Alexandre LEAFE 274948
Xavier NAL 288275
17 Mai 2018

Projet Informatique Rapport : DECONTAMINATORS

Le but de ce projet est de simuler la décontamination d'un site irradié de particules par des robots. Les robots ont des mouvements non-holonomes et doivent nettoyer le site le plus rapidement possible en s'attaquant prioritairement aux particules de plus gros rayon.

I: Architecture logicielle et description de l'implémentation:

Pour le projet, nous n'avons pas changée l'architecture logicielle telle que présentée dans la donnée (cf. Moodle). Nous avons néanmoins utilisé la dépendance possible entre robot et particule.
Nous n'avons pas utilisé les rendus publics mis à disposition.

Afin de stocker les données, nous avons choisi d'utiliser un tableau de structure dynamique pour les robots et une liste chaînée pour les particules.
Comme le nombre de robots ne varie pas au cours de la simulation, nous avons préféré utiliser un tableau, plus simple d'accès pour les modifications des données. En revanche la liste chaînée nous apparaît plus adapté pour le stockage des particules. Car elle nous permet d'ajouter et enlever des particules de la liste n'importe quand et n'importe où dans la liste sans devoir recréer la liste chainée entièrement.

Pour programmer une mise à jour du monde nous avons réparti les tâches selon les trois modules : simulation, robot, particule. Pour la mise à jour, le module simulation appelle la fonction simulation update qui sépare les fonctions qui s'occupent de la décomposition des particules, et les fonctions de la mise à jour des robots, incluant le nettoyage des particules par les robots.
Le module robot pour une mise à jour s'organise autour d'une boucle « while() ». Cette boucle gère pour chaque robot: la coordination de ce robot en lui donnant une particule cible, la mise à jour des déplacements et les collisions de type robot-robot (celles si sont gérées de sorte à ne pas faire de blocages) et robot-particule (avec correction de déplacement et test de décomposition de particules) pour ce robot.
Pour les fonctions dans le module robot qui s'occupent de la coordination des robots et de la collision entre particule-robot, nous avons besoins d'informations du module particule. Mais afin de respecter le type opaque des modules robots et particules, nous avons dû prendre les informations sur les particules à l'aide d'appels de fonction qui envoient les valeurs dont nous avons besoins.
Ainsi, le module particule est appelé par le module simulation pour la décomposition aléatoire des particules, et par le module robot pour obtenir certaines informations (nombre de particules et informations sur les particules) et pour le nettoyage des particules.

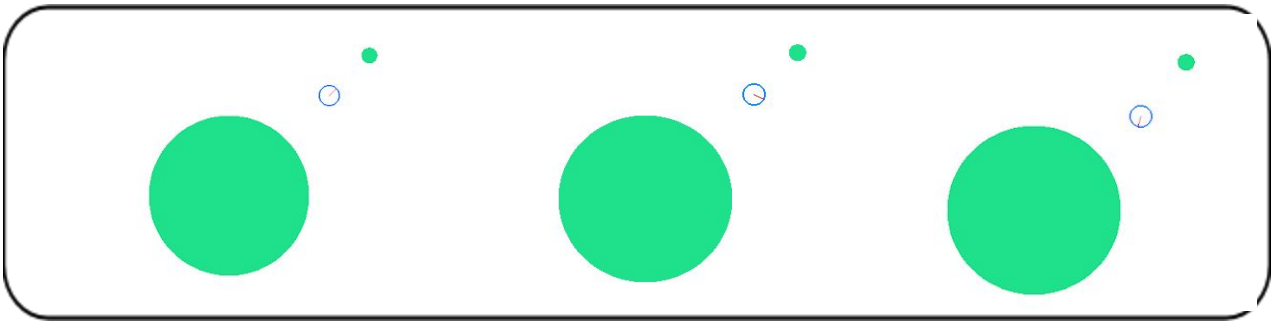
Concernant la stratégie adoptée par les robots pour éliminer les particules, nous avons choisis d'attribuer à chaque particule un score (une note sur 10) calculé en fonction de la distance de celle-ci avec le robot (distance de translation et écart-angulaire) et du rayon de la particule.
Ainsi, nos robots mettront la priorité sur la décontamination d'une particule dans leur distance vitale (distance ≤ 1.5*R_ROBOT), et autrement cherchent celle avec la meilleure note.
Afin de mettre la priorité sur les particules proches, nous avons pondéré le maximum de la note en fonction de la distance (distance vitale, score max 10; distance proche, score max 9, etc). De plus, pour chaque catégorie de distance, la note totale dépendant des pondérations des scores lié à la distance et au rayon, nous modifions ces différents coefficients de pondérations pour privilégier la distance ou le rayon.
Comme nous avons pensé que nous devions trouver une stratégie de décomposition optimale pour ce projet, cette idée nous est apparue et s'avère plus efficace que le programme de démonstration par exemple (environ 800 turns de moins sur le test D03).

Pour le coût calcul, notre estimation en fonction du terme dominant est O(nb_robot*nb_particules²) (pour chaque robot, on parcourt toute la liste des particules pour trouver celle qui a le meilleur score, et une fonction du module particule effectue aussi une recherche dans toute la liste de particule pour renvoyer les informations de celles-ci).
Pour le coût mémoire, celui-ci correspond au coût de stockage des structures de données après la lecture d'un fichier, ce qui correspond à: O(nb_robot+ nb_particules).
Ci-dessous nos structures pour robots et particules respectivement:

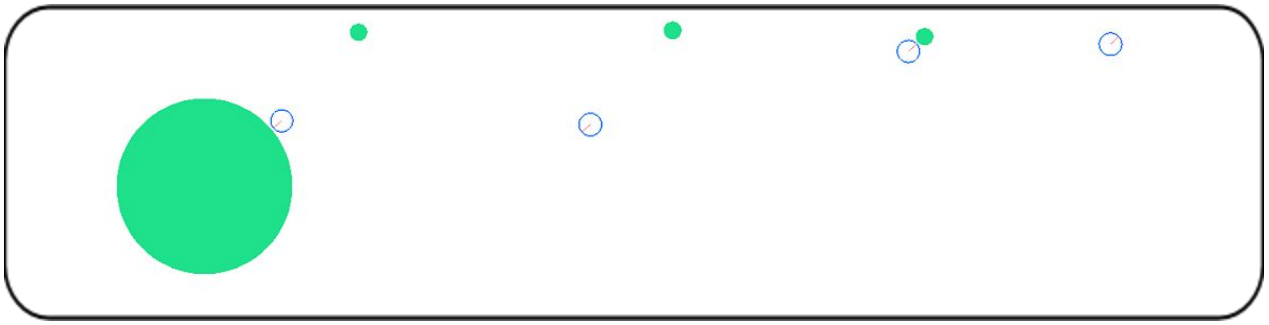
```
struct ROBOT_POSITION
{
    S2D centre;
    double alpha;
    C2D objectif;
    int col_part;
    int mode;
    double vrot;
    double vtran;
};
```

```
struct PARTICULE
{
    int nombre;
    double energie;
    C2D particule;
    PARTICULE *suivant;
};
```

II : Illustrations du fonctionnement d'une simulation :

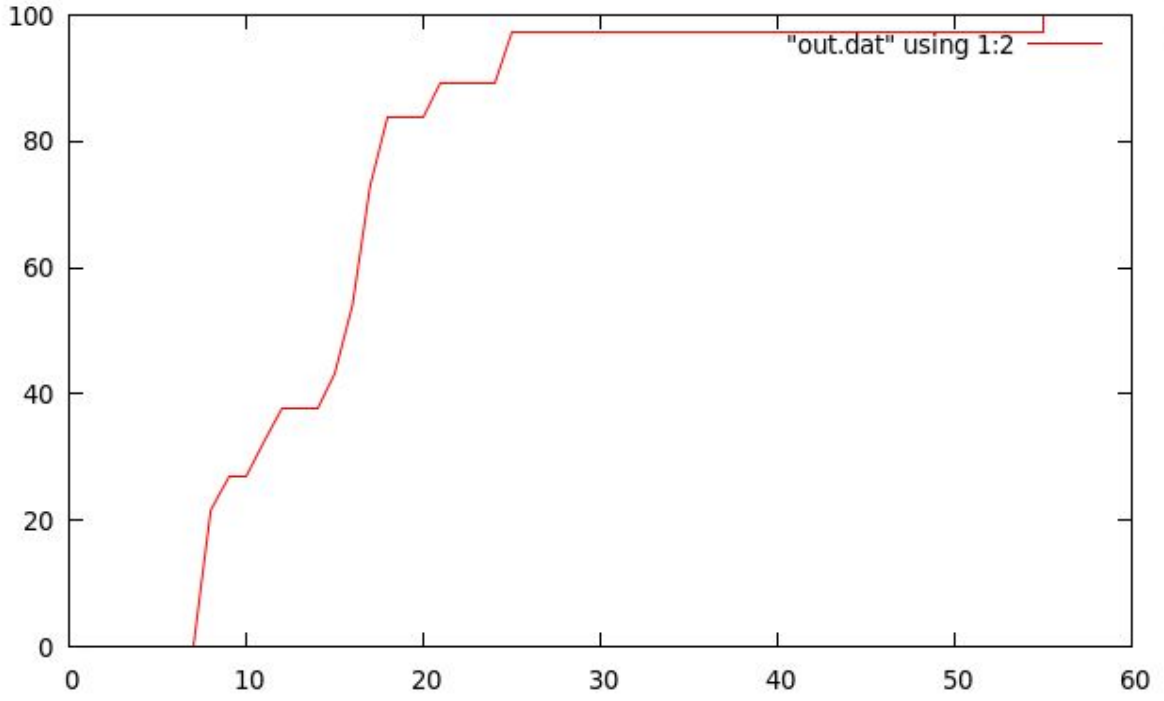


Sur ces trois premières images, nous voyons tout d'abord la situation initiale du test D06. Puis, après avoir attribué un score à chaque particule, le robot choisit de se diriger vers celle au plus gros rayon, et commence sa rotation. Enfin, après avoir dépassé un écart angulaire de π/4, le robot commence un mouvement de rotation et translation vers sa particule cible.



Puis, la prochaine image représente la particule juste avant de nettoyer sa particule cible (avant collision), suivie de la simulation avec particule nettoyée. Enfin, le robot se dirige vers la dernière particule pour réitérer l'opération, avant de s'arrêter (état final).

La simulation enregistrée du test D09 donne l'affichage gnuplot suivant:



Commentaires: *Le nombre de cycle pour atteindre la dernière particule est dû à nos corrections de déplacements faisant soit s'immobiliser soit reculer un robot qui va entrer en collision avec un autre robot (étant donné le nombre de robots dans le test D09, ce résultat semble tout à fait convenable).*

III: Méthodologie:

Concernant la répartition des tâches, Xavier s'est occupé de la lecture des données, test des erreurs, déplacement des robots durant la simulation et de la gestion des différents types de collisions (ainsi que les stratégies à adopter en conséquence, le nettoyage des particules par exemple).
D'autre part, Alexandre s'est occupé de la partie graphique (interface GLUI, dessin OpenGl, fonctionnements des boutons), de la décomposition aléatoire des particules et de leur algorithme de score.

Nous avons donc tous les deux participé à l'écriture de tous les modules, ce qui a d'ailleurs permit une certaine efficacité dans le groupe. De plus, malgré quelques problèmes occasionnels de communication, la gestion de groupe était très bonne et ce fut un projet d'équipe très agréable. Avec du recul, nous aurions sûrement opté pour la même stratégie de groupe.

Dans ce projet nous nous sommes tout d'abord répartis des tâches claires pour chacun, puis avons avancé chacun de notre côté, souvent sur des modules différents, ce qui permettait des échanges faciles de code entre nous. Nous testions ainsi chacun nos parties au fur et à mesure de leur écriture, avant de finalement les mettre en commun et de débugger les parties réunies.
Le plus gros problème rencontré au cours du projet furent des "Segmentation Fault", rencontrés à cause des problèmes de free() et de liste mal chaînées. Nous avons résolus ces soucis à l'aide de nombreux printf() et de réflexion, mais nous n'avons jamais eu de problèmes majeurs nous ayant réellement longtemps bloqué dans notre avancée.

Nous avons apprécié l'opportunité de pouvoir utiliser les rendus publics, cependant nous ne les avons que peu utilisés (pour deux fonctions du module utilitaire seulement), favorisant donc notre propre savoir-faire.

IV: Conclusion:

Pour conclure, ce projet a été très enrichissant pour nous deux. Nous avons appris à être réguliers sur un travail de groupe pour gérer les différentes échéances, à s'adapter au code de l'autre, et à diviser le problème principal en sous-problèmes. Nous avons apprécié le forum, qui nous a été utile grâce aux réponses très rapides, et la présence des assistants et du professeur aux heures de soutien ou pour le debugging.