Ceci est un extrait électronique d'une publication de Diamond Editions :

http://www.ed-diamond.com

Retrouvez sur le site tous les anciens numéros en vente par correspondance ainsi que les tarifs d'abonnement.

Pour vous tenir au courant de l'actualité du magazine, visitez :

http://www.gnulinuxmag.com

Ainsi que :

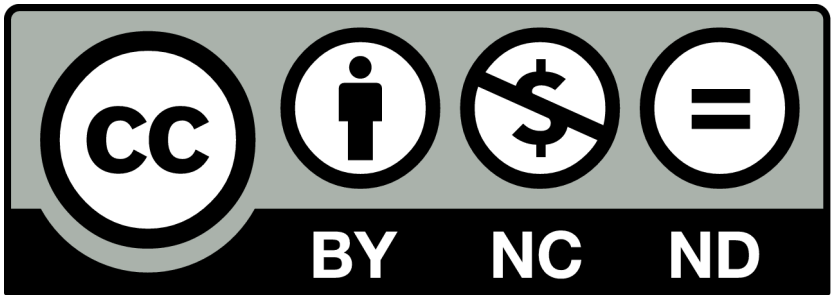
http://www.linux-pratique.com

et

http://www.miscmag.com

Ceci est un extrait électronique d'une publication de Diamond Editions

http://www.ed-diamond.com



Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification

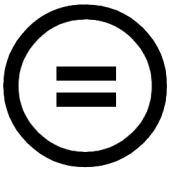
2.0 France

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public.

Paternité. Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'oeuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'oeuvre).

Pas d'Utilisation Commerciale. Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.



Pas de Modification. Vous n'avez pas le droit de modifier, de transformer ou d'adapter cette création.

A chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition.

- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits.
- Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Ceci est le Résumé Explicatif du Code Juridique. La version intégrale du contrat est attachée en fin de document et disponible sur :

http://creativecommons.org/licenses/by-nc-nd/2.0/fr/legalcode

L'**algorithme A*** ou comment **calculer** rapidement un chemin **optimal**

Dans le précédent article de la rubrique I.A. [1], j'ai décrit en détail l'algorithme de Dijkstra qui permet de calculer relativement efficacement un chemin optimal dans un graphe. J'ai aussi expliqué comment appliquer cet algorithme au problème du calcul du chemin optimal d'une unité dans un jeu de stratégie temps réel (STR). Nous avons vu en conclusion de l'article que, malgré une implémentation soignée, les temps de calcul théoriques restaient beaucoup trop élevés pour une utilisation en temps réel sur un terrain de jeu étendu et en présence de nombreuses unités.

Une modification subtile et astucieuse de l'algorithme de Dijkstra permet cependant de réduire de façon importante son coût d'exécution. L'algorithme modifié porte le nom d'algorithme A*. Il est massivement utilisé dans les jeux modernes, à tel point que certains spécialistes considèrent que le problème de base du *pathfinding* est réglé (on utilise A*), seules les subtilités (réglages de l'algorithme, choix des structures de données, etc.) restant à traiter pour implémenter un jeu.

Le but du présent article est de décrire l'algorithme A* en illustrant ses performances et en montrant comment les gains de temps qu'il induit (par rapport à l'algorithme de Dijkstra) permettent d'envisager des applications plus ambitieuses. Attention, l'article considère comme acquis les développements de [1] : je vous conseille donc vivement de relire l'article en question !

Les jeux vidéo de stratégie temps réel modernes ont des exigences très élevées en termes de rendu graphique et d'intelligence artificielle (I.A.). Le joueur attend un jeu en 3D construit à partir de milliers de polygones, pour le plaisir de l'œil, mais il veut aussi un adversaire d'un bon niveau, pour le plaisir de l'esprit. La brique de base de l'I.A., le pathfinding, c'est-à-dire le calcul des chemins suivis par les unités, doit donc être rapide et donner de bons résultats. L'algorithme de Dijkstra, présenté dans Linux Mag 53, garantit des résultats exacts, mais est en général trop lent pour fonctionner dans des conditions stressantes (un grand nombre d'unités, un terrain de jeu très grand, etc.). Cependant, des modifications minimes mais très subtiles permettent de conserver l'exactitude des chemins obtenus tout en accélérant notablement leur calcul. L'algorithme obtenu porte le nom d'algorithme A*.

Pourquoi l'algorithme de Dijkstra est-il lent ?

Un exemple de lenteur

Même si l'algorithme de Dijkstra fonctionne correctement, il donne parfois des résultats décevants. Bien entendu, Dijkstra a démontré mathématiquement l'exactitude de l'algorithme : les chemins obtenus sont toujours optimaux. Par contre, l'algorithme met parfois beaucoup de temps à trouver un chemin évident. Pour le voir, considérons un exemple très simple, celui d'une carte vide dans laquelle chaque case possède un coût de franchissement identique (égal à 1, par exemple). Dans ce cas, le chemin optimal est bien entendu la ligne droite qui relie le point de départ au point d'arrivée, comme le montre l'exemple de la figure 1 (le chemin est donné par la flèche noire).

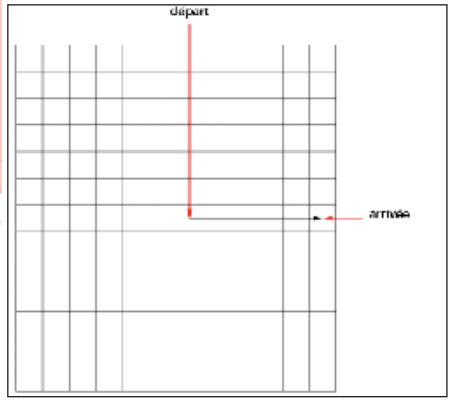


Figure 1

Le problème est que pour arriver à ce chemin optimal, l'algorithme de Dijkstra utilise beaucoup de ressources, comme l'illustre la figure 2. On constate en effet que presque toutes les cases de la carte ont été explorées (elles sont presque toutes dans la liste Fermés), alors qu'intuitivement, il est inutile d'étudier les cases qui éloignent du but !

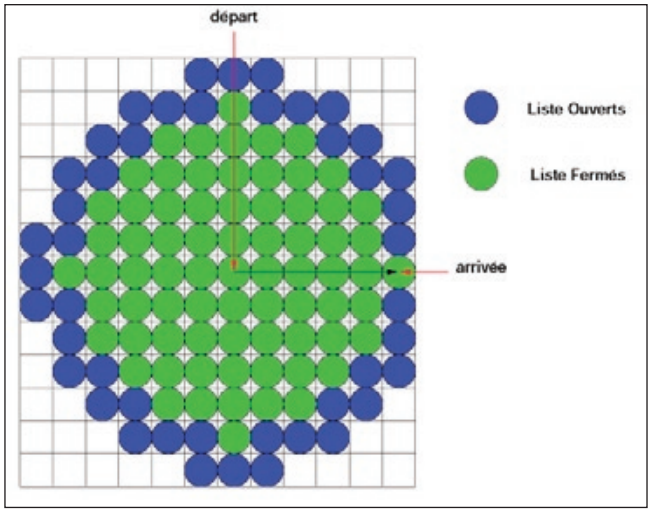


Figure 2

Le comportement de l'algorithme de Dijkstra est pourtant complètement correct. En effet, la carte est traduite en graphe et l'algorithme ne sait pas que certaines cases peuvent le rapprocher du but alors que d'autres l'en éloignent. En fait, l'exploration se fait selon deux critères :

1. Le prochain nœud exploré est celui qui possède le coût cumulé le plus faible dans la liste Ouverts : ce coût cumulé est en quelque sorte la distance entre le point de départ et la case courante. On explore donc la carte en s'éloignant progressivement du point de départ, sans tenir compte de la distance au but.

2. Pour placer de nouveaux nœuds dans la liste Ouverts, l'algorithme étudie les voisins directs du nœud exploré. Encore une fois, la distance au but n'est pas prise en compte. De même, la structure de graphe très particulière de la carte n'est pas exploitée.

Tout ceci fait que l'algorithme de Dijkstra peut être lent, car il explore un grand nombre de nœuds du graphe. Or, l'analyse algorithmique évoquée dans [1] montre que le coût du calcul croît rapidement avec le nombre de cases explorées.

Comment aider l'algorithme ?

On constate que le principal problème de l'algorithme de Dijkstra vient du fait qu'il ne tient pas compte de la distance au but pour choisir le prochain nœud à étudier. En fait, cela est parfaitement normal, car calculer la véritable distance au but est justement ce que fait l'algorithme (indirectement). En effet, si on connaît la distance entre tout point de la carte et la case d'arrivée souhaitée, autrement dit, si on connaît pour tout point de la carte le coût du chemin optimal le reliant à la case d'arrivée, le calcul d'un tel chemin optimal est très simple. Il suffit d'appliquer l'algorithme suivant :

1. point courant = point de départ

2. chercher dans les voisins du point courant le plus proche du point d'arrivée noté pp

3. point courant = pp

4. si le point courant n'est pas le point d'arrivée, retourner en 2

Cet algorithme est extrêmement efficace puisque le temps de calcul est proportionnel à la longueur du chemin. En fait, on ne peut pas faire mieux, mais comme il faut faire tourner un autre algorithme pour obtenir les distances entre tous les points de la carte et le point d'arrivée souhaité, cela n'a pas beaucoup d'intérêt. Notons que l'algorithme de Dijkstra calcule en fait la distance entre le point de départ et (presque) tous les points de la carte (c'est le coût cumulé), ce qui permet de déterminer le chemin optimal avec un algorithme similaire à celui qui vient d'être présenté : la principale différence est qu'on construit le chemin à l'envers, en partant du point d'arrivée et en remontant par les points qui sont les plus proches du point de départ.

On doit donc oublier l'idée de modifier l'algorithme de Dijkstra pour qu'il tienne compte de la véritable distance entre un nœud et le but. Cependant, en utilisant notre connaissance de la structure de la carte, il est possible de donner une estimation de cette distance. Pour la carte vide proposée au-dessus, il suffit par exemple de calculer la distance (au sens classique du terme, c'est-à-dire la distance à vol d'oiseau, plus techniquement, la distance euclidienne) entre le point courant et le but. Bien entendu, la mise en place d'obstacles ou de zones avec des coûts de déplacement distincts rend l'estimation proposée assez fausse, mais ce n'est pas très important (nous verrons dans la suite du texte quelle propriété théorique doit être satisfaite par l'estimation). L'idée n'est pas d'appliquer l'algorithme basique proposé au-dessus, mais d'aider l'algorithme de Dijkstra en lui permettant d'étudier en priorité les nœuds qui le rapprochent du but.

L'algorithme A*

Énoncé

L'algorithme A* [2] est construit sur l'algorithme de Dijkstra associé à une fonction qui estime la distance entre un nœud quelconque du graphe et le nœud cible (par distance, on entend bien sûr le coût du chemin optimal reliant le nœud concerné et le nœud cible). L'idée fondamentale est de modifier le classement dans la liste Ouverts afin de favoriser les nœuds les plus proches du but (au sens de l'estimation).

On suppose donc donnée une fonction `h` qui à un nœud `n` associe `h(n)`, une estimation du coût du chemin optimal

de n au nœud d'arrivée. Cette fonction s'appelle l'heuristique associée à l'algorithme. On calcule alors un chemin optimal de la façon suivante :

1. créer deux listes vides Ouverts et Fermés

2. insérer dans la liste Ouverts le quadruplet formé du nœud de départ, de la valeur h(départ), d'une valeur arbitraire (null par exemple) comme nœud parent et de la valeur 0 comme coût cumulé, soit donc (départ,h(départ),null,0). Le deuxième élément du quadruplet est une estimation du coût du chemin optimal partant du point de départ et se terminant au nœud arrivée en passant par le premier élément du quadruplet. Cette estimation est obtenue comme la somme du coût cumulé (coût du chemin optimal entre le nœud de départ et le nœud courant) et de l'estimation du coût du chemin optimal reliant le nœud courant au nœud cible

3. tant que la liste Ouverts n'est pas vide :

   a. chercher dans Ouverts le quadruplet (N,f,p,c) dont le coût estimé (f) est minimal (parmi les quadruplets contenus dans la liste)

   b. ajouter le quadruplet à la liste Fermés (on peut se contenter du triplet (N,p,c))

   c. si N est le nœud d'arrivée, quitter la boucle et passer à l'étape 4

   d. supprimer le quadruplet de la liste Ouverts

   e. pour chaque successeur S de N (c'est-à-dire chaque nœud S pour lequel il existe une arête de N à S dans le graphe) :

      i. calculer c' comme la somme de c et du coût associé à l'arête qui relie N à S

      ii. si S est dans Fermés :

         1. si le coût cumulé de S dans Fermés est supérieur à c', supprimer S de Fermés et ajouter le quadruplet (S,c'+h(S),N,c') à Ouverts

         2. sinon passer au successeur suivant

      iii. si S est dans Ouverts avec un coût cumulé supérieur à c', mettre à jour le coût cumulé, le nœud parent (qui est maintenant N) et le coût estimé (qui devient c'+h(S))

      iv. sinon ajouter à Ouverts le quadruplet (S,c',N)

4. si le nœud d'arrivée n'est pas dans Fermés, terminer l'algorithme sur un échec : il n'existe pas de chemin depuis le nœud de départ vers le nœud d'arrivée

5. sinon, reconstruire le chemin en suivant l'information parent dans la liste Fermés

Comparaison avec l'algorithme de Dijkstra

A première vue, les deux algorithmes sont très proches. Il est donc important d'insister sur les deux différences fondamentales :

1. Le classement des nœuds dans la liste Ouverts n'est pas réalisé selon le même critère. En effet, dans l'algorithme de Dijkstra, le prochain nœud exploré est le plus proche du point de départ, ce qui explique le comportement décevant de l'algorithme dans certains cas. Par contre, A* choisit d'explorer le nœud par lequel passe le chemin le plus court menant du point de départ au point d'arrivée. Bien entendu, le coût exact du chemin n'est pas connu, mais grâce à l'heuristique, on possède une bonne estimation de celui-ci. A* favorise donc les nœuds qui mènent au but.

2. Dans l'algorithme de Dijkstra, un nœud placé dans Fermés n'est jamais réexaminé. L'introduction d'un ordre différent pour l'examen des nœuds fait que A* étudie parfois plusieurs fois un même nœud. En effet, il arrive qu'on explore un nœud sans connaître le chemin optimal qui y mène depuis le nœud de départ car les nœuds qui le constituent sont mal "notés" par l'heuristique. De ce fait, le coût cumulé du nœud est faux lors de la première exploration. On doit donc stocker dans Fermés le coût cumulé de chaque nœud afin de pouvoir éventuellement réexaminer les nœuds.

L'implémentation de A* ne pose pas de problème particulier par rapport à celle de l'algorithme de Dijkstra. Les algorithmes sont tellement proches que la traduction d'une implémentation de celui de Dijkstra en A* ne prend que quelques minutes. Les structures de données utilisées sont pratiquement les mêmes et les remarques présentées dans [1] restent valables. Il est donc judicieux d'implémenter Ouverts sous forme d'un tas combiné à des annotations dans la carte. Fermés s'implémente aussi très efficacement et facilement sous forme d'annotation dans la carte. La seule véritable différence avec les solutions présentées dans [1] est que la liste Fermés doit maintenant contenir le coût cumulé de chaque nœud parcouru. L'occupation mémoire de l'algorithme A* est donc potentiellement plus importante que celle de l'algorithme de Dijkstra. Notons qu'il faut aussi pouvoir supprimer un nœud de la liste Fermés, ce qui peut être coûteux dans certaines implémentations (mais pas dans celle proposée dans [1]).

Exemples de mise en œuvre

Pour bien comprendre le fonctionnement de A*, étudions son comportement sur l'exemple de la carte vide proposée dans la section précédente. Pour cette carte, nous choisissons comme heuristique la distance euclidienne entre le centre d'une case et le centre de la case cible. Celle-ci est calculée simplement grâce à la formule suivante : $\sqrt{(x-a)^2+(y-b)^2}$. Dans cette formule, x et y sont les coordonnées du centre de la case considérée, alors que a et b sont celles de la case cible. Dans les deux algorithmes, on commence par insérer le point de départ dans la liste Ouverts. Le coût cumulé est nul et donc, pour A*, le classement se fait à partir de la distance entre le point de départ et le point d'arrivée (5 dans notre exemple). Comme dans les deux algorithmes, la liste Ouverts ne contient qu'un élément, la différence de critère de classement ne

change rien et on commence donc par étudier les voisins de la case de départ.

Les différences entre les algorithmes apparaissent dès l'étude des voisins. En effet, comme l'illustre la figure 3, les coûts estimés changent radicalement la donne. Sur cette figure, une case contient un nombre en bleu qui est le coût cumulé pour passer optimalement de la case de départ à celle-ci, ainsi qu'un nombre en rouge qui indique le coût estimé pour aller optimalement de la case au but. Pour l'algorithme de Dijkstra, les quatre voisines directes (situées horizontalement et verticalement) de la case de départ sont équivalentes (coût cumulé 1) et l'algorithme étudiera donc une d'entre elles plus ou moins au hasard. Au contraire, A* range les cases selon la somme du coût cumulé et de l'estimation du coût pour aller au but. Les valeurs approchées indiquées sur la figure permettent de voir que la prochaine case étudiée est celle qui est située juste à droite de la case de départ car son coût total est de 5, alors que les autres voisines non diagonales ont des coûts plus élevés (environ 6,10 et 7), de même d'ailleurs que les voisines situées en diagonales (coûts approximatifs 5,5 et 7,5).

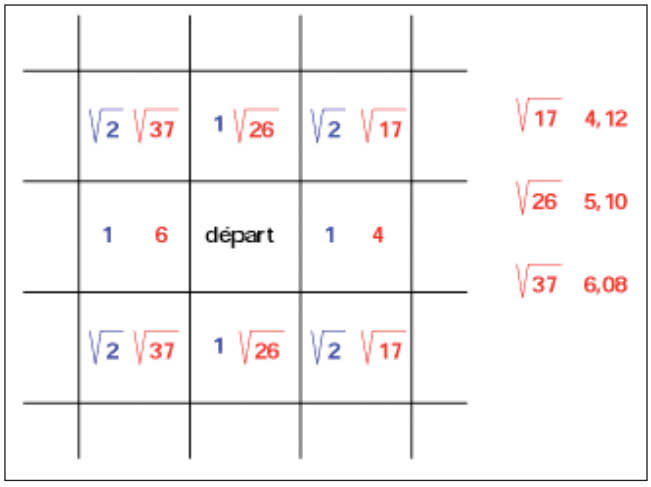


Figure 3

Les différences s'accentuent dès la deuxième case étudiée. Soyons en effet généreux avec Dijkstra et considérons que l'algorithme choisit lui aussi la case située juste à droite de la case de départ. La figure 4 donne les coûts cumulés et les coûts estimés obtenus. Quand on étudie les coûts cumulés, on constate que l'algorithme de Dijkstra va nécessairement étudier les trois autres cases de coût cumulé 1 avant de passer à des cases proches du but.

En effet, les trois nouvelles cases explorées ont toutes un coût cumulé supérieur à 1, ce qui est normal car on s'éloigne du point de départ. Au contraire, on constate que pour A* les choses se présentent sous un bien meilleur jour. En effet, la case située directement à droite de la nouvelle case possède un coût estimé de 5 (2+3), ce qui en fait la nouvelle case prioritaire dans la liste Ouverts. De ce fait, la troisième case étudiée par A* le rapproche encore du but.

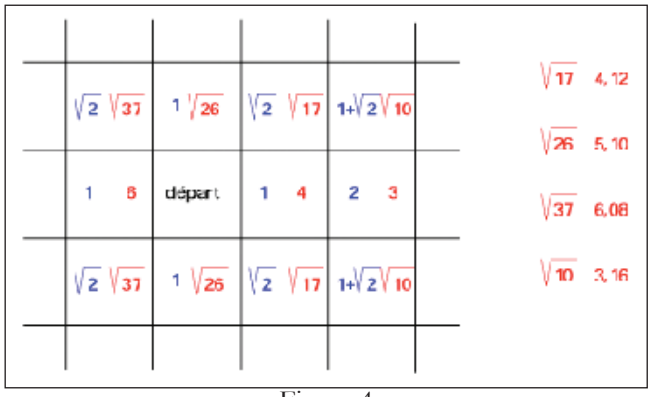


Figure 4

Globalement, la tendance d'A* à se rapprocher du but en priorité donne un résultat très satisfaisant illustré par la figure 5. La comparaison avec la figure 2 est très parlante : l'algorithme de Dijkstra a étudié 71 nœuds alors que A* n'en a étudié que 6 !

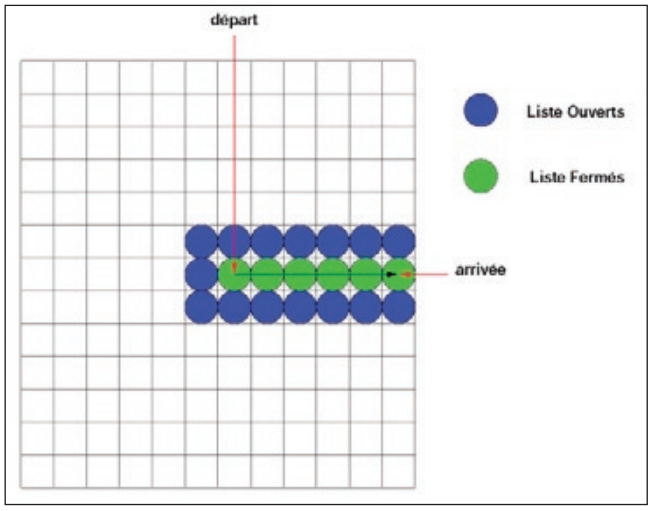


Figure 5

Bien entendu, j'ai choisi un exemple qui est particulièrement favorable à A*, mais en général, il est toujours plus efficace que Dijkstra. En fait, on peut démontrer que si l'heuristique vérifie une propriété théorique que nous verrons dans la section suivante, A* est un algorithme optimal, au sens où tout nœud exploré par A* doit être exploré par tout algorithme capable de trouver un chemin optimal (à condition que l'algorithme n'utilise pas d'autres informations que le graphe et l'heuristique).

La marge de manœuvre réside donc dans le choix de l'heuristique et dans l'implémentation proprement dite. Cette dernière doit être particulièrement soignée pour éviter tout surcoût par rapport à l'algorithme de Dijkstra. Dans certaines situations, les deux algorithmes explorent presque le même nombre de nœuds.

Considérons par exemple la figure 6 qui correspond à une carte de coût uniforme dans laquelle on a ajouté un obstacle infranchissable. On constate que l'algorithme de Dijkstra explore toutes les cases de la carte pour obtenir le chemin optimal.

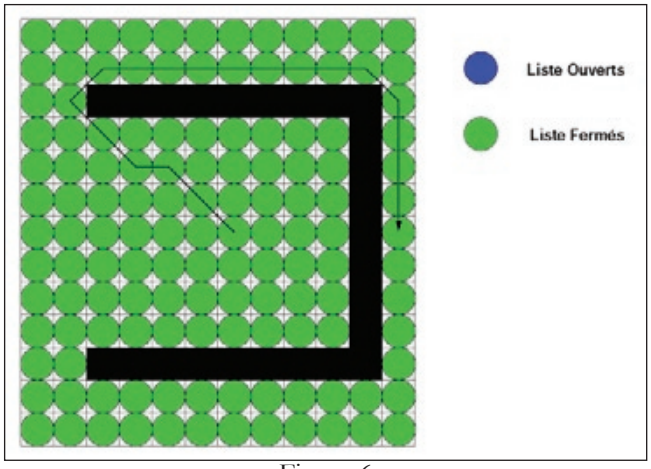


Figure 6

La figure 7 montre que dans une telle situation, A* ne gagne presque rien car il évite d'explorer seulement 10 cases sur 130. Il suffit donc que l'implémentation de A* soit seulement 10% plus lente que celle de l'algorithme de Dijkstra pour que ce dernier prenne l'avantage. Nous remarquons au passage que les chemins trouvés par les deux algorithmes sont différents, même s'ils sont touts deux optimaux. Ce comportement est assez fréquent, car l'heuristique, en modifiant l'ordre d'exploration, change aussi la structure de parents qui est utilisée pour construire le chemin optimal.

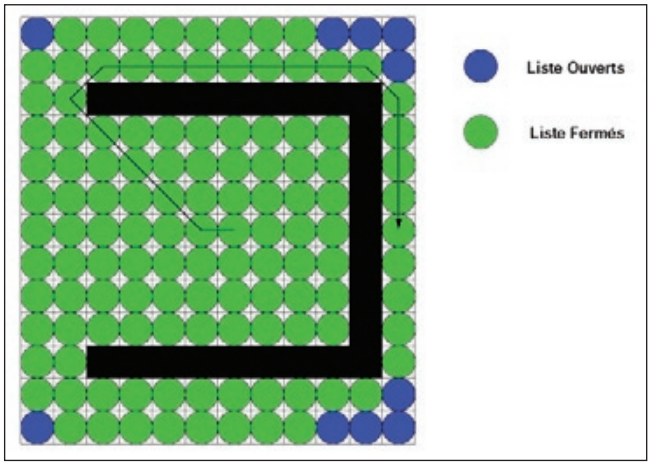


Figure 7

Il est important de noter que A* n'est pas un algorithme magique. Il est fréquent que les développeurs se plaignent de l'algorithme de Dijkstra et demandent comment passer à A*.

Pourtant, le premier réflexe devrait être de vérifier l'implémentation de Dijkstra, car l'utilisation de structures de données inadaptées (en particulier une liste classique pour la liste Ouverts) rend l'algorithme très lent. Or, cette lenteur se retrouvera sur A*...

Une implémentation ?

Il existe de nombreuses implémentations de l'algorithme A* disponibles sur Internet. Voici quelques pistes pour le lecteur curieux :

- Charité bien ordonnée : victime du syndrome du "non inventé ici", j'ai réalisé ma propre implémentation pédagogique en Java, sous licence GPL, disponible à l'URL **http://apiacoa.org/software/pathfinding**. L'intérêt de cette implémentation est de permettre une comparaison simple entre diverses stratégies, en faisant varier les heuristiques, les implémentations des listes Ouverts et Fermés, le modèle de coût de la carte, etc.

- Le moteur libre (GPL) Stratagus (**http://www.nongnu.org/stratagus/index.html**), dont j'ai déjà parlé dans [1], contient une implémentation en C de A* réalisé par votre serviteur, ainsi qu'une autre implémentation d'une version hiérarchique de A*, encore plus efficace, œuvre d'un certain Latimerius.

- Boson (**http://boson.eu.org**), lui aussi évoqué dans [1], contient une implémentation libre (GPL) en C++ d'un A* simple (les structures de données ne sont pas optimales).

- Heroic BattleField (**http://hbf.tuxfamily.org**), toujours évoqué dans [1], propose aussi une implémentation simple d'A* en C sous licence GPL.

On trouvera d'autres liens dans la section référence en fin d'article.

Du bon choix de l'heuristique

L'admissibilité

Pour que A* fonctionne correctement, il faut que l'heuristique qui estime le coût du chemin optimal d'un nœud quelconque au but soit admissible. En pratique, cela signifie que l'heuristique doit sous-estimer le coût du chemin optimal.

Dans ce cas, on montre que les chemins obtenus par A* sont bien optimaux. Au contraire, si l'heuristique n'est pas admissible et donc qu'elle surestime le coût de certains chemins, A* peut obtenir un chemin non optimal.

L'heuristique que nous avons utilisée dans la section précédente est admissible. En effet, elle calcule le coût comme la distance euclidienne entre les deux cases considérées.

Or, le coût du passage d'une case à une de ses voisines est égal à la distance (euclidienne) entre les centres des cases concernées (et ceci en tout point de la carte).

Donc, l'heuristique donne le coût exact pour le passage d'une case à une de ses huit voisines.

Quand le but est plus éloigné, l'heuristique sous-estime le coût. En effet, la distance euclidienne correspond à la distance "à vol d'oiseau", alors que le chemin optimal passe par les centres des cases et est donc en général plus long, comme l'illustre la figure 8.

De plus, l'introduction d'obstacles ne peut qu'allonger les chemins optimaux, alors que l'heuristique n'est pas modifiée. Elle reste donc admissible.

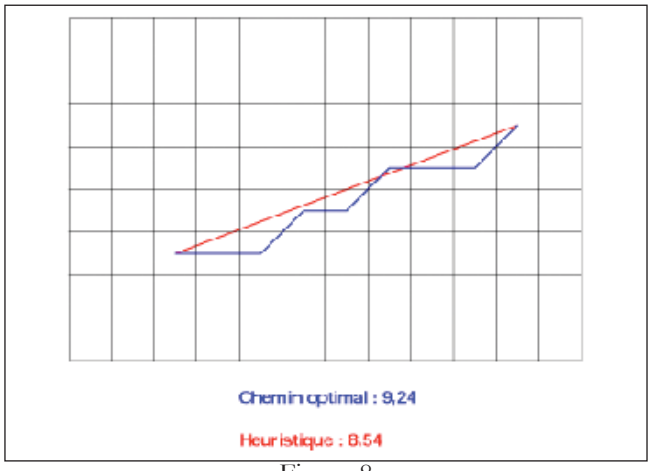


Figure 8

Notons qu'on peut améliorer l'heuristique en tenant compte du fait que les chemins passent par les centres des cases. On obtient la formule suivante : 1,4 * Min(|x-a|,|y-b|) + ||x-a|-|y-b||. Dans celle-ci, les barres verticales désignent la valeur absolue, alors que 1,4 est une approximation de la racine carrée de 2 (on peut d'ailleurs prendre une valeur plus précise). Le premier terme de la formule correspond en fait à la distance parcourue en diagonale, alors que le second correspond à la distance franchie horizontalement ou verticalement. Comme cette heuristique est plus précise que la distance euclidienne, elle améliore encore les performances de l'algorithme.

Sous-estimer...

En fait, les performances de A* dépendent fortement de la qualité de l'heuristique. Quand celle-ci donne une très bonne estimation du coût (par le bas, c'est-à-dire en sous-estimant), A* fonctionne très bien : non seulement il produit un chemin optimal, mais en plus il le fait vite, c'est-à-dire en étudiant peu de nœuds. Plus l'heuristique s'éloigne du coût optimal (toujours par le bas), plus l'algorithme devient lent. Le chemin calculé reste optimal, mais A* étudie beaucoup plus de nœuds. Le cas extrême est atteint quand l'heuristique est uniformément nulle (le coût estimé est alors toujours 0), car A* devient alors identique à l'algorithme de Dijkstra.

Nous avons déjà observé la dégradation de performance dans la section précédente. En effet, l'introduction d'obstacles dans la carte dégrade la qualité de l'heuristique qui sous-estime parfois fortement le coût du chemin optimal. Par exemple, le coût réel du chemin optimal entre le point de départ et le point d'arrivée sur la figure 7 est d'environ 21,5, alors que l'heuristique de la distance euclidienne estime ce coût à 5 !

Pour mieux comprendre l'effet de la sous-estimation, dégradons volontairement notre heuristique, par exemple en utilisant comme estimation la distance euclidienne divisée par 2. L'effet mathématique est bien entendu de changer le classement des nœuds dans la liste Ouverts : l'importance relative du coût cumulé augmente avec la diminution du coût calculé par l'heuristique. On se rapproche ainsi du classement utilisé par Dijkstra : A* explore donc plus facilement des nœuds qui l'éloignent du but. La figure 9 illustre les résultats de cette exploration moins focalisée. Quand on compare avec les figures 2 et 5, on constate qu'on obtient une configuration intermédiaire : A* a étudié 23 cases, contre 71 pour Dijkstra et 6 pour A* avec une heuristique meilleure.

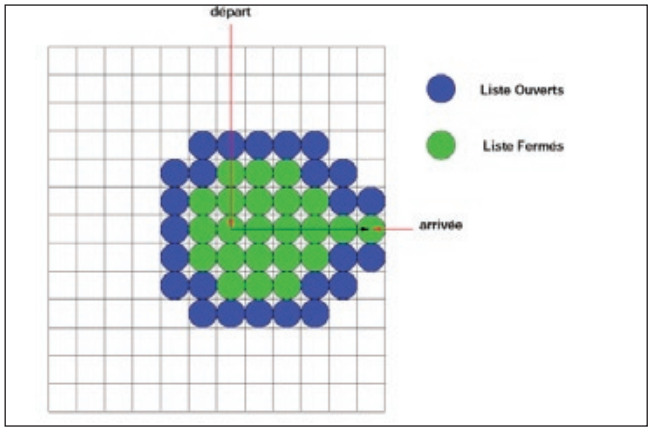


Figure 9

Il est donc important dans la pratique que l'heuristique sous-estime le moins possible les coûts. Pour ce faire, on doit être particulièrement attentif au modèle de mouvement choisi. En effet, dans [1], comme dans le début du présent article, nous avons choisi de prendre en compte la distance parcourue pour passer du centre d'une case au centre d'une de ses voisines afin de calculer le coût du déplacement. Or, dans certains jeux, la distance n'est pas prise en compte. Ainsi, un mouvement en diagonale est-il aussi coûteux qu'un autre horizontal ou vertical. Dans d'autres jeux, au contraire, les mouvements diagonaux sont interdits. Dans les deux cas, l'heuristique de la distance euclidienne n'est pas adaptée. En effet, elle surestime les coûts dans le cas où la distance n'est pas prise en compte (ce qui est très gênant, comme nous allons le voir dans la section suivante), et sous-estime les coûts quand les mouvements diagonaux sont interdits, comme l'illustre la figure 10 (le chemin bleu correspond aux déplacements diagonaux "rapides", alors que le chemin vert correspond aux mouvements diagonaux interdits).

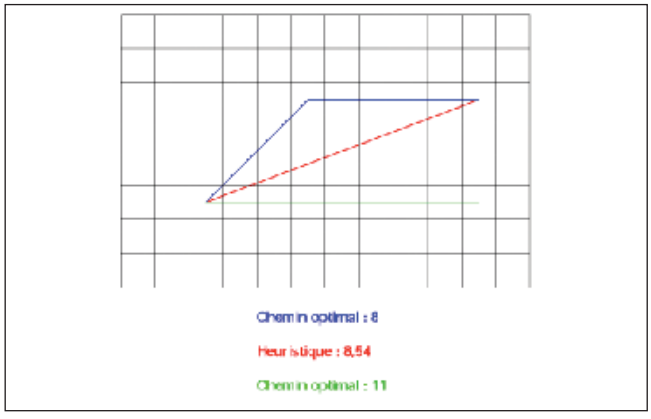


Figure 10

Quand on interdit les déplacements diagonaux, la bonne heuristique consiste à calculer la somme de |x-a|+|y-b| (il s'agit de la distance Manhattan). On constate que la distance obtenue correspond au nombre de colonnes et de lignes qui séparent la case de départ de la case d'arrivée (en comptant la ligne et la colonne de la case d'arrivée). Bien entendu, cette heuristique n'est valable que si le coût de franchissement d'une case est de 1 au minimum.

Quand on autorise les déplacements diagonaux sans les pénaliser, c'est-à-dire sans tenir compte de la distance réellement parcourue, on peut utiliser comme heuristique la formule suivante : max(|x-a|,|y-b|). Ceci correspond à compter le nombre de colonnes et de lignes qui séparent la case de départ de la case d'arrivée, puis à conserver le plus grand des deux nombres.

Ou surestimer ?

Nous venons donc de voir que la sous-estimation du coût d'un chemin ne change pas la correction de A* qui continue dans une telle situation à calculer un chemin optimal. Le seul effet pratique est le ralentissement de l'algorithme. La situation se complique en présence de surestimation. En effet, on ne peut plus garantir dans ce cas que l'algorithme A* produit des résultats corrects. Dans certaines situations, les chemins obtenus ne sont pas optimaux. Mais en payant ce prix (parfois exorbitant), on peut accélérer les calculs !

Étudions un exemple simple pour lequel l'algorithme ne fonctionne pas. Dans une carte de coût uniforme 1, on ajoute une route de coût 0,5. On conserve le modèle de voisinage classique dans lequel on tient compte de la distance parcourue pour calculer le coût du passage d'une case à une de ses voisines. L'heuristique adaptée est donc celle basée sur la distance euclidienne. Cependant, celle-ci n'est pas admissible. En effet, un chemin sur la route ne coûte que 0,5 fois la distance parcourue. Donc, une estimation du coût par la simple distance surestime celui-ci. La figure 11 donne le chemin optimal obtenu en utilisant l'heuristique euclidienne simple. Ce chemin a un coût de 7.

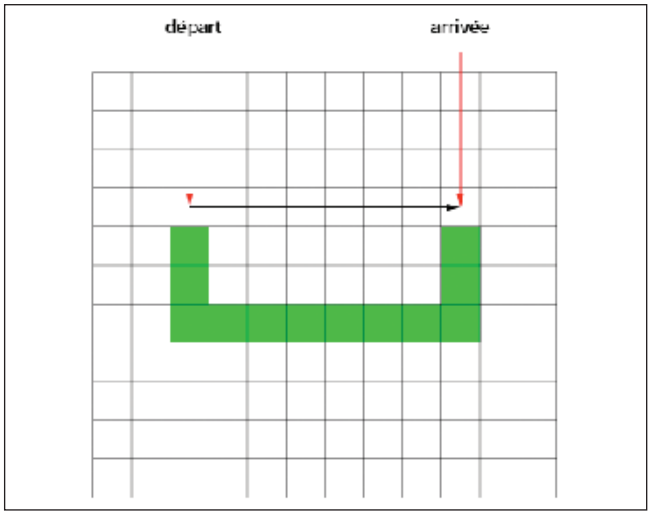


Figure 11

La figure 12 représente le chemin optimal qu'on obtient avec l'algorithme de Dijkstra ou encore avec A* et une heuristique admissible (par exemple 0,5 fois la distance euclidienne). Ce chemin a un coût de 6,4. En termes d'efficacité du calcul, l'heuristique "fausse" est particulièrement agréable car elle ne considère que 8 cases. L'heuristique admissible provoque l'analyse de 44 cases, alors que l'algorithme de Dijkstra étudie presque toute la carte (en fait 100 cases sur 132).

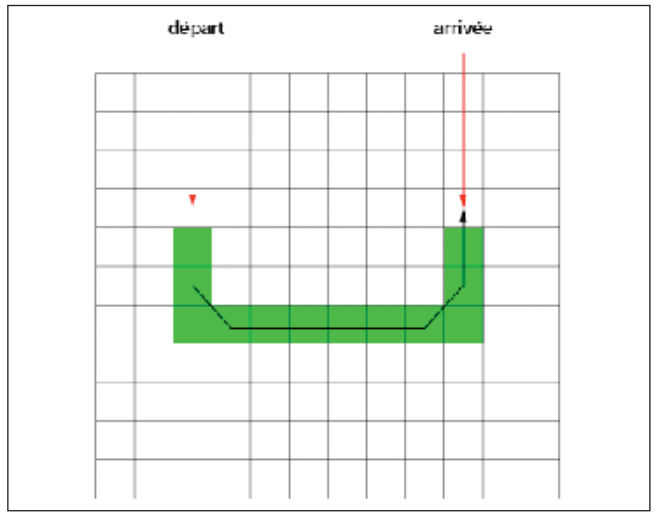


Figure 12

Il est assez difficile de prédire l'effet d'une surestimation du coût d'un chemin optimal. Sur l'exemple proposé, le chemin obtenu est totalement différent de celui cherché. On pourrait d'ailleurs faire en sorte que les chemins soient encore plus éloignés, en jouant avec les coûts de déplacement et la forme du tracé. Cependant, on constate que les coûts des deux chemins sont très proches. Or, en pratique, le tracé du chemin n'est pas toujours important, c'est avant tout le temps qu'on met à atteindre le but qui compte. Si l'utilisation d'une heuristique non admissible permet d'accélérer les calculs, on peut imaginer que l'unité va commencer son déplacement plus tôt que dans le cas d'un calcul lent et donc finalement arriver aussi vite à bon port qu'avec un chemin véritablement optimal. Le problème est que rien ne permet d'affirmer que l'algorithme ne va pas proposer un chemin totalement délirant quand il utilise une heuristique non admissible. C'est pourquoi leur utilisation doit être réservée à des cas bien maîtrisés.

Comment faire en pratique ?

La mise au point d'une heuristique efficace est un problème assez délicat. Je suggère donc de commencer par une heuristique simple, adaptée au modèle de déplacement choisi, comme la distance euclidienne ou la distance Manhattan. On peut ensuite utiliser diverses astuces pour améliorer l'heuristique et donc les performances de A* :

- Dans certaines situations, la prise en compte de caractéristiques particulières de la carte est un plus indéniable. Un exemple assez parlant est celui d'une carte divisée en deux zones, avec un seul point de passage entre

celles-ci. On pense par exemple à une île reliée au continent par un pont, comme sur la figure 13. Si on tente de calculer un chemin optimal entre le continent et l'île, la distance euclidienne sous-estime énormément le coût de ce dernier, car elle fonctionne à vol d'oiseau et utilise donc un chemin (en rouge sur la carte) qu'une unité ne pourra pas emprunter. Le vrai chemin optimal passe en fait nécessairement par l'unique point de passage (chemin vert sur la carte). Or, il est tout à fait possible de construire une heuristique qui tient compte de cette situation. L'idée est de déterminer dans quelle zone sont situés les points d'arrivée et de départ. S'ils sont dans une même zone, l'heuristique reste la distance euclidienne. Par contre, s'ils sont situés dans des zones différentes, l'heuristique évalue le coût comme la somme des distances au point de passage obligé entre les deux zones.

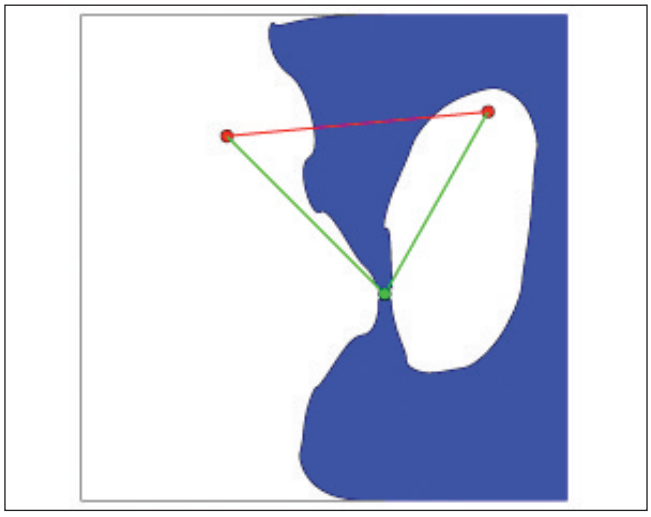


Figure 13

On peut montrer que cette heuristique améliore grandement les performances de A* qui étudie beaucoup moins de cases tout en trouvant toujours le chemin optimal. L'idée se généralise sans difficulté au cas de plusieurs zones, comme par exemple les pièces d'une maison. Cependant, il faut rester attentif au temps de calcul de l'heuristique. En effet, si le terrain étudié comporte de nombreuses zones et donc de nombreux points de passage, l'évaluation de l'heuristique devient un problème complexe, en fait c'est un calcul de chemin optimal entre les zones ! On peut donc perdre en temps de calcul de l'heuristique ce qu'on gagne en nombre de cases étudiées.

L'analyse du terrain de jeu est un domaine en plein essor (cf [3]) qui vise entre autres à produire automatiquement des aides à l'algorithme de pathfinding (et plus généralement à l'IA du jeu), en procédant par exemple à un découpage du terrain en zones qui permettent d'obtenir une heuristique efficace.

Dans d'autres situations, il est intéressant de jouer avec l'admissibilité pour obtenir à la fois un algorithme plus efficace et des effets maîtrisés sur les chemins obtenus. Pour cela, il faut bien comprendre l'effet d'une modification de l'heuristique. Nous savons qu'une heuristique admissible attire A* vers les cases qui le rapprochent du but, ce qui lui permet de l'atteindre plus rapidement. Si nous augmentons le coût estimé du chemin qui relie une case au but (sans changer l'estimation pour les autres cases), nous repoussons le moment où cette case sera explorée par l'algorithme. Au contraire, une réduction du coût favorise une exploration rapide de la case. Si on souhaite, par exemple, favoriser les lignes droites (car un chemin dans lequel une unité change en permanence de direction n'est pas très réaliste), on peut augmenter légèrement le coût estimé d'un chemin qui effectue un tournant (grâce au nœud parent conservé dans la liste Ouverts, on sait d'où le chemin optimal provient et on peut donc calculer l'angle de rotation entre la direction précédente et la direction induite par le passage par un voisin donné). La perte d'admissibilité induite par cette pénalisation a en général des effets complètement négligeables, à condition bien sûr de lui donner une très faible amplitude. On peut poursuivre dans cette voie en mettant en œuvre des adaptations de A* qui produisent des chemins plus réalistes, mais les solutions obtenues sont assez évoluées (cf [5]).

Utilisations avancées

La traduction du terrain en un graphe permet des applications évoluées qui dépassent le modèle que j'ai décrit jusqu'à présent. Il n'est pas question ici de faire une liste exhaustive des possibilités ouvertes par cette traduction (associée au gain algorithmique induit par A*), mais simplement de développer deux thèmes intéressants parmi tant d'autres.

Prise en compte de l'altitude

Le modèle de terrain que j'ai choisi est extrêmement simple et assez peu réaliste. On peut améliorer notablement ce point en prenant en compte l'altitude en plus de la nature de chaque case. Le coût du passage d'une case à une autre est ainsi obtenu comme une combinaison de deux facteurs. Comme dans le modèle simple, on tient compte de la difficulté de progression dans la case liée à la nature de celle-ci, mais on ajoute à ce coût une autre pénalité correspondant au changement d'altitude. Le point intéressant de cette pénalité est qu'elle n'est pas symétrique. En effet, passer de la case A à la case B ne coûte pas la même chose que passer de la case B à la case A. Pour s'en convaincre, il suffit d'observer les coureurs cyclistes dans une étape de montagne ! La vitesse de progression dans la montée est bien évidemment très faible par rapport à celle atteinte dans la descente. Pour plus de réalisme, les coûts peuvent d'ailleurs dépendre de la nature de l'unité considérée : un char est relativement lent, mais aussi assez peu sensible à la nature du terrain, alors qu'une voiture de sport est très rapide sur une piste et ne peut pas emprunter certains types de terrain.

L'algorithme de Dijkstra et A* s'adaptent parfaitement à ce nouveau modèle car ils travaillent sur des graphes orientés pour lesquels le coût de passage d'un nœud à un

autre est porté par l'arête. Le modèle choisi introduit donc 16 arêtes orientées entre chaque nœud au lieu de 8 arêtes non orientées. Du point de vue de l'implémentation, on ne change rien, excepté bien entendu le calcul du coût de passage d'une case à une autre.

Déplacements coordonnés

Dans de nombreux jeux RTS, les unités de base ne sont pas amphibies et doivent donc utiliser des transports de troupes pour traverser une zone de mer.

Or, il est rare que l'algorithme de pathfinding soit programmé pour réaliser cette opération automatiquement, même dans les cas simples. Pourtant, cela ne pose aucun problème théorique.

Considérons en effet le cas d'un fantassin situé sur le continent A qui souhaite se rendre sur le continent B et d'un transport de troupes situé quelque part en mer entre les deux continents. Le problème se décompose en trois phases :

1. déplacer le fantassin et le transport de sorte qu'ils se rencontrent sur les "plages" du continent A ;

2. déplacer le transport contenant le fantassin jusqu'à une "plage" du continent B ;

3. déplacer le fantassin sur le continent B vers son but.

Pour traiter ce problème comme une exploration de graphe, il suffit de considérer que chaque nœud correspond aux positions des deux unités sur le terrain. Une arête du graphe modélise le déplacement simultané des deux unités, ce qui signifie quelles seront nombreuses !

En effet, si on autorise les mouvements diagonaux, on a 9 mouvements possibles par unité (les 8 mouvements réels auxquels on ajoute l'immobilité) et donc 80 mouvements possibles par tour (soit autant d'arêtes), comme par exemple le mouvement simultané suivant : le fantassin se déplace d'une case vers le nord, alors que le transport se déplace d'une case vers le sud-ouest. On compte 80 mouvements au lieu de 81 car on interdit la combinaison dans laquelle les deux unités restent immobiles.

De même, les nœuds sont très nombreux, en théorie N^2 si N désigne le nombre de cases dans le terrain de jeu (en fait moins car certaines combinaisons sont interdites, en particulier quand le fantassin est seul en mer ou quand le transport se retrouve sur le continent). On comprend donc que l'algorithme de Dijkstra risque d'être très long à s'exécuter.

A* permet d'obtenir des résultats, mais il faut envisager pour cela une implémentation spécifique pour les listes Ouverts et Fermés car l'annotation du graphe n'est plus possible en raison de sa taille.

Il est aussi important de mettre en œuvre une heuristique efficace, par exemple en demandant dans la première phase aux deux unités de se rapprocher : ce comportement peut être obtenu en utilisant une technique proche de celle proposée dans la section précédente, quand on introduisait un point intermédiaire sur le pont entre l'île et le continent. On peut ainsi calculer le point `a` d'intersection entre la ligne qui joint les deux unités et la plage du continent.

Le coût estimé est alors la distance entre l'unité la plus éloignée du point `a` et ce dernier, à laquelle on ajoute la distance entre `a` et la cible du fantassin sur le continent `B`, comme l'illustre la figure 14 (à condition que la vitesse de déplacement des deux unités soit la même).

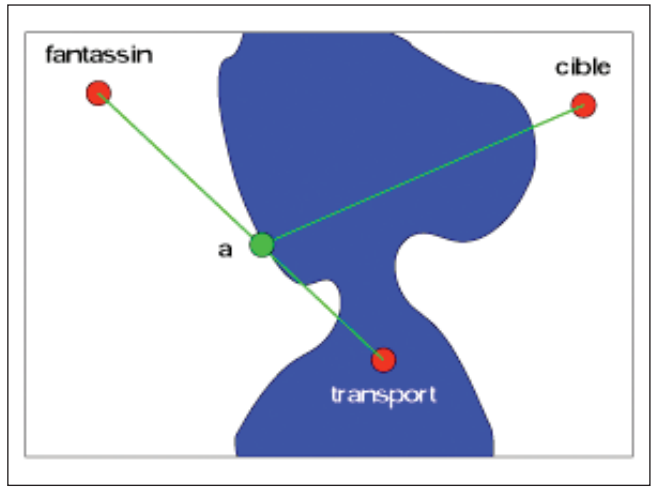


Figure 14

Grâce à la traduction en graphe, on obtient un algorithme qui calcule un chemin optimal mettant en jeu deux unités qui coordonnent leurs actions. L'apport d'A* est ici crucial car la taille du graphe interdit son exploration complète.

On peut même utiliser ici une heuristique non admissible car le plus important est d'obtenir un chemin, même s'il n'est pas optimal.

Notons pour conclure sur cet exemple que les grands graphes s'explorent aussi avec d'autres algorithmes, en particulier IDA* (cf [4]), moins efficaces en termes de nœuds explorés, mais très économes en mémoire.

Conclusion

L'algorithme A* présenté dans cet article améliore radicalement l'algorithme de Dijkstra et permet le calcul de chemins optimaux dans des graphes beaucoup plus volumineux.

Bien implémenté, c'est-à-dire associé à des structures de données efficaces et une heuristique adaptée au problème, A* apporte donc une solution satisfaisante au problème du pathfinding, à tel point que certains participants à la table ronde sur l'IA dans les jeux vidéo animée par Steven Woodcock [6] ont pu affirmer que les questions de base concernant le calcul de chemins optimaux avaient trouvé une réponse.

Cependant, de nombreuses questions restent ouvertes, comme par exemple :

- Comment construire une heuristique efficace ? Les

réponses sont à chercher dans l'analyse de terrain (cf [3]) et dans les approches hiérarchiques (cf [7]).

- Comment travailler sur des cartes ou des graphes gigantesques ? Les réponses sont aussi à chercher dans les approches hiérarchiques [7] ou dans les méthodes multi-résolutions (comme celle utilisée par Stratagus).

- Comment prendre en compte l'aspect dynamique du terrain ? Des solutions existent comme l'algorithme D* (cf [8]), mais elles sont assez coûteuses en termes de stockage et les développeurs préfèrent en général des solutions très simples, comme par exemple un découpage du chemin optimal en tronçon, avec un recalcul systématique (i.e., après chaque mouvement) du tronçon le plus proche.

- Comment résoudre les problèmes de collisions entre unités, ou plus généralement les problèmes de mouvements coordonnées (les formations par exemple) ? Diverses solutions ont été utilisées, comme par exemple celles décrites dans [9] et [10], qui consistent à calculer un chemin optimal classique, puis à l'exécuter astucieusement grâce à des primitives de modification de la formation associées à une gestion des priorités entre les unités pour éviter les inter-blocages. Comme nous l'avons vu plus haut, il est aussi possible de formuler le problème comme une exploration de graphe, mais cette technique pose des problèmes spécifiques. D'autres solutions, inspirées de l'intelligence collective, ont aussi été proposées pour réaliser des mouvements coordonnées (cf [11]).

Le domaine du pathfinding reste donc en perpétuelle évolution, poussé, comme toute l'industrie du jeu vidéo, par les exigences croissantes des joueurs, en particulier en matière de réalisme.

Les jeux vidéo libres ont encore du mal à concurrencer les jeux commerciaux, en particulier en termes de rendu graphique. Pourtant, l'expérience ludique est un tout et l'IA en général, et le pathfinding en particulier, sont souvent des éléments en retrait dans les jeux commerciaux.

Les jeux libres pourraient donc profiter de la richesse de la recherche dans ce domaine pour marcher sur les plates-bandes de leurs concurrents à gros budget.

Fabrice Rossi
Equipe AXIS
INRIA
Fabrice.Rossi@apiacoa.org
http://apiacoa.org

Post Scriptum

Une petite erreur s'est malencontreusement glissée dans l'article précédent [1] : les figures 2 et 10 ont subrepticement échangé leurs places !

Références

Certaines références en ligne proviennent de l'excellent site http://www.gamasutra.com. Ce site consacré à l'industrie du jeu vidéo est une mine d'informations, mais son accès demande une inscription (gratuite).

[1] Fabrice Rossi - Recherche des chemins optimaux pour les jeux vidéo, GNU/Linux Magazine France, No 53, Septembre 2003.

[2] Stout, W. Bryan. - Smart Moves: Intelligent Pathfinding, October/November 1996. Disponible sur gamasutra.com à l'URL http://www.gamasutra.com/features/19970801/pathfinding.htm

[3] Dave C. Pottinger - Ensemble Studios Terrain Analysis in Realtime Strategy. Disponible à l'URL http://www.ensemblestudios.com/news/devnews/terrain1.shtml

[4] Mark Brockington - Pawn Captures Wyvern: How Computer Chess Can Improve Your Pathfinding, June 2000. Disponible sur gamasutra.com à l'URL http://www.gamasutra.com/features/20000626/brockington_01.htm

[5] Marco Pinter - Toward More Realistic Pathfinding, mars 2001. Disponible sur gamasutra.com à l'URL http://www.gamasutra.com/features/20010314/pinter_01.htm

[6] Steven Woodcock - AI Roundtable Moderator's Report 2003, Game Developer's Conference, March 6-8. Disponible à l'URL : http://www.gameai.com/cgdc03notes.htm

[7] Robert Holte, M.B. Perez, Robert Zimmer, and Alan MacDonald - Hierarchical A*: Searching Abstraction Hierarchies Efficiently, Technical report tr-95-18. Disponible à l'URL : http://www.csi.uottawa.ca/~holte/Publications/tr-95-18.ps

[8] A. Stentz - The Focussed D* Algorithm for Real-Time Replanning, , Proceedings of IJCAI-95, August 1995. Disponible à l'URL : http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.ps

[9] Pottinger, Dave C. - Coordinated Unit Movement, January 1999. Disponible sur gamasutra.com à l'URL http://www.gamasutra.com/features/19990122/movement_01.htm

[10] Pottinger, Dave C. - Implementing Coordinated Movement, February 1999. Disponible sur gamasutra.com à l'URL http://www.gamasutra.com/features/19990129/implementing_01.htm

[11] Le site de Craig Reynolds (URL : http://www.red3d.com/cwr/boids/) est une référence en matière de simulation de troupeaux.

[12] Le site de Amit Patel (URL : http://www-cs-students.stanford.edu/~amitp/gameprog.html) est une mine d'informations sur le pathfinding et les jeux vidéo (on y trouve entre autre une implémentation intéressante en C++ de A*, ainsi que des liens vers d'autres implémentations).

[13] Le site de Steven Woodcok (URL : http://www.gameai.com/) est une autre mine d'information sur l'IA dans les jeux vidéo.

Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0

Creative Commons n'est pas un cabinet d'avocats et ne fournit pas de services de conseil juridique. La distribution de la présente version de ce contrat ne crée aucune relation juridique entre les parties au contrat présenté ci-après et Creative Commons. Creative Commons fournit cette offre de contrat-type en l'état, à seule fin d'information. Creative Commons ne saurait être tenu responsable des éventuels préjudices résultant du contenu ou de l'utilisation de ce contrat.

Contrat

L'Oeuvre (telle que définie ci-dessous) est mise à disposition selon les termes du présent contrat appelé Contrat Public Creative Commons (dénommé ici « CPCC » ou « Contrat »). L'Oeuvre est protégée par le droit de la propriété littéraire et artistique (droit d'auteur, droits voisins, droits des producteurs de bases de données) ou toute autre loi applicable. Toute utilisation de l'Oeuvre autrement qu'explicitement autorisée selon ce Contrat ou le droit applicable est interdite.

L'exercice sur l'Oeuvre de tout droit proposé par le présent contrat vaut acceptation de celui-ci. Selon les termes et les obligations du présent contrat, la partie Offrante propose à la partie Acceptante l'exercice de certains droits présentés ci-après, et l'Acceptant en approuve les termes et conditions d'utilisation.

1. Définitions

a. **« Oeuvre »** : oeuvre de l'esprit protégeable par le droit de la propriété littéraire et artistique ou toute loi applicable et qui est mise à disposition selon les termes du présent Contrat.
b. **« Oeuvre dite Collective »** : une oeuvre dans laquelle l'oeuvre, dans sa forme intégrale et non modifiée, est assemblée en un ensemble collectif avec d'autres contributions qui constituent en elles-mêmes des oeuvres séparées et indépendantes. Constituent notamment des Oeuvres dites Collectives les publications périodiques, les anthologies ou les encyclopédies. Aux termes de la présente autorisation, une oeuvre qui constitue une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée (telle que définie ci-après).
c. **« Oeuvre dite Dérivée »** : une oeuvre créée soit à partir de l'Oeuvre seule, soit à partir de l'Oeuvre et d'autres oeuvres préexistantes. Constituent notamment des Oeuvres dites Dérivées les traductions, les arrangements musicaux, les adaptations théâtrales, littéraires ou cinématographiques, les enregistrements sonores, les reproductions par un art ou un procédé quelconque, les résumés, ou toute autre forme sous laquelle l'Oeuvre puisse être remaniée, modifiée, transformée ou adaptée, à l'exception d'une oeuvre qui constitue une Oeuvre dite Collective. Une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée aux termes du présent Contrat. Dans le cas où l'Oeuvre serait une composition musicale ou un enregistrement sonore, la synchronisation de l'oeuvre avec une image animée sera considérée comme une Oeuvre dite Dérivée pour les propos de ce Contrat.
d. **« Auteur original »** : la ou les personnes physiques qui ont créé l'Oeuvre.
e. **« Offrant »** : la ou les personne(s) physique(s) ou morale(s) qui proposent la mise à disposition de l'Oeuvre selon les termes du présent Contrat.
f. **« Acceptant »** : la personne physique ou morale qui accepte le présent contrat et exerce des droits sans en avoir violé les termes au préalable ou qui a reçu l'autorisation expresse de l'Offrant d'exercer des droits dans le cadre du présent contrat malgré une précédente violation de ce contrat.

2. Exceptions aux droits exclusifs. Aucune disposition de ce contrat n'a pour intention de réduire, limiter ou restreindre les prérogatives issues des exceptions aux droits, de l'épuisement des droits ou d'autres limitations aux droits exclusifs des ayants droit selon le droit de la propriété littéraire et artistique ou les autres lois applicables.

3. Autorisation. Soumis aux termes et conditions définis dans cette autorisation, et ceci pendant toute la durée de protection de l'Oeuvre par le droit de la propriété littéraire et artistique ou le droit applicable, l'Offrant accorde à l'Acceptant l'autorisation mondiale d'exercer à titre gratuit et non exclusif les droits suivants :

a. reproduire l'Oeuvre, incorporer l'Oeuvre dans une ou plusieurs Oeuvres dites Collectives et reproduire l'Oeuvre telle qu'incorporée dans lesdites Oeuvres dites Collectives;
b. distribuer des exemplaires ou enregistrements, présenter, représenter ou communiquer l'Oeuvre au public par tout procédé technique, y compris incorporée dans des Oeuvres Collectives;
c. lorsque l'Oeuvre est une base de données, extraire et réutiliser des parties substantielles de l'Oeuvre.

Les droits mentionnés ci-dessus peuvent être exercés sur tous les supports, médias, procédés techniques et formats. Les droits ci-dessus incluent le droit d'effectuer les modifications nécessaires techniquement à l'exercice des droits dans d'autres formats et procédés techniques. L'exercice de tous les droits qui ne sont pas expressément autorisés par l'Offrant ou dont il n'aurait pas la gestion demeure réservé, notamment les mécanismes de gestion collective obligatoire applicables décrits à l'article 4(d).

4. Restrictions. L'autorisation accordée par l'article 3 est expressément assujettie et limitée par le respect des restrictions suivantes :

a. L'Acceptant peut reproduire, distribuer, représenter ou communiquer au public l'Oeuvre y compris par voie numérique uniquement selon les termes de ce Contrat. L'Acceptant doit inclure une copie ou l'adresse Internet (Identifiant Uniforme de Ressource) du présent Contrat à toute reproduction ou enregistrement de l'Oeuvre que l'Acceptant distribue, représente ou communique au public y compris par voie numérique. L'Acceptant ne peut pas offrir ou imposer de conditions d'utilisation de l'Oeuvre qui altèrent ou restreignent les termes du présent Contrat ou l'exercice des droits qui y sont accordés au bénéficiaire. L'Acceptant ne peut pas céder de droits sur l'Oeuvre. L'Acceptant doit conserver intactes toutes les informations qui renvoient à ce Contrat et à l'exonération de responsabilité. L'Acceptant ne peut pas reproduire, distribuer, représenter ou communiquer au public l'Oeuvre, y compris par voie numérique, en utilisant une mesure technique de contrôle d'accès ou de contrôle d'utilisation qui serait contradictoire avec les termes de cet Accord contractuel. Les mentions ci-dessus s'appliquent à l'Oeuvre telle qu'incorporée dans une Oeuvre dite Collective, mais, en dehors de l'Oeuvre en elle-même, ne soumettent pas l'Oeuvre dite Collective, aux termes du présent Contrat. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Offrant, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Offrant, comme demandé. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Auteur, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Auteur, comme demandé.

b. L'Acceptant ne peut exercer aucun des droits conférés par l'article 3 avec l'intention ou l'objectif d'obtenir un profit commercial ou une compensation financière personnelle. L'échange de l'Oeuvre avec d'autres Oeuvres protégées par le droit de la propriété littéraire et artistique par le partage électronique de fichiers, ou par tout autre moyen, n'est pas considéré comme un échange avec l'intention ou l'objectif d'un profit commercial ou d'une compensation financière personnelle, dans la mesure où aucun paiement ou compensation financière n'intervient en relation avec l'échange d'Oeuvres protégées.
c. Si l'Acceptant reproduit, distribue, représente ou communique l'Oeuvre au public, y compris par voie numérique, il doit conserver intactes toutes les informations sur le régime des droits et en attribuer la paternité à l'Auteur Original, de manière raisonnable au regard au médium ou au moyen utilisé. Il doit communiquer le nom de l'Auteur Original ou son éventuel pseudonyme s'il est indiqué ; le titre de l'Oeuvre Originale s'il est indiqué ; dans la mesure du possible, l'adresse Internet ou Identifiant Uniforme de Ressource (URI), s'il existe, spécifié par l'Offrant comme associé à l'Oeuvre, à moins que cette adresse ne renvoie pas aux informations légales (paternité et conditions d'utilisation de l'Oeuvre). Ces obligations d'attribution de paternité doivent être exécutées de manière raisonnable. Cependant, dans le cas d'une Oeuvre dite Collective, ces informations doivent, au minimum, apparaître à la place et de manière aussi visible que celles à laquelle apparaissent les informations de même nature.
d. Dans le cas où une utilisation de l'Oeuvre serait soumise à un régime légal de gestion collective obligatoire, l'Offrant se réserve le droit exclusif de collecter ces redevances par l'intermédiaire de la société de perception et de répartition des droits compétente. Sont notamment concernés la radiodiffusion et la communication dans un lieu public de phonogrammes publiés à des fins de commerce, certains cas de retransmission par câble et satellite, la copie privée d'Oeuvres fixées sur phonogrammes ou vidéogrammes, la reproduction par reprographie.

5. Garantie et exonération de responsabilité

a. En mettant l'Oeuvre à la disposition du public selon les termes de ce Contrat, l'Offrant déclare de bonne foi qu'à sa connaissance et dans les limites d'une enquête raisonnable :
    i. L'Offrant a obtenu tous les droits sur l'Oeuvre nécessaires pour pouvoir autoriser l'exercice des droits accordés par le présent Contrat, et permettre la jouissance paisible et l'exercice licite de ces droits, ceci sans que l'Acceptant n'ait aucune obligation de verser de rémunération ou tout autre paiement ou droits, dans la limite des mécanismes de gestion collective obligatoire applicables décrits à l'article 4(e);
b. L'Oeuvre n'est constitutive ni d'une violation des droits de tiers, notamment du droit de la propriété littéraire et artistique, du droit des marques, du droit de l'information, du droit civil ou de tout autre droit, ni de diffamation, de violation de la vie privée ou de tout autre préjudice délictuel à l'égard de toute tierce partie.
c. A l'exception des situations expressément mentionnées dans le présent Contrat ou dans un autre accord écrit, ou exigées par la loi applicable, l'Oeuvre est mise à disposition en l'état sans garantie d'aucune sorte, qu'elle soit expresse ou tacite, y compris à l'égard du contenu ou de l'exactitude de l'Oeuvre.

6. Limitation de responsabilité. A l'exception des garanties d'ordre public imposées par la loi applicable et des réparations imposées par le régime de la responsabilité vis-à-vis d'un tiers en raison de la violation des garanties prévues par l'article 5 du présent contrat, l'Offrant ne sera en aucun cas tenu responsable vis-à-vis de l'Acceptant, sur la base d'aucune théorie légale ni en raison d'aucun préjudice direct, indirect, matériel ou moral, résultant de l'exécution du présent Contrat ou de l'utilisation de l'Oeuvre, y compris dans l'hypothèse où l'Offrant avait connaissance de la possible existence d'un tel préjudice.

7. Résiliation

a. Tout manquement aux termes du contrat par l'Acceptant entraîne la résiliation automatique du Contrat et la fin des droits qui en découlent. Cependant, le contrat conserve ses effets envers les personnes physiques ou morales qui ont reçu de la part de l'Acceptant, en exécution du présent contrat, la mise à disposition d'Oeuvres dites Dérivées, ou d'Oeuvres dites Collectives, ceci tant qu'elles respectent pleinement leurs obligations. Les sections 1, 2, 5, 6 et 7 du contrat continuent à s'appliquer après la résiliation de celui-ci.
b. Dans les limites indiquées ci-dessus, le présent Contrat s'applique pendant toute la durée de protection de l'Oeuvre selon le droit applicable. Néanmoins, l'Offrant se réserve à tout moment le droit d'exploiter l'Oeuvre sous des conditions contractuelles différentes, ou d'en cesser la diffusion; cependant, le recours à cette option ne doit pas conduire à retirer les effets du présent Contrat (ou de tout contrat qui a été ou doit être accordé selon les termes de ce Contrat), et ce Contrat continuera à s'appliquer dans tous ses effets jusqu'à ce que sa résiliation intervienne dans les conditions décrites ci-dessus.

8. Divers

a. A chaque reproduction ou communication au public par voie numérique de l'Oeuvre ou d'une Oeuvre dite Collective par l'Acceptant, l'Offrant propose au bénéficiaire une offre de mise à disposition de l'Oeuvre dans des termes et conditions identiques à ceux accordés à la partie Acceptante dans le présent Contrat.
b. La nullité ou l'inapplicabilité d'une quelconque disposition de ce Contrat au regard de la loi applicable n'affecte pas celle des autres dispositions qui resteront pleinement valides et applicables. Sans action additionnelle par les parties à cet accord, lesdites dispositions devront être interprétées dans la mesure minimum nécessaire à leur validité et leur applicabilité.
c. Aucune limite, renonciation ou modification des termes ou dispositions du présent Contrat ne pourra être acceptée sans le consentement écrit et signé de la partie compétente.
d. Ce Contrat constitue le seul accord entre les parties à propos de l'Oeuvre mise ici à disposition. Il n'existe aucun élément annexe, accord supplémentaire ou mandat portant sur cette Oeuvre en dehors des éléments mentionnés ici. L'Offrant ne sera tenu par aucune disposition supplémentaire qui pourrait apparaître dans une quelconque communication en provenance de l'Acceptant. Ce Contrat ne peut être modifié sans l'accord mutuel écrit de l'Offrant et de l'Acceptant.
e. Le droit applicable est le droit français.

Creative Commons n'est pas partie à ce Contrat et n'offre aucune forme de garantie relative à l'Oeuvre. Creative Commons décline toute responsabilité à l'égard de l'Acceptant ou de toute autre partie, quel que soit le fondement légal de cette responsabilité et quel que soit le préjudice subi, direct, indirect, matériel ou moral, qui surviendrait en rapport avec le présent Contrat. Cependant, si Creative Commons s'est expressément identifié comme Offrant pour mettre une Oeuvre à disposition selon les termes de ce Contrat, Creative Commons jouira de tous les droits et obligations d'un Offrant.

A l'exception des fins limitées à informer le public que l'Oeuvre est mise à disposition sous CPCC, aucune des parties n'utilisera la marque « Creative Commons » ou toute autre indication ou logo afférent sans le consentement préalable écrit de Creative Commons. Toute utilisation autorisée devra être effectuée en conformité avec les lignes directrices de Creative Commons à jour au moment de l'utilisation, telles qu'elles sont disponibles sur son site Internet ou sur simple demande.

Creative Commons peut être contacté à http://creativecommons.org/.