Rendu Final

Aspirator 2, un monde nouveau

Pour le rendu final, j'ai pris le code Gilles Vetsch.

La raison qui m'a poussé a faire cette insertion, est que j'ai pu rendre mon rendu intermédiaire un peu en retard, du a des circonstances spéciales, et qu'il me fallait un code correct pour continuer mon projet.
J'ai donc cherché un code bien noté qui s'adapte facilement à ma façon d'avoir résolu le problème et qui m'a donc permis d'attaquer la partie graphique du projet plus rapidement. J'ai ainsi réutilisé quasi toutes les fonctions permettant de lire le fichier, et j'y ai inséré les éléments dont j'avais besoin pour créer l'affichage graphique. C'est le rendu intermédiaire de Gilles Vetsch.

Je n'ai quasi pas modifié l'architecture logicielle. J'ai uniquement ajouté un module «parcours» qui gère la mémorisation et l'affichage du parcours du robot, ainsi que tu calcul de la surface couverte. Je n'ai pas implémenté de solution utilisant le parcours pour optimiser le nettoyage.
Ce module est uniquement visible par le module simulation, parcours appartenant au «modèle».

depuis le rendu intermédiaire, j'ai très largement changé ma façon de créer du code opaque, je me suis rendu compte, grâce aux explications d'un collègue qui avait deja fait 4 ans d'informatique, que je j'avais compris ce que signifiait la programmation opaque, mais que je n'avais aucune idée de comment l'écrire. J'ai donc créé une structure par module, une fonction get et set pour chaque champ de ces structures.
Une fois que c'était fait, tout le reste devenait soudainement extrêmement clair, c'est comme si on ne travaillait pas en opaque, mais ça l'est quand même, puisqu'on a accès a tout, partout.
Depuis là, mon code a évolué tranquillement sans surprises.

la Phase «décision»

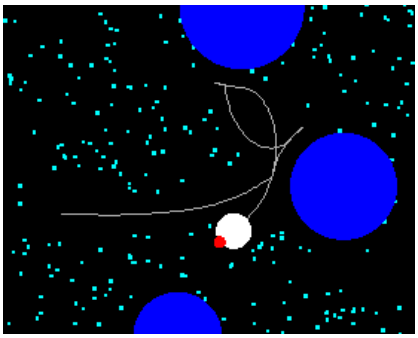
Le mouvement de mon robot est quasi uniquement régit par l'aléatoire. A chaque delta t, la fonction sim_update, génère 2 variations de vitesses, qui sont ensuite appliquées aux 2 vitesses, sauf si l'une des conditions suivantes est valide :

- la variable collisions vient d'être incrémentée :
    - force le robot à tourner vers la droite
- la somme des 2 vitesses est inférieure à W_MAX*1.2
    - force les roues a accélérer uniformément
- l'écart entre les 2 vitesses est supérieur à W_MAX/3.5
    - force la roue qui est en retard a accélérer, et celle qui est en avance à décélérer

La 1ere condition a pour effet d'éviter qu'une collision se reproduise trop longtemps, en obligeant le robot à tourner.

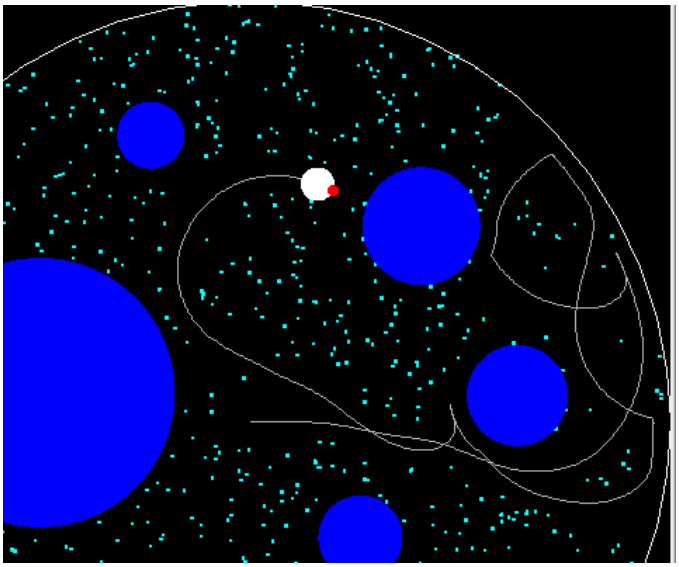
La 2eme condition force le robot a avoir au moins une certaine vitesse, afin qu'il utilise la puissance de son «moteur» à fond durant la simulation.

La 3e condition évite que l'écart entre les 2 roues soit trop important, ce qui l'empêche de faire des tours sur lui même, les 2 roues se suivant constamment, ça permet au robot de décrire de beaux virages.



Finalement, le robot se déplace en général, d'une façon totalement aléatoire, assortie de quelques limites qui doivent être respectées.

Le bonus d'affichage graphique du parcours n'affecte par mes déplacements, mais je l'ai mis en place grâce a une liste chainée qui contient chaque position du robot, elle est ensuite dessinée en affichant un segment entre chacun des éléments de cette liste qui se suivent.



Illustrations

On voit que le robot commence par accélérer uniformément au début (condition 2), sa vitesse étant nulle, ensuite il a rencontré un obstacle et a commencé a redresser sa trajectoire en tournant à droite (condition 1), ce qui ici n'était pas très judicieux, mais j'ai simplifié la réaction à une collision comme cela. c'est tout sauf plus performant, mais ça m'a évité un long ajout de code (à mon avis), la solution n'étant pas simple à ajouter à ce que j'avais déjà fait.

Ici on voit que grâce aux conditions que j'ai imposées aux vitesses du robot l'une par rapport a l'autre (condition 3), il décrit de beaux virages qui lui permettent d'avancer le plus vite possible. tout en changeant de trajectoire aléatoirement, lorsque le robot avance tout droit (aléatoire simple).

Performances

Ces test ont étés faits chez moi avec une machine virtuelle comportant les caractéristiques suivantes :

DD COSUN de l'EPFL

3584 Mo de RAM dédiée
4 Processeurs dédiés
128 Mo de mémoire vidéo dédiée

le tout dans Sun Virtual Box tournant sur un iMac sous mac OS X 10.6.7

test_final_1.txt

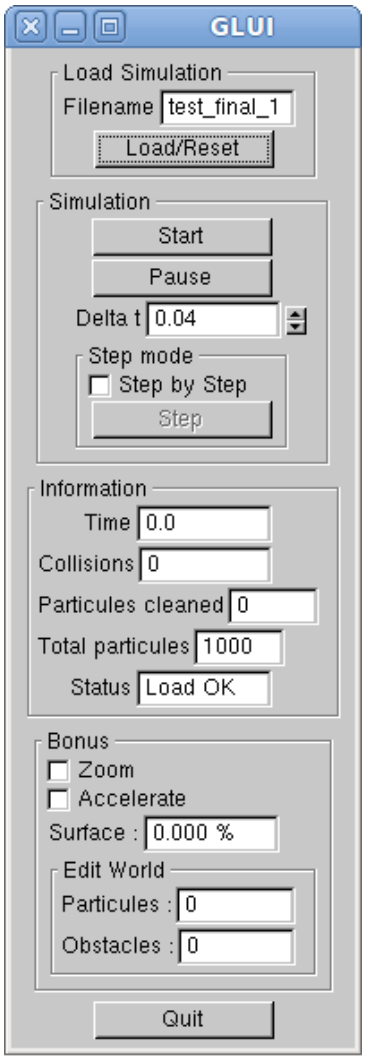
a) delta t = 0.04 :
    a) Temps simulé : 2519,170 s
    b) Collisions : 15 636
    c) Temps de calcul : 2 min 14 (en mode accéléré, sinon TIME OUT)
b) delta t = 0.12 :
    a) Temps simulé : 2184.420 s
    b) Collisions : 5 746
    c) Temps de calcul : 5 min 59
c) delta t = 0.36 :
    a) Temps simulé : 3076,920 s
    b) Collisions : 3 013
    c) Temps de calcul : 3 min 01
d) delta t = 1 :
    a) Temps simulé : 3556,000 s
    b) Collisions : 1 492
    c) Temps de calcul : 1 min 17

test_final_2.txt

a) delta t = 0.04 :
    a) Temps simulé : 1224,430 s
    b) Collisions : 7 910
    c) Temps de calcul : 1 min 16 (en mode accéléré, sinon TIME OUT)
b) delta t = 0.12 :
    a) Temps simulé : 1259,760 s
    b) Collisions : 2 829
    c) Temps de calcul : 4 min 27
c) delta t = 0.36 :
    a) Temps simulé : 1147,640 s
    b) Collisions : 1 138
    c) Temps de calcul : 1 min 29
d) delta t = 1 :
    a) Temps simulé : 2405,000 s
    b) Collisions : 941
    c) Temps de calcul : 1 min 03

Finalement l'efficacité brute de la simulation diminue effectivement en augmentant delta_t, autant au niveau des performances, que de l'affichage graphique qui est mauvais lorsque ce dernier se rapproche de 1.

Pour une simulation aléatoire on peut apercevoir que lorsque seules 95% du nettoyage est demandé, le temps de simulation est quasi divisé par deux. Pour attraper les dernières saletés, il repasse beaucoup par le même endroit inutilement (même si les vrais robots aspirateurs le font pour augmenter la qualité du nettoyage, mais ils le font exprès).



Le nettoyage en une heure, de façon aléatoire, est donc très convaincant et m'a d'ailleurs surpris, comme lors du 1er semestre.

L'interface graphique

Mon interface graphique GLUI se décompose en 4 panels généraux:

- Le panel «Load Simulation»
    - permet d'entrer un nom de fichier présent dans le même dossier que l'exécutable, puis de le charger en appuyant sur enter ou en cliquant sur Load/Reset
- Le panel «Simulation»
    - comporte tout ce qui est nécessaire à l'interaction avec la simulation
        - Start lance la simulation
        - Pause la met en pause
        - lorsque la simulation est en pause, on peut changer les éléments suivants :
            - on peut changer le delta t
            - on peut activer le mode Step by Step en cochant la case de ce même nom, puis effectuer la simulation étape par étape en cliquant sur le bouton Step
- Le panel «Information»
    - contient toutes les informations liées a la simulation en cours :
        - le temps simulé
        - le nombre de collisions entre le robot et le monde
        - le nombre de particules nettoyées
        - le total de particules au début de la simulation
        - l'état d'avancement de la simulation
- Le panel «Bonus»
    - Contient tous les ajouts bonus de ce projet :
        - Zoom sur le robot en live
        - si la case Accelerate est cochée, la simulation se déroule sans affichage afin d'en augmenter la vitesse, l'affichage réapparait lorsque la simulation a atteint une condition d'arrêt
        - la Surface indique le pourcentage de surface nettoyée par le robot
        - le panel «Edit World» permet de changer le nombre de particules et d'obstacles qui sont générés aléatoirement, est utilisable uniquement lorsque la simulation n'a pas encore commencé

Méthodologie

J'ai écrit mon programme dans un peu n'importe quel sens, mon but principal était d'afficher le monde avec les obstacles et particules, à chaque fois que j'avais besoin d'une fonction je la créais là où je pouvais la mettre.
Vers la fin, je réfléchissais à ce dont j'avais besoin, puis je créais une fonction dans projet.cpp, en ajoutant les fonctions pas encore écrites dans simulation, puis je montais dans simulation en écrivant ces fonctions, dans lesquelles j'ajoutais des fonctions pas encore écrites dans les modules en dessous, et finalement j'écrivais les dernières fonctions dans les modules robot.c, monde.c et parcours.c.

Je n'ai pas réellement testé chaque module individuellement, le temps m'aurait manqué pour cela. J'ai donc décidé de tester mon programme avec des fichiers tests, que certaines personnes ont eu la bonne idée de créer, puis de les échanger afin de permettre à chacun de comparer les performances de son robot aux autres.
Le test de l'interface GLUI a été assez long aussi, il m'a fallu essayer le plus de combinaisons de widgets possibles, en vérifiant que chacun désactivait et/ou activait correctement d'autres qui lui étaient associés.

Mon bug le plus fréquent est, sans surprise, le segmentation fault. Sauf que pour ce 2e semestre sa détection a été extrêmement simplifiée par l'utilisation de DDD et GDB qui fournissent quasiment la ligne exacte ou l'erreur a été rencontrée.
La plupart du temps il s'agissait d'erreurs provenant des malloc et free, auquel cas il suffisait de rajouter une condition vérifiant le pointeur «if(pointeur)».
Ils étaient un peu plus coriaces dans le fichier projet.cpp, où je me suis rendu compte, qu'avec un nom de fichier invalide, la simulation le devenant aussi, tous les widgets se transformaient en de potentielles sources de seg. faults.
Il fallait donc aussi ajouter une batterie de «if(sim)» qui empêche l'action de quasi chaque widget si la simulation est incorrecte.

Finalement je pense que le «bug» le plus difficile de ce projet, étaient les problèmes d'ordre géométriques. Trouver une formule me donnant l'angle sectoriel entre 2 obstacles m'a pris plusieurs heures. L'obligation d'utiliser des angles entre -pi et pi n'a pas vraiment aidé la chose. Mais finalement j'ai remplacé mes calculs par ceux d'autres rendus intermédiaires, pour les raisons évoquées plus haut.

Conclusion

Il me semble que ce projet était vraiment un excellent choix, autant du fait qu'il correspond, a bas niveau, certes, à la section, et qu'il est extrêmement motivant de voir son robot se déplacer pour la première fois, je m'en souviens encore.

Avoir une certaine compétition entre les étudiants est aussi stimulant, chacun voulant créer le robot parfait, le plus intelligemment possible.
Cette gestion d'une trajectoire globale qui prend le moins de temps de calcul est vraiment intéressante, même si pour la plupart, dont moi, la «gestion» s'est arrêtée à un simple mouvement aléatoire.

Je suis extrêmement satisfait de mon travail, même si il peut être peaufiné à l'infini, il faut savoir s'arrêter. Il est tellement facile d'implémenter un petit gadget sur l'UI ou dans le

déplacement du robot une fois que la carcasse est là, qu'on a très vite envie de passer des heures à rendre son projet majestueux.
L'une des choses que j'aurais aimé implémenter c'est une gestion des collisions un peu plus réussie, en effet, mon robot ne les évite qu'en tournant vers la droite, quelle que soit son orientation par rapport à l'obstacle ou au monde, il aurait été beaucoup plus productif s'il évitait intelligemment les différents obstacles en tournant le moins possible depuis sa position.
Il me semble cependant que, à nouveau, la satisfaction tirée de tout cela cela vient de l'idée même de ce projet. Parvenir a faire bouger un robot, de façon autonome dans un environnement virtuel. Une fois le but atteint, même de manière simple, le sentiment d'avoir réussi quelque chose de grand en une année est bien présent et je pense que les projets des années a venir devraient être tout aussi stimulants et intéressants.

Peut-être aurait il fallu mettre un accent sur la gestion de la trajectoire, ce problème étant central pour une grosse majorité des microtechniciens qui comptent s'orienter vers les systèmes intelligents. Cela aurait malheureusement nécessité beaucoup de travail, mais certains y sont parvenus, avec des trajectoires élaborées qui nettoient n'importe quelle simulation en moins de 400 secondes.

C'est vraiment ce qui, je trouve, manque à mon projet et à la très grande majorité, l'IA.
Si vous arriviez à forcer la volée suivante à créer un algorithme simple mais efficace pour n'importe quel projet, ce serait super.
Peut être que pour les futures années vous pourriez commencer la création de mini-jeux interactifs. J'ai vu quelqu'un contrôler son robot a l'aide des flèches du clavier, ajouter un composant ludique dans ces lourds projets peut vraiment stimuler d'une façon très importante.
Le projet de l'année passée était souvent trouvé dur par les anciens, mais je pense que c'était surtout lié au fait que le résultat final n'était pas extrêmement passionnant à regarder.

Les contacts que j'ai dans la section me font croire qu'un bon 80% des étudiants MT sont de grands enfants qui ont envie de construire des robots, à stimuler !