FTL : Fit the fittest
Rapport du TPA
Semestre 2

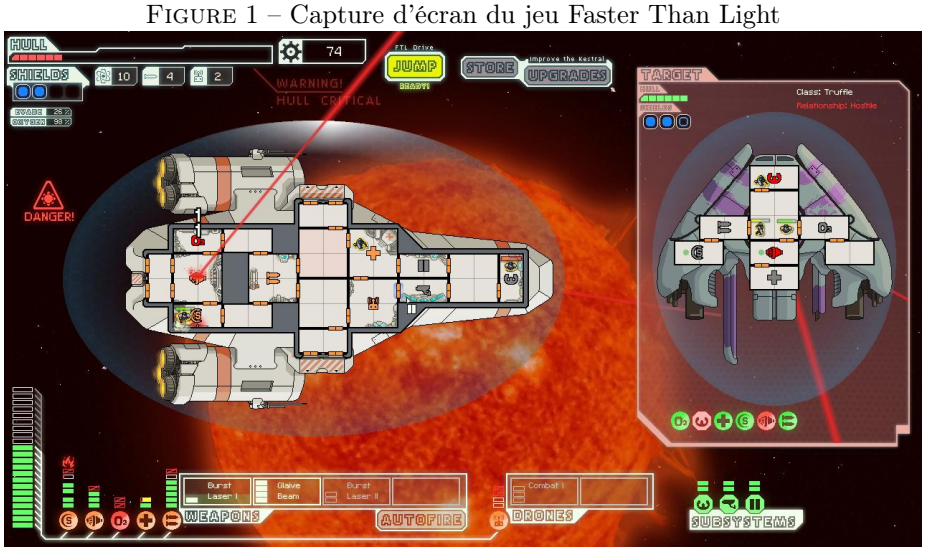
Sébastien Gamblin Simon Bihel Julien Pezant
Paul Leménager Josselin Gueneron

Table des matières

Introduction

FTL : Faster than Light (http://www.ftlgame.com/) est un jeu de science fiction de Subset Games pour PC, mélangeant la gestion, la stratégie et le jeu de rôle à bord d'un vaisseau spatial. Le joueur a pour objectif de conduire son vaisseau d'un point A à un point B de la galaxie, tout en faisant face aux multiples situations qui se présentent à lui aléatoirement. Il doit ainsi survivre face à de nombreux événements : affronter des adversaires, négocier avec d'autres vaisseaux, échanger des ressources, acheter de l'équipement, etc. Le but est donc de rester le plus longtemps en vie pour aller le plus loin possible dans l'espace. Ainsi, le principe primaire du jeu est de s'adapter à un environnement hostile en optimisant son vaisseau, ses équipements (tels que des drônes ou de nombreuses armes), et son équipage qui est composé de plusieurs races possédants chacune des avantages et des inconvénients.

FIGURE 1 – Capture d'écran du jeu Faster Than Light



Ci-dessus (figure 1), nous pouvons voir une capture d'écran du jeu Faster Than Light présentant un vaisseau durant un combat. Nous pouvons y observer la structure géographique d'un vaisseau, avec la distribution de l'énergie dans ses systèmes en bas à gauche.

Au cours du premier semestre, nous avons réussi à réaliser un moteur de combat qui, même s'il n'est pas complet, se rapproche du jeu original, ce qui nous permet maintenant de finaliser notre projet : optimiser la recherche d'un

vaisseau meilleur que les autres.

L'objectif de ce semestre est donc de faire un programme qui détermine le meilleur vaisseau pour un coût maximal donné. Pour cela, on peut séparer le projet en deux parties : faire un générateur de vaisseaux selon un coût donné et faire un algorithme qui détermine le meilleur vaisseau.

Dans ce rapport nous rappellerons le fonctionnement du moteur de combat qui a reçu quelques ajouts durant le second semestre. Ensuite, nous verrons en détail les techniques employées dans la construction de vaisseaux et la recherche du meilleur vaisseau. Enfin, nous pourrons parler de la construction du programme, étudier les résultats donnés par le programme et exprimer quelques critiques.

Chapitre 1

Moteur de combats

1.1 Représentation des vaisseaux

On peut distinguer deux types de représentations qui sont complémentaires. En premier lieu il y a la représentation des vaisseaux de base, ensuite il y a la représentation des vaisseaux personnalisés, c'est-à-dire les vaisseaux auxquels on a ajouté ou enlevé des armes, améliorations, etc.

Dans la représentation de base on stocke différentes données :
- les systèmes présents et leurs niveaux de base (c'est-à-dire le nombre maximal d'énergie que l'on peut mettre dedans) ;
- les armes, les drones et les améliorations ;
- les salles avec : leurs coordonnées, le système qui est dedans ou celui qui peut être dedans, les portes, les membres d'équipage.

Ensuite, dans la représentation des vaisseaux personnalisés, on stocke les ajouts et suppressions d'équipement. Ces ajouts et suppressions correspondent aux interactions possibles que l'on peut avoir dans les magasins dans le jeu original.

1.2 Le déroulement d'un combat

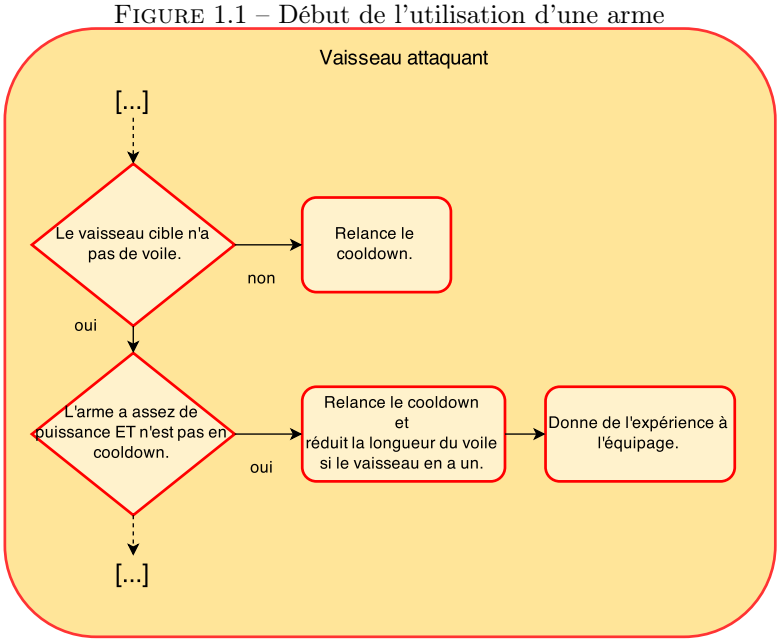
Nous avons décidé de gérer un combat avec un fonctionnement au tour par tour tout en gardant un faible temps entre chaque tour. Cela nous permet de gérer les phases d'attaques et de cooldowns indépendamment, tout en restant proche du jeu original qui se déroule en temps réel.

Le vaisseau gagnant est celui qui n'est pas mort ou celui qui a le plus de vie au bout de 5 secondes. Si les deux vaisseaux ont le même nombre de points de vie, c'est le vaisseau numéro 1 qui est désigné gagnant. Un combat dure en moyenne haute 2 secondes, 5 secondes permettent donc des combats plus longs entre des vaisseaux bien équipés.

1.2.1 Le déroulement d'une attaque

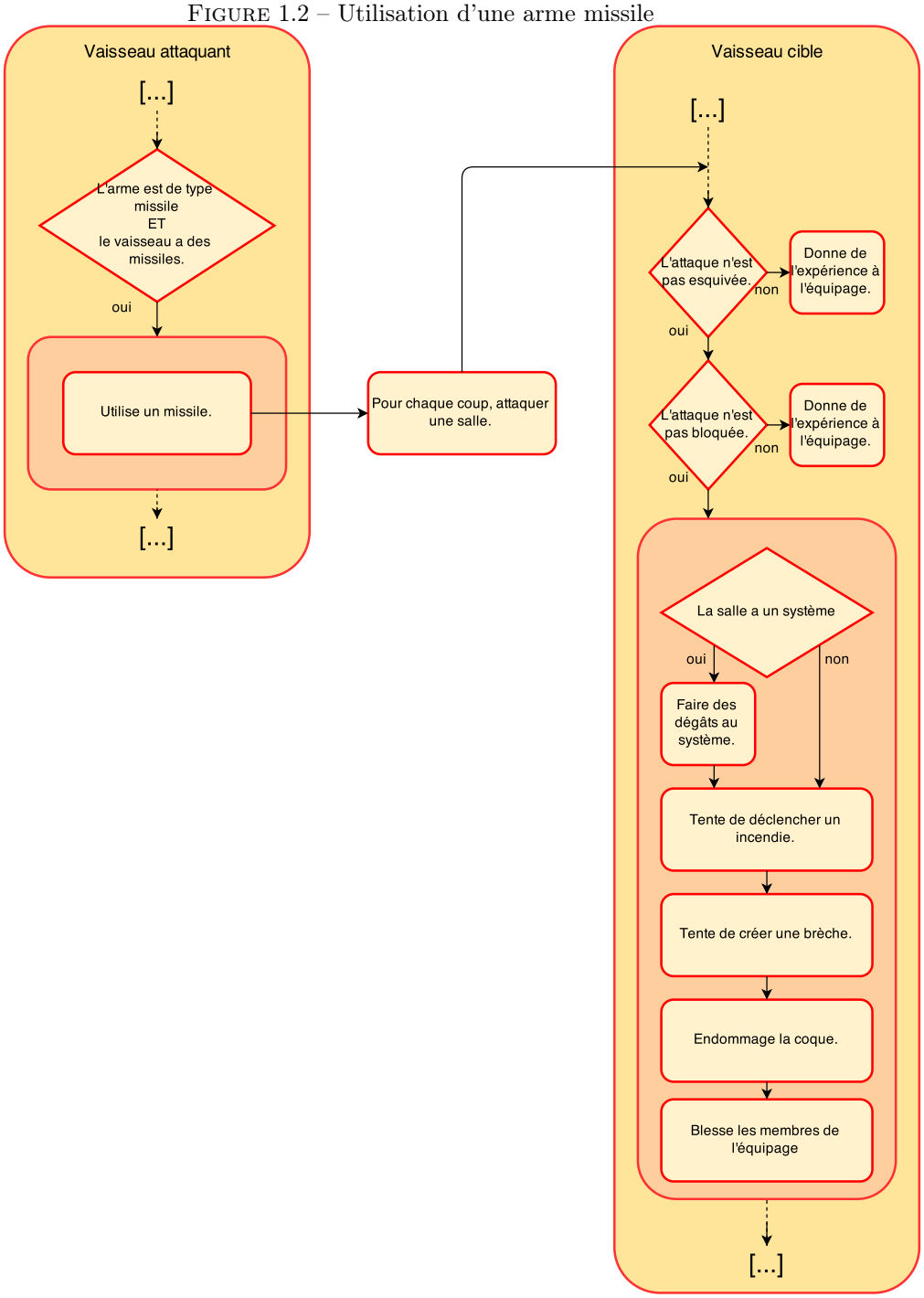
L'utilisation d'une arme Pour chaque arme, la même procédure va être appliquée. Celle-ci est gérée par le système des armes.

FIGURE 1.1 – Début de l'utilisation d'une arme



Comme décrit sur la figure 1.1, le système fait d'abord tous les préparatifs avant d'être sûr qu'il a le droit d'utiliser l'arme. Ensuite si toutes les conditions sont remplies, il l'utilise. Les différents types d'armes ont des fonctionnements différents.

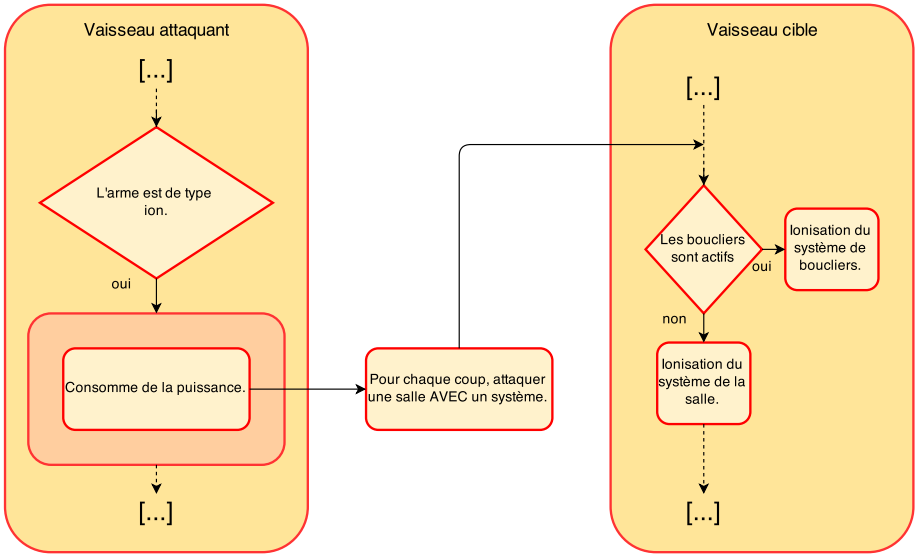
FIGURE 1.2 – Utilisation d'une arme missile



Les armes lasers et missiles ont des fonctionnements similaires (figure 1.2). Seul le déclenchement de l'utilisation d'un laser est différent puisqu'il ne faut pas vérifier qu'il y a des missiles en stock et qu'il ne faut pas utiliser de missile

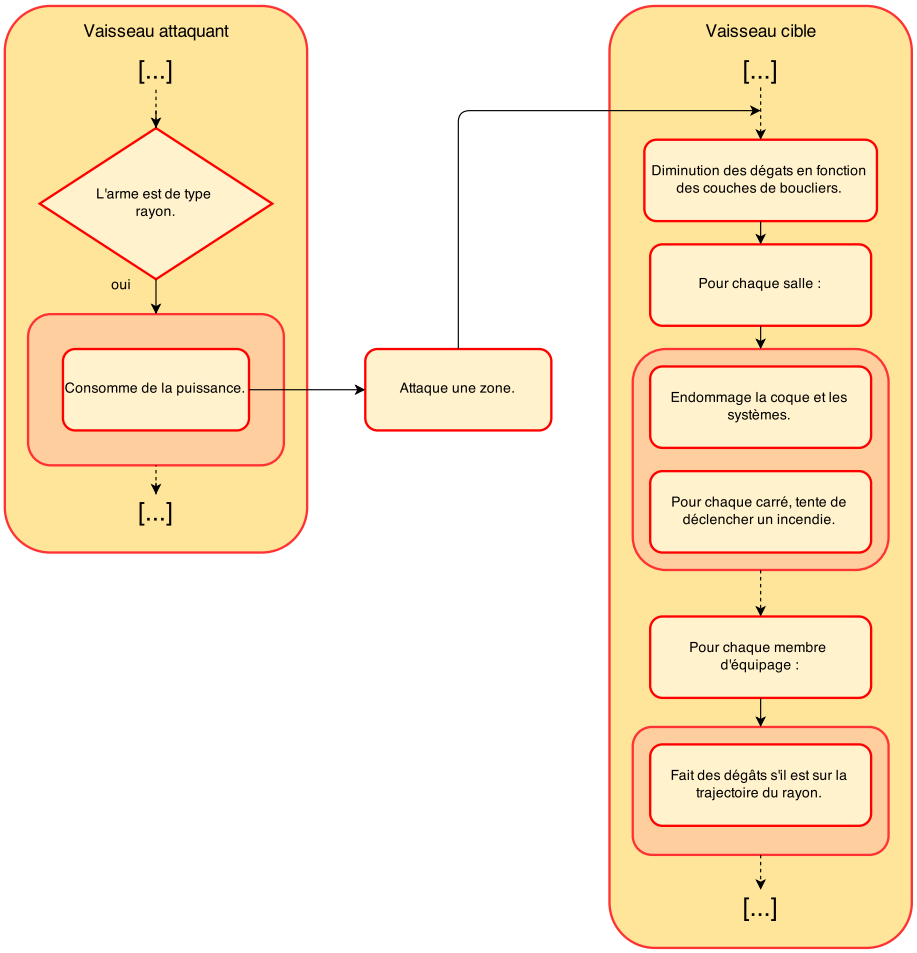
lorsque l'attaque est déclenchée. Ensuite la même méthode du vaisseau adverse est appelée.
Celle-ci manipule d'abord l'esquive et les boucliers. Ensuite, si l'attaque est passée, des dégâts sont appliqués aux systèmes, aux membres d'équipage présents dans la salle, à la coque et des départs de feu ou des brèches peuvent être générés.

FIGURE 1.3 – Utilisation d'une arme à ion



Les armes ions font des dégâts temporaires aux systèmes et seulement à eux (figure 1.3).
Pour utiliser ces armes, il suffit donc d'appeler la méthode du vaisseau adverse. Celle-ci va juste se contenter de d'ioniser les boucliers s'il y a une couche de bouclier opérationnelle, sinon ioniser le système visé originellement.

FIGURE 1.4 – Utilisation d'une arme à rayon



Les rayons font des dégâts sur une ligne dans le vaisseau grâce à des coordonnées aléatoires (figure 1.3).
La méthode du vaisseau adverse devra donc faire des dégâts à tout ce qui se trouve sur le chemin du rayon.

L'utilisation d'un drone Les seuls drones gérés pour l'instant sont des drones d'attaque qui ont un comportement similaire à l'utilisation des armes. À chaque fois qu'ils sont rechargés et qu'ils sont activés par le système des drones, ils attaquent une cible déterminée de la même façon que pour les armes.

1.2.2 La gestion des cooldowns

Le vaisseau L'ordre des différentes instructions n'a pas vraiment d'importance. Nous allons donc lister ces instructions :

1. Demander aux systèmes de gérer leurs cooldowns.

2. Essayer d'alimenter les différents systèmes, armes et drones.
3. Annuler les réparations effectuées d'un système s'il n'y a plus personne dans la salle. (Ce fonctionnement n'est pas optimal et pourrait être changé si les membres d'équipage se déplaçaient car dès qu'un membre quitte une salle on pourrait déclencher la remise à zéro des réparations)
4. Demander aux salles de gérer leurs cooldowns.
5. Demander aux membres d'équipage de gérer leurs déplacements (cette partie ne fonctionnant pas très bien, et n'étant pas la priorité du programme, nous avons décidé de mettre cet élément du jeu en pause).
6. Ajouter de l'oxygène aux salles grâce au système.
7. Essayer d'utiliser l'invisibilité.

Les systèmes Tous les systèmes doivent gérer le cooldown de leur ionisation.
Le système des boucliers doit gérer le cooldown de ses couches de bouclier.
Les systèmes des armes et des drones doivent recharger respectivement les armes et les drones.

Les salles

1. Gérer les différentes tâches des membres d'équipage présents (réparation, extinction...).
2. Gérer les dégâts faits aux membres d'équipage.
3. Gérer le reste de l'oxygène.
4. Gérer les feux et les brèches.

1.3 La discrétisation des évènements

Cette gestion des attaques et des cooldowns implique qu'à chaque tour le vaisseau vérifie plusieurs tests, incrémente des compteurs, etc. Cela prend du temps et n'est forcément utile. Pour éviter cela, nous avons discrétisé certains évènements. Cela consiste à avoir une file d'évènements (implémentée sous la forme d'une liste) dont les évènements de la tête sont effectués durant un tour et on supprime ensuite la tête pour avoir les évènements suivants au tour suivant.

Dans le moteur de combats, chaque vaisseau a deux listes d'évènements, une pour les attaques et une pour les cooldowns. Parmi les évènements discrétisés il y a :

— l'utilisation des drones ;
— l'utilisation de l'invisibilité ;
— l'ionisation des systèmes.

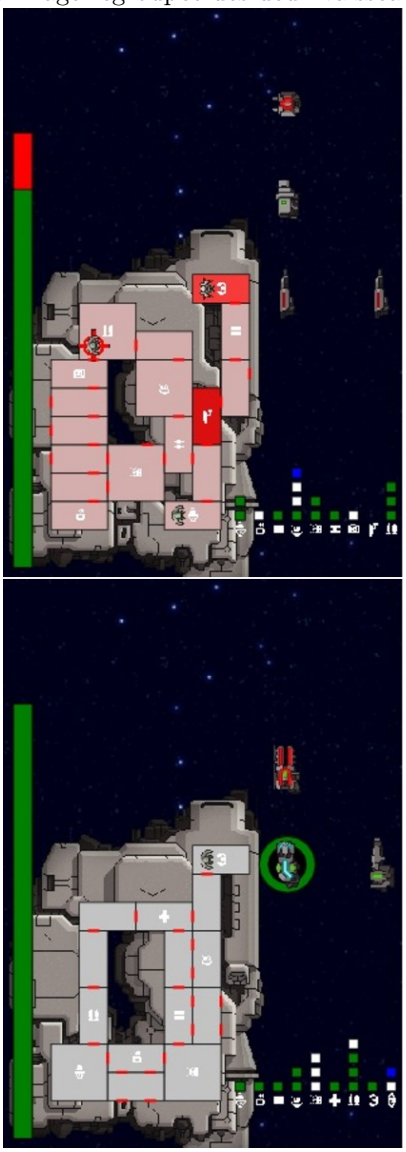
Au final, peu d'évènements ont été discrétisés car la plupart des évènements gérés par notre moteur de combats ont des interactions régulières avec d'autres éléments du vaisseau. Par exemple, le rechargement des armes est affecté par la présence ou non d'un membre d'équipage, donc à chaque tour il faudrait vérifier si un membre d'équipage est présent et vérifier son niveau, pour ensuite déplacer le rechargement de l'arme dans la liste d'évènements ce qui prend du temps. Ces cas sont communs chez les évènements de base du jeu originel, or notre moteur

de combats est composé en grande partie par les aspects basiques du jeu, cela explique donc la petite utilisation de la discrétisation des évènements. De plus, si le moteur de combats est enrichi avec des évènements avancés et donc avec peu d'interactions, le temps moyen d'un combat n'augmentera presque pas.

1.4 Représentation graphique

Optionnellement, nous avons ajouté un système de représentation graphique d'un combat entre deux vaisseaux. En effet cet ajout ne sert pas à l'exploitation des données, et nous permet simplement d'obtenir un aperçu graphique d'un duel. Ces aperçus sont faits grâce à la librairie python *tkinter*.

FIGURE 1.5 – Une image regroupée des deux vaisseaux à un temps donné



Malheureusement l'utilisation que nous voulions en faire en départ n'a pas pu être fait comme nous le souhaitions. En effet, de base nous voulions que le combat s'affiche dans une fenêtre qui s'updaterait. Cependant la personnalisation des classes des fenêtres tkinter ne permet pas l'affichage qui nous permet de faire cela (ou en tout cas, nous n'avons pas réussi à comprendre pourquoi ces affichages ne fonctionnaient pas). C'est pourquoi nous avons opté pour une seconde méthode, un peu plus longue. C'est le fichier scriptps.sh qui permet cela.

1. Dans un premier temps, un combat est lancé.
2. Pour chaque action, et chaque vaisseau, une fenêtre tkinter s'ouvre présentant l'état des deux vaisseaux.
3. Un fichier postscript est extrait de chaque fenêtre tkinter.
4. Chaque fichier postscript est transformé en jpg
5. Toutes les images jpg sont regroupées par deux, regroupant ainsi les deux vaisseaux à un temps donné. (figure 1.5)
6. Enfin, toutes les images sont regroupées en un seul gif.

Toutes les manipulations d'images sont faites par le script bash utilisant le logiciel ImageMagick

1.5 Diverses techniques de jeu

Buffer Cette technique consiste à avoir le niveau d'un système plus haut que nécessaire pour qu'une ionisation ou que des dégâts faits au système ne soient pas trop handicapants car le fonctionnement du système touché ne sera pas affecté.

Chapitre 2

Recherche du meilleur vaisseau

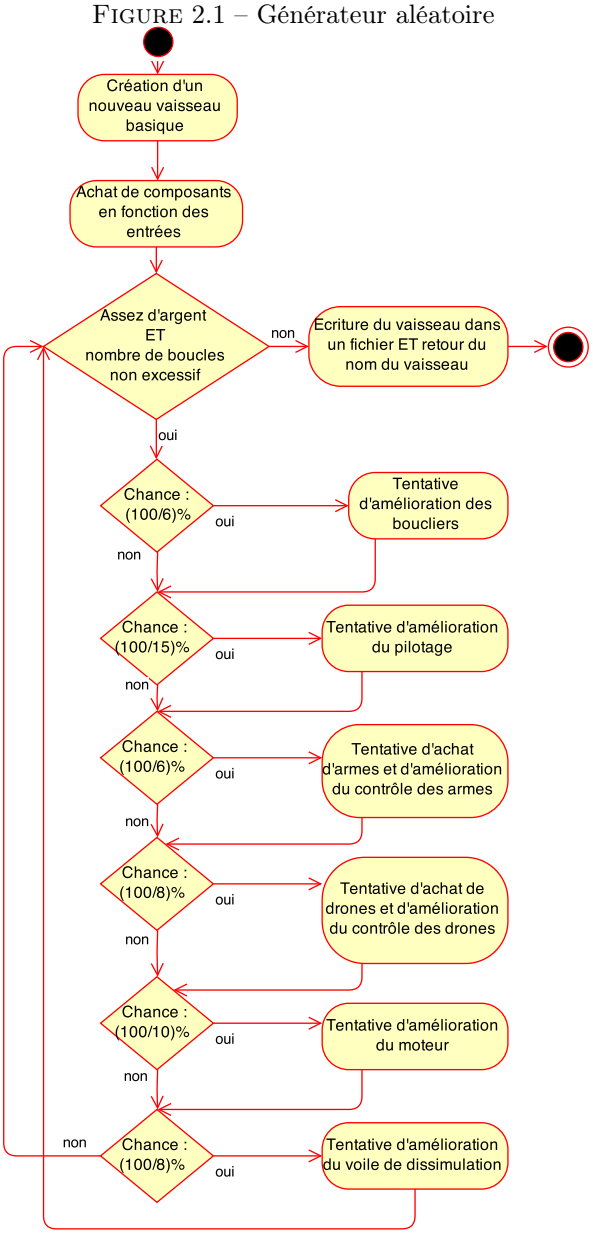
Un vaisseau est considéré comme bon lorsqu'il gagne plus de combats que ses concurrents. On voit donc une logique évolutive où l'on compare tout le temps les vaisseaux entre eux et où l'on peut s'inspirer des meilleurs pour essayer de créer d'encore meilleurs vaisseaux.
Pour cela, il nous a tout d'abord fallu un générateur de vaisseaux. À partir de ce générateur nous avons pu utiliser un algorithme génétique qui permet d'avoir cette logique évolutive. Pour compléter l'optimisation des meilleurs vaisseaux, nous avons utilisé un système expert qui puise ses connaissances dans une base de données où sont stockées les différentes caractéristiques d'un groupe de vaisseaux.

2.1 Algorithmes de génération de vaisseaux

Un générateur doit créer un vaisseau en fonction d'un certain coût car c'est ce critère qui sépare les vaisseaux en différentes catégories. Tout d'abord, un générateur aléatoire est nécessaire pour parcourir tout les combinaisons d'équipement possibles. Ensuite, on veut un algorithme plus intelligent qui puisse parcourir ces combinaisons aléatoires pour tirer les meilleures et éventuellement les affiner.

2.1.1 Générateur aléatoire

FIGURE 2.1 – Générateur aléatoire



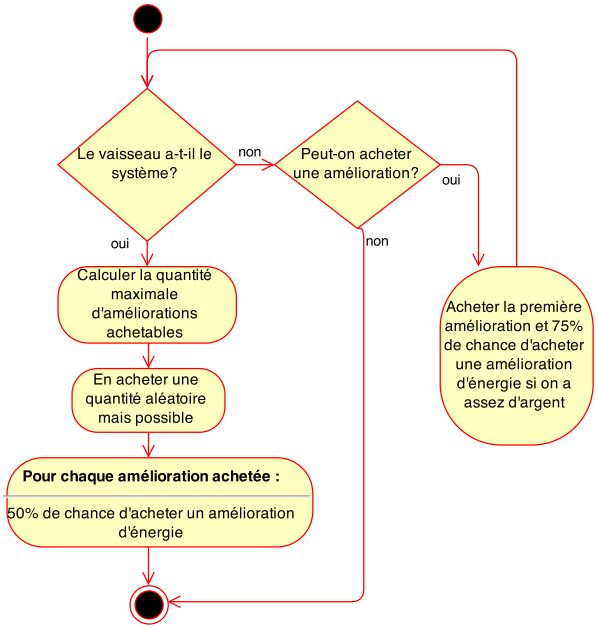
Le générateur aléatoire est simple puisque tant qu'il a de l'argent, il va essayer d'acheter quelque chose de façon aléatoire. Suivant les fonctionnalités gérées par le moteur de combat, nous avons construit le générateur aléatoire selon le schéma de la figure 2.1.

On remarque que l'aléatoire est relativement guidé car certains équipements sont plus ou moins utiles et donc il y a plus ou moins de chance de les acheter. Ce choix a été fait pour éviter de tomber sur le moins de vaisseaux inutiles.

Le fait d'avoir une boucle pour essayer chaque type d'équipement plutôt que de piocher à chaque fois un type d'équipement dans une liste est là pour éviter d'acheter des équipements différents pour créer des synergies. Dans les chances de choisir un type d'équipement il a fallu tenir compte des chances de tomber sur les équipements précédents.

Les détails d'un achat potentiel dépendent en partie de l'équipement choisi. De façon générale pour un système, le générateur va d'abord calculer combien d'améliorations il peut acheter, puis acheter un nombre aléatoire d'améliorations. Directement après, il va essayer d'acheter un nombre aléatoire (inférieur ou égal à l'achat précédent) d'énergie. Prenons l'exemple des moteurs.

FIGURE 2.2 – Exemple des moteurs dans le générateur aléatoire



Parmi les petits détails dépendant de l'équipement choisi, on peut citer :

— pour les boucliers le générateur va essayer d'acheter les améliorations par 2 mais avec une petite chance de n'acheter qu'une seule amélioration pour atteindre un palier pour pouvoir créer un *buffer* ;

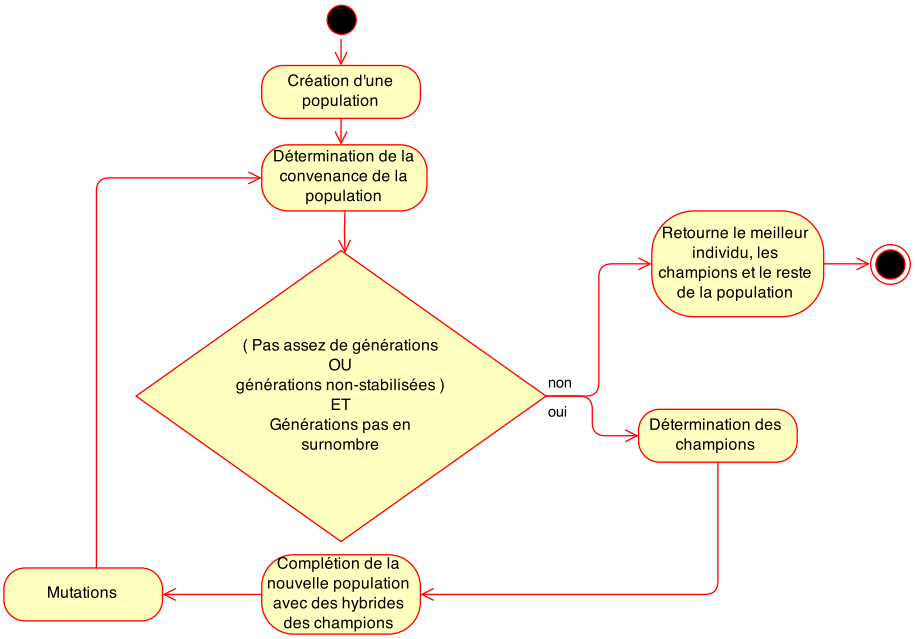
— pour le système des armes et celui des drones, le générateur choisit une arme (ou drone) aléatoirement, regarde s'il y a assez de place, regarde s'il y a assez d'argent, regarde le coût d'améliorations pour pouvoir alimenter toutes les armes et achète l'arme si toutes les conditions sont remplies, ensuite il achète un nombre aléatoire (plus petit ou égal à l'alimentation de l'arme choisie) d'énergie.

2.1.2 Algorithme génétique

Un algorithme génétique reproduit l'évolution naturelle. On part donc d'une population et on détermine les individus qui sont les meilleurs (ceux qui survivraient dans la nature). À partir de ces individus, on crée une nouvelle po-

pulation dont les individus peuvent avoir des mutations. On reproduit ainsi ce processus autant de fois que l'on veut.
Dans notre cas, les individus sont des vaisseaux et les gènes sont les achats effectués. L'intégration de l'algorithme ressemble donc à cela.

FIGURE 2.3 – Algorithme génétique



Pour déterminer les meilleurs vaisseaux, on fait simplement un tournoi et on garde ceux qui ont gagné le plus de combats. Un tournoi consiste à faire combattre chaque vaisseau contre chacun des autres vaisseaux (on construit la partie supérieure à la diagonale d'un tableau carré). Un combat de tournoi (un match) est fait de plusieurs combats du moteur de combats et le gagnant est celui qui a gagné plus de la moitié des combats.
Pour faire les croisements il faut deux vaisseaux issus des champions. Pour chacun de ces vaisseaux on garde en mémoire le nombre d'énergie acheté, les armes et drones achetés et les améliorations des systèmes. Ensuite, on crée un nouveau vaisseau de race et type identiques à un des deux vaisseaux pères. À partir de ce moment, on pioche aléatoirement dans les achats des parents pour les ajouter au nouveau vaisseau s'il a assez d'argent.
Une mutation est tout simplement le vaisseau résultant d'un croisement entre un vaisseau existant et un nouveau vaisseau aléatoire. Les vaisseaux existants pris ne sont pas les champions de la génération précédente pour garder ces derniers et ne pas baisser dans la qualité des champions.

L'algorithme s'arrête lorsqu'il y a eu trop de générations ou s'il y a une stabilisation chez les individus. Il y a stabilisation lorsqu'il n'y a plus de vaisseau vraiment plus fort que les autres, c'est-à-dire que tous les vaisseaux ont environ le même ratio de victoires.

Au final, nous pouvons faire une étude (simple) de complexité de temps dans le pire des cas. Les tournois prennent la quasi-totalité du temps et pour faire

les multiples combats des matchs, ils utilisent tous les processeurs disponibles. Il suffit donc de multiplier le nombre maximal de générations par le nombre de matchs dans un tournoi par le nombre de combats par match et par le temps moyen d'un combat et divisé par le nombre de processeurs de la machine. Si on prend g le nombre de générations, n le nombre de vaisseaux dans une population, c le nombre de combats par match, t le temps d'un combat et p le nombre de processeurs, on obtient la formule suivante.

$$\frac{gn(n-1)ct}{2p}$$

Un algorithme typique et rigoureux aura $g := 30$; $n := 20$; $c := 21$; $t := 2$ secondes et $p := 4$. L'algorithme durera donc au maximum un peu plus de 16 heures.

2.2 Data Mining

Pour les parties de data mining nous avons utilisé le programme *Linear time Closed itemset Miner* de Takeaki Uno qui nous permet d'extraire les caractéristiques (listes des équipements d'un vaisseau) les plus courantes dans une liste de vaisseaux.

Ce programme prend en entrée un fichier où chaque ligne contient des entiers séparés par des espaces, chacune de ces lignes représente un ensemble d'entiers. Le programme retourne ensuite tous les sous-ensembles de chaque ligne et donne le nombre d'occurrences de ces sous-ensembles dans toutes les lignes. La fouille fermée permet de ne garder que les ensembles qui ne sont pas des sous-ensembles d'autres lignes. On peut ensuite décider de ne garder que les sous-ensembles qui apparaissent dans au moins n lignes.

Le fichier que l'on donne au programme LCM a une ligne pour chaque vaisseau et chaque ligne est constituée des équipements achetés et présents de base (dans les équipements on compte les armes, drones, améliorations et l'énergie). Pour utiliser ce programme il nous a donc fallu traduire les caractéristiques des vaisseaux en entiers, ce que nous avons fait de la façon suivante :

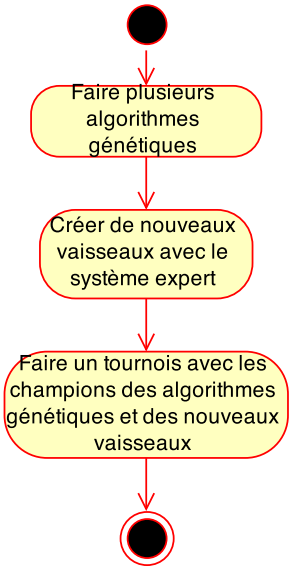
- pour l'énergie, on écrit simplement le nombre d'énergie disponible, donc un nombre compris entre 0 et 25 ;
- pour les améliorations des systèmes, on associe à chaque système un nombre plus grand que 3 et on concatène ce nombre avec le nombre d'améliorations faites (toujours moins que 9), le nombre final est compris entre 31 et 179 ;
- pour les armes, on associe à chaque arme un nombre et on concatène 10 avec ce nombre, le nombre final est compris entre 1001 et 1030 ;
- pour les drones, on associe à chaque drone un nombre et on concatène 11 avec ce nombre, le nombre final est compris entre 1101 et 1120.

À noter que dans le fichier d'entrée on écrit le niveau d'améliorations ou le niveau d'énergie avec les niveaux précédents pour avoir des sous-ensembles (e.g. pour un niveau de 3 d'énergie : energy1 energy2 energy3).

Au final, le fichier résultat est un ensemble de listes d'équipements communs à plus ou moins de vaisseaux.

2.3 Déroulement de l'algorithme final d'optimisation

FIGURE 2.4 – Algorithme final



L'algorithme final (figure 2.4) est l'algorithme utilisé pour trouver le meilleur vaisseau. Pour cela il utilise des algorithmes génétiques et du *data mining*. Au début il a plusieurs algorithmes génétiques distincts pour établir une base de vaisseaux avec des bons (les champions des algorithmes génétiques) et des mauvais vaisseaux. À partir de cette base de vaisseaux, on cherche les caractéristiques communes avec la technique de *data mining* vu précédemment. Ensuite pour chaque caractéristique commune, on calcule le nombre d'occurrences chez les mauvais vaisseaux et aussi chez les bons vaisseaux.

Grâce à ces données on peut calculer le taux de croissance de chaque caractéristique commune avec la formule

$$\rho(P) = \frac{\frac{S_+}{D_+}}{\frac{S_-}{D_-}}$$

où $S_\pm$ est le nombre d'occurrences chez les bons/mauvais vaisseaux et $D_\pm$ est le nombre de bons/mauvais vaisseaux. Si le nombre d'occurrences chez les mauvais vaisseaux est nul alors le taux vaut

$$\rho(P) = \frac{\frac{S_+}{D_+}}{\frac{1}{D_-^2}}$$

Ce taux est toujours positif et si ce taux est plus grand que 1, alors la caractéristique est relativement plus commune chez les bons vaisseaux, donc est peut-être une caractéristique déterminante pour faire d'un vaisseau un bon vaisseau. Pour être vraiment sur d'une caractéristique déterminante, nous utilisons aussi un taux de confiance qui a pour formule

$$\text{conf} = \frac{\rho(P)}{\rho(P) + 1}$$

Ce taux permet d'évaluer la grandeur du taux de croissance et donc évaluer la déterminance d'une caractéristique pour faire de bons vaisseaux.

On obtient donc les caractéristiques déterminantes. Pour chacune de ces caractéristiques, un vaisseau de chaque race et de chaque type va être créé avec le générateur aléatoire auquel on aura donné la caractéristique à acheter en premier. Ensuite on ne garde que les vaisseaux qui ne semblent pas mauvais. Un vaisseau valide ce test si sa race, son type et ses équipements ne font pas partie des *low tiers* de *tiers lists* trouvées sur internet. À partir de cette liste de nouveaux vaisseaux et des vaisseaux gagnants des algorithmes génétiques, un tournoi est fait pour déterminer le meilleur d'entre eux et c'est celui-ci qui est retourné par l'algorithme au final.

Le temps de cet algorithme est équivalent au temps mis par les algorithmes génétiques du début car le programme *LCM* de data mining est très rapide.

Chapitre 3

Réalisation

3.1 Installation et configuration de l'application

Notre projet est codé en Python 3. Des fichiers XML sont utilisés pour le stockage des vaisseaux. Pour la partie système expert il nécessaire d'avoir un shell (par exemple cygwin sur Windows) pour que le programme puisse appeler des commandes UNIX externes comme *make*.

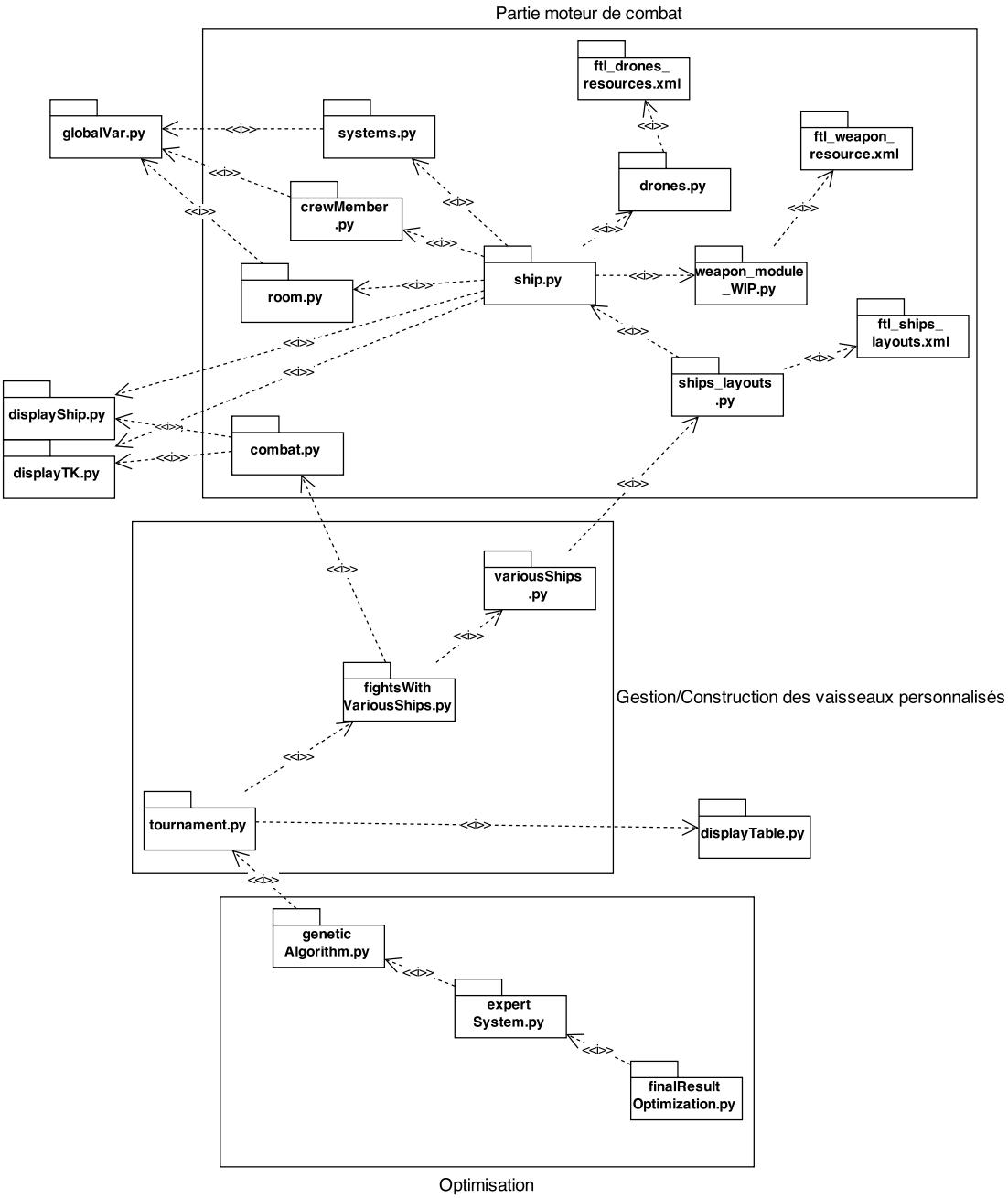
3.2 Organisation et répartition des tâches

Au premier semestre, nous avions dégagé 4 domaines de travail : le programme du moteur de combat, la construction d'une base de données issues du jeu original, les tests et enfin le rapport.
Le moteur de combat a été réalisé en partie par Simon BIHEL avec la contribution de Julien PEZANT pour la classe des armes et l'aide de Sébastien GAMBLIN pour une partie de l'affichage.
La construction de la base de données a été faite en collaboration par Julien PEZANT, Paul LEMENAGER et Simon BIHEL.
Les tests ont été faits en partie par Josselin GUENERON, Sébastien GAMBLIN et en partie par Simon BIHEL.
Enfin le rapport a été construit par Simon BIHEL et Sébastien GAMBLIN avec la contribution de chacun des autres membres par rapport à leurs connaissances du projet.

Au second semestre, nous avons dégagé 4 domaines de travail : l'enrichissement et l'amélioration du moteur de combats ; l'optimisation et la recherche des meilleurs vaisseaux ; des affichages divers pour les vaisseaux et les tournois ; la rédaction du rapport.
La travail sur le moteur de combats et sur la recherche des meilleurs vaisseaux a été effectué par Simon BIHEL.
Les affichages ont été réalisés par Sébastien GAMBLIN.
Le rapport a été dirigé par Simon BIHEL avec l'apport de chacun des membres. Notamment, Julien PEZANT a participé à la refonte et traduction des schémas du rapport du premier semestre et Sébastien GAMBLIN a rédigé la partie des affichages.

3.3 Importation des modules

FIGURE 3.1 – Diagramme des packages



Comme on peut le voir dans la figure 3.1, notre programme est divisé en 3 grandes parties :

— une partie qui gère les combats entre vaisseaux (réalisé au semestre 1) ;

— une partie qui gère les vaisseaux personnalisés ;
— une partie qui utilise les 2 parties précédentes pour déterminer le meilleur vaisseau.

En plus de cela, il y a aussi diverses fonctions d'affichage (graphiques ou textuelles) ainsi qu'un fichier qui contient des variables globales qui déterminent s'il doit y avoir certains affichages, la division du temps pour le moteur de combats, etc.

3.4 Détermination de divers paramètres

Dans les algorithmes utilisés il a certaines variables arbitraires qui permettent de faire des tests ou encore de faire un certain nombre de boucles. Pour déterminer au mieux ces variables il a fallu avoir certaines réflexions et effectuer certaines expérimentations.

Les tournois Dans un tournoi, les vaisseaux font des matchs entre eux et le gagnant d'un match est celui qui aura gagné plus de combats que son adversaire. Pour être assuré que ce soit presque toujours le meilleur vaisseau qui gagne le match il a fallu trouver un bon nombre de combats pour avoir des résultats corrects et un temps raisonnable pour le tournoi complet. À noter aussi que nous avons choisi un nombre impair pour éviter des matchs nuls (sachant qu'un combat a toujours un gagnant). Nous avons choisi 21.

Les algorithmes génétiques Ici, le paramètre qui peut rendre un algorithme génétique obsolète est le nombre maximal de générations. Celui-ci est là pour éviter qu'un *metagame* s'installe et empêche l'algorithme de finir. Un *metagame* est une mode dans les stratégies de jeu. La limite de générations doit donc donner assez de temps aux vaisseaux de se perfectionner sans gaspiller du temps sur des changements de mode. Nous avons choisi d'avoir 30 générations. Ensuite, il y a la stabilisation qui dépend du ratio de victoires maximal pour déterminer si un vaisseau est plus fort que les autres ou si tous les vaisseaux sont à peu près équivalents en terme de nombre de victoires. Nous avons choisi de dire qu'un vaisseau est plus fort que les autres s'il gagne plus des deux tiers de ses matchs.

L'algorithme final Plusieurs paramètres ont dû être déterminés dans l'algorithme final de recherche du meilleur vaisseau.
Le taux de confiance minimum pour le taux de croissance est de ⅓ car cela veut dire que le taux de croissance est d'au plus ½. C'est-à-dire que si 1% des bons vaisseaux ont une caractéristique, alors au maximum 2% des mauvais l'auront. Étant donné la grande différence entre le nombre de mauvais vaisseaux et de bons, on aura les caractéristiques vraiment propres aux bons vaisseaux, cependant il faut avoir bon équilibre dans la proportion du nombre de champions retenus dans une population au cours d'un algorithme génétique. Une taille de population 5 fois supérieure au nombre de champions d'une génération semble marcher au final (pas de réelle réflexion scientifique ici, juste une constatation). Le nombre d'algorithmes génétiques au début doit être plus grand que 1 pour ne pas avoir un seul modèle de vaisseau et donc le meilleur vaisseau au final sera identique ou très proche du vainqueur de l'algorithme génétique. Il n'y a pas de maximum car cela enrichira simplement la diversité des vaisseaux et permettra

d'essayer plus de bonnes combinaisons d'équipement. À noter que ce sont les algorithmes génétiques qui prennent la majorité du temps, et le temps croît de façon exponentielle en fonction du nombre de combats par match, du nombre de vaisseaux dans une population, etc.

3.5 Divers résultats d'optimisation

La visualisation de données peut être un exercice compliqué mais nous pouvons regarder l'évolution de l'équipement des vaisseaux au cours d'un algorithme d'optimisation.

Regardons un exemple d'algorithme génétique avec stabilisation avec une population de 20 individus et 2 champions et un coût de 600 scraps. La population de base est composée (aléatoirement) d'engi cruisers de tous types. Une première étape peut être de faire construire une base de données pour chaque génération avec le programme de data mining LCM. Rappelons les caractéristiques principales de chaque type d'engi cruiser. Le type A a l'arme Ion Blast II et le drone Combat I. Le type B a les armes Heavy Ion et Heavy Laser I. Le type C a l'arme Dual Laser et le drone Beam I.

Pour la population de base, étant donnée la construction aléatoire des vaisseaux, les équipements communs sont ceux de base comme l'énergie, le système des armes ou les boucliers. Les champions sont deux engi cruisers C, l'engi0 et l'engi1. Ces deux vaisseaux ont les augmentations suivantes. L'engi0 a 3 d'énergie, 5 dans le système des armes, 1 dans les moteurs, 2 dans le système des drones, les armes antibioBeam et glaiveBeam et le drone beam1. L'engi1 a 2 d'énergie, 6 dans les boucliers, 1 dans l'invisibilité, 1 dans les moteurs et le drone combat1.

La seconde génération est composée des deux champions précédents et de vaisseaux issus du croisement des deux champions, deux de ces nouveaux vaisseaux ont subi une mutation. En plus des équipements communs de la première génération, on voit que trois quarts des vaisseaux ont l'antiobioBeam. Les champions sont toujours l'engi0 et l'engi1.

Après plusieurs générations on arrive à la dernière, l'engi0 et l'engi1 sont toujours les champions. Parmi les derniers vaisseaux, l'antibioBeam est toujours dans les trois quarts des vaisseaux.

Au final, les croisements n'ont pas créé de meilleurs vaisseaux, sans doute car les équipements des deux champions n'étaient pas compatibles et les mutations n'ont pas réussi non plus à créer de bons vaisseaux.

3.6 Failles et inexactitude des résultats

Notre programme présente quelques failles et n'est pas complet par rapport au jeu originel ce qui rend les résultats donnés pas forcément applicables dans de vraies parties. Voici quelques points à prendre en compte pour la validité des résultats donnés par le programme.

Le moteur de combats Tous les équipements du jeu ne sont pas gérés par le moteur de combats. À cause de cela, les vaisseaux issus des algorithmes de

génération et d'optimisation n'auront pas certains équipements qui sont potentiellement bons. De plus, L'Intelligence Artificielle qui dirige les vaisseaux fait les choses bêtement et n'utilise pas forcément des techniques de jeu. Cela peut rendre des vaisseaux obsolètes alors qu'ils sont bons quand ils sont bien utilisés.

La génération aléatoire de vaisseaux Le fait de pouvoir vendre de l'équipement n'est pas présent dans le générateur aléatoire, donc ce n'est pas présent dans les algorithmes d'optimisation. À cause de cela, il a encore beaucoup de vaisseaux potentiellement bons qui ne sont pas utilisés.

L'algorithme génétique L'algorithme génétique peut s'enfermer dans un *matchup*. C'est-à-dire qu'il peut créer des vaisseaux performants dans un seul type de combat et mauvais contre la plupart des vaisseaux qui ne sont pas présents dans la population de cet algorithme génétique. Au fil du temps et malgré le nombre maximal de générations, un *metagame* peut s'installer et dans ce cas il est difficile de différencier une évolution utile d'un effet de mode.

L'algorithme final Dans l'algorithme final, la base de vaisseaux est constituée à partir d'algorithmes génétiques, donc les champions auront certaines caractéristiques en commun avec les mauvais, cela peut faire baisser le taux de croissance et le taux de confiance. Cela est amplifié par le fait que les algorithmes génétiques s'arrêtent par stabilisation car dans certains cas, les individus de la dernière génération seront similaires.

Conclusion

À la fin de ce second semestre, les tâches prévues ont été réalisées. Le moteur de combats fonctionne bien même s'il n'est pas aussi complet que le jeu originel. Les représentations graphiques permettent de visualiser des combats et des tournois aisément pour des humains. La recherche du meilleur vaisseau fonctionne et semble donner des résultats cohérents par rapport aux éléments gérés par le moteur de combats.

Cependant, certaines parties du travail réalisé peuvent être améliorées et des extensions peuvent s'ajouter au programme présent.
Le déplacement des membres d'équipage a été mis en pause car il prenait du temps et ne fonctionnait pas très bien. Avec l'affichage graphique, il sera plus facile de débuguer le programme qui est déjà bien avancé avec la construction d'un graphe pour représenter les salles et la recherche de plus court chemin d'un graphe.
Un des gros points noirs de l'optimisation est la durée des tournois. Une première solution serait d'arrêter un match lorsqu'un vaisseau a déjà gagné plus de la moitié des combats. Actuellement, chaque combat est un processus et on crée une *pool* (un groupe) de plusieurs combats pour pouvoir faire du parallélisme, il faudrait donc faire une fonction qui divise ce pool des combats pour évaluer à intervalles réguliers le nombre de victoires. Cet intervalle peut être le nombre de combats restants à gagner pour le vaisseau qui a le plus de combats gagnés.