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

Friday Facts N°324
Conception sonore, Arbres animés, Optimisations

Posté par Jitka, Ian, Albert, Allaizn et Rseding

le 06/12/2019

Écussons avec logo Factorio (par Jitka)

Nous aimerions vous présenter nos nouveaux écussons Factorio en tissu, qui sont maintenant disponibles dans notre e-shop. Ces empiècements brodés et cousus sont idéaux pour les vêtements, les chapeaux, les sacs à dos, etc. Les dimensions sont de 2,5 x 12 cm.

Comme nous ne connaissons pas l'ampleur de la demande pour ces produits, nous n'avons pour l'instant qu'un stock limité disponible.

Veuillez noter que notre boutique en ligne n'expédie qu'une fois par semaine (tous les mercredis), et qu'il est fort possible que les commandes passées maintenant ne soient pas livrées avant le 25 décembre, ceci s'applique particulièrement aux commandes expédiées hors d'Europe.

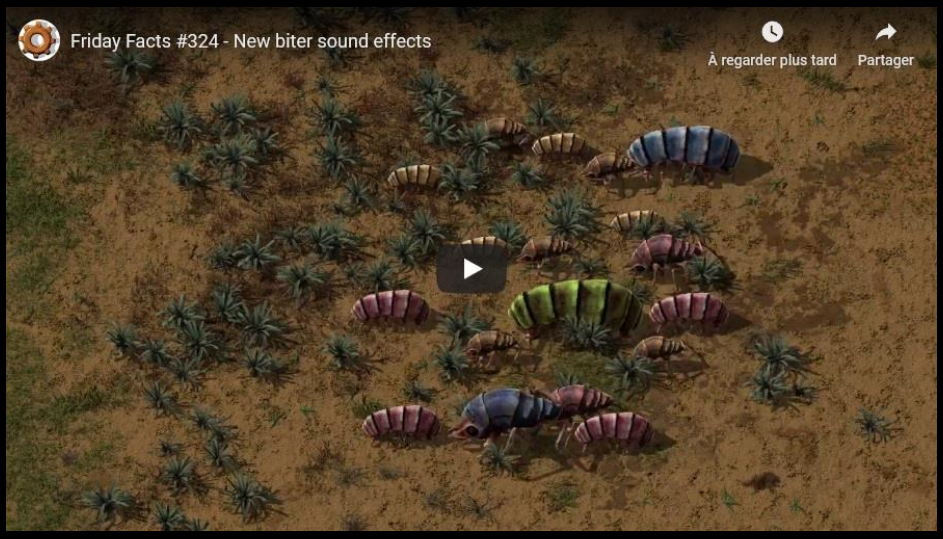
Conception sonore (par Ian)

J'ai été embauché chez Factorio pour finir le son du jeu pour la version 1.0. On a estimé qu'un concepteur sonore était nécessaire pour travailler au bureau de Prague, afin d'aider à la mise en œuvre des sons et à l'amélioration de la perception audio.

Avec le désir de faire des conquêtes immédiates, l'une de mes premières tâches a été d'ajouter le son des pas des ennemis, ce qui, selon nous, les rendrait vraiment vivants. Malheureusement, il s'est avéré que la technologie que nous voulions utiliser pour cela (chaque pas étant lié au bon terrain, comme les pas du joueur), allait être trop chère pour le jeu en termes de processeurs. Après tout, certaines de ces créatures ont 12 pattes. En me souvenant d'un cauchemar similaire avec le déplacement d'araignées géantes sur un jeu Harry Potter il y a des années, j'ai décidé de faire une solution plus simple. Ce que nous faisons maintenant, c'est de jouer un son ponctuel pour chaque cycle de l'animation de la marche.

Tout d'abord, j'ai commencé à chercher des sons dans une bibliothèque pour pouvoir faire rapidement un prototype. En prenant des craquements de coquilles d'œufs et en les ajoutant à une vidéo avec les animations de la marche, j'ai réussi à créer quelque chose d'assez bon pour les pas ou les mouvements des biters. J'ai l'intention d'enregistrer quelques sons supplémentaires pour ces derniers plus tard, mais pour l'instant ils fonctionnent bien.

Les sons sont différents pour chaque ennemi cependant, les mordeurs ont plus de craquements et les cracheurs ont des bruits plus sourds. Plus l'ennemi est grand, plus les pas sont grands. Ces sons devraient vous donner un plus grand sentiment d'immersion dans leur monde, lorsque vous les entendez se précipiter vers vous juste avant de les voir.



[NdT : cliquez pour voir la vidéo]

En ce qui concerne les autres sons ennemis, il me semblait que les cracheurs et les mordeurs se ressemblaient trop et c'est quelque chose que je voulais changer. Si l'on peut distinguer les sons émis par les ennemis, cela ajoute à la variété et aussi au plaisir. L'autre concepteur sonore, Val, a fait de nouveaux sons pour les cracheurs inactifs, afin de faire ressortir leur aspect visqueux et de les rendre plus répugnants. En attendant, j'ai ajouté ces sons au jeu, en les testant et en les ajustant. Par exemple, j'ai choisi de meilleurs sons pour les ennemis agonisants, afin de donner des informations plus claires au joueur quand il a réussi son tir.

Par ailleurs, j'ai pas mal travaillé sur le mixage du jeu dans son ensemble et l'amélioration des sons que je souhaitais, mais je vais laisser ça pour une autre fois !

Arbres animés. Évidemment (par Albert)

Dans le FFF de la semaine dernière, nous avons présenté l'eau animée comme une mini-série de "petits" ajouts sur les sensations de l'environnement.
Beaucoup de réaction sont venues me dire que maintenant les arbres ont l'air plutôt morts par rapport à l'eau. Nous le savions à l'avance, et il semble que vous lisiez dans nos pensées parce que pendant la préparation de l'eau, nous avons aussi travaillé sur les arbres. Aujourd'hui enfin nous pouvons présenter ce travail terminé.

Le son du vent qui accompagne toujours le jeu se ressent beaucoup mieux avec ces nouvelles animations, sans oublier les sons à venir. Les ombres projetées par les arbres sont également animées, et cela rend l'effet sur la surface de l'eau bien meilleur.

[NdT : cliquez pour voir l'animation]

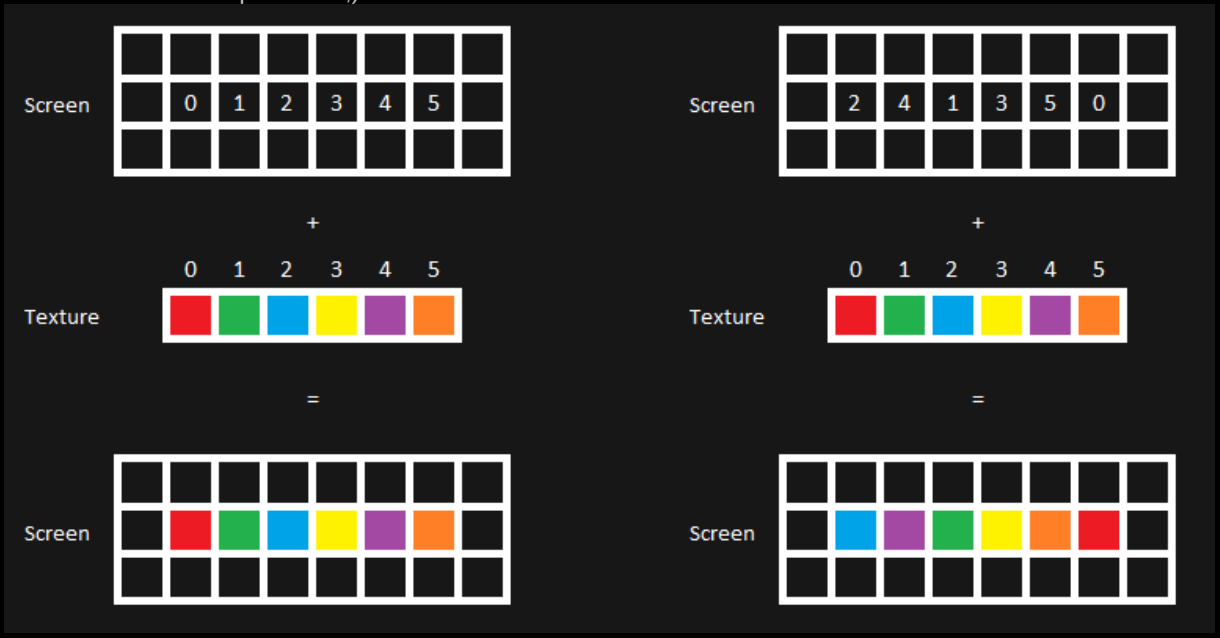
L'idée d'animer les feuilles des arbres est vieille comme Factorio, mais nous n'avons jamais eu le temps à cause d'autres priorités évidentes. Un jour, Tom ("wheybags") est venu avec un très beau modèle, et je me suis de nouveau très impliqué dans cette idée. Quelque temps après, Ernestas a réalisé un nouveau prototype basé sur différentes techniques. C'était aussi très intéressant. Le sujet bougeait assez fermement, mais ce n'était pas assez bien. Ensuite, Viktor ("Allaizn"), a eu l'idée d'utiliser les normal mappings des arbres au lieu d'un bruit générique pour déplacer les feuilles, et le résultat de cette expérience fut fantastique.

Le reste sera expliqué par Allaizn lui-même.

Intégration du gestionnaire de nuance pour les arbres - si cela fonctionne dès le premier essai, c'est que vous avez probablement oublié quelque chose (par Allaizn)

Ma première "grande" tâche a été d'intégrer le gestionnaire de nuances qu'Ernestas avait créé dans le moteur de jeu, ce qui était excitant du fait de son apparence, mais aussi un peu stressant vu que je ne regardais que rarement cette partie du code du jeu jusque-là. La première étape est généralement de comprendre comment il fonctionne, alors permettez-moi de vous donner une petite explication de ce qui se passe.

Le processeur graphique rend une texture pixel par pixel, et (en gros) chaque pixel ne connait d'abord que son emplacement à l'écran. Un gestionnaire de nuances est alors nécessaire pour lui donner les informations supplémentaires nécessaires pour arriver à la couleur qu'il est supposé avoir en fin de compte, où il agit un peu comme un jeu de coloriage avec des numéros - nous préparons une texture pour les couleurs à utiliser, et lui donnons quelques valeurs qui indiquent quelle partie de la texture est à utiliser (leur nom technique sont les coordonnées UV). Lors du rendu de sprites, nous souhaitons presque toujours transmettre ces nombres de manière à ce que la texture soit copiée à l'écran (voir l'image ci-dessous) - mais rien ne nous empêche de jouer avec ces nombres au préalable ;)



À gauche : vous voyez les nombres qui sont choisis pour obtenir une copie à l'écran de la texture fournie.
À droite : le résultat mélangé si vous fournissez juste des nombres aléatoires.

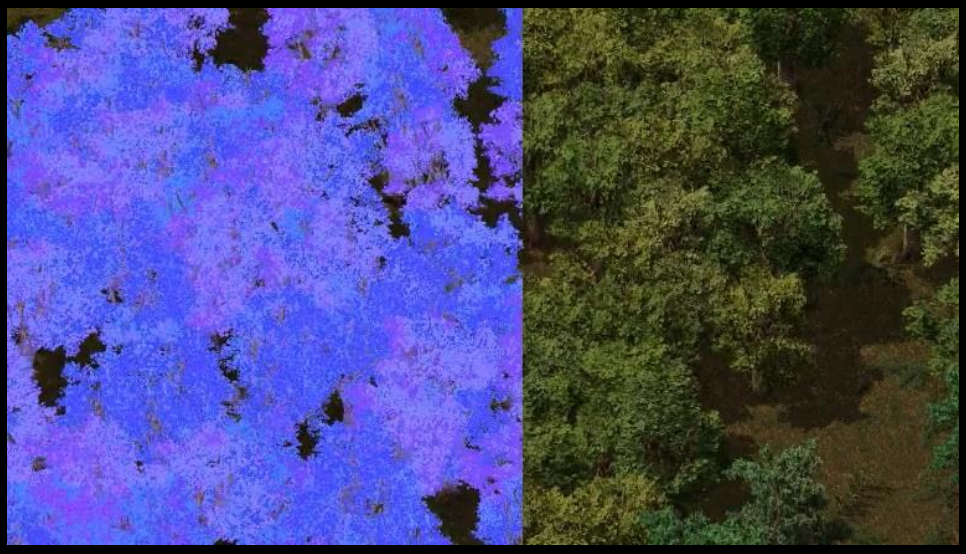
Passer en coordonnées UV aléatoires donnera généralement une image totalement méconnaissable, mais nous pouvons être plus rusés que cela : nous pouvons transmettre le nombre correspondant au pixel en-dessous du pixel habituel, ou à celui au-dessus - et si nous le faisons de manière stratégique, l'image semble un peu décalée. Varier ce décalage dans le temps, et le résultat est l'apparition d'un léger mouvement sur votre image ! Ce décalage d'un pixel ou deux est appelé distorsion, et il est généralement fourni au gestionnaire de nuances par une seconde texture (dont nous interprétons simplement les valeurs de couleur comme des valeurs de décalage nécessaires) appelée "modification de la distorsion".

Pour en revenir à l'implémentation, il a été étonnamment facile d'arriver à une version de travail puisque j'ai pu copier la partie la plus difficile (le gestionnaire de nuances lui-même) directement du prototype d'Ernestas - pour ensuite réaliser que le titre de ce paragraphe est presque toujours vrai ! Les arbres peuvent être dessinés de bien des façons :

- Textures à haute résolution par rapport aux textures normales.
- Compression de haute ou de basse qualité, ou aucune compression.
- Profondeur de couleur complète ou partielle.
- Les feuilles diminuent en quantité en raison des dommages dus à la pollution et ont en tout 4 stades.
- Les feuilles se dégradent avec l'augmentation de la pollution.
- Les arbres peuvent être rendus comme faisant partie du monde du jeu, ou comme faisant partie de l'interface graphique.

Si l'on ne tient pas compte de la dégradation, il y a donc 48 façons différentes de rendre le même sprite d'arbre, et nous voulons bien sûr qu'ils fonctionnent tous, ce qui me contraindra pendant quelques jours à chercher les cas oubliés.

Pendant ce temps, tout ne s'est pas bien passé : parfois tout semblait correct avec le code, mais les arbres semblaient refuser de bouger. Il y avait donc toujours la question de savoir si l'effet était actif ou s'il l'était, dans quelle mesure il l'était réellement. Cela m'a amené à écrire une visualisation de débogage dans le gestionnaire de nuances :



[NdT : cliquez pour voir l'animation]

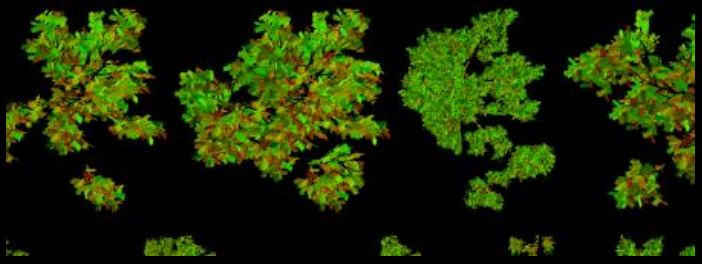
L'effet a été un peu accéléré pour faciliter la visibilité.

Les trois canaux de couleur encodent les trois propriétés principales avec lesquelles le gestionnaire de nuances doit travailler :

- Le canal rouge indique le déplacement dans le sens horizontal - pas de rouge signifie un décalage dans un sens, et du rouge dans l'autre.
- Le canal vert indique la même chose pour le déplacement vertical.
- Le canal bleu indique à quel point la distorsion doit être mise à l'échelle pour tenir compte des différentes résolutions de texture - le bleu profond n'entraîne pas de mise à l'échelle, le bleu à moitié correspond à une mise à l'échelle de 0,5, aucun bleu n'entraînera donc aucune distorsion, car il est à la valeur 0.

Le thème du titre m'a également interpellé d'une autre manière : l'effet dépend fortement de la carte de distorsion fournie, et notre première version a donné lieu à un look que l'on peut décrire comme "ça semble bien si vous le diminuez jusqu'à ce que ce soit quasiment invisible" - même la meilleure mise en œuvre au monde importe peu, aussi longtemps que le résultat final ne paraît pas super. Étant donné que la carte de distorsion initiale était principalement du bruit, j'ai plutôt essayé l'approche tout à fait opposée : trouver une texture qui est fortement corrélée avec la texture des feuilles de l'arbre et l'utiliser à la place - en se basant sur le normal mapping des feuilles d'arbre.

Le gestionnaire de nuances lui-même n'utilise que les canaux rouge et vert des cartes normales, ce qui en fait une bonne cible pour la compression BC5, que nous n'avions pas encore utilisée dans le jeu (voir FFF-281 pour en savoir plus sur la compression). Après un temps étonnamment court, la compression était opérationnelle - du moins je le pensais, puisque j'ai été une fois de plus frappé par "avez-vous pensé à ça ?". Le coupable cette fois-ci était le MIP mapping, qui n'était pas conscient de la compression et a donc réduit l'image compressée au lieu de la décompresser, la réduire et la re-compresser à nouveau.



Normal mappings tels que vus dans l'atlas des sprites

Quand le projet était presque terminé, j'ai été frappé une dernière fois par le sentiment que j'avais oublié quelque chose : le déplacement des feuilles d'arbres devrait faire bouger leurs ombres aussi, non ? J'ai donc passé un peu plus de temps à implémenter un gestionnaire de nuances spécial pour elles, qui fait exactement cela en utilisant le bruit généré au lieu d'une carte de distorsion.

Optimisations (par Rseding)

Il y a quelques parties clés dans la base de code qui finissent par être "lentes" par rapport à tout le reste et c'est la raison pour laquelle on simplifie les choses presque toujours au maximum.

"Pourquoi ..... est-il si lent ?" -> "Parce qu'il doit vérifier A, B, C et D à chaque fois qu'il fait n'importe quelle logique".
"Pourquoi ? Ça n'arrive presque jamais" -> "Parce que sans les contrôles, la logique est tout simplement fausse".

Il y a environ 3 ans, j'en avais ma version avec des bras. Les bras finissent par être l'une des entités les plus communes pour le simple fait que : vous devez les avoir si vous voulez que quelque chose fonctionne. Mais les bras font aussi partie de ces entités "plus lentes" alors que l'idée de base de ce qu'ils font semble si simple : "comment le déplacement d'objets de A à B en un arc de cercle peut-il se révéler si lent ?" (par rapport à tout le reste bien sûr).

J'ai donc regardé les résultats du profilage et le code qu'il indiquait :

- À chaque tick, vérifiez si le bras a une source d'énergie thermique. Si c'est le cas :
    - Vérifiez si la source d'énergie est complètement vide et endormez-vous si c'est le cas (sauf pour le carburant, sinon il ne pourrait pas le récupérer étant incapable de bouger)
    - Vérifiez si l'élément dans la pince peut être utilisé comme combustible pour cette machine. Si c'est possible :
    - Déplacez la pince vers le bras lui-même.
- À chaque tick, vérifiez si la destination a bougé (téléporté/véhicule parti).
- À chaque tick, vérifiez si la source a bougé (téléporté/véhicule parti).
- À chaque tick, vérifiez si la cible ou la source est marquée pour déconstruction.
- À chaque tick, vérifiez si la source a changé d'équipe
- À chaque tick, vérifiez si une condition d'activation existe et si le bras est désactivé par elle.

Si tout cela passe, déplacez la pince vers la position source/destination.

Il n'est pas surprenant que cela finisse par être "lent". Mais vous ne pouvez pas simplement ne rien faire de tout cela et dire "c'est bon". La réponse évidente à tout cela, c'est "ce n'est pas fréquent / ce sont des cas extrêmes ; ils devraient tous être faits par le biais d'événements". Mais alors, comment ? En d'autres termes, il n'y aurait aucun événement dont le bras pourrait se servir pour savoir quand ces choses se produisent ; il devrait donc vérifier chaque tick si l'une d'elles se produit ? J'ai fini par laisser tomber et je me suis remis à travailler sur d'autres choses. Mais ça m'est toujours resté dans la tête ; cuisiner - essayer de trouver une solution. 3 ans plus tard, je l'ai trouvé : les viseurs.

Événements pilotés par des viseurs

Nous avons créé et perfectionné les viseurs au fil des ans pour deux raisons principales :

1. Ce sont des pointeurs qui disparaissent ; quand la chose que vous "ciblez" est effacée, votre pointeur est effacé (défini sur `nullptr`).
2. Ce sont des pointeurs enregistrables/chargeables ; vous pouvez faire perdurer un pointeur via `save -> quit -> load`.

La plupart des choses dans le jeu qui ont besoin de "pointer" vers quelque chose d'autre les utiliseront (à quelques exceptions près bien sûr). Les bras les utilisent. Mon idée était assez simple : tout ce qui peut être "ciblé" peut passer sur tout ce qui peut le cibler et lui faire savoir quand un événement rare arrive (tout ce que le bras devrait vérifier et quelques autres). Les événements ne sont pas gratuits - mais parce que ces cas ne se produisent pas couramment, le fait de ne pas avoir à faire ces vérifications rend le coût des événements lorsqu'ils se produisent insignifiant dans les graphiques de performance globale.

Boule de neige

Avec la logique de mise à jour des bras considérablement simplifiée et avec les nouveaux événements pilotés par viseurs à ma disposition, j'ai commencé à remarquer des choses :

- Les foreuses minières partagent la plupart des mêmes vérifications que les bras - elles ont donc reçu le même traitement.
- Les locomotives, les wagons de marchandises et les wagons-citernes ont le même genre de vérifications ; ils n'ont donc plus besoin d'être actifs dans plus de 99 % des cas.
- Les triangles bleus demandeurs de modules ont le même type de vérifications ; ils n'ont donc plus besoin d'être actifs dans plus de 99% des cas.
- Les robots logistiques et de construction ont le même type de vérifications pour "la cible a-t-elle bougé ?" donc maintenant ils n'ont plus besoin de vérifier cela.

Après avoir fini avec ceux que j'ai retravaillés et vu que ces entités prennent beaucoup moins de temps, différentes choses intéressantes ont commencé à apparaître :

- La logique de la source d'énergie thermique faisait plusieurs vérifications lentes relatives à des comportements extrêmes.
- Les tapis effectuaient un contrôle de type O(N) à chaque fois qu'ils déplaçaient des objets alors que cela pouvait être fait en temps O(1).
- Tout ce qui produisait de la fumée provenant de la consommation d'énergie faisait plusieurs vérifications lentes pour essayer de produire de la fumée alors qu'ils n'en avaient pas besoin dans la plupart des cas.

Et finalement, une dernière chose est apparue : les conduites de chaleur. Chaque fois que je rends quelque chose plus rapide, quelque chose d'autre prend sa place dans le "temps passé sur chaque tick" (ce qui est prévisible), mais cela signifie aussi que cela révèle de nouvelles choses que je n'avais peut-être pas remarquées auparavant.

Conduites de chaleur

La première chose que j'ai remarquée avec les conduites de chaleur est que toute la logique réelle de l'écoulement des conduites de chaleur n'est même pas dans l'entité même des conduites de chaleur. L'entité ne fait que reporter la logique à la classe "`Heat Buffer`". Cela m'a fait me demander : pourquoi la logique de "mise à jour" passe-t-elle par l'entité si elle ne fait rien du tout ? Quelques jours plus tard et beaucoup plus de codes que ce que j'avais prévu d'écrire ; j'ai déplacé toute la logique de mise à jour du flux thermique dans sa propre classe de gestionnaire dédiée (presque identique à la façon dont les fluides et le flux électrique ont une classe de gestionnaire).

Cela semblait trop beau pour être vrai ; ce qui valait 0,55 ms/tick affichait maintenant 0,17 ms/tick (un peu plus de 3 fois plus vite) simplement en ne passant pas par l'entité pour faire circuler la chaleur à chaque tick. Beaucoup de tests plus tard et les résultats étaient corrects ; c'était juste beaucoup plus rapide. L'algorithme sous-jacent n'a pas changé mais il a juste fonctionné > 3x plus vite en affectant moins de mémoire maintenant. C'est un autre bel exemple que "Factorio n'est pas lié au processeur, c'est lié à la latence de la mémoire". Plus de cœurs n'iraient pas plus vite - parce qu'il n'a jamais été limité par la vitesse à laquelle le processeur fonctionne.

Conclusion

Réseaux électriques.... Réseaux de fluides.... Réseaux de conduites de chaleur.... aucun d'entre eux n'interagit entre eux ou avec quoi que ce soit en dehors d'eux-mêmes. Que se passe-t-il si je mets à jour les 3 en parallèle ? Comme on pouvait s'y attendre, chacun d'entre eux est devenu légèrement plus lent (parce qu'ils se disputent l'accès à la mémoire), mais dans l'ensemble, c'est quand même plus rapide, de façon mesurable. Ce qui est intéressant, c'est que l'un des trois prend toujours beaucoup plus de temps à terminer que les autres. Cela signifie que les autres finissent par être essentiellement "disponibles" ; le jeu devant de toute façon attendre que le plus lent soit terminé, si bien que le 2 des 3 plus rapides obtiennent un "tour gratuit" pour avoir fini bien avant que le jeu ne finisse, attendant que le plus lent se soit terminé.

Chaque fichier de sauvegarde que j'ai testé a fini par s'exécuter sensiblement plus vite à la fin. Le plus extrême (beaucoup de fonte d'acier dans des fours) a montré une accélération de 2,3 fois.

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]