
# Rapport du Projet : Bataille Navale et Recherche d'Objets

LU3IN005

Yuxiang ZHANG

Sam ASLO

**Rapport du Projet : Bataille Navale et Recherche d'Objets**


- Introduction
- Partie 1 : Modélisation de la Grille et Placement des Bateaux
  - Modélisation de la Grille
  - Expériences de Placement des Bateaux
  - Analyse
  - Dénombrement des Configurations de Grilles
    - Q1. Borne Supérieure du Nombre de Configurations
    - Q2 et Q3. Calcul exact du Nombre de Configurations
    - Q4. Estimation de la Probabilité d'une Grille Donnée
    - Q5. Algorithme d'Approximation
    - Q6. Méthodes d'Amélioration
  - Conclusion
- Partie 3 : Modélisation Probabiliste du Jeu
  - 1. Version aléatoire
    - Hypothèses :
    - Simulation
    - Résultats
    - Analyse
  - 2.Version Heuristique
    - Comportement :
    - Simulation
    - Résultats
    - Analyse
  - 3. Version Probabiliste Simplifiée
    - Simulation
    - Résultats
    - Analyse
  - 4. Version Monte Carlo
    - Simulation
    - Résultats
    - Analyse
  - Conclusion
- Partie 4 : Recherche d'Objets par Approche Bayésienne
  - Modélisation et Algorithmes
    - Q1. Loi de $Y_i$ et loi de $Z_i | Y_i$?
    - Q2. Probabilité de $Z_k=0$ sachant $Y_k=1$?

Q3. Mise à jour de la probabilité k
Q4. Mise à jour de i pour ik
Méthodologie
Expériences
Analyse des Résultats
Conclusion
Conclusion Générale

# Introduction

Ce projet vise à étudier la bataille navale d'un point de vue probabiliste. L'objectif est de modéliser le jeu pour optimiser les chances de gagner en explorant plusieurs approches : d'abord, une analyse combinatoire pour estimer le nombre de configurations possibles sur la grille, puis la simulation de différentes stratégies de tir, allant de l'aléatoire à des approches plus avancées.

Nous explorons également des méthodes probabilistes pour ajuster les chances de toucher un bateau en fonction des coups précédents. Enfin, nous appliquons un algorithme bayésien pour la recherche d'objets perdus, inspiré d'un exemple réel, le sous-marin USS Scorpion.

« Ce rapport est divisé en deux grandes sections : la modélisation du jeu de bataille navale et l'exploration bayésienne de la recherche d'objets. »

# Partie 1 : Modélisation de la Grille et Placement des Bateaux

## Modélisation de la Grille

Nous avons débuté par la création d'une grille de jeu de 10 par 10 pour simuler un jeu de bataille navale. Chaque case de la grille est modélisée comme une matrice d'entiers, où chaque bateau est représenté par un identifiant unique. Nous avons utilisé le module **NumPy** pour simplifier les opérations sur la grille et manipuler les matrices.

Les principales classes et fonctions implémentées sont :

**Bateau** et **Grille** : pour la modélisation des bateaux et de la grille.

**Fonctions :**

peut_placer(grille, bateau, position, direction) : vérifie si un bateau peut être placé à une position donnée.

place(grille, bateau, position, direction) : place un bateau sur la grille.

place_alea(grille, bateau) : place un bateau aléatoirement.

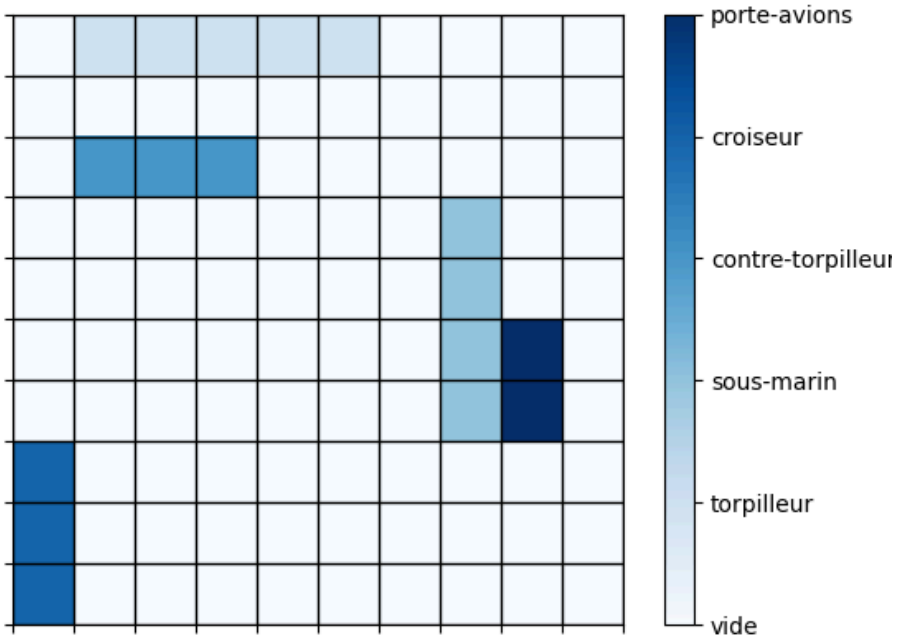
affiche(grille) : affiche l'état de la grille.

eq(grilleA, grilleB) : compare deux grilles pour vérifier si elles sont identiques.

genere_grille() : génère une grille complète avec tous les bateaux placés aléatoirement.

## Expériences de Placement des Bateaux

Nous avons réalisé plusieurs simulations pour tester les fonctions ci-dessus. L'une de nos premières réussites a été l'affichage correct de la grille avec les bateaux placés, comme illustré dans la figure ci-dessous (affiche_grille_bateau.png):




## Analyse

Le modèle implémenté fonctionne bien pour des grilles de taille standard (10×10), mais nous avons identifié que la complexité du placement aléatoire peut devenir un défi lorsqu'on augmente la taille de la grille ou le nombre de bateaux. Les prochaines sections abordent cette problématique avec une approche plus formelle du dénombrement des configurations possibles.

# Partie 2 : Exploration Combinatoire des Configurations de Grilles (Tous les fonctions suivants nous avons écrit dans 'functions.py')

## Dénombrement des Configurations de Grilles

### Q1. Borne Supérieure du Nombre de Configurations

Nous avons calculé une borne supérieure du nombre de configurations possibles pour le placement des bateaux sur une grille de 10×10 en supposant des placements uniquement horizontaux ou verticaux. Pour chaque bateau de taille n, le nombre total de placements possibles est donné par : **Total = 10×(10−n+1)×2**

Puisque un bateau de taille n, peut être placé soit horizontalement soit verticalement.

Position horizontale : Il peut être placé dans 10 lignes, et chaque ligne peut accueillir **10−n+1**.

Position verticale : Il peut être placé dans 10 colonnes, et chaque colonne peut accueillir **10−n+1**.

Donc pour un bateau de taille n, le nombre total de façons de le placer est : **10×(10−n+1)×2**

Pour la liste complète de bateaux, une estimation simple est réalisée en multipliant le nombre de façons de placer chaque bateau:

Ntotal=(10×(10−2+1)×2)×(10×(10−3+1)×2)×(10×(10−3+1)×2)×(10×(10−4+1)×2)×(10×(10−5+1)×2)=**77,414,400,000 configurations**.

Ce résultat est une estimation brute et ne tient pas compte des contraintes supplémentaires telles que les chevauchements de bateaux.

### Q2 et Q3. Calcul exact du Nombre de Configurations

Nous avons implémenté une fonction nb_placer() pour calculer de manière exacte le nombre de placements possibles d'un bateau donné sur une grille vide. Ensuite, la fonction nb_total_placer() permet de calculer le nombre total de configurations possibles en tenant compte de plusieurs bateaux. Comme attendu, le résultat a confirmé la borne supérieure calculée précédemment

### Discussion

Bien que cette approche fournisse une estimation rapide, elle devient rapidement impraticable à mesure que la taille de la grille ou le nombre de bateaux augmente. Le calcul combinatoire est limité par l'explosion exponentielle des possibilités.

### Q4. Estimation de la Probabilité d'une Grille Donnée

Le lien entre le nombre total de grilles possibles et la probabilité de tirer une grille donnée est que, dans un ensemble d'équiprobabilité, chaque configuration de grille a une probabilité équivalente. Si nous générons aléatoirement des grilles jusqu'à ce qu'une grille donnée soit atteinte, le nombre de tentatives peut être utilisé pour estimer la probabilité

Nous avons réalisé cette question avec la fonction nb_grilles(grille: Grille)

**Q5. Algorithme d'Approximation**

Nous avons développé un algorithme récursif pour approximer le nombre total de grilles possibles, en simplifiant le problème en limitant le placement des bateaux à une seule direction à la fois. L'algorithme est relativement efficace pour des grilles de petite taille, mais ses performances se dégradent rapidement à mesure que la complexité augmente:

1. Condition de terminaison: Si tous les bateaux ont été placés (`index == len(bateaux)`), l'algorithme retourne 1, indiquant une configuration valide.

2. Le bateau actuel à placer est récupéré de la liste des bateaux à l'index donné.

3. On parcourt toutes les positions possibles `(x, y)` sur la grille.

4.Pour chaque position, on tente de placer le bateau. Si le placement réussi :

- On appelle la fonction récursivement pour placer le bateau suivant.
- Le bateau est ensuite retiré de la grille.

Cet algorithme suppose que tous les bateaux peuvent être placés dans la même direction, ce qui simplifie les calculs.

Nous calculons donc toutes les différentes configurations dans une direction puis on renvoi le nombre multiplié par deux

Le problème avec cet algorithme est qu'il ne prend pas en compte le placement des bateaux dans les deux directions.

Et si la taille du grill ou la liste des bateaux est grande, les calculs deviennent impossibles.

Nous avons réalisé cet algorithme avec deux fonctions remove(grille: Grille, bateau: Bateau)et count_configs(grille: Grille, bateaux: list[Bateau], index: int = 0)

**Q6. Méthodes d'Amélioration**

Nous avons proposé plusieurs méthodes pour améliorer ces approximations :

**Échantillonnage stratifié** : cette méthode consiste à diviser la grille en plusieurs zones et à échantillonner des configurations au sein de chaque zone.

**Algorithmes génétiques** : nous pouvons également envisager des algorithmes d'optimisation basés sur des principes évolutionnaires pour explorer l'espace des solutions.

**Modèles probabilistes** : un modèle basé sur les probabilités conditionnelles pourrait fournir une meilleure estimation des configurations possibles.

### Conclusion pour la partie 2

Le dénombrement exact des configurations possibles reste un défi combinatoire majeur. Néanmoins, nos algorithmes et estimations offrent une bonne approximation du problème dans des contextes contrôlés (grilles de petite taille, peu de bateaux).

## Partie 3 : Modélisation Probabiliste du Jeu

Dans cette partie, nous nous intéressons à la modélisation probabiliste du jeu de bataille navale, avec comme objectif principal l'analyse du nombre de coups nécessaires pour couler tous les bateaux d'une grille de taille donnée. Nous avons implémenté plusieurs versions de joueurs (aléatoire, heuristique et probabiliste) afin de simuler différentes stratégies.

Les simulations permettent de calculer l'espérance du nombre de coups joués avant de couler tous les bateaux et d'analyser la distribution de cette variable aléatoire.

### Calcul Théorique

Avant de commencer à faire les modélisations, nous allons calculer la valeur théorique du nombre de coups pour toucher les 17 Cases.

Considérons une grille de taille 10x10, soit 100 cases. Cinq bateaux de tailles 5, 4, 3, 3 et 2 sont placés de manière aléatoire dans ces cases, occupant ainsi 17 cases. Les tirs sont effectués de manière aléatoire et chaque tir est unique, c'est-à-dire qu'aucune case ne peut être visée plus d'une fois.

1. **Probabilité initiale** : La probabilité de toucher une case occupée au début du jeu est
$$p(t_1) = \frac{17}{100}$$
2. **Espérance initiale** : L'espérance du nombre de coups pour toucher une case est
$$E(t_1) = \frac{1}{p(t_1)} = 5.88$$

Supposons qu'après avoir pris 5.88 coups, une case occupée est touchée. La nouvelle probabilité de toucher une case occupée devient

$$p(t_2) = \frac{16}{100 - 5.88}$$

L'espérance correspondante est

$$E(t_2) = \frac{1}{p(t_2)}$$

En continuant ce processus jusqu'à ce qu'il ne reste plus de cases occupées, et en calculant la somme des espérances, on trouve que le nombre total de coups nécessaires est approximativement égal à 94.11, **soit 94 coups**. Et maintenant nous allons commencer à faire les modélisations avec les méthodes "aléatoire, heuristique et probabilité simplifiée".

## 1. Version aléatoire

La version aléatoire repose sur une hypothèse simple : chaque coup est tiré de manière entièrement aléatoire, sans tenir compte des résultats précédents. Voici les hypothèses formulées pour ce modèle :

**Hypothèses :**

1. La grille est de taille fixe (généralement 10×10).
2. Les bateaux sont placés aléatoirement sur la grille.
3. Les coups sont tirés aléatoirement, avec une élimination des positions déjà attaquées pour éviter les répétitions.

La fonction clé dans cette version est jouer_un_coup() qui choisit aléatoirement une position non encore attaquée.

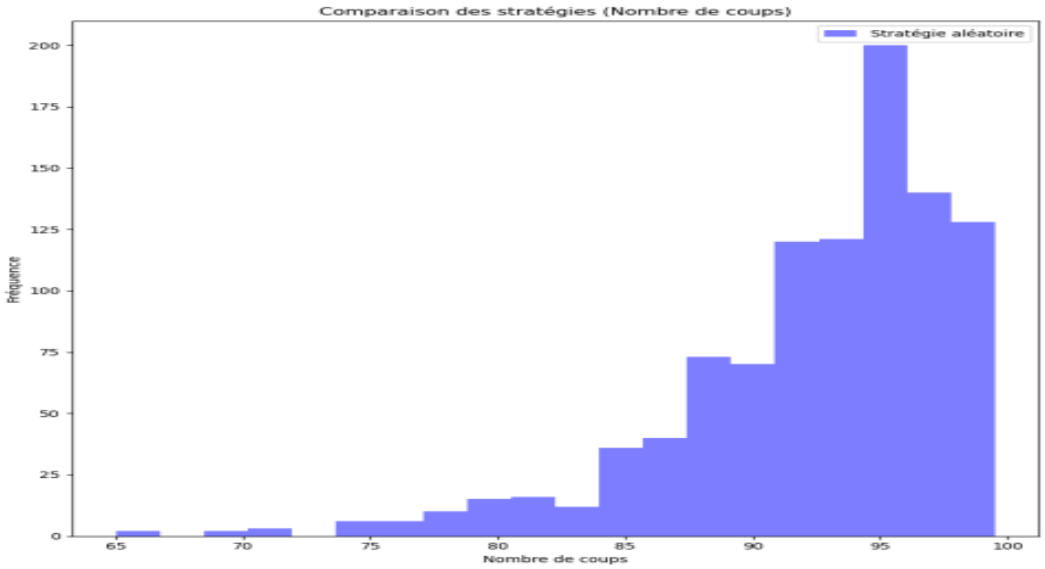
**Simulation**

Nous avons simulé plusieurs parties avec des placements aléatoires des bateaux et des tirs aléatoires. Pour chaque partie, nous comptons le nombre de coups nécessaires pour couler tous les bateaux.

**Résultats**

**Espérance théorique** : Pour une grille de 10×10 (100 cases) avec un total de 17 cases occupées par des bateaux, l'espérance du nombre de coups est **92 tirs** approximée en considérant que les tirs sont indépendants.

**Distribution des coups** : Nous avons calculé la distribution des coups et l'avons comparée à l'espérance théorique. Le graphique suivant montre cette distribution (voir figure 1 distribution aléatoire).



**Analyse**

La version aléatoire présente un comportement attendu, avec une espérance du nombre de coups relativement élevée, due au fait que l'algorithme ne prend pas en compte les coups victorieux précédents. Ce modèle est sous-optimal mais fournit une base de comparaison pour les stratégies plus avancées.

## 2.Version Heuristique

Pour améliorer la version aléatoire, nous avons implémenté une version heuristique qui exploite les informations des coups victorieux. Lorsque l'on touche un bateau, l'algorithme explore les cases adjacentes pour maximiser les chances de toucher de nouveau.

**Comportement :**

Si aucun bateau n'est touché, les tirs sont effectués aléatoirement.

Si un bateau est touché, les cases adjacentes sont explorées en priorité pour détecter le reste du bateau.

**Simulation**

Dans cette version, le joueur garde une liste des positions adjacentes à celles déjà touchées, et choisit ses prochains tirs en fonction de cette information.

**Résultats**

**Espérance** : Comme attendu, l'espérance du nombre de coups est significativement réduite par rapport à la version aléatoire. L'espérance du nombre de coups est **60 tirs** approximée

**Distribution** : La distribution montre une plus grande concentration autour de valeurs plus faibles, confirmant que cette stratégie est plus efficace (voir figure 2 distribution_heuristique).

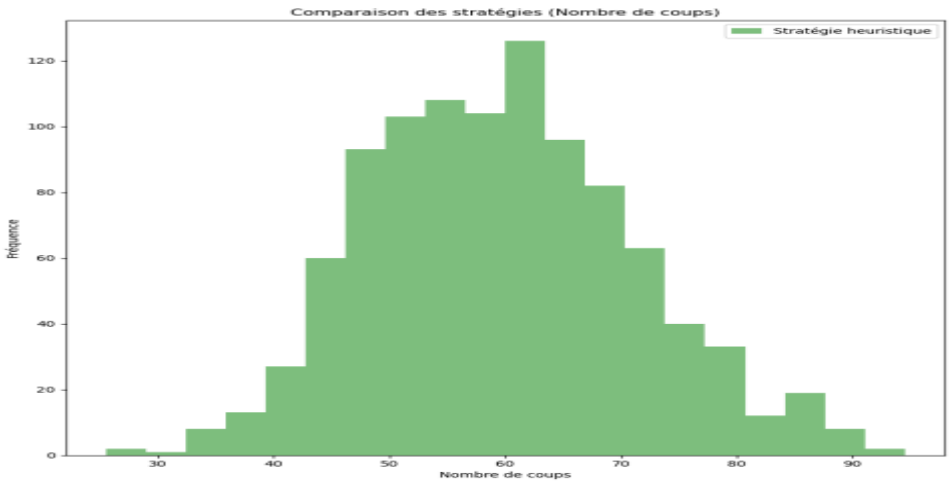



Figure 2: distribution heuristique

**Analyse**

L’approche heuristique apporte une nette amélioration en termes de coups nécessaires pour couler tous les bateaux. Cette stratégie montre qu’en exploitant les informations disponibles, on peut réduire drastiquement le nombre de coups joués.

## 3. Version Probabiliste Simplifiée

Pour chaque case, nous calculons la probabilité d’existence d’un bateau. Afin de simplifier le calcul, nous considérons que tous les bateaux sont indépendants (deux bateaux peuvent exister dans la même case), bien que cela ne soit pas réaliste, car cela simplifie grandement les calculs. Lorsqu’une case est touchée, nous arrêtons de calculer la probabilité et adoptons une stratégie heuristique pour attaquer les cases adjacentes. Cette approche peut parfois poser des problèmes. Nous pourrions améliorer l’algorithme en calculant les probabilités pour les cases adjacentes afin de déterminer laquelle attaquer, mais pour simplifier, nous ne le faisons pas.

Un autre problème avec cette approche est que lorsque les bateaux sont côte à côte, il y a une chance que la stratégie manque un bateau. Ainsi, lorsque la liste des cibles proches est vide, nous recalculons la probabilité. Supposons que la stratégie heuristique ait manqué une seule case du bateau, la fonction de probabilité considérera cette case comme vide. C’est pourquoi nous avons ajouté une boucle imbriquée pour que la probabilité des cibles explorées soit de -1, tandis que la probabilité des cases non encore touchées peut être au minimum de 0. Nous pensons donc que cet algorithme peut être amélioré en termes d’efficacité et de complexité, mais même sans ces améliorations, il reste la meilleure méthode que nous ayons utilisée jusqu’à présent.

**Simulation**

Le joueur probabiliste met à jour une grille de probabilités à chaque coup, en calculant pour chaque bateau encore en jeu les chances qu'il occupe une case donnée. En examinant toutes les positions possibles du bateau sur la grille, nous obtenons pour chaque case le nombre de fois où le bateau apparaît potentiellement. On dérive ainsi la probabilité jointe de la présence d’un bateau sur une case.

**Résultats**

**Espérance** : L'espérance du nombre de coups pour cette version est inférieure à celle de la version heuristique, mais la différence est modérée.  L'espérance du nombre de coups est **57 tirs** approximée.

**Distribution** : La distribution des coups montre une meilleure efficacité dans certaines parties, mais une variabilité plus grande due aux hypothèses d'indépendance (voir figure 3 distribution probabiliste).

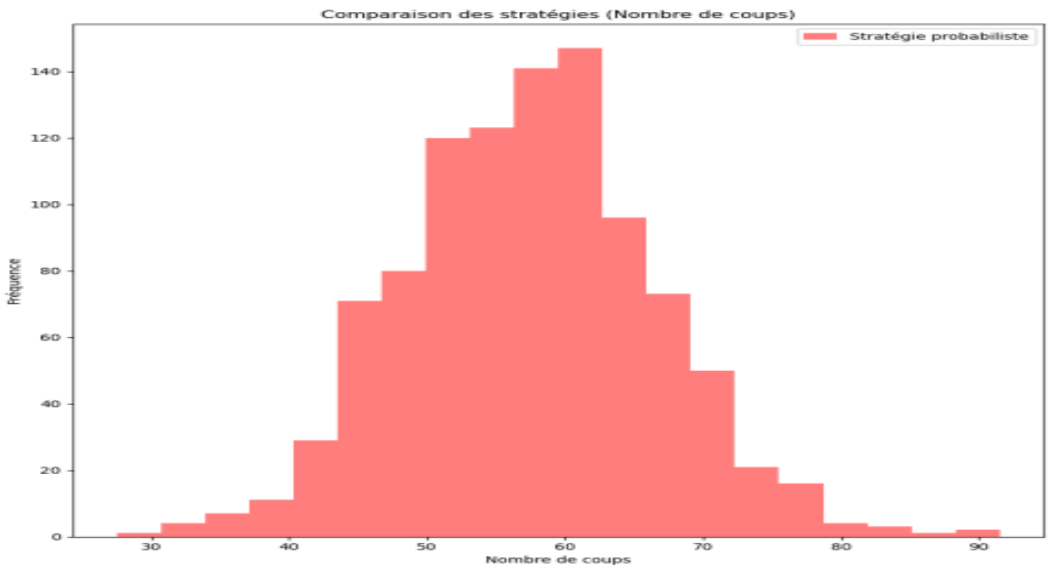



Figure 3: distribution probabiliste

**Analyse**

Cette approche est plus sophistiquée, mais l'hypothèse d'indépendance des bateaux limite son efficacité. En effet, en réalité, les bateaux ne peuvent pas occuper les mêmes cases, ce qui induit des corrélations non prises en compte. Par exemple, les positions des bateaux ne sont pas indépendantes ; la présence d'un bateau à une certaine position peut influencer la validité des positions pour d'autres bateaux. Ignorer ces interactions en supposant l'indépendance mène à des estimations de probabilité inexactes.

Nous avons proposé une implémentation qui utilise cette méthode d'indépendance pour calculer les probabilités de présence des bateaux sur la grille. À chaque tour, nous mettons à jour les probabilités en fonction des coups précédents, ce qui permet d'affiner notre estimation de la position des bateaux.

**Comparaison des résultats** : Pour évaluer l'efficacité de cette méthode, nous avons effectué plusieurs simulations et enregistré le nombre de coups nécessaires pour couler tous les bateaux. Les résultats montrent une variabilité significative en raison de l'hypothèse d'indépendance, mais une tendance générale vers une réduction du nombre de coups nécessaires par rapport à une approche aléatoire.

## 4. Version Monte Carlo Simplifiée

Dans cette version, nous utilisons un algorithme de Monte Carlo pour estimer la probabilité de toucher un bateau à une position donnée sur la grille. L'idée principale est de réaliser plusieurs simulations pour chaque position, en testant si un bateau peut se trouver à cet endroit en fonction des informations disponibles. Contrairement à une approche purement heuristique ou probabiliste, cette méthode repose sur la répétition de tests aléatoires pour évaluer chaque coup possible.

**Simulation**

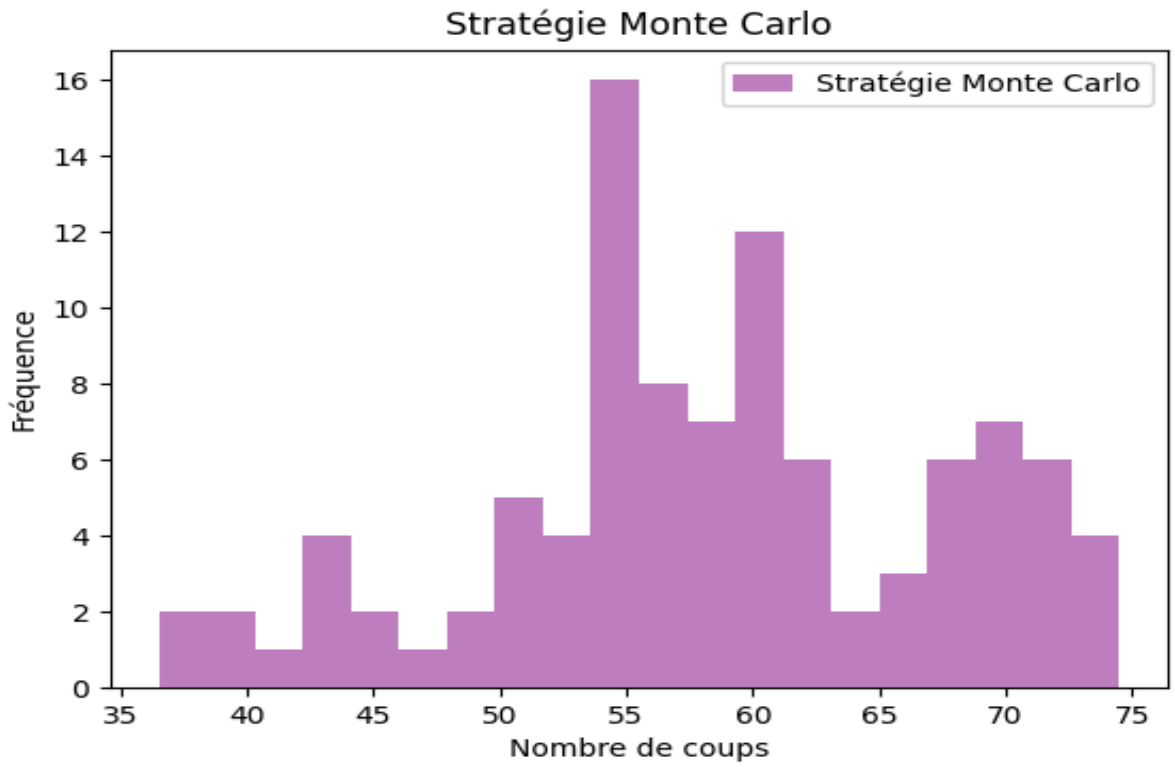
Pour chaque coup, l'algorithme simule plusieurs scénarios dans lesquels un bateau pourrait occuper la case cible. Ces simulations sont répétées un grand nombre de fois (**n**), et si la case est touchée dans une proportion suffisamment élevée des essais, elle est sélectionnée pour l'attaque.

En fonction du résultat de chaque essai, l'algorithme ajuste son estimation de la probabilité de toucher un bateau à cette position. En outre, un facteur de probabilité **p** est introduit pour moduler la décision en cas d'échec répété, évitant ainsi de s'attarder sur des positions qui semblent peu prometteuses après un certain nombre de tentatives.

**Résultats**

L'algorithme de Monte Carlo se révèle plus efficace que la simple attaque aléatoire. Le nombre moyen de coups nécessaires pour couler tous les bateaux est inférieur à celui obtenu avec une stratégie purement aléatoire ou heuristique.

En particulier, l'espérance du nombre de tirs nécessaires se situe autour de **56** coups, avec une répartition moins dispersée que dans les méthodes probabilistes plus simples. Cela s'explique par la capacité de Monte Carlo à s'ajuster progressivement aux informations acquises au fil des coups, en affinant la probabilité pour chaque case.



### Analyse

L'algorithme de Monte Carlo offre une meilleure performance en combinant l'évaluation probabiliste et la simulation répétée. Cependant, il n'est pas exempt de limitations. Par exemple, le nombre de simulations (`n`) doit être suffisamment élevé pour que les résultats convergent vers une estimation fiable, ce qui peut augmenter le temps de calcul. De plus, le facteur de probabilité `p`, utilisé pour éviter les échecs trop fréquents, peut parfois conduire à des erreurs dans le choix de la prochaine case à attaquer.

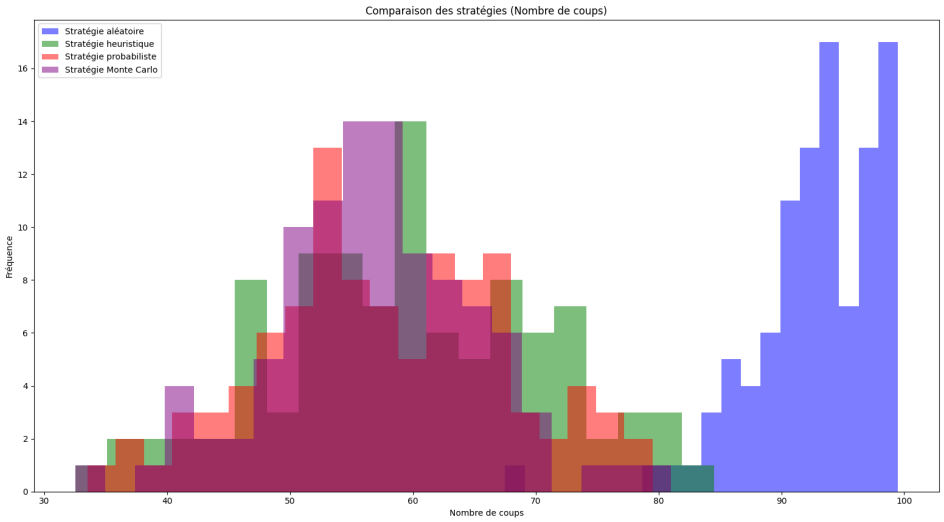
Enfin, tout comme l'approche probabiliste simplifiée, l'algorithme de Monte Carlo ne prend pas directement en compte l'interdépendance des positions des bateaux. Cela signifie qu'il traite chaque tentative d'attaque de manière relativement indépendante, ce qui peut entraîner des décisions sous-optimales dans certaines configurations de jeu.

### Comparaison des résultats

Lorsque nous comparons les résultats de l'algorithme de Monte Carlo avec d'autres approches (heuristiques, probabilistes), nous observons une amélioration globale en termes d'efficacité. Cependant, la variabilité dans le nombre de coups nécessaires demeure significative, particulièrement lorsque les simulations Monte Carlo ne convergent pas rapidement vers une probabilité élevée pour certaines positions. Néanmoins, cette approche s'avère être un compromis intéressant entre les méthodes purement probabilistes et les stratégies heuristiques plus rigides.

En conclusion, bien que des améliorations puissent être apportées, notamment en prenant en compte l'interaction entre les bateaux, l'algorithme Monte Carlo présente une stratégie robuste et adaptable pour optimiser le choix des coups dans un jeu de bataille navale.

## Conclusion pour la partie 3



Le graphique ci-dessus illustre une comparaison des performances des quatre stratégies utilisées pour attaquer dans le jeu de bataille navale : **la stratégie aléatoire, la stratégie heuristique, la stratégie probabiliste, et la stratégie Monte Carlo**. Chaque méthode présente des résultats différents en termes du nombre de coups nécessaires pour couler tous les bateaux, et nous pouvons observer des écarts significatifs dans les distributions et les espérances correspondantes.

**- Stratégie aléatoire :** Avec une espérance de **93.2** coups cette méthode est de loin la moins efficace, comme le montre la concentration élevée de coups autour des valeurs les plus élevées dans le graphique. Elle manque de toute approche systématique ou probabiliste, d'où son inefficacité par rapport aux autres méthodes.

**- Stratégie heuristique :** L'espérance de cette méthode est de **59.915** coups, indiquant une amélioration notable par rapport à l'approche aléatoire. Elle se base sur des règles logiques simples, comme l'attaque des cases adjacentes après un hit, mais reste limitée en l'absence d'un modèle probabiliste plus avancé.

**- Stratégie probabiliste :** Cette approche, avec une espérance de **57.995** coups, montre une légère amélioration par rapport à la stratégie heuristique. En prenant en compte les probabilités de présence des bateaux, elle permet de mieux cibler les attaques et ainsi réduire le nombre moyen de coups.

**- Stratégie Monte Carlo :** La méthode Monte Carlo présente l'espérance la plus basse avec **56.645** coups, se révélant ainsi la plus efficace parmi les quatre. En combinant simulations aléatoires répétées et probabilités ajustées, cette stratégie optimise progressivement les chances de toucher un bateau à chaque coup.

**Analyse finale:**

La comparaison montre que les méthodes probabilistes et Monte Carlo surpassent largement les approches aléatoires et heuristiques. Bien que la différence entre les espérances des méthodes probabilistes et Monte Carlo soit modérée, cette dernière se démarque par sa robustesse et sa capacité à s'adapter aux informations acquises durant la partie. Cela fait d'elle la stratégie la plus performante parmi celles testées.

# Partie 4 : Recherche d'Objets par Approche Bayésienne

## Modélisation et Algorithmes

Nous avons utilisé une approche bayésienne pour modéliser la recherche d'un objet (un sous-marin) sur une grille. Les probabilités de localisation de l'objet sont mises à jour à chaque itération, en fonction des résultats de détection.

**Q1. Loi de $Y_i$ et loi de $Z_i|Y_i$?**

Loi de $Y_i$ : La variable $Y_i$ indique si l'objet (le sous-marin dans ce cas) se trouve ou non dans la case $i$. Comme il n'y a qu'une seule case où l'objet peut se trouver (c'est-à-dire que $\sum_{i=1}^{N} Y_i = 1$), $Y_i$ suit une loi de Bernoulli avec paramètre $\pi_i$, qui est la probabilité a priori que l'objet se trouve dans la case $i$. Donc :

$$P(Y_i = 1) = \pi_i \quad et \quad P(Y_i = 0) = 1 - \pi_i$$

où $\pi_i$ est la probabilité a priori associée à la case $i$, qui reflète l'opinion d'experts sur la localisation probable du sous-marin.

Loi de $Z_i|Y_i$ : La variable $Z_i$ correspond au résultat du sondage de la case $i$, où $Z_i = 1$ signifie que l'objet est détecté dans la case $i$ et $Z_i = 0$ qu'il n'est pas détecté.

- Si $Y_i = 1$ (l'objet est effectivement dans la case $i$), alors le senseur détecte l'objet avec une probabilité $p_s$. Donc :

$$P(Z_i = 1|Y_i = 1) = p_s \quad et \quad P(Z_i = 0|Y_i = 1) = 1 - p_s$$

- Si $Y_i = 0$ (l'objet n'est pas dans la case $i$), le senseur ne détectera rien. Donc :

$$P(Z_i = 1|Y_i = 0) = 0 \quad et \quad P(Z_i = 0|Y_i = 0) = 1$$

**Q2. Probabilité de $Z_k = 0$ sachant $Y_k = 1$?**

Supposons que l'objet se trouve dans la case $k$ (donc $Y_k = 1$), mais que la détection n'a pas fonctionné (c'est-à-dire que $Z_k = 0$). Nous voulons calculer la probabilité de cet événement $P(Z_k = 0|Y_k = 1)$, ce qui correspond simplement à la probabilité que la détection échoue malgré la présence de l'objet.

D'après le modèle, cette probabilité est :

$$P(Z_k = 0|Y_k = 1) = 1 - p_s$$

**Q3. Mise à jour de la probabilité $\pi_k$**

Si après un sondage en case $k$, nous obtenons $Z_k = 0$, nous voulons mettre à jour la probabilité a priori $\pi_k$ pour tenir compte du fait que l'objet n'a pas été détecté dans cette case.

Nous allons utiliser le théorème de Bayes pour mettre à jour la probabilité $\pi_k$, conditionnellement au fait que $Z_k = 0$. La probabilité a posteriori que l'objet se trouve dans la case $k$, étant donné qu'il n'a pas été détecté, est donnée par :

$$P(Y_k = 1|Z_k = 0) = \frac{P(Z_k=0|Y_k=1)P(Y_k=1)}{P(Z_k=0)}$$

Développons chaque terme :

$P(Z_k = 0|Y_k = 1) = 1 - p_s$ (probabilité de non-détection conditionnelle à la présence de l'objet dans la case $k$).

$P(Y_k = 1) = \pi_k$ (la probabilité a priori que l'objet soit dans la case $k$).

$P(Z_k = 0)$ est la probabilité totale de ne pas détecter l'objet dans la case $k$, qui peut être obtenue par la loi des probabilités totales :

$$P(Z_k = 0) = P(Z_k = 0|Y_k = 1)P(Y_k = 1) + P(Z_k = 0|Y_k = 0)P(Y_k = 0)$$

Comme $P(Z_k = 0|Y_k = 0) = 1$ (puisque le sous-marin ne peut pas être détecté s'il n'est pas là), on obtient :

$$P(Z_k = 0) = (1 - p_s)\pi_k + 1 \cdot (1 - \pi_k) = 1 - p_s\pi_k$$

Donc, la probabilité a posteriori devient :

$$P(Y_k = 1|Z_k = 0) = \frac{(1-p_s)\pi_k}{1-p_s\pi_k}$$

Cette expression permet de mettre à jour la probabilité $\pi_k$ en fonction du fait qu'on n'a pas détecté l'objet dans la case $k$.

Pour mettre à jour les probabilités a priori $\pi_i$ pour les autres cases $i \neq k$ après avoir sondé la case $k$ sans détection (c'est-à-dire $Z_k = 0$), nous devons tenir compte de l'information supplémentaire que nous avons acquise. Cette information indique que l'objet n'est pas présent dans la case $k$, ce qui peut augmenter la probabilité que l'objet se trouve dans les autres cases.

**Q4. Mise à jour de $\pi_i$ pour $i \neq k$**

Nous allons utiliser le théorème de Bayes pour mettre à jour les probabilités $\pi_i$ des autres cases en tenant compte de la non-détection dans la case $k$.

Nous voulons calculer $P(Y_i = 1|Z_k = 0)$ pour $i \neq k$. Selon le théorème de Bayes, nous avons :

$$P(Y_i = 1|Z_k = 0) = \frac{P(Z_k=0|Y_i=1)P(Y_i=1)}{P(Z_k=0)}$$

Développons chaque terme :

$P(Z_k = 0|Y_i = 1)$

Si l'objet se trouve dans la case $i$ (c'est-à-dire $Y_i = 1$), alors il n'est pas dans la case $k$ (puisque $i \neq k$). Cela signifie que la détection de la case $k$ ne dépend pas de la case $i$. Ainsi, dans ce cas, $Z_k = 0$ avec probabilité 1, c'est-à-dire :

$$P(Z_k = 0|Y_i = 1) = 1$$

$P(Y_i = 1) = \pi_i$ (la probabilité a priori que l'objet soit dans la case i).

$P(Z_k = 0)$ est la probabilité totale de ne pas détecter l'objet dans la case k, qui peut être obtenue par la loi des probabilités totales :

$$P(Z_k = 0) = P(Z_k = 0|Y_k = 1)P(Y_k = 1) + P(Z_k = 0|Y_k = 0)P(Y_k = 0)$$

En tenant compte de $P(Z_k = 0|Y_k = 1) = 1 - p_s$ et $P(Z_k = 0|Y_k = 0) = 1$, on obtient :

$$P(Z_k = 0) = (1 - p_s)\pi_k + (1)(1 - \pi_k) = 1 - p_s\pi_k$$

Maintenant, nous pouvons substituer ces résultats dans notre formule :

$$P(Y_i = 1|Z_k = 0) = \frac{1\cdot\pi_i}{1-p_s\pi_k}$$

Pour toutes les cases $i \neq k$, la mise à jour des probabilités a priori se traduit donc par:

$$\pi_i' = P(Y_i = 1|Z_k = 0) = \frac{\pi_i}{1-p_s\pi_k}$$

Comme la somme des probabilités doit être égale à 1, nous devons normaliser ces probabilités mises à jour. En d'autres termes, la somme de toutes les $\pi_i'$ (pour $i \neq k$) doit être recalculée pour garantir que :

$$\sum_{i=1,i\neq k}^{N} \pi_i' = 1$$

Ainsi, on obtient :

$$\pi_i' = \frac{\pi_i}{\sum_{j \neq k} \pi_j} \quad pour\ i \neq k$$

En résumé, après avoir sondé la case $k$ sans détection, la mise à jour des probabilités a priori pour les autres cases est donnée par :

$$\pi_i' = \frac{\pi_i}{1 - p_s \pi_k}$$

et nous nous assurons que la somme de toutes les mises à jour des probabilités reste égale à 1.

Nous avons modélisé un processus de recherche d'un objet dans une grille probabiliste. L'objectif est d'explorer diverses distributions de probabilités pour évaluer leur impact sur l'efficacité de la recherche et mettre à jour les probabilités en fonction des résultats. via l'algorithme de bayesian_search_2d.py

**Méthodologie**

1. Initialisation de la grille

Une grille de taille est créée, où chaque cellule représente une probabilité d'occurrence de l'objet. Les types de distributions incluent : Uniforme, Coins, Centre, Bords et Aléatoire

2. Détection de l'objet

La fonction 'detect_object' simule la recherche dans une cellule choisie aléatoirement, tenant compte du taux de succès de détection et retourne 1 si l'objet est détecté, 0 sinon.

3. Mise à jour des probabilités

Après chaque détection, la fonction 'update_probabilities' ajuste les probabilités. Si l'objet est détecté, la cellule correspondante obtient une probabilité de 1, tandis que les autres sont mises à 0. Si non détecté, la probabilité de la cellule est diminuée d'après la formule nous avon trouvé dans le question 3: $\pi_k' = \frac{1 \cdot \pi_i}{1 - p_s \pi_k}$ et les autres sont ajustées proportionnellement d'après la formule nous avons trouvé dans le question 4: $\pi_i' = \frac{\pi_i}{1 - p_s \pi_k}$

4. Simulation de la recherche

La fonction 'search_object' gère la recherche jusqu'à un maximum d'itérations ou jusqu'à la détection de l'objet, en mettant à jour la grille à chaque itération.

5. Visualisation des résultats

À la fin, la distribution de probabilité est affichée sous forme de carte thermique pour illustrer l'état final des probabilités.

**Expériences**

Nous avons implémenté plusieurs types de distributions pour modéliser la probabilité a priori de la localisation de l'objet :

1. **Distribution uniforme** : chaque case a la même probabilité.
2. **Distribution centrée** : l'objet est plus susceptible d'être au centre de la grille.
3. **Distribution aux coins** : l'objet est plus susceptible d'être dans les coins.
4. **Distribution aux bords** : l'objet est plus susceptible d'être sur les bords.
5. **Distribution aléatoire** : chaque case a une probabilité tirée aléatoirement.

Nous avons également simulé le processus de détection en utilisant une fonction detect_object et mis à jour les probabilités via update_probabilities.

Dans l'affichage de chaque itération pour chaque distribution, nous pouvons observer le contenu suivant : 'Itération X: Recherche dans la cellule (i, j) - Détecté : True ou False ou 0' accompagné d'une matrice de probabilité correspondante.

Nous pouvons observer trois cas possibles :

1. **Le bateau est dans la case (i, j) et nous réussissons à le détecter** :

   Dans ce cas, Détecté = True.

   La matrice de probabilité correspondante aura la valeur 1 pour la case (i, j) et 0 pour toutes les autres cases. Cela signifie que la probabilité est désormais concentrée uniquement sur cette case, car l'objet a été trouvé.

2. **Le bateau est dans la case (i, j) mais nous échouons à le détecter** :

   Dans ce cas, Détecté = False.

   Nous mettons à jour la probabilité de la case (i, j) en utilisant la formule suivante pour réduire la probabilité de cette case : $\pi_k' = \frac{(1-p_s)\pi_k}{1-p_s\pi_k}$.

   Ensuite, pour toutes les autres cases, la probabilité est mise à jour avec la formule : $\pi_i' = \frac{\pi_i}{1-p_s\pi_k}$, ce qui augmente légèrement la probabilité dans les autres cellules.

3. **Le bateau n'est pas dans la case (i, j)** :

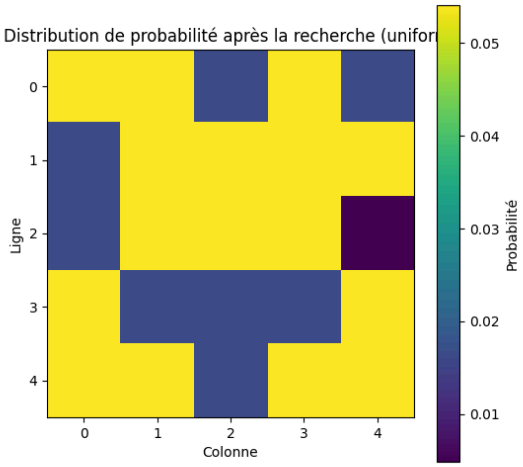
   Dans ce cas, Détecté = 0.

   La probabilité pour cette case (i, j) sera mise à jour avec la valeur 0, car nous savons avec certitude que le bateau n'est pas là.

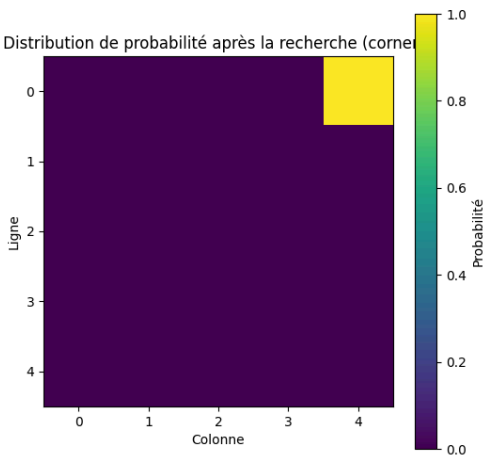
Pour les autres cases, la probabilité augmente avec la formule : $\pi_i' = \frac{\pi_i}{1-p_s\pi_k}$, ce qui permet de redistribuer les probabilités de manière équitable en fonction de la situation actuelle.

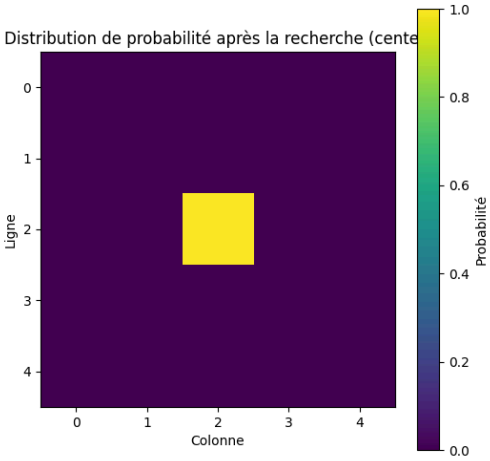
**Analyse des Résultats**

**Uniforme** : Comme prévu, la recherche est plus lente car toutes les cases sont explorées de manière homogène.

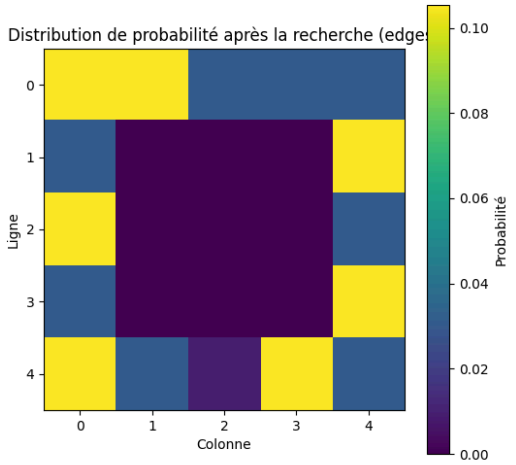



**Coins et Centre**: Ces distributions focalisent la recherche sur des zones spécifiques, ce qui peut accélérer le processus si l'objet s'y trouve

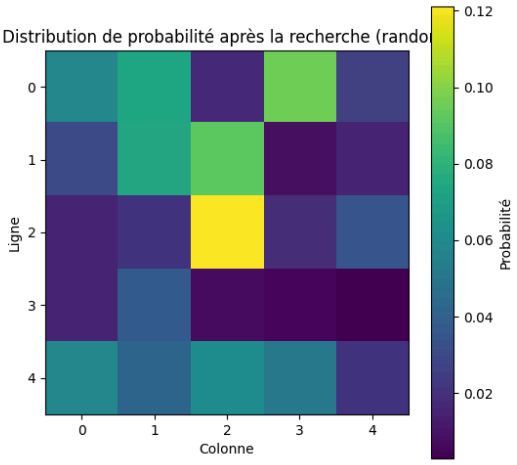


**Bords** : Cette distribution augmente les chances de détection en concentrant la recherche sur des zones accessibles, mais réduit également la diversité des emplacements possibles, ce qui peut limiter les options stratégiques.




**Aléatoire** : Cette approche produit des résultats très variables.




### Conclusion

L'approche bayésienne permet d'améliorer significativement la recherche d'objets en utilisant des informations a priori. Selon la distribution initiale, la performance de la recherche varie, démontrant l'importance du choix de la distribution a priori.

# Conclusion Générale

Ce projet s’est articulé autour de deux axes principaux.

Dans la première partie, consacrée au jeu de la bataille navale, nous avons modélisé le jeu en étudiant la combinatoire des placements de bateaux et en testant différentes stratégies de tir. L’objectif était de minimiser le nombre de coups nécessaires pour couler tous les bateaux, en passant d’une approche aléatoire à des stratégies plus complexes, comme l’utilisation de probabilités et algorithme de Monte Carlo pour guider les tirs.

Dans la deuxième partie, nous avons exploré un scénario de recherche de bateau perdu, en nous inspirant de la recherche de l’USS Scorpion. Nous avons appliqué des méthodes bayésiennes pour ajuster les probabilités de localisation en fonction des détections (ou non-détections), afin d’optimiser les chances de retrouver le bateau. Ces deux parties montrent l’application des mathématiques probabilistes à des jeux et des problèmes réels.