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

Friday Facts N°349
Le plan 1.0

Posté par Klonan, Rseding et Boskid

Le 29/05/2020

Bonjour, nous avons aujourd'hui de grandes nouvelles.

Le plan 1.0 (par Klonan)

Dans le FFF-321, nous avons annoncé une date de sortie pour la version 1.0. Compte tenu des événements récents, nous avons décidé de modifier la date de la version 1.0. La nouvelle date que nous visons est le vendredi 14 août 2020, soit 5 semaines plus tôt que la date initiale.

La principale raison de changer la date de sortie est la sortie de Cyberpunk 2077. En janvier de cette année, CD Projekt Red a annoncé un report de la sortie de Cyberpunk 2077 au 17 septembre, soit 1 semaine avant le lancement de Factorio 1.0. Nous pensons que toute sortie proche d'un jeu aussi monumental va avoir des effets négatifs, comme le fait que tout le monde joue et couvre Cyberpunk et détourne l'attention des autres jeux.

Nous avons donc pensé qu'il était préférable d'essayer de sortir avant Cyberpunk ou assez longtemps après. Compte tenu de ces deux choix, nous avons choisi d'avancer la date de sortie. Il y a plusieurs raisons pour lesquelles nous avons choisi de publier plus tôt :

Objectifs revus

Lorsque nous avons annoncé la date (FFF-321), nous avions prévu que de nombreux éléments figureraient dans la version finale. Les principaux sujets étaient la nouvelle Campagne, l'amélioration de l'algorithme des fluides et la réécriture complète de l'interface graphique. Pour des raisons indépendantes, nous avons annulé la nouvelle Campagne (FFF-331), reporté les améliorations de l'algorithme des fluides et réduit de nombreux aspects de la réécriture de l'interface graphique (FFF-348).

Le respect du calendrier

Outre la révision de certaines fonctionnalités, les autres travaux que nous avons réalisés ont progressé à un bon rythme. La structure de la version expérimentale 0.18 (FFF-314) nous aide vraiment à garder les choses sur la bonne voie. L'estimation initiale a été faite avec une certaine concession pour les retards, à savoir que "les choses prennent toujours plus de temps que prévu". Eh bien, ces six derniers mois, la plupart des choses n'ont pas pris plus de temps que prévu, et nous avons terminé les sujets assez efficacement.

Le plus tôt sera le mieux

Le sentiment général au bureau est que le jeu est pratiquement terminé et que nous voulons le sortir le plus rapidement possible. Plus vite nous aurons terminé la version 1.0, plus vite nous pourrons commencer à penser à de nouvelles choses amusantes et excitantes.

En raison de la coïncidence de l'annulation de plusieurs grands titres, nous pouvons donc nous permettre d'avancer la date de sortie. Pour être clair, nous n'avons annulé ou reporté aucune fonctionnalité en raison de la date de sortie de Cyberpunk.

Cette nouvelle date de sortie nous donne 10 semaines, et à partir de ce moment et jusqu'au vendredi 14 août, l'équipe se concentrera principalement sur la finalisation du jeu, la mise à jour de la bande annonce et la préparation des supports marketing.

Cliquez pour voir en haute résolution

Projet et gel des traductions (par Klonan)

Une partie de la finalisation du jeu consiste à finaliser les traductions communautaires du jeu. Depuis longtemps, nous utilisons Crowdin pour la recherche de toutes les traductions. Crowdin a très bien fonctionné et est profondément intégré dans notre flux de travail (FFF-48).

Toutefois, certaines langues ne sont pas couvertes à 100 % et il n'y a pas eu de relecture globale. C'est pourquoi nous avons choisi de rechercher une société de traduction professionnelle pour nous aider à combler les lacunes et à tout relire. Nous avions spécifiquement besoin d'une société qui travaillerait par l'intermédiaire de Crowdin, car la communauté a des années d'expérience dans ce domaine et le système n'aura pas besoin de gestion de notre part.

Après avoir consulté de nombreuses entreprises et de nombreux autres développeurs de jeux pour connaître leur avis, nous avons décidé de nous associer avec Altagram, basée à Berlin, en Allemagne.

La dernière mise à jour de l'interface graphique étant terminée, nous avons gelé les traductions, ce qui signifie qu'il n'y aura plus d'ajouts ni de modifications (dans la mesure du raisonnable). Il y aura une relecture des textes sources en anglais, et ensuite une relecture de toutes les langues cibles.

Pour une clarté absolue, Altagram a détaillé le projet et le processus de leur point de vue :

> *Une fois que nous aurons le feu vert que la communauté a terminé ses contributions aux traductions dans Crowdin, nous commencerons la relecture du texte source anglais afin d'apporter toutes améliorations grammaticales ou stylistiques nécessaires. Une fois que l'anglais aura été peaufiné, nous commencerons la relecture des langues secondaires.*
>
> *Les linguistes, qui sont tous des joueurs eux-mêmes, et des experts en traductions de jeux, feront travailler leur magie pour s'assurer que la langue cible est aussi fidèle que possible au texte source, pour garantir que tous les joueurs de Factorio, quelle que soit la langue dans laquelle ils jouent, auront la même expérience.*
>
> *Les linguistes vérifieront certains éléments lors de la relecture de la langue cible : s'assurer que le texte lui-même, en particulier tous les termes du jeu, est cohérent dans son ensemble, que l'orthographe et la grammaire sont correctes et que les traductions ont le même sens et la même émotion que ceux prévus. Après avoir proposé nos suggestions, les textes seront renvoyés à la communauté pour leur approbation finale avant d'être mis en œuvre dans le jeu.*

Jusqu'à la version 1.0, le gel des traductions signifie principalement que nous ne travaillerons que sur des sujets qui ne nécessitent pas de nouvelles chaînes de caractères, tels que les corrections de bugs, les nouveaux graphiques, la conception sonore, etc.

Interface graphique de l'explorateur de prototypes / Interface graphique des prototypes (par Rseding)

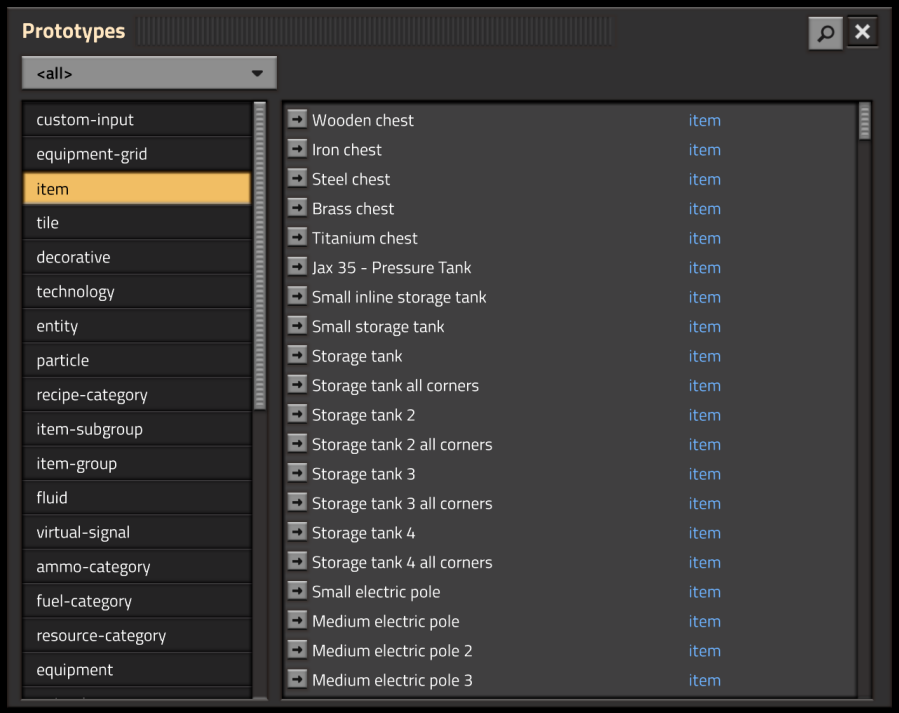
Inspiration

Factorio a intégré de nombreuses fonctionnalités et outils de débogage au fil des ans. Certains sont très utilisés (voir les FPS/UPS) et pour d'autres nous nous demandons comment nous avons pu nous en passer (infobulle d'inspection de style d'interface graphique). Chacun d'entre eux a été ajouté dans un but précis et ont fini par offrir une utilité bien plus grande que leur but initial. En gardant cela à l'esprit, et parce qu'ils finissent aussi par être très amusants (pour moi), je travaillais à résoudre un problème que j'ai trouvé avec la logique de l'infobulle d'inspection de style d'interface graphique et je me suis dit : ne serait-il pas agréable d'avoir quelque chose comme ça pour tous les prototypes du jeu ? Est-ce que c'est quelque chose que je pourrais faire réellement ? De quoi cela aurait-il l'air, comment cela gérerait-il tous les imbrications qui se produisent... mais cela avait l'air amusant.

J'ai donc décidé de voir quelle utilité une telle chose pouvait avoir :

- Déterminer si un mod a modifié ou changé quelque chose - ou s'il était censé le faire et ne l'a pas fait (ce qui est courant dans les rapports de bugs avec des mods et pendant le développement d'un mod)
- Fournir un endroit pour extraire des informations que le jeu ne montre nulle part ailleurs (tout n'est pas exposé à travers l'API du mod, et il n'est pas réaliste de s'attendre à ce que quelqu'un se souvienne de l'API entière)
- Lien vers le Wiki expliquant plus en détail les concepts du jeu.

La liste des avantages semblait valoir la peine que l'on bricole au moins l'idée.



Conception technique

La première partie que j'ai dû comprendre était la suivante : comment allais-je faire pour que tout soit transféré dans une interface graphique ? Factorio est écrit en C++ et le C++ n'a pas de réflexion. Il n'y a pas de moyen facile de dire "pour toutes les variables que cette chose a, faites ceci". En fait, la seule façon de tout couvrir est d'envoyer chaque chose dans l'interface graphique. Ce n'est pas joli, mais nous ne modifions pas non plus fréquemment les prototypes à ce stade du développement. De plus, si quelque chose "ne va pas", cela ne provoque pas de crash ou d'erreurs ; c'est une solution facile que tout le monde peut faire. Beaucoup de travaux de codage ennuyeux plus tard, cette partie a été traitée.

Rien n'est jamais facile ni simple

Pour chaque chose à afficher : afficher le nom, afficher la valeur, afficher le type. Cela semblait simple, mais ça ne l'est jamais.

- Le type peut être incroyablement verbeux et ou simplement inutile pour un humain : qu'est-ce que cela signifie ? "`class std::basic_string,class std::allocator` >" (c'est une chaîne de caractères...)
- La valeur peut être énorme - il a donc fallu créer des regroupements.
- La valeur peut être un tableau de quelque chose, donc "vide" doit être indiqué pour les tableaux vides
- Les tableaux de choses ayant une valeur de 1 ne devraient en réalité que montrer cette valeur de 1
- Les éléments optionnels doivent être marqués "vide" lorsqu'ils ne sont pas définis
- Certaines choses sont liées à d'autres, il fallait donc créer des liens
- Tout cela doit fonctionner à n'importe quel niveau hiérarchique.

Mais c'est Factorio, et c'est le vernis qui le rend agréable à utiliser. Je ne regrette rien de tout cela.

Liens vers le Wiki

Aux premières étapes du développement, j'ai décidé que le moyen le plus simple de faire comprendre à toute personne utilisant ce système ce qu'est un "type" et comment il est censé être utilisé, est de montrer la page Wiki à ce sujet. Le Wiki contient des informations très détaillées sur une grande partie de ce que cela va montrer et il semblait logique de l'utiliser. Mais je ne voulais pas coder les liens en dur... ça ne finit jamais bien.

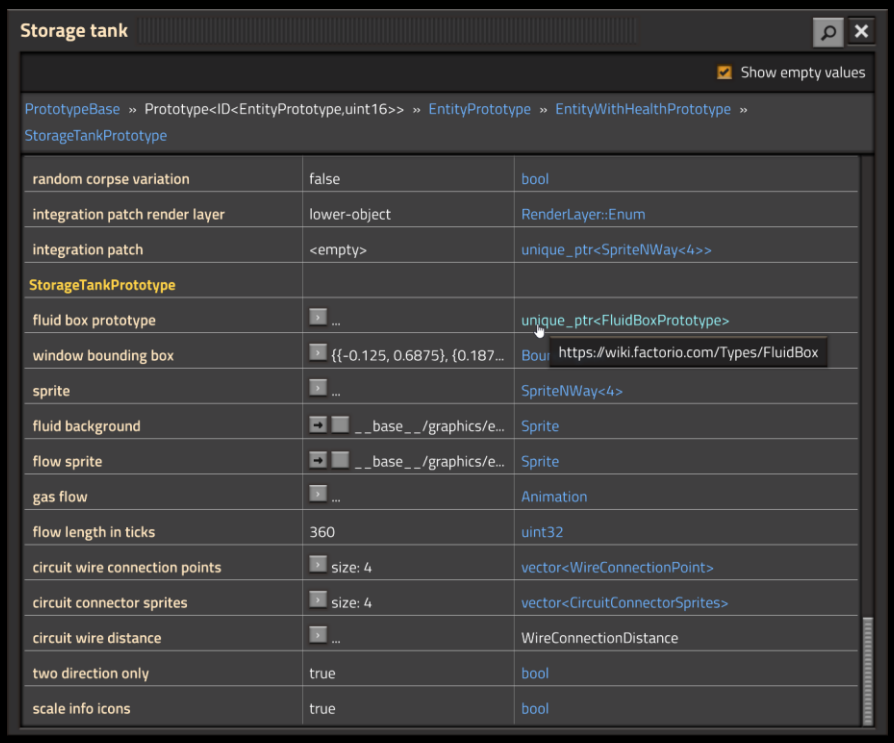
Mon idée : une page sur le Wiki qui fournit une cartographie des types du jeu => adresse web du Wiki. Le jeu téléchargerait la cartographie et, en remplissant l'interface graphique, s'il trouvait un type qui existait dans la cartographie, il le relierait au Wiki. Bilka a rapidement mis en place la partie Wiki. Du côté du jeu... "rien n'est jamais facile ou simple".

- Je ne voulais pas télécharger la page Wiki à chaque fois que l'interface graphique s'ouvre - ce serait du gaspillage
- Je ne voulais pas télécharger la page Wiki à chaque lancement du jeu - la page ne change pas fréquemment et ce serait donc du gaspillage
- Mais la page doit aussi être re-téléchargée lorsqu'elle est modifiée
- Je ne voulais pas que le jeu s'arrête pendant le téléchargement
- Elle doit être tolérante aux erreurs (tout le monde n'a pas de connexion internet, ou on ne peut même pas dire que la page se téléchargera correctement)

Et ainsi, un simple "lien du type vers le Wiki" s'est transformé en :

- Il se souvient de la dernière fois qu'il a téléchargé la page Wiki et de l'identifiant de révision
- Il ne tente de télécharger l'identifiant de révision que la première fois qu'une des interfaces graphiques est ouverte
- Si l'identifiant de révision a changé, ou s'il ne dispose pas de la cartographie en local, il essaie de télécharger la dernière cartographie Wiki
- Si le téléchargement réussit, il l'enregistre pour la prochaine fois et remplit les liens de l'interface graphique
- Si l'un d'entre eux échoue, il enregistre ce qui s'est passé et continue à fonctionner silencieusement (c'est l'Internet après tout, on s'attend à des échecs aléatoires).
- Tout se passe en arrière-plan pour que le jeu ne s'arrête pas pendant que cette logique fonctionne

Et tout cela fonctionne parfaitement.



Interpréteur Lua en natif (par Boskid)

La semaine dernière, Rseding m'a demandé d'étudier comment nous pourrions mettre en œuvre un interpréteur de table Lua performant, car nous interprétons souvent sur des tables Lua du côté C++, et cette opération est lente. Cela m'a obligé à apprendre quelques principes internes de Lua et comment il stocke les tables. Jusqu'à présent, lorsqu'une carte était sauvegardée et que l'état Lua devait être parcourus, nous utilisions `serpent.dump` (de la bibliothèque Serpent) pour convertir la variable appelée en global en une chaîne du côté Lua, puis la sortir et la stocker dans la sauvegarde.

Comme il était assez facile de parcourir les tables Lua depuis le côté C++, j'ai décidé de mettre en place, pour une expérience, un interpréteur Lua en natif. Cela nous a permis d'éviter complètement l'utilisation de `serpent.dump` et à la place de les sauvegarder directement. Mon objectif premier était de réduire le temps de chargement puisque dans l'ancien format, les données sauvegardées étaient une chaîne de caractères que Lua devait analyser et exécuter.

Comme on l'a remarqué plus tard (pas par moi), la vitesse de sauvegarde s'est beaucoup améliorée du fait qu'aucune opération Lua n'est exécutée pendant la sauvegarde, juste un pur parcours sur les données à sauvegarder en un temps linéaire.

Pour les mesures, j'ai utilisé un fichier de sauvegarde qui contient beaucoup de données de script (`script.dat` fait environ 60 Mo), le résultat est la moyenne sur 3 essais :

- Sauvegarde :
  - Ancien = 285,429 s
  - Nouveau = 2,847 s
- Chargement :
  - Ancien = 47,034 s
  - Nouveau = 22,755 s

Cette valeur inclut également certaines optimisations mises en œuvre par Rseding.

Le changement de l'interpréteur a cependant un coût. `Serpent.dump` faisait aussi l'interprétation des fonctions Lua stockées dans les variables globales. Elles n'étaient de toute façon pas officiellement prises en charge par nous, mais certains mods les utilisaient "puisqu'elles semblent fonctionner". Avec le nouvel interpréteur, j'ai décidé de ne pas l'implémenter du tout en raison de sa complexité et de ses limites inhérentes (les fermetures étaient de toute façon cassées). Cela a cassé certains mods (même certains scénarios du jeu de base) mais c'est assez facile à réparer.

Une autre considération que nous avions était de savoir s'il devait échouer à sauvegarder lorsqu'il y a des fonctions Lua en global pendant la sauvegarde, ou s'il devait les supprimer silencieusement (comme cela arrive avec les métatables). La première approche a été considérée comme la meilleure pour détecter rapidement tous les mods non conformes, mais nous avons ensuite décidé que les supprimer lors de la sauvegarde (et fournir quelques informations dans le fichier de log) était préférable car une migration était presque impossible à cause d'une migration du jeu de base pour la version 0.18.28 qui demande de recharger tous les scripts, ce qui a pour effet secondaire de sauvegarder l'état Lua, ce qui annule immédiatement la sauvegarde en raison des fonctions Lua encore présentes dans les variables globales.

Discutez sur notre forum

Discutez sur Reddit

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