LU3IN005
Probas, Stats et Info–2020fev
UPMC

Projet 1 : Bataille navale

PEREIRA GAMA Gustavo (3301503)
EL BEBLAWY Rami (3670209)

19-03-2020

Sommaire

Introduction

Le but de ce projet est d'étudier les probabilités liées au célèbre jeu de la Bataille Navale.

Nous allons d'abord nous allons implémenter le jeu lui même.
Ensuite on se focalisera sur la partie combinatoire du jeu(nous allons dénombrer le nombre de façons de placer des bateaux sur une grille etc).
Et pour finir on va s'intéresser à la modélisation du jeu pour améliorer les chances de victoire du joueur. Premièrement on va implémenter une stratégie très simple qui consiste a jouer aléatoirement, deuxièmement on va implémenter une qui exploite les coups précédents en analysant les cases voisines. Troisièmement on va s'intéresser à une méthode qui utilise les probabilités afin d'améliorer les chances de victoire.
On s'intéressera bien sûr aux résultats et à la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie pour chaque stratégie.

Nous avons la configuration suivante pour le jeu de la bataille navale sur un plateau 10*10:

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

1 Combinatoire du jeu

Question 1) Donner une borne supérieure du nombre de configurations possibles pour la liste complète de bateaux sur une grille de taille 10 (calcul à la main).

Nous avons 5 bateaux différents qui occupent un certain nombre de cases, on a donc : 2+3*2+4+5= 17 cases. La borne supérieure possible consiste donc a placer ces 17 cases parmi les 100 présentes dans la grille. On peut l'obtenir en calculant le nombre d'arrangements possibles avec la formule : $A_n^k = \frac{n!}{(n-k)!}$, ce qui nous donne : A_{100}^{17}.

Question 2)Donner d'abord une fonction qui permet de dénombrer le nombre de façons de placer un bateau donné sur une grille vide. Comparer au résultat théorique.

Soit un bateau de taille n. Sur une ligne de 10 cases on peut obtenir $10-n+1$ façons de placer un bateau juste pour une ligne. Pour la grille entière(10*10) on a donc: $2 * 10 * (10 - n + 1)$ car on peut placer le bateau soit verticalement ou horizontalement.
Comparons maintenant ce résultat théorique avec nos résultats de la fonction *denombre_places_bateau(grille,bateau)* qui prend en entrée une grille vide ainsi qu'un bateau auquel on attribue une taille et s'appuie sur la fonction *peut_placer* pour effectuer le dénombrement selon les bordures et la taille du bateau :

* Pour un bateau de taille 2 : il y a 180 possibilités.
* Pour un bateau de taille 3 : il y a 160 possibilités.
* Pour un bateau de taille 4 : il y a 140 possibilités.
* Pour un bateau de taille 5 : il y a 120 possibilités.

On obtient bien le même résultat que l'application théorique pour un bateau de taille n.
Remarque: Cette fonction se trouve dans le fichier *bat_naval.py*, il suffit de lancer ce fichier et de choisir l'option 1 pour avoir les résultats de la fonction.

Question 3)Donner une fonction qui permet de dénombrer le nombre de façon de placer une liste de bateaux sur une grille vide. Calculer le nombre de grilles différentes pour 1, 2 et 3 bateaux. Est-il possible de calculer de cette manière le nombre de grilles pour la liste complète de bateau ?

Les fonctions qui permettent de calculer le nombre de grilles différentes pour les différents bateaux se trouvent dans le fichier *bat_naval.py*, il suffit de lancer ce fichier et de suivre les indications du terminal pour choisir la fonction qu'on désire tester.
Cette façon de calculer le nombre de grilles n'est pas adaptée. En effet on remarque très vite qu'il y a énormément de grilles plus la liste de bateaux augmente et la fonction prend déjà pas mal de temps pour 3 bateaux(environ 277secondes) donc pour une liste de bateaux complète ça serait très coûteux(d'ailleurs la fonction s'arrête à un moment donné, peut être à cause des problèmes de mémoire etc).

Question 4) En considérant toutes les grilles équiprobables, quel est le lien entre le nombre de grilles et la probabilité de tirer une grille donnée ? Donner une fonction qui prend en paramètre une grille, génère des grilles aléatoirement jusqu'à ce que la grille générée soit égale à la grille passée en paramètre et qui renvoie le nombre de grilles générées.

Le lien entre le nombre de grilles et la probabilité d'en tirer une donnée est définie par la relation suivante dans l'univers Ω:
- *événements élémentaires équiprobables:* $P(\omega) = \frac{1}{card(\Omega)} \forall \omega \in \Omega$.
On pose alors *P(G)* comme événement élémentaire le fait de tirer une grille et T étant le nombre total de grilles, ça donne: $P(G) = \frac{1}{T}$.
Remarque: Pour tester cette fonction qui génère des grilles aléatoirement et les compare à une grille donnée il suffit de lancer `bat_naval.py` et de choisir l'option 5 dans le terminal.

Question 5) Donner un algorithme qui permet d'approximer le nombre total de grilles pour une liste de bateaux. Comparer les résultats avec le dénombrement des questions précédentes. Est-ce une bonne manière de procéder pour la liste complète de bateaux ?

On prend une grille initialement vide, et on boucle sur la liste de bateau. À chaque itérations on regarde combien il y a de façons de placer le bateau, puis on le place aléatoirement. Le produit des façons de placer chaque bateau à chaque étape nous donne une approximation du nombre de configurations possibles. Cette méthode est bonne car rapide, et elle peut donc être utilisée sur des (modérément) longues listes de bateaux.

Remarque: Pour tester il faut lancer `bat_naval.py` et choisir l'option 4.

2 Modélisation probabiliste du jeu

2.1 Version aléatoire

- Partie théorique sur l'espérance de la stratégie de tirs aléatoires
Hypothèses:
a)Variables indépendantes et identiquement distribuées.
b)Distribution de probabilité discrète.
c) Les événements sont mutuellement exclusifs.

Nous sommes dans un scénario tel que nous cherchons a avoir le nombre de tirs X réussis pour toucher exactement k cases. On a donc une population finie de N =100 qui est le nombre total de cases et parmi ces cases on a M = 17 qui est le nombre de cases correspondant aux bateaux(on a donc ici M=k pour finir la partie). Suite aux hypothèses émises précédemment, nous avons songé a diverses possibilités de distribution tel que la loi géométrique ou encore la loi binomiale. Cependant, ces dernières ne répondant pas au problème posé, car la géométrique s'arrête au premier succès rencontré, or ici nous avons 17 succès qui sont les 17 coups pour détruire les bateaux, et la binomiale elle ne peut pas correspondre à la bataille navale car on a pas une probabilité de succès fixée. Ainsi, nous avons pensé qu'il s'agissait d'une loi binomiale négative car cette dernière ce définissant comme :

"une expérience consistant en une série de tirages indépendants, donnant un succès avec probabilité p (constante durant toute l'expérience) et un échec avec une probabilité complémentaire." -Loi binomiale négative - Wikipédia

Cela correspondait bien à notre problème à une exception près, nous avons des tirages sans remise et sans contrainte de temps.

Pour conclure, on a compris, qu'ils s'agissait d'une distribution binomiales négative, avec remise, aussi appelée Loi Hyper-géometrique Négative, avec comme fonction densité qui se définit comme:

$f(x) = P(X=x) = \frac{C_{M-k}^{N-x} * C_{k-1}^{x-1}}{C_M^N}$ *pour* $x \in \{k, k+1, ..., k+N-M\}$.

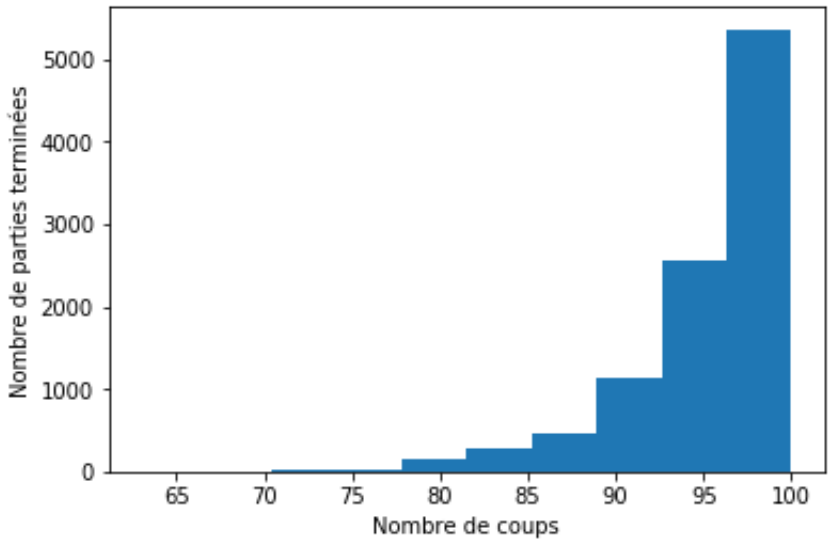
Suite à cela, nous avons ainsi pu définir l'expression de l'espérance car comme nous le savons, l'espérance se définit comme :

Si X est une v.a.r discrète prenant les valeurs $(x)_i \in I$ et si la série:

$\sum_{i \in I} |x_i| P(X = x_i)$ est convergente, alors : $E(x) = \sum_{i \in I}^{n} x_i P(X = x_i)$ Ainsi on en déduit l'expression de l'espérance qui s'exprime comme:
$e = k * \frac{N+1}{M+1}$
Pour le calcul théorique on a donc : $17 * \frac{100+1}{17+1} = 95,39$ coups en moyenne pour finir une partie.

- Code de la version joueur aléatoire et représentation de la distribution:

L'implémentation de la classe RandomPlayer se trouve dans le fichier *strategy.py* et pour la tester il suffit de lancer *theMain.py* et de choisir sur le terminal l'option 1. Cela va calculer le nombre de coups nécessaires en moyenne pour finir une partie et générer les graphiques qui représentent la distribution.
Voici l'histogramme pour le joueur aléatoire (pour 10000 parties):

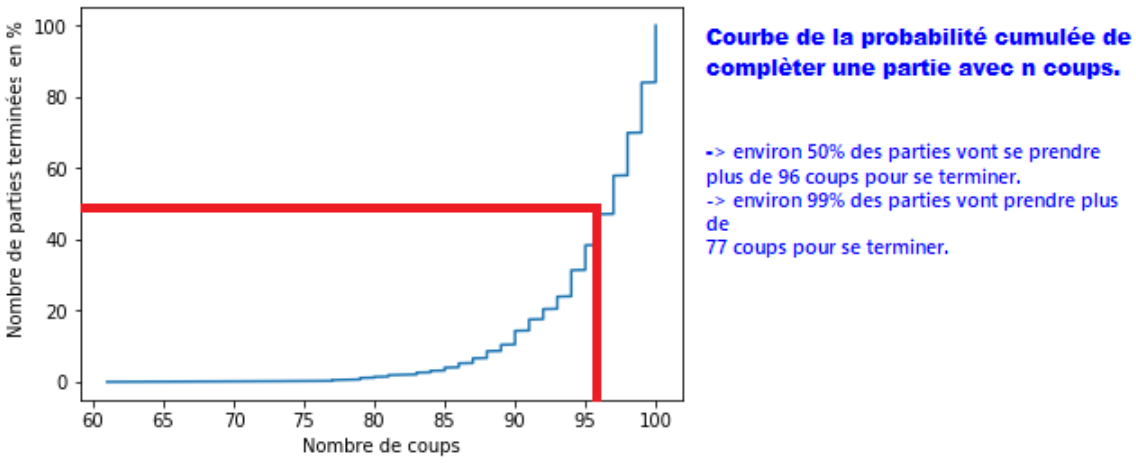


Résultat pratique observé: environ 95 coups en moyenne pour terminer une partie.

- Comparaison théorique/pratique et remarques par rapport aux hypothèses:

En conclusion, il s'avère que nous trouvons bien un résultat similaire, que ce soit suite a l'évaluation théorique ou pratique qui est de approximativement 95 coups, validant bien ainsi le modèle posé.
En ce qui concerne les hypothèses, il y a un problème avec l'indépendance, plus le nombre de parties augmente, plus la médiane va tendre vers une valeur précise du coup les partie ne sont pas "si indépendantes que ça".

Voici la courbe des probabilités cumulées de finir une partie avec n coups aléatoires:

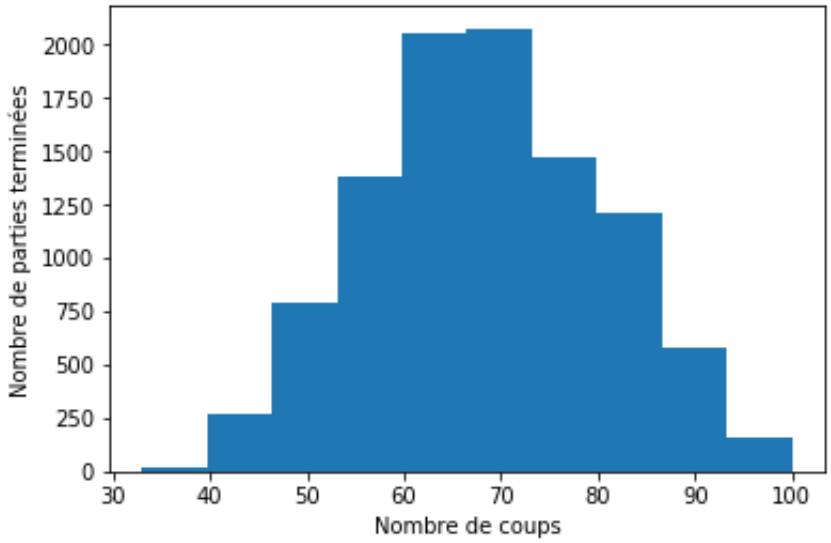


2.2 Version heuristique

La version heuristique consiste à améliorer la version précédente en ajoutant certaines règles. En effet on va jouer en 2 phases, tout d'abord nous allons jouer de manière aléatoire jusqu'à toucher un bateau adverse. Quand c'est le cas nous passons en mode "agressif" et on cherchera à couler le bateau en exploitant les cases connexes a celle touchée aléatoirement.

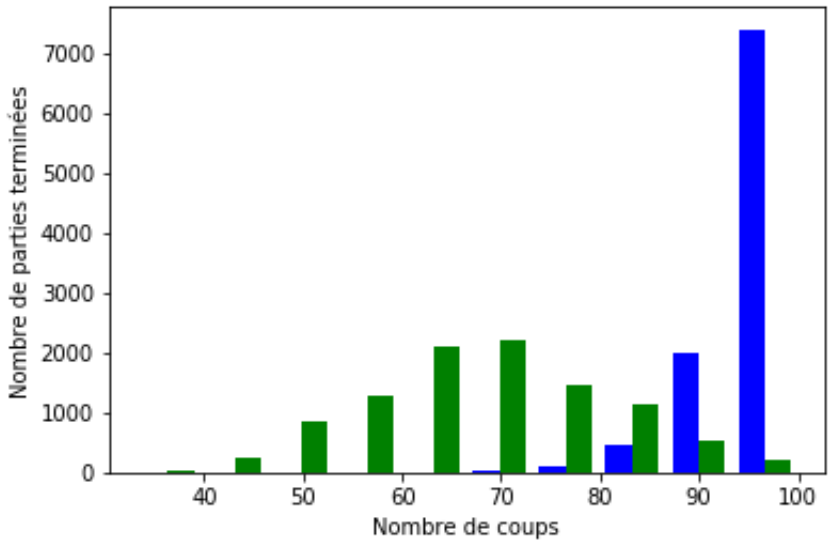
- Code de la version heuristique et représentation de la distribution:

L'implémentation de la classe HeuresticPlayer se trouve dans le fichier *`strategy.py`* et pour la tester il suffit de lancer *`theMain.py`* et de choisir sur le terminal l'option 2. Cela va calculer le nombre de coups nécessaires en moyenne pour finir une partie et générer les graphiques qui représentent la distribution.
Voici l'histogramme pour le joueur heuristique (pour 10000 parties):



Résultat observé: environ 68 coups en moyenne pour terminer une partie.

- Comparaison version heuristique et version version aléatoire:



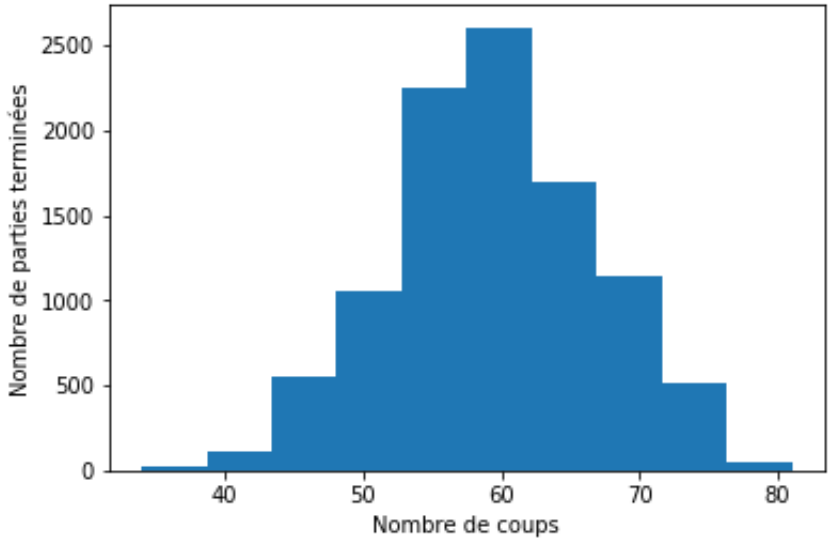
Vert:Heuristique // Bleu: Aléatoire
On voit bien une nette amélioration si on utilise la stratégie heuristique(68 coups en moyenne) si on compare avec la stratégie aléatoire(95 coups en moyenne).

2.3 Version probabiliste simplifiée

Dans cette nouvelle version nous avons cherché à réduire encore une fois le nombre de coups nécessaires pour finir une partie, cette fois en utilisant les probabilités de chaque case de contenir un bateau.
Principe de l'algorithme qui a lui aussi 2 phases(mode recherche probabiliste ou mode "agressif" qui va analyser les cases connexes), a chaque tour:
Tout d'abord on va calculer les probabilités de une case de contenir un bateau, donc pour chaque bateau on va examiner ses positions sur la grille et attribuer un score a chaque case. On choisis la case la plus probable qui répond aux contraintes(qui possède le score le plus important).
Si c'est un bateau qui est touché par le tir,on va changer de phase, en passe en mode "agressif" et on va explorer les cases voisines, comme pour la version heuristique.

- Code de la version probabiliste et représentation de la distribution:

L'implémentation de la classe ProbaPlayer se trouve dans le fichier *strategy.py* et pour la tester il suffit de lancer *theMain.py* et de choisir sur le terminal l'option 3. Cela va calculer le nombre de coups nécessaires en moyenne pour finir une partie et générer les graphiques qui représentent la distribution.
Voici l'histogramme pour le joueur heuristique (pour 10000 parties):



Résultat observé: environ 59 coups en moyenne pour terminer une partie.

- Formalisation et hypothèse d'indépendance fausse:

Par définition la loi jointe est:
"Soit (Ω, P) un espace de probabilité, et soient X et Y deux v.a. sur cet espace, a valeur resp. dans F et G. (X, Y) est une v.a., appelée loi conjointe de X et Y ; les valeurs de (X, Y) sont dans $F * G$."

On peut déduire la loi jointe de X et Y car :

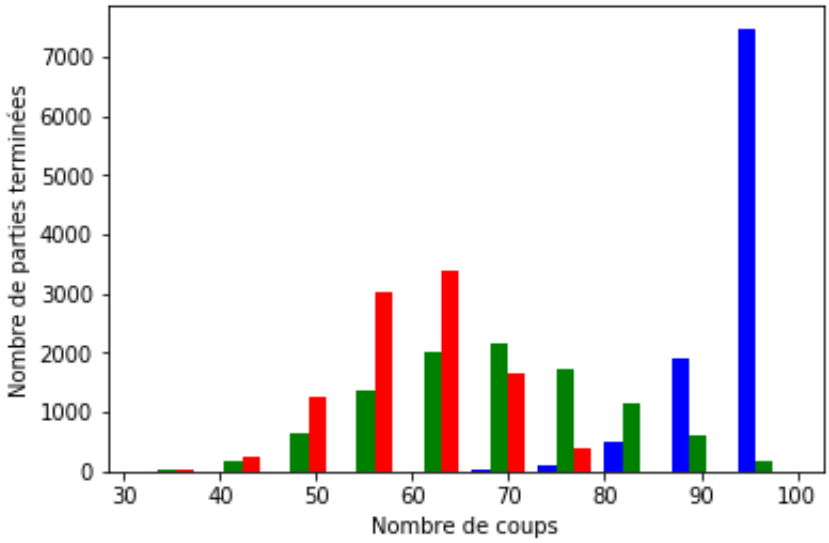
- X est connu
- Les deux paramètres sont indépendants.

Posons X (un entier), qui est la taille du bateau compris entre 2 et 5. On pose Y (un nombre entier aléatoire) qui est le nombre de tirs effectué pour couler le bateau. Y étant entre 2 et 100 (On choisis 2 car c'est le nombre de tir minimal pour détruire un bateau de la plus petite taille).
On pose i une taille de bateau et j un nombre de tir.
$P(X = i; Y = j) = 0$, $\forall j > i$ (on peut pas détruire un grand bateau avec moins de coups que sa taille).
De plus,$P(X = i)$ qui se traduit par la probabilité que le nombre X(la taille du bateau)vaut i.
donc $P(X = i) = \frac{1}{4}$; (car il y a 4 types de taille de bateaux possibles).
En connaissant la taille du bateau, supposés i, on peut ainsi déduire la probabilité conditionnel du nombre de coup nécessaire pour détruire ce bateau par : $P(Y = j|X = i) = \frac{i}{3i+2}$.
Donc on en déduit la loi jointe associé qui est $P(X = i; Y = j) = \frac{i}{4(3i+2)}$ $\forall j <= i$.

L'hypothèse d'indépendance est fausse car nous sommes dans un même univers donc les événements dépendent les uns des autres. En effet les informations qu'on dispose à chaque tour vont évoluer d'un tour à l'autre, par exemple les cases qui sont découvertes, les bateaux qui sont déjà coulés etc, font que les probabilités de toucher une bateau changent au coup suivant.

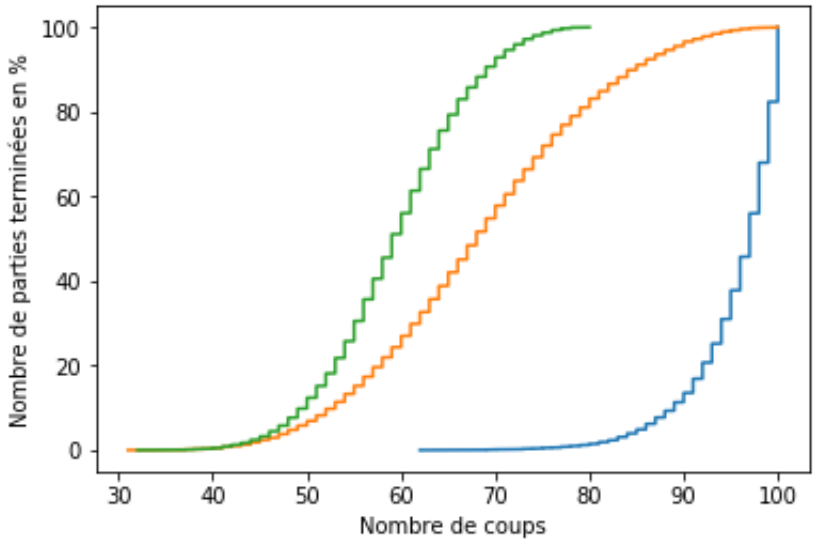
- Comparaisons avec les autres stratégies:

Voici l'histogramme des 3 stratégies(pour 10000 parties):



Vert:Heuristique // Bleu: Aléatoire // Rouge: Probabiliste Simplifiée
La version probabiliste simplifiée est donc la plus optimale, elle gagne en moyenne en 59 coups, tandis que la version heuristique et aléatoire ont respectivement une moyenne de 68 et 95 coups.

Voici maintenant la courbe des probabilités cumulées de compléter une partie en n coups des 3 stratégies:



Vert:Probabiliste Simplifiée // Bleu: Aléatoire // Orange: Heuristique
On remarque que pour la dernière version étudiée on dépasse jamais 79 coups, ce qui montre bien son efficacité.

Conclusion

Pour conclure, on a vu que le jeu de la bataille navale possède de nombreuses stratégies et que avec l'aide des probabilités on peut optimiser nos chances de gagner.
En effet en analysant le plateau minutieusement à chaque tour et on exploitant toutes les informations en notre possession on peut avoir un nette avantage par rapport à une stratégie totalement aléatoire.

En guise d'approfondissement du projet on pourrait également s'intéresser au fait que la méthode probabiliste simplifiée commence par les mêmes coups car on examine toujours les mêmes cases en début de jeu.
On peut aussi s'intéresser à des nouveaux patterns pour les différentes stratégies, comme par exemple examiner des cases par leur parité afin d'améliorer la recherche d'un bateau.
Ainsi qu'a des meilleurs méthodes qui font usage des probabilités(comme Monte-Carlo).