Projet Prog 2 - Troisième rendu

Grégoire DHIMOÏLA, Léo JUGUET, Tristan PARCOLLET

16 mai 2023

1 Introduction

Ce rapport contient les avancées dans notre jeu vidéo, ainsi que les plans pour l'avenir. Pour rappel, le but du projet est de réaliser un jeu de type RTS fortement inspiré du jeu Faster Than Light et adapté aux besoins d'un RTS.

2 Le jeu final

2.1 Vue générale

Le jeu consiste en plusieurs clans qui s'affrontent dans l'espace avec pour but la destruction de la base adverse. Le joueur contrôle l'un des clans. Chaque clan a à sa disposition une base, quelques vaisseaux-mères, et une flotte de drones/petits vaisseaux. Le monde contiendra diverses ceintures d'astéroïdes, parmis lesquels certains seront minables.

2.2 Les différentes unités

Les vaisseaux-mères peuvent être aménagés via un menu adapté et c'est au joueur de les spécialiser ou non. Ils peuvent installer des modules parmis les suivant :

- Les modules de combat consisteront simplement en différentes armes et upgrades.
- Les modules de soutien serviront à la maintenance des drones et des vaisseaux-mères, et potentiellement réhabiliter les épaves, ou encore à générer un bouclier.
- Les modules constructeurs serviront à la construction des drones avec diverses caractéristiques
- Les modules de minage permettront d'être plus efficace dans le raffinement des matériaux et ainsi d'en extraire plus par astéroïde. Il peuvent également servir à scrapper les épaves.

Les drones sont extrêmement basiques et peuvent être controlés individuellement ou en groupe via une sélection apropriée. Ils peuvent attaquer un vaisseau ou un groupe de vaisseaux ennemis, miner des astéroides, et transférer des ressources entre eux, la base, et les vaisseaux-mères.

2.3 La gestion de la base et des ressources

La base consiste en un astéroïde plus gros que les autres et aménagé pour servir de QG à l'état-major des armées en jeu. Elle servira de point de ralliement pour tous les vaisseaux, et permettra de stocker les ressources récoltées. Ce sera éventuellement elle qui permettra de construire les vaisseaux-mères.

La déstruction d'un vaisseau génère une épave, minable également, et servant à récupérer principalement du scrap. Elle pourra aussi être réhabilitée par des modules de réparation pour un coût très réduit.

Les ressources seront utiles pour faire fonctionner les vaisseaux-mères, en particulier l'uranium et l'etherum. Les autres ressources servent à la construction ou à la maintenance.

3 Ce qui est actuellement implémenté

Cette section est une réminiscence de son analogue du rapport précédent. Cependant, elle peut être résumée en disant simplement que tout est implémenté, excepté les quelques points évoqués en section 5.

Actuellement, toutes les actions et tous les comportements sont implémentés même si tous ne sont pas utilisés pour la démo. De même, tous les outils utiles au jeu sont développés. Les drones peuvent attaquer, miner, transférer des ressources. Les vaisseaux mères peuvent simplement se déplacer, c'est à leurs modules de faire des actions. Les actions des modules sont fonctionnelles, mais le controller n'y a pas encore assigné d'input.

Tous les types de boutons et de menus qui seront utiles au joueur sont implémentés. En particulier, le menu de construction de modules et d'intéraction avec eux, pour ce qui concerne les vaisseaux mères, et les boutons qui servent notamment à la navigation entre les différents menus et à la sélection de certaines actions.

La gestion des événements, que ce soit les événements d'input classiques du joueur, ou bien des événements personnalisés, est en place. Typiquement, des acteurs peuvent s'abonner à des événements personnalisés, qui appeleront automatiquement une certaine fonction lors de leur réalisation.

La génération d'aléatoire cohérente qui sert à la génération et au placement des astéroides, des unités et de tous les objets, ainsi que de l'arrière plan à l'initialisation de la partie est implémentée.

Les outils utiles au contrôle et au bon déroulement du jeu sont implémentés. Il y a le gamestate, la caméra, les controllers, le manager de textures, la GUI, le générateur d'aléatoire et le manager d'événements.

Les unités ont un comportement de groupe pour ce qui est des déplacement (voir section 4.6) ce qui rend les déplacements de nombreuses unités bien plus fluide et organique.

Les modules peuvent maintenant être construits de manière arborescente (voir section 4.4), ils ne sont plus que des champs du vaisseau mère.

L'initialisation de la partie et l'IA (sections 4.2 et 4.3) sont disponibles et fonctionnels, mais ont été retirés de la démo actuelle pour des raisons de performances.

4 Ce qui a été ajouté ou modifié depuis le dernier rendu

4.1 Controller

Le controller de l'IA a subit quelques améliorations pour que ses actions soient moins aléatoires, et toujours valides, mais reste globalement assez stupide. Il est à noter que cela est pour des raisons de démonstrations et de performances. Nous avons réalisés de nombreux tests d'IA plus avancés qui sont décrits ci-dessous, mais qui ralentissaient tellement le jeu qu'ils n'ont pas été gardé pour ce rendu.

Ils seront réintégrés lorsque de réelles structures de données auront été implémentées, en particulier pour l'ordonnancement spatial. Ceci permettrait de se faire une véritable idée de l'état actuel et local de la situation en un temps raisonnable. Une section plus bas explorera des idées pour tenter de régler ce problème.

Cependant voici quelques notes sur ce qui a été expérimenté :

Décision de la probabilité de faire certaines actions plutôt que d'aurtes basé sur l'état local du monde :

si l'IA a très peu de ressources et beaucoup de vaisseaux capitaux à faire tourner, alors il lui faut un certain nombres d'unités dédiées au minage pour assurer au moins une entrée constante de ressources.

Si un drone voit dans sa proximité un trop grand nombre d'ennemis par rapport aux alliés, alors il aura plus tendance à fuir vers le "centre de gravité local" des unités alliées.

A l'inverse, si un combat est en cours et que les alliés ne sont pas en train de le perdre unilatéralement, il aura de grandes chances de décider d'aller se battre.

Si rien ne se passe localement, alors l'IA décidera d'envoyer un certain nombre d'unités en éclaireurs pour chercher des ressources intéressantes, ou pour chercher l'ennemi par exemple.

D'autres comportements encore ont été testés, comme le regroupement ou la dispertion, la gestion des attaques de groupe etc... Mais comme tous nécessitent un certain niveau de connaissance de la proximité, ils sont tous beaucoup trop

coûteux pour le jeu en l'état.

Il est important de noter que le fait de décider d'une certaine mesure de probabilité plutôt que d'un choix brut est très utile et facile. En effet, comme dans le jeu final, le joueur et l'IA seront amenés à contrôler un grand nombre d'unités, cela permet de garantir que tous les rôles sont à peu près distribués selon les besoins actuels. De plus, en terme de calcul pure, il "suffit" de calculer le nombre d'occurences de certains événements, ce qui en retour augmente ou diminue certains compteurs sur les actions possibles, et enfin on normalise le tout pour avoir une distribution de probabilité.

On se rend également compte qu'il est possible d'encore réduire le coût en calcul, puisqu'une unité très proche d'une autre aura quasiment la même vision locale. On peut donc chercher des heuristiques qui prennent en compte les probabilités déjà calculées pour les voisins et les modifient légèrement sans faire l'entièreté des calculs normalement nécessaires.

Avoir une probabilité permet donc une certaine cohérence et continuité qui accélère le temps de calcul sur toutes les unités.

4.2 Map

Depuis la dernière fois, notre génération d'aléatoire nous a permis de gérer l'initialisation de la map, avec les placements d'astéroides, de ressources, et de bases. Un exemple d'une telle disposition est montré en figure 1. Les astéroides sont placés en formations qui ressemblent à des ceintures, et qui entourent des régions plus vide de la carte. Les ressources sont dispersées de manière pseudo cohérente, en filon, au sein des champs d'astéroides.

Pour cela, il suffit de placer les astéroides dans une bande de valeur (typiquement, $0.5 \pm \epsilon$) sur un bruit de perlin.

Les bases quand à elles sont placées aux points les plus éloignés de tout astéroide au départ. Ceci se fait sans coût supplémentaire en terme de calcul. En effet, il suffit de les placer aux points d'intensité la plus faible et la plus forte dans le bruit qui a permis de générer les astéroides.

Quelques drones initiaux sont placés autour de la base.

Cependant, cela génère un grand nombre d'unités et d'astéroides, ce qui fait énormément lagger le jeu. Le code de cette partie est donc commenté, mais fonctionnel, comme pour la section précédente sur l'IA.

4.3 Modules

Nous n'avons pas changé le comportement des modules en soi, qui était déjà finalisé. Cependant, nous avons rendu la construction de modules beaucoup plus libre. En effet, désormais les vaisseaux mères sont un assemblage de modules choisit par le joueur. Ainsi, le joueur pour construire un vaisseau mère, assemble des modules entres eux pour donner les fonctionnalités et la forme qu'il souhaite au vaisseau.

Leur comportement prenait déjà en compte le fait qu'ils n'ont pas la même position que le vaisseau lui même, et donc il faudra maintenant bien faire attention au contrôle du vaisseau mère pour que ses modules soient à portée de leur cible.

4.4 Collisions

Le système de collisions a été refait presque intégralement.

Les collisions précédentes implémentaient la physique de collision avec perte d'énergie entre deux corps solides, tous les corps étant considérés circulaires.

Cependant, de part l'absence de path finding, il s'est avéré que ce n'était pas une bonne idée d'un point de vue gameplay, car il arrivait très souvent qu'un vaisseau se retrouve coincé.

De plus, graphiquement, l'angle d'un vaisseau suit le vecteur vitesse, et de telles collisions provoquent des changements instantanés de vitesse trop importants. Dans certains cas, cela rendait une impression désagréable que le vaisseau "glitchait" alors même que ce n'était pas le cas, il était simplement coincé.

Pour remédier à cela, nous avons considéré des collisions avec un "champ de force" répulsive croissant avec l'inverse de la distance. Cela revient, au point de contact, aux collisions précédentes, avec une force répulsive infinie, mais ça rend le processus beaucoup plus lisse et sans accoups.

Cependant, il reste le problème de l'absence de path finding. En effet, même si on a lissé les collisions, il n'en reste pas moins que les acteurs se retrouve quasi systématiquement coincés, et que pendant les déplacements, des collisions ont lieu trop souvennt ce qui entrave la progression. Pour corriger ce problème nous avons implémenté un comportement qui se rapproche de celui des oiseaux dans un murmure.

4.5 Murmure

Pour gérer le grand nombre de drones à disposition du joueur et de l'IA pendant leurs déplacements, nous nous sommes inspirés du vol des oiseaux. En effet, il est très joli, cohérent, et permet d'éviter les crash tout en suivant un algorithme simple.

Chaque unité suit donc les règles de bases suivantes avec différents coefficients pour mettre à jours sa vitesse :

- Se rapprocher du centre de gravité local,
- Se rapprocher de la cible finale,
- Aligner sa vitesse avec les unités locales,
- Éviter les collisions avec les unités locales,
- Ajout d'un brin de chaos.

Ceci plus l'introduction de bornes maximales sur chaque composantes permet de régler le problème du path finding, puisqu'il est possible de traverser les autres unités. Les "collisions" ne sont qu'une force répulsive mais il n'y a aucun "mur" infranchissable.

Ici aussi, la complexité est au carré du nombre d'unités, mais comme ce ne sont que des opérations locales, une bonne structure de donnée permettrait d'accélérer les calculs drastiquement.

4.6 Épaves

Les épaves sont finalisées. Lorsqu'un drone meurt, il génère une épave à la bonne position et orientation. Elle est minable.

4.7 Bugfixes

Le jeu ne plante plus à la destruction d'un drone. Les combats fonctionnent donc correctement.

Les successions automatiques d'actions fonctionnent (par exemple mining $\rightarrow$ move (si pas assez proche) $\rightarrow$ mining $\rightarrow$ iddle (si ressource épuisée)). Une mauvaise succession de fonctions les rendait inopérantes.

5 Ce qu'il reste à faire

- Intégrer le temps, et pas juste les frame,
- Des structures pour ordonner spatialement les unités et accélérer les points 4.2 (IA) et 4.5/6 (collisions et murmures). Une premère idée serait de découper la map en chunks, mais ce n'est pas très efficace,
- Implémenter des animations,
- Ajouter des projectiles visuels,
- Ajouter des modules de boucliers,