Nguyen Thanh-Mai Christine Section Math Groupe n°55
Sekarski Samuel Section Math Groupe n°55

JOURNAL

Voici les grandes lignes de ce que nous avons fait hebdomadairement pour le projet. Nous avons aussi inclus nos états d'âme/commentaires comme suggéré dans le descriptif du projet, ceux-ci sont en italiques. *Nous vous souhaitons une bonne correction !*

Semaine 2 :

- ✓ Conception de la classe Vecteur, et premier codage de celle-ci
- ✓ Séparation de la classe en « Vecteur.h » et « Vecteur.cc », et élaboration du fichier testVecteur.cc. (Modularisation)
- ✓ Ecriture du Makefile
- ✓ Tous les tests réussis, aucun problème à déclarer
- ✓ Création du fichier « REPONSE » et réponse à la 1ere question
- ✓ Création du Journal et premières entrées.

Semaine 3 :

- ✓ Rien(on a fait la modularisation à la semaine 2 déjà)
- ✓ *A la place on vous a fait des petites « ASCII art » dans les fichiers c++ ^_^*

Semaine 4 :

- ✓ Révisions de la classe Vecteur : Surcharge des opérateurs +,-,*,^ en externe **à la place** des méthodes correspondantes pour faciliter la manipulation de vecteurs. Avec optimisation c++2011 : le 1er paramètre est passé par valeur, et c'est lui qui est retourné par la fonction, et le 2ème paramètre est passé par référence constante.
- ✓ Surcharge des opérateurs d'auto affectation +=,-=,*=,^= en interne, avec type de retour Classe&
- ✓ Surcharge de l'opérateur d'affichage <<, ainsi que des opérateurs de comparaison ==,!= **à partir** des méthodes dédiées existantes
- ✓ Modification de testVecteur.cc pour tester nos mises à jour
- ✓ Implémentation d'un constructeur explicite, ainsi qu'un constructeur par défaut pour le vecteur nul

Semaine 5 :

- ✓ *Huh… J'ai faim.*
- ✓ Création de la classe Particule et du fichier test correspondant
- ✓ On a doté la classe Particule de 3 constructeurs, dont un par défaut initialisant une particule d'hélium au repos à l'origine. (cf. CONCEPTION), les deux autres explicites (un avec des vecteurs, et l'autre avec des doubles). On a fait des accesseurs pour les deux attributs, mais un setter uniquement pour la vitesse. (On considère que la masse reste constante (pas d'enrichissement de particule ni de fusion/fission nucléaire) et la position est gérée par une méthode.) On a surchargé l'opérateur d'affichage.
- ✓ On a ajouté une méthode à la classe Vecteur qui renvoie le vecteur libre symétrique par rapport à un plan (comme c'est des vecteurs libre, on passe juste le vecteur normal au plan a la méthode). On a mis une méthode « mirroir » dans Particule qui fait la symétrie de la particule par rapport à un plan (on voulait faire une classe plan, mais finalement on a uniquement passée les vecteurs normal et position du plan a la méthode).
- ✓ Création de la classe enceinte : des getters pour les trois attributs, et des setters aussi (on imagine faire des parois mobiles), une méthode qui calcule le volume (qui pourrait être utile si on veut collecter des informations sur le système)

Semaine 6 :

- ✓ Nous avons ajouté à notre projet la possibilité d'avoir différents gaz (monoatomique et toujours parfait : hélium, néon, argon, krypton, xénon, radon), concrètement en créant des classes spécialisées de particules (héritage), qui s'affichent en mode texte.
- ✓ Création de la superclasse « Dessinable », qui est abstraite. Les particules sont dessinable, le système aussi, comme elles auront d'autres façons de se dessiner on a mis la méthode « dessine » de « Dessinable » en virtuelle pure. Comme on nous demande de ne pas redéfinir la méthode dessine dans « Particule », la classe « Particule » est devenu abstraite aussi (=> non distanciable). On a, par conséquent, gardé une ancienne version de la classe particule n'héritant pas de « dessinable » pour le fichier testParticule
- ✓ *Raaaah ! mais pourquoi Word me souligne dessinable en rouge -.- Comment ça s'écrit dessinable ??*
- ✓ Création de la classe Système, la classe qui va s'occuper de faire tourner le moteur de simulation : constructeurs demandés, méthodes ajouter/supprimer comme demandé, et un destructeur.
- ✓ Comme le système aura certainement beaucoup de particules, il serait peu judicieux d'en faire des copies, on a donc supprimé le constructeur de copie et l'opérateur d'affectation.
- ✓ *Euh…J' peine un peu à savoir ce qu'il faut mettre ici, (dans l'instance courante du journal xD) et ce qu'il faut plutôt mettre dans CONCEPTION… 😑*

Semaine 7 :

- ✓ Etant donné qu'il y avait la série noté de prog la semaine suivante, on a décidé de reporter cette étape du projet à la semaine suivante
- ✓ Par contre, on a vérifié que les fichiers conceptions et réponses aux questions était MAJ

Semaine 8 :

- ✓ *C'est bientôt les vacances !!! *feeling exited**
- ✓ On a fait plein de choses:
- ✓ Codage de la méthode de simulation (évolue), qui déplace toutes les particules du système d'un MRU, détermine s'il y a des collisions entre certaines particules, et le cas échéant « fait le choc », c'est-à-dire mettre à jour les vitesses des particules subissant le choc.
- ✓ Nous avons donc rajouté une méthode évolue à la classe Particule
- ✓ Ah et aussi on gère les rebonds contre les parois de l'enceinte, avec une méthode outil (private), qui est appelée dans la méthode évolue de Système
- ✓ On a dû mettre en place une façon de faire des tirages aléatoires
- ✓ Pour savoir si plusieurs particules se rencontrent, on fait un découpage de l'espace (échantillonnage)
- ✓ On a rajouté deux méthodes à la classe particule qui détermine si la particule en rencontre une autre passé en paramètre pour un certain epsilon passé aussi en paramètre. La première utilise un pavage cubique de l'espace, la deuxième un pavage sphérique
- ✓ Fichier de test pour voir si le truc marche ; pour cela, on utilise le fait qu'on ne peut pas vraiment faire de trucs aléatoire sur un ordi, c'est juste des formules complexes qui calcule à partir d'un nombre des autres nombre qui n'est pas facilement calculable => en connaissant le nombre de base, on peut prévoir exactement comment tout va se passer.
- ✓ *C'était un peu long à faire.*
- ✓ *J'ai l'impression de me répéter 3-4 fois (ici, conception, commentaire du code, et des fois réponse aux questions)… on s'excuse pour ça (je trouve qu'écrire est plus difficile que coder)*

Semaine 9 :

- ✓ *C'est les Vacances !!! \o/ enfin !! yay ☺*
- ✓ *Tous a Polymanga !! (c'était fait par l'epfl à la base, vous le saviez hein ?)*
- ✓ On a décidé se séparer nos fichiers en sous dossiers source(src), entête(include), executable (bin) objet (build), documentation(doc), tests(tests) ; on a modifié le makefile bien sûr, et lui on l'as laissé en dehors des sous dossiers (i.e. dans le dossier de du projet, ou à la racine du projet si vous préférez)
- ✓ On a suivi le tutoriel sur la partie graphique, celui de Mme Sam

- ✓ *Vous savez faire des moelleux au choc ?? Moi non, mais je connais de supers personnes qui peuvent en faire ! :3*
- ✓ *Bravo, tu m'as donné hyper faim avec tes moelleux au chocolat *q**
- ✓ *Joyeuses Pâques ! (enfin quand vous lirez ceci, ça ne sera plus Pâques… mais c'est l'intention qui compte !)*

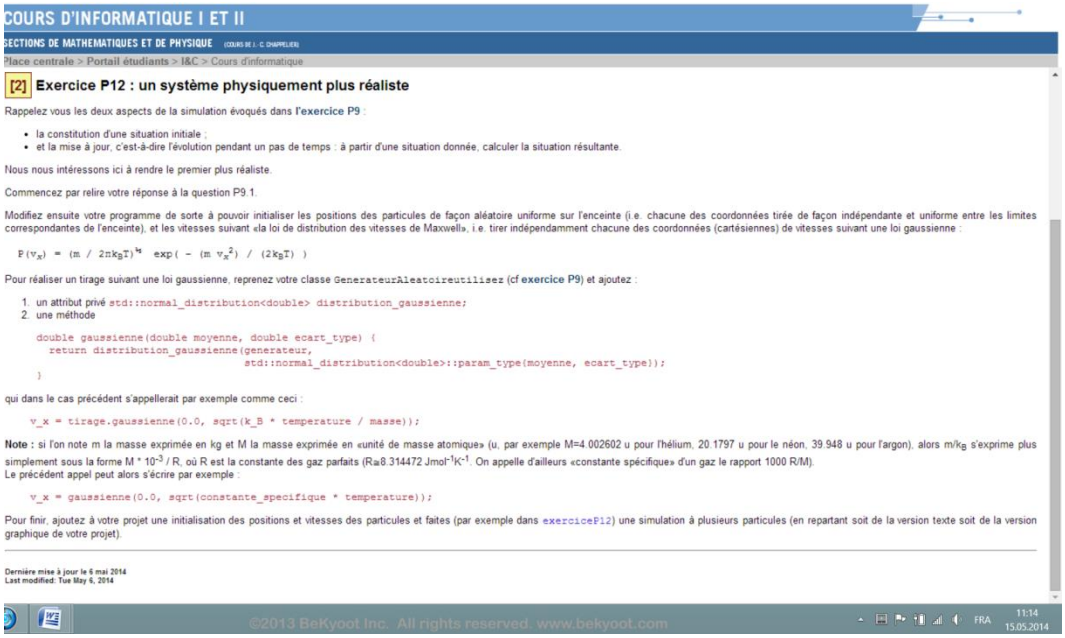
Semaine 10 :

- ✓ *Blopblopblop !*
- ✓ *Prout ! :D*
- ✓ On s'est attaqué à la partie graphique.
- ✓ Pas grand-chose à dire par rapport à P10. (cf. exerciceP10.cc)
- ✓ P11 : On a refait P9, mais graphiquement…. *Aaaaargh !!! Je me meurs (enfin… on crève… bwef)….*
- ✓ On a fait trois classes : GUI (pour l'application en général), FenetreGL (fenêtre destinée à accueillir un contexte OpenGl), et Vue_OpenGL (pour se charger de dessiner le système)
- ✓ Sans trop rentrer dans les détails, on a GUI qui possède une FenetreGL, qui elle possède une Vue_OpenGL, qui possède un Système. Quand Vue_OpenGL s'initialise, il initialise le système et lance le timer qui va appeler la méthode évolue de Système à chaque pas de temps du chronomètre (ici 20ms) et faire un rafraichissement de l'écran (i.e. : dessiner le système)
- ✓ On a aussi dû créer des classes spécialisées de particules (comme on l'avait fait pour la simulation texte : héritant de Particule) avec les commandes de dessine graphique (GlHelium, GlNeon, etc…). Puis pour la question P11, on a changé l'attribut du système « enceinte » en un pointeur sur une enceinte (pour avoir du polymorphisme sur Enceinte), et on a fait hérité Enceinte de Dessinable (une enceinte est déssinable selon nous) et nous avons créé les sous classes TXTEnceinte pour la simulation texte et GLEnceinte pour la simulation graphique. On a donc dû modifier la classe système pour prendre en compte le changement.
- ✓ On n'a pas jugé nécessaire de faire deux fichiers pour chaque sous classe de gaz (vu qu'il n'y a vraiment pas beaucoup de code, et ça ferait beaucoup de fichiers, en plus on imagine que des gens qui voudraient utiliser ces gaz, les utiliseraient probablement comme un tout, comme nous, au lieu de le faire séparément). On a aussi jugé qu'il ne servait à rien de séparer la déclaration de GUI, sa définition et l'implémentation du main (qui tiens sur une ligne) en trois fichiers, comme c'est très cours (une méthode). Du coup, on a laissé dans un même fichier (par exemple exerciceP11.cc). Peut-être dans l'avenir nous changerons si le besoin se présente.
- ✓ On a décidé de dessiner les sphères représentant les particules en « fils de fer », cela permet de mieux visualiser la 3D
- ✓ *Aussi la partie graphique lague, c'est nul >.<*

Semaine 11 :

- ✓ *Salut ! Comment ça va ? :D*

- ✓ On vous laisse relire la consigne, c'est ce qu'on a fait



COURS D'INFORMATIQUE I ET II

SECTIONS DE MATHEMATIQUES ET DE PHYSIQUE

Place centrale > Portail étudiants > I&C > Cours d'informatique

[2] Exercice P12 : un système physiquement plus réaliste

Rappelez vous les deux aspects de la simulation évoqués dans l'exercice P9 :

- la constitution d'une situation initiale ;
- et la mise à jour, c'est-à-dire l'évolution pendant un pas de temps : à partir d'une situation donnée, calculer la situation résultante.

Nous nous intéressons ici à rendre le premier plus réaliste.

Commencez par relire votre réponse à la question P9.1.

Modifiez ensuite votre programme de sorte à pouvoir initialiser les positions des particules de façon aléatoire uniforme sur l'enceinte (i.e. chacune des coordonnées tirée de façon indépendante et uniforme entre les limites correspondantes de l'enceinte), et les vitesses suivant «la loi de distribution des vitesses de Maxwell», i.e. tirer indépendamment chacune des coordonnées (cartésiennes) de vitesses suivant une loi gaussienne :

$$P(v_x) = (m / 2\pi k_B T)^{\frac{1}{2}} \exp(- (m v_x^2) / (2 k_B T))$$

Pour réaliser un tirage suivant une loi gaussienne, reprenez votre classe GenerateurAleatoireutilisez (cf exercice P9) et ajoutez :

1. un attribut privé `std::normal_distribution<double> distribution_gaussienne;`
2. une méthode

```
double gaussienne(double moyenne, double ecart_type) {
  return distribution_gaussienne(generateur,
                                 std::normal_distribution<double>::param_type(moyenne, ecart_type));
}
```

qui dans le cas précédent s'appellerait par exemple comme ceci :

```
v_x = tirage.gaussienne(0.0, sqrt(k_B * temperature / masse));
```

Note : si l'on note m la masse exprimée en kg et M la masse exprimée en «unité de masse atomique» (u, par exemple M=4.002602 u pour l'hélium, 20.1797 u pour le néon, 39.948 u pour l'argon), alors m/k_B s'exprime plus simplement sous la forme $M * 10^{-3} / R$, où R est la constante des gaz parfaits ($R \approx 8.314472\ Jmol^{-1}K^{-1}$. On appelle d'ailleurs «constante spécifique» d'un gaz le rapport 1000 R/M).
Le précédent appel peut alors s'écrire par exemple :

```
v_x = gaussienne(0.0, sqrt(constante_specifique * temperature));
```

Pour finir, ajoutez à votre projet une initialisation des positions et vitesses des particules et faites (par exemple dans exerciceP12) une simulation à plusieurs particules (en repartant soit de la version texte soit de la version graphique de votre projet).

Dernière mise à jour le 6 mai 2014
Last modified: Tue May 6, 2014

FRA 11:14 15.05.2014

- ✓ On plaisante, hein…
- ✓ On a relu le P9…. *Nom d'une baleine, on a suivi les instructions !!*
- ✓ On a fait la méthode « gaussienne » de GenerateurAléatoire comme désiré
- ✓ Ajout de deux méthodes publiques au Système : une pour tirer une vitesse aléatoire selon la méthode gaussienne, et une pour tirer aléatoirement une position à l'intérieur de l'enceinte de manière uniforme. On fait l'initialisation du Système dans Vue_OpengGL.
- ✓ On a testé avec notre version graphique : ça marche
- ✓ Nous avons profilé et utilisé « gprof » pour voir un peu quels méthodes prenait le plus de temps (par appelle et en fonction du nombre d'appels) et serait à améliorer
- ✓ Un truc qui est embêtant, c'est que le système ne prend pas en compte les tailles des particules (pour la simulation), parce qu'on a des particules d'hélium qui sont beaucoup plus petites que les particule de Radon par exemples, et il serait bien de pouvoir voir la différence, or, si on suit les instruction de P6 (la classe enceinte), on nous dit qu'une unité de double est égal à 0,1nm et en cherchant (cf. Wikipédia) les rayons approximatifs des particules susmentionnées, on arrive à des trucs assez gros (ordre de grandeur : 0,1nm de rayon) comparé à l'enceinte, du coup les particules « sortent » en partie de l'enceinte ☹ : sur une enceinte de 100x100x100 c'est moins visible, mais nous avons quand même décidé de diviser les rayon par 4. (Une autre solution aurait été de ne dessiner que le centre des particules, par exemple avec des points, mais c'est moins classe)
- ✓ Sinon… on a également fait le P13 (particules tracées) de la semaine prochaine !!! *HAHAHAHA !!!*
- ✓ Création d'une superclasse abstraite « Tracable » avec un tableau de Vecteurs (pour garder en mémoire les positions passées) avec une méthode pour ajouter un position à la collection et une méthode trace, qui trace la trace. *Woaaaah ! Trois fois le mot « trace » !*
- ✓ Ooh ! Et on s'est amusé à faire la partie avancée avec le « deque ».

- ✓ *C'est les 70 ans de George Lucas cette semaine ! Du coup pour lui rendre hommage, on a fait des « crédits » pour notre projet, de la forme du célèbre « text crawl » qu'il y a au début de chaque Star Wars :*

- ✓ On a créé une classe « Lettre » et des sous classes héritant de « Lettre », pour les 26 lettres de l'alphabet, chacune avec sa propre méthode dessine redéfinie (on va utiliser du polymorphisme)

- ✓ On a créé une classe « Mot » avec une collection hétérogène de « Lettre », une classe « Text » avec une collection hétérogène de « Mot » ces classes ont des méthodes pour ajouter les éléments de leurs collections hétérogènes.

- ✓ Et finalement il y a une classe qui a un contexte OpenGl pour faire défilé les crédits.

- ✓ *Bon vu que c'est un bonus a notre projet, le code n'est surement pas très bon, donc regardez plutôt ce que ça donne que le code lui-même ^^' si on avait eu plus de temps on aurait pu revoir tout ça et le faire correctement ^^*

- ✓ Il y avait un problème de segmentation fault lorsqu'on fermait la fenetreGL de la simulation ou des crédits sans stopper le timer. Nous avons résolu le problème en gérant nous-même leur fermeture.

Semaine 12 :

- ✓ On a fait le P14 !!!! *Youpiiiiiiiiiiiie ! \o/*

- ✓ On s'est décidé de ne pas faire de classe « cases », et nous avons décidé de faire des cases un attribut du Système et non pas de l'enceinte. (Notre raisonnement est que les cases ont forcément des particules (une liste de particule en fait), mais qu'une enceinte pourrait être utilisé pour autre chose que la simulation d'un gaz, en clair on ne veut pas que l'enceinte possède des particules, même indirectement)

- ✓ Nous avons fait des typdefs pour simplifier l'écriture des cases (qui sont un tableau tridimensionnel de liste de particule, qu'on a en premier implémenté avec un vector, puis avec une list)

- ✓ Ajout d'une méthode pour remplir les cases, appelée à chaque évolution et ajout d'une nouvelle méthode pour gérer les chocs (je vous passe les détailles d'implémentation)

- ✓ On a fait un exerciceP14.cc pour tester cette nouvelle version de simulation

- ✓ Et ça fonctionne avec 1000 particules (même plus)!

- ✓ On a testé avec 10'000 particules, et ça fonctionnait encore, mais de manière extrêmement saccadée (il y avait seulement 1 rafraichissement de l'écran toutes les 1-2 secondes environ)

- ✓ On a retesté avec « gprof » pour voir la différence sur le temps par méthode

- ✓ Il ne reste plus qu'à faire quelques petites améliorations… (les extensions)

- ✓ On a fait une interface graphique : une fenêtre qui permet à l'utilisateur de préciser le nombre de particules de chaque gaz qu'il veut initialiser dans le système fait avec des sliders, qui se bloque si un nombre max de particules est atteint. Avec deux boutons : un pour lancer la simulation, ainsi qu'un autre pour lancer les crédits.

- ✓ Il y a un comportement non voulu que je vais essayer de régler : si la simulation est lancée, et l'utilisateur appuie à nouveau sur « lancer la simulation », une 2ème fenêtre s'ouvre avec une autre simulation dedans ^^
- ✓ *Hum… Il y a un certain nombre d'erreur dans le tutoriel de Mme Sam…heureusement qu'en réfléchissant un peu, et avec de la documentation, on s'en sort, mais il faudrait peut-être les corriger pour les futurs lecteurs*

Semaine 13 :

- ✓ *Aaaaaaaahh !!! Maudit examen !!! **On n'a pas eu assez de temps** ! D'ailleurs, après ce massacre, nos cerveaux ont fondu… ._.*
- ✓ Création d'une fenêtre a option, où on peut choisir certaines modifications à faire subir au système : rotation de l'enceinte et zoom (en plus de pouvoir les faire graphiquement), accélérer ou ralentir le temps, modifier la taille de l'enceinte. On a aussi fait l'extension de placer la caméra sur une particule
- ✓ Relecture du projet en entier, vérification que tous les fichiers tests fonctionnent, finalisation du journal, de la conception et des réponses aux questions.
- ✓ Création du README, et de NOMS
- ✓ Comme les rendus ne doivent pas être en format propriétaire fermé, on a converti CONCEPTION, JOURNAL, REPONSES, README en Pdf
- ✓ Résolution d'un certain nombre de bug. Correction du problème de rebonds contre les parois lors d'enceinte petite et pas de temps grand.
- ✓ Résolution des segmentation fault a la fermeture des fenêtreGL.
- ✓ *Dommage qu'on s'est fait un peu prendre par le temps à la fin… on aurait voulu faire plus d'extension, et améliorer ce qu'on avait déjà fait… enfin tant pis vous allez devoir vous contenter de notre projet comme il est.*