**Combien de particules de neige arrivez-vous à ajouter avant d’obtenir le premier message de lag dans la console?**

**1. Avant de modifier le programme :**

2’500

**2. Après avoir ajouté l’Object Pool :**

10’000

**3. Après avoir retiré les allocations inutiles :**

14’000

**4. Après avoir réduit la taille des objets :**

30'000, voir plus

**Q1 – Quelle est la taille (en bytes, avec sizeof()) des objets suivants si vous compilez le code en 64 bits?**

**sizeof(GameObject) :** 12400

**sizeof(Player) :** 12408

**sizeof(Snowflake) :** 12400

**sizeof(SnowflakeNoContact) :** 12400

**sizeof(SnowflakeRebound) :** 12408

**sizeof(SnowflakeSine) :** 12408

**Q2 – Vous allez avoir un problème si vous essayez d’ajouter différents types de Snowflake au tableau de Snowflake[].**

**Expliquez pourquoi ça ne peut pas fonctionner en C++, alors que ça fonctionnerait sans problème en C#.**

**Vous pouvez me faire un dessin si ça vous aide à expliquer :**

J’ai mie des cout dans mes updates et dans mes constructeurs de mes différents constructeurs et update de tous mes types de Snowflake.

Les constructeurs de tout mes types d’objet sont appelé une fois, parcontre les updates ne sont pas appelé pour aucun des autres types de Snowflake, sauf celui de la version de base Snowflake::Update().

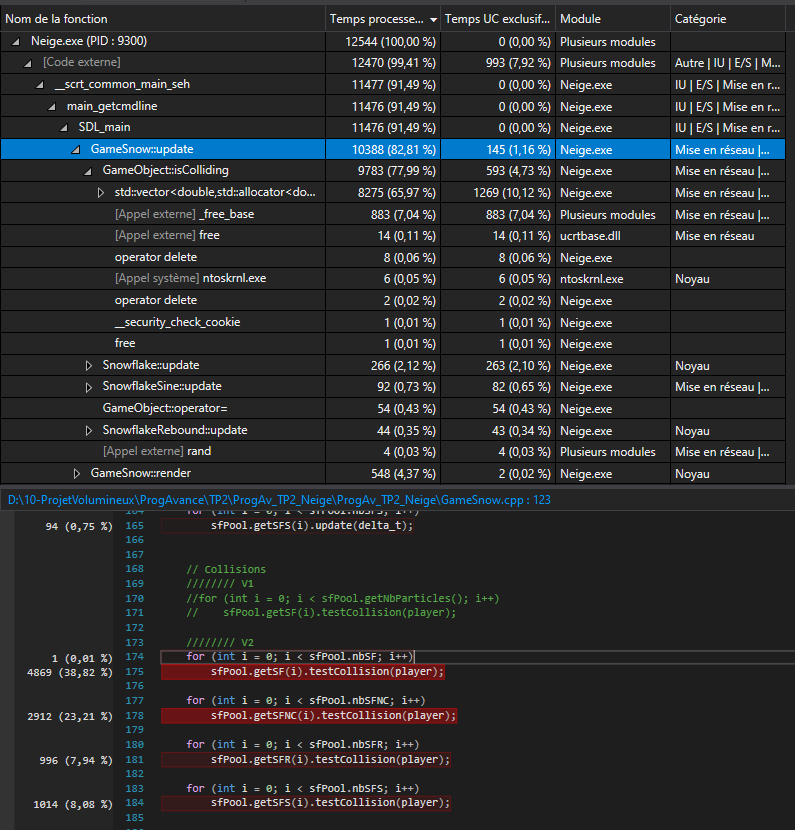
Donc on créer correctement une fois tout les types, mais quand on effectue nos updates ils redeviennent probablement des Snowflake de base.

Une chose très particulière est que tout cela se produisait en mêmes temps que je Render tous les différents types de particules Snowflake différentes.

En C# cela fonctionnerais car quand on travaille avec un objet, c’est toujours avec un pointeur.

Dans notre projet nous n’avons pas des pointeurs, donc quand on essaie de faire ce que nous faisions, nous copion une sous-classe dans une super-classe. Comme vue dans le cours où nous abordions le polymorphisme, quand nous faisons cela, la copie perd toutes les variables propres à la sous-classe. Amenant ainsi à de drôle de comportement à notre programme…

**Q3 – Donnez une capture d’écran du profileur qui montre que la section du code que vous avez modifiée valait la peine d’être optimisée.**



Une image contenant texte

Description générée automatiquement

La méthode GameSnow::Update prend 82% de mon temps d’execution.

Dans ce 82%, 38% et 23% sont les testCollision(player) en raison de la boucle qui appel la fonction isColliding qui ELLE fait des push\_back dans un vector. Cela revient à quelque chose de similaire à faire un new, donc si cela est lourd, créer toutes les particules avec des news se doit d’être lourd aussi.

**Q4 – Comment expliquez-vous le fait que retirer des tableaux qui n’étaient pas utilisés accélère le programme?**

**(Si jamais ça n’a pas d’impact chez vous, je vous confirme que ça me permet d’ajouter pas mal de particules sans lag sur mon ordinateur!)**

3. À cause de l’impact sur l’utilisation de la cache

Je ne mettrais pas ma main au feu, mais étant donné que les objets sont plus petits après la modification, on peut probablement profiter du principe de proximité des objets dans la cache. Étant donné qu’ils sont plus petit, la distance entre eux est plus court.

Aussi étant donné qu’on a de moins grosse donné à loader dans la cache, on va considérablement plus vite car on accède un qui est plus facile à chercher dans la ram pour mettre dans la cache.

**Q5 – Pourriez-vous ajouter l’optimisation de Grille d’accélération que vous aviez implantée dans le TP1? Est-ce que ça serait utile ici?**

**Ne l’implantez pas, répondez seulement théoriquement!**

Non. Le TP1 toutes les particules peuvent tous interagir entre les uns et les autres alors que le TP2 les particules interagissent principalement avec le joueur ou les délimitations de l’écran. Étant donné qu’on peut facilement savoir quand une particule interagit avec le joueur ou un des coins de l’écran, on a déjà un nombre de vérification de collision réduit qui est amplement gérable dans la situation actuelle.