LU3IN005 Projet 1

Rayane Nasri

October 2024

1 Introduction

Ce petit projet a comme objectif d'étudier le jeu de la bataille navale d'un point de vue probabiliste. Il s'agira dans un premier temps d'étudier la combinatoire du jeu, puis de proposer une modélisation du jeu afin d'optimiser les chances d'un joueur de gagner, et enfin d'étudier une variante du jeu plus réaliste.
Pour rappel, Le jeu de la bataille navale se joue sur une grille de 10 cases par 10 cases. L'adversaire place sur cette grille un certain nombre de bateaux qui sont caractérisés par leur longueur :

- un porte-avions (5 cases);
- un croiseur (4 cases);
- un contre-torpilleurs;
- un sous-marin;
- un torpilleur;

Il a le droit de les placer verticalement ou horizontalement. Le positionnement des bateaux reste secret. L'objectif du joueur est de couler tous les bateaux de l'adversaire en un nombre minimum de coup. A chaque tour de jeu, il choisit une case où il tire : si il n'y a aucun bateau sur la case, la réponse est vide ; si la case est occupée par une partie d'un bateau, la réponse est touché ; si toutes les cases d'un bateau ont été touchées, alors la réponse est coulé et les cases correspondantes sont révélées. Lorsque tous les bateaux ont été coulés, le jeu s'arrête et le score du joueur est le nombre de coups qui ont été joués. Plus le score est petit, plus le joueur est performant.

2 Modélisation et fonctions simples

L'échiquier est modélisé par un tableau **Numpy** de taille 10 par 10. Chaque bateau est identifié par un entier unique : 1 pour le torpilleur, 2 pour le sous-marin, 3 pour le contre-torpilleurs, 4 pour le croiseur et 5 pour le porte avion. Ainsi les cases mise à 0 du tableau correspondent au cases vides de l'échiquier.
Dans cette partie nous avons implémenté avec succès les fonctions présente dans l'énonce dont le code Python associé se trouve dans le fichier *mod.py*.

3 Combinatoire du jeu

Dans cette partie, nous avons étudié le problème du dénombrement des configurations de grilles possibles sous différentes conditions, afin d'analyser la combinatoire du jeu. Nous avons commencé par établir une borne supérieure simple sur le nombre total de configurations pour l'ensemble des bateaux, sans considérer les contraintes liées à leur chevauchement.

Proposition 1. Soit un bateau de taille x, le nombre de façons différentes de placer ce bateau dans une grille de taille 10 par 10 est donnée par : $220 - 20.x$

Preuve. Dans une grille de dimensions 10 par 10, le nombre de cases où un bateau de longueur x peut être placé à la fois horizontalement et verticalement est donné par l'expression $[10-(x-1)]^2$. Ensuite, le nombre de cases où le bateau ne peut être placé que dans une seule direction — soit uniquement horizontalement, soit uniquement verticalement — est donné par l'expression $2.[10-(x-1)].[x-1]$. En multipliant par deux la première expression, additionnant au résultat la deuxième expression et procédant à la simplification, nous parvenons à démontrer la formule.

Après avoir déterminé le nombre de façons différentes de placer un bateau de taille x dans une grille de dimensions 10 par 10, il est aisé d'obtenir le nombre total de configurations pour l'ensemble des bateaux, sans tenir compte des contraintes de chevauchement. Ce nombre est donné par l'expression suivante :

$$\prod_{x \in T} 220 - 20.x$$

où T représente le multi-ensemble contenant les tailles des bateaux.
Une fois cette borne théorique simple établie, nous avons développé la fonction *compterPlacement(bateau)*, qui calcule le nombre de façons de placer un bateau dans une grille (initialement vide). Cela permet de vérifier si les résultats produits par cette fonction concordent avec les prévisions théoriques.

Taille du bateau	**Résultat théorique**	**Résultat expérimentale**
2	180	180
3	160	160
4	140	140
5	120	120

Table 1: Comparaison des sorties de la fonction *compterPlacement* au résultats théorique

La question que nous nous posons désormais est la suivante : Combien existe-t-il de configurations différentes de grilles valides ?

3.1 Approche de force brute

Pour y répondre, nous avons développé la fonction *compterPlacementsBateaux(bats)*, qui calcule le nombre de façons de placer la liste de bateaux fournie en paramètre sur une grille vide en utilisant une approche de force brute. Cependant, le principal inconvénient de cette approche est sa complexité qui est exponentielle. La *Table 2* synthétise les sorties de la fonction pour une liste de bateaux, quantifiant ainsi sa complexité algorithmique en termes de temps d'exécution.

Liste de bateaux	**Nombre de configurations**	**Temps CPU (sec)**
[1]	180	0.0002
[1, 2]	27336	0.042
[1, 2, 3]	3 848 040	7.08
[1, 2, 3, 4]	411 770 168	628.18
[1, 2, 3, 4, 5]	??	??

Table 2: Résultats de la fonction *compterPlacementsBateaux* pour différentes listes de bateaux

Les résultats obtenus révèlent une croissance exponentielle du temps de calcul en fonction de la taille de la liste de bateaux. Bien que l'algorithme fournisse des résultats exacts pour des entrées de taille limitée, sa complexité prohibitive le rend inutilisable pour la liste complète des bateaux.

3.2 Estimation du nombre de configurations

Reconnaissant les limites de l'approche de force brute, nous avons élaboré un algorithme heuristique capable de fournir une estimation du nombre de configurations possibles, offrant ainsi une solution approchée mais efficace au problème.
Il s'agit de déterminer le ratio λ représentant la proportion de grilles valides au sein de l'ensemble des grilles non valides. Nous employons une méthode de Monte-Carlo en générant aléatoirement N configurations initiales, sans imposer de contraintes de non-chevauchement. En dénombrant les configurations valides n parmi ces N essais, nous estimons la proportion de solutions valides par le ratio : $\lambda = \frac{n}{N}$.

Pour un échantillon de $N = 10^6$ configurations aléatoires, nous obtenons un ratio $\lambda = 0.388814$. Ce résultat suggère qu'environ 38.88% des configurations possibles satisfont les critères de validité. Ainsi, le nombre total estimé de configurations valides est de $\lambda . \prod_{x \in T} 220 - 20.x \simeq 3.10^{10}$

La *table 3* met en évidence le compromis entre la vitesse de calcul et la précision, la méthode proposée étant plus rapide mais moins exacte que la méthode par force brute.

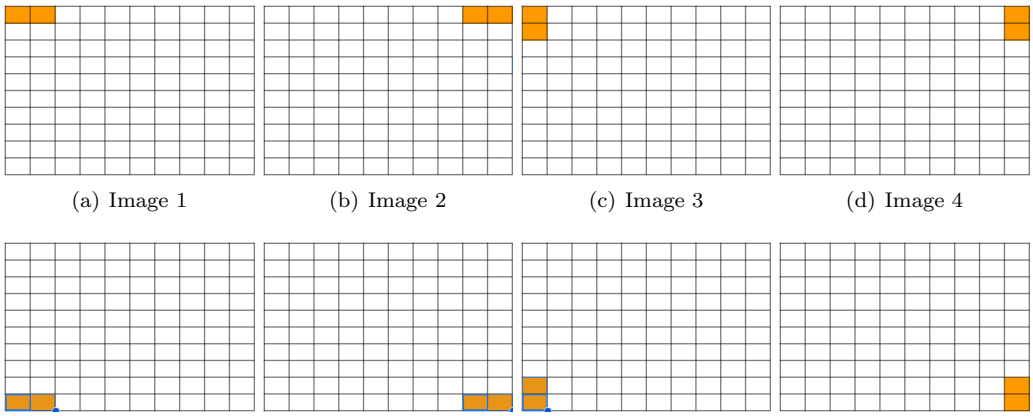
Liste de bateaux	**Force Brute**	**Monte-Carlo**	**Pourcentage d'écart**
[1]	180	70	157 %
[1, 2]	27 336	11 200	144 %
[1, 2, 3]	3 848 040	1 800 000	114 %
[1, 2, 3, 4]	411 770 168	250 000 000	64.7 %
[1, 2, 3, 4, 5]	??	3.10^{10}	??

Table 3: Comparaison des sorties de *compterPlacementsBateaux* et la méthode *Monte-Carlo*

Un avantage notable réside dans la décroissance stricte de l'écart en pourcentage en fonction du nombre de bateaux ajoutés, tandis que le temps de calcul demeure constant. Ainsi, plus la liste de bateaux est longue, plus la méthode de *Monte-Carlo* devient efficace et pertinente.

3.3 Programmation dynamique

Cette section propose une approche théorique visant à déterminer le nombre exact de configurations distinctes de grilles valides. Le principe fondamental de cette approche consiste à mémoriser les résultats afin d'éviter la répétition des mêmes calculs.
En effet, plusieurs positions différentes du même bateau peuvent entraîner une séquence de calculs identique, comme l'illustre la figure ci-dessous.



(a) Image 1 (b) Image 2 (c) Image 3 (d) Image 4

(e) Image 5 (f) Image 6 (g) Image 7 (h) Image 8

Figure 1: Exemples de positions équivalentes d'un bateau générant la même séquence de calculs

On observe que toutes les grilles de b à h peuvent être obtenues à partir de la grille a respectivement par les transformations suivantes : une symétrie par rapport à l'axe $y = 5$, une transposition, une rotation de 90°, une symétrie par rapport à l'axe $x = 5$, une rotation de 180°, une rotation de -90°, ainsi qu'une transposition suivie d'une rotation de 180°.
Afin d'éviter donc tout les calculs répétitifs inutiles on propose l'algorithme suivant :

Algorithm 1: Calcul du nombre de façons de positionner une liste de bateaux sur une grille

Input: Grille G (initialement vide), Liste de bateaux B, Cache C
Output: Le nombre de façons de positionner la liste de bateaux sur la grille

```
1  res ← 0;
2  if Liste de bateaux B est vide then
3  |  return 0;
4  end
5  if Liste de bateaux B = [x] then
6  |  return compterPlacement(x, G);
7  end
8  Prendre le premier bateau i de la liste B;
9  for chaque position (x, y) dans [0, 9]^2 do
10 |  if peutPlacer(G, i, (x, y), direction = 0) then
   |  |  // Vérifier si une configuration équivalente existe dans le cache
11 |  |  tmp ← isInCache(C, G);
12 |  |  if tmp ≥ 0 then
13 |  |  |  res ← res + tmp;
14 |  |  end
15 |  |  else
   |  |  |  // Placer le bateau et poursuivre la recherche récursive
16 |  |  |  Placer le bateau i à (x, y);
17 |  |  |  tmp ← appel récursif avec la grille mise à jour, la liste B \ {i}, et le cache C;
18 |  |  |  C[hash(G)] ← tmp;
19 |  |  |  res ← res + tmp;
20 |  |  end
21 |  end
22 |  if peutPlacer(G, i, (x, y), direction = 1) then
   |  |  // Similaire à direction = 0
23 |  end
24 end
25 return res // Retourner le nombre total de configurations
```

Nous allons détailler les fonctions auxiliaires employées dans cet algorithme.

1. *peutPlacer* vérifie si on peut placer le bateau i dans la grille G en position (x, y) dans la direction donnée *(0 Horizontal, 1 Vertical)*.
2. *isInCache* vérifie si la grille G ou une grille équivalente, comme illustré dans la *figure 1*, est présente dans le cache.
3. *hash* encode la grille en un vecteur binaire puis le soumet à une opération de hachage.

4 Modélisation probabiliste du jeu

Dans cette section nous allons mener une étude comparative de différentes stratégies visant à minimiser le nombre moyen de coups requis pour neutraliser l'ensemble des cibles sur une grille de jeu. Le code Python détaillé associé à cette section est disponible dans le module *prob.py*.

4.1 Version aléatoire

Dans cette section nous ferons l'hypothèse que toutes les cases de la grille sont équiprobables.

Proposition 2. Soient k et s respectivement le nombre total de coups et la taille totale de la flotte. La loi de probabilité de la variable aléatoire X représentant le nombre de coups nécessaires pour atteindre un état où tous les bateaux sont coulés, sous l'hypothèse d'une distribution uniforme des tirs, est donnée par

$$P(X = k) = \begin{cases} 0, & \text{si } k < 17, \\ \frac{\binom{k-1}{s-1}}{\binom{100}{s}}, & \text{sinon.} \end{cases}$$

Preuve. La destruction complète de la flotte implique nécessairement un nombre de coups supérieur ou égal à la taille cumulée des navires, ce qui justifie notre premier point. La destruction complète de la flotte requiert un dernier coup décisif. Ainsi, le problème se ramène à choisir $k-1$ coups parmi les $s-1$ cases restantes, puis à normaliser ce nombre par le total des configurations possibles.

Proposition 3. Soient k et s respectivement le nombre total de coups et la taille totale de la flotte. L'espérance de la variable aléatoire X représentant le nombre de coups nécessaires pour atteindre un état où tous les bateaux sont coulés, sous l'hypothèse d'une distribution uniforme des tirs, est donnée par

$$E[X] = \sum_{k=17}^{100} \frac{\binom{k-1}{s-1}}{\binom{100}{s}}.k$$

Preuve. Définition de l'espérance.

Afin de valider le résultat théorique, nous avons mis en place une simulation aléatoire. Une grille est générée de manière aléatoire, puis chaque coup est choisi au hasard, en excluant les cases déjà jouées, jusqu'à ce que tous les bateaux soient touchés. L'expérience est répétée 100 fois. La distribution du nombre de coups nécessaires pour terminer une partie est présentée dans la *Figure 2*.

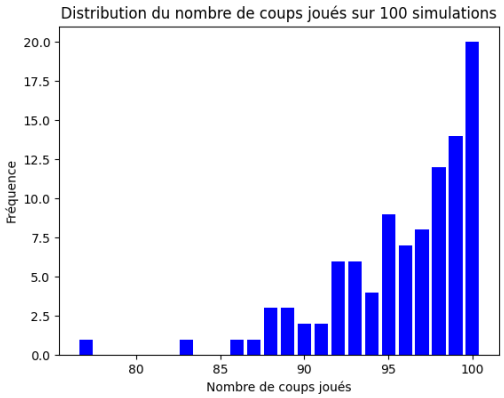


Figure 2: Histogramme de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie (Version aléatoire).

On observe que le nombre moyen de coups joués sur les 100 simulations est d'environ 95. Ce résultat concorde parfaitement avec la prédiction théorique énoncée dans la proposition 3 pour un paramètre $s = 17$. Il apparaît clairement que la version aléatoire se révèle inefficace, ce qui nous conduit à développer une approche plus optimisée, basée sur une méthode heuristique.

4.2 Version heuristique

L'élément clé de cette approche repose sur l'exploitation des coups gagnants au moment où ils surviennent. La version heuristique est composée de deux comportements : Un comportement aléatoire est adopté tant qu'aucun bateau n'est touché, tandis qu'une stratégie ciblée est employée pour tenter de détruire le bateau une fois qu'il est atteint.

Afin de comparer l'efficacité des deux approches développées jusqu'à présent, à savoir la version aléatoire et la version heuristique, nous analysons leurs performances respectives en termes de nombre moyen de coups nécessaires pour terminer une partie. La *Figure 3* présente la distribution de la variable aléatoire associée au nombre de coups nécessaires pour conclure une partie en suivant une stratégie heuristique.

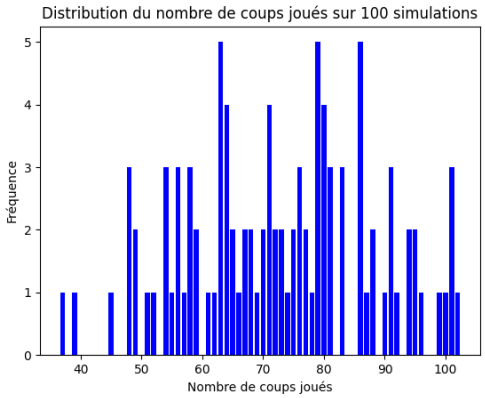


Figure 3: Histogramme de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie (Version heuristique).

La visualisation de la moyenne dans ce graphique s'avère plus complexe, le calcul de celle-ci ayant abouti à une valeur de $71,1$. En tirant parti des coups gagnants, nous avons réussi à diminuer

la moyenne du nombre de coups joués, la faisant passer de 95 à $71,1$, soit une réduction d'environ 24 coups. L'approche heuristique se révèle donc plus efficace que l'approche aléatoire, bien qu'elles adoptent un comportement similaire initialement.

L'efficacité de l'approche heuristique repose principalement sur l'exploitation des coups gagnants et la tentative de couler le bateau en visant les cases adjacentes. En effet, la probabilité qu'un coup adjacent à un coup victorieux soit également victorieux est déterminée et est donnée par : $p_h = 0,46$, ce qui est supérieur à la probabilité de toucher un bateau en effectuant des tirs aléatoires. Bien que cette approche soit supérieure à la première version aléatoire, elle présente néanmoins un inconvénient, à savoir qu'elle effectue des tirs aléatoires lors de sa première étape. Ce point sera corrigé dans la section suivante.

4.3 Version probabiliste simple

L'élément central de cette approche consiste à identifier la première case du bateau sans recourir à un tirage aléatoire. En effet, certaines cases présentent une probabilité plus élevée de contenir une partie du bateau que d'autres. Par exemple, pour un bateau de taille 2, la case (0, 0) peut contenir ce bateau de deux manières différentes. En revanche, la case (4, 4) peut le contenir de quatre manières. Ainsi, la case (4, 4) présente une probabilité deux fois plus élevée d'abriter une partie du bateau comme le montre la *figure 4*.

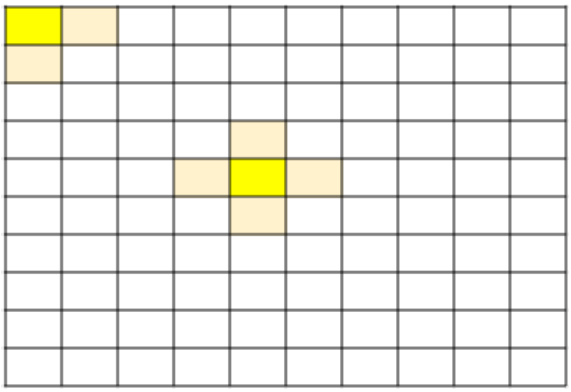
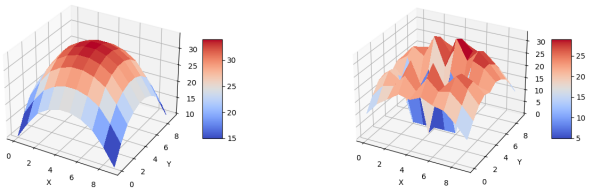


Figure 4: Comparaison des probabilités de localisation d'un bateau de taille 2 dans différentes cases.

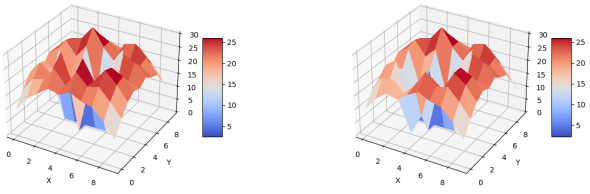
Nous proposons une approche probabiliste pour la résolution du jeu de bataille navale. Une distribution de probabilité discrète est initialisée sur un espace d'états discret représentant la grille de jeu. À chaque itération, l'état avec la probabilité a posteriori maximale est sélectionné. Suite à l'observation du résultat de ce tir, la distribution est mise à jour. Une heuristique locale est employée pour cibler les cases adjacentes aux coups réussis. Ce processus itératif se poursuit jusqu'à l'extinction de tous les navires adverses.
Afin d'illustrer ce concept, nous simulerons un jeu de taille 10×10 et visualiserons l'évolution temporelle de la distribution de probabilité au cours des dix premières itérations. Ceci est illustré dans la *figure 5* ci dessous.

Comme attendu, nos simulations révèlent une concentration initiale des probabilités vers le centre de la grille, reflétant une stratégie intuitive de ciblage des zones à plus haute densité potentielle. Au fil des itérations, on observe une dynamique intéressante : chaque tir entraîne une mise à jour locale de la distribution de probabilité. Les cases adjacentes à un coup manqué voient leur probabilité diminuer, tandis que celles situées à proximité d'un coup touché voient leur probabilité augmenter, reflétant ainsi l'adaptation de notre modèle aux nouvelles informations. Cette évolution progressive de la distribution illustre l'apprentissage continu de notre algorithme et sa capacité à affiner ses prédictions au fur et à mesure du jeu.

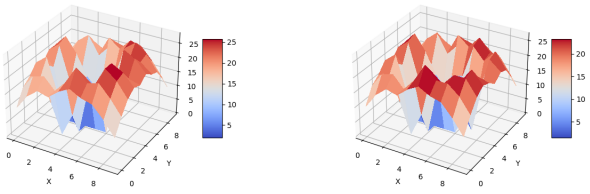
Cette stratégie se révèle significativement plus performante que les approches heuristiques et aléatoires précédemment étudiées. En effet, la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie est illustrée dans la *figure 6*. La visualisation de la moyenne dans ce graphique s'avère plus complexe, le calcul de celle-ci ayant abouti à une valeur de $53,59$. L'implémentation d'une approche probabiliste a permis de réduire de manière significative le nombre moyen de coups nécessaire pour achever une partie, passant de 71,1 à 53,59. Cette amélioration de 24% souligne l'intérêt d'une modélisation probabiliste pour optimiser la stratégie de jeu.



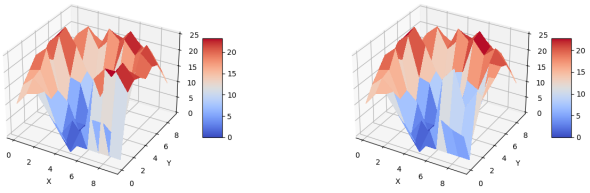
(a) Itération 1 (b) Itération 2



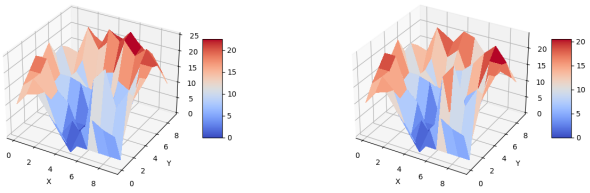
(c) Itération 3 (d) Itération 4



(e) Itération 5 (f) Itération 6



(g) Itération 7 (h) Itération 8



(i) Itération 9 (j) Itération 10

Figure 5: Distribution de probabilités pour les 10 premières itérations de l'algorithme probabiliste simple.

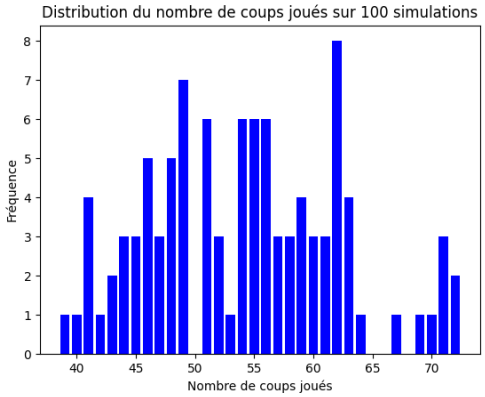


Figure 6: Histogramme de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie (Version probabiliste simple).

5 Senseur imparfait : à la recherche de l'USS Scorpion

Dans cette section, nous aborderons un problème plus réaliste. Nous chercherons un objet perdu, cependant cette fois on ne dispose pas d'un senseur exact. À chaque fois qu'une région de l'espace est sondée pour savoir si l'objet recherché s'y trouve, soit l'objet ne s'y trouve pas et le senseur ne détecte rien, soit l'objet s'y trouve et le senseur détecte l'objet avec une probabilité $ps < 1$ (et donc ne le détecte pas avec une probabilité $1 - ps$).
La modélisation de ce problème est la suivante :

- l'espace de recherche est divisé en un quadrillage de N cellules, chacune numérotée par un entier entre 1 et N;
- chaque case i a une probabilité a priori π_i de contenir l'objet recherché, correspondant à un avis d'expert qui, pour chaque région de l'espace, indique les chances que l'objet s'y trouve;
- on note $Y_i \in \{0, 1\}$ la variable aléatoire qui vaut 1 pour la case où le sous-marin se trouve et 0 partout ailleurs;
- on notera par la variable aléatoire Z_i le résultat d'une recherche en case i, valant 1 en cas de détection détection et 0 sinon;
- on suppose que les réponses aux sondages des cases sont indépendantes et identiquement distribuées.

Avec le formalisme établi, les deux propositions suivantes en résultent.

Proposition 4. Soit $k \in [\![1;\ \mathrm{N}]\!]$, la probabilité que l'objet se trouve dans la case k, conditionnellement au fait que le capteur n'ait rien détecté dans cette case, est exprimée par :

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

Preuve. D'après la formule de Bayes, on a l'égalité suivante :

$$P(Y_k = 1 \mid Z_k = 0) \cdot P(Z_k = 0) = P(Z_k = 0 \mid Y_k = 1) \cdot P(Y_k = 1)$$

D'après le contexte, on sait que :

$$P(Z_k = 0 \mid Y_k = 1) = 1 - p_s \quad \text{et} \quad P(Y_k = 1) = \pi_k$$

Pour calculer $P(Z_k = 0)$, il faut utiliser la loi des probabilités totales :

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

Ce qui donne :

$$P(Z_k = 0) = P(Z_k = 0 \mid Y_k = 0) \cdot P(Y_k = 0) + P(Z_k = 0 \mid Y_k = 1) \cdot P(Y_k = 1)$$

En appliquant les résultats précédents, on obtient :

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

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

Cela permet de calculer $P(Z_k = 0)$ en fonction de p_s et de π_k, ce qui achève la preuve.

Proposition 5. Soit $k \in [\![1;\ \mathrm{N}]\!]$, la probabilité que l'objet se trouve dans la case $i\,(i \neq k)$, conditionnellement au fait que le capteur n'ait rien détecté dans la case k, est exprimée par :

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

Preuve. D'après la formule de Bayes, on a l'égalité suivante :

$$P(Y_i = 1 \mid Z_k = 0) \cdot P(Z_k = 0) = P(Z_k = 0 \mid Y_i = 1) \cdot P(Y_i = 1)$$

D'après le contexte, on sait que :

$$P(Z_k = 0 \mid Y_i = 1) = 1 \quad \text{et} \quad P(Y_i = 1) = \pi_i$$

On a montré auparavant que :

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

Ce qui achève la preuve.

À partir des deux propositions susmentionnées, il est possible de déduire un algorithme visant à localiser l'objet égaré. Le principe est le suivant : initialement, nous disposons d'une distribution de probabilités π_i : $(i \in [\![1, N]\!])$. Nous sélectionnons la case k qui maximise cette probabilité. Si l'objet y est trouvé, la procédure est terminée. Dans le cas contraire, nous mettons à jour les probabilités des cases de la manière suivante :

$$\pi'_k = \frac{(1 - p_s) \cdot \pi_k}{1 - p_s \cdot \pi_k}$$

$$\forall i,\ i \in [\![1;\ \mathrm{N}]\!] \wedge i \neq k \quad \pi'_i = \frac{\pi_i}{1 - p_s \cdot \pi_k}$$

Puis, la recherche est relancée en utilisant les probabilités mises à jour, et le processus se poursuit jusqu'à ce que l'objet soit retrouvé. L'algorithme se présente donc comme suit.

Algorithm 2: Recherche de l'USS Scorpion

Input: Grille G de probabilités
Output: Position (x, y) de l'objet recherché

x, y ← *maximise(G)*
while *!check(G, (x, y))* **do**
    $G[x][y] \leftarrow \frac{(1-p_s)\cdot G[x][y]}{1-p_s\cdot G[x][y]}$;
    for *chaque position* $(i,j) \in [\![1;N]\!]^2$ **do**
        if $(i,j) \neq (x,y)$ **then**
            $G[i][j] \leftarrow \frac{G[i][j]}{1-p_s\cdot G[x][y]}$;
        end
    end
    x, y ← *maximise(G)*
end
return x, y

Nous allons détailler les fonctions auxiliaires employées dans cet algorithme.

1. *maximise(G)* retourne les coordonnées (x, y) de la case ayant la plus grande probabilité dans G.

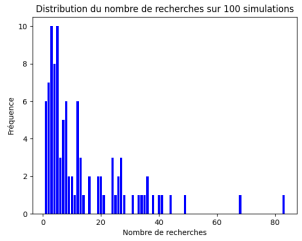
2. *check(G, (x, y))* vérifie avec une probabilité p_s si l'objet figure dans la case de position (x, y).

Afin d'évaluer les performances de l'algorithme, nous avons mené une série de simulations sur un ensemble de 100 grilles carrées de dimension $N \times N$. Pour chaque grille, les positions des objets perdus ont été générées aléatoirement selon une loi normale centrée en $\mu = \frac{N}{2}$ et d'écart-type $\sigma = 1$, simulant ainsi une distribution de densité plus élevée au centre. La probabilité de succès p_s a été variée de 0,2 à 1 par incréments de 0,2, ce qui a permis d'obtenir les cinq graphiques présentés en *figure 7*.

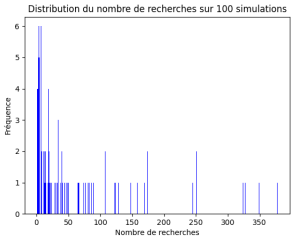
Cette étude a mis en évidence l'importance de la variance. En effet, comme le montrent les graphiques, le nombre d'itérations est généralement proche de 1. Cependant, le calcul de la moyenne révèle des valeurs significativement éloignées de 1. Cela s'explique par le fait que, dans un petit nombre de simulations, le capteur a localisé l'objet après un délai considérable, faussant ainsi la moyenne. Le tableau ci-dessous présente, pour chaque simulation, la moyenne du nombre d'itérations ainsi que l'écart-type correspondant.

p_s	**Moyenne**	**Écart type**
1	13.29	14.67
0.8	30.54	41.83
0.6	55.26	101.44
0.4	67.42	101.74
0.2	184.0	315.9

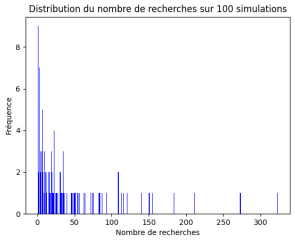
Table 4: Moyenne et écart-type du nombre d'itérations pour chaque simulation



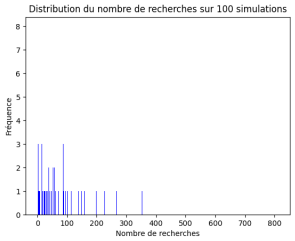
(a) $p_s = 1$



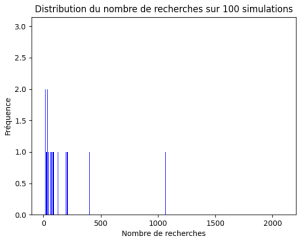
(b) $p_s = 0,8$



(c) $p_s = 0,6$



(d) $p_s = 0,4$



(e) $p_s = 0,2$

Figure 7: Distribution du nombre d'itérations nécessaires pour localiser un objet égaré en fonction de la précision du senseur p_s

6 Conclusion

Dans ce projet, nous avons entrepris une analyse probabiliste approfondie du jeu de la bataille navale. En premier lieu, nous avons exploré la complexité combinatoire en nous intéressant au nombre total de configurations possibles de la grille de jeu. Étant donné que le calcul exact de ce nombre par une approche exhaustive s'avère impraticable en raison de sa complexité, nous avons conçu et mis en œuvre des méthodes d'estimation basées sur les simulations Monte-Carlo. Ces approches nous ont permis d'obtenir des approximations des configurations possibles, surmontant ainsi les limitations inhérentes aux techniques de force brute. Nous avons ensuite élaboré un algorithme théorique reposant sur la programmation dynamique, capable, en principe, de calculer de manière exacte le nombre total de configurations possibles. Cette approche exploite les sous-structures répétitives du problème, offrant ainsi une solution potentielle plus efficace et rigoureuse que les méthodes d'estimation précédentes. La seconde partie du projet s'est focalisée sur la modélisation probabiliste du jeu, avec une analyse comparative des diverses stratégies visant à minimiser le nombre moyen de coups nécessaires pour détruire l'ensemble des navires sur la grille. Cette étude a permis d'évaluer l'efficacité des différentes approches : Version aléatoire, version heuristique et version probabiliste simple. Enfin, dans la dernière partie, nous avons abordé un problème plus réaliste, celui du capteur imparfait. En nous appuyant sur le théorème de Bayes, nous avons développé un algorithme probabiliste qui intègre l'incertitude liée aux détections erronées, permettant ainsi d'optimiser les stratégies de jeu en présence de données incomplètes ou bruitées.

La prochaine étape consiste à implémenter l'algorithme de programmation dynamique afin de calculer de manière exacte le nombre total de configurations possibles. Cette mise en œuvre permettra de valider la faisabilité théorique de l'algorithme et d'obtenir des résultats précis en contraste avec les approximations fournies par les méthodes Monte-Carlo.