UNIVERSITÉ DE
SHERBROOKE

DÉPARTEMENT D'INFORMATIQUE
Faculté des sciences
Université de Sherbrooke

Capture du drapeau

Par
FÉLIX-ANTOINE OUELLET 09137551
felix-antoine.ouellet@usherbrooke.ca

Dans le cadre du cours :

Planification en intelligence artificielle
IFT702

Travail présenté à :
Froduald Kabanza

Sherbrooke
11 avril 2013

Table des matières

Description de la problématique

Ce projet s'inscrit dans le cadre de la compétition *Capture The Flag* lancée en partenariat entre *AiGameDev.com* et *Guerilla Games*. Brièvement, le but de la compétition est de concevoir et d'implémenter ce que les organisateurs dénomment un commandant, une intelligence artificielle capable de performer dans le cadre du jeu de capture de drapeau.

Pour bien se situer, la version de ce jeu tel que présenté dans la compétition sera décrite. Tout d'abord, chaque équipe sera composée de 4 à 15 agents que le commandant devra coordonner dans un contexte temps réel. Ainsi, il ne disposera que d'environ 80 millisecondes pour évaluer l'état courant de la partie et faire part de ses décisions à ses agents. S'il dépasse cette limite, il perdra sa connexion avec le serveur de jeu et il accumulera une défaite. De plus, le commandant dispose aussi de 5 secondes avant le début officiel de la partie pour analyser la carte ou lire des informations qu'il aurait conservées dans des fichiers. Il faut aussi mentionner qu'il y aura périodiquement des résurrections de masse. En d'autres termes, à chaque intervalle, et ce peu importe l'appartenance de l'agent, tout agent mort réapparaîtra dans la base qu'il lui a été assigné et sera de nouveau un joueur actif dans la partie en cours.

Il existe pour chaque équipe trois lieux de grande importance. Tout d'abord, tel que le paragraphe ci-dessus le mentionnait, chaque équipe dispose d'une base à partir de laquelle il débutera la partie et reviendrait au jeu s'il devait tomber au combat. Par la suite, chaque équipe se voit attribuer un drapeau qui se situe dans un endroit déterminé par le niveau joué. Finalement, il existe aussi un lieu unique à chaque équipe où un agent en possession du drapeau devra se rendre pour comptabiliser un point.

Il faut aussi mentionner les informations dont dispose chaque commandant. Ils ont bien évidemment accès à toutes les informations que peuvent leur offrir les agents à leur charge. Les commandants ont aussi connaissance des positions des drapeaux tout au long de la partie. Par contre, ils n'ont qu'une connaissance limitée des agents adverses. On doit comprendre ici que le commandant n'aura conscience de la position d'un agent ennemi que si un de ses propres agents aperçoit un adversaire sur le terrain.

Pour ce qui est du déroulement d'une partie, elle se déroule de la manière suivante. Après cinq secondes d'initialisation, les commandants commencent à envoyer leurs ordres aux agents à leur charge. Le but principal qu'ils poursuivent est de s'emparer du drapeau adverse et de l'apporter dans ce qu'on pourrait appeler sa zone de but pour ainsi augmenter le pointage de son équipe. Bien entendu, on va tenter en même temps d'empêcher les agents adverses de se saisir de notre drapeau et ainsi permettre à l'équipe adverse de comptabiliser un point. Il est donc intéressant de noter que chaque agent présent sur le terrain dispose d'une arme de tir capable de neutraliser un autre agent. Par contre, un agent mort ne restera pas dans cet état pour le reste de partie tel que mentionné plus haut. Le commandant aura donc à considérer tous ces aspects de la partie lorsque

viendra le temps de décider de la marche à suivre de ses agents. Ici, on doit préciser que le commandant est soumis à des contraintes temps réel. Outre qu'il ne peut prendre plus de cinq secondes pour s'initialiser, il devra, à chaque 80 millisecondes, être en mesure de décider des actions que devront entreprendre les agents disponibles et de leur transmettre ses ordres. Tout manquement à ces contraintes mènera à la défaite de l'agent fautif. Finalement, après cinq minutes de jeu, la partie se termine et l'équipe ayant capturé le plus souvent le drapeau adverse gagne.

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.

Algorithmes

Pour résoudre le problème proposé, il a fallu mettre en place deux algorithmes pour traiter des deux aspects dont il est constitué soit la planification des trajectoires et la planification tactique.

Pour ce qui est de la planification des trajectoires, l'algorithme retenu est celui introduit par [Hier2] : *Hierarchical Path-Finding A** (HPA*). Grosso modo, cet algorithme se veut une alternative à l'algorithme A* dans des contextes où l'on retrouve de fortes contraintes au niveau du temps. Pour se faire, il va diviser le terrain en secteurs qui se voudront des abstractions au-dessus des nœuds de base de la grille de jeu traitée. Par la suite, on va créer des portes à l'intérieur de ces secteurs. Par porte on doit comprendre un ensemble de nœuds permettant le passage vers un secteur adjacent. . Pour ne pas créer trop de nœuds de haut niveau, seul un certain nombre de nœuds de bas niveau seront choisis

comme base des nœuds de haut niveau. Ce nombre sera déterminé par des facteurs tels que la taille de porte désirée et le nombre d'obstacles présent sur la bordure entre deux nœuds. Une fois ces nœuds créés, on va joindre d'un lien toute paire de nœuds appartenant à un secteur donné entre lesquels un passage de bas niveau existe. Ceci nous donnera un graphe de secteurs qui nous permettra de planifier une trajectoire de haut niveau. Conséquemment, la recherche dans un graphe en utilisant cette tactique permettra de répondre à des contraintes temporelles. Ainsi, le nombre réduit de nœuds dans un graphe de haut niveau réduit de beaucoup le temps de calcul de l'algorithme. De surcroît, le chemin qu'un agent doit suivre au niveau de base ne sera construit qu'au fur et à mesure qu'il parcourt les secteurs.

En ce qui a trait à la planification tactique, l'algorithme retenu est *Q-Learning* tel que décrit dans [RLearn]. Très brièvement, cet algorithme va d'apprendre une fonction action-valeur en se promenant dans un processus de décision markovien. Chaque transition entre états donne une certaine récompense à l'agent. Son but sera donc de tenter de maximiser la récompense obtenue en découvrant la suite d'états et de transitions entre ces états qui lui rapporte le plus. Une fois l'agent entraîné, le plan qu'il suivra se résumera à effectuer les actions nécessaires pour passer aux états qui lui ont rapporté la plus grande récompense.

Implémentation

Un aspect important de l'implémentation à mentionner est la modélisation du problème sous la forme d'un processus de décision markovien, quels sont les états et quels sont les actions. Ici, il faut mentionner que la modélisation proposée s'inspire largement de la modélisation utilisée par [Cerb]. Ainsi, un état dans le commandant implémenté représente aussi les réponses aux questions suivantes : est-ce que notre drapeau est à la base? Est-ce que le drapeau ennemi est à la base ennemie? Est-ce que l'agent courant possède le drapeau ennemi? Est-ce que notre drapeau s'est fait échapper? Est-ce que le drapeau ennemi s'est fait échapper? De surcroît, l'implémentation proposée ajoute la question est-ce qu'un allié possède le drapeau ennemi?

Une fois les états possibles établis, on peut s'attarder aux transitions possibles entre ces états. Autrement dit, quelles sont les actions de haut niveau qu'un agent donné peut effectuer? Dans notre cas, six actions existent. Tout d'abord, le commandant peut ordonner à un agent d'aller chercher le drapeau de l'équipe adverse. Naturellement, on peut donner à un agent l'ordre de retourner à la base ce qui sera fort apprécié lorsqu'il sera en possession du drapeau des opposants. De plus, il peut demander à un agent de défendre la position courante du drapeau. Il peut aussi lui demander de supporter son coéquipier qui est présentement en possession du drapeau de l'ennemi. À l'inverse, on peut aussi ordonner à un agent de trouver et d'abattre l'agent ennemi qui a pris possession de notre drapeau pour empêcher l'adversaire de marquer un point. Dernière

action possible, on peut aussi demander à un agent de se rendre à l'endroit où un ennemi doit apporter le drapeau et attendre qu'un adversaire se pointe pour l'abattre.

Pour compléter cette modélisation, il faut maintenant définir des récompenses et des punitions pour nos agents. Dans l'implémentation proposée, le système de récompenses va observer les événements qui se sont produits récemment pour déterminer si l'agent a pris une bonne décision lorsqu'il a passé d'un état à un autre en effectuant une action donnée. Le tableau ci-dessus illustre les récompenses et les punitions qui peuvent être dispensées.

Action	Condition	Récompense
GetEnemyFlag	L'agent s'est emparé du drapeau	240
WaitEnemyBase	Un coéquipier s'est emparé du drapeau	120
	L'agent s'est emparé du drapeau	80
	SINON	-32
Defend	L'agent est en possession du drapeau	-80
	L'agent a tué un ennemi	80
	SINON	20
KillFlagCarrier	L'agent a tué celui qui portait le drapeau	80
	Un ennemi est en possession du drapeau	20
	SINON	-10
SupportFlagCarrier	Un coéquipier s'est emparé du drapeau	80
	Un coéquipier est en possession du drapeau	20
	SINON	-8
ReturnToBase	L'agent a capturé le drapeau	320
	L'agent est en possession du drapeau	80
	SINON	-120

À ceci vont s'ajouter une récompense et une punition plus globale. Lorsque l'équipe capture le drapeau, on ajoute 1600 points au score de tous les agents. Inversement, si le drapeau de l'équipe se fait capturer, on retire 400 points au pointage des membres de l'équipe. On soustrait aussi 100 points à un agent s'il se fait tuer, peu importe l'action qu'il effectuait.

Un autre aspect qu'il fallut considérer lors de l'implémentation de la planification tactique fut le dilemme exploration/exploitation. Rapidement, le dilemme réside entre toujours choisir l'action jugée qui va maximiser la récompense potentielle en fin de jeu selon et choisir une autre action quelconque pouvant mener à une meilleure suite d'action qui va améliorer notre récompense en fin de partie. Le commandant implémenté traite ce dilemme à l'aide d'un paramètre ε qui dénote la probabilité de choisir une action au hasard au lieu de choisir l'action optimale à la vue des *qvalues* courantes. L'algorithme *Q-Learning* est donc modifié pour intégrer ce paramètre lors de son choix de la prochaine action à entreprendre.

Résultats

L'ensemble des résultats des simulations effectuées se retrouve dans le fichier *results.txt* situé dans le dossier *MyCommander*. Brièvement, les simulations visaient d'abord et avant tout à cibler la valeur de ε optimale. Pour ce faire, trois parties étaient jouées avec un jeu de paramètres donnés. Par la suite, on jouait deux autres parties. Dans une de celles-ci, le commandant exécutait un plan selon les *qvalues* apprises lors d'une partie d'entraînement avec les paramètres l'ayant conduit au plus grand nombre de victoires. Dans l'autre, il tentait d'améliorer son plan précédemment obtenu en entraînement.

Un détail à mentionner, est que le taux d'apprentissage et le taux d'escompte n'ont jamais variés lors de toutes ces expériences. Les valeurs qu'ils ont prises sont celles ayant été jugées comme optimales dans [Cerb]. Ce choix peut s'expliquer par deux raisons. D'une part, le but ici était d'essayer de prendre une position dans le dilemme exploration/exploitation et non de vérifier quels paramètres conduisait à un meilleur apprentissage des *qvalues*. D'autre part, le temps commençait à manquer lorsque les simulations ont débuté. Il fallait donc choisir ses batailles, choisir les tests les plus pertinents.

Un autre aspect des simulations qui va affecter les résultats est le fait que les caractéristiques de la partie varient de simulation en simulation. Il se peut donc que certaines configurations de terrain soient plus ou moins avantageuses pour le commandant implémenté. Aussi, le nombre d'agents à coordonner diffère entre les parties demandant ainsi une plus ou moins grande charge de calcul au commandant. De surcroît, plus le nombre d'agents contrôlés augmente, plus le nombre de mises à jour la table des *qvalues* va augmenter. Le commandant découvrira donc plus rapidement un plan, mais l'optimalité de ce plan ne sera pas garantie.

Discussion

Lorsqu'on regarde une partie, on peut très vite s'apercevoir que le mouvement des agents commandés n'est pas ce qu'il devrait être. On remarque ainsi que leurs mouvements sont erratiques et pas toujours optimaux. La nature erratique de leurs déplacements résulte du fait que l'ordre de se déplacer d'un point A à un point B en passant par des endroits

précis n'est donner qu'à chaque transition de secteurs. Pour remédier partiellement à ce problème, le chemin concret aurait pu être mis en cache une fois que l'agent aurait parcouru tous les secteurs sur le chemin de son but. L'autre partie de la solution à ce problème serait d'avoir un moyen de continuellement prolonger le chemin concret que doit suivre un agent et non pas de lui envoyer un nouveau chemin concret à chaque fois qu'il arrive dans un nouveau secteur. Malheureusement, ceci n'est pas une solution applicable dans le contexte présent. Ainsi, le système fourni par les organisateurs de la compétition va exiger du commandant qu'il envoie des messages bien clairs sur les ordres donnés. Il ne pourra donc pas simplement remplir une liste de position qui pourrait être consommée par le serveur de jeu.

Une autre chose que l'on peut remarquer à propos des déplacements des agents est que les chemins qu'ils empruntent ne sont pas optimaux. Ceci est dû à l'algorithme utilisé qui met l'emphase sur la rapidité d'exécution et non sur l'optimalité de la solution produite. Effectivement, le mouvement que l'on aperçoit découle du fait qu'un chemin optimal est calculé pour passer d'une porte d'un secteur à une autre. Cependant, il n'y a aucune garantie offerte que les portes sont placées de façon à garantir des chemins optimaux. Pour obtenir de meilleurs chemins, il faudrait appliquer un algorithme de raffinement sur le chemin concret total produit par HPA*.

De surcroît, on peut observer que les agents se déplacent en n'accordant que très peu de considération aux agents du commandant ennemi. On peut attribuer deux raisons à ce comportement. D'une part, il s'agit d'une décision de design. En effet, le commandant a été écrit de telle sorte que la traduction d'actions de haut niveau en action de bas niveau devait permettre aux agents de se débarrasser des ennemis se trouvant dans leur chemin. D'autre part, il n'y a aucune modélisation de la menace dans le commandant créé. Le but ici était de produire une implémentation de HPA* respectant des contraintes de temps réel. Par conséquent, la modélisation de la menace que posaient les agents ennemis sortait du cadre du projet. Néanmoins, ce serait une voie intéressante à explorer qui pourrait venir bonifier le commandant produit.

Malgré les remarques ci-dessus, l'implémentation offerte de HPA* possède tout de même des qualités intéressantes. La plus importante d'entre elles est qu'elle est très rapide et qu'elle satisfait donc très facilement les contraintes de temps qui lui sont imposées. Qui plus est l'implémentation offerte va garantir que le temps d'exécution de l'algorithme va décroître tout au long de la partie. Un autre bénéfice est que le fait qu'un agent s'arrête à la porte de chaque secteur lui permet de mieux se débrouiller lors d'engagements avec l'ennemi. En effet, le fait qu'il soit arrêté réduit sa pénalité sur le temps de viser et faire feu sur l'adversaire.

L'autre point à considérer lorsqu'on analyse les performances du commandant implémenté est le module de planification tactique. Ici, l'endroit où le bât blesse n'est pas

nécessairement l'algorithme choisi ou son implémentation. En effet, le projet de tests permet d'affirmer que l'algorithme est correct. En réalité, la partie qui peut faire défaut est la modélisation du problème en tant que processus de décision markovien. La modélisation utilisée ici était largement inspirée de celle proposée dans [Cerb] et qui avait produit des résultats forts intéressants. Cependant, il se peut que la modélisation produite ne tienne pas assez en compte des particularités du jeu de capture du drapeau tel que défini par les organisateurs de la compétition.

Au-delà de la modélisation, le plus gros point de défaillance possible est la traduction des actions de haut niveau en action de bas niveau. De fait, cette traduction est le point critique du commandant, car c'est elle qui communique les intentions du commandant au serveur de jeu. Dans l'implémentation fournie, la traduction fait en sorte que les agents utilisent plus souvent la commande «charge» qui fait en sorte que l'agent coure vers la direction donnée en suivant un chemin donné. Si l'agent aperçoit un ennemi, il s'arrêtera pour le mettre en joue et le tenter de l'abattre. C'est une solution acceptable, mais qui peut parfois être trop proactive dans des situations où la prudence serait de mise.

Finalement, une meilleure analyse du fonctionnement de la communication entre le serveur de jeu et le commandant implémenté aurait peut-être permis d'éviter certains comportements étranges. Par comportements étranges, j'entends des situations où un agent affiche une certaine intention en effectuant un tout autre comportement. Il faut tout même mettre un bémol sur l'amélioration potentielle du commandant en utilisant cette approche. En vérité, l'application serveur fournie par les organisateurs de la compétition comporte un bon nombre de bogues. D'ailleurs, une liste exhaustive peut être retrouvée dans le fichier *bugs.txt* situé dans le fichier *MyCommander*. Conséquemment, certaines choses ne peuvent tout simplement pas être améliorées. Heureusement, les gens d'*AiGameDev* sont présentement en train de refaire le module de communication et la prochaine version du SDK ne devrait pas être encombrée par ces bogues.

Conclusion

Ce qu'on peut retenir de ce projet est que l'utilisation d'une technique issue du milieu de l'apprentissage machine, soit Q-Learning, pour bonifier la planification tactique dans un jeu de stratégie en temps réel est une pratique qui demande beaucoup de finesse. La modélisation du problème doit être longuement réfléchie pour que les agents puissent effectivement arriver à continuellement améliorer leur plan original. De plus, une approche hiérarchique à la planification de trajectoire est très intéressante dans un contexte temps réel, car elle permet de respecter des contraintes de temps peu importe la taille du graphe. Par contre, cet algorithme ne garantit pas des chemins optimaux et il faut donc lisser les chemins produits. Bref, les algorithmes implémentés forment une bonne base pour effectuer de la planification multi-agents dans un contexte temps réel, mais ils

ne sont pas suffisants en eux-mêmes et requièrent donc d'être optimisés pour atteindre les performances désirées.

Médiagraphie

[AISandbox] AiGameDev, *Capture The Flag*, 2012, http://aisandbox.com/

[Cerb] Hefny, A., Hatem, A., Shalaby, M. and Atiya A., *Cerberus:Applying Supervised and Reinforcement Learning Techniques to Capture the Flag Games*, Cairo University, Faculty of Engineering, Computer Engineering Departement, 2008

[Hier1] Thomas, *Hierarchical pathfinding: explained*, 2011, http://jagedev.blogspot.ca/2011/08/hierarchical-pathfinding-explained_28.html

[Hier2], Botea, A., Müller, M. and Schaeffer, J., Near Optimal Hierarchical Path-Finding, University of Alberta, Department of Computing Science, 2004

[RLearn] Sutton, S. and Barto, G. "Reinforcement Learning: An Introduction", MIT Press, Cambridge, MA, 1998