Notions de pathfinding

par c0unt0

4 juillet 2003

1 Le pathfinding c'est quoi t'est-ce donc t'il ?

Le "pathfinding", ou "recherche de chemin", en bon françoy, décrit une série d'algorithmes permettant à un objet O de se déplacer d'un point P à un point P' dans un environnement E, c'est, en gros, l'algorithme A qui permet à un tank mammouth M de se déplacer de sa position courante PC vers les petits machins verts qui courent en rigolant PMVQCER dans la direction D de votre base B.*(ndc1 : ouf)*

Mais attention, il ne faut pas le confondre avec l'algo qui se charge de la prise de décision, ce qui est un tout autre problème ! Le pathfinding n'est utilisé qu'à partir du moment où *(ndc2 : DTC proof :P)* l'on sait où l'on est et où l'on va, il se contrebat du pourquoi, du comment et de la réponse ultime (42, soit dit en passant) *(ndc1 : mon dieu, ils sont partout)(nda : oui).*

Le nombre d'algo potentiels pour résoudre ce genre de problèmes est énorme, de plus la complexité du problème lui même en fait un sujet d'études très couru des chercheurs en blouse blanche et en IA, (pour les forts en math : c'est de la théorie des graphs bourrue, et c'est un problème n(p) complexe...super...).

Histoire de ne pas partir dans le grand vide, je vais tout baser sur l'idée d'un jeu se passant sur un landscape, comme ça, ça sera plus simple, j'expliquerai aussi plus tard comment on pourrait adapter la chose sur un fps, ou même sur "Barbie Chaval".

Comme base on utilisera la map Fig. 1. Le vert c'est la position de départ, en bleu la destination, et en noir les obstacles. On va jeter un bref coup d'oeil à 3 solutions : la 1, la 2 et la 3. La 1 étant à la limite de l'idiotie, et la 3 à la limite du génie...il est bien évident qu'il existe bien d'autre solutions et variations sur le thème.

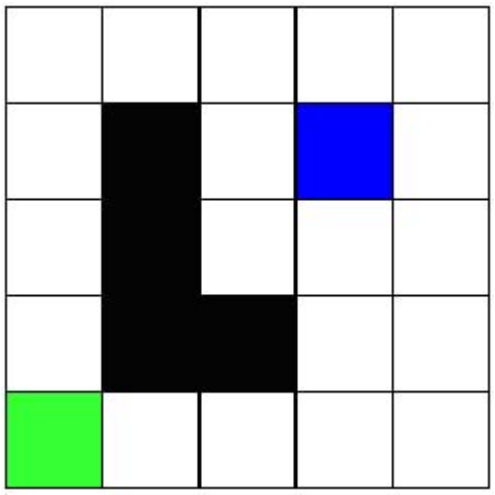


Fig. 1 – Map d'étude

2 Solution 1 : Le "bourrin"

Solution réellement appliquée dans le mémorable "Trucks" édité par Microfolies en 199quelque chose. (enfin en tout cas moi je me le rappelle, et je pense que Tbf doit s'en souvenir (parce que Tbf se souvient de tout)).

Le principe est simple, voir simpliste, il s'agit de se dire que si le joueur ne regarde pas dans sa direction, tout ce qu'une unité a à faire, c'est se déplacer directement vers la destination, en prenant bien soin d'ignorer les obstacles, et si jamais le joueur la surprend, elle prend un air dégagé et attaque le joueur, celui-ci n'ayant que le temps de se dire "mais comment il e...". *(ndc2 : Le fameux "mais qu'est-ce que..." dans Splinter Cell :P)* C'est tricher, mais ça marche plutôt pas mal :

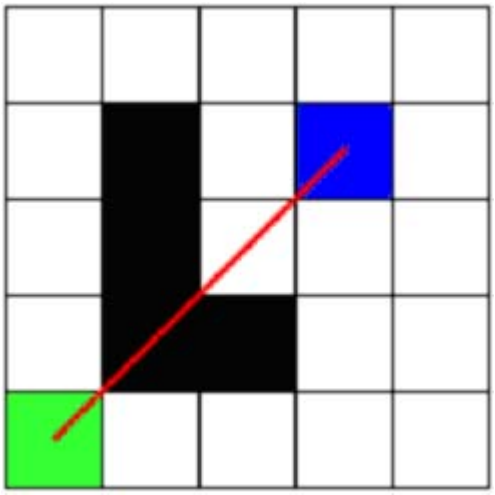


FIG. 2 – Méthode bourrine

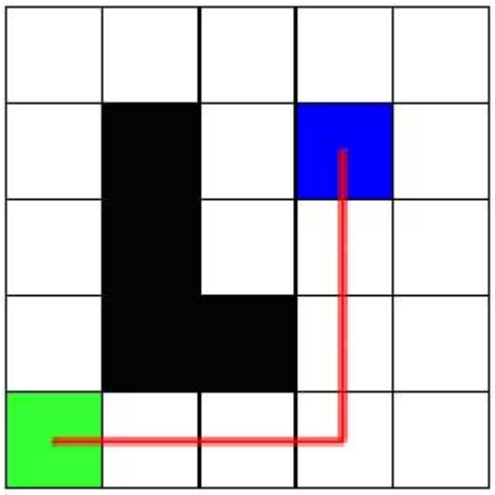


FIG. 3 – Mmmmhh, pas mal !

3 Solution 2 : "Je suis plus con que j'en ai l'air"

La solution 2 repose sur du hasard, l'unité va toujours tenter de se déplacer vers sa destination, si elle ne peut pas, elle va alors choisir une autre direction possible, mais au hasard. Des fois ça marche, des fois ça marche pas, c'est le problème avec le hasard...Il peut (souvent) arriver que la direction tirée au hasard éloigne l'unité, ou lui fait prendre un détour. Quand ça marche bien, ça peut donner la Fig. 3. Sinon, ça donne plus souvent la Fig. 4.

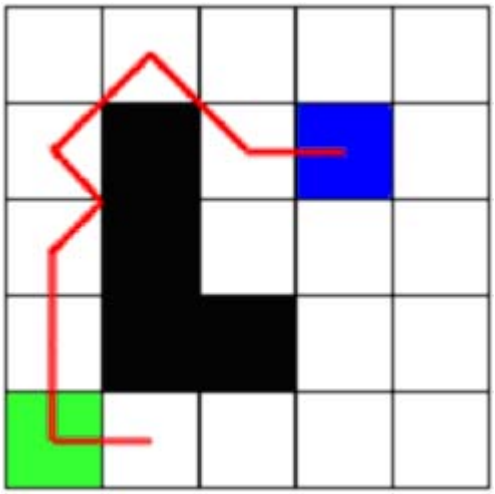


FIG. 4 – Tu t'es vu quand t'as bu !

4 Solution 3, dite "À la cosaque"

(aussi appelée, chez les chercheurs en IA et en blouses blanches "$A*$" ou "$AStar$" (comme dans $A \Rightarrow A$, $* \Rightarrow Star$))

Là ça devient un poil plus musclé...

Le principe de base est de se dire que pour chaque case, on peut calculer un score, et que plus ce score est bas, plus la case a la de chance d'être sur le meilleur chemin. Toute la difficulté vient de la façon de calculer ce fameux score (par ex., un bien fameux score 6-3 3-6 6-0 6-0).

Pour calculer ce score, on va utiliser deux choses : un coût et une heuristique. Dans des versions simples, le coût n'est que le nombre de cases parcourues depuis la case de départ. Pour corser les choses, on peut ajouter d'autres trucs dans ce coût, comme le type de terrain

(genre béton = +0, herbe = +1, boue = +10, v'voyez l'esprit quoi).

L'heuristique, c'est une règle bidon (c'est-à-dire qui marche mais sans véritable raison) qu'on fixe et qui permet de juger de l'importance d'une position par rapport au chemin à rechercher. La plupart des algos d'$A*$ utilisent soit la distance entre la case courante et la destination : $\sqrt{(PC.x - PO.x)^2 + (PC.y - PO.y)^2)}$, soit, plus rapide à calculer, le *Manhattan* [1], qui est une approximation grosso modo valable (bidon je vous avais dit...) dont la formule est $(PC.x - PO.x) + (PC.y - PO.y)$.

Une fois qu'on a défini ces deux valeurs, on parcourt la carte en partant du point de départ, on calcule le score de toutes les destinations possibles, puis on recommence sur toutes les destinations, en commençant par celles qui ont le plus petit score...

Vous avez compris ?

Non ?

Moi non plus...

Donc j'ai fait des petits dessins.

[1] Le *Manhattan* s'appelle comme cela car on compte en fait les distances par blocs (d'immeubles donc, comme à N-Y ;)). Ainsi, le *Manhattan* est simplement le nombre de lignes et de colonnes entre 2 cases.

4.1 Étape 1

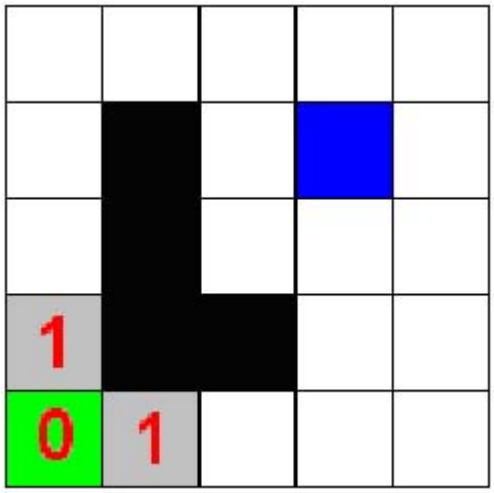


Fig. 5 – Coûts

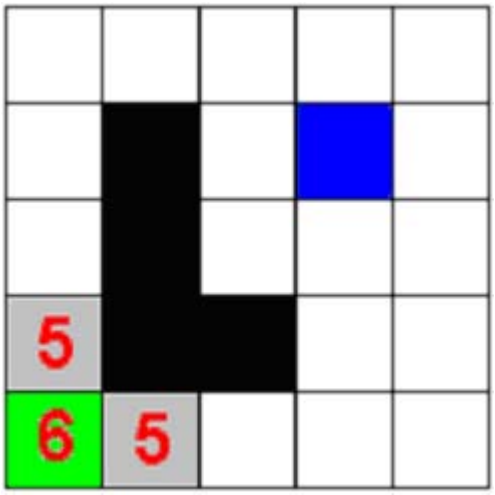


Fig. 6 – Heuristique

Conclusion de l'opération : Nos deux nouvelles cases potentielles ont toutes les deux un score de 6, donc on va tester ces deux-là...
En route pour l'étape 2.

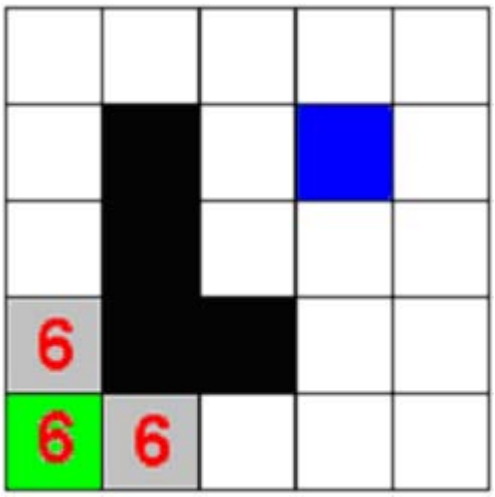


Fig. 7 – Score total

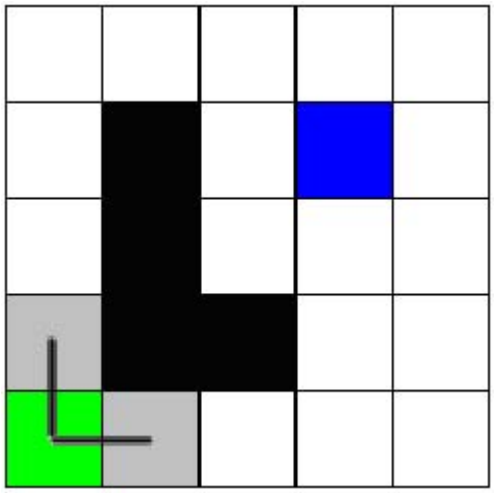


Fig. 8 – Déplacements possibles à partir de la position de départ

4.2 Étape 2

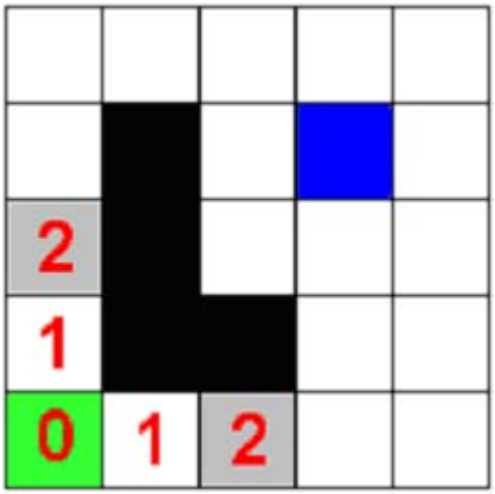


Fig. 9 – Coûts

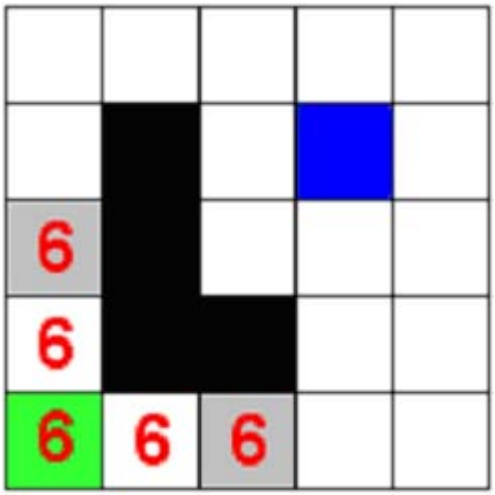


Fig. 11 – Score total

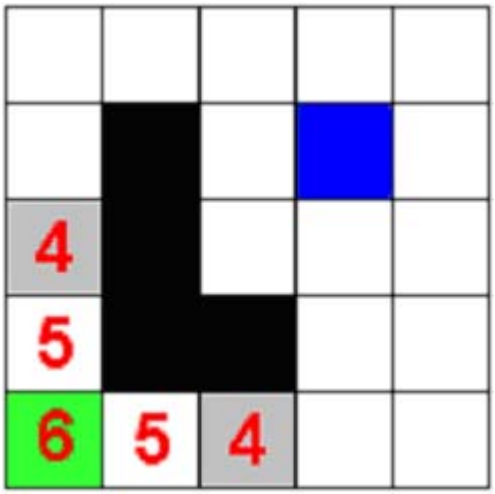


Fig. 10 – Heuristique

Là encore, toutes les nouvelles cases ont le même score 6 toujours, donc on persiste.
Et zou : étape 3.

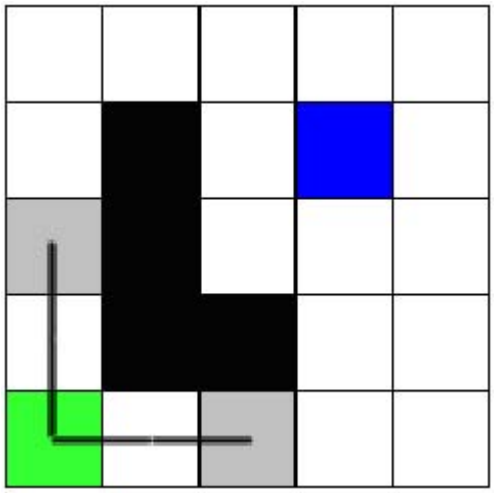


Fig. 12 – Déplacements possibles à partir de la position de départ

4.3 Étape 3

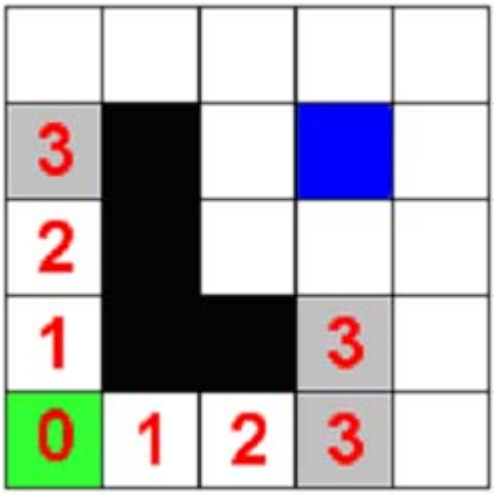


FIG. 13 – Coûts

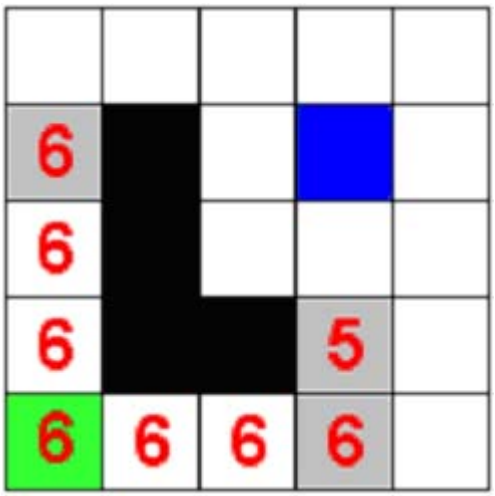


FIG. 15 – Score total

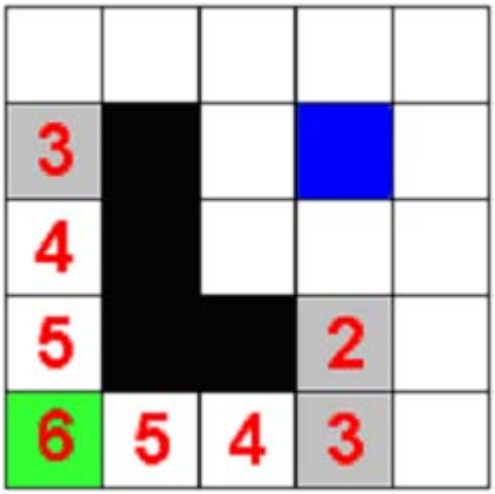


FIG. 14 – Heuristique

Et là, affolant, un truc se passe : une case a un score de 5 alors que les deux autres plafonnent à 6, c'est sans doute dans cette direction qu'il faut aller...
Let's rock'n step 4.

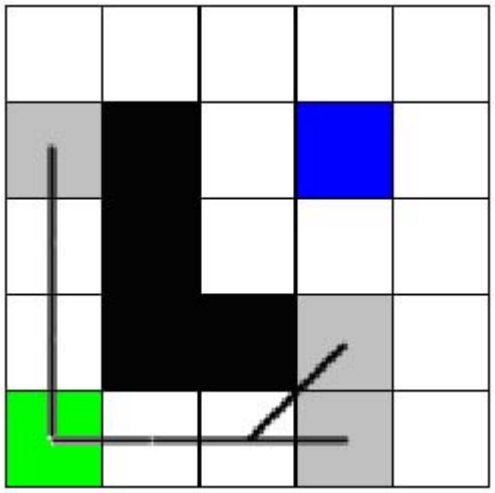


FIG. 16 – Déplacements possibles à partir de la position de départ

4.4 Étape 4

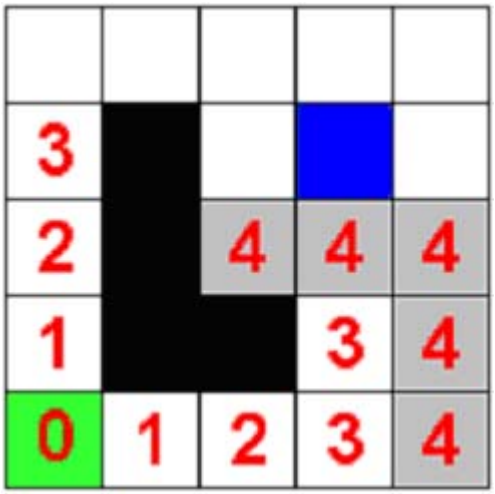


Fig. 17 – Coûts

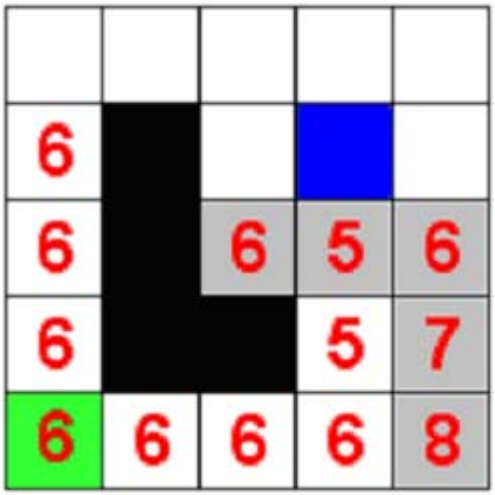


Fig. 19 – Score total

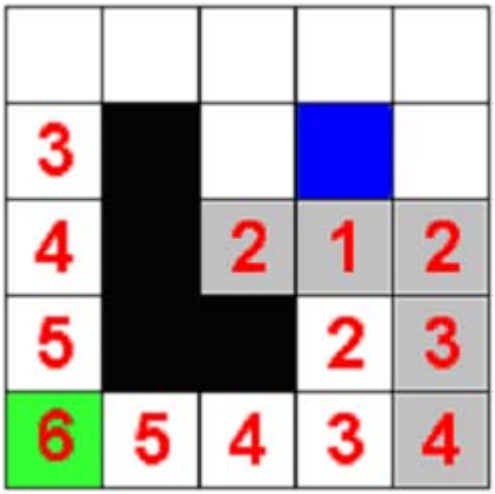


Fig. 18 – Heuristique

Et là encore le miracle se produit ! Une case ressort de la masse avec un score inférieur aux autres !
Allons voir à l'étapes 5, si, enfin cette recherche pourrait arriver à quelque chose (parce que là les chtits dessins : J'EN PEUX PLUS).

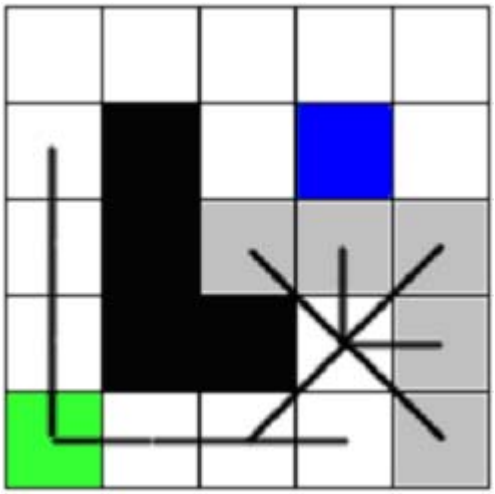


Fig. 20 – Déplacements possibles à partir de la position de départ

4.5 Étape 5

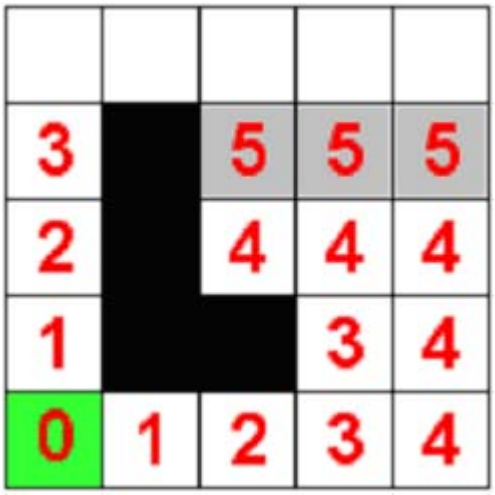


FIG. 21 – Coûts

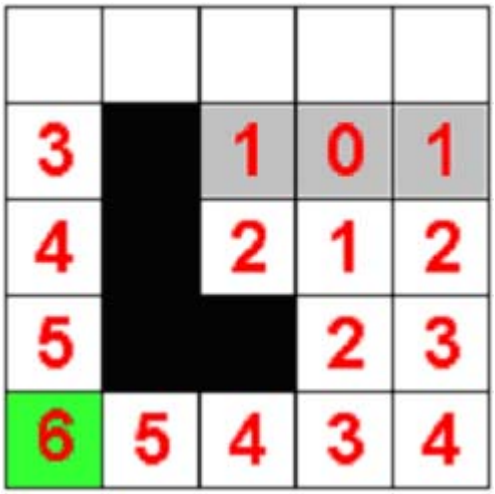


FIG. 22 – Heuristique

Glory glory allelyuah ! On a trouvé la destination ! Et ça c'est bien.

Maintenant tout ce qu'il reste à faire : c'est de remonter le chemin depuis la case de destination, jusqu'à la case de départ, chose facilité par le fait qu'on a toujours fait super gaffe à

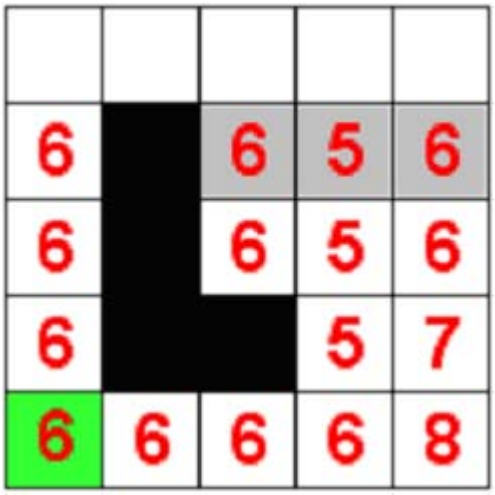


FIG. 23 – Score total

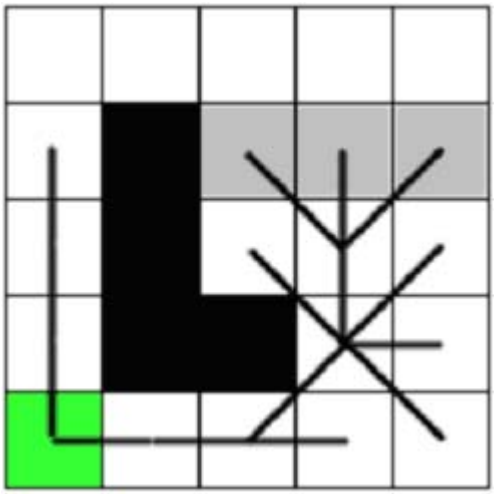


FIG. 24 – Déplacements possibles à partir de la position de départ

partir d'une position vers plusieurs, mais jamais le contraire (super clair, hein ?), en bref : quand on est sur une case on ne peut revenir en arrière que vers une et une seule case.

4.6 Solution

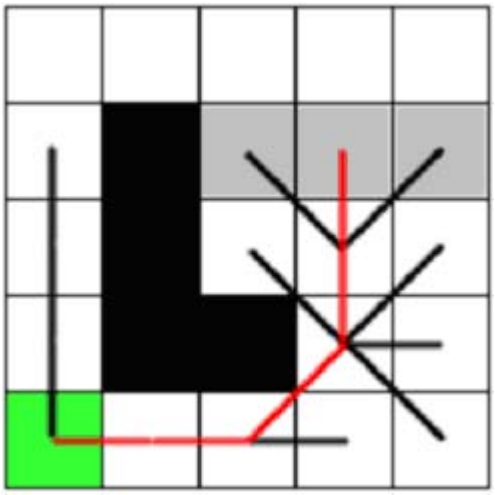
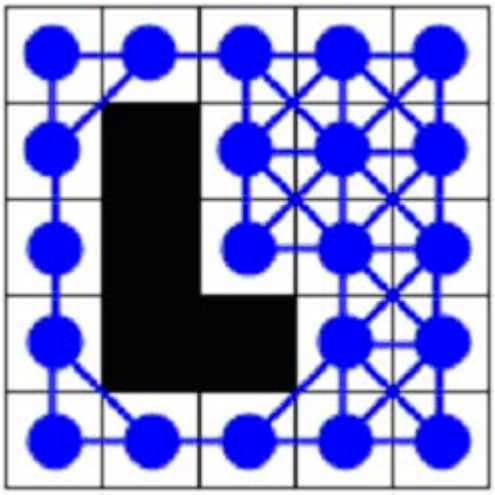


FIG. 25 – Et hop, la solution !

Voilà, l'$A*$ a une fois de plus prouvé sa force !

Enfin presque, le problème principal de l'$A*$ (et de la plupart des algos de pathfinding, d'ailleurs) c'est que le coût CPU n'est pas constant, certains chemins vont être super rapides à calculer, d'autres plus lents, et il faut aussi prendre en compte que les obstacles bougent dans la Vraie Vie (vous avez déjà vu un tank attendre au feu rouge, vous ?) *(ndc1 : oui, à GTA VC)(nda : genre je l'attendais pas, celle-là)* obligeant parfois à recalculer le chemin en cours depuis le début !
Et après, il y a plein de détails d'implémentation et d'optimisation, que vous pouvez faire varier pour adapter l'algo à votre jeu.

Super, et comment j'adapte tes putains de cases à un beau FPS (ou à Barbie Chaval *(ndc2 : QUOI ? ? ?)(nda : Pffff...)*) ?
Bah c'est très bête : finalement, quand on la regarde de près, la map de case, on pourrait dire que c'est un réseau de cellules interconnectées.



Et qu'est-ce qui nous oblige à les organiser comme ça, les cellules du réseau ?

Bah rien...

Et c'est obligé d'être que des angles fixes comme ça ?

Bah non plus...

Donc on pourrait déformer le graphe, pour suivre les contours de Q3-DM3 ?

Bah ouais, et le plus grave dans tout ça ? Hey bah c'est que ça marche ! *(ndc1 : encore heureux putain)(nda : ça fait plaisir, y en au moins 1 que ça intéresse...)*

Quand un Level-Designer prépare un niveau, en plus des portes, armes, pickup et autres boutons divers et variés, il va aussi (souvent aidé par des outils qui vont "découvrir" le décors automatiquement) poser son réseau de pathfinding, comme ça les bots auront l'air moins cons !

5 Conclusion

Et voilà!!!

(Bon j'avoue, j'ai pas fait l'éclairage, mais ça m'a pris plus longtemps que prévu, cette petite chose...)

Le bon lien qui va bien pour l'IA : http://www.gameia.com
Le bon lien qui va bien pour le développement de jeux en général : http://www.gamasutra.com
Une autre platée de liens sur l'IA en général : http://www.vieartificielle.com/index.php?action=annuaire&op=viewlink&cid=12
Et merci à tous les gens qui ont écrit des articles sur le pathfinding que j'ai pu piller comme un sale!
Et merci à tous les 4 gens qui ont corrigé : messieurs lije, End[ER], Moktar *(nda : on peut dire "contrebats" comme dans "je m'en bats...")* et xentyr.
Et merci à xentyr pour l'hébergement.
Et le lien de Barbie Chaval, car, apparemment, les gens ne connaissent pas : http://www.gamekult.com/tout/jeux/fiches/J000051864.html
Et enfin, tout de même, merci à moi-même, parce que bon là, quand même, j'assure...