Architecture logicielle et description de l'implémentation :

Nous avons gardé la même architecture logicielle que celle de la donnée. Nous avons réutilisé le rendu public 1.

Pour la structuration des données des robots nous avons enregistré les données suivantes : le numéro des robots selon leur ordre d'apparition dans le fichier de lecture, leur position dans l'espace, l'angle que fait l'axe Xr du robot avec l'axe X du repère monde (le sens dans lequel pointe le robot), l'adresse du robot suivant dans la liste chaînée, la position dans l'espace de la particule cible du robot et son état, c'est-à-dire s'il est en mode manuel ou automatique. Pour les particules, nous avons leur numéro selon leur ordre d'apparition dans le fichier, leur énergie, leur rayon, leur position dans l'espace, l'adresse de la particule suivante dans la liste chaînée et le nombre de robot qui ont cette particule pour cible.

Concernant l'algorithme de coordination, nous avons procédé de la manière suivante : tout d'abord, dans le module particule, nous créons un tableau qui contient le numéro des particules triées selon leur rayon, dans l'ordre décroissant. Ensuite, nous passons ce tableau au module robot qui le parcourt à partir du numéro de la plus grande particule jusqu'à celui de la plus petite. Pour chaque numéro, il va chercher l'adresse de la particule correspondante et sa chercher la position dans l'espace. Puis, pour chaque particule, il compare parmi tous les robots lequel est le plus proche (en termes de distance translation et de rotation) et lui associe la particule en question. Après cela, de retour dans le module particule, nous allons signaler qu'il y a un robot qui pointe sur cette particule. Pour finir, dans simulation on teste s'il y a eu élimination ou décomposition d'une particule et si c'est le cas on appelle la mise à jour de l'algorithme de coordination.

Nous avons opté pour cette approche, car étant donné que le module particule est inclut dans le module robot, nous pensons que la plus grande partie de l'algorithme de coordination devrait se faire dans ce dernier. De plus, dans le module simulation nous voulons rester les plus abstraits possibles, c'est pourquoi nous ne faisons qu'évaluer les conditions nécessaires pour la mise à jour de la coordination et appeler celle-ci, si au moins l'une d'elles est remplie.

Coût calcul et mémoire pour un pas :

	Coût calcul	Coût mémoire
Décomposition		
Boucle avec toutes les particules	NbParticules	0
Toutes les particules se décomposent	0	Nb_Particules* 4- Nb_Particules = 3*Nb_Particules
Mise à jour de la coordination		
Trier les particules selon leur rayon	NbPartcules	0

Associer le robot le plus proche	min(NbParticules,NbRobots)	0
Trouver la particule correspond au numéro donné	NbParticules	0
Trouver le robot le plus proche	NbRobots	0
Donner une particule comme cible à un robot	NbRobots	0

Total provisoire. Coût calcul = O(min(NbParticules,NbRobots)*(NbParticules+2NbRobots)).

Coût mémoire = O(NbParticules).

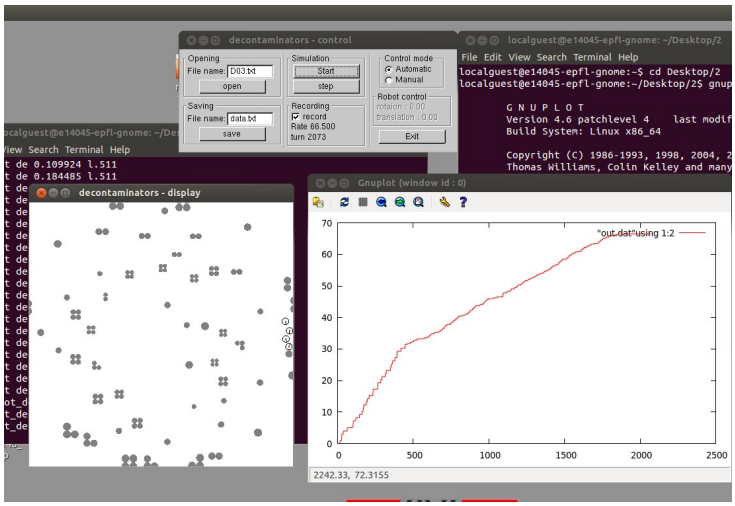
Méthodologie et conclusion :

Tout le monde a travaillé sur tous les modules mais pour effectuer différentes tâches. Jose s'est chargé de l'algorithme de coordination, de la décomposition des particules, des algorithmes concernant la lecture des fichiers et l'analyse des erreurs, et de l'affichage OPEN-GL. Damian s'est chargé de la partie GLUI et GLUT : création de la fenêtre et gestion des options de celle-ci. Il s'est aussi chargé de debuggé et tester le programme et de la gestion du développement de la simulation. Selina s'est occupée de tout ce qui concerne le déplacement des robots, de l'algorithme du taux de décontamination, du module draw, chargé du dessin OPEN-GL. Tous deux se sont se sont occupés des algorithmes de collision entre les particules et les robots, et de l'élimination des particules. Par ailleurs, nous avons réutilisé le rendu public 1, parce que nous voulions éliminer tout risque d'erreur basique, dans ce module de base que nous utilisons tout le long du projet.

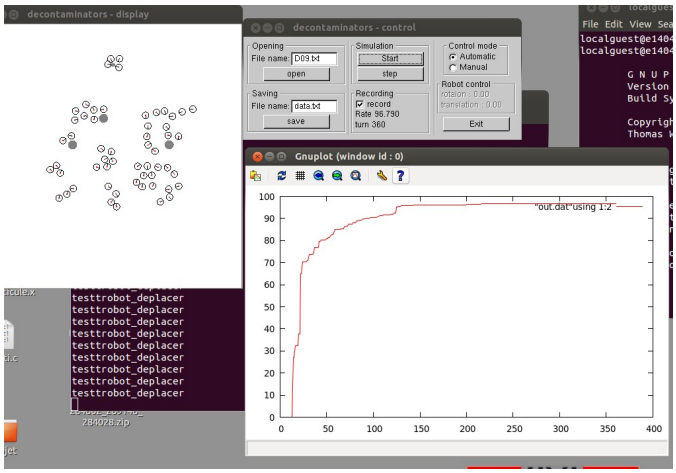
Pour Jose, le bug le plus problématique était en rapport avec l'algorithme de coordination. C'était un problème entre la gestion des différentes boucles imbriquées et les entités à ignorer. Pour Damian, c'était le problème de la gestion des fonctions GLUT, qui ne s'appelaient qu'au moment où on quitte le programme. Pour Selina, le bug le plus courant était les segmentations faults.

Nous avons divisé les tâches sans faire attention aux modules impliqués. Nous nous sommes repartis les points les plus importants (decomposition, déplacement, ...), et nous mettions en commun les différentes parties sur Github. Nous avons testé certaines fonctions dans un autre fichier, mais la plupart directement dans le programme principal, avec l'utilisation des printf.

Avec le recul, nous pensons que nous aurions pu documenter nos fonctions au fur et à mesure.



L'image ci-dessus montre le graphe après l'éxécution du test D03.txt. On remarque, que la simulation ne se termine pas totalement, en raison du blocage des robots.



De même, pour la simulation D09.txt, on note un problème de blocage des robots autour des particules restantes.