FTL : Fit the fittest
Rapport du TPA

Sébastien Gamblin Simon Bihel Julien Pezant
Paul Leménager Josselin Gueneron

Table des matières

Chapitre 1

Introduction

1.1 Présentation du jeu

FTL : Faster than Light est un jeu pour PC de science fiction, 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, 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 possédant chacune des avantages et des inconvénients.

1.2 Présentation du projet

Le but du projet est de concevoir un programme qui calcule quel est le meilleur vaisseau du jeu. Pour cela, l'utilisateur devra indiquer combien de scraps (monnaie du jeu) il veut investir dans l'optimisation de son vaisseau, et quelle est la catégorie du vaisseau qu'il joue parmi les vingt-sept classes possibles.
Le projet doit alors se baser sur de nombreuses statistiques intrinsèques au jeu afin de modéliser un duel de vaisseau dans FTL. En effet, de nombreux mécanismes de jeu sont à reproduire de par les statistiques présentes, ainsi que les diverses actions ou armes. Pour cela, nous allons nous baser sur le site http://ftl.wikia.com/wiki/FTL:_Faster_Than_Light_Wiki qui regroupe les nombreuses informations qui nous seront nécessaires.
En outre, notre projet est réalisé en Python version 3.4 et ne comporte pas de modules supplémentaires à la librairie Python.

1.3 Objectifs

Le premier semestre Dans un premier temps, le projet consiste à modéliser un seul et unique vaisseau, ce qui est en soi la partie la plus complexe à réaliser. En effet, les duels de vaisseaux sont composés de nombreuses composantes difficiles à modéliser : gestion de nombreuses armes, déplacement de l'équipage, niveaux de chaque membre d'équipage qui influe sur le vaisseau, différents systèmes présents dans un vaisseau, de nombreux bonus, etc. Un vaisseau est composé de salles, dans lesquelles peuvent être présents des systèmes, et des membres d'équipages. Selon les niveaux des systèmes et des membres d'équipage, les bonus d'attaque et d'esquive varient. C'est pourquoi beaucoup de paramètres sont à prendre en compte.Toutes ces données sont donc à comprendre et à appliquer au module de combat, ce qui représente un grand travail d'extraction et de compréhension du fonctionnement du jeu.
Au début, le vaisseau sera basique, et ses systèmes de combats simplifiés. Ainsi des combats pourront être modélisés. Au fur et à mesure du premier semestre, nous allons améliorer ce système de combat, et étoffer le fonctionnement du vaisseau. Ainsi, à la fin du premier semestre, nous espérons avoir un moteur de combat qui prenne en compte toutes les statitiques présentes dans le jeu, ainsi que des intelligences artificielles qui permettent l'utilisation de l'équipage et des choix "tactiques" du vaisseau.

Le second semestre Une fois le moteur de combat accompli et complet, nous pourrons générer (aléatoirement ou non) des vaisseaux qui seront munis d'armes, d'équipements spécifiques, en fonction du nombre de scraps disponibles. Il va nous falloir optimiser la constitution de ses vaisseaux grâce à des règles simples ou non pour qu'il soit le plus performant possible. L'utilisation de systèmes experts pourra être envisagée.
Ces vaisseaux devront ensuite, grâce à un module de combat, générer de nombreux duels. Ainsi, sur ces nombreux essais, nous pourons évaluer quel est le meilleur vaisseau entre tous. L'étape pourra être reproduite de nombreuses fois pour optimiser les informations recueillies. Enfin, nous pourrons utiliser des algorithmes génétiques afin de générer le meilleur des vaisseaux.

Chapitre 2

Concepts à maîtriser

Le but du premier semestre consistant, en fait, simplement à reproduire le jeu original, il y a juste quelques points à aborder.
D'abord, nous avons dû extraire les informations du jeu, par exemple celles des vaisseaux de base. Pour cela, nous avons décidé d'utiliser des fichiers xml. Concernant la représentation des vaisseaux et des éléments qui les composent, nous avons utilisé la Programmation Orientée Objet.
Pour s'assurer d'avoir une représentation correcte du jeu original, sans rien avoir oublier, nous avons construit une bible que voici.

2.1 La bible

2.1.1 Base de données des infos de base tirées du jeu

☑Travail accompli

- XML pour les armes
  - tous les missiles sauf swarm
  - tous les rayons
  - tous les lasers de l'édition de base
  - tous les ions
- XML pour les vaisseaux
  - Kestrel A et B
  - Engi A

Travail à terminer

- XML pour les armes restantes
- XML pour les vaisseaux restants

2.1.2 Cooldowns et dots (damages over time)

☑Travail accompli

- Recharge des boucliers
- Ions

- Réparation par les membres d'équipage
- Dommages aux systèmes dûs aux feux
- Explosion des systèmes :
  - Déclencheur de feu
  - Dommages aux membres d'équipage présents
- Expansion du feu aléatoire
- Extinction des feux par les membres d'équipage
- Réparation des brèches et des systèmes par les membres d'équipage
- Gestion de l'oxygène
  - Baisse du taux d'oxygène due aux brèches, feux, membres d'équipages de la race Lanius, système non-alimenté
  - Expansion de l'oxygène (entre salles et portes ouvertes vers l'espace)
  - Extinction des feux par manque d'oxygène

Travail à terminer

- Baisse du taux d'oxygène quand le système de l'oxygène est hacké
- Expansion du feu plus proche du jeu originel

2.1.3 Gestion de l'équipage

☑ Travail accompli

- Avantages aux systèmes (lorsque le personnage n'est pas en mouvement)
  - Rechargement plus rapide des boucliers et des armes
  - Augmentation de l'esquive grâce aux réacteurs et au pilotage
- Gain d'expérience pour chaque "coup" final de la réparation d'un système et pour chaque tir bloqué, esquivé ou effectué (cela dépend bien sûr du système présent dans la salle)
- Mort d'un membre d'équipage quand il n'a plus de vie, résurection si le vaisseau a la clone bay
- Dégâts par manque d'air
- Remise à 0 des réparations des systèmes lorsqu'il n'y a pas de membre d'équipage dans leurs salles

Travail à terminer

- Gestion des déplacements (travail en pause)

2.1.4 Utilisation des armes et drones

☑ Travail accompli

- Double Dégâts au hull pour les salles sans systèmes par certaines armes
- Déclencheur de feux et de brèches
- Dégâts aux membres d'équipage présents dans la salle par certaines armes

Travail à terminer

- Utilisation des armes ions
  - Double "dommages" aux boucliers zoltan
  - L'ionisation et les nouveaux vrais dommages ne s'accumulent pas
  - Un système ionisé ne peut pas recevoir de bonus grâce aux membres d'équipage

2.1.5 Gestion de l'énergie

☑Travail accompli

- Impossible d'ajouter de l'énergie si le système est ionisé
- Lorsqu'un système n'est plus ionisé, redonne automatiquement au système l'énergie qui a été enlevée
- Désactivation des armes/drones lorsque le système des armes/drones prend des dégâts
- Re-alimente automatiquement les armes/drones qui ont dû être désactivés (de façon rudimentaire)
- Les vaisseaux essayent de dépenser leur énergie

2.1.6 Gestion particulière de certains systèmes

☑Travail accompli

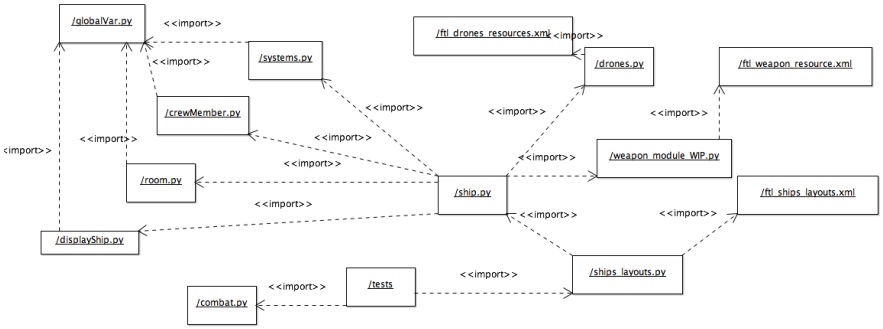
- L'invisibilité
  - Activation lors de la gestion des cooldowns
  - Remet à 0 le rechargement des armes et drones du vaisseau adverse
  - Ionisation 5 fois de suite du système après utilisation
- La résurection
  - Création d'une copie du membre d'équipage qui est supprimé lorsque la copie est faite
  - Réduction des niveaux de compétence
  - Après un certain temps, ajout de l'objet dans la liste des membres d'équipage

Chapitre 3

Organisation générale de l'application

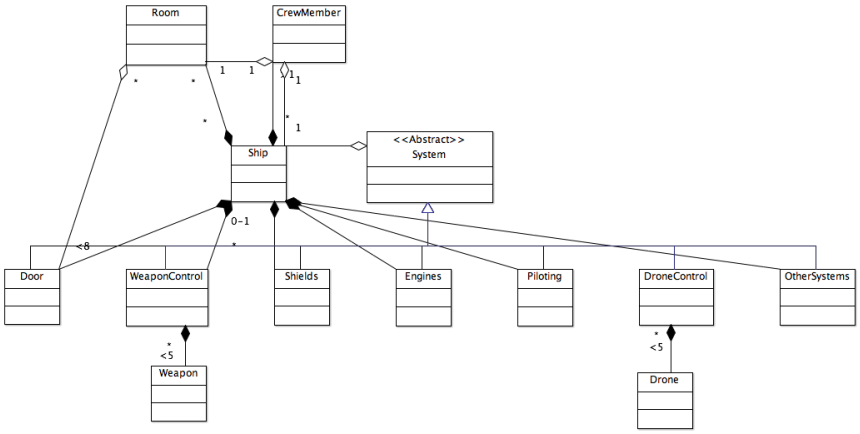
3.1 Importation des modules

Notre programme consistant principalement en des instances de classes qui interragissent entre elles, nous avons décidé de séparer notre code avec un fichier par classe.



Comme nous pouvons le voir sur le graphique, le programme est lancé par les modules de tests (test-combats et tournament-test) qui importent combat qui est simplement la fonction qui lance un combat. De plus les tests ont besoin de générer un vaisseau, et c'est là que l'arboresence se complique car le vaisseau est contitué de nombreuses classes (une classe est représenté par un module), et a besoin de plusieurs fichiers xml.

3.2 Diagramme des classes et représentation des vaisseaux



Toutes les classes ont un rapport direct avec la représentation d'un vaisseau. Le diagramme montre donc la représentation d'un vaisseau qui se décompose ainsi :

— Un vaisseau contient des systèmes, des salles et des membres d'équipage.
— Les drones et armes appartiennent à leurs systèmes respectifs.
— Les systèmes et membres d'équipage ont aussi le vaisseau en attribut car ils agissent souvent sur le vaisseau.
— Les membres d'équipage possèdent la salle dans laquelle ils sont.
— Les salles possèdent des portes.

3.3 Stockage des données

Comme dit précédement, les données issues du jeu original sont mises dans des fichiers xml. Ces données concernent les vaisseaux de base, les armes et drones.

L'importation des données inscrites dans les fichiers xml sont extraites grâce au module python xml.etree.ElementTree

Pour les vaisseaux, nous avons stocké les données suivantes :
— Les systèmes présents et leurs énergies de base.
— Les armes et les drones.
— Les salles avec : leurs coordonnées, le système qui est dedans ou celui qui peut être dedans, les portes, les membres d'équipage.

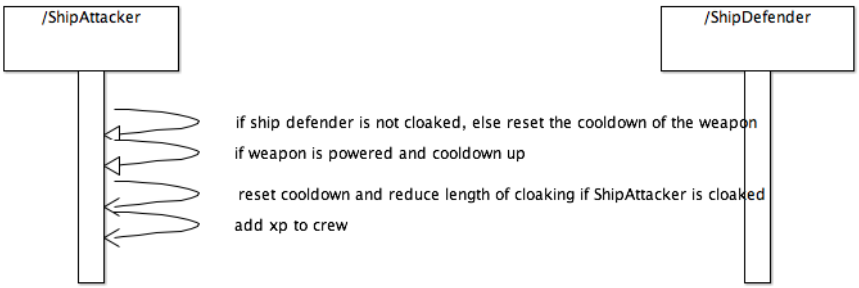
Pour les armes et drones, nous avons stocké les temps de rechargement et d'autres informations qui dépendent de chaque objet.

3.4 Le déroulement d'un combat

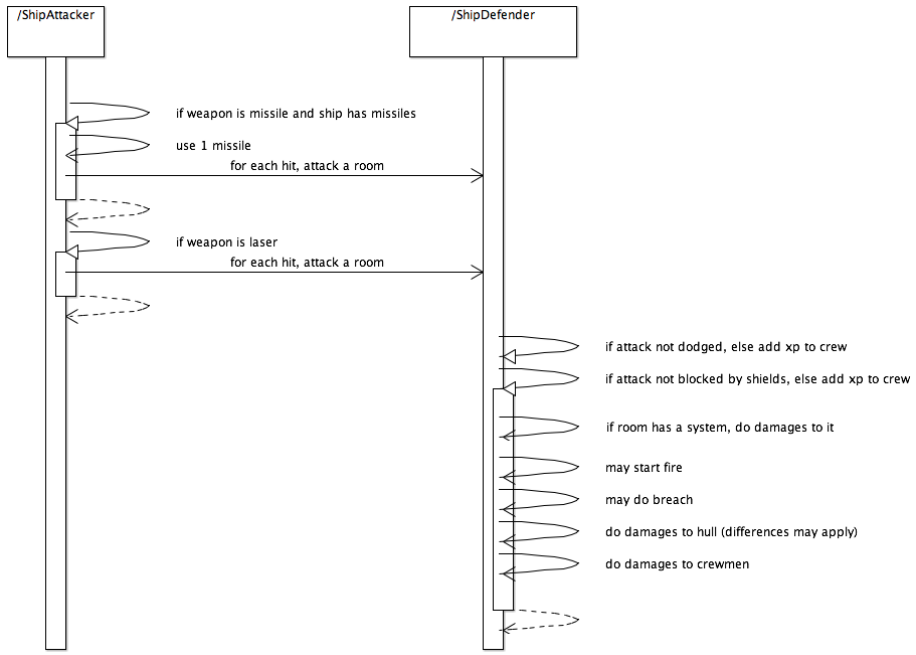
Nous avons décidé de gérer un combat avec un fonctionnement au tour-par-tour tout en gardant un faible temps entre chaque tour. Cela nous permet de gérer les phases d'attaques et de cooldowns indépendamment, tout en restant proche du jeu original qui est en temps réel.

3.4.1 Le déroulement d'une attaque

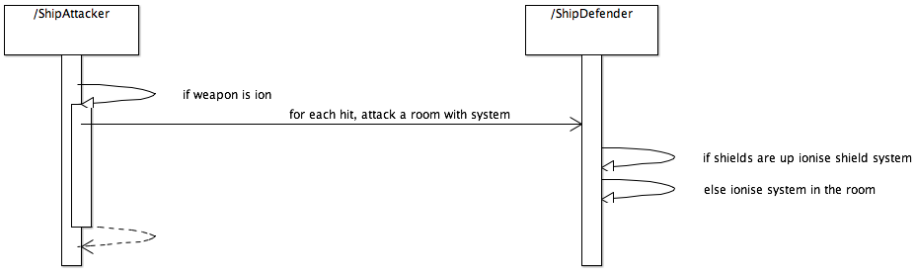
L'utilisation d'une arme :
Pour chaque arme, la même procédure va être appliquée. Celle-ci est gérée par le système des armes.



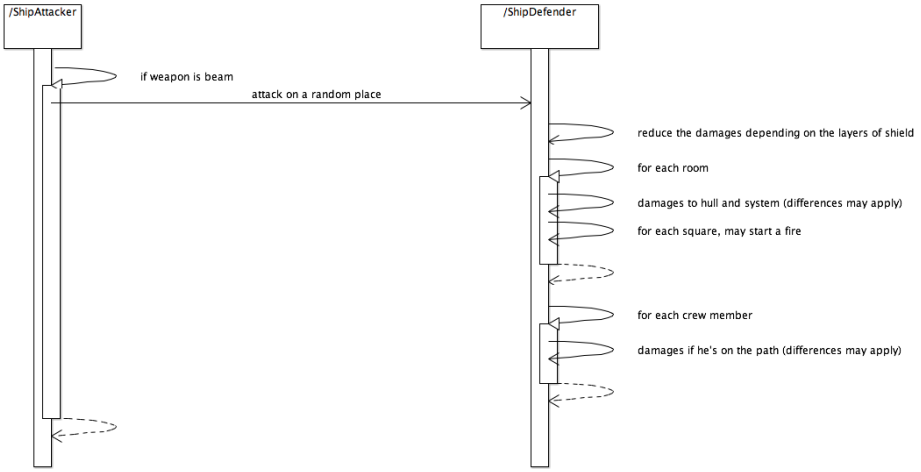
Le système fait d'abord tous les préparatifs avant d'être sûr qu'il a le droit d'utiliser l'arme. Ensuite si toutes les conditions sont remplies, il l'utilise. Les différents types d'armes ont des fonctionnements différents.



Les armes lasers et missiles ont des fonctionnements similaire, il faut seulement s'assurer de la bonne gestion des missiles. 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 dégats sont appliqués aux systèmes, aux membres d'équipage présents dans la salle, à la coque (hull) et des départs de feu ou des brèches peuvent être générés.



Les armes ions font des dégats temporaires aux systèmes et seulement à eux. 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.



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

3.4.2 La gestion des cooldowns

Le vaisseau : L'ordre des différentes instruction 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 differents systèmes, armes et drones.
3. Annuler les réparations efféctuées d'un système s'il n'y a plus personne dans la salle.
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.

Chapitre 4

Réalisation

4.1 Installation et configuration de l'application

Notre projet est codé en Python 3 Pour tester notre programme il faut donc utiliser un terminal python correspondant.
A ce stade, notre programme est juste composé d'un moteur de combat, pour le voir fonctionner il faut donc lancer un test parmis ceux disponibles dans les fichiers tests_combats.py et tournament_test.py.

4.2 Tests et expérimentations réalisées

Le but des tests est de voir si les combats se déroulent correctement. Les test à faire combattre des vaisseaux entre eux dont certains sont plus ou moins avantagés par rapport à leurs adversaires.
Exécuter de nombreux combats entre les mêmes vaisseaux de départ nous permet alors de voir à la fin le taux de victoire de chaque vaisseau. Il nous restera alors à vérifier les résultats sont cohérents avec les issues attendues.

De plus nous pouvons suivre les comportements des vaisseaux : qui tire, avec quelle arme, sur quelle salle, quels sont les répercutions. C'est le module *displayShip.py* qui nous permet de visualiser toutes ces actions en affichant les états des vaisseaux dans le terminal.

4.3 Gestion du projet : organisation et répartition des tâches

Nous avons 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 de notre projet.

Chapitre 5

Conclusion

5.1 Le travail effectué

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 permettra de finaliser notre projet : optimiser la recherche d'un vaisseau meilleur que les autres.

5.2 L'avenir du projet

Notre programme, même s'il fonctionne, pourra être amélioré. En effet, vérifier à chaque tour les différents cooldowns prend du temps. On pourrait donc réfléchir à faire un système de liste d'évènements.

Cependant, mise à part l'envie de rendre le moteur de combat encore plus complet, au second semestre il faudra arriver à un programme qui pourra déterminer le meilleur vaisseau de chaque catégorie.
Pour cela il faudra d'abord dévolopper un générateur de vaisseaux qui rende la création de vaisseaux aisée et automatique.
Ensuite il faudra appliquer des algorithmes d'optimisation qui nous permettront d'arriver à notre but. Ainsi nous pourrons envisager plusieurs pistes :

- Etablir arbitrairement des règles d'optimisation pour un vaisseau
- Utiliser des systèmes experts afin de générer une intelligence artificielle qui pourra préférer tel équipement dans tel cas, optimiser l'utilisation des énergies des vaisseaux, ne pas avoir telle arme si elle est inutilisable, etc
- Utiliser un système d'algorithme génétique qui nous permettra de conserver les vaisseaux gagnants, et de les combiner pour qu'ils soient plus performants.