Projet 1 : Bataille navale

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.
Les fonctions programmées ont pour nom le numéro des questions auxquelles elles repondent.

Q1. Afin de déterminer une borne supérieure simple du nombre de configurations possibles pour la liste complète de bateaux sur une grille de taille 10, nous faisons l'hypothèse que les bateaux sont superposables (nous ne retirons pas les cases que nous occupons).
Pour chaque bateau, on a (10 - taillebateau +1) configurations possibles par ligne et (10 - taillebateau +1) configurations possibles par colonne. On a alors pour chaque bateau 2x10x(10 - taillebateau +1) configurations possibles => $32 \times 10^5 \times (6 \times 7 \times 8 \times 9) = 96\,768 \times 10^5$ configurations possibles.

Q2. La fonction q2 permet de calculer le nombre de façons de placer un bateau donné sur une grille vide. Les résultats attendus sont les mêmes que les résultats obtenus, tests pour des tailles différentes:
q2(3)=160
q2(5)=120

Q3.liste_bateau_grille(10,[4]) renvoie 160
liste_bateau_grille(10,[4,5]) renvoie 28212
liste_bateau_grille(10,[4,5,1]) renvoie 3037336
La complexité de cette fonction est exponentielle car elle doit calculer toute les configurations : pour l'appel liste_bateau_grille(10,[4,5,1]). Pour calculer toutes les grilles, la fonction tournerait pendant un temps extrêmement long. Donc on ne peut pas calculer ainsi toutes les grilles de la liste.

Q4.Le lien entre le nombre de grilles n et la probabilité de tirer une grille donnée est 1/n. Cela nous permet d'avoir une approximation du nombre total de grilles avec notre fonction. Exemple; on exécute
liste_bateau=[4,5]
print(q4(genere_grille(liste_bateau),liste_bateau))
Le résultat est 20812 qui est assez proche du nombre de grilles total 28212.Cela ne semble pas être une bonne solution car le résultat peut varier très fortement avec la réalité : avec la liste [4,5,1] on a comme résultat 1 503 467 au lieu de 3 037 336. De plus, le temps de calcul des 2 fonctions est assez similaire.

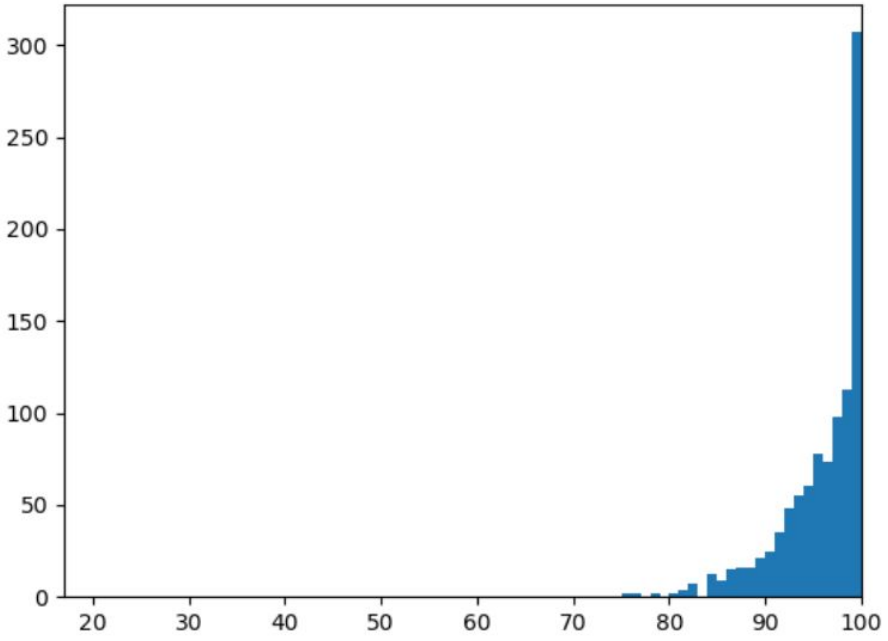
Q5.Pour approximer le nombre total de grilles on exécute la fonction précédente avec le liste de tous les bateaux. Cela nous donne une approximation du nombre total de grille

Q6.Pour approximer plus justement le nombre de configurations, on doit prendre en compte la superposition des bateaux: il faudrait réduire le nombre de cases disponibles en fonction de la taille du bateau placé précédemment.

3 Modélisation probabiliste du jeu

Version aléatoire

On suppose que le nombre de cases total et de 100 et que le nombre de cases bateau est de 17. Le joueur tire des cases jusqu'à couler tous les bateaux (il ne s'arrête qu'une fois les 17 cases bateau coulées), on note y le nombre de cases jouées. De plus, nous faisons l'hypothèse de ne pas rejouer 2 fois sur une même case déjà tirée.
Dans le meilleur cas (nombre de coup le plus faible), on fait 17 coups où on ne touche que des bateaux:on a alors une combinaison de 17 cases bateau parmi 17 sur une combinaison de 17 coups parmi 100.
Dans le cas où on ne touche pas de bateaux sur un coup mais qu'on les touche sur les autres coups (donc 18 coups), on a une combinaison de 17 cases bateau parmi 17, une combinaison d'une case parmi 83 sur une combinaison de 18 coups parmi 100.
Dans le cas où on ne touche pas de bateaux sur deux coup mais qu'on les touche sur les autres coups (donc 19 coups), on a une combinaison de 17 cases bateau parmi 17, de deux cases parmi 83 sur une combinaison de 19 coups parmi 100.
Pour y cases jouées, nous avons 17 cases bateau parmi 17 et y-17 autres cases parmi 83 sur une combinaison de 17 y coups parmi 100.
Comme la combinaison de 17 parmi 17 renvoie 1, on crée une fonction proba_alea(y) qui retourne le nombre de possibilités des y-17 autres cases parmi 83 sur celui des y coups parmi 100.
Afin de déterminer l'espérance, on crée la fonction esperance().
Celle-ci renvoie une espérance de 94,68 coups

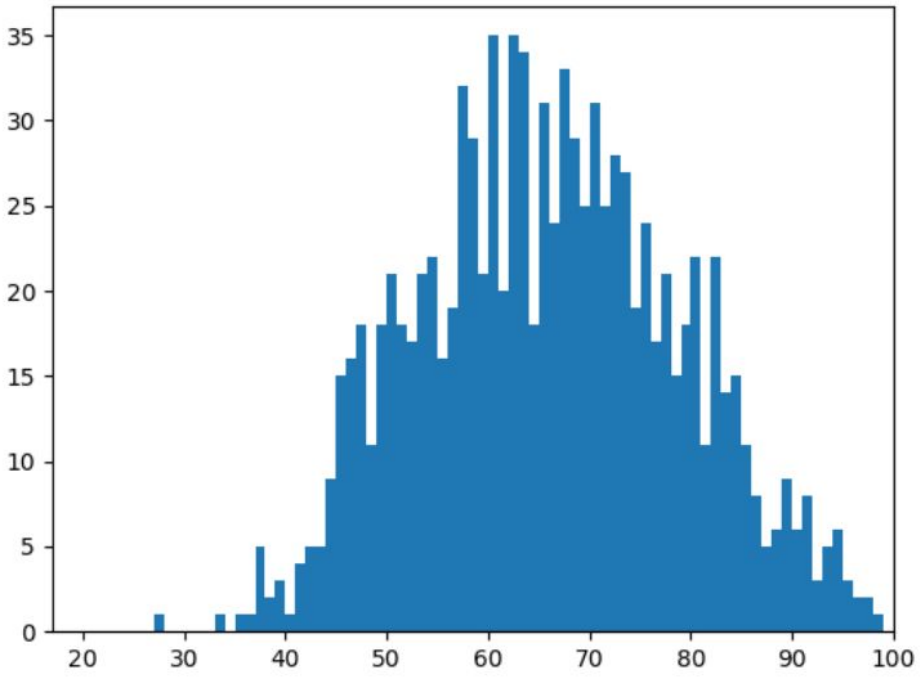


Graphique de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie aléatoire

On remarque dans l'histogramme représentant la distribution de probabilité après 1000 essais que la très grande majorité des parties ont entre 90 et 100 coups avec ce modèle avec environ 1/3 des parties qui ont 100 coups. Cette distribution correspond bien à l'espérance calculée qui est de 95 coups.

Version heuristique

On remarque dans l'histogramme représentant la distribution de probabilité après 1000 essais que les parties avec entre 60 et 70 coups sont bien plus nombreuses que les autres ce qui confirme bien l'espérance calculée avec python qui est de 66 coup avec ce modèle. On a donc bien une stratégie plus efficace que la méthode aléatoire qui a une espérance de 95 coups.

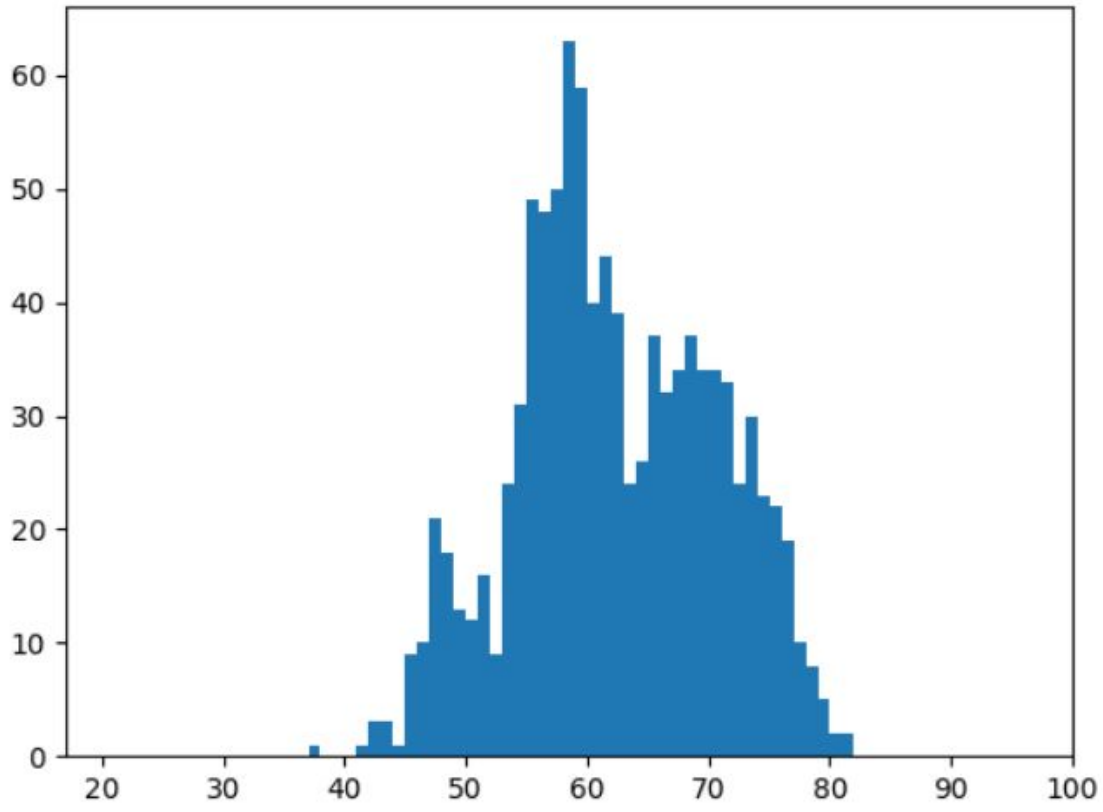


Graphique de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie heuristique

Version probabiliste simplifiée

On ne va pas calculer à chaque tour la distribution jointe des bateaux car cela prendrait beaucoup trop de temps de calcul. Avec une loi jointe, il faudrait calculer toutes les positions possibles des bateaux restants multiplié par le nombre de coups, on aurait un temps de calcul global extrêmement long. Donc la méthode qu'on va utiliser est une loi d'indépendance entre les bateaux ce qui va permettre de calculer la distribution de probabilité des bateaux sur les cases indépendamment les uns des autres c'est à dire en ne tenant pas compte de la position potentielle des autres bateaux.
L'hypothèse d'indépendance est fausse car pour calculer la probabilité d'un bateau, on ne tient pas compte de la position des autres bateaux, or la position d'un bateau influe sur la position des autres: 2 bateaux ne peuvent pas avoir de cases en commun.

L'implémentation sera la suivante: On a une grille 10*10 semblable à la grille représentant la partie mais chaque case représente la probabilité de contenir un bateau. Tant qu'on a pas gagné :
Pour chaque case non visitée de la grille, on calcule sa probabilité : le nombre de fois que la case apparaît dans les positions possibles des bateaux divisé par le nombre des positions des bateaux.
A chaque tour on tire sur la case [x,y] qui a la plus grande probabilité de cacher un bateau. Si cette case renvoie l'identifiant d'un bateau, on modifie ainsi les positions possibles des bateaux.Ensuite on parcourt toute les cases en calculant les probabilités de présence de chaque bateau.
On a l'histogramme de distribution de probabilité suivant après 1000 essais:



Graphique de la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie probabiliste

On a une espérance de 61 coups environ ce qui est plus efficace que la version heuristique.
Une amélioration serait d'utiliser la méthode probabiliste jusqu'à ce qu'on tire sur un bateau, ensuite on passe en modèle heuristique. Quand on a tiré sur toutes les cases du bateau, on repasse en modèle probabiliste.

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

1- La loi de Yi est une loi de Bernoulli:

$\forall$ y $\in$ {0,1}, P(Yi =y) $(\pi_i)^y . (1-\pi_i)^{1-y}$=

et la loi de P(Zi|Yi) :

P(Zi=0 | Yi=0)= 1 P(Zi=0 | Yi=1)= 1-ps
P(Zi=1 | Yi=0)= 0 P(Zi=1 | Yi=1)= ps

2- Cet événement a une probabilité de $P(Z_k = 0 \cap Y_k = 1)$

3- $P(Z_k = 0 \cap Y_k = 1)$ = P(Zk =0 | Yk =1) x P(Yk = 1) Or, P(Yk = 1) = π_k

D'après 1 : P(Zk =0 | Yk =1)= 1-ps on a donc $P(Z_k = 0 \cap Y_k = 1) = (1-p_s).\ \pi_k$

Si le senseur n'a rien détecté dans la case k alors on met à jour π_k :
Le nouveau π_k est égal à P(Yk =1 | Zk =0) = $\frac{P(Z_k=0 \cap Y_k=1)}{P(Z_k=0)}$

$P(Z_k = 0) = P(Z_k = 0 \cap Y_k = 1) + P(Z_k = 0 \cap Y_k = 0)$
$= (1-ps).\ \pi_k + P(Z_k = 0 \mid Y_k = 0).\ P(Y_k = 0)$
$= (1-ps).\pi_k + 1.(1-\pi_k) = 1 - ps.\pi_k \ Donc\ P(Z_k = 0) = 1 - ps.\pi_k$

$Donc\ \frac{P(Z_k=0 \cap Y_k=1)}{P(Z_k=0)} = \frac{(1-ps).\pi_k}{1-ps.\pi_k}$

$Donc\ si\ le\ senseur\ ne\ détecte\ rien :\ \pi_{knew} = \frac{(1-ps).\pi_k}{1-ps.\pi_k}$

4- La probabilité de cet événement est $P(Y_i = 1 \mid Z_k = 0)_{i \neq k}$

$= \frac{P(Y_i=1 \cap Z_k=0)}{P(Z_k=0)} = P(Z_k = 0 \mid Y_i = 1).\ \frac{P(Y_i=1)}{P(Z_k=0)}$

or, (Zk=0 | Yi=1) est l'événement où le senseur ne détecte rien sur k et que l'objet est sur une case différente i. Donc P(Zk=0 | Yi=1)=1 car le senseur ne va toujours rien détecter sur k
et $P(Y_i = 1) = \pi_i\ et\ P(Z_k = 0) = 1 - ps.\pi_k$

$Donc\ P(Y_i = 1 \mid Z_k = 0) = \frac{\pi_i}{1-ps.\pi_k}$

$Donc\ \pi_{i\ new} = \frac{\pi_i}{1-ps.\pi_k}$

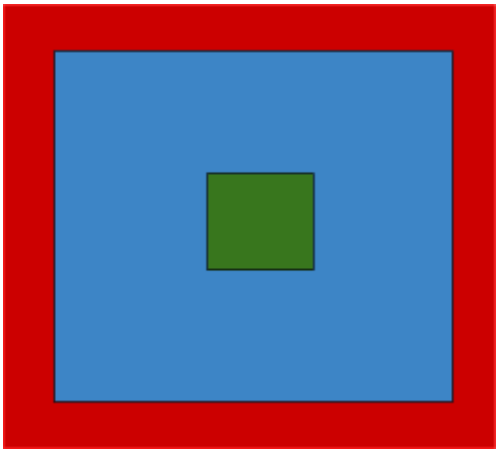
L'algorithme pour rechercher un objet perdu est le suivant:

Tant qu'on a pas trouvé l'objet:

On repère la case avec la plus grande probabilité
On teste cette case
Si on trouve l'objet->fin
Sinon on met à jour les probabilités des cases

Pour implémenter cet algorithme, on prend une grille de forme carré de 100 cellules (10*10).

1ere grille : Cette grille s'apparente à un lac, l'objet perdu à plus de chances de se trouver près de la berge qu'au centre, on a donc la distribution suivante:

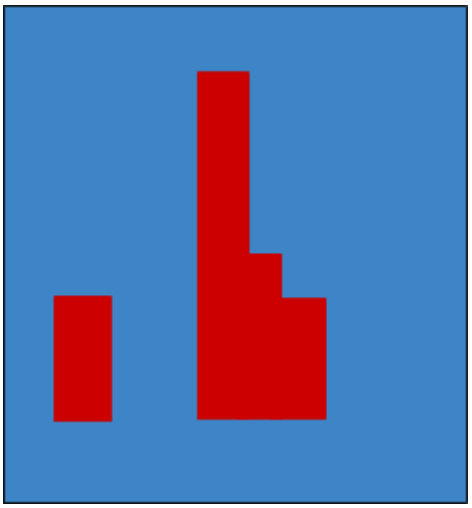


Avec 60% de chances que l'objet se trouve dans la zone rouge
39% de chances que l'objet se trouve dans la zone bleue
1% de chances que l'objet se trouve dans la zone verte

Donc à l'initialisation, chaque case de la zone rouge a une probabilité de 0,5/30=1/60
chaque case de la zone bleue a une probabilité de 0,39/60=13/2000
chaque case de la zone verte a une probabilité de 0,01/4=1/400

On voit bien que lorsqu'on met l'objet sur la zone bleue, l'algorithme parcourt d'abord la zone rouge en entier avant de chercher dans la zone bleue. Si l'objet est en [8,5] (zone bleue) avec ps=0.9 on parcourt le plus souvent (avec une proba de 0.9), 93 cases avant de trouver l'objet (zone rouge puis zone bleu) et sinon on peut parcourir 180 cases (ou plus) car le senseur n'a pas détecté l'objet la première fois donc il doit parcourir au moins une partie de la grille.

2e grille: cette grille a une distribution moins «simpliste» que la 1ere:



Avec 70% de chances que l'objet se trouve dans la zone rouge
30% de chances que l'objet se trouve dans la zone bleue.

Donc à l'initialisation,
chaque case de la zone rouge a une probabilité de 0,7/12=7/120
chaque case de la zone bleue a une probabilité de 0,3/88=3/880

Comme dans la 1ere grille on voit que l'algorithme va d'abord chercher dans les cases rouges avant de regarder dans les cases bleues.
Si l'objet est en [5,5] avec ps=0.9 on parcourt le plus souvent (avec une proba de 0.9), 9 cases avant de trouver l'objet et sinon on peut parcourir 23 cases (ou plus) car le senseur n'a pas détecté l'objet la première fois donc il doit parcourir la zone rouge qui a gardé une probabilité supérieure à celle de la zone bleue.

Bonus Monte Carlo:

Cette version ne marche pas : après plusieurs coup le programme tourne pour calculer les grilles aléatoires respectant les contraintes mais ne renvoie rien.
Néanmoins la fonction grille_aleatoire_rec marche d'après les tests effectués dans le main et le fait qu'elle calcule bien des grilles aléatoires avec des contraintes pour les premiers coups.
Partie_Monte_Carlo qui utilise cette fonction marche aussi d'après les tests fait avec les «print».
On en conclut qu' avec un grand nombres de contraintes à respecter grille_aleatoire_rec a peu de chances de trouver aléatoirement une grille qui les respecte toutes donc la fonction prend beaucoup de temps à calculer d'où le temps extrêmement long de calcul.
On a essayé d'améliorer grille_aleatoire_rec en ajoutant un cas où lorsqu'un bateau a été coulé, pour ce bateau il faut trouver une position dont toutes les cases sont dans liste_tir mais il y a du y avoir un problème dans l'implémentation de cette amélioration car la fonction continue de tourner sans rien trouver.