Projet 1 : Bataille navale

Résumé

Ce petit projet a comme objectif d'étudier le jeu de la bataille navale d'un point de vue probabiliste. Il s'agira dans un premier temps d'étudier la combinatoire du jeu, puis de proposer une modélisation du jeu afin d'optimiser les chances d'un joueur de gagner, et enfin d'étudier une variante du jeu plus réaliste.

Le jeu de la bataille navale se joue sur une grille de 10 cases par 10 cases. L'adversaire place sur cette grille un certain nombre de bateaux qui sont caractérisés par leur longueur :

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

Il a le droit de les placer verticalement ou horizontalement. Le positionnement des bateaux reste secret. L'objectif du joueur est de couler tous les bateaux de l'adversaire en un nombre minimum de coup. A chaque tour de jeu, il choisit une case où il tire : si il n'y a aucun bateau sur la case, la réponse est *vide* ; si la case est occupée par une partie d'un bateau, la réponse est *touché* ; si toutes les cases d'un bateau ont été touchées, alors la réponse est *coulé* et les cases correspondantes sont révélées. Lorsque tous les bateaux ont été coulés, le jeu s'arrête et le score du joueur est le nombre de coups qui ont été joués. Plus le score est petit, plus le joueur est performant.

1 Modélisation et fonctions simples

Dans toute la suite, les signatures et la définition des fonctions ne sont données qu'à titre indicatif. Vous pouvez adapter comme vous le souhaitez l'énoncé afin de construire un code plus propre ou plus générique. Vous pouvez en particulier utiliser une modélisation objet. La qualité de votre code sera prise en compte dans la notation.

Nous modéliserons la grille par une matrice d'entiers de taille 10 par 10. Chaque bateau sera codé par un identifiant entier : par exemple, 1 pour le porte-avions, 2 pour le croiseur, 3 pour le contre-torpilleurs, 4 pour le sous-marin, 5 pour le torpilleur. Ainsi, une grille de jeu est constituée de cases à 0 (c'est-à-dire vides) et de cases qui chacune contiennent un entier correspondant au bateau qui l'occupe. Il est conseillé d'utiliser dans la suite le module `numpy` qui permet de faire des opérations matricielles.

Implémentez :

- une fonction `peut_placer(grille, bateau, position, direction)` qui rend vrai s'il est possible de placer le bateau sur la grille (c'est-à-dire que toutes les cases que doit occuper le bateau sont libres) à la position et dans la direction donnée (vous pourrez coder la position comme un couple d'entiers et la direction par un entier, avec par exemple 1 pour horizontale et 2 pour verticale) ;
- une fonction `place(grille, bateau, position, direction)` qui, si l'opération est possible, rend la grille modifiée où le bateau a été placé comme indiqué ;
- une fonction `place_alea(grille, bateau)` qui place aléatoirement le bateau dans la grille en tirant uniformément une position et une direction aléatoires jusqu'à ce que le positionnement choisi soit admissible et effectué ;

- une fonction `affiche(grille)` qui affiche la grille de jeu (vous pourrez utiliser par exemple `imshow` du module `matplotlib.pyplot`) ;
- une fonction `eq(grilleA, grilleB)` qui permet de tester l'égalité entre deux grilles ;
- une fonction `genere_grille()` qui rend une grille comprenant l'ensemble des bateaux du jeu disposés de manière aléatoire.

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.

1. Donnez une borne supérieure simple (à calculer à la main) du nombre de configurations possibles pour la liste complète de bateaux sur une grille de taille 10.
2. Ecrivez d'abord une fonction qui permet de calculer le nombre de façons de placer un bateau donné sur une grille vide. Comparez sa sortie au résultat théorique.
3. Implémentez une fonction qui permet de calculer le nombre de façon de placer une liste de bateaux sur une grille vide. Appliquer cette fonction sur des listes de un, deux et trois bateaux. Est-il possible de calculer de cette manière le nombre de grilles pour la liste complète de bateau ?
4. En considérant toutes les grilles équiprobables, quel est le lien entre le nombre de grilles et la probabilité de tirer une grille donnée ? Donnez 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.
5. Trouvez un algorithme qui permet d'approximer le nombre total de grilles pour une liste de bateaux. Comparez ses résultats avec le dénombrement des questions précédentes. Est-ce une bonne manière de procéder pour la liste complète de bateaux ?
6. (Bonus) Proposer des solutions pour approximer plus justement le nombre de configurations.

3 Modélisation probabiliste du jeu

Pour les questions suivantes, il est conseillé d'utiliser une classe `Bataille` qui contient une grille initialement choisie de manière aléatoire et des méthodes telles que `joue(self, position)`, `victoire(self)` et `reset(self)`, ainsi que des classes `Joueur` qui implémentent les différentes stratégies.

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 chaque coup est tiré aléatoirement ?

Programmez 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 joués.

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

Version heuristique

La version aléatoire n'exploite pas les coups victorieux précédents, car elle continue à explorer aléatoirement tout l'échiquier alors que les coups victorieux apportent de l'information sur l'emplacement des cases restantes des bateaux touchés. Proposez une version heuristique composée de deux comportements : un comportement aléatoire tant que rien n'est touché et un comportement qui va explorer les cases connexes lorsqu'un coup touche un bateau. Comparez les résultats et tracez le graphique de la distribution de la variable aléatoire du nombre de coups joués pendant la partie.

Version probabiliste simplifiée

La version précédente ne tient pas compte du type de bateaux restants ni de l'admissibilité de leur positionnement : or, certaines positions dans la grille ne pourront correspondre à des positions valides de bateaux en fonction de la localisation de la case touchée (par exemple, à coté des bords de la grille ou si un autre bateau a déjà été touché dans la région connexe). Chaque coup nous renseigne sur une position impossible, ou possible, d'un bateau. Pourquoi est-il hors de question de calculer à chaque tour la nouvelle distribution jointe sur tous les bateaux ?

On peut cependant faire une simplification et considérer chaque bateau de manière indépendante. À chaque tour, pour chaque bateau restant, on calcule la probabilité pour chaque case de contenir ce bateau sans tenir compte de la position des autres bateaux. Pour cela, en examinant toutes les positions possibles du bateau sur la grille, on obtient pour chaque case le nombre de fois où le bateau apparaît potentiellement. On dérive ainsi la probabilité jointe de la présence d'un bateau sur une case (en considérant que la position des bateaux est indépendante). Donnez une petite formalisation de ce texte. Pourquoi l'hypothèse d'indépendance est-elle fausse ? Proposez une implémentation intelligente de cette stratégie et comparez vos résultats.

Version Monte-Carlo (bonus, difficile)

Le principal problème de la version précédente est l'hypothèse d'indépendance des bateaux. Pour la prendre en compte, plutôt que de tenter de dénombrer les configurations admissibles, nous allons utiliser un algorithme de Monte-Carlo pour estimer la probabilité d'un bateau sur une case : il s'agit de tirer aléatoirement et uniformément des grilles correspondant aux contraintes connues (les positions déjà touchées, les bateaux déjà coulés) et d'en moyenner les résultats. Plus le nombre d'échantillons augmentent, plus les valeurs estimées sont fiables. Le plus difficile est de créer une grille complète qui prend en compte les contraintes. Une possibilité est de le faire par récurrence :

- soit la liste des bateaux non encore coulés ;
- on choisit aléatoirement un bateau dans cette liste ;
- on considère la liste de tous les placements admissibles de ce bateau (s'il y a des cases touchées non occupées, le bateau doit couvrir au moins une de ces cases) :
  - ▶ on choisit aléatoirement une de ces positions ;
  - ▶ on enlève le bateau de la liste des bateaux et on appelle par récurrence la même procédure sur la nouvelle liste et la nouvelle grille ;

  - ▶ s'il n'y a plus de bateaux dans la liste, on vérifie que toutes les contraintes sont satisfaites (toutes les cases touchées sont occupées par un bateau) : si c'est le cas, on renvoie cette grille, sinon on passe dans la boucle appellante à la prochaine position admissible ;
- on moyenne les résultats de N grilles tirées aléatoirement.

Donnez une implémentation et comparez vos résultats.

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

En mai 1968, le sous-marin USS Scorpion de la marine américaine s'est perdu en mer avant d'atteindre sa base navale de Norfolk en Virginie. Afin d'essayer de localiser le sous-marin, une approche bayésienne a été employée. [1] Depuis, cette technique est très utilisée pour la recherche d'objets perdus, comme par exemple les boîtes noires d'avions. La principale difficulté dans ce type de recherche est que l'on ne dispose pas d'un senseur exact : à chaque fois qu'une région de l'espace est sondée pour savoir si l'objet recherché s'y trouve, soit l'objet ne s'y trouve pas et le senseur ne détecte rien, soit l'objet s'y trouve et le senseur détecte l'objet avec une probabilité $p_s < 1$ (et donc ne le détecte pas avec une probabilité $1 - p_s$).

La modélisation de ce problème est la suivante :

- l'espace de recherche est divisé en un quadrillage de N cellules, chacune numérotée par un entier entre 1 et N ;
- chaque case i a une probabilité *a priori* π_i de contenir l'objet recherché, correspondant à un avis d'expert qui, pour chaque région de l'espace, indique les chances que l'objet s'y trouve ; [2]
- on note $Y_i \in \{0, 1\}$ la variable aléatoire qui vaut 1 pour la case où le sous-marin se trouve et 0 partout ailleurs ;
- on notera par la variable aléatoire Z_i le résultat d'une recherche en case i, valant 1 en cas de détection détection et 0 sinon ;
- on suppose que les réponses aux sondages des cases sont indépendantes et identiquement distribuées.

Réfléchissez aux questions suivantes :

1. Quelle est la loi de Y_i ? Celle de $Z_i|Y_i$?
2. Supposons que $Y_k = 1$ mais que la détection n'a pas fonctionné en case k, c'est-à-dire $Z_k = 0$. Comment exprimer la probabilité de cet événement à l'aide de Y_k et Z_k ?
3. Développez l'expression précédente afin de l'exprimer en fonction de π_i et p_s. Comment mettre à jour π_k ?
4. On considère toujours que l'on a sondé la case k et qu'il n'y a pas eu de détection. Intuitivement, cela devrait augmenter la probabilité des autres cases. De la même manière que précédemment, proposez une mise à jour de π_i pour $i \neq k$.

Proposez un algorithme pour rechercher l'objet perdu. Implémentez-le et testez-le sur différentes grilles. Vous pouvez étudier plusieurs distributions π de la probabilité *a priori* de la localisation de l'objet sur la grille, par exemple en privilégiant les bords ou le centre.

1. Voir l'article de H.R. Richardson et L.D. Stone de 1971, Operations analysis during the underwater search for Scorpion, *Naval Research Logistic Quaterly*, vol. 18(2) pp.141-157.

2. Par exemple, dans le cas du sous-marin, il est très peu probable qu'il ait coulé près de la terre ferme ou dans une région peu profonde. Sans aucune connaissance, cette probabilité est uniforme.