

DÉPARTEMENT D’INFORMATIQUE

Faculté des sciences

Université de Sherbrooke

Livrable 2 – Capture du drapeau

Par

FÉLIX-ANTOINE OUELLET 09137551

Dans le cadre du cours :

Conception de systèmes temps réel

IFT729

Travail présenté à :

Patrice Roy

Sherbrooke

22 avril 2013

Sommaire

[Produit livré 1](#_Toc353700943)

[Utilisation 2](#_Toc353700944)

[Tests 2](#_Toc353700945)

[Résultats 3](#_Toc353700946)

[Problèmes rencontrés 3](#_Toc353700947)

[Conclusion 4](#_Toc353700948)

## Produit livré

Pour une description plus complète de l’intelligence artificielle produite, je vous invite à consulter le rapport écrit dans le cadre du cours IFT702 qui se retrouve dans le dossier *Docs/IFT702*. Dans le présent document, la description et l’analyse du produit portera essentiellement sur son volet temps réel.

Côté Planificateur, ce qui a été livré correspond exactement à ce qui avait été promis. De fait, l’algorithme à implémenté est relativement simple et il fallait uniquement s’assurer que son implémentation respecte les contraintes de temps qui lui était imposé. Il n’y avait donc pas de composante tems réel spécifique dans ce module. Il devait uniquement s’exécuter assez rapidement pour ne pas causer la défaite du Commandant.

Pour ce qui est du Navigateur, l’algorithme choisi s’est révélé un très bon choix de par le fait qu’il se prête assez bien à un contexte temps réel. Pour faire un bref résumé de l’algorithme, on commence par diviser la grille de jeu en secteurs qui formeront les nœuds d’un graphe de haut niveau. Ce graphe sera alors utilisé pour trouver un chemin entre deux points donnés. Tout au long du parcours de ce chemin de haut niveau, un agent demandera au Navigateur de lui fournir un chemin sur la grille de base. On constate donc qu’il est facile de répondre à des contraintes de temps lorsqu’on ne fait qu’une fraction du calcul du chemin au lieu de le faire en entier. Qui plus est, les fonctions pour faire le calcul des chemins abstraits et concrets sont toutes deux interruptibles. Ils peuvent donc être arrêtés si le temps accordé est presque qu’entièrement consommé. Le Navigateur garde en mémoire l’endroit où le calcul s’est arrêté et peu donc le reprendre lors du prochain tic de jeu.

Une amélioration potentielle qui pourrait être apporté au Navigateur serait la séparation du traitement de la grille de jeu de son utilisation par le Navigateur. Présentement, les deux sont fortement couplés et rendent ne permettent pas grande liberté pour améliorer la navigation des agents. De surcroît, ceci aurait permis d’implémenter une abstraction au-dessus de la grille, telle que les cartes d’influence, qui aurait permis de rendre compte de la nature dynamique de l’environnement. Compte tenu, du temps moyen requis pour calculer un chemin, la charge de calcul supplémentaire associée au maintient d’une carte d’influence n’aurait probablement pas causé un dépassement des limites de temps imposées.

Finalement, le Commandant rempli adéquatement le rôle qu’il lui avait été attribué. De fait, il est le responsable de la distribution du temps de calcul entre les divers modules. En ce sens, il se rapproche du concept d’exécutif vu en classe. Il est cependant beaucoup moins sophistiqué, car la priorité des tâches est relativement séquentielle. Ainsi, il faut d’abord que le Planificateur produise une décision pour un agent. Par la suite, le Navigateur doit guider cet agent. Une fois que ces deux actions ont été répétées pour l’ensemble des agents présentement disponibles, il peut alors donner le restant du temps de calcul au Navigateur pour que ce dernier découvre des chemins les chemins optimaux dans les secteurs qu’il a formés.

## Utilisation

Il existe deux méthodes par lesquelles ont peu mettre en marche l’application produite. D’une part, un fichier appelé Play.bat, situé dans le sous-dossier *bin*, permet de lancer une partie en le double-cliquant. D’autre part, l’application peut aussi être compilée à l’aide des environnements de développement Visual Studio 2010 et 2012. Tel que mentionné dans le fichier README.txt, il est impératif de ne pas mettre à jour le compilateur utilisé, car ceci briserait les liens de l’application avec des librairies tierces. La compilation permet de régler certains détails de l’application si l’usager le désire. Ainsi, il existe trois directives au préprocesseur qui modifieront le comportement de l’application. De fait, définir \_TRAIN fera en sorte que le planificateur tactique appliquera l’algorithme *Q-Learning* lors de chaque partie qu’il jouera. Si cette directive n’est pas définie, il choisira alors la meilleure action selon les valeurs qu’il aura lues d’un fichier. De plus, on peut définir \_LOG\_PERF si on désire se donner une idée du temps que prend le commandant pour donner des ordres à l’ensemble des agents disponibles à en recevoir. De surcroît, la directive \_LOAD\_PLAN, lorsque définit, indique à l’application de charger les *qvalues* produites lors d’une partie précédente. De base, les binaires remis ont été compilés avec les directives \_TRAIN et \_LOG\_PERF définies. Une fois la compilation terminée, on peut lancer une nouvelle partie en se servant toujours du fichier Play.bat. Pour lancer les tests, il suffit de partir *Tests.exe*. Mise en garde, ils peuvent prendre beaucoup de temps à compléter, car, tel que mentionné précédemment, je ne crois pas qu’une seule exécution puisse m’aider à me faire une idée du temps moyen requis pour la complétion d’une fonction. Le nombre d’exécutions total de l’ensemble des tests est donc faramineux et peut faire attendre un correcteur pendant longtemps.

## Tests

Les tests implémentés sont ceux promis dans le livrable 01. Ainsi, chaque module, Commandant, Planificateur et Navigateur, est d’abord testé pour son bon fonctionnement. Par la suite, on teste chacun des modules pour son efficacité en termes de temps. Pour ce qui est du Navigateur, on va s’assurer qu’il est en mesure de découvrir un chemin d’un extrême à l’autre d’un niveau en moins de 80 millisecondes, ce qui est le temps usuel d’un tic de jeu. Il devra aussi être capable d’effectuer son analyse préliminaire de la grille de jeu en moins de cinq secondes, le temps habituellement accordé à l’initialisation des commandants. Le Planificateur sera aussi soumis à des tests de performance pour vérifier qu’il est en mesure de fournir une décision en moins de temps qu’il en est usuellement accordé pour un tic de jeu. Finalement, le Commandant est aussi testé pour s’assurer que les opérations qui lui sont propres ainsi que l’amalgame des trois modules principaux du projet sont en mesure de répondre aux critères temps réel du projet autant du point de vue de l’initialisation que de la prise de décision lors d’un tic de jeu.

Lorsque chaque module fut testé pour son bon fonctionnement et sa vitesse, on testa l’ensemble de l’application dans un contexte réel, c’est-à-dire une partie de capture du drapeau. La vérification des performances pu être faite de deux manières dans ce contexte. Tout d’abord, il faut évidemment que le Commandant puisse terminer une partie sans causer de *timeout*. Par la suite, on va vérifier que le temps de calcul utilisé lors de chaque tic de jeu demeure dans un intervalle acceptable. Pour ce faire, j’ai modifié le client fourni avec l’API pour qu’il puisse mesurer le temps pris par le Commandant à chaque tic si le développeur le désire. Le client conserve chaque mesure de temps qu’il effectue avec *Boost.Chrono* dans un vecteur standard dont il écrira par la suite le contenu dans un fichier à la toute fin de la partie. Le programme de test pourra alors lire ce fichier et déterminer si le Commandant respecte effectivement les contraintes de temps qui avait été déterminé au préalable.

Point important à mentionner, j’ai utilisé le terme usuel lorsque je décrivais les temps ci-haut, car j’ai pu remarquer lors du développement du commandant que les limites de temps mentionnés dans la documentation n’étaient peut-être pas nécessairement celles que je retrouvais dans mes tests. En effet, le temps accordé par tic de jeu semblait se situer dans les 90 millisecondes plutôt que dans les 80 millisecondes. De plus, la documentation mentionnait qu’un cinq secondes de préparation allait être accordé aux commandants avant le début d’une partie. Dans les faits, le temps d’initialisation pouvait aller jusqu’à sept secondes sans causer de problèmes. À la vue de ces disparités entre la réalité et la documentation, j’ai tout de même décidé de conserver les limites telles qu’indiquées dans la documentation comme celles que je devais respecter. Est-ce que cela peut amener mon Commandant à ne rien faire alors qu’il lui reste du temps de calcul? C’est très possible. Malheureusement, un choix doit être fait et celui que j’ai fait m’assure tout du moins que mon Commandant ne provoquera pas de *timeout* même s’il gaspille du temps, un crime majeur dans les systèmes temps réel.

## Résultats

Les résultats des matchs effectués pour tester l’efficacité de mon commandant en temps que planificateur dans un contexte temps réel et multi-agents se retrouvent dans le fichier *results.txt*. Les commandants *BalancedCommander* et *RandomCommander* faisaient partie de ceux offerts avec le SDK. *Terminator* est, quant lui, le médaillé d’argent de la dernière phase de la compétition.

Les mesures de temps de calcul sont quant à eux conservées dans

## Problèmes rencontrés

Le principal problème rencontré lors du développement du commandant fut l’immaturité du SDK et de l’API fournis par les organisateurs de la compétition. De fait, une liste exhaustive des problèmes et des comportements bizarres qui pourraient survenir lors d’une partie se retrouve dans le fichier *bugs.txt*. Le plus critique de ces problèmes concernent le module de communication qui éprouve parfois des problèmes lors de la livraison de messages aux commandants impliqués dans une partie. En effet, il arrive que le traitement côté serveur prenne un peu trop de temps et que cela conduise à l’arrêt abrupt d’une partie. Ceci est très problématique, car il s’agit d’un problème hors de mon contrôle. Il y a quand même de la lumière au bout du tunnel dans le fait que les organisateurs de la compétition ont reconnus qu’il y avait des problèmes au niveau du module de communication. Cependant, la nouvelle version d’AI Sandbox comportant les correctifs promis ne sera disponible qu’après la fin de la session courante.

L’autre problème qui a grandement ralenti le processus de développement fut qu’une partie du débogage du se faire en mode *Release*. En effet, quand arriva le temps de tester le commandant dans un contexte réel, il fallu demander au compilateur d’optimiser le code produit sous peine de ne pas pouvoir initialiser le commandant dans l’intervalle de temps alloué. Évidemment, le compilateur n’est pas tenu de respecter l’ordre d’exécution des instructions tel que prescrite par le développeur, il est uniquement tenu respecter l’intention du programmeur. Encore pire, il a le droit d’éliminer des instructions qui sont soient superflues soient qui ne seront jamais atteintes dans l’exécution du programme. Par conséquent, le débogueur va cruellement manquer d’informations pour aider le développeur à déterminer les erreurs présentes dans son code. Le débogage se résume donc à tenter de reconstituer l’environnement du programme lorsqu’il a planté avec les informations incomplètes que possèdent le débogueur à cet instant. Morale de l’histoire, déboguer du code optimisé se situe dans les mêmes sphères de difficulté que de déboguer du code multiprogrammé.

## Conclusion

Lorsqu’on regarde l’aspect temps réel du produit livré, on peut constater qu’il respecte les contraintes qui avaient été annoncées en début de projet. Bien entendu, l’efficacité du commandant en tant que planificateur laisse encore à désirer, mais ceci est matière à une analyse pour un autre cours soit IFT702.