Sorbonne Université

Etude probabiliste du jeu Bataille Navale

Antoine Audras
Ali Benabdallah

Introduction

Le jeu bataille navale est un jeu dans lequel des navires sont placés secrètement dans une grille et le joueur a pour but de couler tous les navires en tirant sur les cases de la grille. Les navires sont coulés lorsque toutes leurs cases ont été touchées. Le score est le nombre de coups joués pour couler tous les bateaux. Plus celui-ci est petit, plus le joueur est performant.

Le jeu se joue sur une grille de dimension 10 cases par 10 cases. Les bateaux sont caractérisés par leur taille :

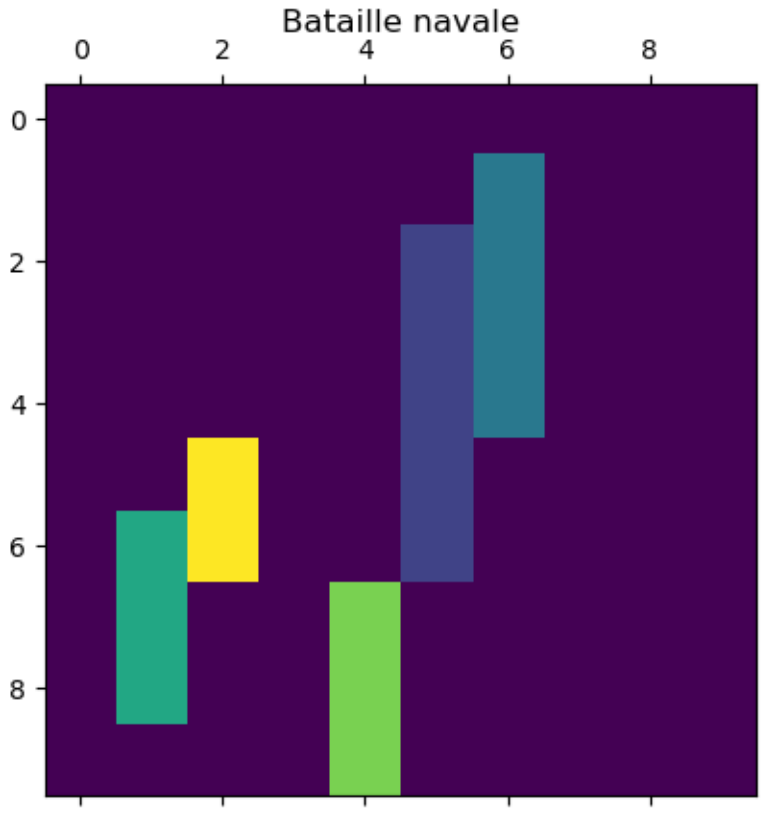
-Un porte-avions (5 cases)

-Un croiseur (4 cases)

-Un contre-torpilleur (3 cases)

-Un sous-marin (3 cases)

-Un torpilleur (2 cases)



Ce jeu fournit une situation de recherche d'objet perdus et les méthodes utilisés dans celui-ci peuvent s'appliquer dans des problèmes réels. L'objectif de ce projet est d'étudier le jeu de bataille navale d'un point de vue probabiliste pour arriver à optimiser nos chances de gagner. Dans une première partie, nous étudierons la combinatoire du jeu puis nous proposerons un modèle pour optimiser les chances de gagner. Enfin nous traiterons un problème réel, celui de la recherche du sous-marin USS Scorpion de la marine américaine perdu en mer.

Partie 1 : Combinatoire du jeu

Nous nous intéressons d'abord à l'aspect combinatoire du jeu. Pour cela, il sera nécessaire dans un premier temps de calculer une borne supérieure du nombre de combinaisons puis de fournir une approche exacte grâce à des algorithmes dénombrant le nombre de combinaisons si l'on place une liste de bateau. Enfin nous essaieront d'approcher le nombre de combinaisons par des méthodes statistiques.

1) Calcul d'une borne supérieure du nombre de combinaisons

Nous souhaitons dans un premier temps donner une borne supérieure simple du nombre de configurations possibles pour la liste complète de bateaux sur une grille de taille 10. Essayons d'abord de décomposer les paramètres du problème.

La grille est de taille 10 x 10 = 100 cases. Nous devons placer 5 bateaux de tailles 5, 4, 3, 3 et 2 cases soit au total 17 cases. Les cases d'un même bateau doivent être adjacentes et les tirages ne sont pas indépendants. Essayons de calculer une borne supérieure simple.

Faisons l'hypothèse que les cases d'un même bateau n'ont pas à être adjacentes. Ainsi, nous avons 17 cases à placer parmi 100 ce qui correspond à :

100! / ((100-17)! * 17!)

= 100!/(83!*17!)

Soit environ $6.65 * 10^{18}$ combinaisons ! Cette hypothèse est bien entendue fausse mais nous permet d'avoir une première borne supérieure à notre nombre de combinaisons.

Oublions cette hypothèse mais considérons que les tirages sont indépendants. Soit t la taille d'un bateau. On peut placer sur une seule ligne ou colonne 11-t bateaux. Sachant qu'il y a 10 lignes et 10 colonnes, il y a 20*(11-t) combinaisons par bateau. Ainsi, on peut calculer le nombre total de combinaisons :

$20^5 *(11-5) * (11-4) * (11-3)^2 * (11-2)$

= 77414400000 combinaisons !

Cette borne supérieure est nettement inférieure à la précédente et donc bien plus précise. Néanmoins, nous négligeons toujours le fait que les placements des bateaux ne sont pas indépendants. En effet, si le premier bateau fait une taille t, le second perdra au minimum 2t possibilités de placements. Cela correspond au fait que t cases sont occupées et que ces t cases auraient pu être l'une des extrémités du bateau suivant. L'hypothèse est renforcée par le fait qu'on place les bateaux du plus grand au plus petit. Ainsi, une dernière borne serait :

20 *(11-5)*(20*(11-4) – 2*5)*(20*(11-3)-2*(5+4))*(20*(11-3)-2*(5+4+3))*(20*(11-2)-2*(5+4+3+3))

= 45190080000 combinaisons !

2) Approche exacte

1. Nombre de combinaisons pour un seul bateau sur une grille vide

Nous codons d'abord une fonction **nb_pos** capable de calculer le nombre de façons de placer un seul bateau sur une grille vide. Nous l'avons implémenté de façon à ce qu'elle soit utile sur tout type de grille. Celle-ci va parcourir chaque ligne et chaque colonne pour compter chaque façon de poser un bateau.

Observations :

Bateau	Taille	Combinaisons
Porte-avions	5	120
Croiseur	4	140
Contre-torpilleur	3	160
Sous-marin	3	160
Torpilleur	2	200

Tableau des tests de nb_pos sur une grille vide pour chaque bateau

Les résultats concordent à la théorie, nous avons pour chacun 20*(11-t) façons de placer un bateau.

2. Nombre de combinaisons pour une liste de bateaux sur une liste vide

Nous voulons maintenant coder une fonction **nb_pos_list** capable de placer une liste de n bateaux sur une grille vide. Pour cela, nous procédons par récurrence pour générer toutes les grilles de n-1 bateaux et appelons notre fonction nb_pos pour compter le nombre de façons de placer le dernier bateau.

Observations :

Liste de bateaux	Combinaisons
[1]	120
[1,2]	14808
[1,2,3]	1980954

Tableau des résultats de nb_pos_list sur une grille initialement vide pour des listes de 1 à 3 bateaux

La fonction renvoie exactement le nombre de combinaisons requises pour des listes de petites tailles mais nous voyons bien que le nombre de combinaisons ne cesse d'augmenter. Pour chaque combinaison, on génère un très grand nombre de grilles équivalent au nombre de combinaisons pour une liste amputée du dernier bateau. Une fois que la liste de bateaux passe une taille de 3, la fonction entraine un dépassement de la pile. Elle n'est donc pas appropriée pour dénombrer le nombre de combinaisons pour une liste de 5 bateaux. Il nous faut donc tenter une nouvelle approche.

3) Approche approximative

1. Méthode des grilles

S'il est impossible de donner le nombre de configurations exactes, il est possible d'approximer nos résultats avec une certaine justesse. Nous tenterons donc plusieurs approches. La première sera de générer des grilles aléatoires jusqu'à retomber sur notre grille d'origine en comptant le nombre de grilles générées. Ainsi, en implémentant un algorithme effectuant plusieurs fois cette opération et faisant la moyenne des résultats ainsi trouvés, nous pourrions obtenir une approximation du nombre total de grilles pour une liste de bateaux donnée.

Observations :

Liste de bateaux	n	Combinaisons
[1]	10000	119.5
[1,2]	100	14446.7

[1,2,3]	5	987832

Tableaux des résultats approximant le nombre de bateaux par génération de grilles pour des listes de 1 à 3 bateaux

Il était nécessaire de réduire notre n pour une liste de 3 bateaux car le temps de traitement est très long cependant la justesse du résultat dépend de n. En effet, il est nécessaire d'avoir un grand échantillon pour se rapprocher de la valeur exacte, comme pour la liste d'un seul ou de deux bateaux. Dans ces deux cas, on trouve des valeurs très proches de la réalité.

Dans cet algorithme, on génère autant de grilles qu'il y a de combinaisons cependant on les détruit au fur et à mesure alors il n'y a pas de dépassement de la pile mais le temps de traitement est très long. Pour des grilles de taille supérieure ou égale à 3, cette méthode pas appropriée. Il est donc nécessaire d'en creuser d'autres.

2. Moyennes employant nb_pos et nb_pos_list

Nous avons donc défini 2 nouvelles méthodes pour approcher le nombre de combinaisons qui emploient nos fonctions nb_pos et nb_pos_list pour approcher le nombre exact de combinaisons. Cette fois-ci nous traiterons directement des listes de 5 bateaux.

Notre première méthode emploie uniquement nb_pos :

Calcule la moyenne sur n itérations du nombre de combinaisons possibles en suivant ces différentes étapes.

Calcule, pour chaque bateau, le nombre nb de façons de le poser dans une grille initialement vide

Le place aléatoirement dans celle-ci.

La multiplication des nb donne le nombre de combinaisons possibles pour une seule grille.

Répète cette opération n fois

Retourne la moyenne des résultats

Notre seconde méthode emploie nb_pos_list :

Calcule la moyenne sur n itérations du nombre de combinaisons possibles.

On génère aléatoirement n grilles avec les 3 premiers bateaux

On calcule exactement n fois le nombre de combinaisons des 2 bateaux restants sur ces grilles pour trouver la valeur moyenne.

Retourne la multiplication de cette moyenne par le nombre de combinaisons des 3 premiers bateaux sur une grille vide.

Ces deux méthodes sont efficaces car elles fournissent des moyennes ainsi plus n sera grand et plus les résultats tendront vers la valeur exacte du nombre de combinaisons possibles. De plus, elles ont des temps d'exécution acceptables pour des n très grands.

Observations :

n	nb_pos	Nb_pos_list
10	**39445834932.0**	**35286535506.6**
100	**35538025468.8**	**35479500235.7**

Tableaux des résultats approximant le nombre de bateaux des méthodes approximatives employant nb_pos et nb_pos_list pour des listes de 5 bateaux

Nous observons que n n'a pas a être très grand pour notre version employant nb_pos_list. En revanche, pour l'autre version, c'est nécessaire car l'écart-type est bien plus grand. On note aussi que les deux versions tendent environ vers 35000000000 quand n est grand. On peut donc supposer que 35000000000 est une bonne estimation de la borne supérieure du nombre de combinaisons des 5 bateaux.

Partie 2 : Modélisation probabiliste du jeu

Nous souhaitons implémenter un algorithme efficace pour gagner une partie de Bataille Navale en un minimum de coup. Dans un premier temps, nous nous modéliserons une version où chaque coup est tiré aléatoirement. Dans un second temps, nous verrons ce qu'il en est pour une version heuristique du jeu. Enfin, nous opterons pour une méthode probabiliste.

1) Version aléatoire

On note $X \in \mathbb{N}$ la variable aléatoire correspondant au nombre de coups joués avant de couler tous les bateaux si chaque coup est joué aléatoirement.

- On suppose dans un premier temps que les coups sont indépendants et qu'il y a remise.

Décomposons notre variable aléatoire pour l'étudier avec des lois simples. Dans ce cas, il y a au départ 17 cases où se trouvent un bateau et 83 vides. On pose $Y_p \in \mathbb{N}$ la variable aléatoire correspondant au nombre de coups nécessaires avant de toucher une case du bateau. La variable Y_p suit une loi géométrique de paramètre 0.17. Son espérance est de 1/0.17=5.9. Il faut donc environ 6 coups avant de toucher une case d'un bateau.

On souhaite ensuite toucher une autre case, il y a donc 16 cases possibles. Y_p suit cette fois ci une loi géométrique de paramètre 0.16 et d'espérance 1/0.16=6.25.

On peut donc approximer le nombre de coups nécessaires, soit l'espérance de X par la formule suivante :

$$E(X) = \sum_{k=1}^{17} 1/(\frac{k}{100}) = 344$$

- On suppose maintenant qu'il y a remise

Cette fois-ci, il est nécessaire de se référer à une loi hypergéométrique $\mathcal{H}(\mathrm{n},0.17,100)$. En effet :

$$Pn(Y = 17) = \frac{\binom{83}{n-17}}{\binom{100}{n}}$$

Cette probabilité se traduit : J'ai P(Y=17) chances de gagner la partie si je tire n coup. La variable aléatoire Y représentant le nombre de bons tirages ne nous intéresse pas. Nous souhaitons étudier le nombre de coups nécessaires pour gagner. P(X=k) se calcule donc en cherchant mes chances de gagner si je tire n coups en soustrayant mes chances de gagner si je tire n-1 coups. X suit donc la loi suivante :

$$P(X = n) = Pn(Y = 17) - P_{n-1}(Y = 17) = \frac{\binom{83}{n-17}}{\binom{100}{n}} - \frac{\binom{83}{n-18}}{\binom{100}{n-1}}$$

On vérifie que c'est bien une distribution car la somme des P(X) vaut 1 :

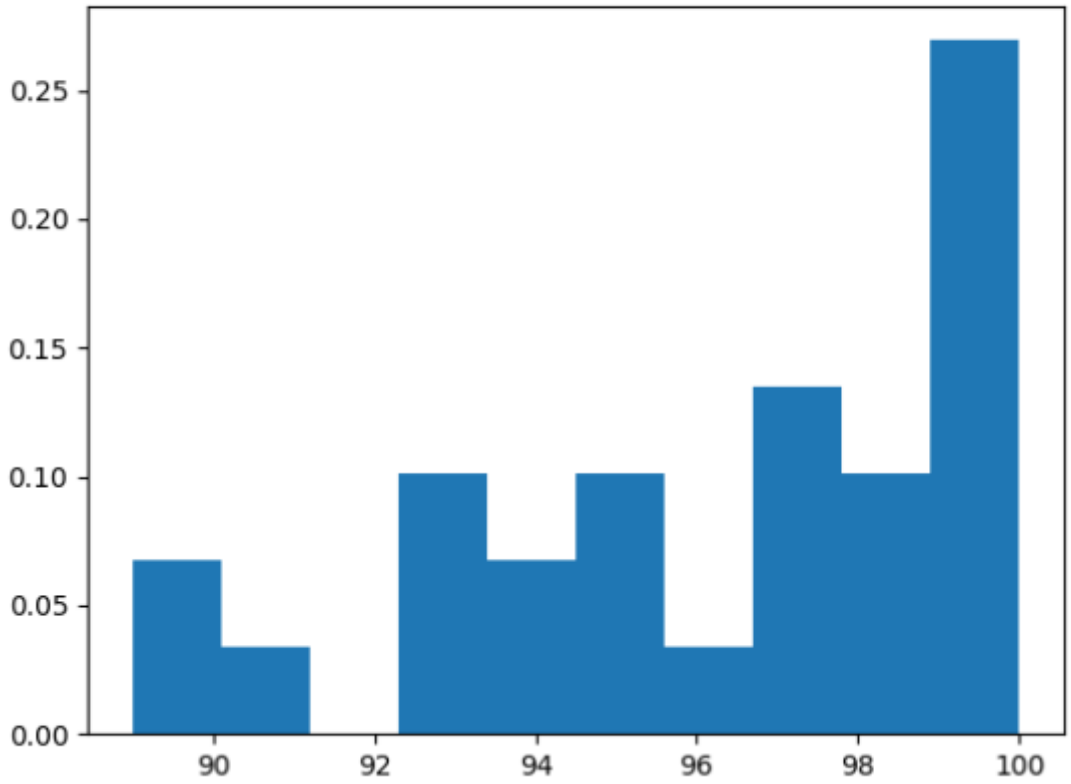
$$\sum_{17}^{100} P(X = n) = \sum_{n=17}^{100} \frac{\binom{83}{n-17}}{\binom{100}{n}} - \frac{\binom{83}{n-18}}{\binom{100}{n-1}} = 1$$

Pour calculer l'espérance de X, on utilise la formule suivante :

$$E(X) = \sum_{n=17}^{100} n * \left(\frac{\binom{83}{n-17}}{\binom{100}{n}} - \frac{\binom{83}{n-18}}{\binom{100}{n-1}}\right) = 95.38$$

Il faut donc en moyenne 95.38 coups pour gagner une partie si on joue totalement aléatoirement.

En implémentant ce modèle aléatoire et en calculant une moyenne sur 30 essais, nous obtenons la distribution suivante.

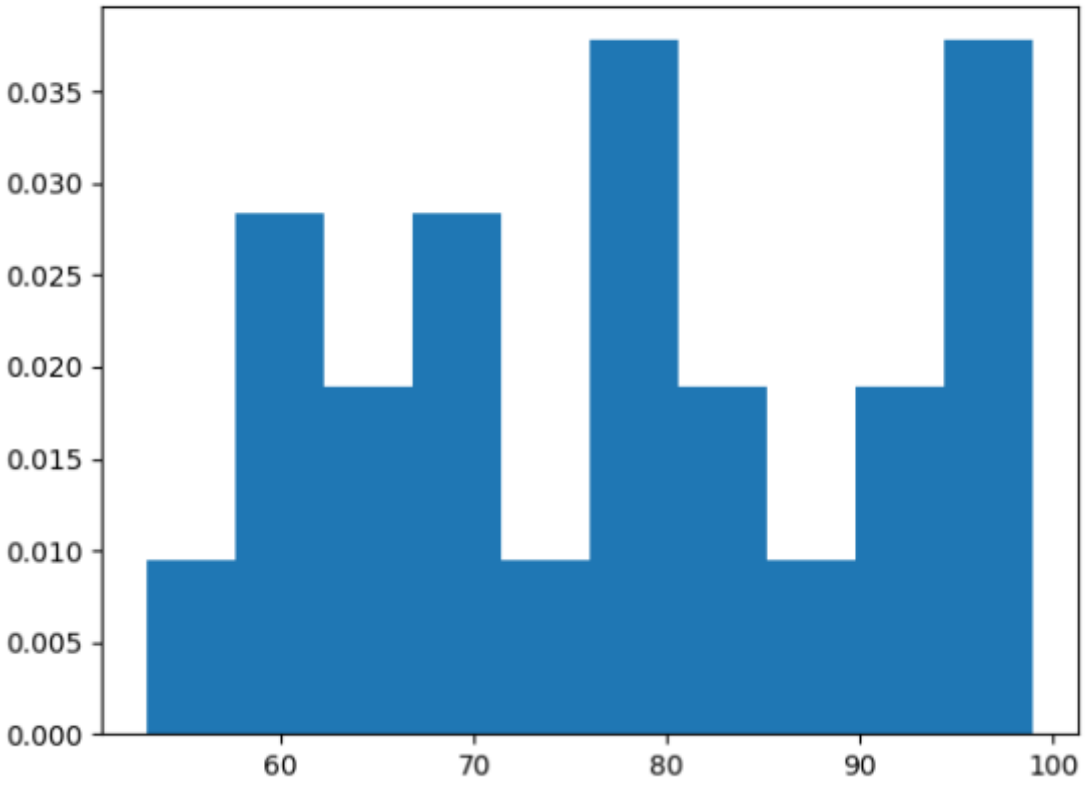


Histogramme représentant la distribution de la variable aléatoire X pour 30 essais de la version aléatoire

L'espérance pratique est de 96.24 coups. Cette valeur est très proche de l'espérance théorique calculée car les hypothèses formulées sont pertinentes. En revanche, on est très loin des 344 coups de l'hypothèse précédentes. En effet, celle-ci ne prenait pas en compte le fait qu'il n'y a pas de remise.

2) Version heuristique

Nous fournissons maintenant un modèle heuristique du jeu. Précédemment, lorsque l'on touchait une case du tableau, l'algorithme ne regardait pas autour de celle-ci pour terminer de détruire le bateau. Cette fois-ci, lorsque la case d'un tableau est trouvée, notre algorithme va parcourir les cases adjacentes jusqu'à détruire toutes les cases d'un même bateau. Nous obtenons cette distribution :

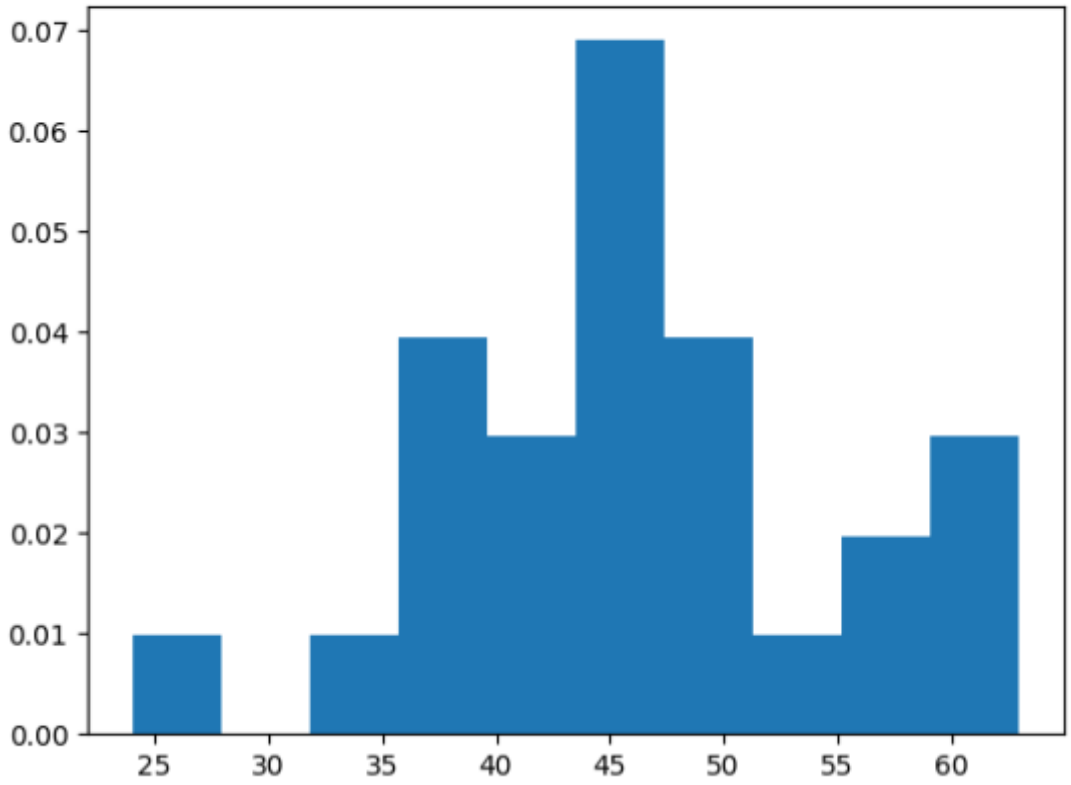


Histogramme représentant la distribution de la variable aléatoire X pour 30 essais de la version heuristique

L'espérance est cette fois-ci de 77.4 coups. Le score moyen s'est donc amélioré. L'algorithme heuristique est plus efficace que l'algorithme aléatoire.

3) Version probabiliste

Nous allons maintenant fournir un modèle probabiliste du jeu. Le modèle précédent ne tenait pas compte du type de bateaux de restant et de l'admissibilité de leur positionnement. Chaque coup nous renseigne sur une position possible ou impossible d'un bateau. On emploiera dans ce modèle une matrice de probabilité pour chaque bateau et une matrice de probabilité totale correspondant à la somme des matrices de chaque bateau multiplié par un coefficient. En effet, plus le bateau est grand et plus il y a de chances de le trouver. Il est toutefois hors de question de calculer la nouvelle distribution jointe sur tous les bateaux car les bateaux sont considérés indépendants. La loi jointe nécessiterait de calculer l'ensemble des combinaisons possibles pour tous les bateaux en les considérants dépendant. Cette opération est bien trop couteuse pour être effectuée. L'hypothèse d'indépendance est toutefois fausse car deux bateaux ne peuvent pas partager une même case et donc le nombre de combinaisons possibles pour placer un bateau sur la grille quand celle-ci contient déjà des bateaux est inférieure au nombre de combinaisons possibles si celle-ci est vide.



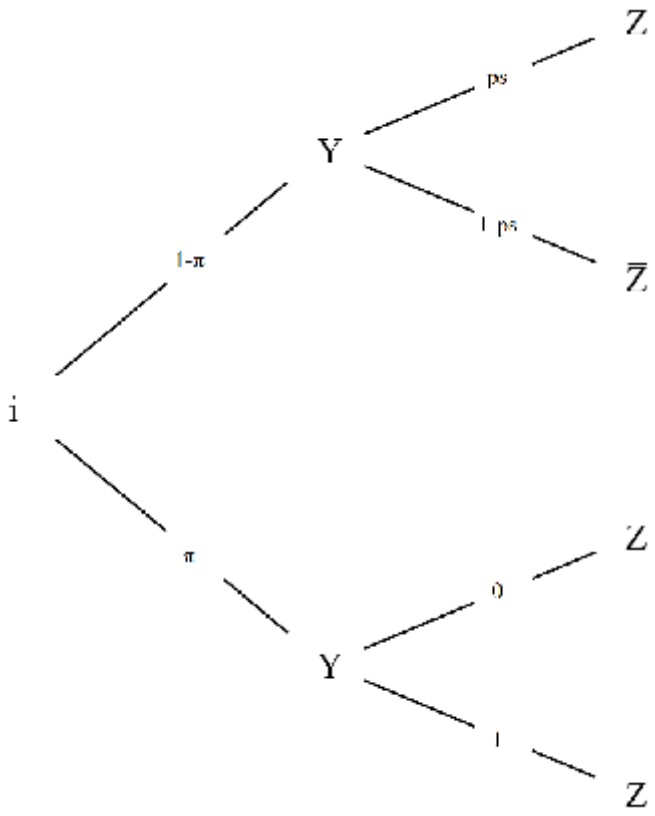
Histogramme représentant la distribution de la variable aléatoire X pour 30 essais de la version probabiliste

On trouve cette fois-ci une espérance de 45.9. Le score moyen s'est encore amélioré. Ce modèle est le plus appropriée pour obtenir un bon score au jeu de bataille navale.

PARTIE 3 : Application à un problème réel, la recherche d'objets perdus

Le jeu de bataille navale nous fournit des méthodes efficaces de recherche d'objet perdus. En 1968, le sous-marin USS Scorpion s'est perdu en mer. Pour le retrouver, il a fallu employer une méthode Bayésienne. Les seules différences sont que cette fois-ci, on utilise un senseur imparfait pour détecter le sous-marin et que le sous-marin ne fait qu'une seule case. Pour sa recherche, le modèle suivant a été proposé :

- L'espace de recherche est divisé en un quadrillage de N cellules numérotées de 1 à N
- Chaque case a une probabilité à priori π_i de contenir le sous-marin
- On note $Y_i \in \{0, 1\}$ la variable aléatoire qui vaut 1 pour la case où se trouve le sous-marin sinon 0. La loi de Y_i est donc :
  - $P(Y_i = 1) = \pi_i$
  - $P(Y_i = 0) = 1 - \pi_i$
- On note $Z_i \in \{0, 1\}$ la variable aléatoire qui vaut 1 en cas de détection, 0 sinon. Il n'y a détection possible que si $P(Yi = 1)$ donc Z_i et Y_i sont dépendantes. La loi de $Z_i \mid Y_i$ est :
  - $P(Z_i = 0 \mid Y_i = 0) = 1$
  - $P(Z_i = 1 \mid Y_i = 0) = 0$
  - $P(Z_i = 0 \mid Y_i = 1) = 1 - p_s$
  - $P(Z_i = 1 \mid Y_i = 1) = p_s$
- On suppose que les réponses aux sondages des cases sont indépendantes et identiquement distribuées



Arbre de probabilité de l'expérience de recherche du senseur à la i -ème case

Nous cherchons à implémenter un algorithme pour la recherche de l'objet perdu. Bien évidemment, nous ne savons pas où il se situe donc si pour une case k, $Z_k = 0$, nous ne pouvons pas savoir si c'est parce que l'objet ne se situe pas dedans ou bien si c'est le senseur qui a échoué. Nous supposerons donc à chaque fois que c'est le senseur qui l'a échoué. La probabilité de cet évènement est :

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

$= P(Y_k = 1).P(Z_k = 0|Y_k = 1)$ *D'après la formule de bayes*

$= \pi_k (1-p_s)$

Assez logiquement, à chaque fois que l'on vérifie une case, la probabilité que l'objet se trouve à l'intérieur est diminuée. En effet, il se peut que le senseur se trompe mais il ne faut pas oublier qu'il y a une probabilité que l'objet ne s'y trouve pas. Il faut donc mettre à jour π_k. Cela revient donc à calculer :

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

$= P(Y_k = 1, Z_k = 0) / P(Z_k = 0)$ *D'après la formule de Bayes*

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

Donc :

$P(Y_k = 1 \mid Z_k = 0) = (\pi_k (1-p_s)) / (1 - \pi_k.p_s)$

Bien évidemment, la distribution de Y doit être telle que la somme de tous les π_i doit valoir 1. Ainsi, si π_k diminue alors le poids perdu par celui-ci doit être réparti aux autres π_i. C'est une répartition relative : plus π_i est grand et plus il recevra une grande part du poids de π_k. Le problème revient donc à mettre à jour π_i sachant que Z_k est nul et donc à calculer la probabilité suivante :

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

$= P(Y_i = 1, Z_k = 0) / P(Z_k = 0)$ *D'après la formule de Bayes*

$= P(Y_i = 1). P(Z_k = 0 \mid Y_i = 1) / P(Z_k = 0)$ *D'après la formule de Bayes*

$= \pi_i / (1 - \pi_k.p_s)$

Pour implémenter un algorithme qui résout ce problème, il nous faut générer une matrice de probabilité dont nous choisissons la distribution initiale. Nous proposons une distribution centrée, en bordure, aléatoire ou avec un flanc gauche défavorisé. Il s'agit ensuite de définir les coordonnées du sous-marin placé aléatoirement selon les valeurs de notre matrice de probabilité puis de sonder aléatoirement les cases avec une probabilité p_s que le senseur ne se trompe pas en sondant la bonne case. S'il ne trouve pas la bonne case, il met à jour la matrice de probabilité selon les valeurs calculées précédemment. Sinon, l'algorithme retourne le nombre de fois que nous avons cherchés.

Observations :

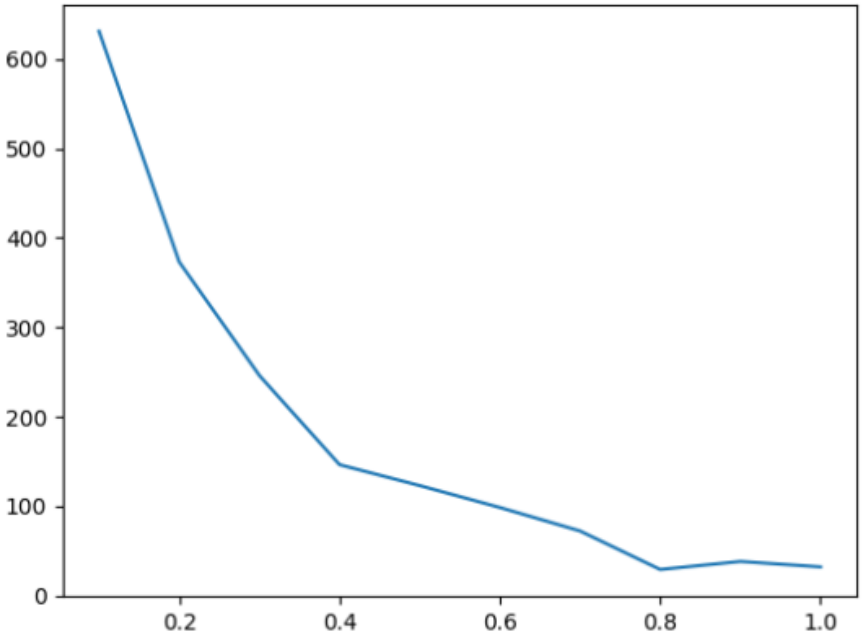
Nous allons faire varier dans ces tests deux facteurs :

- La fiabilité du senseur p_s
- La répartition

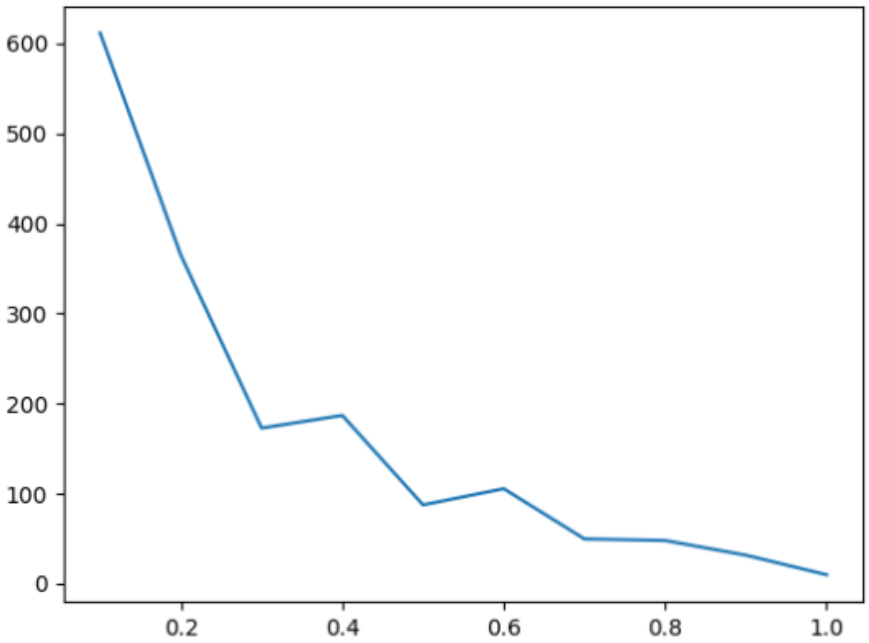
La dimension de la matrice de test sera fixe et sera de 10 lignes sur 10 colonnes.

Nous effectuerons à chaque fois 30 essais pour une fiabilité donnée et représenterons le nombre moyens d'essais pour chacune des répartitions.

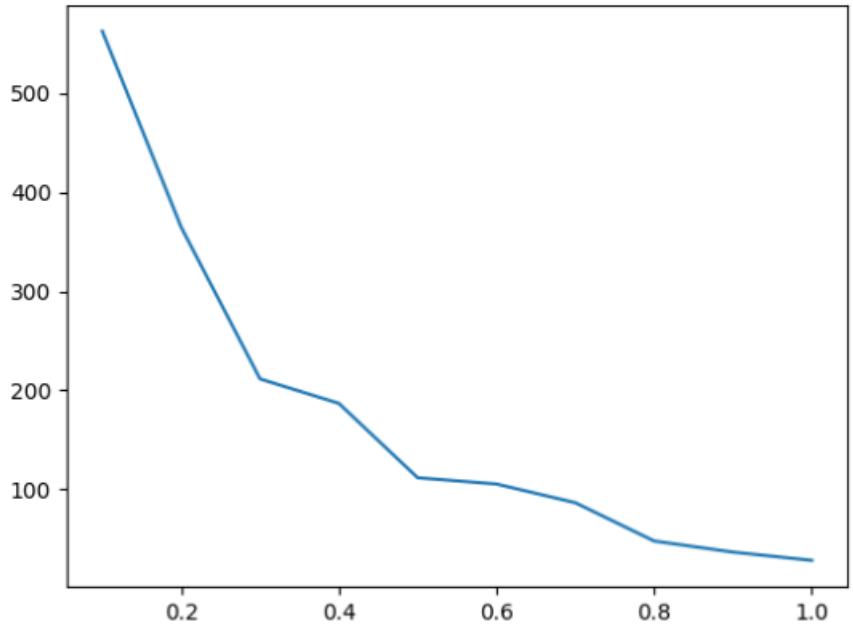
Nous devrions observer que plus le senseur est fiable et plus le nombre d'itérations sera faible. En effet, en augmentant la fiabilité, on devrait retomber dans le modèle probabiliste de l'exercice précédent avec un seul bateau d'une seule case.



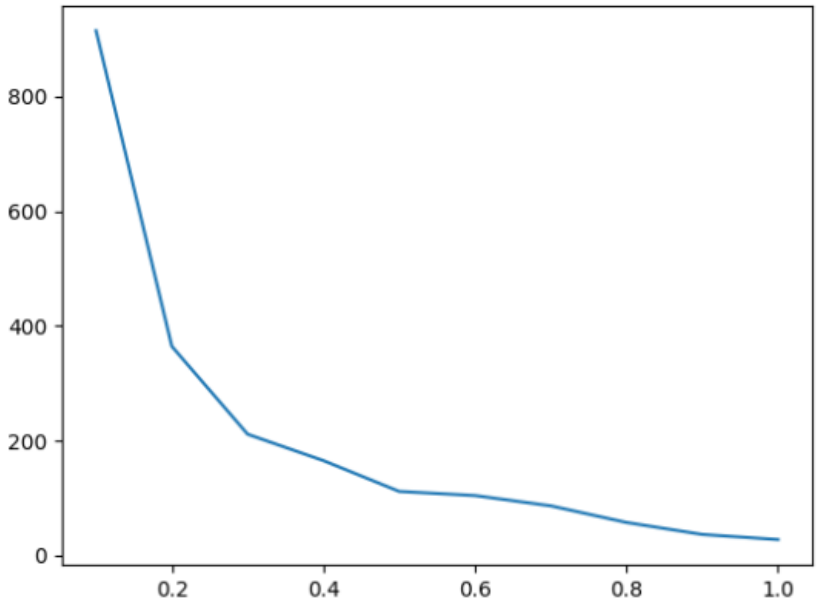
Nombre d'itérations en fonction de la probabilité p_s pour une répartition centrée



Nombre d'itérations en fonction de la probabilité p_s pour une répartition en bordure



Nombre d'itérations en fonction de la probabilité p_s pour une répartition en flanc droit



Nombre d'itérations en fonction de la probabilité p_s pour une répartition aléatoire

On remarque que les courbes ont toutes la même allure. Plus la fiabilité est élevée et plus on se ramène au cas du modèle probabiliste du jeu de bataille navale avec un bateau d'une seule case. En

revanche, la répartition peut changer légèrement la pente des courbes et le nombre d'essais. On remarque que le nombre d'essais quand le flanc gauche est défavorisé est plus petit car les dimensions de la grille sont artificiellement réduites et que le nombre d'essais en aléatoire est plus grand.

Conclusion :

Dans ce projet, nous avons d'abord porté un regard sur l'aspect combinatoire du jeu de bataille navale. Nous avons pu constater tout d'abord qu'il est difficile d'obtenir la borne supérieure exacte du nombre de combinaisons possibles mais qu'il est possible de l'approcher par des méthodes approximatives. Nous avons ensuite proposé divers algorithmes jouant au jeu selon des méthodes plus ou moins efficaces. Nous avons mesuré leur efficacité en comparant leur espérance. Notre méthode la plus efficace était celle employant une approche probabiliste. Enfin nous avons traité un cas réel, celui de la recherche d'objet perdu avec un facteur supplémentaire : le senseur était imparfait. Nous avons pu constater dans ce cas que la fiabilité du senseur était importante pour réduire le nombre d'essais et que la distribution de la probabilité de trouver l'objet dans une case était tout aussi importante. Les méthodes employées dans notre modèle probabiliste du jeu bataille navale ont été réutilisées dans ce cas réel. Elles nous soulignent l'importance d'identifier les différents paramètres importants pour résoudre un problème concret mais aussi de savoir poser les bonnes hypothèses et en éliminer d'autres pour traiter efficacement notre problème. Il serait intéressant de reproduire la recherche d'un objet perdu dans des conditions réelles pour éventuellement identifier d'autres facteurs qui pourraient influencer nos résultats.