

Guide d'utilisation

Framework Starship Battle Royal

SOMMAIRE

1. Généralité	3
1.1. Installation	3
1.2. Lancer le jeu	3
1.2.1. Client	3
1.2.2. Serveur	4
1.3. Raccourcis	5
1.4. Physique des objets	5
1.5. Dommages	5
1.6. Dead zone	6
1.7. Adresse IP local	6
2. Interface	8
2.1. Workspace > Interface > MobileObjectInterface	8
2.2. Workspace > Interface > ClientInterface	9
2.2.1. Données des objets mobiles	9
2.2.2. Commande de mouvement	10
2.2.3. Données de la partie	10
2.2.4. État du jeu	11
2.2.5. Fonctions utiles	11
2.3. Workspace > Interface > MotionCommandInterface	12
3. Workspace > myWorkspace	14
3.1. Votre code d'intelligence artificielle	14
3.2. Tester son intelligence artificielle	14
3.3. Architecture basique	14
3.4. Stratégies	15

1. Généralité

1.1. Installation

Avant d'installer le framework, vous devez avoir au moins installé préalablement Python3.7+. Si ce n'est pas le cas, vous pouvez le télécharger ici :

<https://www.python.org/downloads/>

Pour installer l'environnement, nous vous conseillons de créer un environnement virtuelle pour isoler l'installation des librairies uniquement nécessaire pour le framework. Ouvrez un terminal et exécutez les commandes suivantes :

- Décompressez le fichier ieee-competition-h2020.zip
- `cd ieee-competition-h2020/`
- `python -m pip install virtualenv`
- `python -m virtualenv venv`
- `python -m pip install -r requirements.txt`
- `./venv/bin/activate` (ou `./venv/Scripts/activate`)

Nous vous conseillons fortement de télécharger et d'installer l'IDE **PyCharm Community Edition** disponible ici : <https://www.jetbrains.com/pycharm/download/>

C'est simple d'utilisation !

1.2. Lancer le jeu

1.2.1. Client

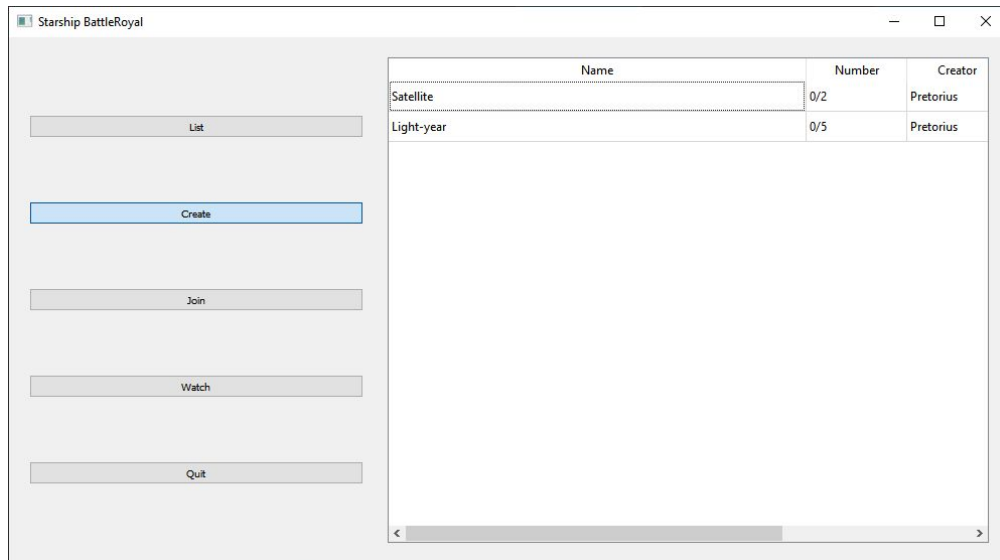
Pour lancer le jeu en mode client, vous devez le faire en ligne de commande comme suit :

- `cd src/`
- `python main.py [NomJoueur] client --ip [IPServer] --port [NumeroPort]`

En cas d'erreur, regarder le résultat dans la console. **Vous ne pouvez pas lancer un client si vous n'avez pas de serveur à l'adresse indiquée.**

> Exemple : `python main.py Steve client --ip 192.168.0.10 --port 8123`

Lorsque le client est lancé, vous avez la fenêtre suivante :



Plusieurs options s'offre à vous :

- **List :**
Permet de lister les parties qui sont disponibles sur le serveur. Le rafraîchissement ne se fait pas automatiquement, donc n'hésitez pas à rappuyer.
- **Create :**
Permet de créer une partie sur le serveur, sans la joindre.
- **Join :**
Permet de joindre une partie existante dans la liste de droite. Si vous êtes déjà dans la partie ou que la partie des complètes, vous ne pourrez la rejoindre.
- **Watch :**
Permet d'observer une partie en mode spectateur.
- **Quit :**
Permet de fermer le client.

1.2.2. Serveur

Pour lancer le jeu en mode serveur, vous devez le faire en ligne de commande comme suit :

- `cd src/`
- `python main.py [NomServeur] server --ip [IPServer] --port [NumeroPort]`

> Exemple : `python main.py Master server --ip 192.168.0.10 --port 8123`

À partir de là, vous n'avez plus rien à faire à part donner l'adresse de votre serveur aux autres joueurs pour qu'ils puissent rejoindre la partie.

1.3. Raccourcis

Lorsqu'une partie est en cours, vous avez quelques raccourcis de disponibles :

- **Touche M**

Permet de changer de mode de vision de la partie parmi:

- Graphisme HD
- Graphisme HD + minimap en bas à droite
- Minimap seulement



> ASTUCE: Si vous avez des problèmes de performance, utiliser la **Minimap seulement**.

- **Touche Espace**

Permet de synchroniser ou non le déplacement de la caméra avec votre vaisseau.

Lorsque la caméra est en mode libre, vous pouvez utiliser votre souris pour vous déplacer dans la partie.

Si vous êtes en mode spectateur ou que votre vaisseau est détruit, alors vous pouvez utiliser les touches ← et → pour changer le focus de la caméra en allant d'un joueur à un autre.

1.4. Physique des objets

Il s'agit de la même physique que vous pouvez rencontrer lorsque vous jouez au billard. Le poids et le rayon des objets mobiles sont pris en compte pour calculer les trajectoires.

Pour les plus scientifiques d'entre vous, il s'agit de collision élastique classique que vous avez pu étudier au CEGEP.

1.5. Dommages

Les dommages sont à la touches et n'est pas proportionnel à votre vitesse. Toutes les touches se trouvent dans le tableau suivant :

Objet	Dégâts	Points de vie
-------	--------	---------------

Vaisseau (ShipModel)	0.10	1.0
Astéroïde (AsteroidModel)	0.025	-
Projectile (BoltModel)	0.05	0.01
Dead zone	0.20 / seconde	-

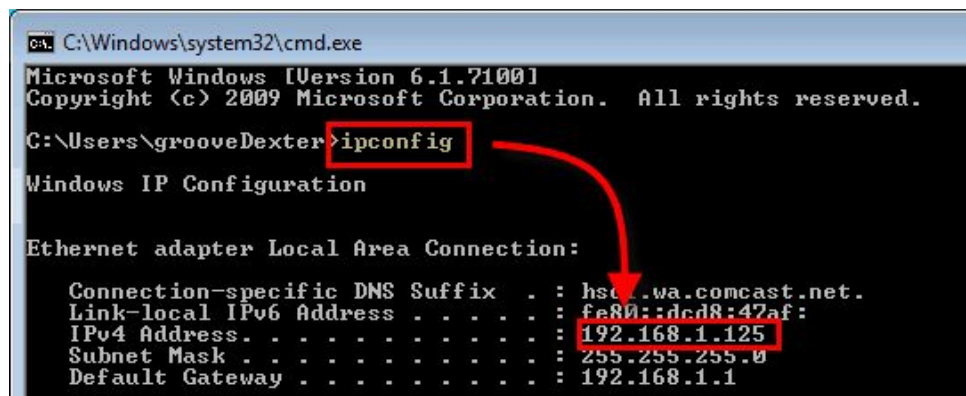
1.6. Dead zone

Une zone circulaire entoure la zone de combat. Cette zone rétrécit jusqu'à englober l'entièreté de la zone au bout de 2:30 minutes.

Lorsque votre vaisseau se trouve dans la dead zone, il subit des dégâts (voir 1.5 pour la valeur). Aussi, tous les projectiles sont détruits dans la dead zone. Ce qui signifie que vous ne pouvez pas tirer, mais vous ne pouvez pas non plus recevoir de tir.

1.7. Adresse IP local

- Windows
 - Ouvrez un terminal de commande
 - Tapez : `ipconfig`



```

C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7100]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\grooveDexter>ipconfig

Windows IP Configuration

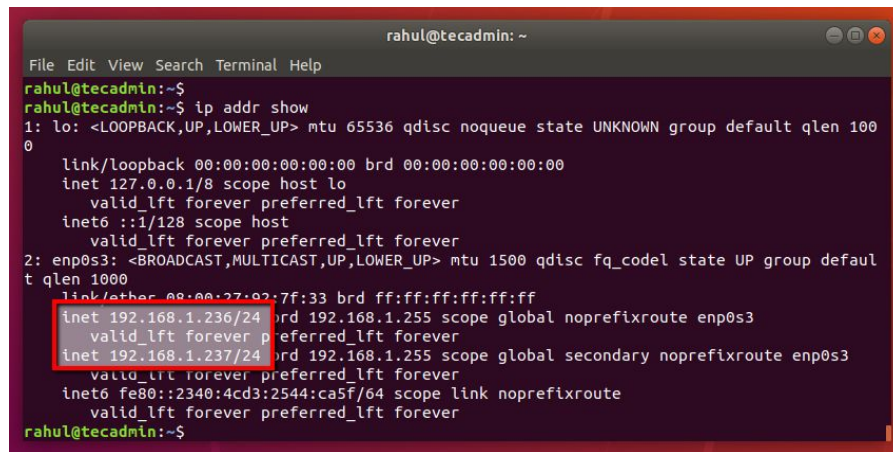
Ethernet adapter Local Area Connection:

    Connection-specific DNS Suffix . : hsd1.wa.comcast.net.
    Link-local IPv6 Address . . . . . : fe80::dcd8:47af:
    IPv4 Address. . . . . : 192.168.1.125
    Subnet Mask . . . . . : 255.255.255.0
    Default Gateway . . . . . : 192.168.1.1

```

- Ubuntu (Linux)
 - Ouvrez un terminal de commande (Ctrl + T)

- Tapez: `ip addr show`

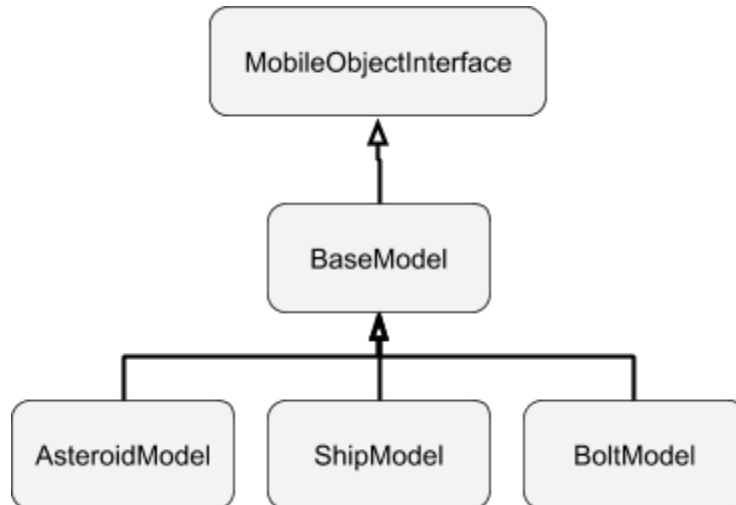


```
rahul@tecadmin: ~  
File Edit View Search Terminal Help  
rahul@tecadmin:~$  
rahul@tecadmin:~$ ip addr show  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000  
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00  
    inet 127.0.0.1/8 scope host lo  
        valid_lft forever preferred_lft forever  
    inet6 ::1/128 scope host  
        valid_lft forever preferred_lft forever  
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000  
    link/ether 00:00:27:02:7f:33 brd ff:ff:ff:ff:ff:ff  
    inet 192.168.1.236/24 brd 192.168.1.255 scope global noprefixroute enp0s3  
        valid_lft forever preferred_lft forever  
    inet 192.168.1.237/24 brd 192.168.1.255 scope global secondary noprefixroute enp0s3  
        valid_lft forever preferred_lft forever  
    inet6 fe80::2340:4cd3:2544:ca5f/64 scope link noprefixroute  
        valid_lft forever preferred_lft forever  
rahul@tecadmin:~$
```

2. Interface

2.1. Workspace > Interface > MobileObjectInterface

BaseModel est la classe mère des classes AsteroidModel, ShipModel et BoltModel. Il s'agit des classes qui sont utilisées pour stocker l'information des objets mobiles dans le jeu.



Les méthodes suivantes sont les plus utiles pour le développement de votre intelligence artificielle :

BaseModel.life

Retourne un float qui correspond au nombre de point de vie restant de l'objet mobile.

BaseModel.is_alive

Retourne un bool pour savoir si l'objet mobile est en vie ou non.

BaseModel.radius

Retourne un int qui correspond au rayon de l'objet mobile.

BaseModel.x

Retourne un float qui correspond à la position de l'objet mobile dans le repère des X

BaseModel.y

Retourne un float qui correspond à la position de l'objet mobile dans le repère des Y

BaseModel.xy

Retourne un ndarray (exemple: np.array([0, 0])) qui correspond à la position de l'objet mobile dans le repère des XY sous la forme d'un vecteur 2 dimensions.

BaseModel.dir

Retourne un ndarray (exemple: np.array([0, 1])) qui correspond à la direction dans laquelle est positionné l'objet mobile sous la forme d'un vecteur 2 dimensions.

Astuce: Vous pouvez déterminer la position à 100 pixels devant l'objet mobile avec la formule suivante :

```
> mob.xy + mob.dir * 100
```

BaseModel.dir_x

Retourn un float qui correspond à la direction dans laquelle est positionné l'objet mobile dans le repère des X.

BaseModel.dir_y

Retourn un float qui correspond à la direction dans laquelle est positionné l'objet mobile dans le repère des Y.

BaseModel.angle

Retourne un float qui correspond à l'angle entre 0 et 360 degré dans laquelle est positionné l'objet mobile.

BaseModel.get_propulsion_speed()

Retourne un float qui correspond à la vitesse en pixels par seconde de l'objet mobile dans la direction à laquelle il est positionné. La vitesse ne peut pas être négative.

BaseModel.get_rotation_speed()

Retourne un float qui correspond à la vitesse de rotation en pixels par seconde de l'objet mobile. L'objet mobile tourne à droite lorsque la valeur est positive et à gauche lorsque la valeur est négative.

2.2. Workspace > Interface > ClientInterface

2.2.1. Données des objets mobiles

ClientController.get_name()

Retourne votre pseudonyme que vous avez défini.

ClientController.get_mobs_in_radius(near_point, radius, enemy_only=False)

Retourne une liste des objets mobiles dans un périmètre circulaire de centre *near_point* et de rayon *radius*. Il est possible de spécifier que vous souhaitez n'avoir que les objets mobiles ennemies avec l'option *enemy_only*.

ClientController.get_data_from_players(enemy_only=False)

Retourne une liste de *ShipModel* de tous les joueurs dans la partie. Il est possible d'exclure votre vaisseau grâce à l'option *enemy_only*.

ClientController.get_data_from_mobs(enemy_only=False)

Retourne une liste de *ShipModel* et/ou d'*AsteroidModel* et/ou de *BoltModel* dans la partie. Il est possible d'exclure votre vaisseau et ses projectiles grâce à l'option *enemy_only*.

ClientController.get_player(name)

Retourne le *ShipModel* du joueur dont vous avez spécifié le nom grâce à l'argument *name*.

ClientController.get_players_name()

Retourne la liste contenant tous les noms des joueurs qui sont dans la partie.

ClientController.get_data_from_asteroids()

Retourne la liste d'*AsteroidModel* qui se trouve dans la partie.

ClientController.get_asteroid(name)

Retourne l'*AsteroidModel* spécifié par *name*.

ClientController.get_data_fromBolts()

Retourne une liste des *BoltModel* qui se trouvent dans la partie.

2.2.2. Commande de mouvement

ClientController.set_motion_command(motion)

Permet de spécifier une commande de déplacement *motion* qui est un *MotionModel* à destination du serveur pour commander votre vaisseau.

ClientController.get_motion_from_party()

Retourne une liste des *MotionModel* de tous les joueurs.

2.2.3. Données de la partie

ClientController.wait_until_event()

Fonction qui va bloquer tant que le serveur n'a pas mis à jour les données de la partie. Il y a un time out de 1.0 seconde. Lorsque les données sont mise à jour, la fonction retourne *True*, lorsqu'elle est en time out elle retourne *False*.

ClientController.get_time_left()

Retourne le temps restant pour passer d'un état à un autre de la partie. Il se peut que la partie finisse lorsqu'il ne reste plus qu'un joueur par exemple.

ClientController.get_dead_zone_radius()

Retourne la distance à laquelle se trouve la dead zone.

ClientController.get_dead_zone_center()

Retourne la position (X, Y) du centre de la dead zone.

ClientController.get_winner()

Retourne le nom de la personne qui est victorieuse de la partie.

ClientController.get_rank_board()

Retourne le classement des joueurs lorsque la partie est terminée.

2.2.4. État du jeu

ClientController.is_party_done()

Permet de savoir lorsqu'une partie est terminée.

ClientController.is_party_waiting()

Permet de savoir si le serveur cherche d'autres joueurs pour atteindre la limite de la partie.

ClientController.is_party_ready_to_fight()

Permet de savoir si le décompte de la partie est en cours.

ClientController.is_party_in_progress()

Permet de savoir si la partie est en cours de progression.

ClientController.has_party_winner()

Permet de savoir si une partie possède un vainqueur.

ClientController.is_spectator()

Permet de savoir si le client est un spectateur ou un joueur.

2.2.5. Fonctions utiles

ClientController.get_the_nearest_mob(point, mob_list)

Retourne la distance et l'objet mobile le plus proche de la position *point* provenant de la list de *mob_list*.

ClientController.get_mobs_inside_area(point, radius, mob_list)

Permet de filtrer les objets mobiles de *mob_list* pour savoir s'ils sont dans une zone circulaire de centre *point* et de rayon *radius*.

ClientController.get_distance_between(mob1, mob2)

Retourne la distance entre deux objets mobiles.

ClientController.get_direction_between(origin_mob, target_mob)

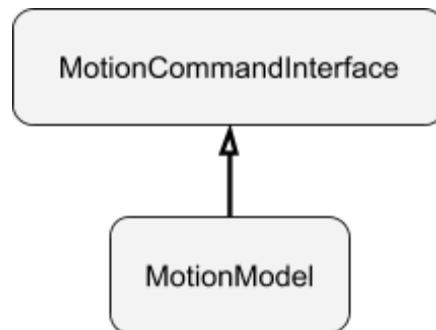
Retourne la direction (vecteur unitaire 2 dimensions) de l'objet mobile *target_mob* en fonction du point de vue de *origin_mob*.

ClientController.get_middle_point_between(origin_mob, target_mob)

Retourne la position (vecteur 2 dimensions) qui se trouve à mi-distance entre les deux objets mobiles.

2.3. Workspace > Interface > MotionCommandInterface

La classe MotionCommandInterface est la classe mère pour formater une commande pour contrôler votre vaisseau du côté du serveur.



Les méthodes suivantes sont les plus utiles pour le développement de votre intelligence artificielle :

MotionModel.acceleration

Retourne la valeur d'accélération en pourcentage (entre -1.0 et 1.0) qui se trouve dans la commande destinée au serveur. Il s'agit de l'accélération qui sera appliquée à votre vaisseau par le serveur. -1.0 correspond à une décélération maximale alors que 1.0 correspond à une accélération maximal.

MotionModel.rotation

Retourne la valeur de rotation en pourcentage (entre -1.0 et 1.0) qui se trouve dans la commande destinée au serveur. Il s'agit de l'accélération qui sera appliquée à votre vaisseau par le serveur. -1.0 correspond à un virage à gauche, alors que 1.0 correspond à un virage à droite.

MotionModel.set_rotation(percent)

Applique la valeur en pourcentage (entre -1.0 et 1.0) pour le sens de rotation qui sera enregistré dans le MotionModel à destination du serveur.

MotionModel.set_acceleration(percent)

Applique la valeur en pourcentage (entre -1.0 et 1.0) pour l'accélération qui sera enregistré dans le MotionModel à destination du serveur.

MotionModel.shoot()

Applique l'état SHOOT à votre vaisseau. Lorsque le vaisseau est en mode SHOOT, le serveur va ignorer les commandes de rotation et d'accélération. Le vaisseau ne peut pas tirer plus d'un laser toutes les 250 ms (4 tirs par secondes).

MotionModel.move()

Applique l'état MOVE à votre vaisseau. Lorsque le vaisseau est en mode MOVE, le serveur appliquera les commandes de rotation et d'accélération. Le vaisseau ne peut pas tirer en même temps que se déplacer.

MotionModel.stop()

Applique l'état STOP à votre vaisseau. Lorsque le vaisseau est en mode STOP, le serveur appliquera des commandes pour l'immobiliser le plus rapidement possible.

.

MotionModel.set_shoot_dir(direction)

Appliquer une direction pour tirer. Il s'agit d'un vecteur unitaire de 2 dimensions.

Astuce: Je veux que mon vaisseau tire à la position Nord-Ouest (135 degrés) alors :

```
> motion.set_shoot_dir( np.array( [-1.42, 1.42 ] ) )
```

3. Workspace > myWorkspace

3.1. Votre code d'intelligence artificielle

Vous devez travailler avec une classe qui hérite de la classe *BaseWorkspace* dans le dossier *src/Workspace/*. Un fichier *ExampleWorkspace* est disponible pour que vous puissiez comprendre le fonctionnement de cette classe.

Votre espace de travail se trouve dans le fichier *Workspace/myWorkspace.py* dans la méthode *loop* de la classe *myWorkspace*. Cette classe vous donne accès à une interface client (voir 2.2) par le biais de *self.ctrl*.

Aussi, la classe qui sera utilisée par le framework pour exécuter votre intelligence artificielle est déterminé lors de la création de la classe *ClientController* à la ligne 32 du fichier *src/main.py*.

3.2. Tester son intelligence artificielle

Nous vous conseillons vivement de tester votre intelligence artificielle directement sur votre ordinateur.

Vous pouvez lancer à la fois un serveur et plusieurs clients en local. Cela vous permettra de créer des parties et de tester la robustesse de votre code avant de vous mesurer à vos concurrents.

3.3. Architecture basique

Basiquement, vous aurez 3 modules à développer dans votre intelligence artificielle.

Le premier est le **module de navigation** qui doit vous permettre de naviguer tout au long de la partie sans vous faire percuter par un astéroïde qui permet également de ne pas vous retrouver dans la dead zone et d'éviter les tirs des autres joueurs.

Le second est le **module de tir** qui doit vous permettre de tirer dans une direction pour faire du dégât à votre adversaire. Attention ! Lorsque vous tirer, vous ne pouvez pas changer de direction, ni jouer avec votre propulsion.

Le troisième est le **module stratégique** qui doit vous permettre d'analyser la partie pour prendre des décisions stratégiques sur le déplacement de votre vaisseau et cible à abattre.

3.4. Stratégies

Si vous n'avez pas trop d'idée sur la stratégie à adopter pour votre intelligence artificielle, vous êtes à la bonne place.

N'oubliez pas que vous êtes dans une partie "Chacun pour soi" et que vous pouvez vous retrouver à vous faire tirer dessus par plusieurs assaillants.

Il y a donc plusieurs stratégies que vous pouvez utiliser :

- **Cache-cache**

La meilleure défense est de se cacher des autres joueurs. Vous pourrez vous trouver un astéroïde et vous cacher derrière tout au long de la partie. Si vos adversaires sont assez mauvais, vous pourrez gagner la partie !

C'est aussi la meilleure stratégie pour les débutants, car elle va vous permettre de vous concentrer sur le module de déplacement en négligeant les autres modules.

- **Chasseur**

La meilleure défense, c'est l'attaque ! Vous êtes à l'aise avec le développement de tous les modules et vous êtes déterminés à neutraliser tous les joueurs en les chassant.

C'est la stratégie la plus complexe à élaborer, mais vous aurez clairement l'avantage sur vos adversaires qui n'auront pas développé assez leur intelligence artificielle.

- **Cogneur**

Votre seul but est de pousser les adversaires en dehors de la carte en leur fonçant dessus. Attendez le bon moment pour pousser un adversaire dans un banc d'astéroïdes ou dans la dead zone. Sinon, vous pouvez tout simplement foncer dans un banc d'astéroïdes et profiter des multiples rebonds pour tirer en vous déplaçant.

C'est une stratégie risquée, mais qui peut donner des résultats surprenants !

BONNE CHANCE !