[Note du traducteur : ceci est une traduction en français libre et non officielle du FFF paru sur le forum]

Friday Facts N°317
Nouvel algorithme de recherche de trajectoires

Posté parOxyd, TOGoS et Klonan

le 18/10/2019

Nouvel algorithme de recherche de trajectoires (par Oxyd)

La semaine dernière, nous avons mentionné le changement pour que les biters n'entrent pas en collision les uns avec les autres, mais ce n'était pas la seule mise à jour sur les biters publiée la semaine dernière. Par une certaine coïncidence, les mises à jour de cette semaine comprenaient quelque chose sur lequel je travaillais depuis quelques semaines : une mise à niveau du système d'orientation de l'ennemi.

Recherche de trajectoires

Lorsqu'une unité veut aller quelque part, elle doit d'abord trouver comment s'y rendre. Cela pourrait être aussi simple que d'aller droit au but, mais il peut y avoir des obstacles - comme des falaises, des arbres, des nids, des entités de joueurs - dans le chemin. Pour ce faire, il indiquera à la **recherche de trajectoires** sa position actuelle et sa position de destination, et la recherche répondra - éventuellement après plusieurs ticks - par une **trajectoire**, qui est simplement une série de repères pour que cette unité arrive jusqu'à sa destination.

Pour faire son travail, le système utilise un algorithme appelé A* (prononcez "A étoile"). Un exemple simple de recherche de trajectoires A* est montré dans la vidéo ci-dessous : un biter veut contourner des falaises. Le système commence une exploration de la carte autour du biter (représentée par des points blancs). Tout d'abord, il essaie d'aller droit au but, mais dès qu'il atteint les falaises, il "se répand" dans les deux sens, cherchant une position qui lui permettra de se diriger à nouveau vers cette destination.

[NdT : Cliquez pour voir la vidéo]

Dans cette vidéo, l'algorithme est ralenti pour mieux montrer comment il fonctionne.

Chaque point de l'animation représente un nœud. Chaque nœud se souvient de sa distance depuis le début de la recherche et d'une estimation de la distance entre ce nœud et la destination - cette estimation est fournie par ce qu'on appelle une fonction heuristique. Les fonctions heuristiques sont ce qui fait fonctionner A* - c'est ce qui dirige l'algorithme dans la bonne direction.

Un choix simple pour cette fonction est simplement la distance en ligne droite entre le nœud et la position de destination - c'est ce que nous utilisons dans Factorio depuis toujours, et c'est ce qui rend l'algorithme initialement direct. Ce n'est pas le seul choix, cependant - si la fonction heuristique connaissait certains des obstacles, elle pourrait piloter l'algorithme autour d'eux, ce qui permettrait une recherche plus rapide, puisqu'elle n'aurait pas à explorer des nœuds supplémentaires. Évidemment, plus l'heuristique est intelligente, plus elle est difficile à mettre en œuvre.

La fonction heuristique simple en ligne droite est idéale pour la recherche de trajectoires sur des distances relativement courtes. C'était correct dans les versions précédentes de Factorio - à peu près la seule recherche de trajectoires à longue distance était faite par des biters mis en colère par la pollution, et cela n'arrive pas très souvent, relativement parlant. De nos jours, cependant, nous avons de l'artillerie. L'artillerie peut facilement tirer - et provoquer - un grand nombre de biters à l'extrémité d'un grand lac, qui vont alors tous essayer de trouver la trajectoire autour du lac. La vidéo ci-dessous montre à quoi cela ressemble quand l'algorithme A* simple que nous avons utilisé jusqu'à présent essaie de contourner un lac.

[NdT : Cliquez pour voir la vidéo]

Cette vidéo montre à quelle vitesse l'algorithme fonctionne en réalité ; il n'a pas été ralenti.

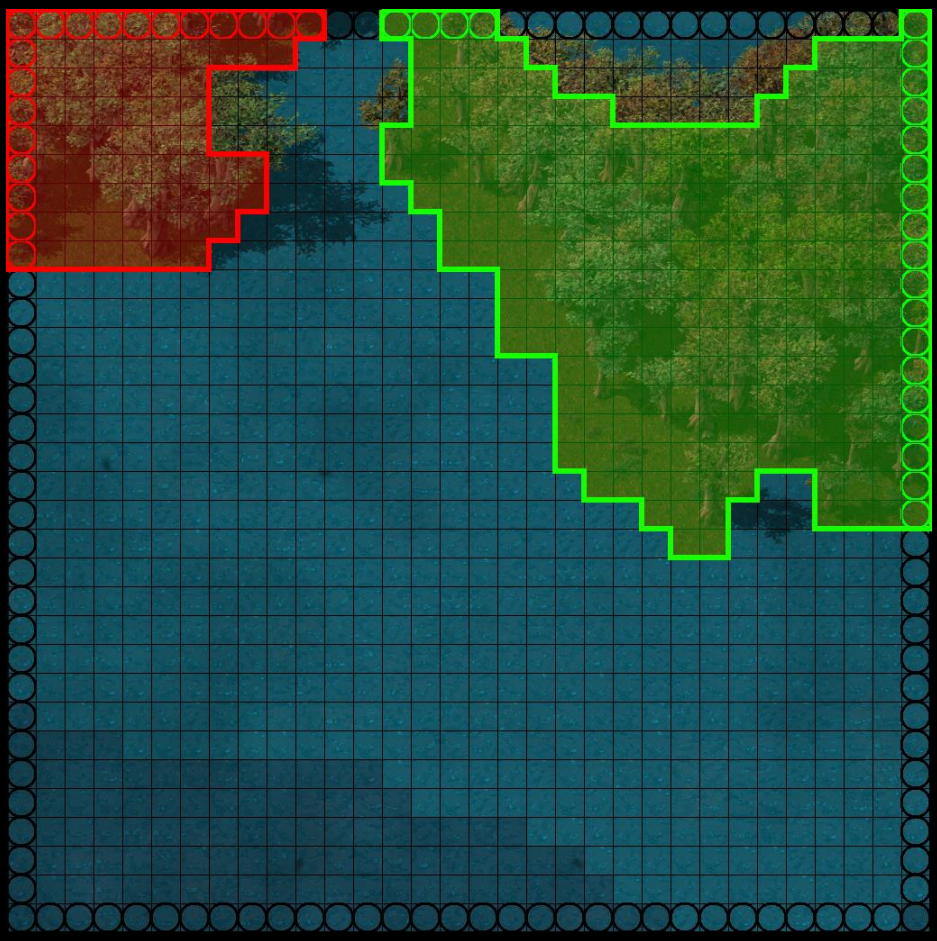
Contraction des blocs

La recherche de trajectoires est un vieux problème, et il existe donc de nombreuses techniques pour l'améliorer. Certaines de ces techniques entrent dans la catégorie de la **recherche hiérarchique de trajectoires** - là où la carte est d'abord simplifiée, une trajectoire est trouvée dans la carte simplifiée, et celle-ci est ensuite utilisée pour aider à trouver la vraie trajectoire. Encore une fois, il existe plusieurs techniques pour le faire exactement, mais toutes nécessitent une simplification de l'espace de recherche.

Comment simplifier un monde Factorio ? Nos cartes sont générées au hasard et changent constamment - la mise en place et la suppression d'entités (telles que les assembleurs, les murs ou les tourelles) sont probablement les mécaniques les plus importantes du jeu en soi. Nous ne voulons pas recalculer cette simplification à chaque fois qu'une entité est ajoutée ou supprimée. En même temps, le fait de réorganiser la carte à partir de zéro à chaque fois que nous voulons trouver une trajectoire pourrait très facilement annuler tous les gains de performance réalisés.

C'est l'un de nos spécialistes de l'accès aux sources, Allaizn, qui a eu l'idée que j'ai fini par mettre en œuvre. Rétrospectivement, l'idée est évidente.

Le jeu est basé sur des blocs de tuiles de 32x32. Le processus de simplification remplacera chaque bloc par un ou plusieurs nœuds abstraits. Puisque notre but est d'améliorer la recherche de trajectoires autour des lacs, nous pouvons ignorer toutes les entités et considérer uniquement les tuiles - l'eau est infranchissable, la terre est praticable. Nous divisons chaque bloc en composants séparés - un composant est une zone de tuiles où une unité peut passer de n'importe quelle tuile dans le composant à n'importe quelle autre dans le même composant. L'image ci-dessous montre un bloc divisé en deux composantes distinctes, rouge et vert. Chacun de ces composants deviendra un seul nœud abstrait - fondamentalement, le bloc entier est réduit en deux "points".



L'idée clé d'Allaizn était que nous n'avons pas besoin de stocker le composant pour chaque tuile sur la carte - il suffit de se rappeler les composants pour les tuiles sur le périmètre de chaque bloc. C'est parce que ce qui nous importe vraiment, c'est de savoir à quels autres composants (dans les blocs voisins) chaque composant est connecté – ce qui ne peut dépendre que des tuiles qui sont sur le bord même du bloc.

Recherche hiérarchique de trajectoires

Nous avons trouvé comment simplifier la carte, alors comment l'utiliser pour trouver des trajectoires ? Comme je l'ai dit plus tôt, il existe de multiples techniques de recherches hiérarchiques de trajectoires. L'idée la plus simple serait de simplement trouver une trajectoire en utilisant des nœuds abstraits du début à la fin - c'est-à-dire, la trajectoire serait une liste de composants de bloc que nous devons visiter - et ensuite utiliser une série de vieilles recherches A* simples pour trouver comment passer exactement du composant d'un bloc à un autre.

Le problème ici est que nous avons un peu trop simplifié la carte : et s'il n'est pas possible de passer d'un bloc à l'autre à cause de certaines entités (comme les falaises) qui bloquent le passage ? Lorsque nous contractons des blocs, nous ignorons toutes les entités, donc nous savons simplement que les tuiles entre les blocs sont reliées d'une manière ou d'une autre, mais nous ne savons pas s'il est réellement possible de passer de l'une à l'autre ou pas.

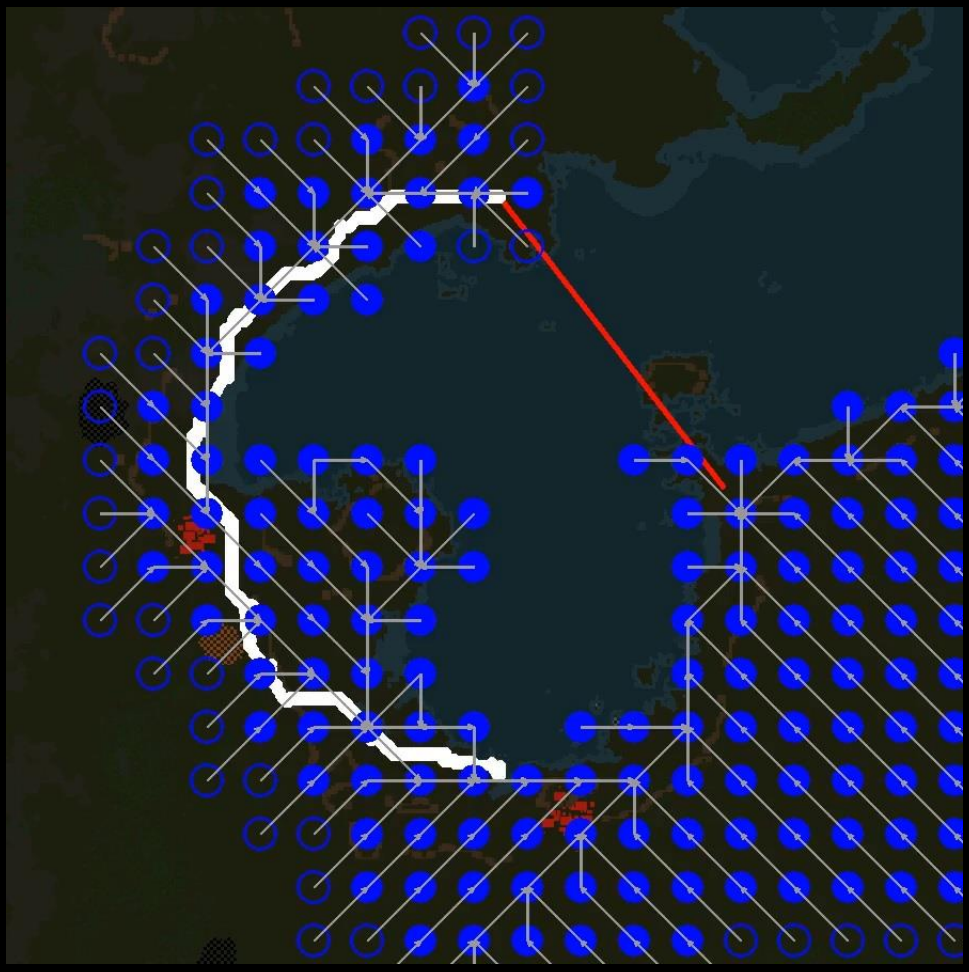
La solution est d'utiliser la simplification simplement comme une "suggestion" pour la recherche réelle. Plus précisément, nous l'utiliserons pour fournir une version intelligente de la fonction heuristique pour la recherche.

Donc ce qu'on a en fin de compte, c'est ceci : nous avons deux recherches, appelées la recherche de base, qui trouve la trajectoire réelle, et la recherche abstraite, qui fournit la fonction heuristique pour la recherche de base. Chaque fois que la recherche de base crée un nouveau nœud de base, il appelle la recherche abstraite pour obtenir l'estimation de la distance à la destination. La recherche abstraite travaille à l'envers - elle commence par la destination de la recherche, et se dirige vers son origine, sautant d'un composant à l'autre du bloc. Une fois que la recherche abstraite trouve le bloc et le composant dans lequel le nouveau nœud de base est créé, elle utilise la distance entre l'origine de la recherche abstraite (qui, là encore, est la position de destination de la recherche finale) pour calculer la distance estimée entre le nouveau nœud de base et la destination finale.

Exécuter une recherche de trajectoires complète pour chaque nœud de base serait cependant tout sauf rapide, même si la recherche abstraite saute d'un bloc à l'autre. Au lieu de cela, nous utilisons ce qu'on appelle la Reprise inverse A*. **Inverse** signifie simplement qu'il va de la destination au départ, comme je l'ai déjà dit. **Reprise** signifie qu'une fois qu'il a trouvé le bloc qui l'intéresse, nous gardons tous ses nœuds en mémoire. La prochaine fois que la recherche de base crée un nouveau nœud et a besoin de connaître son estimation de distance, nous regardons simplement les nœuds abstraits que nous avons conservés de la recherche précédente, avec une bonne chance que le nœud abstrait requis soit toujours là (après tout, un nœud abstrait couvre une grande partie d'un bloc et, le plus souvent, tout le bloc).

Même si le recherche de base crée un nœud qui est dans un bloc non couvert par un nœud abstrait, nous n'avons pas besoin de refaire une recherche abstraite complète. Une propriété intéressante de l'algorithme A* est que même après qu'il ait 'fini' et trouvé une trajectoire, il peut continuer à explorer les nœuds autour de ceux qu'il a déjà explorés. Et c'est exactement ce que nous faisons si nous avons besoin d'une estimation de distance pour un nœud de base situé dans un bloc non encore couvert par la recherche abstraite : nous reprenons la recherche abstraite à partir des nœuds que nous avons gardés en mémoire, jusqu'à ce qu'elle s'étende au nœud dont nous avons besoin.

La vidéo ci-dessous montre le nouveau système pour la recherche de trajectoires en action. Les cercles bleus sont les nœuds abstraits ; les points blancs sont la recherche de base. La recherche a été considérablement ralentie pour réaliser cette vidéo, pour montrer comment elle fonctionne. A vitesse normale, l'ensemble de la recherche ne prend que quelques ticks. Remarquez comment la recherche de base, qui utilise toujours le même vieil algorithme que nous avons toujours utilisé, "sait" simplement où faire le tour du lac, comme par magie.



[NdT : Cliquez pour voir la vidéo]

Puisque la recherche abstraite n'est utilisée que pour fournir les estimations heuristiques des distances, la recherche de base peut assez facilement s'écarter de la trajectoire suggérée par la recherche abstraite. Cela signifie que même si notre schéma de contraction des blocs ignore toutes les entités, la recherche de base peut encore les contourner avec peu de difficulté. Ignorer les entités dans le processus de simplification de la carte signifie que nous n'avons pas à le refaire à chaque fois qu'une entité est placée ou supprimée, et que nous n'avons qu'à vérifier les tuiles modifiées (comme dans le cas des remblais) – ce qui arrivent beaucoup moins souvent que dans le cas de placements d'entités.

Cela signifie aussi que nous utilisons toujours essentiellement la même vieille recherche de trajectoires que nous utilisons depuis des années, mais avec une fonction heuristique améliorée. Cela signifie que ce changement ne devrait pas affecter trop de choses, à part la vitesse de la recherche.

Au revoir TOGoS (par TOGoS)

Salut tout le monde !
Je vais démissionner du personnel officiel de Factorio après aujourd'hui pour commencer un nouvel emploi plus près de chez moi.

La raison principale de ce changement est que le travail à distance s'est avéré être quelque chose que je n'ai pas été capable de gérer aussi bien. Mais aussi, ma principale contribution au projet, le générateur de cartes programmable, est terminée, stable et assez bien comprise par au moins une autre personne, ce qui me semblait le moment propice pour passer à autre chose. Travailler dans une ferme cubique à écrire des applications Android pour tapis de course, apparemment.
On verra comment ça se passe !

Ce fut un honneur de faire partie de ce projet génial et d'y laisser mon empreinte. Et travailler sur une grande base de code C++ assez bien gérée, pleine de choses un peu différentes de ce à quoi je m'attendais, a été une ouverture d'esprit.

Je tiens également à remercier la communauté pour votre patience constante, vos commentaires honnêtes et la pression occasionnelle que vous exercez sur nous pour que le jeu soit aussi bon qu'il le devrait.

Je continuerai à lire les Friday Facts chaque semaine, à rôder sur les forums et probablement à mettre à jour la documentation ici et là. Avec un peu de chance, je trouverai le temps de mettre au point mes propres mods pour modifier le terrain. Et j'ai hâte de voir ce que vous en ferez d'autre. (Ce qui inclut le traitement d'un arriéré de mods -- je n'ai pas encore atteint la partie exploration spatiale de Space Exploration !)

La paix pour l'instant.

Projecteur sur la communauté - Micro Usine 13x9 (par Klonan)

Il y a plus de 100 Friday Facts, nous avons présenté la Micro Usine 9x14 de DaveMcW (FFF-197). Bien que cela ait pris plus de 2 ans, il a encore amélioré son design, et le résultat est tout aussi impressionnant qu'avant.

[NdT : Cliquez pour voir la vidéo]

Il donne plus d'explications sur sa Micro Usine dans ce message sur le forum.

Comme toujours, faites-nous savoir ce que vous en pensez sur notre forum.

Discuter sur nos forums

Discuter sur Reddit

[NdT : Traduit avec l'aide de www.DeepL.com/Translator]