Journal de Développement

Création d'un Moteur de Jeu avec OpenFrameworks

Membres du groupe 8 :

- Alexandre BÉLISLE-HUARD
- Maxime BOURGEOIS
- Julien DE TOFFOLI
- Nicolas FRACHE
- Guillaume IMHOFF

Les difficultés rencontrées :

- Manque de communication :

En effet, un manque de communication lors de la phase de projet ainsi qu'un manquement précis d'organisation aura mené à un manque de rigueur, ce qui aura pu impacter certains membres du groupe. Ceci se passait également dans l'autre sens, où certain(s) membre(s) du groupe furent absent(s) tout au long de la phase de projet sans donner aucune nouvelle.

- Problème lors de l'application des forces :

Lors de l'application des forces nous avons eu un problème qui a pris une demi semaine à corriger. Nous avions pour chaque particule une liste de forces, dont la valeur change selon le contexte (ex: Ressort…). Or la non utilisation de pointeurs empêchait de modifier les valeurs des forces. Cependant, le passage à un tableau de pointeurs de forces faisait instantanément planter l'application. La solution à ce problème fut que le pointeur d'un objet Force local était certes valide, mais que le contenu de ce pointeur ne l'était plus car n'était pas conservé, il aura fallu non pas générer des objets Force, mais générer des pointeurs d'objets Force, permettant de rendre permanent ces objets.

- Identifier les particules une par une à la place de les identifier comme un « tout ».

Nos choix techniques :

Détection de type de collision :

Nous faisons un calcul entre deux particules, avec la soustraction de la particule « this » avec la particule « other », cette opération nous crée un vecteur « offset », on utilise la norme de « offset » pour avoir la distance.

Pour l'interpénétration, nous faisons un simple « if » sur la norme de « offset » et la somme des rayons des 2 particules. Dans le cas où la distance est inférieure à la somme de nos rayons, on obtient donc une collision nous appliquons une petite correction pour replacer les particules afin qu'elles ne se traversent pas.

Pour les contacts au repos, c'est le même raisonnement mais sans la correction qui est faîte.

Pour le type tige, la distance entre deux particules reste identique donc on a juste besoin d'avoir deux particules qui ne se rapprochent pas ou qui ne s'éloignent pas. On la modéliser par un fil rouge

Pour les types câbles, avec le calcule de la distance de deux particules. On s'intéresse cette fois-ci si la distance est supérieure à une « certaine valeur ». S'il elle est supérieur la collision est brisée. On a modélisé la détection par un fil vert.

Gestion des forces

Nous nous sommes basés sur les indications du cours pour la gestion de nos forces avec des classes représentant chaque force (ex : RessortForce, WaterForce, ElastForce etc…) qui héritent d'une classe mère « Force »

Notre manière de gérer les forces sur les particules diffère de celle du cours, car au lieu de faire un registre de force, nous avons fait en sorte que chaque Particule possède un tableau de pointeur de Forces, ce qui lui permet de gérer individuellement ses forces. Nous avons fait ce choix tout d'abord car en tant qu'objet il semblait logique que chaque particule contienne ses forces, mais également par simplicité à faire coïncider les différents concepts. Ce choix aura permis de simplement créer une force avant de l'appliquer à une particule, cependant cela nécessite donc de générer deux forces dans les cas prenant en compte deux particules, comme le ressort, car le ressort de A vers B et le ressort de B vers A sont différents de par leur sens, et également car la deuxième particule dans une force sert uniquement aux calculs et n'est pas modifiée.

Parmi les différentes forces, nous avons pu utiliser pour les forces de type ressort la loi de Hooke afin de calculer facilement le vecteur de force à appliquer à la particule. Nous avons également ajouté à notre outillage la force DampingForce, dont l'utilité aura été de rendre une particule stable, permettant à celle-ci de s'auto-stabiliser une fois qu'aucune autre force ne lui est appliquée; nous avons pu utiliser cela sur les particules du blob afin de s'assurer que les forces de déplacements entrées par l'utilisateur n'empêche pas de modifier la position du blob dû à une vélocité trop importante.

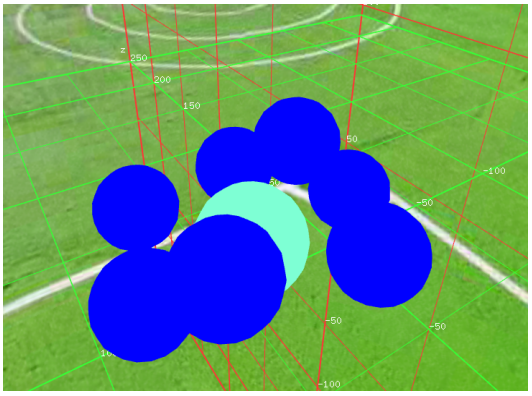
Mini jeu : Le Blob

L'objectif principal de cette phase avec ce que nous avons implémenté était la mise en place d'un "blob" c'est à dire de pouvoir déplacer un amas de particules restant les unes avec les autres, et de potentiellement séparer et regrouper ce genre de structure dans le cas de croisement de particules avec celle-ci.

Nous avons décidé d'approcher cette problématique en créant une particule de "Blob" pouvant s'attacher aux particules de type "Ball". Celle-ci possède un tableau de particules, et lors d'un ajout d'une particule à ce tableau, le blob insère une force ressort entre lui et cette particule, et applique un ressort entre celle-ci et toutes les autres particules du tableau. Cela permet de faire une structure assez plate au sol, mais qui suit le blob principal.

En plus des forces ressort entre les particules de la structure, le blob gère un peu différemment ses propres collisions, c'est-à-dire que lorsqu'il entre en contact avec une particule, il gère sa collision normalement, mais également vérifie si la particule n'est pas dans son tableau, auquel cas il l'ajoute à ses particules ce qui lui permet de grandir et d'ajouter davantage de particules à son réseau.

Dans le cas de la séparation des particules, nous avons fait en sorte de retirer les forces ressorts entre chaque particules ainsi qu'avec le blob, avant d'utiliser une force de type explosion afin de propulser légèrement les particules en dehors du champ d'action du blob.



Exemple de Blob

Explosion du blob lors de l'appui sur la touche Entrée