Chapitre 8

Des stratégies qui s'adaptent à la situation dans les jeux de stratégie temps réel

8.1. Introduction

Les jeux de stratégie temps réel consistent à faire s'affronter sur une carte plusieurs groupes d'unités. Les unités se déplacent simultanément, et l'enjeu de la bataille est souvent la domination de la totalité de la carte. Une composante importante de ces jeux est l'acquisition de ressources qui permettent de construire de nouvelles unités. Certains de ces jeux comme Command and Conquer, Total Annilihation, Ages of Empire ou Warcraft sont très populaires. Ces jeux sont le plus souvent joués en réseau contre d'autres adversaires humains car l'Intelligence Artificielle fournie avec le jeu est généralement plus faible que des joueurs expérimentés.

Ce chapitre compare différentes stratégies qui peuvent être utilisées dans un jeu de stratégie temps-réel simple. Certaines de ces stratégies sont statiques : elles ne dépendent pas de la situation sur la carte. D'autres sont dynamiques : elles s'adaptent à la situation globale. Nous y présentons un jeu temps réel élémentaire, puis nous analysons les résultats obtenus en faisant jouer entre elles différentes stratégies dans des conditions variées.

Chapitre rédigé par Tristan CAZENAVE.

8.2. État de l'art

Parmi les algorithmes qui choisissent dynamiquement des actions en fonction de la situation courante, les algorithmes de Monte-Carlo forment une famille d'algorithmes qui reposent tous sur un ensemble de simulations plus ou moins aléatoires.

Les méthodes de Monte-Carlo ont été utilisées pour de nombreux jeux à information incomplète. Au Bridge, GIB utilise les méthodes de Monte-Carlo afin de faire des statistiques sur des résolutions de mains ouvertes [GIN 99]. Au Poker, et plus spécifiquement au Texas Hold'em, le programme Poki utilise des méthodes d'échantillonage sélectif, et des simulations pour décider de ses stratégies de mises [BIL 02]. Au Scrabble, le programme Maven contrôle des simulations sélectives [SHE 04]. L'utilisation de simulations façon Monte-Carlo a aussi été appliqué au Go fantôme, en plaçant aléatoirement les pierres dont la position est inconnue avant chaque simulation [CAZ 05a].

Les méthodes de Monte-Carlo ont également été utilisées pour les jeux à informations complète. Ainsi B. Abramson a proposé d'utiliser l'échantillonnage comme une fonction générale d'évaluation pour les jeux [ABR 90].

La méthode a été appliquée au jeu de Go par B. Bruegmann [BRU 93]. La combinaison de méthodes de Monte-Carlo avec d'autres techniques de programmation du jeu de Go comme les recherches tactiques [CAZ 05b], les formes [BOU 06] ou la recherche globale [COU 06] donne de bons résultats.

La combinaison de recherche arborescente et de Monte-Carlo pour les problèmes mono-agents a été décrite par H. Juillé [JUI 99].

Concernant les jeux de stratégie temps-réel, des tests préliminaires ont été effectués par T. Cazenave et ont montré que l'approche pouvait donner de meilleurs résultats qu'une stratégie simple [CAZ 04]. Dans la même veine, M. Chung a utilisé des méthodes de Monte-Carlo et de la planification pour un jeu de capture de drapeau [CHU 05].

Une autre approche de la programmation de l'Intelligence Artificielle dans les jeux de stratégie temps réel est l'apprentissage. P. Spronck et M. Ponsen ont utilisé l'apprentissage dans les jeux de stratégie temps réel [PON 05] et dans les jeux de rôles sur ordinateur [SPR 06]. Dans ces travaux, le programme apprend quelles séquences d'actions (c'est à dire tactiques) effectuer pour bien jouer face à un adversaire. Cet apprentissage peut être utilisé en ligne pour adapter le programme au joueur humain contre lequel il joue, ou hors ligne pour engendrer automatiquement des tactiques.

8.3. Un jeu de stratégie temps réel simple

Dans cette section, nous présentons le jeu de stratégie que nous avons utilisé pour tester notre approche. Pour simplifier les calculs et accélérer la prise de décision, nous avons utilisé une abstraction de jeu de stratégie. Le jeu de stratégie abstrait permet de représenter de façon plus concise les éléments d'un véritable jeu de stratégie temps réel. La carte du jeu abstrait contient moins d'éléments, et les actions regroupent plusieurs actions (par exemple les unités tirent plusieurs fois de suite chacune à leur tour après s'être déplacées). La carte est quadrillée, et chaque carré est représenté dans la carte abstraite par un symbole.

La carte contient une base immobile pour chacun des joueurs. Dans le cas à deux joueurs, il y a une base située près du milieu du bord haut et sa base adverse située près du milieu du bord bas. Au début, soit la carte ne contient que les bases, soit elle contient les bases et un nombre fixé d'unités mobiles de chaque couleur, proches de leurs bases. Les bases servent à construire des unités, pour cela elles utilisent du métal. Au début chaque joueur dispose d'un capital de métal, toutefois s'il veut continuer à construire des unités une fois son capital épuisé, un joueur doit aller puiser du métal dans des zones prédéfinies sur la carte. Pour puiser du métal, il positionne une unité sur une source de métal. Les sources de métal sont disposées symétriquement sur la carte. Le métal, une fois puisé, est immédiatement disponible pour construire de nouvelles unités.

Les unités mobiles peuvent tirer sur les unités adverses et les détruire. Chaque unité a un niveau de santé ; à chaque fois qu'elle se fait tirer dessus son niveau de santé baisse. Une unité ne peut tirer sur une unité adverse que si celle-ci se trouve dans un de ses huit voisins sur la carte abstraite (une unité ne peut atteindre avec son tir que les unités sur les cases voisines horizontales, verticales et diagonales de la carte abstraite). Après chaque déplacement chaque unité peut tirer un nombre limité de fois sur l'unité adverse qu'elle choisit.

Une partie est terminée lorsque toutes les unités ennemies ont été détruites, ou lorsque le nombre de pas de la simulation dépasse un seuil.

La fonction d'évaluation renvoie la différence entre la somme des santés des unités amies et la somme des santés des unités adverses.

La boucle principale du jeu est la suivante. Son corps est exécutée à chaque pas de la simulation :

```
void BouclePrincipale () {
    RecupereMetal ();
    ConstructionUnités ();
```

```
    ChoixDesDéplacements ();
    Déplacements ();
    ChoisirLesUnitésCibles ();
    TirerSurCibles ();
}
```

L'Intelligence Artificielle est principalement contenue dans la fonction ChoixDes-Déplacements. C'est dans cette fonction que sont programmées les différentes stratégies que nous exposons par la suite.

Une des décisions que prend l'Intelligence Artificielle est la cible que poursuit chaque unité. Ce choix a son importance lors de la phase de tir sur les cibles. Le programme fait tirer une unité mobile en priorité sur la cible choisie par l'Intelligence Artificielle si la cible est voisine de l'unité mobile. Dans le cas ou la cible n'est pas dans le voisinage, le tir s'effectue sur une des autres unités mobiles adverses voisines de l'unité mobile. Quand il n'y a pas d'unités adverses voisines, l'unité mobile ne tire pas.

8.4. Les stratégies statiques

Dans cette section, nous décrivons trois stratégies qui ne s'adaptent pas. Elles sont robustes et ne se modifient pas au cours de la partie.

8.4.1. *La stratégie Nearest*

La stratégie Nearest consiste à déterminer pour chaque unité l'unité adverse la plus proche. Une fois ce choix fait, chaque unité se déplace dans la direction de l'unité adverse qui lui correspond. Si l'unité adverse occupe une de ses cases voisines, les tirs se feront en priorité sur elle.

Concernant la gestion du métal, la stratégie Nearest construit des unités tant qu'il y a du métal disponible.

8.4.2. *La stratégie Rush*

La stratégie Rush est parfois utilisée par les joueurs humains dans certains jeux de stratégie temps réel. Elle consiste à envoyer toutes ses unités disponibles à l'attaque de la base ennemie, dans le but de la détruire et d'empêcher l'ennemi de se développer et de construire de nouvelles unités.

De même que pour la stratégie Nearest, la stratégie Rush gère le métal de façon simpliste. Elle construit des unités tant qu'elle dispose des ressources disponibles mais ne s'occupe pas d'en récolter de nouvelles.

8.4.3. *La stratégie M(n)*

Les stratégies $M(n)$ ont une gestion du métal plus élaborée. Elles affectent en priorité n unités à la collecte de métal. L'algorithme hongrois permet de trouver relativement rapidement l'affectation optimale de n unités parmi toutes les unités disponibles. Il a déjà été utilisé à Sokoban par exemple, pour calculer une heuristique admissible correspondant au nombre minimal de déplacements à effectuer pour affecter chaque caisse à un emplacement cible [JUN 99].

Pour $M(n)$ nous avons retenu un algorithme plus simple et moins coûteux en temps qui donne toutefois de bons résultats, même s'il ne trouve pas une affectation optimale dans tous les cas. Il consiste à chercher parmi toutes les unités amies celle qui est la plus proche d'une source de métal non affectée. Il affecte ensuite l'unité à cette source de métal, puis recommence avec les unités restantes jusqu'à ce que n unités aient été affectées, ou qu'il n'y ait plus d'unités disponibles.

Pour les unités restantes qui ne sont pas affectées à une source de métal, la stratégie Nearest est employée.

La base amie construit autant d'unités qu'elle le peut, c'est à dire tant qu'elle a du métal.

8.5. La stratégie Monte-Carlo

La première application d'un algorithme de Monte-Carlo que nous ayons faite consistait à essayer un coup aléatoirement pour chaque unité, puis à simuler la stratégie Nearest pour toutes les unités, de façon à avoir une évaluation de tous les coups joués au début de la simulation. Chaque coup est associé à de nombreuses évaluations, et son score moyen est la somme des évaluations divisé par le nombre d'échantillons. Lorsque le nombre d'échantillons est de n, la stratégie Monte-Carlo est notée $MC(n)$.

Dans des travaux précédents [CAZ 04], nous avons montré qu'utiliser des statistiques sur l'achèvement de buts donne de meilleurs résultats que des statistiques sur les coups. Ceci a été vérifié aussi bien pour un jeu de stratégie temps réel que pour le jeu de Go [CAZ 05b].

Une amélioration a consisté à choisir aléatoirement un but pour chaque unité avant chaque simulation. Puis lors de la simulation, les unités poursuivent leurs buts, et se replient sur une stratégie Nearest une fois que leur but est atteint. Après chaque simulation, la note du but associé à une unité est mise à jour en fonction du résultat de la simulation.

Lors des simulations, l'adversaire utilise la stratégie Nearest.

Les buts possibles sont de chasser une unité adverse ou de se rendre sur une source de métal.

8.6. Les stratégies adaptatives

Les deux stratégies que nous présentons reposent sur les stratégies statiques, mais contrairement à celles-ci, elles n'appliquent pas la même stratégie dans toutes les situations. Nous avons programmé deux stratégies StaticSelect et Select.

8.6.1. *La stratégie StaticSelect*

StaticSelect consiste à faire jouer dans la position initiale une ensemble de stratégies statiques contre ce même ensemble de stratégies statiques.

Le programme fait jouer chacune des stratégies contre toutes les stratégies, y compris elle même. Pour chaque rencontre entre deux stratégies, une simulation est effectuée pour un grand nombre de pas. Si la simulation se termine par la victoire de la stratégie sélectionnée, ou si après un grand nombre de pas l'évaluation de la position est positive, le programme comptera un point pour la stratégie sélectionnée. Dans le cas ou la stratégie perd, ou si l'évaluation est finalement négative, aucun point ne sera compté.

Chaque stratégie est alors associée à un nombre de points. Le programme sélectionne la stratégie qui a obtenu le plus de point dans la position initiale, puis applique par la suite cette stratégie dans tout le reste de la partie.

8.6.2. *La stratégie Select*

La stratégie Select est plus dynamique, puisqu'au lieu d'appliquer la stratégie sélectionnée dans la position initiale pendant toute la partie, elle recommence à chaque pas de la partie toutes les simulations entre stratégies. Elle sélectionne à chaque pas de la partie la stratégie qui a le plus de points.

Contrairement à StaticSelect, Select peut changer de stratégie au cours d'une partie. Select peut se rendre compte que la situation change et devient plus favorable à l'application d'une autre stratégie que la stratégie en cours.

8.7. Résultats expérimentaux

Les stratégies ont été implémentées et testées sur un PC 1.7 GHz sous Linux. Les temps de réponses des différentes stratégies vont de temps négligeables pour les stratégies statiques jusqu'à un maximum de cinq dixièmes de secondes par déplacement pour $MC(100)$.

8.7.1. *Les configurations de départ*

Afin de tester les stratégies dans des configurations de jeu différentes, nous avons utilisé plusieurs cartes avec des conditions initiales différentes. Nous avons fait varier la taille des cartes, le nombre de sources de métal ainsi que le nombre d'unités mobiles disponibles au début de la partie.

Les deux tailles de cartes utilisées sont dix-neuf par dix-neuf et trente-neuf par trente-neuf. Pour ces deux tailles, la carte peut soit contenir quatre sources de métal (une à chaque coin), soit contenir treize sources de métal (quatre dans les coins, cinq au centre, et quatre entre les bases et le centre). Pour chaque taille, et chaque répartition de sources de métal, la carte peut contenir au début de la partie soit dix unités mobiles par couleur, soit aucune unité mobile.

Cela fait donc un total de huit configurations différentes qui ont été testées. Pour chaque configuration, chaque stratégie peut prendre soi la couleur de la base en haut de la carte, soit la couleur de la base en bas de la carte, ce qui amène à seize confrontations entre chaque couple de stratégies.

8.7.2. *Les paramètres du jeu*

Les paramètres du jeu ont été fixés afin de rendre le jeu équilibré entre gestion des ressources et attaque, et de façon à ce qu'ils ne soient pas trop éloigné des paramètres habituels des jeux de stratégie temps réel.

Les unités peuvent se déplacer d'une case abstraite à chaque pas de simulation. Elles peuvent tirer après chaque déplacement, et chaque tir fait baisser de cinq unités le niveau de santé de la cible. Au début de la simulation, chaque base a un niveau de santé de cent unités. Chaque unité mobile commence avec un niveau de santé de trente.

Au début d'une partie, chaque base dispose de cent unités de métal. A chaque pas de la simulation, elle peut en utiliser dix pour construire de nouvelles unités mobiles. Chaque unité mobile nécessite vingt unités de métal pour être construite. Une base peut donc créer au maximum une unité mobile tous les deux pas de simulation.

Chaque source de métal sur laquelle est disposée une unité mobile rapporte à la base une unité de métal par pas de simulation.

8.7.3. *La confrontation entre les stratégies*

Pour estimer chacune des stratégies, nous avons organisé un tournoi entre toutes les stratégies. Le nombre de simulations pour la stratégie Monte-Carlo a été fixé à cent, ce qui permet un temps de réponse très rapide pour chaque décision de déplacement, elle est notée $MC(100)$. En pratique, la stratégie Monte-Carlo manque de persévérance. Par exemple, au lieu de rester sans se déplacer sur une source de métal, il lui arrive de faire des aller-retours entre la source et une de ses cases voisines.

Les stratégies statiques et le stratégie StaticSelect ont des temps de réponse négligeables.

La stratégie Select à un temps de réponse compris entre celui des stratégies statiques et celui de $MC(100)$. Le temps est toutefois bien inférieur à celui de $MC(100)$ puisque seules seize simulations sont effectuées avant chaque déplacement. Les quatre stratégies statiques utilisées pour faire les simulations de Select sont Nearest, $M(2)$, $M(4)$ et $M(8)$. Chacune de ces quatre stratégies est jouée contre les quatre stratégies ce qui fait bien seize simulations. A chaque pas de la partie, celle qui obtient le meilleur score est choisie pour décider du déplacement des unités.

$Nearest$	$M(2)$	$M(4)$	$M(8)$	$MC(100)$	$Select$	$StaticSelect$	$Rush$
64	73	65	49	70	92	63	36

Tableau 8.1. *Tableau des scores cumulés des stratégies*

Le tableau 8.1 donne le nombre total de parties gagnées par chaque stratégie. La meilleure stratégie est Select avec quatre-vingt-douze victoires sur un total de cent vingt parties.

La plus mauvaise est Rush qui n'obtient que trente-six victoires contre les autres stratégies. On peut aussi observer que $M(2)$ est une stratégie particulièrement adaptée aux configurations de jeu que nous avons décrites, puisqu'elle arrive en deuxième position ce qui est remarquable pour une stratégie simple et peu gourmande en temps de calcul. Presque à égalité se trouve $MC(100)$ qui domine d'assez peu $M(4)$ et Nearest. La stratégie $M(8)$ obtient un assez mauvais score et termine avant dernière, probablement parce que les configurations testées imposent des batailles décisives rapides et sont assez peu portées sur le long terme et la construction d'une grande armée qui nécessite beaucoup de métal.

Le tableau 8.2 détaille les résultats des confrontations directes entre chaque stratégie. Chaque case du tableau donne le nombre de parties gagnées par une stratégie contre une autre sur l'ensemble des configurations testées. Par exemple, le premier nombre de la troisième ligne qui vaut six, correspond au nombre de victoires de $M(4)$ contre Nearest. Chaque ligne du tableau correspond à une stratégie, et donne le nombre de ses victoires pour chaque confrontation avec une autre stratégie.

Il y a seize confrontations possibles entre deux stratégies (deux cartes possibles, fois deux configurations de sources de métal, fois deux configurations initiales de nombre d'unités, fois deux couleurs possibles). La somme d'un chiffre du tableau et de son symétrique par rapport à la diagonale vaut donc toujours seize (il n'y a pas de partie nulle, si une partie dépasse le seuil de pas fixé à mille, le gagnant est celui dont la somme des santés de ses unités est la plus grande).

$Algorithme$	$Nearest$	$M(2)$	$M(4)$	$M(8)$	$MC(100)$	$Select$	$StaticSelect$	$Rush$
$Nearest$	8	8	10	10	1	4	7	16
$M(2)$	8	8	10	12	9	4	9	13
$M(4)$	6	6	8	11	13	6	8	7
$M(8)$	6	4	5	8	9	4	6	7
$MC(100)$	15	7	3	7	8	4	14	12
$Select$	12	12	10	12	12	8	13	13
$StaticSelect$	9	7	8	10	2	3	8	16
$Rush$	0	3	9	9	4	3	0	8

Tableau 8.2. *Tableau des confrontations entre stratégies*

On peut mieux analyser les comportements des stratégies les unes par rapport aux autres avec le tableau 8.2 qu'avec le tableau 8.1. Par exemple, on voit que $MC(100)$ domine largement Nearest (quinze victoires sur seize) alors qu'elle est dominée par $M(2)$, $M(4)$ et $M(6)$ (de trois à sept victoires sur seize).

On voit aussi que le comportement de $MC(100)$ face à StaticSelect (quatorze victoires) est bien meilleur que face à Select (quatre victoires).

La stratégie Select a des résultats bons et ce qui est important, uniformes par rapport aux autres stratégies. Le nombre de victoires varie de dix à treize (sans compter le huit quand elle joue contre elle même, mais c'est une résultat qui par symétrie ne peut pas prendre une autre valeur).

Concernant la stratégie Rush, on note qu'elle est souvent mauvaise, sauf contre les stratégies qui privilégient le développement à long terme comme $M(4)$ et $M(8)$.

StaticSelect se comporte plutôt bien contre les stratégies statiques, mais a de mauvais scores contre $MC(100)$ et Select (deux victoires et trois victoires). StaticSelect ne s'adapte qu'à la configuration initiale d'une partie, alors que $MC(100)$ et Select s'adaptent en permanence à la configuration courante. On voit bien ici l'avantage d'une adaptation dynamique.

La stratégie Nearest se comporte plutôt bien contre les autres stratégies statiques. Elle gagne même plus de la moitié de ses parties contre $M(4)$ et $M(8)$, ce qui est bien pour une stratégie qui est plus simple à programmer. En revanche, elle obtient de mauvais résultats contre les stratégies dynamiques.

La stratégie $M(2)$ se comporte bien face à toutes les autres stratégies sauf Select (seulement quatre victoires sur seize). C'est une stratégie statique qui semble bien adaptée aux configurations de jeu testées. Toutefois pour un jeu plus complexe il n'est pas sûr qu'elle continue à se comporter aussi bien.

8.8. Conclusion

Nous avons montré que pour un jeu de stratégie temps réel simple, des stratégies adaptatives peuvent mieux se comporter que des stratégies statiques.

Nous avons présenté une évolution des stratégies Monte-Carlo vers des stratégies de sélection dynamique de stratégies statiques. Les stratégies ont été testées dans un tournoi en les faisant jouer les unes contre les autres. Les résultats expérimentaux montrent une domination de la stratégie de sélection qui s'adapte à chaque pas de la partie en cours.

8.9. Bibliographie

[ABR 90] ABRAMSON B., « Expected-outcome : a general model of static evaluation », *IEEE Transactions on PAMI*, vol. 12, n°2, p. 182-193, 1990.

[BIL 02] BILLINGS D., DAVIDSON A., SCHAEFFER J., SZAFRON D., « The Challenge of Poker », *Artificial Intelligence*, vol. 134, n°1-2, p. 210-240, 2002.

[BOU 06] BOUZY B., CHASLOT G., « Expériences d'apprentissage par renforcement dans une architecture Monte-Carlo Go », CAZENAVE T., Ed., *Intelligence Artificielle et Jeux*, Hermes Science, 2006.

[BRU 93] BRUEGMANN B., Monte Carlo Go, www.joy.ne.jp/welcome/igs/Go/computer/-mcgo.tex.Z, 1993.

[CAZ 04] CAZENAVE T., « Monte-Carlo Real-Time Strategy », *CGAIDE 2004*, Reading, UK, p. 298-298, 2004.

[CAZ 05a] CAZENAVE T., « A Phantom Go program », *Advances in Computer Games 11*, Taipei, Taiwan, 2005.

[CAZ 05b] CAZENAVE T., HELMSTETTER B., « Combining tactical search and Monte-Carlo in the game of Go », *CIG'05*, Colchester, UK, p. 171-175, 2005.

[CHU 05] CHUNG M., BURO M., SCHAEFFER J., « Monte Carlo Planning in RTS Games », *CIG 2005*, Colchester, UK, p. 117-124, 2005.

[COU 06] COULOM R., « Efficient Selectivity and Back-up Operators in Monte-Carlo Tree Search », *Proceedings Computers and Games 2006*, 2006.

[GIN 99] GINSBERG M. L., « GIB : Steps toward an Expert-Level Bridge-Playing Program », *IJCAI-99*, Stockholm, Sweden, p. 584-589, 1999.

[JUI 99] JUILLÉ H., Methods for statistical inference : extending the evolutionary computation paradigm, Phd thesis, Brandeis University, Department of Computer Science, May 1999.

[JUN 99] JUNGHANNS A., Pushing the Limits : New Developments in Single-Agent Search, Phd thesis, University of Alberta, Department of Computing Science, 1999.

[PON 05] PONSEN M. J., MUÑOZ-AVILA H., SPRONCK P., AHA D. W., « Automatically Acquiring Adaptive Real-Time Strategy Game Opponents Using Evolutionary Learning », *The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference*, AAAI Press, p. 1535-1540, 2005.

[SHE 04] SHEPPARD B., « Efficient control of selective simulations », *ICGA Journal*, vol. 27, n°2, p. 67-80, June 2004.

[SPR 06] SPRONCK P., PONSEN M., SPRINKHUIZEN-KUYPER I., POSTMA E., « Adaptive Game AI with Dynamic Scripting », *Machine Learning*, vol. 63, n°3, p. 217-248, 2006.