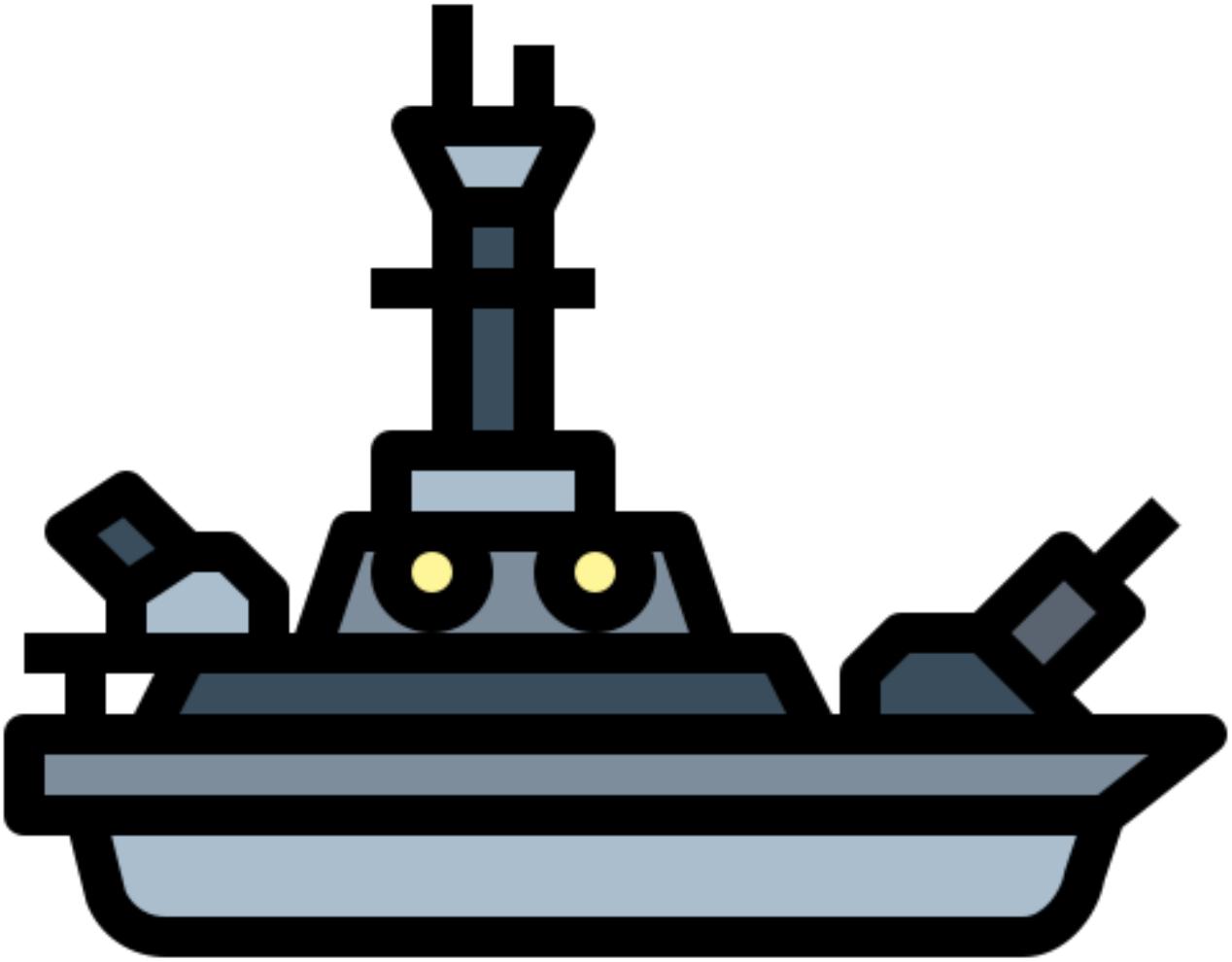


U.E. 1.3 : PROJET INFORMATIQUE
« BATAILLE NAVALE »

ENSTA Bretagne
2 rue F. Verny
29806 Brest Cedex 9,
France

JAUGEY Romain, TD1
romain.jaugey@ensta-bretagne.org

JACOBS Arne, TD1
arne.jacobs@ensta-bretagne.org

Sommaire

Introduction

Le programme a pour but de recréer le jeu de bataille navale grâce à Python. L'idée est de faire une bataille navale contre la machine ou contre un ami à vos côtés. Nous avons pour but de créer trois versions d'IA présentant chacune une difficulté différente allant de facile à difficile en passant par moyen.

Le joueur peut au début du jeu choisir : « Facile », « Moyen », « Difficile » ou « 2 joueurs ». Ensuite, pour jouer il commencera par donner la localisation de ses navires en donnant les coordonnées dans une grille de 10 sur 10. La partie se finit lorsque l'un des joueurs ou l'IA fait couler tous les navires de l'adversaire.

1 Idées générales

Avant de s'intéresser à toutes les classes du programme on peut commencer par donner une description assez globale de son fonctionnement. Ce programme est majoritairement centré autour de la classe IA. Celle-ci contient la plupart des méthodes nécessaires au bon déroulement du jeu. Ainsi, l'on y trouve les deux méthodes les plus importantes que sont : *create_grille_def()* et *place_bateau()*. Comme on peut assez facilement le deviner avec leurs noms, la première crée la grille de défense que l'on remplit ensuite avec les navires grâce au second. Je parle ici de grille de défense car en effet il y a aussi une grille d'attaque. Celle de défense représente notre propre grille avec la position de nos navires que va devoir attaquer l'adversaire. Elle n'est modifiée qu'en début de partie lorsqu'on place ses navires. La grille d'attaque quant à elle est constamment modifiée à travers la partie lorsque les joueurs choisissent des cibles.

Autrement, il y a 3 sous-classes pour les différentes difficultés de jeu et une classe JOUEUR pour permettre au joueur de créer sa grille et interagir avec l'adversaire. On trouve aussi une sous-classe TEMOIN pour analyser les trois difficultés.

Finalement, nous aimerions créer une dernière classe nommé AUTODIDACTE qui devrait, comme AlphaZero aux échecs, s'améliorer au fil des parties afin d'être capable de s'adapter au jeu de son adversaire et être ainsi plus performante. Pour cela nous créerions une base de données dans laquelle seront stockées toutes les informations utiles pour faire évoluer cette IA.

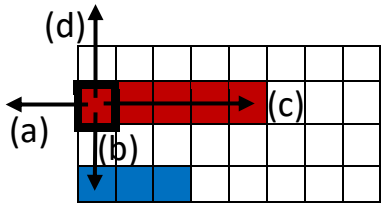
2 Description des classes

2.1 La classe IA()

Cette classe contient toutes les méthodes et variables utiles à la mise en place du jeu. On y crée la grille d'attaque, les différents navires, et quelques autres variables. De plus on y trouve un certains nombres de petites méthodes tels que *clean()*, *defend()*, *est_touche()*, … permettant de mettre en place les différentes parties du jeu et de récolter des informations qui seront utiles lors de la création de l'interface homme-machine. Plus particulièrement, on y trouve la méthode *create_grille_def()* et *place_bateau()* qui sont les programmes le plus importants. Finalement, la classe IA contient cinq sous-classes que sont les trois difficultés de jeu, le témoin afin de pouvoir lancer des tests unitaires, c'est-à-dire d'obtenir le nombre de coups moyens pour finir une partie et DUELTEST pour lancer la partie entre le témoin et l'une des IA.

2.2.1 La méthode *create_grille_def()*

Cette méthode permet à l'IA de créer sa grille de défense. On crée d'abord la grille de dimensions (10, 10) puis on ajoute un par un les navires. Lorsque les navires sont positionnés aléatoirement on fait bien attention à ce que les navires ne dépassent pas du plateau, ne se chevauchent pas et qu'ils y soient bien tous. Donc on choisit aléatoirement un point de départ pour le navire puis on vérifie s'il peut être placé en commençant par ce point dans une des 4 directions possibles.



Voici un exemple pour le positionnement des navires. En voulant placer le navire de longueur 5 le programme choisit la case en gras. Les direction a et d sont hors tableau et la direction b est bloquée par un autre navire donc le programme choisit la direction c.

2.2.1 La méthode *place_bateau()*

Cette méthode utilise *create_grille_def()* pour créer la grille de l'IA puis se sert d'une méthode interne *verif_bonne_grille()* pour vérifier si le placement des navires est bien valide. Si ce n'est pas le cas, le programme se relance jusqu'à avoir une grille correcte.

2.2 La classe FACILE(IA)

Cette classe représente le premier niveau d'adversaire automatique. C'est la version la plus simple. Dans sa méthode d'attaque nommée *attaque()* cette IA lance simplement des attaques aléatoires en évitant les cases déjà attaquées mais sans prendre en compte les informations données par une touche.

2.3 La classe MOYENNE(IA)

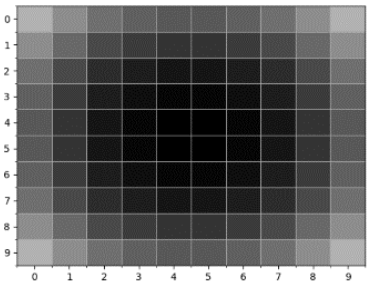
Cette classe représente le second niveau d'adversaire automatique. Cette version joue premièrement comme la précédente mais dès qu'il touche un navire il continue avec des attaques liées à cette case pour faire couler le navire. Dans sa méthode d'attaque nommée *attaque()* cette IA, dès qu'elle fait une touche, attaque les cases autour jusqu'à refaire une touche. Alors l'IA connait la direction du navire (horizontal ou vertical) et continue d'attaquer donc d'attaquer selon cette direction pour être certain de le couler.

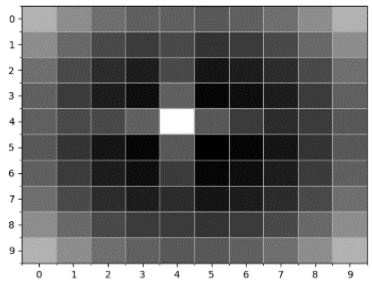
2.4 La classe DIFFICILE(IA)

Cette classe représente le troisième niveau d'adversaire automatique. Cette version joue d'une toute autre façon que les deux précédentes. Dans sa méthode d'attaque nommée *attaque()* cette IA utilise les probabilités qu'un navire soit à un endroit en fonction de la taille de ceux-ci, des navires restants en jeu et des cases déjà attaquées auparavant. On parle de probabilités mais en réalité on calcule simplement le nombre de navires qui peuvent être placés en passant par cette case. Lorsqu'il touche une case il cherche aussi à finir de couler l'entièreté du navire comme précédemment mais pour cela l'IA utilise encore les probabilités afin d'attaquer les meilleurs côtés en premier. Afin de trouver ces probabilités le programme recalcule à chaque fois toutes les possibilités de placement des navires encore en jeu puis regarde sur quelle case la probabilité est la plus forte.

Illustrons par un exemple rapide cette méthode :

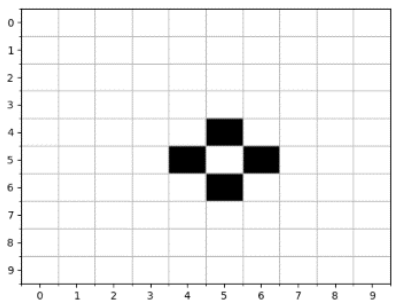
Au début du jeu la grille de probabilités ressemble à ceci (foncé = probabilité élevée)



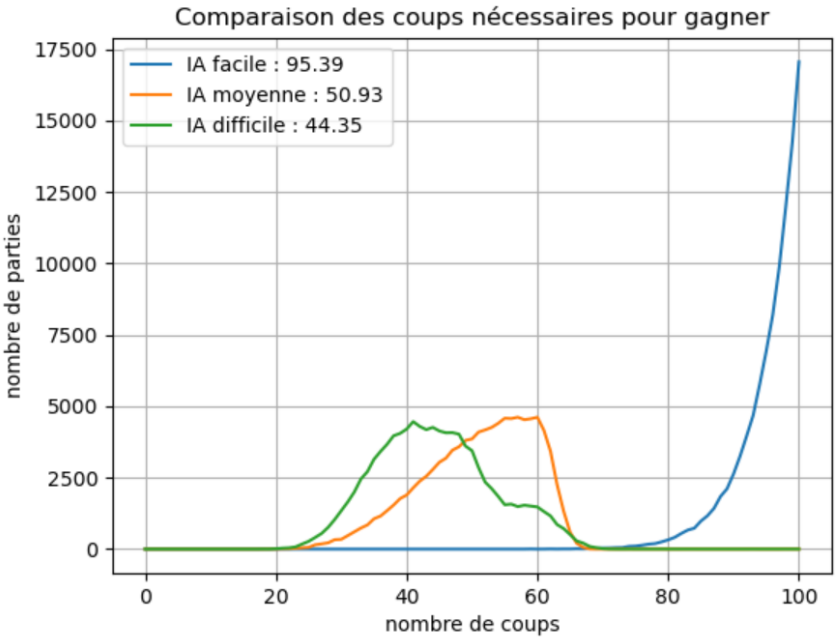


On joue un premier coup en (4, 4) où la probabilité est la plus élevée (ici on n'a touché aucun bateau adverse)

On joue ensuite en (5, 5) et on touche enfin un bateau ennemi ce qui transforme notre grille de probabilités de la manière suivante : ici on se trouve au centre de la grille donc rien ne gêne la pose de bateaux ce qui explique que chaque probabilités adjacente soient égales (ce n'est pas toujours le cas)



Voici un graphe représentant l'efficacité de ces 3 dernières classes lors de 100 000 parties simulées pour chaque IA ainsi que le nombre moyens de coups utilisés :



Il est bon de faire une remarque à propos des temps d'exécutions : les IA facile et difficile peuvent terminer environ 1000 parties par seconde (l'IA moyenne est plus rapide car plus efficace mais cela reste négligeable) tandis que l'IA difficile termine près de 50 parties par seconde. Ainsi malgré le nombre de calculs importants, lors d'une partie contre un joueur le résultat d'un seul coup est perçu comme instantané.

2.5 La classe TEMOIN(IA)

Cette classe sert d'adversaire aux autres trois autres classes présentés précédemment. Il est utilisé comme adversaire témoin lors d'un grand nombre de parties pour voir le nombre de coups min, max et moyen nécessaire à l'IA opposée pour le battre et couler tous ses navires. Le témoin attaque toujours la même case pour être sûr que l'IA testé gagne bien. Avec ces résultats on peut alors vérifier que FACILE mets plus de coups que MOYEN et que ce dernier en mets plus que DIFFICILE et aussi que les différences soient assez grande pour pouvoir vraiment parler de différents niveaux.

2.6 La classe DUELTEST()

Cette classe est utilisé pour lancer les combats entre le TEMOIN et la classe FACILE, MOYEN ou DIFFICILE. Elle utilise la méthode *combat()* qui lance le combat entre les deux joueurs, ici le TEMOIN et une des trois autres IA. Pour cela on a les fonctions *etape_1_sur_2(self, j1, j2)* et *etape_2_sur_1(self, j1, j2)* qui servent à lancer les attaques du joueur 1 sur 2 et à mettre à jour la grille de défense du joueur 2 et inversement. Cette alternance continue jusqu'à ce que le joueur 2 coule tous les navires du TEMOIN.

2.7 La classe DUELREEL()

Cette classe est utilisé pour lancer les combats entre le joueur humain et la classe FACILE, MOYEN, DIFFICILE ou un deuxième joueur. Elle utilise la méthode *combat()* qui lance le combat entre les deux joueurs. Pour cela on a les fonctions *etape_1_sur_2(self, j1, j2)* et *etape_2_sur_1(self, j1, j2)* qui servent à lancer les attaques du joueur 1 sur 2 et à mettre à jour la grille de défense du joueur 2 et inversement. Cette alternance continue jusqu'à ce que l'un des deux joueurs humains ou l'IA gagne en coulant tous les navires de l'adversaire.

Autrement, cette classe possède aussi le module *compte_rendu()* qui comme son nom l'indique sert à rédiger un compte rendu de la partie. Le module rédige au fur et à mesure les coups joués et s'ils ont touché ou coulé un navire. Ainsi, à la fin de la partie on a un fichier texte dans lequel est rédigé le déroulement de celle-ci.

2.8 La classe JOUEUR()

Cette classe sert d'intermédiaire au joueur. Elle permet dans un premier de placer les bateaux selon la volonté du joueur (avec toujours la possibilité du placement aléatoire si le joueur le veut). Elle permet aussi au joueur de rentrer les coordonnées des cases qu'il veut attaquer et de lancer l'attaque sur la grille de l'IA ou du joueur en face. La méthode *attaque()* est ici simplement une récupération de ce que le joueur humain veut comme coordonnées d'attaque. Ces coordonnées sont alors transmises à la classe DUELREEL qui lance une attaque avec.

2.9 La classe AUTODIDACTE()

Cette classe est un quatrième adversaire possible pour un joueur humain. Elle a pour but de s'améliorer au fil des parties en se basant sur ses propres parties ou les autres parties contenant au moins un joueur humain. Cette IA utilise la base de données de ces parties pour voir quels coups sont optimaux. Elle regarde sur l'ensemble des parties là où l'humain mets le plus souvent ses navires pour attaquer à ces endroits mais aussi là où l'humain attaque le moins souvent pour y placer ses navires. Cette IA est donc au fil du temps sensée devenir l'adversaire le plus difficile mais il est difficile de prédire si cela se déroulera bien de la sorte n'ayant pas vraiment de contrôle sur les humains qui rempliront la base de données au fur et à mesure.

Le problème de cette méthode est donc le changement de comportement de l'humain car s'il décide après avoir joué un grand nombre de parties de changer complètement de stratégie : l'IA se retrouve complètement faussée par les résultats précédents et mettra plus de temps qu'habituellement pour finir la partie.

Lorsque l'IA a épuisé toutes les cases où les personnes posent généralement leurs bateaux, il passe en mode « Difficile », c'est-à-dire qu'il continuera le jeu de la même façon que le niveau difficile.

3 Base de données

3.1 Exemple pour la BDD

MATCH					
N_M	Vainqueur	Durée	Nb_Coups	NIVEAU	Joueur
1	J1	00 :15 :32	51	M	Bob
2	J2	00 :17 :14	79	F	Kévin
3	J2	00 :12 :11	40	D	Marie
4	J1	00 :25 :55	81	A	Antoine
5.1	J1	00 :21 :49	44	Raphaëlle	Louis
5.2	J1	00 :21 :49	44	Louis	Raphaëlle

NAVIRES_J					
N1	N_J	Nom	N_B	B_X	B_Y
1	1	Bob	2	1	2
2	1	Bob	2	2	2
3	1	Bob	31	4	5
4	1	Bob	31	4	6
5	1	Bob	31	4	7
6	1	Bob	32	7	0
7	1	Bob	32	8	0
8	1	Bob	32	9	0
9	1	Bob	4	8	5
10	1	Bob	4	8	6
11	1	Bob	4	8	7
12	1	Bob	4	8	8
13	1	Bob	5	2	3

ATTAQUES_J						
N2	N_J	Nom	N_A	A_X	A_Y	Etat
1	1	Bob	1	0	4	P
2	1	Bob	2	5	2	P
3	1	Bob	3	6	3	P
4	1	Bob	4	8	7	P
5	1	Bob	5	1	6	P
6	1	Bob	6	5	9	P
7	1	Bob	7	8	3	T
8	1	Bob	8	8	4	P
9	1	Bob	9	7	3	T
10	1	Bob	10	6	3	T
11	1	Bob	11	5	3	C
12	1	Bob	12	9	8	P
13	1	Bob	13	9	6	T

La table MATCH nous permet d'avoir les informations générales du match : le numéro du match, le vainqueur, la durée, le nombre de coups, le type (entre deux joueurs : nom de l'autre ou contre une IA : Facile, Moyenne, Difficile, Autodidacte) et le nom du joueur. Lorsqu'on a deux joueurs humains on a deux lignes pour avoir les deux noms.

La table NAVIRES_J répertorie tous les placements de navires des joueurs humains pour ensuite pouvoir faire des statistiques sur les endroits où les humains posent le plus souvent leurs navires. N_B représente le numéro du navire 2, 3 (31 et 32 car il y a deux navires de longueur 3), 4 et 5.

La table ATTAQUES_J répertorie toutes les attaques effectuées par tous les joueurs humains pour ensuite pouvoir faire des statistiques sur les endroits où les humains attaquent le plus souvent. On les répertorie dans l'ordre car l'ordre est important pour savoir quelles cases sont visées en premier car plus on avance et plus les attaques deviennent basées sur les connaissances acquises par les attaques précédentes. C'est aussi pourquoi nous avons la colonne Etat qui permet de savoir si l'attaque a Touché ou Coulé un navire ou s'il a fait Plouf. Ainsi, lorsqu'une attaque

touche un navire, celles qui suivent deviennent moins intéressantes car elles se baseront sur la première jusqu'à couler le navire touché.

3.2 Servir à l'AUTODIDACTE()

Toutes les données stockées dans cette base de données et plus particulièrement dans les tables NAVIRES_J et ATTAQUES_J serviront à faire évoluer l'IA AUTODIDACTE qui se basera entièrement sur elles. Ainsi pour savoir où poser ses navires l'IA s'intéressera aux cases que les joueurs visent le plus souvent et le plus tôt pour ne pas y poser ses navires. Pour ce qui est de l'attaque, l'IA regardera là où les joueurs posent le plus souvent leurs bateaux pour y attaquer en premier.

Ainsi, plus il y aura de parties plus la base de données deviendra importante et plus l'IA AUTODIDACTE aura d'échantillons lui permettant de faire les meilleurs choix pour gagner la partie. Si tout se déroule bien cette IA devrais après un certain temps dépasser l'IA DIFFICILE et pour le vérifier nous pourrions regarder toutes les parties contre chaque IA et voir quelle IA gagne le plus souvent. On ne peut malheureusement pas tester l'IA AUTODIDACTE avec le DUELTEST et le TEMOIN car le témoin pose toujours ses bateaux aléatoirement donc les données reçus des parties humaines n'aideront en rien l'IA à gagner. L'IA AUTODIDACTE() serait entièrement spécialisée pour battre un humain en particulier et non pas d'autres machines ou IA.

3.3 Les requêtes pour la récupération des données

```
select_query2 = '''WITH numbered_rows AS (
  SELECT *,
         ROW_NUMBER() OVER (PARTITION BY N_J ORDER BY N2 ASC) AS row_num
  FROM ATQ_J
),
min_row_nums AS (
  SELECT N_J,
         MIN(row_num) AS min_row_num
  FROM numbered_rows
  WHERE ETAT IN ('T', 'C')
  GROUP BY N_J
)
SELECT N_A, A_X, A_Y, ETAT
FROM numbered_rows
WHERE N_J IN (SELECT N_J FROM min_row_nums)
  AND row_num <= (SELECT min_row_num FROM min_row_nums WHERE min_row_nums.N_J = numbered_rows.N_J);
'''
```

Voici la requête pour récupérer les attaques de chaque joueur jusqu'à ce qu'il touche un navire. On s'arrête au premier touché car ensuite il cherchera juste à couler le navire touché et ses autres attaques seront influencées par ce navire coulé.

Cette requête renvoie donc tous les premiers attaques jusqu'au premier Touché ('T') inclus. La réponse est sous forme d'une liste Pos_Att, par exemple de la forme :

[(1, 0, 4, 'P'), (2, 2, 2, 'P'), (3, 4, 5, 'P'), (4, 8, 9, 'T'), (1, 1, 2, 'P'), (2, 4, 4, 'P'), (3, 6, 5, 'T'), (1, 5, 2, 'P'), (2, 2, 3, 'P'), (3, 4, 5, 'P'), (4, 4, 6, 'P'), (5, 4, 7, 'T')]

Ensuite on l'utilise pour créer une matrice qui représente la grille 10x10 avec des valeurs représentant à quelle point chaque case est attaqué.

```
Pos_Att = []

cursor.execute(select_query2)

rows = cursor.fetchall()

for row in rows:
    Pos_Att.append(row)

PA = np.zeros((10, 10))

for triplet in Pos_Att:
    value = 2 - (triplet[0] - 1) * 0.05
    position = (triplet[1], triplet[2])

    PA[position] += value
```

Ici on a donc transformé la liste Pos_Att en matrice.

On a value = 2 – (triplet[0] – 1) * 0,05 pour faire jouer quand l'attaque a été lancé. Ainsi, la première attaque comptera pour 2 et la septième pour 2 – 0,05*6 = 1,7.

L'IA utilisera alors cette matrice pour poser ses bateaux là où les valeurs sont les plus faibles.

```
select_query = '''
    SELECT B_X, B_Y FROM NAV_J
'''

cursor.execute(select_query)

rows = cursor.fetchall()
Pos_Bat = []

for row in rows:
    Pos_Bat.append(row)

PB = np.zeros((10, 10))
for i in range(10):
    for j in range(10):
        PB[i][j] = Pos_Bat.count((i, j))
```

Cette requête nous donne les positions des navires du joueur que nous mettons dans une liste Pos_Bat. Cette liste contient toutes les positions de tous les humains ayant joué. Ensuite, on transforme cette liste en matrice (grille) où chaque case possède le nombre de navires qui ont été posés dessus. Ainsi, l'IA attaquera d'abord les cases avec la valeur la plus élevée.

Conclusion

Nous avons déjà codé une bonne partie du jeu mais il faut encore s'attaquer à la partie la plus complexe c'est-à-dire l'IA AUTODIDACTE et sa base de données. Au moins durant cette première étape et notamment avec la rédaction de ce document nous avons une bonne idée de ce qu'il reste à faire et comment s'y prendre. Les objectifs de la seconde étape seront donc de réussir à aller au bout de nos idées et de réussir à créer une cette IA évolutive appelée AUTODIDACTE.

Figures réalisées :

- Factorisation du code : au moins trois modules et noms de classes distincts
- Création d'au moins trois types d'objet (classe) : ils devront contenir plusieurs variables d'instance (eg. Véhicule, Voiture, Vélo, Roue)
- Héritage depuis un type intégré (hors IHM)
- Documentation et commentaires du code
- Figure personnalisé : créer une classe évolutive (AUTODIDACTE)
- Ecriture de fichier : compte rendu partie
- BDD