

ENSEIRB-MATMECA

PROJET DU SEMESTRE 5
INFORMATIQUE

Planetwars

Auteurs :

Jean-Baptiste ZACCHELLO
Florian LE VERN
Amine CHEKNOUN

Responsable :

M. Laurent SIMON

Decembre 2014

Sommaire

Introduction	1
1 Conception du projet	2
1.1 Traitement des données reçues depuis le serveur de jeu.	2
1.2 Stratégie de l'intelligence artificielle.	2
1.2.1 Recherche de la meilleure stratégie	2
1.2.2 Phase d'expansion	3
1.2.3 Phase de défense	5
1.2.4 Phase d'attaque	6
1.2.5 Projet avorté : IAdéfensive	7
2 Réalisation du projet	8
2.1 Communication avec le serveur du jeu	8
2.2 Développement de l'intelligence artificielle	9
2.2.1 Étapes de construction	9
2.2.2 Résultats	11
2.2.3 Critique sur les choix adoptés et améliorations	12
Conclusion	14

Introduction

Le projet Planetwars organisé par Google avait pour but de faire s'affronter les intelligences artificielles du monde entier afin d'obtenir la plus performante. Dans le cadre de notre projet, nous nous mettons dans des circonstances similaires, mais cette fois-ci, nous nous limiterons à l'ensemble des groupes de projet pour comparer les IA.

Techniquement, les intelligences artificielles reçoivent les données sur l'entrée standard, et envoient leurs ordres sur la sortie standard. Chacune des IA a au plus 1 seconde pour décider de ce qu'elle doit faire, c'est là que se trouve la réelle difficulté pour la conception de l'IA. En effet, celle-ci doit être capable de traiter toutes les données du jeu qui lui sont nécessaires pour ensuite décider qu'elle stratégie adopter face à l'ennemie.

L'objectif est, d'une part, l'application et la maîtrise des connaissances en programmation C pour mener à bien un projet ainsi que l'apprentissage du travail en équipe et d'autre part, la mise en œuvre de solutions pour traiter un problème bien défini.

Nous verrons tout au long du rapport, les choix que nous avons adoptés lors de la réalisation de ce projet, les difficultés auxquelles nous nous sommes heurtés ainsi que leurs résolutions.

1 Conception du projet

Dans le projet, nous avons distingué deux étapes :

- Comment sauvegarder les données reçues depuis le serveur pour faciliter ensuite leurs utilisations ?
- Quelles stratégies faut-il adopter pour avoir les meilleures performances possibles.

1.1 Traitement des données reçues depuis le serveur de jeu.

Une partie que l'on peut considérer comme essentielle dans ce projet est, bien sûr, la récupération des données envoyées par le serveur. Afin de faciliter la programmation de l'intelligence artificielle, celle-ci doit être optimale. Nous avons donc choisi d'utiliser l'allocation dynamique du langage C afin de sauvegarder dans différentes structures ces données. Ainsi, à chaque tour du jeu nous aurons une structure contenant d'autres structures possédant toutes les données nécessaire qu'une IA a besoin pour fonctionner.

struct planet : contient toutes les informations sur une planète (propriétaire, taille, position, nombre de vaisseaux, id) ;

struct fleet : contient toutes les informations sur une flotte (propriétaire, nombre de vaisseaux, planète source, planète de destination, nombre de tours pour le parcours, nombre de tours avant l'arrivée) ;

struct map_t : contient un tableau répertoriant les planètes, un tableau répertoriant les flottes, et la taille de ces tableaux.

La fonction **get_map()** permet de récupérer les données du serveur en construisant une structure "struct map_t". Les tableaux qu'elle contient sont alloués dynamiquement étant donné qu'on ne connaît pas à l'avance leur taille. Pour éviter d'utiliser trop de ressources mémoire, la fonction **remove_data(struct map_t)** libère l'espace alloué dans **get_map()** à la fin de chaque traitement des données par notre IA.

Il aurait été possible d'améliorer la complexité de cette prise de données en ne considérant que les changements par rapport au tour précédent au lieu de recalculer puis effacer les données envoyées par le serveur à chaque tour.

Ensuite, des fonctions nécessaires pour l'implémentation de n'importe quel Bot ont été écrites sur la base du raisonnement suivant :

- un Bot aura souvent besoin de sélectionner ses propres planètes ou celles des adversaires ou celles neutres.
- il aura besoin de savoir la distance entre deux planètes en terme de temps en tours pour envoyer des flottes.
- il aura besoin de connaître le nombre total de vaisseaux à sa disposition et à la disposition de l'adversaire (pour décider d'adopter une stratégie particulière en fonction du joueur qui semble gagner par exemple).

1.2 Stratégie de l'intelligence artificielle.

1.2.1 Recherche de la meilleure stratégie

Le deuxième sous problème mentionné précédemment est celui de la stratégie de notre intelligence artificielle. Dans ce sujet, les possibilités offertes pour la stratégie sont presque illimitées.

Seule la contrainte de temps oblige chaque joueur à répondre au serveur en moins d'une seconde. Au début du projet, nous avons testé différentes stratégies avant d'obtenir celle que nous pensons la plus performante.

En effet, nous étions partis sur une IA qui devait envoyer des flottes sous forme de chaîne entre ses planètes vers les planètes les plus proches de l'ennemie et ainsi maximiser le nombre de flotte disponible pour attaquer. Nous nous sommes ensuite vite aperçus qu'il serait alors assez difficile de gérer la défense avec ce genre de stratégie, car elle est facilement inefficace contre une IA aggressive telle que le RageBot qui nous a été fourni. Cette IA a donc été rejetée.

Une autre stratégie consistait à capturer les planètes neutres en fonction d'un calcul de barycentre sur les planètes avec leurs taux de croissance comme pondération ; cela aurait permis de conserver un noyau fort facile à défendre. Mais là encore, cette stratégie ne faisait clairement pas le poids face à de vraies stratégies, la capture étant trop inefficace.

Au final, nous avons opté pour une IA qui classe les planètes grâce à un taux de rentabilité. Cette IA se divise en plusieurs phases, une phase d'expansion et de défense ainsi qu'une phase d'attaque. On distingue clairement ces deux phases car elles ont un comportement totalement différent.

Dans ce qui suit, nous allons voir plus en détails le fonctionnement de notre IA et des phases qui la constituent.

1.2.2 Phase d'expansion

Dans l'explication qui suit, nous faisons référence à la distance temporelle qui sépare deux planètes, ce que l'on entend par distance temporelle, c'est le nombre de tours que met une flotte pour aller d'une planète à une autre. Tout au long du programme, toutes les fonctions utilisent des distances temporelles car ce sont les seules qui apportent des informations que l'on peut traiter directement.

Pour la phase d'expansion, il était clair qu'il fallait trouver un moyen d'affecter à chaque planète un coefficient qui permettrait ainsi de les distinguer et donc de les classer par ordre de préférence. On a donc défini une rentabilité pour chacune des planètes grâce à deux formules, car en effet on ne calcule pas de la même manière la rentabilité d'une planète neutre et celle d'une planète ennemie. Chacune de nos planètes va affecter une rentabilité à toutes les autres planètes de la carte en calculant le gain fourni par sa capture sur une durée arbitraire que l'on a fixé à 20 tours. La rentabilité dépend aussi de la planète qui va capturer une planète intéressante. A partir de l'une de nos planètes, nous avons les formules suivantes :

- Pour les planètes neutres, nous avons :

$$\begin{aligned} \text{Rentabilité} = & \text{planet_src.numShips} + 20 \times \text{planet_src.growth} \\ & + (20 - \text{distance_temporelle}) \times \text{planet_dest.growth} - (\text{planet_dest.numShips} + 1) \end{aligned} \quad (1)$$

planet_src : Planète source à partir de laquelle on fait le test de rentabilité.

planet_dest : Planète sur laquelle on teste la rentabilité.

planet_dest.numShips+1 : Quantité optimale de flotte qu'il faut envoyer pour capturer une planète.

Dans la formule, les données du jeu sont représentées comme des structures. Ici, *numShips* représente donc le nombre de flottes disponibles sur une planète et *growth* son taux de croissance.

- Pour les planètes ennemis, la différence est qu'elles ont déjà un taux de croissance donc la quantité optimale de flotte à envoyer sera supérieure, car il faudra prendre en compte la distance temporelle qui la sépare de la planète source.

Nous avons donc la formule :

$$\begin{aligned} \text{Rentabilité} = & \text{planete_src.numShips} + 20 \times \text{planete_src.growth} \\ & + (20 - \text{distance_temporelle}) \times \text{planete_dest.growth} \\ & - (\text{planet_dest.numShips} + \text{distance_temporelle} \times \text{planete_dest.growth} + 1) \end{aligned}$$

Que l'on peut simplifier par :

$$\begin{aligned} \text{Rentabilité} = & \text{planete_src.numShips} + 20 \times \text{planete_src.growth} \\ & + (20 - 2 \times \text{distance_temporelle}) \times \text{planete_dest.growth} - (\text{planete_dest.numShips} + 1) \end{aligned} \quad (2)$$

On applique cette formule à partir de toutes nos planètes pour toutes les autres planètes, aussi bien neutres qu'ennemis. Ces deux formules nous donnent le gain que l'on va obtenir si l'on capture une certaine planète donc pour avoir la planète la plus rentable, il suffit de chercher le test qui renvoie la plus grande valeur.

Le calcul de la rentabilité se fait en temps constant donc la détermination de la planète la plus rentable se fait en temps quadratique par rapport au nombres de planètes sur la map.

L'un des autres aspects de la stratégie auxquels nous nous sommes intéressés, est la méthode d'envoi des flottes. Vaut-il mieux envoyer des petites flottes ou des grandes ? En effet, chaque méthode a ses avantages et ses inconvénients.

- **Envoyer le nombre de flottes nécessaire en une seule fois** : le plus gros inconvénient de cette stratégie est qu'une fois que la planète a envoyé ses flottes, elle se trouve vulnérable face un ennemi qui aurait par exemple pour stratégie d'attaque toutes les planètes ennemis à partir du moment où son nombre de flotte serait au-dessous d'un certain seuil. L'avantage est, bien sûr, que la planète qui justement attend avant d'envoyer ses flottes est en sécurité, car elle cumule beaucoup de flottes et donc a moins de chance de se faire attaquer. De plus, cette méthode est très simple à programmer, car la planète attendra de grossir avant d'attaquer et finira par attaquer une cible unique (pas besoin de gérer une attaque multiple en provenance de plusieurs planètes alliées vers une même cible).
- **Envoyer une suite de petites flottes** : l'avantage de cette méthode est le contraire de celui de la première, car la planète reste avec un nombre de flotte assez constant au cours de l'attaque, mais il n'est pas très grand donc nous avons jugé qu'il y avait une plus grande probabilité de se faire attaquer par d'autres IA. De plus, lorsque l'on envoie les flottes, il peut arriver de l'ennemie fasse de même sur la même planète donc celui qui va conquérir la planète en premier sera le gagnant. En envoyant de petit flottes, à moins d'avoir déjà envoyé toutes les flottes nécessaires et être plus proche que l'ennemie, nous

aurions peu de chance de gagner.

Au final, nous avons opté pour la première méthode, car nous avons trouvé qu'elle avait un meilleur compromis entre attaque et défense. Nous allons voir dans la suite que cette méthode nous est aussi utile lors de la stratégie de défense. De plus afin d'optimiser nos ressources, donc les flottes à disposition, nous avons décidé d'envoyer le strict minimum pour conquérir une planète. Qu'elle soit neutre ou ennemie, on envoie la quantité de flotte nécessaire à sa capture plus 1, la différence tenant au fait que pour une planète ennemie, il faut aussi prendre en compte son taux de croissance multiplié par la distance temporelle qui nous sépare dans le calcul de la quantité de flotte nécessaire. On voit d'ailleurs cette différence dans les formules de rentabilité (1) et (2).

La phase d'expansion, comme son nom l'indique permet de maximiser son nombre total de flottes afin de pouvoir ensuite se focaliser sur l'attaque de l'ennemie. Mais tout d'abord, intéressons-nous à la défense, une défense qui se fera en parallèle avec la phase d'expansion.

1.2.3 Phase de défense

Le problème de la défense était de savoir comment défendre une planète, c'est-à-dire, comment se comporter en cas d'attaque et donc de différencier les défenses intéressantes de celles qui impliqueraient une perte inutile de vaisseaux. En effet, à quoi bon protéger une planète s'il faut, pour la défendre, plus de flottes que ce dont on dispose.

Là encore, nous avons dû faire un choix sur le type d'envoi des flottes. Nous avons décidé d'envoyer des flottes à partir de la planète alliée la plus proche et qui a un taux de croissance supérieur à deux, car nous avons pensé que cela serait plus simple à gérer de cette façon et surtout moins risqué pour la planète qui défend. La planète qui défend ne doit pas se mettre en danger lorsqu'elle protège une autre planète donc il faut qu'elle puisse rapidement récupérer des flottes d'où l'intérêt de ne pas prendre les planètes avec un petit taux de croissance. Nous verrons plus tard que ce n'était pas forcément la meilleure solution. En premiers lieux, il faut savoir si l'ennemie envoie assez de flottes pour que l'une de nos planètes soit en danger et si tel est le cas, dans un second temps, on envoie juste assez pour la protéger.

Il a donc fallu déterminer ce qu'il fallait envoyer. Afin d'être sûr de protéger une planète, on s'intéresse à la distance temporelle avec la flotte la plus proche qui lui arrive dessus. On distingue alors deux cas :

- Si la planète qui défend est plus proche que la flotte qui attaque, on envoie :

$$\text{envoi} = \text{total_flottes_envoyées} - \text{distance_avant_arrivée} \times \text{planete_attaquée.growth} - \text{planete_attaquée.numShips} \quad (3)$$

distance_arrivée : Distance temporelle dont nous faisions référence juste avant.
planete_attaquée : Planète qui est attaquée.

— Si la planète qui défend est plus loin que la flotte qui attaque, on envoie :

$$\begin{aligned}
 \text{envoi} = & \text{total_flettes_envoyées} - \text{distance_avant_arrivée} \times \text{planete_attaquée.growth} \\
 & - \text{planete_attaquée.numShips} \\
 & + (\text{distance_entre_défense} - \text{distance_avant_arrivée}) \times \text{planete_attaquée.growth}
 \end{aligned} \tag{4}$$

distance_entre_défense : Distance temporelle entre la planète attaquée et celle qui défend.

Bien évidemment, la défense n'est possible que si la planète qui défend a assez pour le faire. Et donc, sur toute la durée de l'attaque, si l'on n'a pas les capacités de défendre alors on ne le fait pas. De plus, on a vu que le fait d'envoyer le total des flottes nécessaires à la capture d'une planète est un avantage pour la défense. En effet, si une planète est attaquée, elle prend en compte cette attaque, c'est-à-dire que ces flottes sont soustraites à ses ressources disponibles pour attaquer. Elle ne risque donc pas d'être capturée par l'ennemie si elle est attaquée et qu'elle envoie des flottes ailleurs. Il lui suffira donc juste d'attendre quelques tours pour qu'elle soit en sécurité. C'est donc aussi pour cette raison que nous avons choisi ce mode d'envoi.

1.2.4 Phase d'attaque

La deuxième phase de la stratégie est basée sur l'attaque. En effet, à partir d'un certain moment, on considère que la partie est quasiment gagnée, il suffit alors de se concentrer sur l'attaque. Arbitrairement, nous avons choisi qu'à partir du moment où l'on avait au total trois fois plus de flottes que l'ennemie ou que s'il ne restait plus de planètes neutres et que l'on avait plus de flottes que l'ennemie, nous pouvions commencer à attaquer.

Cette fois-ci, on ne s'intéresse plus à la défense au sens propre. On continue d'envoyer uniquement dans la mesure où l'on garde la planète, mais on se focalise vraiment sur l'attaque des planètes ennemis.

Le but étant de capturer toutes les planètes ennemis, nous sommes tout d'abord partis sur une stratégie de conquête un peu similaire à celle utilisée dans la phase d'expansion, mais nous nous sommes vite rendu compte qu'il y avait un problème. En effet, en attendant qu'une planète comporte assez de vaisseaux avant d'attaquer, il pouvait arriver que nous aillons beaucoup plus de flottes que l'ennemie au total, mais que celles-ci soient réparties sur nos différentes planètes de telle sorte qu'aucune d'elles ait assez pour pouvoir conquérir celle de l'ennemie d'un seul coup.

Le choix que l'on avait fait sur l'envoi des flottes par total nécessaire est donc inutile et contraincant surtout que l'on souhaite gagner le plus vite donc il vaut mieux attaquer l'ennemie avec toutes nos ressources directement.

Dans la mesure où une planète est en sécurité, elle va envoyer à toutes les planètes ennemis son taux de croissance. Nous avons choisi le taux de croissance comme valeur car cela permet de conserver une nombre de vaisseaux constants sur nos planètes et ainsi ne pas être vulnérable à un soudain changement de stratégie de l'adversaire.

1.2.5 Projet avorté : IAdéfensive

Une deuxième IA était en cours d'écriture mais celle-ci avait de trop grosses lacunes.
Elle ne devait pas attaquer l'adversaire, seulement conquérir des planètes neutres et se défendre.

Son principe était de choisir une planète mère parmi celles possédées où toutes les autres planètes enverraient l'intégralité de leurs flottes.
Cette planète devait être la plus proche des planètes ennemis.

Les avantages de cette IA furent :

- Facile à coder.
- Défense efficace.
- Bon match-up contre les stratégies très offensives (comme RageBot).

Facile à coder

Le code était relativement simple :

Chercher la planète mère.

Pour toutes les autres planètes faire

Toutes les planètes non attaquées envoient l'intégralité de leurs troupes à la planète mère.

Si une planète est attaquée : la planète mère y envoie un minimum de renforts pour la conserver.

Si l'ennemi a une production plus élevée que la notre :

Chercher la planète neutre la plus rentable.

Si celle-ci devient la nouvelle planète mère :

Y envoyer toutes les troupes.

Sinon, envoyer un minimum de troupes pour la capturer

Défense efficace

Nos planètes attaquées sont en théorie plus proche de la planète mère que de la planète attaquante.

Donc les troupes alliées arrivent toujours avant celles ennemis.

De plus, le temps que les troupes ennemis arrivent, la planète attaquée produit des troupes. Pour cela, la défense était en théorie très bonne.

Bon match-up contre les stratégies très offensives

Les stratégies très offensives, comme RageBot, sont inefficaces contre cette IA : même non terminée elle avait un ratio de victoire de 98% contre ce bot.

Cette IA a été arrêtée car en pratique une seule planète mère n'est pas suffisante.

Il est possible qu'une planète attaquée soit plus proche d'une planète ennemi que de la planète mère.

En cas de statut quo : les 2 joueurs possèdent la moitié des planètes, la perte d'une de ses planètes ferait perdre la partie.

Bien qu'avorté, le développement de cette IA nous a fait réfléchir sur de nombreuses stratégies et anticiper sur des cas que nous n'aurions pas prévus.

Elle a aussi servi de base pour développer la défense pour l'IA précédente.

2 Réalisation du projet

2.1 Communication avec le serveur du jeu

A chaque tour, le serveur écrit l'état du jeu dans l'entrée standard où sur chaque ligne se trouvent les informations d'une planète ou d'une flotte selon la forme suivante :

- Pour une planète : P {abscisse} {ordonnée} {propriétaire} {population} {production}
- Pour une flotte : F {propriétaire} {population} {source} {destination} {tours totaux} {tours restants}

La fonction `get_map()` lit le fichier txt que nous envoie le serveur et nous renvoie une structure contenant :

- un tableau de structure : planètes
- la taille de ce tableau
- un tableau de structure : flottes
- la taille de ce tableau

Structure planete :

```
ID : entier
x : réel
y : réel
propriétaire : entier
population : entier
production : entier
```

Structure flotte :

```
propriétaire : entier
population : entier
source : entier (ID de la planète source)
destination : entier (ID de la planète receptrice)
tours totaux : entier
tours restants : entier
```

Fonction `get_map() : structure map`

```
creer structure map
creer 2 entiers : nbPlanete et nbFlotte
créer structure tableau dynamique planete
creer structure tableau dynamique flotte
tant que la ligne lue n'est pas vide faire
    Lire la ligne, c'est une planete ?
        Si oui, l'ajouter au tableau planete et incrementer nbPlanete
        Si non, l'ajouter au tableau flotte et incrementer nbFlotte
        supprimer la ligne
renvoyer map.
```

2.2 Développement de l'intelligence artificielle

2.2.1 Étapes de construction

Afin de réaliser le programme de l'intelligence artificielle, nous avons procédé dans le même ordre que lors de la conception de celle-ci. Nous avons tout d'abord commencé par programmer la phase d'expansion car elle représente le noyau de notre stratégie.

L'idée était d'écrire une fonction qui permettrait de déterminer un certain type de planète (neutre, alliée, ennemie), la fonction *find_planets* répond à cette demande. Par contre nous eurent un problème lorsqu'il a fallut calculer la planète la plus rentable parmi celles qui ne nous appartenaient pas. En effet, la fonction qui retourne ces planètes ne fonctionnait sur cette stratégie alors qu'elle fonctionnait très bien sur le bot aléatoire. Nous avons donc dû faire autrement.

Nous déterminons la planète la plus rentable parmi celles qui sont neutres, puis parmi celles de l'ennemie et nous comparons leurs rentabilités pour garder la meilleure. De ce fait, nous avons du distinguer les cas où il reste des planètes neutres et celui où il n'en reste plus.

Pour calculer la rentabilité d'une planète, nous utilisons deux fonctions :

- *profitability* : Cette fonction calcule la rentabilité d'une planète à partir d'une autre planète.
- *most_profitable_planet* : Cette fonction fait le calcul de rentabilité pour toutes les planètes alliées vers toutes les autres planètes d'un certain type (neutres ou ennemis).

Cette stratégie d'expansion est très performante, car en une trentaine de tours, elle permet de conquérir un grand nombre de planètes, elle a donc une grande capacité d'expansion, on peut d'ailleurs en juger sur un match contre le *DualBot* :

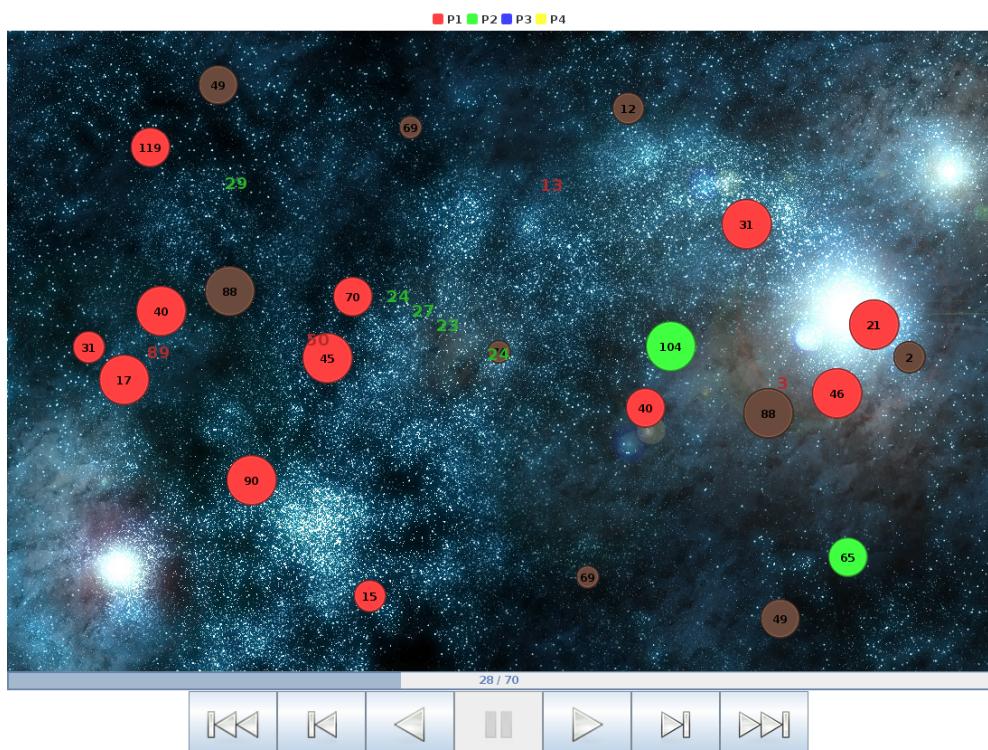


FIGURE 1 – Illustration de la vitesse d'expansion

Grâce uniquement à la stratégie de défense, il nous a été possible de battre plusieurs bots fournis sur une majorité des cartes. Nous verrons dans la suite un tableau récapitulatif des performances constatées.

Après l'expansion, nous nous sommes intéressés à la défense. Celle-ci fonctionne à priori très bien, nous n'avons pas constaté d'erreur dans son fonctionnement. Elle protège très bien les planètes alliées et malgré les attaques ennemis, elle n'en perd que très peu.

Sur la figure 2 ci-dessous, on voit une séquence de jeu contre le *DualBot* (qui est en vert) qui illustre le mécanisme de défense :

- Étape 1 : L'ennemis envoie 27 flottes vers une planète alliée.
- Étape 2 : Le tour suivant, la planète la plus proche a jugé qu'il y avait un danger et décide d'envoyer 3 flottes.
- Étape 3 : Dix tours plus tard, les flottes sont sur le point d'arriver sur la planète alliée.
- Étape 4 : Une fois arrivée, on remarque que la planète a une seule flotte et qu'elle nous appartient toujours.

Nous avons donc démontré l'efficacité de notre programme de défense.

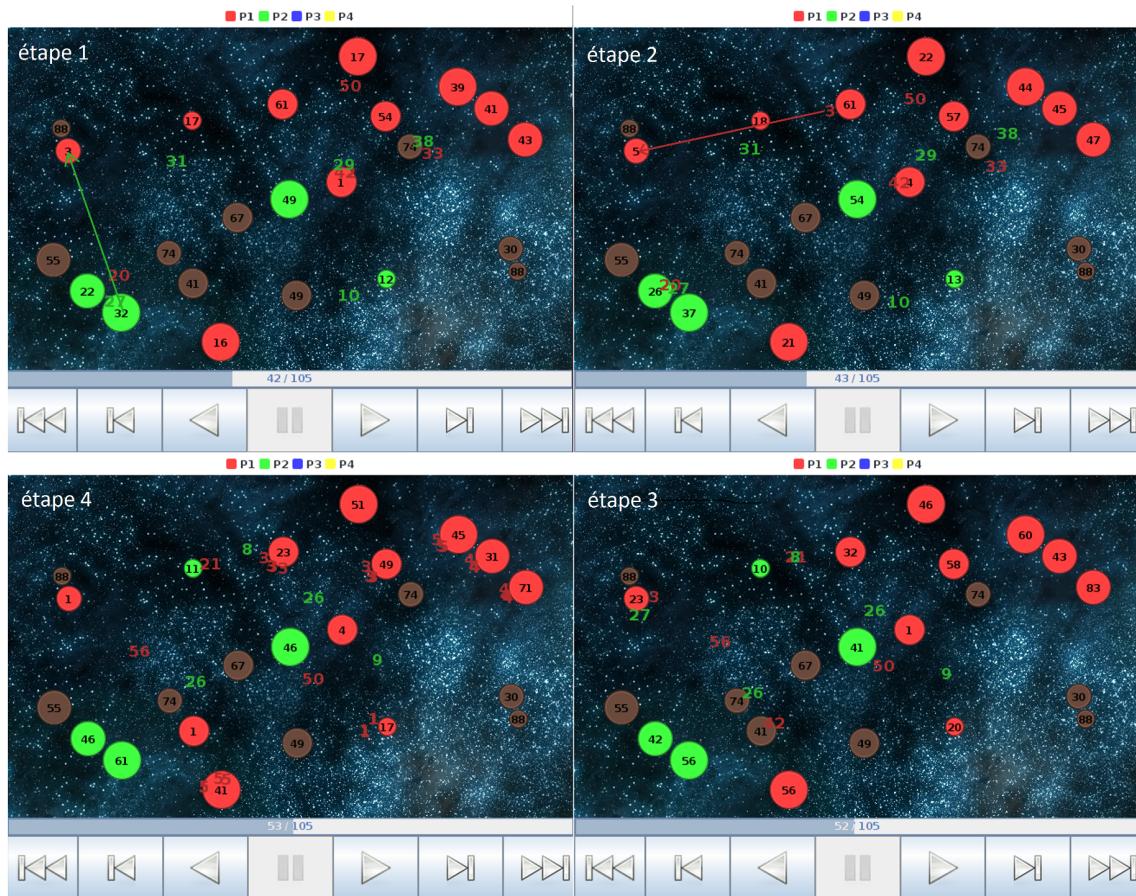


FIGURE 2 – Séquence de jeu illustrant la défense

Une fois la programmation de la défense terminée, nous avons encore une fois tester les performances de notre IA face à celles qui nous sont fournies.

La dernière étape de programmation fut celle de l'attaque. Cette phase d'attaque ne se met en place que sous certaines conditions expliquées plus tôt.

Dans la figure 3 ci-dessous, on voit le comportement totalement offensive de IA. Elle envoie de petites flottes sur les planètes ennemis afin de les capturer.

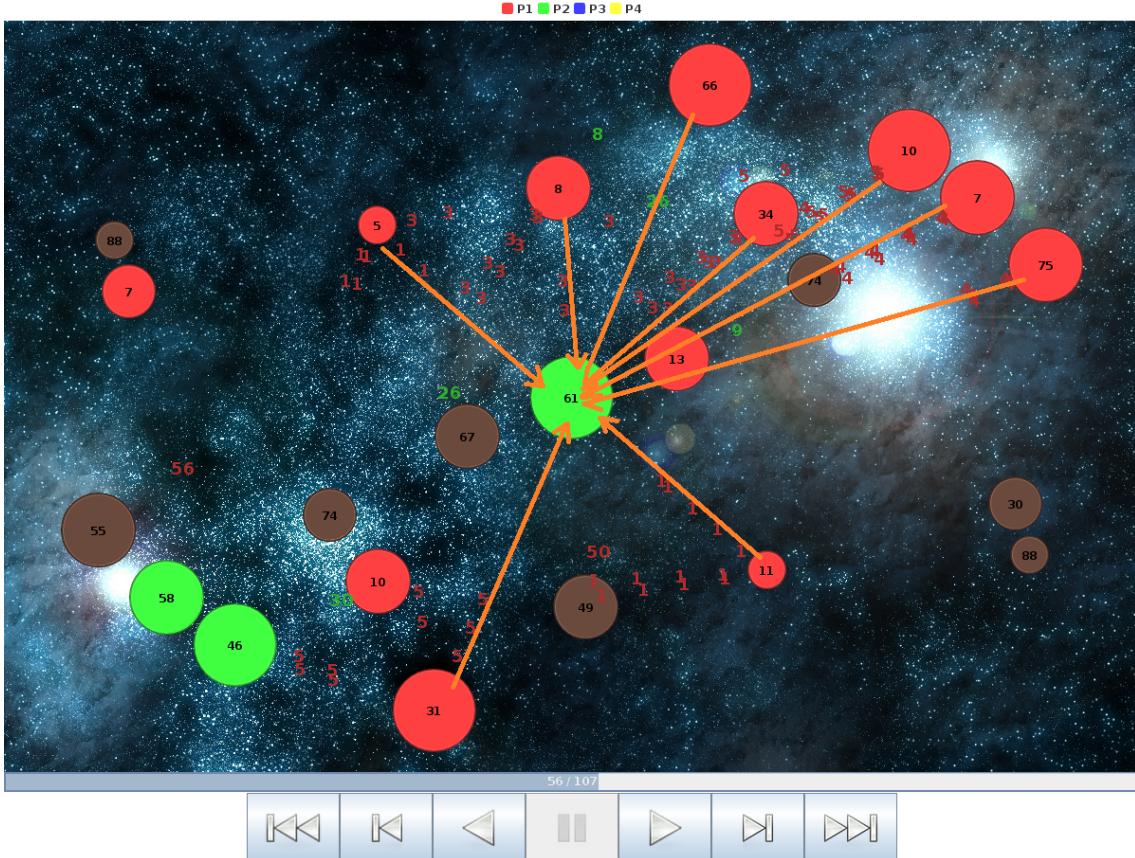


FIGURE 3 – Exemple du comportement de l'IA lors de la phase d'attaque sur une planète ennemie

2.2.2 Résultats

Aux différentes étapes de la programmation, nous avons effectué des tests que l'on peut résumer sous la forme de deux tableaux :

IA fournie	Phase d'expansion	expansion+défense	IA final
RageBot	80	97	98
DualBot	90	100	100
Prospector	96	99	99
BullyBot	95	100	100

FIGURE 4 – Pourcentage de victoire contre différentes IA

Le tableau ci-dessus donne le pourcentage de victoire contre certaine IA. Les trois colonnes de droite représentent notre IA aux différents stades de programmation. On voit que malgré

les différentes améliorations fournies lors de la programmation, le taux de victoire contre les IA fournies ne varie pas énormément. La Seule IA sur laquelle nous avons pu influer au cours de la progression du projet est le *RageBot*.

Dans notre cas, les performances de notre IA ne se reflètent pas vraiment sur ce genre de tests. Nous avons donc décidé de regarder le nombre moyen de tours qu'il faut pour gagner une partie. Dans le tableau ci-dessous, on remarque qu'il y a une progression à chaque étape de la réalisation. Les choix que nous avons fait au niveau de la stratégie se sont donc avérés utiles.

IA fournie	expansion	expansion+défense	IA final
RageBot	130	90	65
DualBot	135	110	95
Prospector	140	115	90
BullyBot	125	115	105

FIGURE 5 – Moyenne du nombre de tours pour gagner une partie

En plus des IA déjà fournies, nous avons eu l'occasion de faire des tests contre celles des autres groupes de la promotion. Nous avons été heureux de constater que sur un quinzaine d'IA différentes, il n'y eu qu'une seule qui était plus performante que la nôtre. Pour déterminer la meilleure IA, nous avons testé les IA sur toutes les cartes, ce qui nous a permis de déterminer le pourcentage de victoire et de défaite pour deux IA. Ce que nous avons constaté, c'est que la majorité de ces IA envoyait plein de petites flottes contrairement à nous. Notre choix sur le type d'envoi est donc plutôt efficace car il nous a pour l'instant donné une large majorité de victoires.

Nous avons aussi eu la chance de pouvoir affronter la meilleure IA française du concours *Planetwars*. Nous ne faisions clairement pas le poids mais cela nous a permis de réfléchir quant aux performances de notre IA et à ce que l'on pouvait modifier, voir penser totalement autrement pour corriger certains comportements.

2.2.3 Critique sur les choix adoptés et améliorations

Hormis les problèmes techniques liés à nos connaissances non absolues du langage C, nous avons réussi à surmonter les différents problèmes rencontrés lors de la programmation.

Nous avons remarqué que notre stratégie ne gère pas les gaspillages de flotte, c'est-à-dire qu'elle envoie quand même des flottes si l'ennemie en a déjà envoyé assez pour la conquérir une planète. En effet, ce problème n'est pas si simple à résoudre. On pourrait penser qu'il suffit d'envoyer un vaisseau de plus que l'ennemie. Mais il faut aussi prendre en compte la distance entre ses flottes et la planète attaquée et la distance qui nous sépare de celle-ci. Car une fois conquise, elle obtient un taux de croissance. Ce n'est donc peut-être pas avantageux de la prendre. Il faudrait trouver une autre formule de rentabilité pour ce genre de situation.

En ce qui concerne la défense, le plus gros problème était de déterminer ce qu'il fallait envoyer et d'où. Comme cela a été expliquer précédemment, la solution que nous avons choisi n'était pas vraiment la meilleure car en protégeant avec une seule planète, on peut risquer de la perdre si celle qui défend n'a pas assez de flottes.

On aurait aimé faire une défense qui utilise toutes les autres planètes alliées. Il y aurait alors eu moins de risques de perdre une planète.

Il y a aussi une autre idée que l'on aurait voulu mettre en place pour la défense. En effet, lorsqu'une planète est attaquée et qu'elle n'est pas défendue, elle pourrait rapatrier ses troupes vers une planète alliée. Mais dans ce cas, l'adversaire aurait déjà une planète avec un gros

nombre de troupes.

Pour l'attaque, nous étions initialement partis sur l'idée de charger des planètes *mères*, c'est-à-dire les planètes qui seraient les plus intéressantes pour attaquer. Si l'on charge une planète alliée, l'ennemie ne le verrait sûrement pas comme une attaque et une fois qu'il est attaqué, il n'aurait pas les moyens de se défendre. Pourtant malgré le bon fonctionnement de cette stratégie, elle n'était sûrement pas très optimale à cause du temps perdu inutilement à charger. C'est pour cela que nous avons changé de stratégie pour celle qui attaque directement les planètes ennemis.

Conclusion

Tout au long de la réalisation, nous n'avons pas noté de problème particulier au niveau de l'utilisation des données de jeu donc nous pouvons en conclure que la structuration des données utilisées a été efficace et que le premier sous-problème a été résolu avec succès. Dans le cas de la stratégie de l'IA, nous sommes assez satisfait de ses performances. Biensûr, elle n'est pas sans défaut, il y a quelques points que nous aurions bien souhaités corriger ou améliorer si l'on avait eu plus de temps, mais de ce que nous avons pu constater, elle nous donne de très bons résultats.

Ce projet nous a permis de mettre en pratique les connaissances en C que nous avons acquises et de comprendre le cheminement à suivre pour mener à bien un projet. Nous nous sommes également aperçus des difficultés du travail d'équipe sur un projet de ce type. En effet, nous avons trouvé qu'il est assez difficile de répartir le travail et d'écrire des fonctions compatibles sans concertations. Malgré les moments difficiles, le sujet nous a beaucoup intéressé par les nombreuses possibilités qu'il offrait.