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

Friday Facts N°302
Le méga-paquet multijoueur

Posté par Twinsen et Klonan

le 05/07/2019

Le méga-paquet multijoueur (par Twinsen)

Le mois dernier, j'ai participé à l'événement massivement multijoueur de KatherineOfSky en tant que joueur. J'ai remarqué qu'après avoir atteint un certain nombre de joueurs, toutes les quelques minutes, un groupe d'entre eux se faisait exclure. Heureusement pour vous (mais malheureusement pour moi), j'ai été l'un des joueurs qui ont été déconnectés à chaque fois, même si j'avais une bonne connexion. J'ai donc pris l'affaire personnellement et j'ai commencé à examiner le problème. Après 3 semaines de débogage, de test et de correction, le problème est enfin résolu, mais le voyage n'a pas été si facile.

Les problèmes multijoueurs sont très difficiles à déceler. Habituellement, ils ne se produisent que dans des conditions de réseau très spécifiques, dans des conditions de jeu très spécifiques (dans ce cas, avoir plus de 200 joueurs). Même lorsque vous pouvez reproduire le problème, il est impossible de le déboguer correctement, car placer un point d'arrêt arrête le jeu, perturbe les chronomètres et interrompt généralement la connexion. Mais grâce à une certaine persévérance et grâce à un outil génial appelé **clumsy**, j'ai réussi à comprendre ce qui se passait.

La version courte est la suivante : à cause d'un bug et d'une implémentation incomplète de la simulation d'état de latence, un client se retrouvait parfois dans une situation où il envoyait un paquet au réseau d'environ 400 actions de sélection d'entités en un seul tick (ce que l'on appelle le 'méga-paquet'). Le serveur doit alors non seulement recevoir correctement ces actions en entrée, mais aussi les envoyer à tous les autres. Cela devient rapidement un problème lorsque vous avez 200 clients. Il sature rapidement le flux sortant du serveur, provoque la perte de paquets et provoque une cascade de paquets redemandés. Les actions d'entrée retardées provoquent alors l'envoi de méga-paquets par un plus grand nombre de clients, avec encore plus d'effets en cascade. Les clients chanceux réussissent à se rétablir, les autres finissent par être largués.

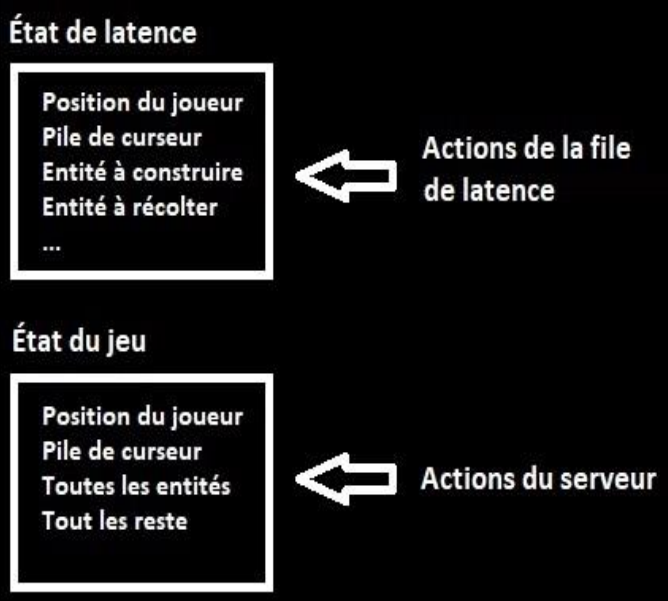
Le problème était tout à fait fondamental et il a fallu deux semaines pour le régler. C'est assez technique, donc je vais vous expliquer avec des détails techniques juteux ci-dessous. Mais ce que vous devez savoir, c'est que depuis la sortie de la version 0.17.54 hier, le jeu multijoueur sera plus stable et le masque de latence sera beaucoup moins problématique (moins de tiraillement élastique et de téléportations) si des problèmes

temporaires se produisent dans les connexions. J'ai aussi changé la façon dont le masque de latence est géré en combat, en espérant qu'il ait l'air un peu plus lisse.

Le méga-paquet multijoueur : La partie technique (par Twinsen)

Le principe de base de notre mode multijoueur est que tous les clients simulent l'état du jeu et qu'ils ne reçoivent et n'envoient que les entrées du joueur (appelées **Actions d'entrée**). La principale responsabilité du serveur est d'effectuer les **actions d'entrée** par délégation et de s'assurer que tous les clients exécutent les mêmes actions dans le même tick. Plus de détails dans la FFF-149

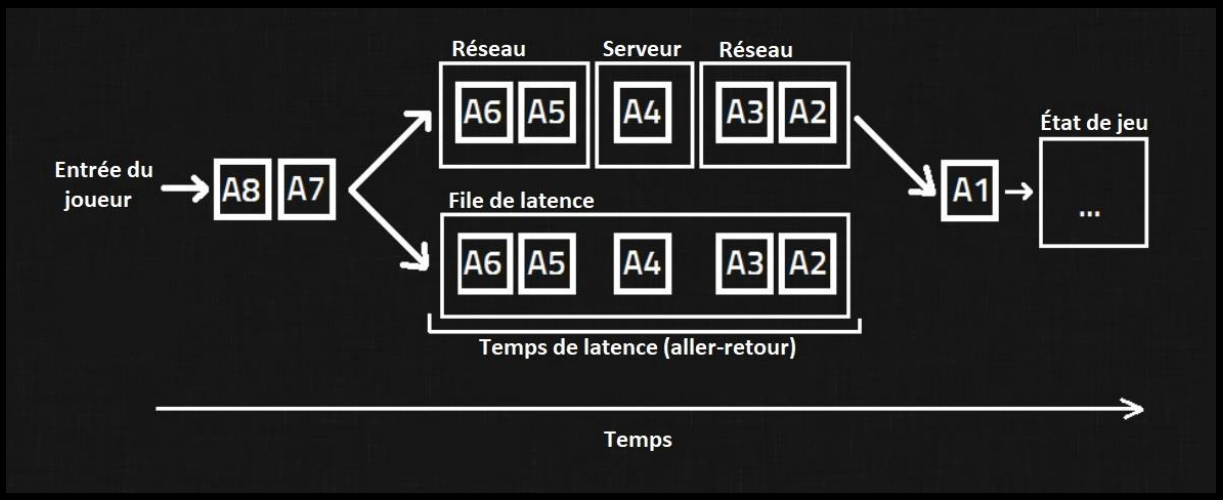
Puisque le serveur a besoin de choisir le moment où les actions sont exécutées, une action du joueur se déplace comme ceci : Action du joueur => Client de jeu => Réseau => Serveur => Réseau => Client de jeu. Cela signifie que chaque action du joueur n'est exécutée qu'une fois qu'il a fait un aller-retour sur le réseau. Cela rendrait le jeu vraiment lent, c'est pourquoi le masque de latence a été un mécanisme ajouté dans le jeu presque depuis l'introduction du jeu multijoueur. Le masque de latence fonctionne en simulant l'entrée du joueur, sans tenir compte des actions des autres joueurs ni de l'arbitrage du serveur.



Dans Factorio nous avons l'**État du jeu**, c'est l'état complet de la carte, le joueur, les entités, tout. Il est simulé de façon déterministe sur tous les clients, en fonction des actions reçues du serveur. C'est sacré et si jamais c'est différent du serveur ou de tout autre client, une désynchronisation a lieu.

En plus de l'**État du jeu**, nous avons l'**État de latence**. Il contient un petit sous-ensemble de l'état principal. L'**État de latence** n'est pas sacré et il représente simplement à quoi nous pensons que l'état du jeu ressemblera dans le futur en fonction des **actions d'entrée** effectuées par le joueur.

Pour ce faire, nous conservons une copie des **actions d'entrée** que nous effectuons, dans une **file de latence**.



En fin de compte, le processus, du côté du client, ressemble à ceci :

1. Appliquer toutes les **actions d'entrée** de tous les joueurs à l'**État du jeu**, tel qu'il est reçu du serveur.
2. Supprimez toutes les **actions d'entrée** de la file de latence qui ont été appliquées à l'**État du jeu**, en fonction du serveur.
3. Supprimez l'**État de latence** et réinitialisez-le pour qu'il ressemble à l'**État du jeu**.
4. Appliquer toutes les actions de la file de latence à l'**État de latence**.
5. Restituer le jeu au joueur, en se basant sur les informations de l'**État du jeu** et de l'**État de latence**.

Ceci est répété à chaque tick.

Ça se complique ? Accroche-toi à ton pantalon. Pour tenir compte du manque de fiabilité des connexions Internet, nous avons deux mécanismes :

- Saut de tick : Quand le serveur décide quelles **actions d'entrée** seront exécutées dans quel tick de jeu, s'il n'a pas les **actions d'entrée** d'un certain joueur (par exemple à cause d'un pic de retard), il n'attendra pas, mais dira au client "Je n'ai pas inclus vos **actions d'entrée**, je vais essayer de les inclure lors du tick suivant". Ainsi, lorsqu'un client a des problèmes de connexion (ou des problèmes informatiques), il ne ralentira pas la mise à jour de la carte pour tout le monde. Notez que les **actions d'entrée** ne sont jamais ignorées, elles sont seulement retardées.
- Latence aller-retour : Le serveur essaie de deviner quel est le délai aller-retour entre le client et le serveur, pour chaque client. Toutes les 5 secondes, il négociera une nouvelle latence avec le client, si nécessaire, en fonction du comportement de la connexion dans le passé et la latence aller-retour sera augmentée et réduite en conséquence.

En soi, ils sont assez simples, mais lorsqu'ils se produisent ensemble (ce qui est courant lorsqu'il y a des problèmes de connexion), la logique du code commence à devenir pesante, avec un grand nombre de cas limites. De plus, le serveur et la file de latence doivent injecter correctement une **action d'entrée** spéciale appelée `StopMovementInTheNextTick` [NdT : Arrête le Mouvement Au Prochain Tick] lorsque les mécanismes ci-dessus entrent en jeu. Cela empêche votre personnage de courir seul (par exemple devant un train) en cas de problème de connexion.

Maintenant, il est temps d'expliquer comment fonctionne notre sélection d'entités. L'un des types d'**action d'entrée** que nous envoyons est le changement de sélection d'entité, qui indique à chacun sur quelle entité chaque joueur a sa souris. Comme vous pouvez l'imaginer, c'est de loin l'action d'entrée la plus courante envoyée par les clients, donc elle a été optimisée pour utiliser le moins d'espace possible, pour économiser la bande passante. La façon dont cela a été fait est que chaque sélection d'entité, au lieu d'enregistrer des coordonnées absolues et de haute précision de la carte, enregistre un écart relatif de faible précision par rapport à la sélection précédente. Cela fonctionne bien, car une sélection est généralement très proche de la sélection précédente. Cela crée 2 exigences importantes : Les **actions d'entrée** ne peuvent jamais être ignorées et doivent être exécutées dans l'ordre correct. Ces exigences sont remplies pour l'**État du jeu**. Mais, puisque le but de l'**État de latence** est de paraître "assez bien" pour le joueur, ces exigences n'étaient pas remplies. L'**État de latence** n'a pas pris en compte de nombreux cas extrêmes liés aux sautes de ticks et aux changements de latence aller-retour.

Vous pouvez donc probablement voir où cela nous mène. Enfin, la question du méga-paquet a commencé à apparaître. Le dernier problème était que la logique de sélection de l'entité s'appuyait sur l'**État de latence** pour décider s'il devait envoyer une action de sélection modifiée, mais l'**État de latence** ne possédait pas toujours les informations correctes.
Donc, le méga-paquet a été généré quelque chose comme ça :

1. Le joueur a des problèmes de connexion.
2. Le saut de tick et les mécanismes d'ajustement de latence aller-retour commencent à s'activer.
3. La file d'État de latence ne tient pas compte de ces mécanismes. Ceci conduit à une action supprimée prématurément ou exécutée dans le mauvais ordre, ce qui conduit à un **État de latence** incorrect.
4. Le joueur récupère de son problème de connexion et simule jusqu'à 400 ticks afin de rattraper son retard sur le serveur.
5. Pour chaque tick, une nouvelle action de changement de sélection d'entité est générée et préparée pour être envoyée au serveur.
6. Le client envoie au serveur un méga-paquet avec plus de 400 changements de sélection d'entités (et d'autres actions également. L'état de tir, l'état de marche, etc. ont également souffert de ce problème).
7. Le serveur reçoit 400 actions d'entrée. Puisqu'il n'est pas autorisé de sauter une action d'entrée, il indique à tous les clients d'exécuter ces actions et les envoie sur le réseau.

Ironiquement, le mécanisme qui était censé économiser de la bande passante réseau a fini par créer des paquets de réseau massifs.

Dans ce but, cela a été résolu en corrigeant tous les cas limites de mise à jour et de maintien de la file de latence. Bien que cela ait pris un certain temps, en fin de compte, cela valait probablement la peine de faire une correction appropriée au lieu de faire quelques manipulations rapides.

Clusterio - Le Gridlock Cluster (par Klonan)

Clusterio est un système scénario/serveur qui ajoute la communication des données de jeu entre différents serveurs. Par exemple, en envoyant des articles entre différents serveurs, vous pouvez avoir un "serveur minier", qui exploitera tout le minerai de fer dont vous avez besoin, et l'envoyer au "serveur fonderie" qui le fondra et le fera passer plus loin.

L'année dernière, il y a eu un événement utilisant le système et qui a relié plus de 30 serveurs ensemble pour atteindre un objectif combiné de 60 000 potions par minute. MangledPork a une vidéo du début de l'événement de l'année dernière sur YouTube.

Les grands esprits et les communautés à l'origine du dernier événement se sont réunis de nouveau pour accueillir un autre événement du Clusterio : Le Gridlock Cluster. L'objectif cette fois-ci est de repousser encore plus loin les limites, d'explorer et de coloniser de nouveaux nœuds au fur et à mesure qu'ils sont générés, et de relever le défi de construire une méga-usine répartie sur plusieurs serveurs.

Si vous êtes intéressé à participer à cet événement communautaire, tous les détails sont listés dans le message Reddit, et vous pouvez rejoindre le serveur Discord du Gridlock Cluster.

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

Discuter sur nos forums

Discuter sur Reddit