Bataille navale

Étude statistique du jeu Bataille Navale

Boudrouss Réda n°28712638

Zhenyao Lin n°28708274

Sorbonne Université
France
20 octobre 2022

Contents

Introduction

L'objectif de ce projet et de faire une étude statistique du jeu "bataille navale". Le jeu consiste en une grille de 10 x 10 cases sur laquelle sont placés 5 bateaux de taille respective 5, 4, 3, 3 et 2. Pour chaque coup, le jouer doit tirer sur une case révélant son état : Raté, touché, ou coulé.

Quelle est donc la meilleur statégie que vas nous permettre de gagner avec un minimum de nombre de coup ?

Description du code

Le code de notre proje s'organise comme suit :

```
Bataille/
├── rapport/
├── src/
│   ├── data/
│   ├── Game/
│   │   ├── Bateau.py
│   │   ├── Engine.py
│   │   └── EngineStats.py
│   ├── Player/
│   │   ├── AbstractPlayer.py
│   │   ├── HeuristicPlayer.py
│   │   ├── HumanPlayer.py
│   │   ├── MCPlayer.py
│   │   ├── ProbPlayer.py
│   │   └── RandomPlayer.py
│   └── utils/
│       └── constants.py
└── main.py
```

Le dossier `rapport/` est le dossier qui contient tout les fichiers qui ont été nécessaires à la création du présent rapport.

Le dossier `src/` contient l'essentiel du code et de no différentes simulation.

Le fichier `main.py` est le fichier qui permet d'executer le code. Pour lancer le programe il faut `cd src/` et ensuite `python main.py`.

Le dossier `data/` contient toutes sortes d'informations sur les différences de données telque les résultats des joueurs, des images ou des données nécessaire à la bonne excécution du programme.

Le dossier `utils/` contient toutes sorte de fonctions qui ont été utiles au projet. Mais il contient aussi et surtout le fichier `constants.py` qui possède toutes les configurations du jeu et le convention de programmation. Hésitez pas à changer des paramètres dessus.

Le dossier `Game` contient les codes de la partie logique du jeu. C'est un sorte de "game engine" (d'où le nom `Engine.py`). `Engine.py` pour le jeu en lui même et `Bateau.py` pour le code de l'objet "Bateau" qui nous permet de facilité et centralisé notre code pour le bateau. `EngineStats.py` a les mêmes caractéritiques que `Engine.py` mais avec quelques fonctions assez techniques en plus. C'est ici que vous trouverais quelques une des fonctions demandés dans le sujet.

Nous avons fait le choix de connaitre les bateaux en amont de leur placement, nous les avons alors indexer avec ce que nous appelons leur `type`. C'est le numéro du bateau dans une liste trié par taille. Par défaux nous avons :

Bateau	Taille	type
Porte avion	5	1
Croiseur	4	2
Contre-torpilleurs	3	3
Sous-marin	3	4
torpilleur	2	5

Combinatoire du jeu

Dans un premier temps, intéressons nous au combinatoire du jeu "Bataille navale". Est-il possible possible de déterminer le nombre de grilles possible ?

Approche naïve

Une borne supérieur naïve serait $A_{100}^{17} \approx 2 \times 10^{33}$. En effet nous avons un échiquier de taille 10 x 10 soit 100 cases et nous devons placer en tout $5+4+3+3+2=17$, En ignorant toutes les règles, le nombre maximal de plateau possible est donc toutes les manières différentes de poser les 17 cases parmis les 100 disponibles, soit donc un arrangement de 17 parmis 100.

Cependant, ce nombre ne prends pas en compte le fait que les cases d'un même bateau doivent être adjacente. Il est possible aussi de calculer une autre borne supérieur un peu plus précise on comptabilisant manuellement le nombre de façon possible de poser un bateau de taille n dans un 10x10. Voici les résultat obtenu :

Bateau	Taille	nb
Porte avion	5	120
Croiseur	4	140
Contre-torpilleurs	3	160
Sous-marin	3	160
torpilleur	2	180

Ce qui nous donne alors comme nombre de configuration maximal théorique : 7.74×10^{10}. Cette borne cependant inclue les plateaux où les bateaux se supperpose.

Vérifions ces valeurs avec notre implémentation du jeu. Les fonctions pour cela se trouvent dans `src/Game/EngineStats.py`.

La fonction `EngineStats.nb_placer(type)` fait exactement ce que nous voulons, elle parcours chaque case du plateau et vérifie avec `Engine.peut_placer(type)` si le bateau donnée en paramètre peut être placé à la case, si oui elle ajoute 1. Nous obtenons bien les résultats théorique avec cette fonction.

La fonction `EngineStats.nb_placerL(types)` utilise la fonction précédente pour calculer les plateaux possibles avec la méthode utilisé précédemment (donc elle inclue les plateaux où les bateaux se superpose). Et on retrouve avec le résultat théorique obtenue avant. La fonction est assez simple, vous faisons juste la mutuplication de chacun des résultats.

```
def nb_placerL(types: list[int])->int:
    nb = 1
    for type in types:
        nb *= EngineStats.nb_placer(type)
    return nb
```

Mais n'est-il pas possible d'obtenir une approximation encore plus précise du nombre exacte de plateau possible ? Essayons la méthode brute.

Brute force

La fonction naïve et brute `EngineStats.nb_placerL_brute(types)` peut en théorie nous donner le nombre de plateaux exacte possible. Elle procèdes ainsi :

- Tout les bateaux sont posés à la première position possible en faisant bien attention de pas les superposer.
- Une fois tout les bateaux posés, elle incrémente le compteur de position possible et place le dernier bateau à toutes ses positions en incrémentant à chaque fois.
- Une fois que le dernier bateau a écouté toutes ses positions, on place l'avant dernier bateau à sa prochaine position viable.
- Et ainsi de suite jusqu'à ce que tout les bateaux aient écoulé toutes leur positions possible.

Voici les valeurs qu'on a pu obtenir :

Bateaux	`nb_placerL_brute()`
1	120
1, 2	14400
1, 2, 3	1850736

Cette fonction marche très bien pour des petites listes de bateaux ou pour des petites grilles, mais pour notre cas le temps d'excecution est énorme, même en optimisant avec les symètries cela prendrais toujours trop de temps, notre fonction se rapproche d'une complexité $O((3n)^m)$ avec n la taille de la dimension du plateau (supposé carré) et m le nombre de bateau.

Python n'est pas le langage pour ce genre de calcul intensif, on sait que notre résultat doit être aux alentours de 10^{10}. Le bout de code simplite suivant prend déjà trop longtemps à s'excécuter :

```
i = 0
while i < 1e10:
    i+=1
```

N'y a-t-il pas une autre méthode qui nécessite pas de faire tant de calcul ?

Approche Aléatoire

Étudions dans un premier temps le le lien entre le nombre de grille et la probabilité d'en tirer une aléatoirement.

La probabilité uniforme sur un univers fini Ω est définie par la fonction de masse :

$$p(\omega) = \frac{1}{card(\Omega)} = \frac{1}{g}$$

Avec ω l'événement élémentaire "tirer une grille donnée", Ω toutes les grilles possibles et g le nimbre de grilles.

Bien évidement on suppose ici que toutes les grilles sont équiprobable.

Nous pouvons donc explorer la formule suivante pour déduire d'une approximation du nombre totale de grille :

$$g = \frac{1}{p(\omega)}$$