LU3IN005 - STATISTIQUES ET INFORMATIQUE-2020 OCT

RENDU DU PROJET 1 :
BATAILLE NAVALE

MOHELLEBI Melissa 3971070
SAICHI Lina 3803151

Guidé par monsieur Jean-Yves Franceschi

UPMC - Informatique -Sorbonne Université

Contents

0.1 INTRODUCTION

Le jeu de la bataille navale se joue à deux sur une grille de 10 cases par 10 cases.Chaque joueur dispose de cinq bateaux à placer à des positions de leur choix sur la grille. Chaque bateau est caractérisé par son nom et sa longueur :

- un porte-avions (5 cases) ;
- un croiseur (4 cases) ;
- un contre-torpilleurs (3 cases) ;
- un sous-marin (3 cases) ;
- un torpilleur (2 cases).

Dans la version standard, à chaque tour de jeu, le joueur choisit une case aléatoirement.Si la grille de l'adversaire contient un bateau à cette case là, ce dernier crie "touché". On dit qu'on a coulé un bateau si on a touché toutes ses cases.
Le premier joueur à avoir coulé tous les bateaux de l'adversaire a gagné la partie.

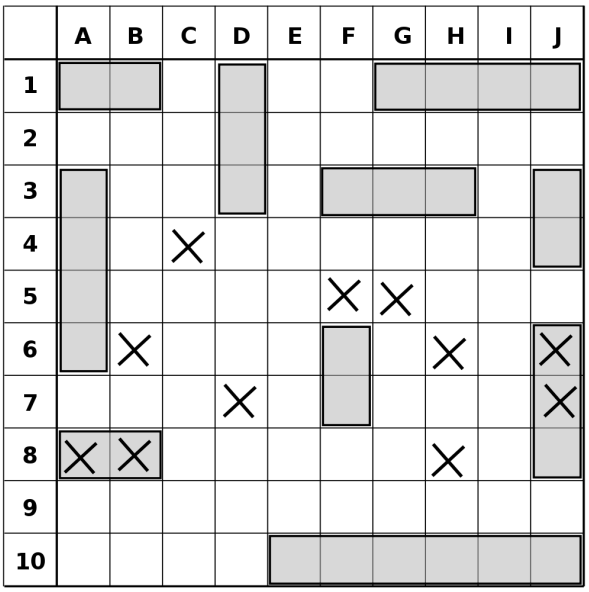

Image 1 : image représentant une grille du jeu de la bataille navale.

0.2 BUT DU PROJET

Le but du projet est de réaliser en python un jeu de la bataille navale qui doit respecter les régles du jeu standard.Le présent rapport décrit les différentes méthodes pour gagner à ce jeux en utilisant une approche probabiliste.
Ce rapport est composé de quatres parties. La première partie consiste a modéliser la grille du jeu.Dans la deuxième, il s'agit d'étudier la combinatoire du jeu, la troisième porte sur la proposition d'une modélisation du jeu afin d'optimiser les chances d'un joueur pour gagner, enfin nous allons etudier une variante du jeu plus réaliste.

0.3 MATÉRIELS ET MÉTHODES

Le langage de programmation utilisé dans ce projet est Python.Nous avons utilisé les modules Numpy, Matplotlib et Random.
Le projet est divisé en plusieurs fichiers, les fichiers qui implémentent les classes de la première partie sont: grille.py et bataille.py.
Nous avons écrit les fonctions demandées dans cette partie ainsi que dans la seconde partie dans un autre fichier intitulé fonctions.py.
En ce qui concerne la troisième partie, nous avons implementé les trois versions de jeu dans un meme fichier, bataille.py qui contient la classe bataille, et aussi les fonctions demandées dans cette partie.
Enfin, nous avons implémenté la dernière partie dans un fichier bayes.py qui contient l'algorithme de recherche bayésienne.

Chapter 1

Modélisation et fonctions simples

Cette partie consiste à modéliser le jeu de la bataille navale.
Nous modéliserons la grille en utilisant une matrice entière de taille 10 x 10. Chaque navire sera codé avec un identifiant entirer: 1 pour le porte-avions, 2 pour le croiseur, 3 pour le destroyer, 4 le sous-marin et enfin 5 pour le torpilleur.
Par conséquent, la grille de jeu contiendra cent cases ou la valeur de la case diffère selon ce qu'elle contient. Si celle-ci contient une case d'un bateau, alors sa valeur sera égale à la valeur du bateau (ce sont les valeurs décrites précedemment). Par ailleurs, si la case est vide, elle contiendra la valeur 0.
Somme toutes,au début du jeu, notre grille contiendra **17 cases occupées** par des bateaux, et donc 83 cases seront vides.
Dans cette partie, nous avons implémenté toutes les fonctions basiques nécessaires pour le bon fonctionnement du jeu. C'est-à-dire les fonctions qui initialisent la grille du jeu, y positionnent les bateaux aléatoirement, et permettent aussi l'affichage de la grille..etc

Chapter 2

Combinatoire du jeu

Dans cette partie, nous nous intéressons au dénombrement du nombre de grilles possibles dans différentes conditions pour appréhender la combinatoire du jeu.

2.1 RÉPONSE 1

Le nombre de configurations possibles pour la liste complète de bateaux sur une grille de taille 10 :pour un seul bateau, soit long la longueur du bateau alors le résultat est :
10*2*(10-long +1) ...(1)
Explication de la démarche : Soit E un ensemble de n élements, le nombre de configurations possibles de poser un mot M de longueur k avec k<=n de maniére contigue est le nombre de positions ou on peut poser le premier élement de M et écrire tout M.
Dans notre example, c'est le nombre de façons de poser un bateau de longueur long sur une ligne de longueur 10. On multiplie ensuite ce résultat par 10 car il y a 10 lignes puis par 2 car il y a autant de lignes que de colonnes.
Le calcul de la formule (1) pour chaque élément de la liste :

- (10 – 5 + 1)*2*10 = 120
- (10 – 4 + 1)*2*10 = 140
- (10 – 3 + 1)*2*10 = 160
- (10 – 3 + 1)*2*10 = 160
- (10 – 2 + 1)*2*10 = 180

Nous voulons calculer une borne supérieure.On pose l'hypothese de l'indépendance des bateaux, c'est à dire qu'on ignore le fait que des bateaux peuvent se chevaucher sur des cases.

Dans ce cas nous allons donc multiplier le résultat de la formule (1) pour chaque élément de la liste des bateaux.
Ce qui donne un résultat de : Ce qui donne un total de 120*140*160*160*180 = 77414400000 combinaisons.

2.2 RÉPONSE 2

Ecrire une fonction qui permet de calculer le nombre de façons de placer un bateau donné sur une grille vide. Pour implémenter cette fonction on procède de la manière suivante:
D'abord pour une position donnée on vérifie si celle ci se situe à l'interieur de la grille. Ensuite on vérifie dans la case qui se trouve à côté d'elle et selon la direction donnée en paramètre si elle est vide. Deux cas se présentent à nous :

- Soit la case qui suit est vide alors on regarde la case qui la suit et ceci se répète n fois (n étant le nombre de cases qu'occupe le bateau passé en paramètre)
- Soit la case n'est pas libre c'est à dire qu'un bateau s'y trouve déjà ou c'est une case inexistante dans la grille dans ce cas la on sort de la boucle en retournant "false".

  Nous avons comparé le résultat de cette fonction àu résultat de la question précédente **appliquée sur un seul bateau** : nous obtenons les mêmes résultats.

Si on arrive a sortir de la boucle alors les n cases qui suivent la positions donnée en paramètres sont libres dans la limite de la grille.Dans ce cas on retourne un "true".

2.3 RÉPONSE 3

Dans cette question, on s'intéresse à l'implémentation d'une fonction qui calcule le nombre exact de configurations possibles pour placer une liste de bateaux sur la grille : On lui passe en paramètres la liste des bateaux qu'on souhaite placer sur la grille, ainsi qu'une grille vide préalablement initialisée.
L'algorithme commence par chercher toutes les cases possibles ou on peut placer le premier bateau (grâce a la fonction liste_positions_possibles qu'on a définie avant) , ensuite en parcourant la liste des positions possibles, pour une case et une direction donnée (horizontale ou verticale), nous allons placer le bateau et augmenter le compteur.
Après ça, nous allons appeler la fonction récursivement sur le reste de la liste ds bateaux.

Cette fonction rend un résultat pour une liste de un, deux bateaux. Cependant, **il n'est pas possible** de calculer pour la liste complète de bateaux, car elle est très couteuse en mémoire, en vue des nombreux appels récursifs qu'on fait.

2.4 RÉPONSE 4

Nous nous intéressons ici à la probabilité de tirer aléatoirement une grille donnée , les événements élémentaires sont les grilles qu'on peut tirer. On considère l'univers Ω qui contient l'ensemble des grilles possibles. Toutes les grilles sont équiprobables, donc
$P(\{\omega\}) = \frac{1}{card(\Omega)} \forall \omega \in \Omega$

2.5 RÉPONSE 5

Quand le nombre d'essais augmente, les fréquences observées sont de plus en plus proches des valeurs théoriques données par la loi de probabilité. C'est pourquoi, quand on ne peut pas determiner la loi d'une v.a,on considère que les valeurs empiriques (c'est-'a-dire mesurées sur un échantillon) sont des valeurs approchées des probabilités.

Chapter 3

Modélisation probabiliste du jeu

3.1 VERSION ALÉATOIRE

On tire aléatoirement une case. La probabilité qu'un bateau s'y trouve est de 17 %
On note X le nombre de tirages nécessaires pour tirer toutes les cases du bateau (tirer 17 fois une case où il y'a un bateau)
X prend ses valeurs dans {17,18,...,100} On s'arrête lorsque on a tiré la dernière case restante d'un bateau
Si X = k avec k $\in$ [17 , 100], alors le dernier lancer est une case d'un bateau. P (X = k) = C_{k-1}^{17-1} $p^{17}(1-p)^{17} avec p = 0.17$
C_{k-1}^{17-1} car on sait que le dernier tirage est réussi, et avant on a eu 16 tirages réussi, donc 16 parmis k-1, avec k le nombre total de tirages faits. L'espérance qu'on obient ici sera égale à n/p , avec n le nombre de réussites qu'on a , et donc ce qui nous donne un résultat de **100**.
Remarque : l'hypothèse qu'on a faite ici, est de considerer un tirage sans remise.Cependant,dans la question suivante il est explicitment demandé de ne pas tirer les positions déjà choisises. En ayant simulé la fonction un grand nombre de fois, nous avons remarqué que la moyenne des résultats obtenus se situaient entre 94 et 99. Ce résultat est inférieur à l'espérance théorique, ce qui peut se justifier par l'hypothese qu'on a formulée d'un tirage avec remise. Ce qui n'est pas le cas dans la fonction qu'on a écrite.

La fonction qu'on a implémentée dans cette partie joue aléatoirement au jeu. A chaque tour du jeu, une position et une direction sont tirée aléatoirement. On vérifie ensuite si cette case contient un bateau, et si c'est le cas, on incrémente le compteur. Ce pendant, dans tous les cas, on l'ajoute à une liste: liste_deja_tirée où on stocke les positions déjà tirée, afin de gagner en efficacité

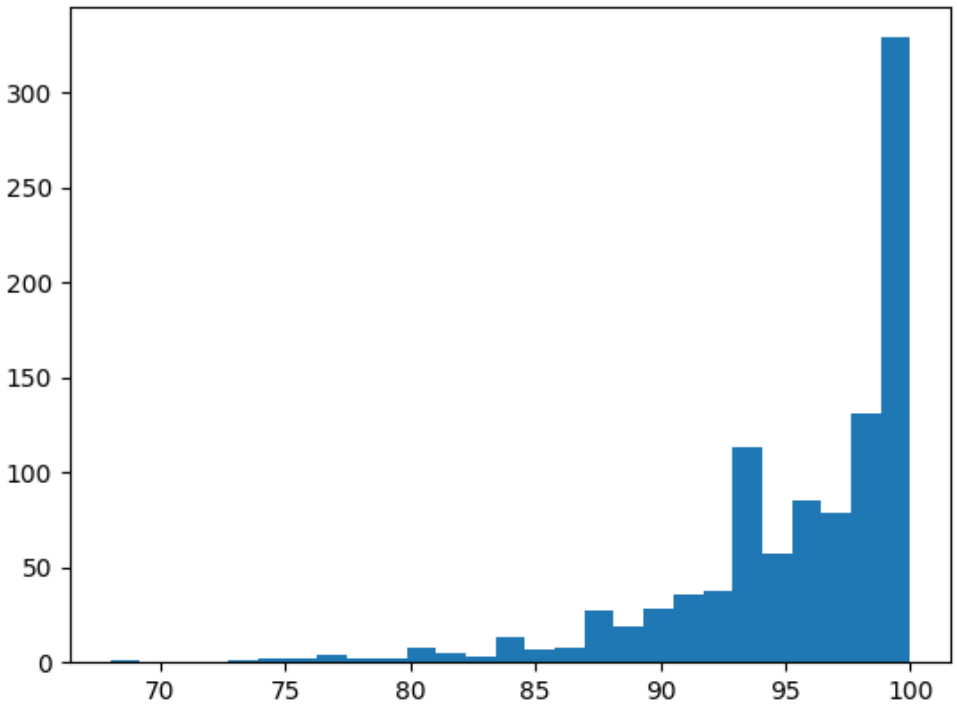

figure 1 : Histogramme représentatif de la distribution du nombre de coups pour la version aléatoire. N=1000 essais. L'axe des abcisses repésente le nombre de coups joués avant de terminer la partie.L'axe des ordonnées représente la fréquence de leur appartion .

La figure 1 représente la distribution du nombre de tirs nécessaires pour gagner la partie avec la stratégie aléatoire. Nous observons que la distribution ne suit pas une loi statistique particulière.Par ailleurs,le nombre maximal d'occurences se situe entre 94 et 99 (plus de 330 simulations). A l'inverse on remarque une absence de résultats lorsque le nombre de coups joués est inférieur à 80.Ainsi, on en déduit que cette stratégie est inneficace.

3.2 VERSION HEURISTIQUE

Dans cette version, nous allons essayer d'améliorer la version précédente. du fait que les cases des bateaux sont placées de manière contigué. si on tire une case et qu'on y trouve un bateau, cela nous apporte une information supplémentaire sur l'emplacement des cases

restantes.
Cette fonction commence avec un comportement aléatoire, où on tire une case aléatoirement(il est aussi important de stocker les cases déjà tirée dans une liste)
Une fois un bateau touché, nous allons cherccher dans ses cases voisines celles qui contiennent aussi un bateau(ici on ne tient pas en compte le type du bateau qu'elles contiennent)
On vas ensuite faire une recherche récursive sur ces cases voisines(pour eller voir les voisines des voisines...etc)
Nous veillerons à ajouter toutes ces cases dans la liste des positions déjà tirées afin d'éviter d'y revenir .
Enfin, à chaque fois qu'on trouve une case voisine qui contient un bateau nous allons incrémenter le compteur des cases de bateaux touchées.
Une fois, une case touchée et qu'on a récupéré toutes ses cases voisines occupées nous allons revenir au comportement aléatoire et répéter le processus

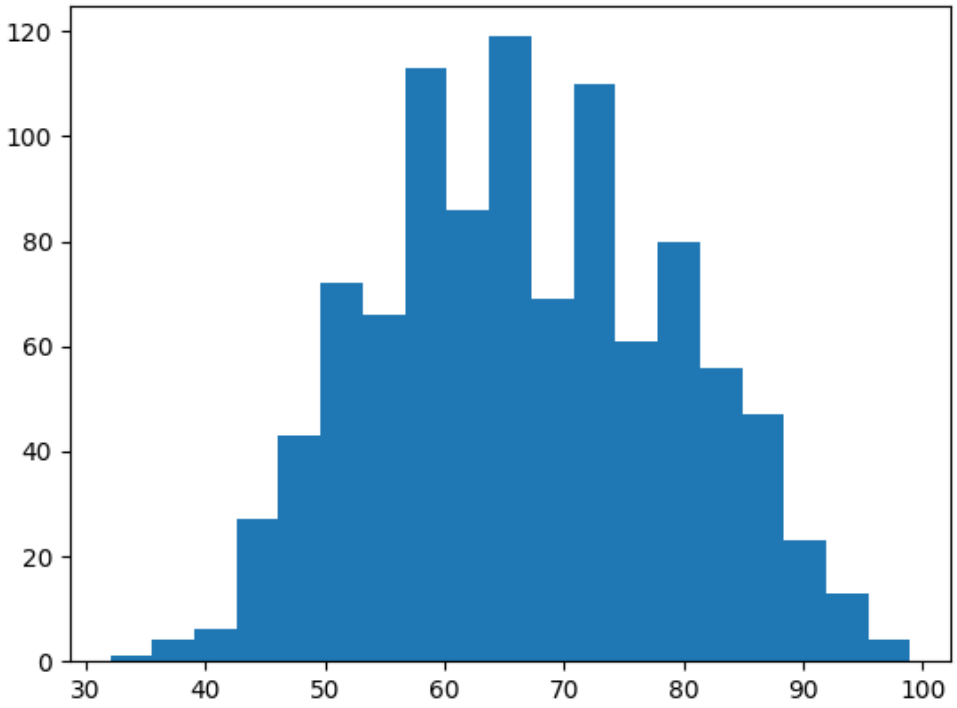

figure 2 : Histogramme représentatif de la distribution du nombre de coups pour la version heuristique. N=1000 essais. L'axe des abcisses repésente le nombre de coups joués avant de terminer la partie.L'axe des ordonnées représente la fréquence de leur appartion .

La figure 2 représente la distribution du nombre de tirs nécessaires pour gagner la partie avec la stratégie heuristique. Nous observons que la distribution est unimodale en cloche.Par ailleurs,le nombre maximal d'occurences se situe à 68 coups (plus de 190 simulations).A l'inverse de la figure 1, au delà de 65 coups, le nombre de simulations ayant donné ce résultat diminue. On en déduit que cette implémentation est plus efficace que la précédente.
Nous justifions ce résultat par la stratégie déployée dans cette partie. En effet,lorsqu'on touche une case qui contient un bateau, nous allons chercher dans toutes ses cases adjacentes, et toucher toutes celles qui ne sont pas vides. Enfin, comme pour la partie précédente, nous ls stockons dans une liste afin de ne pas les tirer encore une fois.

3.3 VERSION PROBABILISTE SIMPLIFIÉE

Ici nous allons implémenter une version probabiliste pour finir le jeu le plus rapidement. Dans les versions précédentes, nous ne tenons pas compte des bateaux restants ni de l'admissibilité de leur positionement: or nous savons par exemple que les cases du milieu ont plus de chances de contenir un bateau que celles du bord de la grille.
Dans cette version, nous allons nous servir d'une grille de possibilitées où chaque case va d'abord contenir le nombre de façons d'y positionner un bateau. Ensuite on vas diviser le résulatat de chaque case par le nombre de placements possibles de ce bateau sur la grille. On aura donc une probabilité par case.
Si on exprime cela methematiquement nous avons pour chaque case la probabilité qu'elle contienne un baeau donné.
Ce que nous voulons, c'est la probabilité pour chaque case de contenir un des bateaux et donc c'est la probabilité :
P(bateau 1 occupe cette case OU bateau 2 occupe cette case ... etc) =
1 - P(bateau 1 n'occupe pas cette case ET bateau 2 n'occupe pas cette case...etc)
Cependant , nous avons fait l'hypothèse qu'on considère chaque bateau de manière indépendante et donc on aura que
P(bateau 1 n'occupe pas la case(i,j) ET bateau 2 n'occupe pas la case(i,j)...etc)=
P(bateau 1 n'occupe pas)× P(bateau 2 n'occupe pas)× ... etc
Ici l'hypotèse d'indépendance est fausse car on doit tenir compte de la position des autres bateaux lors du calcul de la probabilité de trouver un bateau donné sur la grille. De plus, en sachant qu'un bateau a été touché dans une région nous donne des informations sur l'éventuelle position des autres bateaux.

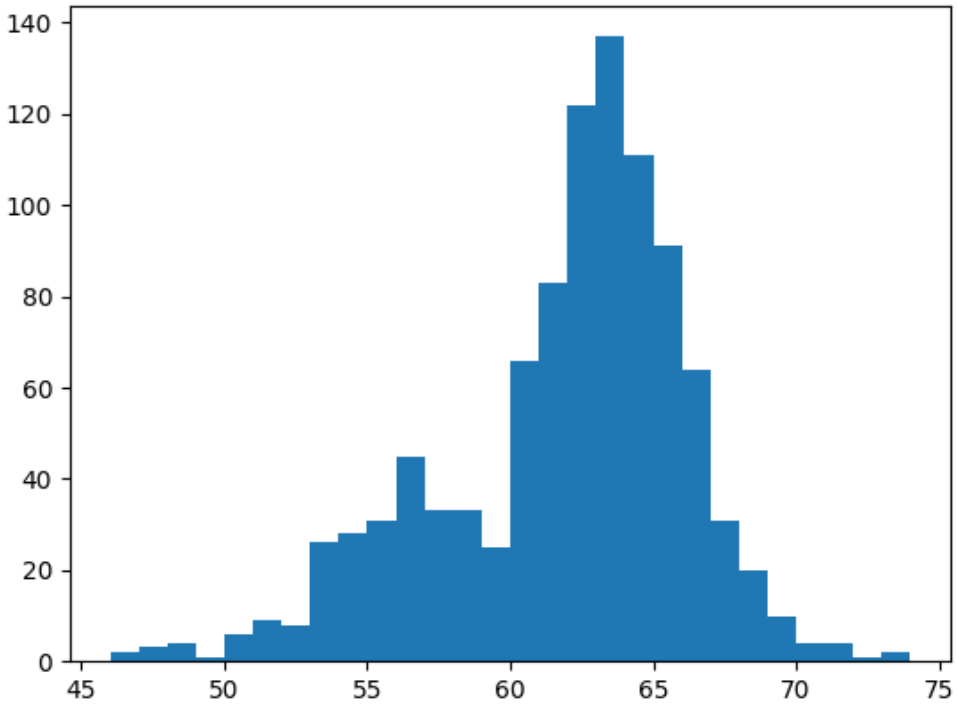

figure 3 : Histogramme représentatif de la distribution du nombre de coups pour la version probabiliste simplifiée. N=1000 essais. L'axe des abcisses repésente le nombre de coups joués avant de terminer la partie.L'axe des ordonnées représente la fréquence de leur appartion

La figure 3 représente la distribution du nombre de coups nécessaires à l'écoulement des navires avec la stratégie probabiliste simplifiée. Nous observons que la distribution des coups est bi-modale; il existe ainsi deux pics représentatifs des maxima du nombre de coups. Le premier pic (nS=40 simulations) est atteint à 56 coups; le second pic (nS=140) est atteint à 65 coups. Cela signifie que cette stratégie simplifiée permet de faire couler l'ensemble des navires en un nombre réduit de coups (n=56 coups) qui est ateint dans un premier record maximal de 40 simulations.

3.4 DISCUSSION ET CONCLUSION

Compte tenu de la l'analyse comparative entre les trois stratégie décrites précedemment, la version probabliste simplifiée s'avère être la stratégie la plus optimale pour terminer le jeu et faire couler tous les navires en un moindre nombre de coups. En effet, dans le cas de la

stratégie purement aléatoire, les variations du nombre de simulations relatives aux nombres de coups nécessaires sont stochastiques. Le nombre de coups minimal pour terminer le jeu était strictement supérieur à 75. Néanmoins, le nombre de simulations qui reproduisent ce score est très faible (<10 simulations). En parallèle, il aurait fallu jusqu'à 300 simulations ayant un nombre de coups équivalent à 100 pour terminer le jeu. Cette première stratégie n'est donc pas optimale. La seconde stratégie, reposait sur une version heuristique, possiblement modélisée par une loi de probabilité. La distribution "en cloche", centrée sur 65, indique que le nombre maximal de simulations (nS=120) correspond à n=65 coups. Ce nombre est inférieur à celui proposée par la 1ère stratégie (n>75). Cela indique que la version heuristique est plus optimale que la version aléatoire. Enfin, la version probabiliste simplifiée suit une distribution bi-modale: indication de deux pics du nombre maximal de simulations (nS=40 et nS=140 respectivement) nécessaires à l'aboutissement du jeu. EN effet, cette stratégie a permis d'atteindre l'objectif avec seulement n=57 coups, réalisables en nS=40 simulations. Ce nombre de coups observé est le plus petit des trois nombres proposés par chaque stratégie.

En conclusion, la stratégie simplifiée est la plus optimale pour faire couler l"ensemble des navires, en un nombre minimal de coups, réalisables en 40 simulations uniquement.

Chapter 4

Senseur imparfait : à la recherche de l'USS Scorpion

4.1 Réponse 1

$Y_i \in \{0,1\}$ est la variable aléatoir qui vaut 1 dans la case qui contient le bateau, et 0 partout ailleurs
Y_i suit une lois de Bernouli de parametre M_i
Z_i / X_i : en sachant si le sous_marin se trouve ou pas dans la case i, on veut mesurer la probabilité de le détecter
$(Z_i = 0 \,|\, Y_i = 0\,) = 1$
$(Z_i = 0 \,|\, Y_i = 1\,) = 1 - p_s$
$(Z_i = 1 \,|\, Y_i = 0\,) = 0$
$(Z_i = 1 \,|\, Y_i = 1\,) = p_s$

4.2 Réponse 2

On exprime la probabilité de cette événement ainsi:
$P(Y_k = 1 \cap Z_k = 0)$

4.3 RÉPONSE 3

Développement:
P($Y_k = 1 \cap Z_k = 0$) = P($Z_k = 0 | Y_k = 1$)×P($Y_k = 1$) = (A-p_s)×π_k
La probabilité $\pi_k$vas changer au cours de la recherche, car en sondant une case si on ne trouve rien (cependant on n'est pas à 100% sur que le bateau n'y pas), le π_k de cette case va diminuer.
La probabilité qui nous intéresse ici est :

$$\begin{aligned} P(Y_k = 1 | Z_k = 0) &= \frac{P(Y_k = 1 \cap Z_k = 0}{P(Z_k = 0)} \\ &= \frac{(1 - p_s) \times \pi_k}{\pi_k \times (1 - p_s) + 1(1 - \pi_k)} \\ &= \frac{(1 - p_s)\pi_k}{1 - p_s \pi_k} \end{aligned}$$

et donc :

$$\pi_{knouveau} = \frac{(1 - p_s)\pi_{kancien}}{1 - p_s \pi_{kancien}}$$

Dans cette équation, on suppose que si on a pas trouvé le sous_marin dans la case k alors il se trouve dans une autre case $i \neq k$ et donc nous étudions la probabilité P($Y_i = 1 \,| Z_k = 0$)

$$\begin{aligned} P(Y_i = 1 | Z_k = 0) &= \frac{P(Y_i = 1 \cap Z_k = 0)}{P(Z_k = 0)} \\ &= \frac{P(Z_k = 0 | Y_i = 1) P(Y_i = 1)}{P(Z_k = 0)} \\ &= \frac{1 - \pi_i}{1 - \pi_k ps} \end{aligned}$$

$P(Z_k = 0 | Y_i = 1) = 1$ car si le bateau se trouve dans la case i, alors il ne se trouve surement pas dans les autres cases et donc on aura

$$\pi_{inouveau} = \frac{\pi_{iancien}}{1 - \pi_{kancien} p_s}$$

4.4 RÉPONSE 4

L'algorithme :
1- Tant que le sous_marin n'est pas trouvé nous itérons la recherche.
2-À chaque étape, on choisit la cellule ayant la probabilité la plus élevée.
3-Nous effectuons la recherche à l'aide d'une fonction qui détecte le bateau avec une probabilité ps.
4-Si le sous_marin n'est pas trouvé, nous mettons à jour les probabilités de toutes les cellules et nous réitérons la recherche.