rayon au moyen de la fonction **ray_creation** et dessin de ce deriner au moyen de la fonction **ray_trace**. (complexité constante).
- Affectation des nouvelles valeurs au nouveau point de départ du rayon (complexité constante).

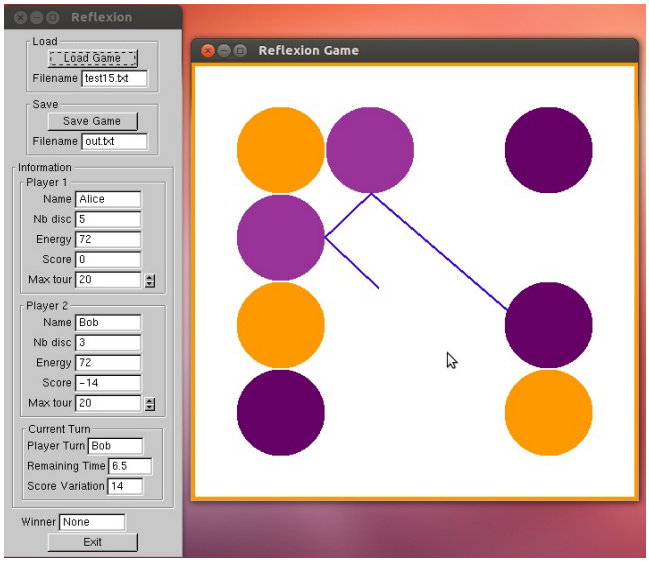
Toutes ces opérations sont éxecutée R fois, donc la complexité finale est d'ordre R*N

Approche sur l'exemple “test15.txt”

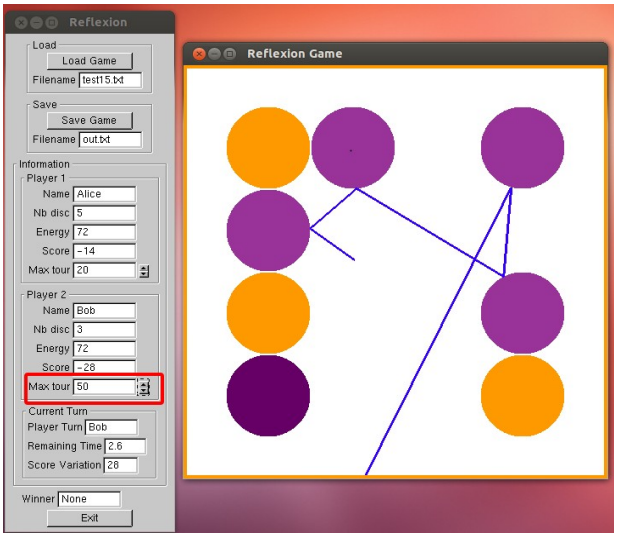
Si un disque est détruit, il est successivement ignoré par le rayon. Ceci ce fait avec la condition « *if (check->temp_destr == 0)* » dans la fonction **disc_intersected**.



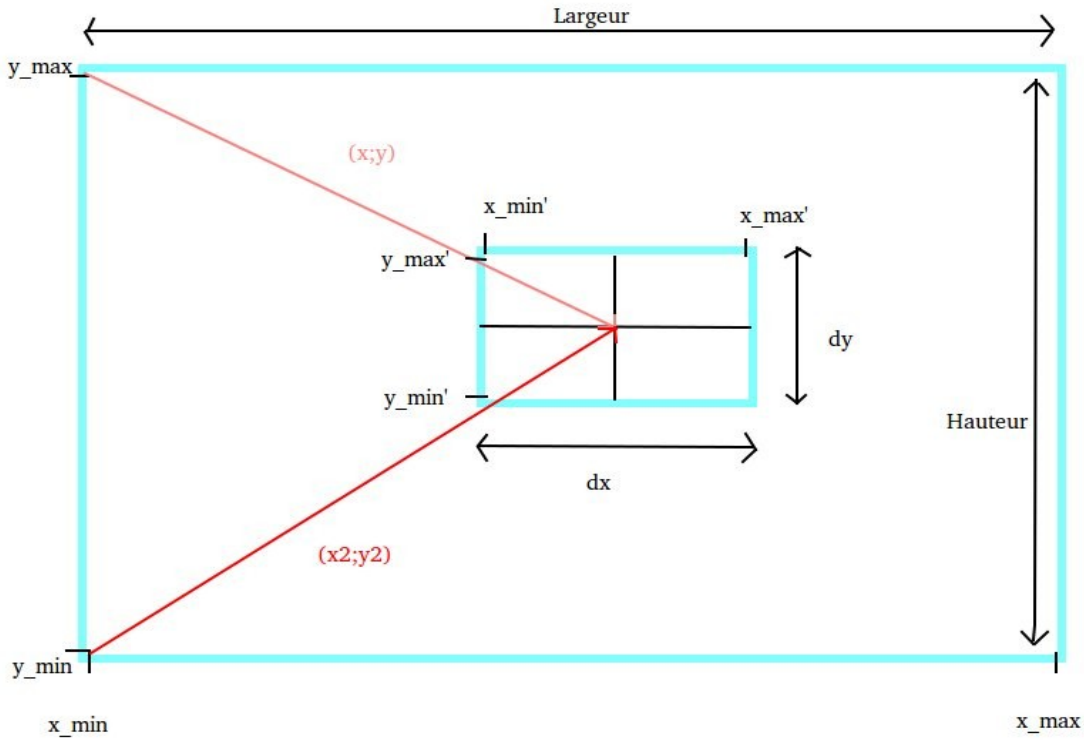
Le rayon rebondit sur les disque tant que l'énergie totale est inférieure a MAXTOUR valant par défaut 20



Nous avons fait le bonus MAXTOUR variable. Ainsi Bob peu choisir l'énergie qu'il veut dépenser en un tour, ici Bob a choisi comme valeur de MAXTOUR 50 il peu donc détruire plus de disques. Pour ce bonus, deux paramètres sont envoyés de main à jeu et finalement à ray : un paramètre **ene_tour** qui varie de 20 à la valeur choisie avec le spinner et un paramètre **max_ene** égal à l'énergie maximale du joueur. L'énergie dépensée doit être inférieure au deux.



Nous avons également fait le bonus zoom, le joueur peu cliquer droit pour faire un ou plusieurs zoom x3 et tirer avec plus de précision et dé-zoomer avec le clic du milieux. Pour mettre en œuvre ce bonus nous avons premièrement transformé les coordonnées de la souri (x;y) en coordonnées de la fenêtre (x2;y2), puis calculer les nouvelles largeur (dx=Largeur/3) et hauteur (dy=Hauteur/3) du nouvelle affichage. Et finalement par simple addition ou soustraction on calcul les nouveaux x_min,x_max,y_min,_max :



Remarque : On peu bien sur interagir et créer des rayon sans problème dans la nouvelle fenêtre

Méthodologie et conclusion

La division du travail a été organisée non pas par modules, mais par taches. Chaque étudiant a codé les fonctions nécessaires à l'accomplissement d'une tâche transversalement aux modules. En ligne générale l'étudiant FELLEY s'est occupé de tout ce qui concerne la partie graphique, et la création/destruction des listes chaînées. L'étudiant DUCCI s'est occupé de ce qui concerne le jeu en lui-même : gestion des fichiers, propagation du rayon et définition des types de données.
Les modules ont été testés selon l'approche bottom-up, en testant la validité de chaque module ou fonction, avant de l'implémenter.

Le bug le plus fréquent était celui de l'oubli de la réinitialisation des paramètres, auquel nous avons dû faire face plusieurs fois : (variable globale **error** dans le module jeu, variable ***distance** dans la fonction **disc_intersect** …). Le bug qui nous a posé le plus de problèmes, était une petite erreur de pointeurs sur structures dans la fonction **disc_destruction** qui causait un segmentation fault, lors du dessin des disques, si deux disques consécutifs avaient été détruits par le même rayon. Il a été résolu avec, premièrement l'identification de la fonction qui causait le crash du programme (**disc_drawing**), puis avec un contrôle scrupuleux de toutes les fonctions qui avaient manipulé la chaîne.

Pour ce qui concerne l'auto-évaluation de nôtre travail ; une bonne répartition des taches a permis une réalisation du projet assez efficace et rapide. Pour ce qui concerne l'évaluation de l'environnement, certaines indications étaient trop vagues, et leur précision à quelque jour du rendu nous ont forcé à faire des modifications à la dernière minute.