SCIENCES SORBONNE UNIVERSITÉ

Projet 1 : Bataille navale

Nasser CHAKER et Bruce ROSE

16 mars 2020

Introduction

1 Modélisation et fonctions simples

2 Combinatoire du jeu

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 disposons d'un échiquier de 10 par 10 cases, ainsi que de 5 bateaux qui ensemble comptabilisent $5+4+3+3+2=17$ cases. Une manière simple d'obtenir une borne supérieure du nombre de configurations possibles pour la liste complète de bateaux consiste à placer les 17 cases parmi 100, qui se traduit par un arrangement A_{100}^{17}. Ce résultat est supérieur au nombre de configurations réel puisqu'elle ignore que les cases constituant un bateau doivent être adjacentes.

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.

En théorie, sur une ligne, un bateau de taille n donné dispose de $10-n+1$ configurations possibles. Etant donné que la grille contient 10 lignes et qu'un bateau peut être soit horizontal soit vertical, il existe donc $2\times 10\times(10-n+1)$ façons de positionner un bateau de taille n sur la grille entière.

Notre fonction `pos_un_bateau` prend en entrée une grille et le numéro d'un bateau et renvoie le nombre de possibilités de placer ce bateau sur cette grille. Elle passe en revue chaque case de la grille et appelle la fonction `peut_placer` qui renvoie `vrai` si le bateau peut être placé avec pour origine cette case. Cette dernière prend en compte la

longueur du bateau, les bordures ainsi que les bateaux éventuellement déjà présents sur la grille. La table 1 présente nos résultats.

Résultats \ $n=$	2	3	4	5
théoriques	180	160	140	120
obtenus	180	160	140	120

TABLE 1 : Comparaison du nombre de façons de placer un bateau de longueur n calculé de manière théorique et expérimentale

Donner une fonction qui permet de dénombrer le nombre de façons 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 ?

Pour calculer le nombre de façons de placer une liste de bateaux sur une grille vide, nous avons conçu la fonction récursive `pos_des_bateaux` qui prend en paramètre une liste de numéros de bateaux. La fonction procède ainsi : elle place le premier bateau à la première position viable sur une grille vide au départ. Puis, le second est posé à sa première position viable, etc, jusqu'à ce que tous les bateaux aient été placés. À ce moment là, le dernier bateau est placé à sa seconde position viable, puis à sa troisième jusqu'à ce qu'il ait épuisé toutes ses positions. L'avant dernier bateau est alors posé à sa deuxième position viable et on recommence à poser le dernier bateau à toutes ses positions viables comme précédemment. On réitère l'opération jusqu'à ce que tous les bateaux aient épuisés leurs positions viables. On renvoie finalement le nombre de grilles possibles contenant tous les bateaux rencontrées pendant l'énumération.

Fonction \ Nombre de bateaux	1 $n = 2$	2 $n_1 = 2, n_2 = 3$	3 $n_1 = 2, n_2 = n_3 = 3$
`pos_des_bateaux`	180	27336	> 27336

TABLE 2 : Résultats renvoyés par la fonction `pos_des_bateaux` avec 1, 2 et 3 bateaux

Dans la mesure où le calcul du nombre de grilles avec 3 bateaux prend déjà trop de temps pour pouvoir l'ajouter au tableau, il n'est pas judicieux de faire de même avec la liste complète de bateaux.

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.

La probabilité uniforme sur un univers fini Ω est définie par la fonction de masse $p(\omega)=\frac{1}{card(\Omega)}$. Par conséquent, tout événement E de Ω a pour probabilité $P(E)=\frac{card(E)}{card(\Omega)}$. En posant E l'événement « tirer une grille donnée », qui est un événement élémentaire, on obtient $P(E)=\frac{card(E)}{card(\Omega)}=\frac{1}{\text{le nombre de grilles}}$.

Soit la fonction `grilleAleaEgale`. Elle génère des grilles aléatoires avec la liste de bateaux passée en paramètre et s'arrête quand l'une d'elle est égale à celle passée en paramètre. Finalement, elle renvoie le nombre de grilles générées. Pour tester cette fonction, nous avons implémenté une fonction `test_grilleAleaEgale` qui pour une durée donnée, itère autant que possible sur la fonction `grilleAleaEgale` pour renvoyer au final une moyenne des résultats obtenus. Cette fonction de test nous permet de tester la fonction `grilleAleaEgale` en un temps raisonnable.

Nombre de bateaux (n : taille du bateau)	`test_grilleAleaEgale`	
	Nb d'itérations possibles en 15s	Moyenne des nombres de grilles retournés
1 ($n = 2$)	2 997	182
2 ($n_1 = 2,\ n_2 = 3$)	16	29 566
3 ($n_3 = 3$)	2	1 744 521
4 ($n_4 = 4$)	/	/
5 ($n_5 = 5$)	/	/

TABLE 3 : Nombre d'itérations effectuées en 15 secondes et moyenne de l'ensemble des résultats renvoyés sur les itérations en fonction du nombre de bateaux passés en paramètre à la fonction `test_grilleAleaEgale`.

Malheureusement, la fonction `test_grilleAleaEgale` appliquée à plus de trois bateaux termine en un temps indéterminé supérieur à 15 minutes. Par conséquent, nous n'avons pas pu ajouter les résultats correspondants dans le tableau.

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 ?

Soit la fonction `approx_nbGrille2` qui prend en paramètre une liste (de numéros) de bateaux et qui utilise la fonction `place_alea` qui prend en paramètre une grille et un bateau et place ce dernier à une position aléatoire dans la grille.

```
approx_nbGrille2
    Entrée :
                listNum, liste de numéros de bateaux.
    Sortie :
                Une approximation du nombre de grilles différentes possibles
                contenant la liste de bateaux passée en paramètre.
   res ⇐ 1
   grilleCour est une grille vide
   Pour chaque bateau b de listNum
       res ⇐ res × pos_un_bateau(grilleCour, b)
       place_alea(grilleCour, b)

   retourner res
```

(bonus) Proposer des solutions pour approximer plus justement le nombre de configurations.

3 Modélisation probabiliste du jeu

3.1 Version aléatoire

En faisant des hypothèses que vous préciserez, quelle est l'espérance du nombre de coups joués avant de couler tous les bateaux si on tire aléatoirement chaque coup ?

Soit notre grille contenant $N = 100$ cases, dont m sont des bateaux. Les autres sont la mer et sont au nombre de $N - m$. On tire alors un échantillon de k cases. On calcule d'abord le nombre de combinaisons correspondant à n cases bateaux en multipliant le nombre de possibilités de tirage de n cases bateaux parmi m par le nombre de possibilités de tirage du reste, soit $k - n$ cases mer parmi $100 - m$. On divise ensuite ce nombre de possibilités par le nombre total de tirages. La probabilité d'obtenir n cases bateaux est par conséquent donnée par une loi hypergéométrique. Si on appelle X le nombre de cases bateaux tirées, la probabilité d'en avoir n s'écrit $\mathbb{P}(X = n)$ et vaut :

$$p(X = n) = \frac{\binom{m}{n}\binom{N-m}{k-n}}{\binom{N}{k}} = \frac{\binom{17}{n}\binom{83}{k-n}}{\binom{100}{k}}$$

De cette formule on déduit la probabilité d'avoir au moins 17 cases bateaux parmi k tirées qui vaut donc :

$$p\left(X \leq k\right) = \frac{\binom{17}{17}\binom{83}{k-17}}{\binom{100}{k}} = \frac{\binom{83}{k-17}}{\binom{100}{k}}$$

Ainsi, la probabilité d'avoir besoin d'exactement k coups pour obtenir les 17 cases bateaux est donnée par la simple soustraction

$$p\left(X = k\right) = p\left(X \leq k\right) - p\left(X \leq k-1\right)$$

sachant que $p\left(X \leq 16\right) = 0$ (impossible d'obtenir les 17 cases bateaux en jouant moins de 17 coups) et que $p\left(X > 100\right) = 0$ (impossible de jouer plus de 100 coups).

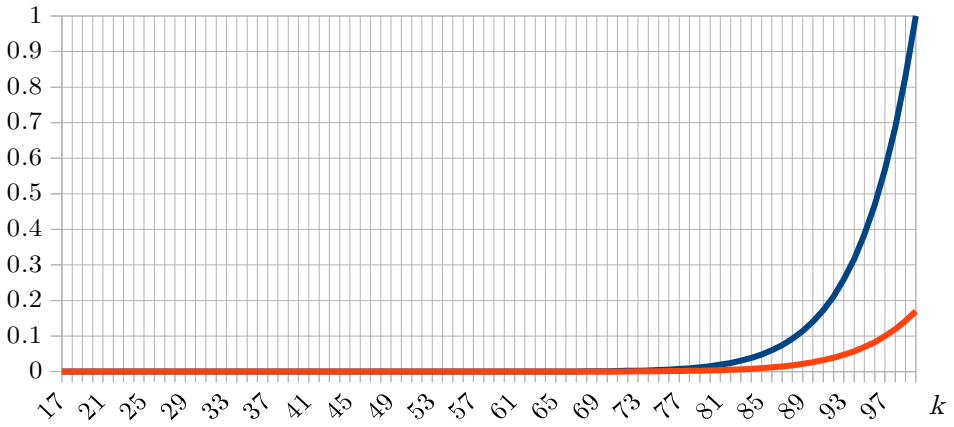

FIGURE 1 : En bleu, probabilité $P\left(X \leq k\right)$; en rouge, probabilité $P\left(X = k\right)$

On calcule ainsi une espérance égale à $\simeq 95{,}39$. On peut espérer gagner une partie de bataille navale en 95 tours environ en moyenne si on joue au hasard.

Programmer une fonction qui joue aléatoirement au jeu : une grille aléatoire est tirée, chaque coup du jeu est ensuite tiré aléatoirement (on pourra tout de même éliminer les positions déjà jouées) jusqu'à ce que tous les bateaux soient touchés. Votre fonction devra renvoyer le nombre de coups utilisés pour terminer le jeu.

Nous avons conçu une classe `JoueurAlea` qui contient une méthode `joue(self, bataille)`. `bataille` est un objet de classe `Bataille` qui contient un attribut `grille` et un attribut `vies`, ainsi qu'une méthode `joue(self, position)`, `victoire(self)`, `affiche(self)`, et `checkbound(self, position)`.

La méthode joue de `JoueurAlea` a pour but de décider d'une position `(x, y)` à jouer à un tour donné. Comme la stratégie du `JoueurAlea` est de tirer aléatoirement, cette méthode se contente de générer une position au hasard qu'elle retourne.

Comment calculer la distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie ? Tracer le graphique de la distribution. Comparer à l'espérance théorique précédemment calculée. Que remarquez vous par rapport aux hypothèses formulées ? Y'a-t-il une justification ?

Pour calculer l'espérance de la variable aléatoire correspondant au nombre de coups pour terminer une partie, nous lançons 1000 parties de bataille navale avec `JoueurAlea`, puis nous divisons la somme des nombres de coups joués par 1000. Nous obtenons un résultat égal à $\simeq 95{,}48$, très proche du résultat de notre calcul théorique.

En ce qui concerne la distribution, nous lançons 1000 parties et enregistrons dans un tableau le nombre de victoires en fonction du nombre de coups.

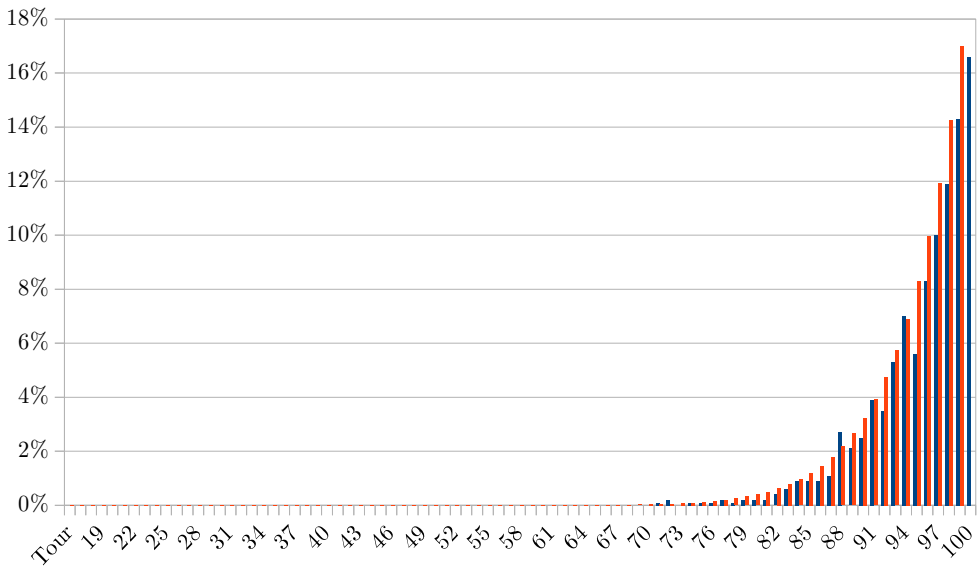

FIGURE 2 : Distribution de la variable aléatoire correspondant au nombre de coups pour terminer une partie. En bleu : résultats expérimentaux. En rouge : résultats théoriques.

3.2 Version heuristique

La version aléatoire n'exploite pas les coups victorieux précédents, elle continue à explorer aléatoirement tout l'échiquier. Proposer une version heuristique (à base de règles) composée de deux comportements : un comportement aléatoire tant que rien n'est touché, un comportement qui va explorer les cases connexes lorsqu'un coup touche un bateau. Comparer les résultats et tracer le graphique de la distribution de la variable aléatoire.

Nous avons conçu une classe `JoueurHeuristique`. Ce joueur tient à jour une liste de coups réussis et une liste de coups à suivre. Initialement, ces deux listes sont vides. Quand il est amené à jouer, si la liste *prochains_coups* est vide, le joueur tire un coup aléatoirement. Sinon, il joue la première case de *prochains_coups* et la retire de la liste. S'il touche un bateau, cette case touchée est ajoutée à *coups_reussis* et si l'une des cases adjacentes est dans *coups_reussis*, la case adjacente contraire est ajoutée à *prochains_coups*. Si la case touchée n'a pas de case adjacente dans *coups_reussis*, alors toutes les cases adjacentes sont ajoutées à *prochains_coups*.

Exemple

Tour 1 : initialement, *coups_reussis* = `[]` et *prochains_coups* = `[]`.

Comme *prochains_coups* est vide, le joueur tire au hasard `(5, 4)`. Touché ! *coups_reussis* = `[(5, 4)]`. Aucune case adjacente n'est dans *coups_reussis*, par conséquent elles sont toutes ajoutées à *prochains_coups*.

Tour 2 :

coups_reussis = `[(5, 4)]`,

prochains_coups = `[(5, 3), (4, 4), (5, 5), (6, 4)]`.

Le joueur tire `(5, 3)`. Touché ! Comme `(5, 4)` est dans *coups_reussis*, on s'aperçoit qu'il y a de grande chances pour qu'on ait touché un bateau vertical; `(5, 2)` est ajouté à *prochains_coups*.

etc.

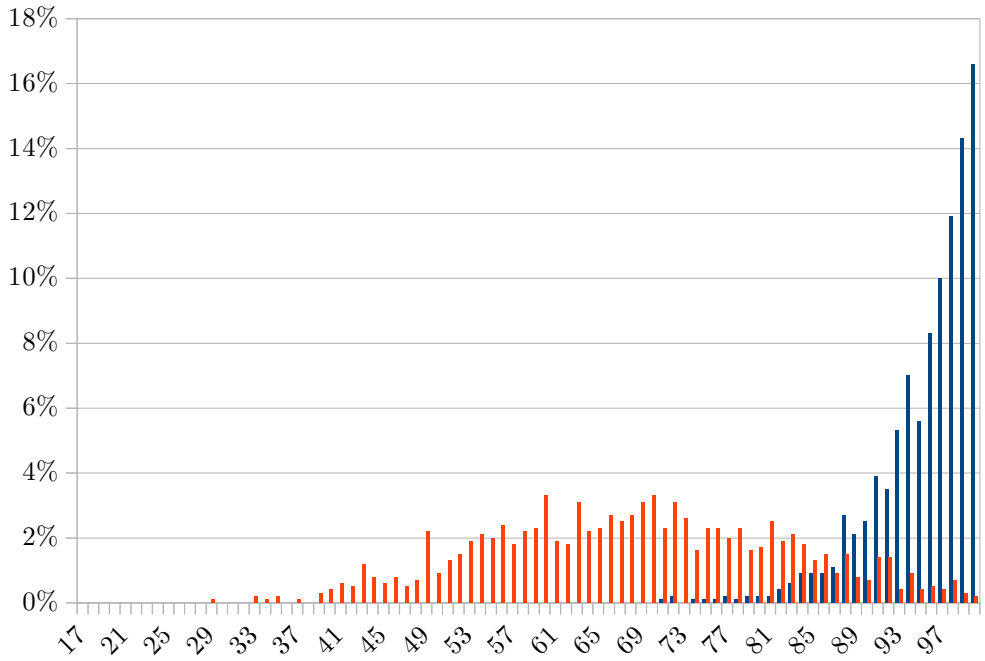

FIGURE 3 : Distribution du nombre de coups pour terminer une partie. En rouge, résultats expérimentaux obtenus avec la stratégie heuristique. En bleu, les résultats espérimentaux obtenus avec la stratégie aléatoire.

3.3 Version probabiliste simplifiée