Projet 1 : Bataille navale

Résumé

Ce petit projet à comme objectif d'étudier le jeu de la bataille navale d'un point de vue probabiliste.

Il est à faire en binôme et à déposer sur le moodle du cours (rapport et code).

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 bien sûr prise en compte dans la notation.

Nous modéliserons la grille par une matrice d'entier 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 des cases qui contiendront un entier correspondant au bateau qui l'occupent. Il est conseillé d'utiliser dans la suite le module `numpy` qui permet de faire des opérations matricielles.

Implémenter :

- une fonction `peut_placer(grille,bateau,position,direction)` qui rend vrai s'il est possible de placer le bateau sur la grille (i.e. toutes les cases que doit occuper le bateau sont libres) à la position et dans la direction donnée (vous pouvez coder la position comme un couple d'entier et la direction par 2 entiers, par exemple 1 pour horizontal, 2 pour vertical).
- une fonction `place(grille,bateau,position,direction)` qui rend la grille modifiée (s'il est possible de placer le bateau).
- une fonction `place_alea(grille, bateau)` qui place aléatoirement le bateau dans la grille : la fonction tire uniformément une position et une direction aléatoires et tente

de placer le bateau ; s'il n'est pas possible de placer le bateau, un nouveau tirage est effectué et ce jusqu'à ce que le positionnement soit admissible.
- une fonction `affiche(grille)` qui affiche la grille de jeu (utiliser `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 avec les bateaux 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 avoir une idée de la combinatoire du jeu.

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).
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.
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 ?
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.
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 ?
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 initiée aléatoirement et des méthodes telles que `joue(self,position)`, `victoire(self)`, `reset(self)`, ... et 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 on tire aléatoirement chaque coup ?
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.
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 calculer. 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, elle continue à explorer aléatoirement tout l'échiquier. Proposer une version heuristique (à base de règles) composée de deux comportements : une comportement aléatoire tant que rien n'est touchée, un comportement qui va explorer les cases connexes lorsque un coup touche un bateau. Comparer les résultats et tracer le graphique de la distribution de la variable aléatoire.

Version probabiliste simplifiée
La version précédente ne prend pas en compte le type de bateaux restant ni si le positionnement du bateau est admissible : or, certaines positions ne seront pas possibles 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). Il est bien sûr hors de questions de calculer à chaque tour la nouvelle distribution jointe sur tous les bateaux en entier (pourquoi ?).
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, pour chaque case on obtient 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). Donner une petite formalisation de ce texte. Pourquoi l'hypothèse d'indépendance est fausse ? Proposer une implémentation intelligente de cette stratégie. Comparer 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 manière de faire est 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 des 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.

Donner une implémentation et comparer 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 (boîtes noires d'avions par exemple). 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 divisée en un quadrillage de N cellules, chacune numérotée par un entier entre 1 et N (comme pour la bataille navale) ;
- chaque case a une probabilité π_i *a priori* de contenir l'objet recherché : il s'agit d'un avis d'expert qui pour chaque région de l'espace indique les chances que l'objet s'y trouve. Par exemple dans le cas du sous-marin, il ést très peu probable qu'il est coulé près de la terre ferme ou dans une région peu profonde. Sans aucune connaissance, cette probabilité est uniforme ;
- 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 (1 si détection, 0 sinon) ;
- on suppose que les réponses aux sondages des cases sont i.i.d.

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, $Z_k = 0$. Comment traduire cette probabilité à l'aide de Y_k et Z_k ?
3. Développer 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 même manière que précédemment, proposer une mise-à-jour de π_i pour $i \neq k$.

Proposer un algorithme pour rechercher l'objet perdu. Implémenter le et tester 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.