UNIVERSITÉ DE SHERBROOKE

IFT630

TP4

Simulateur de livreur de pizza

Antoine DANEAU	11 035 275
Samuel CHAPDELAINE	11 052 923
Patrick GONTHIER	11 022 001

23 avril 2013

SOMMAIRE

1 – PROBLÉMATIQUE

Le concept de base de l'application est de simuler une ville en temps réel avec plusieurs voitures qui circulent dans celle-ci. Le joueur doit quant à lui effectuer des livraisons de pizzas dans la ville en parcourant le chemin le plus rapide en tenant compte de la circulation. Plusieurs problèmes se posent donc devant nous.

- Il faut être en mesure d'avoir une structure représentant la ville et la circulation en temps réel. La structure de la ville doit être idéalement générée aléatoirement.

- Il faut s'assurer que les voitures contrôlées par l'ordinateur soient dotées d'une certaine intelligence pour être en mesure de respecter la signalisation routière ainsi que les autres voitures qui circulent dans la ville. Les voitures doivent également être capables d'atteindre un point précis dans la ville et de s'y rendre de façon assez efficace.

- L'usager doit être capable de savoir facilement quel chemin il doit emprunter. Ce chemin doit être mis à jour en temps réel pour s'ajuster à la circulation qui change constamment ou bien si l'usager décide de ne pas suivre le chemin recommandé. L'algorithme qui aura la tâche de trouver ce plus court chemin doit être efficace et rapide, donc il ne doit pas tout recalculer à chaque fois que le graphe ou bien que le point de départ change.

- Afin de simuler les accidents possibles, il doit y avoir une validation de collision entre la voiture contrôlée par l'usager et celles manipulées par l'ordinateur.

2 – MISE EN OEUVRE

Pour débuter, il était primordial d'avoir des fondations solides, c'est pourquoi nous avons créé notre propre structure de graphe pour modéliser la ville. Chaque arc du graphe est en mémoire en plus de leur poids, le nombre de voiture actuellement sur celui-ci et la dernière voiture qui vient d'y entrer (pour la gestion des collisions). Des sémaphores sont également installés sur les arcs et les nœuds, nous y reviendront dans la section *Parallélisme.*

Grâce à notre graphe, chacune des voitures manipulées par l'ordinateur peut savoir où est la voiture qui se trouve devant elle. Comme ces voitures n'ont pas vraiment besoin de connaître le chemin le plus optimal avec circulation, nous avons implanté A* qui permet de leur fourni un itinéraire fiable sans tenir compte de la circulation pour se rendre à leur destination. Les IAs régénèrent un nouveau chemin une fois arrivé à destination.

Pour la voiture de l'usager, nous voulions trouver le chemin optimal en tenant compte de la circulation. L'algorithme D* Lite s'est donc imposé comme le choix logique. Cet algorithme nous permet d'avoir un point de départ qui change constamment et de supporter des poids d'arc changeant régulièrement de façon beaucoup plus optimale qu'A* qui devrait tout recalculer. D* Lite garde en mémoire le travail déjà accompli et ne fait que des ajustements chaque fois que les poids changent en vérifiant la consistance des nœuds. Les arcs dont les poids ont été modifiés sont stockés dans une file et à intervalle régulier D* Lite mets à jour ces nœuds du graphe et recalcule le chemin le plus court. Seulement les nœuds les plus intéressants pour la solution sont rééquilibrés. Pour ce faire, la liste des nœuds ouverts est triée par une heuristique pour la distance avec le nœud de départ additionnée à la valeur G qui est la distance par rapport au nœud d'arrivée.

Pour la gestion des collisions entre la voiture du joueur et celles contrôlées par l'ordinateur, nous avons implémenté le « *Separating Axis Theorem* ». Cette gestion des collisions s'applique pour deux objets de forme concave. L'algorithme teste la projection des deux formes sur tous les axes normaux de ces deux objets et s'il en trouve une pour laquelle les deux projections n'ont pas de superpositions, il peut conclure aussitôt que les deux objets n'entrent pas en collision. Cela en fait un algorithme très rapide pour la validation en temps réel, puisque l'on peut souvent conclure à la solution rapidement lors des tests des axes.

2.1 – PARALLÉLISME

Pour la conception de notre projet, l'utilisation de notions de parallélisme s'est imposée d'elle-même! Nous avons décidé de modéliser chacune des voitures de la ville comme un fil d'exécution et d'implanter des sémaphores sur les intersections (nœuds du graphe) et sur les routes (arc du graphe). Comme cela, les voitures respectent la priorité de passage aux intersections et les sémaphores sur les arcs sont utilisés pour limiter la circulation sur les routes et prévenir que des voitures se retrouvent coincées en plein milieu de l'intersection, car le tronçon de route est complètement bloqué! Pour éviter les interblocages, nous recalculons un nouvel itinéraire lorsque la voie dans laquelle une voiture veut se diriger est congestionnée.

Nous avons également implanté D* Lite sur un fil d'exécution séparé. Cela permet d'exécuter l'algorithme en temps réel et de mettre à jour le chemin le plus court à la fréquence de temps désiré. Chaque fois que le poids d'un arc se modifie, cet arc est placé dans une collection bloquante (permet de gérer les accès multiples à une liste en C#) et D* Lite n'a plus qu'a défilé cette liste lorsqu'il le désire pour mettre à jour le chemin le plus court.

3 – DÉMARRAGE DE L'APPLICATION

3.1 – ÉXECUTION

L'application a été développée avec le langage C# et le Framework XNA sous Visual Studio 2010. Pour être en mesure de lancer l'application, il faut tout d'abord installer le redistribuable d'XNA inclus dans le répertoire *bin* contenant l'exécutable. Une fois le redistribuable installé, il suffit de double-cliquer sur l'exécutable et en théorie, tout devrait fonctionner. Évidemment, le tout doit rouler sur Windows.

3.2 – COMPILATION

Pour être en mesure de compiler le code source, il faut utiliser Visual Studio 2008 ou 2010 (pas 2012, car le framework n'est plus supporté). Il faut ensuite installer le framework XNA 4.0 à l'adresse suivante :

- http://www.microsoft.com/en-us/download/details.aspx?id=23714

Une fois le tout accompli, il suffit d'ouvrir la solution et de compiler le projet dans Visual Studio (DS95.sln).

4 – SOURCES

L'implémentation de D* Lite a été basée sur le document suivant :

http://www.cs.cmu.edu/~maxim/files/hsplanguide_icaps05ws.pdf

L'implémentation de SAT a été basée sur le document suivant :

http://www.codeproject.com/Articles/15573/2D-Polygon-Collision-Detection

5 – ANNEXES

5.1 – CAPTURE D'ÉCRAN