LA BATAILLE NAVALE

Sujet : **La Bataille Navale**

Élèves : **Alyson Fontaine, Emilien Garnier, Owen Godin, Bulat Kuzyaev, Emilien Spiquel - - Bandin, Camille Vignau**

Enseignants : **Monsieur Créchet, Madame Herminier, Monsieur Pelletier**

Enseignant-Chercheur : **Monsieur Benjamin Nguyen**

Etablissement : **Lycée Marguerite de Navarre**

Nous avons choisi de travailler sur un sujet en suivant la base d'un jeu appelé la bataille navale. A l'origine, ce jeu consistait à jouer à deux. Chacun des deux joueurs devait placer cinq bateaux, puis il fallait trouver les bateaux placés par l'adversaire. Nous nous sommes intéressés à la meilleure stratégie pour trouver au mieux les bateaux. Nous avons défini nos propres règles, on place un seul bateau, lorsque ce bateau est touché, il est automatiquement coulé. Pour ce faire, nous nous sommes réparti le travail en trois groupes :

- **La stratégie optimale**
- **L'application de la stratégie**
- **Comparaison de la stratégie avec l'aléatoire**

Afin de nous aider à développer notre idée et d'avancer dans nos recherches, nous avons reçu l'aide de Monsieur Nguyen, un professeur de l'INSA Centre Val de Loire, en plus de nos professeurs encadrants.

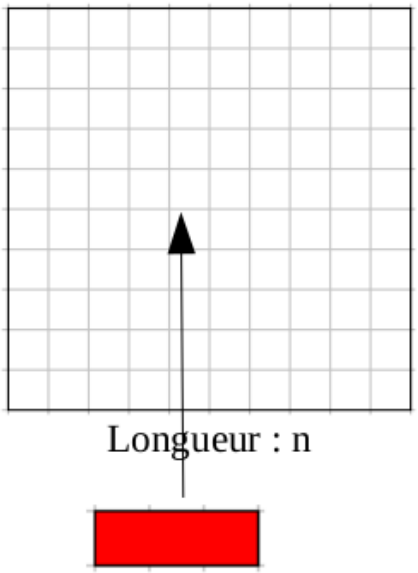
Dans un premier temps, nous verrons la stratégie optimale, celle-ci consiste à placer un minimum de mines sur l'ensemble d'une grille définie afin de trouver le bateau avec le moins de coup possible.

Dans un second temps, nous expliquerons notre application de la stratégie précédente selon la taille de la grille. Une fois que nous avions trouvé la stratégie optimale en diagonale, nous devions générer un tableau contenant les coordonnées des mines à toucher sur Python. Le but étant de pouvoir procéder par la suite à des simulations statistiques pour prouver l'efficacité de notre stratégie.

La stratégie optimale

1. Détermination du nombre minimal théorique de mine

Il s'agit tout d'abord de trouver le nombre minimal théorique de mines requises pour empêcher tout bateau d'être posé.



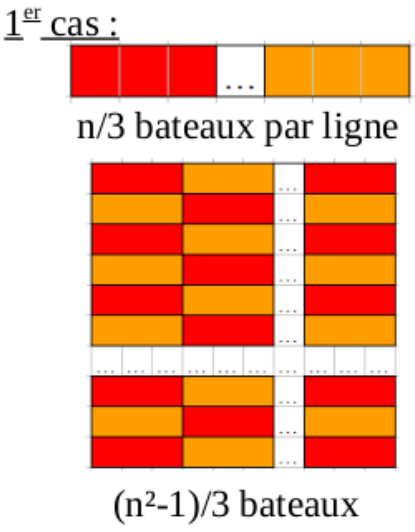
Démarche :

On va tout d'abord trouver le pavage optimale du quadrillage avec des bateaux de 3 sur 1. Puis, on déduira le nombre M de mines suffisantes pour couler le bateau. Et comme dans toute configuration, on ne pourra mettre plus de M bateaux, on aura ainsi trouvé le nombre minimal théorique de mines.

On note n la longueur d'un côté du quadrillage. Les bateaux que l'on peut placer auront toujours la même taille, trois par un. On distinguera trois cas : dans le premier, n est divisible par trois, dans le second, il reste un à la division euclidienne de n par trois et dans le dernier cas il reste deux.

Dans le premier cas où n est divisible par trois, il ne reste aucune case après avoir placé un maximum de bateaux sur la grille. Sur n lignes, nous avons donc placé $\frac{n}{3}$ bateaux.

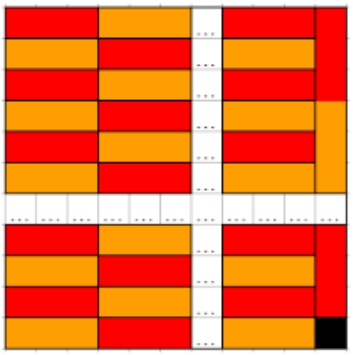
Le nombre total de bateaux est $n \times \frac{n}{3} = \frac{n^2}{3}$.



Dans le deuxième cas, nous avons pu placer $\frac{n-1}{3}$ bateaux par ligne sur n lignes. Il reste une colonne à droite que nous pouvons remplir par $\frac{n-1}{3}$ bateaux.

2nd cas :

(n-1)/3 bateaux par ligne



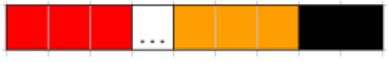
(n²-1)/3 bateaux

Le nombre de bateaux est :

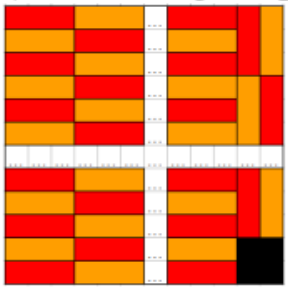
$$n \times \frac{n-1}{3} + \frac{n-1}{3} = \frac{n^2-n}{3} + \frac{n-1}{3} = \frac{n^2-1}{3} .$$

Dans le dernier cas, en utilisant la même technique, nous remplissons n lignes par $\frac{n-2}{3}$ bateaux et les deux dernières colonnes par $\frac{n-2}{3}$. Il reste quatre cases noires en bas à droite de la grille.

3ème cas :



(n-2)/3 bateaux par ligne

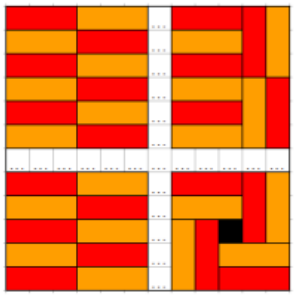


En pivotant un carré de trois bateaux en bas à droite, nous pouvons en pivoter un et rajouter un bateau.

Nous avons donc placé :

$$n \times \frac{n-2}{3} + 2 \times \frac{n-2}{3} + 1 = \frac{n^2-2n}{3} + \frac{2n-4}{3} + \frac{3}{3} = \frac{n^2-1}{3} .$$

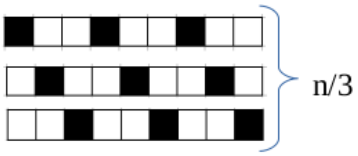
Une mine est donc nécessaire pour chaque bateau.



n²/3 bateaux

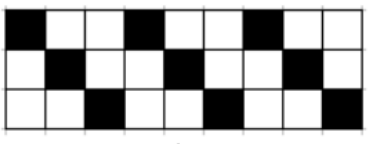
Conclusion : Le minimum théorique de mines est donc $\frac{n^2}{3}$ pour le premier cas et $\frac{n^2-1}{3}$ pour les deux autres cas.

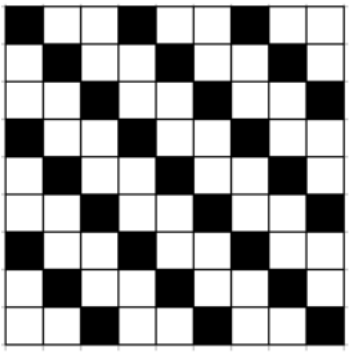
2. La stratégie en damier



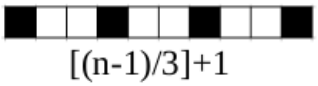
Nous souhaitons ensuite démontrer que poser les mines en diagonale permet d'atteindre ce minimum théorique.

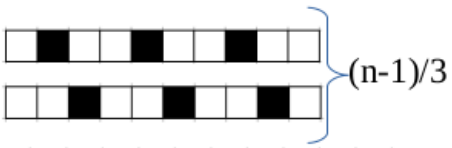
Dans le premier cas, n est divisible par trois. Comme pour chaque ligne nous posons une mine toutes les trois cases, nous avons besoin de $\frac{n}{3}$ mines par lignes. Si nous découpons ensuite le quadrillage par portions de trois lignes, nous utilisons n mines par portions. Il y a $\frac{n}{3}$ portions pour remplir le quadrillage. Le nombre de mines utilisées est donc n x $\frac{n}{3}$ mines soit $\frac{n^2}{3}$. Le nombre minimum théorique est atteint, la stratégie est parfaite pour tout nombre n divisible par trois.

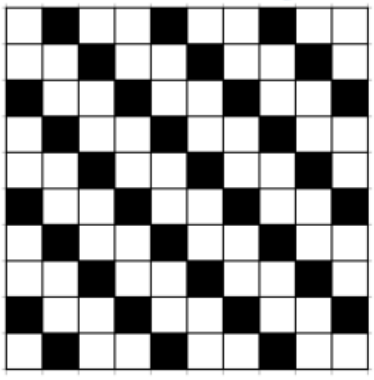




Ensuite, il reste un à la division euclidienne de n par 3. Les lignes où la première mine est sur la première case, le nombre de mines sera $\frac{n-1}{3}$ +1. Le nombre de mines d'un bloc de trois lignes sera $\frac{3(n-1)}{3}$ +1 = n. Il faut placer $\frac{n-1}{3}$ blocs de trois lignes. Il reste une ligne où aucune mine n'est encore présente. Il faut donc rajouter une ligne dont la première mine est sur la deuxième ou troisième case pour garder un nombre minimal de mines.





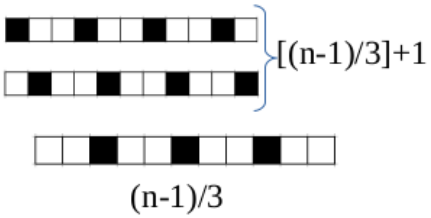


En effet, la ligne de mines où la première mine est sur la première case contient une mine de plus que les deux autres lignes. Le nombre total de mine sera donc :

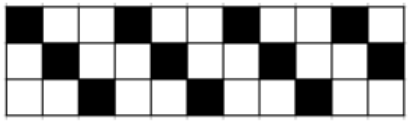
$$n \times \frac{n-1}{3} + \frac{n-1}{3} = \frac{n^2-n+n-1}{3} = \frac{n^2-1}{3} .$$

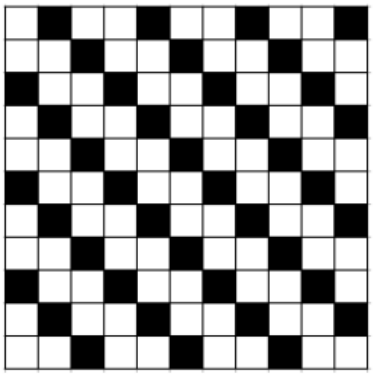
Le nombre minimum théorique est atteint, la stratégie est donc également parfaite.

Pour la configuration où la division euclidienne de n par trois a un reste de deux, la démarche est la même sauf qu'il y a deux lignes qui ont $\frac{n-2}{3}$ +1 mines par ligne et une seule qui n'en a que $\frac{n-2}{3}$ mines. Une des deux lignes restantes après la pose des blocs de trois lignes doit comporter $\frac{n-2}{3}$ pour que le nombre de mines reste minimal. Le nombre total de mines est donc :



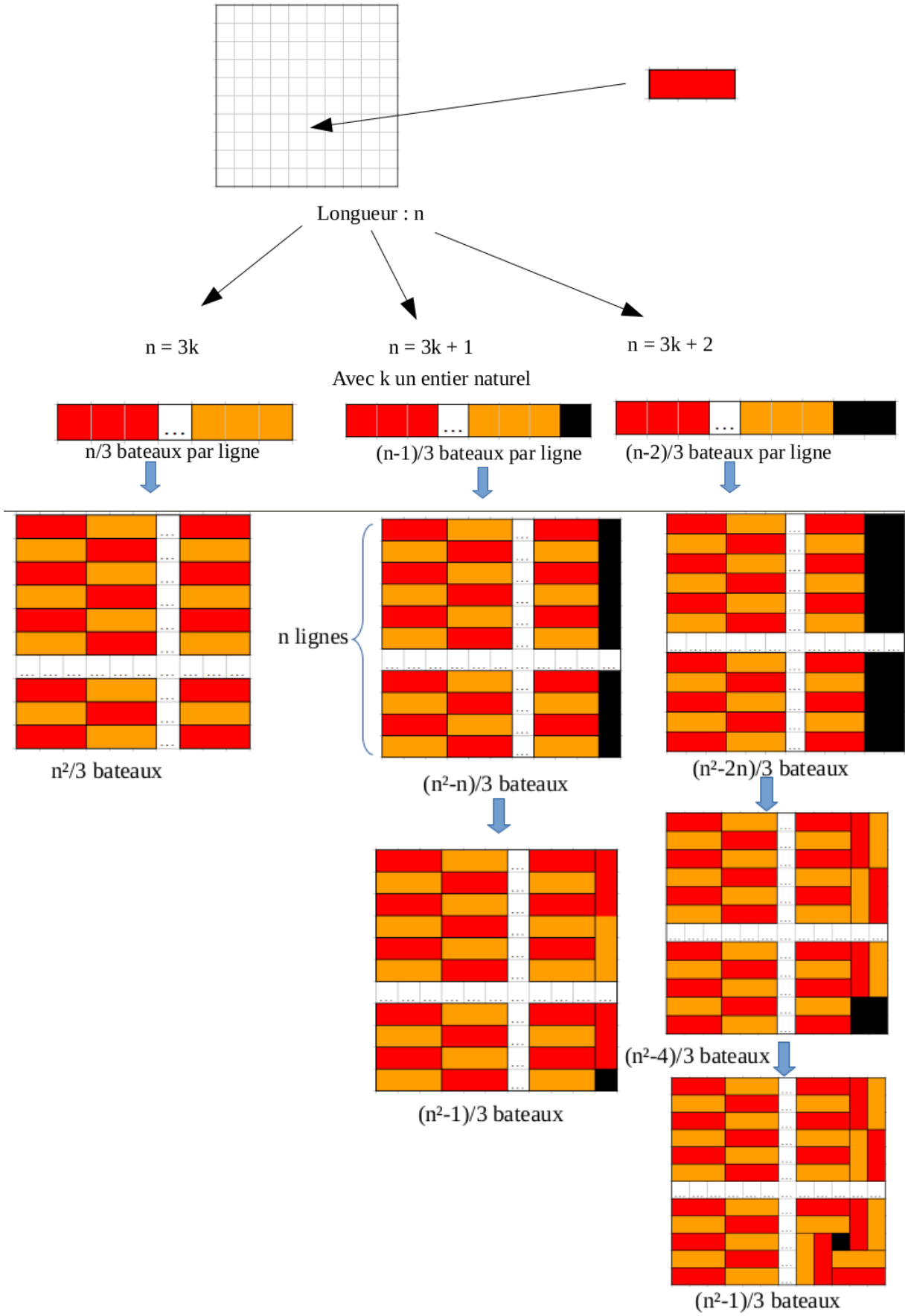
(n-1)/3

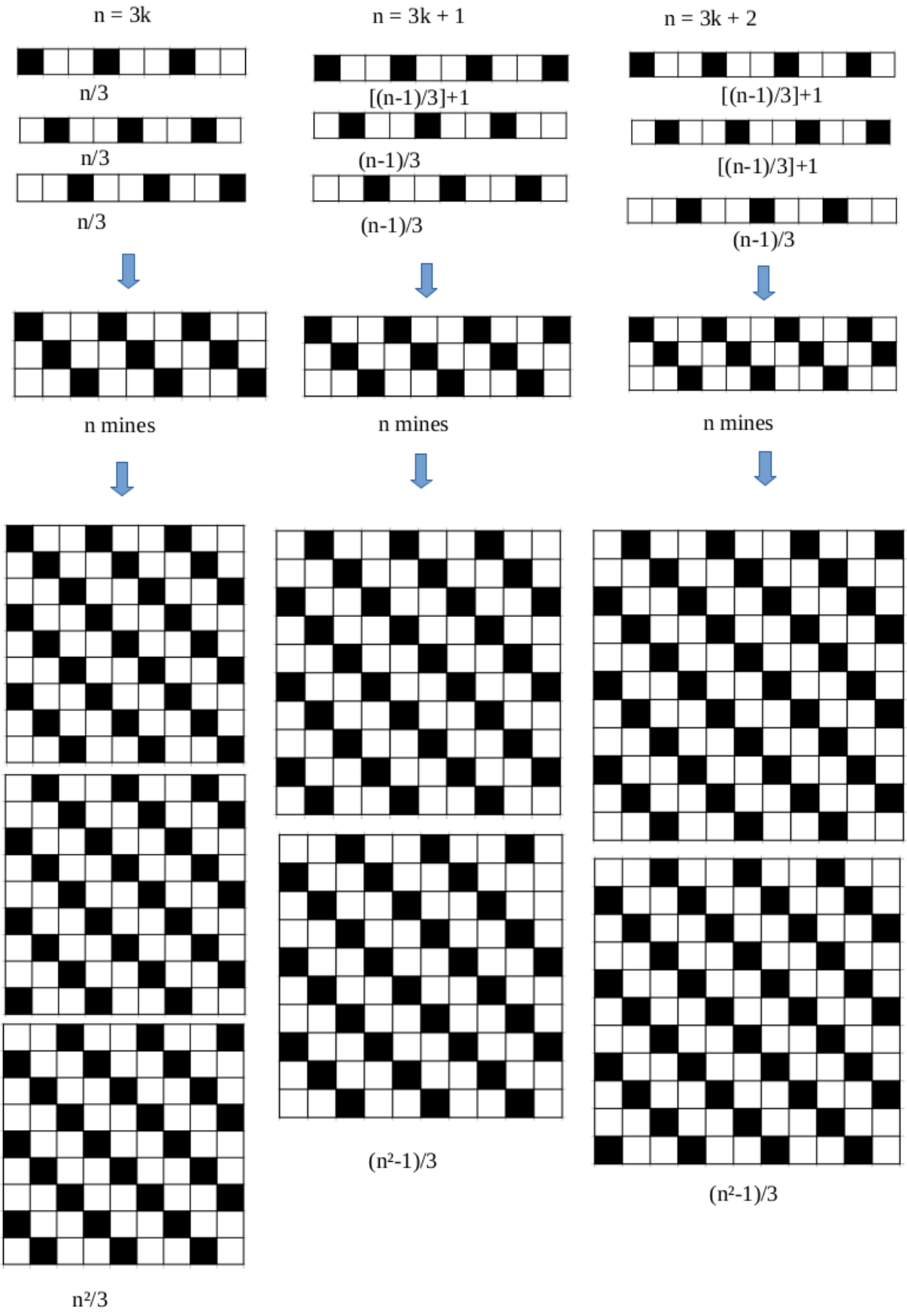

n mines


(n²-1)/3

$$n \times \frac{n-2}{3} + \frac{n-2}{3} + \frac{n-2}{3} + 1 = \frac{n^2-2n+2n-4}{3} + \frac{3}{3} = \frac{n^2-1}{3} .$$

La stratégie qui consiste à placer des mines en diagonale est donc parfaite car elle atteint le nombre minimal théorique de mines à placer pour qu'aucun des bateaux ne puisse être posé sur la grille.


Longueur : n
n = 3k
n = 3k + 1
n = 3k + 2
Avec k un entier naturel
n/3 bateaux par ligne
(n-1)/3 bateaux par ligne
(n-2)/3 bateaux par ligne
n lignes
n²/3 bateaux
(n²-n)/3 bateaux
(n²-2n)/3 bateaux
(n²-4)/3 bateaux
(n²-1)/3 bateaux
(n²-1)/3 bateaux


n = 3k
n/3
n/3
n/3
n mines
n²/3
n = 3k + 1
[(n-1)/3]+1
(n-1)/3
(n-1)/3
n mines
(n²-1)/3
n = 3k + 2
[(n-1)/3]+1
[(n-1)/3]+1
(n-1)/3
n mines
(n²-1)/3

L'application de la stratégie

Étape 1 :

Pour cela, nous avons créé une liste répertoriant les coordonnées de tous les emplacements à toucher pour être certain de couler le bateau ennemi en fonction de trois variables :

N = La longueur du quadrillage
n = La largeur du quadrillage
k = La taille du bateau.

Seules les coordonnées suivant la technique de la stratégie en damier seront répertoriées afin de minimiser le nombre de cases à toucher.

Pour créer les listes de coordonnées sur Python, il nous a fallu répertorier une à une ces coordonnées de façon logique afin de pouvoir établir des listes pour des tailles de quadrillage et de bateau souhaitées.

Pour cela :

→ Il faut premièrement entrer dans la liste les coordonnées des croix de la colonne de gauche (toutes les $k^{èmes}$ cases).

→ Pour chacune d'entre elles, il faut ensuite entrer les coordonnées des croix montantes en diagonales.

→ Puis entrer les coordonnées des croix de la ligne du bas (toutes les $k^{èmes}$ cases).

→ Pour chacune d'entre elles, il faut encore entrer les coordonnées des croix montantes en diagonales.

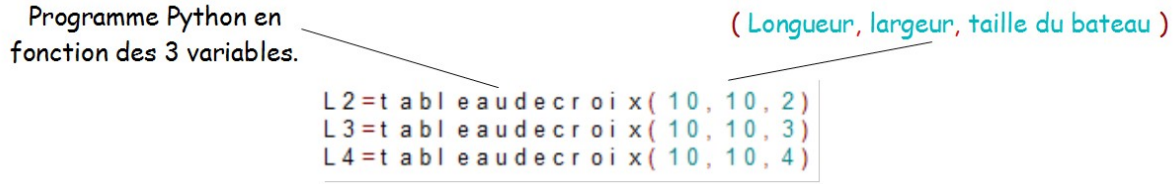
```
from lycee import *

def tableaucroix(n,N,k):        # n hauteur  N largeur et k taille du bateau
    r=n  k
    L3=[]
    q=n//k
    for i in range (q):
        L3.append([(i+1)*k,1])   #croix de la 1ère colonne
        x=(i+1)*k
        y=1
        while x>1 and y<N:
            L3.append([x-1,y+1])    # les autres croix en diagonale
            x=x-1
            y=y+1
    p=floor((N-k+r-1)/k))
    for i in range (p+1):
        L3.append([n,(k-r+1+i*k)]) #1ère croix du bas dernière ligne
        x=n
        y=k-r+1+i*k
        while x>1 and y<N:
            L3.append([x-1,y+1])    # les autres croix en diagonale
            x=x-1
            y=y+1
    return L3
```

Notre liste comprend ainsi pour un bateau de longueur k les coordonnées de tous les emplacements à miner pour être certain de toucher ce bateau, en fonction de la taille du quadrillage utilisé.

Étape 2 :
Nous n'avons plus qu'à entrer les données (longueur, largeur du quadrillage et taille du bateau) de notre choix afin de créer des liste de coordonnées d'emplacements pour un quadrillage et un bateau précis.

Nous avons décidé de prendre pour exemple un quadrillage de 10x10 avec des bateaux de longueur 2, 3 et 4.



Nous avons ainsi trois listes de coordonnées, pour des bateaux de 2, 3 et 4 de longueur car il s'agit de nos exemples (maintenant que le programme « tableaudecroix » est établi, nous pourrions créer d'autres listes pour des tailles de variables différentes si besoin).

Grâce à ces listes, nous allons pouvoir simuler des parties afin de prouver la supériorité de notre stratégie par rapport à une stratégie aléatoire.

Comparaison de la stratégie avec l'aléatoire

Enfin dans une dernière partie, nous avons comparé la stratégie développée jusqu'à présent avec une stratégie aléatoire.

Afin de vous démontrer que cette stratégie en damier est donc la meilleure solution entre aléatoire et agir de manière réfléchie.

Création de grille :

Au début de notre programme, nous avons créé une grille carrée de 10 carreaux de côté, représentant le terrain dans lequel évoluera le combat. Mais cette taille peut être changée selon les envies de l'utilisateur. Pour faire cela l'utilisateur devra changer la valeur de n, qui est ici de 10, afin d'avoir un terrain selon sa bonne envie. Ainsi son terrain peut-être de 5 fois 5 tout comme de 15 fois 15.

```
def creation_grille(self):
    n=10
    for i in range(n):
        ligne = []
        for j in range(n):
            ligne += [" "]
        self.quadrillage += [ligne]
```

Ainsi la grille est créée.

Placement bateau :

Après nous avons défini le placement du bateau dans la grille.

```
def  place_bateau(self,taille):
     orientation = ["vertical","horizontal"]
     o=orientation[randint(0,1)]

     if  o=="vertical":
          (x,y)=(randint(0,9-(taille-1)),randint(0,9))

          for  i  in range(taille):
               self.quadrillage[y+i][x]="b"
               self.bateaux+=[[y+i,x]]
     else:
          (y,x)=(randtin(0,9),randint(0,9-(taille-1)))

          for  i  in range(taille):
               self.qadrillage[y][x+i]="b"
               self.bateaux=+[[y,x+i]]
```

Les bateaux sont mis aléatoirement dans la grille créée avant.

Élaboration de stratégie :

```
def jouer_alea(self):
    coords=choice(self.liste)

    self.liste.remove(coords)
    self.mine(coords[0]-1,coords[1]-1)
```

Afin de comparer nos deux stratégies, nous avons créé un programme qui confronte la stratégie aléatoire et la stratégie en diagonale.

En fait, la stratégie aléatoire est légèrement réfléchie puisque quand l'ordinateur a joué une coordonnée de la grille, il ne peut pas rejouer à nouveau cette coordonnée. Cela évite qu'un ordinateur joue plusieurs fois la même coordonnée.

Mine :

```
def mine(self,x,y):
    if self.quadrillage[y][x]=="b":

        self.quadrillage[y][x]="t"

        self.bateaux.remove([y,x])
        self.fin_partie=True

    else:

        self.quadrillage[y][x]="r"
```

Mais afin de pouvoir vérifier quelle est la meilleure stratégie, nous avons créé des mines qui seront placées aléatoirement par l'ordinateur dans la grille. Il ne peut y avoir qu'une mine par case seulement. La partie est terminée lorsque le bateau est touché car un bateau coule directement lorsqu'une mine le touche.

Moyenne de mines placées + résultat :

Ainsi notre dernière partie du programme, joue 10 000 parties à la suite. Il retient le nombre de coups moyens que l'ordinateur aura joué et nous l'affiche.

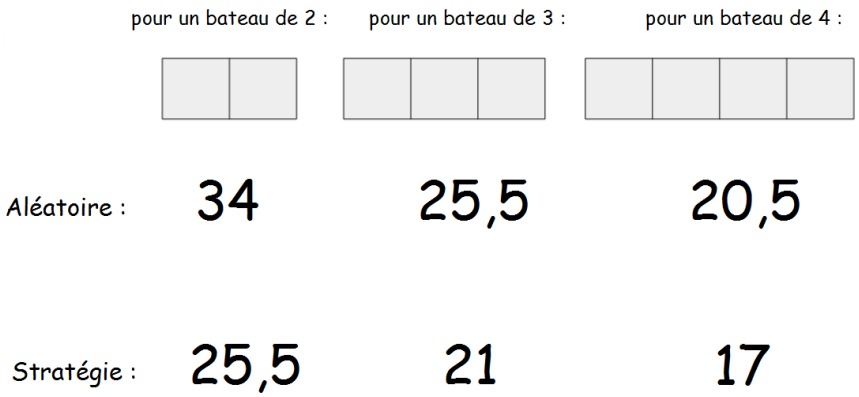
Simulation :

La mise en forme de notre simulation est simple et lisible, pour que tout le monde puisse la lancer.

Cette simulation permet de constater, grâce à l'affichage les écarts de coups entre notre stratégie aléatoire et notre stratégie en diagonales.

Conclusion de nos recherches

Nos résultats sont concluants car on peut voir que pour 10000 parties jouées, il faut en moyenne 34 coups pour trouver un bateau de 2 posé aléatoirement sur un quadrillage de 10x10 en suivant la stratégie aléatoire alors qu'il en faut moins de 26 avec notre stratégie. Et l'on retrouve cette supériorité en terme d'efficacité avec n'importe quelle taille de bateau.



Nous remarquons cependant que plus la taille du bateau se rapproche de la taille du quadrillage, moins notre stratégie est efficace comparée à l'aléatoire, sans pour autant devenir moins performante.

Pour améliorer nos recherches, nous aimerions par la suite être capable de placer plusieurs bateaux sur la grille tout en conservant une stratégie plus efficace que l'aléatoire. Pour cela, il nous faudrait combiner les différentes listes dans un certain ordre en fonction des bateaux et cases touchées au fur et à mesure du jeu. Plus le nombre de bateaux de tailles différentes sera élevé, plus le programme sera long et compliqué.
De plus, certaines tailles de bateaux sont plus difficiles à combiner que d'autres, en fonction de leurs multiples communs par exemple (2, 4 et 8 sont plus simples à combiner que 2, 3 et 5 par exemple).