Rapport du Projet 1 : Bataille Navale

ZHOU Runlin 28717281

MA Peiran 28717249

Introduction :

*Ce projet a comme objectif d'étudier le jeu de la "Bataille Navale" d'un point de vue probabiliste. Le but de ce jeu est de couler tous les bateaux(touché tous les point de bateau) sur la grille 10*10 en un minimum de coups.*

On divise ce projet à 4 partie pour trouver des moyens d'amener les joueurs à la victoire plus rapidement

- **Partie 1:** *Initaliser un jeu (la création d'un jeu / les positions des bateaux)*
- **Partie 2:**
- **Partie 3:** *Exploration du nombre d'étapes requises pour quatre stratégies différentes*
- **Partie 4:**

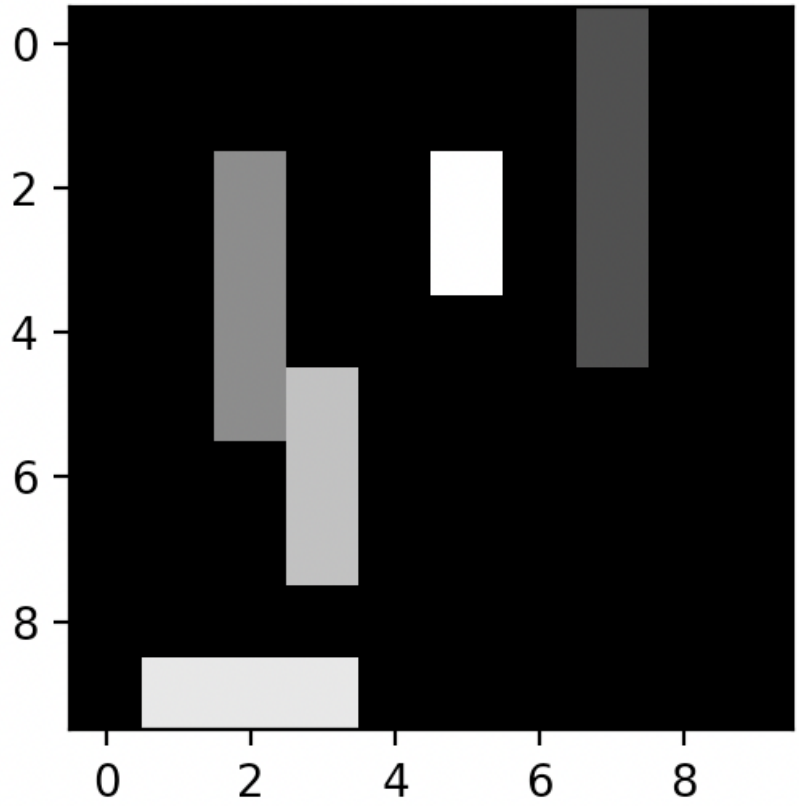
Présentation du répertoire :

Partie 1:

Dans cette section, nous utiliserons souvent les variables suivantes

- **grille:** *Zone de jeu, tous les navires doivent se trouver dans cette zone. On l'initialise comme une matrice 10*10 vide.*
- **bateau:** *Il existe cinq types au total. Chaque bateau sera codé par un identifiant entier: 1 pour le porte-avions, 2 pour le croiseur, 3 pour le contre-torpilleurs, 4 pour le sous-marin, 5 pour le torpilleur. Et chaque type de bateau a une taille correspondant.*
- **position:** *La tête d'un certain bateau. C'est une coordonnée constituée de deux entiers.*
- **direction:** *Un entier qui représente la direction d'un bateau(1 pour horizontale et 2 pour verticale)*

Exemple d'un test de programme:



Les nuances de couleur indiquent le type de navire, le noir pour les ensembles vides.

Partie 2:

Partie 3:

Dans cette section, nous commençons par créer une classe ***"Bataille",*** avec trois fonctions :

- *joue(self, position)* : tenter de toucher un bateau
- *victoire(self)* : vérifier si tous les point de chaque bateau sont touchés
- *reset(self)* : Redémarrer un jeu

Nous construisons deux matrices vide ***grille*** et ***record.***

- ***Grille*** stocke tous les information de bateau
- ***Record*** enregistre les coordonnées de tous les hits, le point touché deviendra changer à 1.

Nous comptons donc simplement le nombre de points qui passent à 1 dans ***victoire(self)***, et si nous atteignons 17, alors nous gagnons la bataille.

Nous commençons à présenter successivement les algorithmes et l'analyse des différentes stratégies:

Version aléatoire:

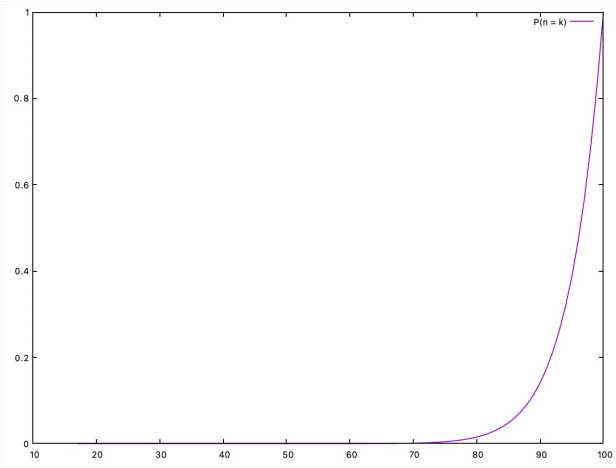
Cette stratégie consiste à frapper les coordonnées au hasard (sans les répéter).

Supposons que n actions permettent de gagner la bataille, alors cela revient à prendre n cases sur 100 et que les coordonnées des positions des bateaux sont dans la liste de n cases, car $17 \leq n \leq 100$.

On peut dériver la probabilité que le jeu peut fini dans ***n*** actions:

$$P(n) = \frac{C_{n-17}^{(100-17)}}{C_{100}^{n}}$$

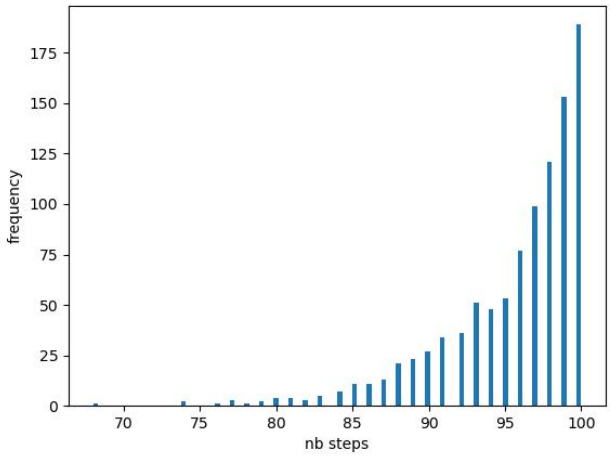
A l'aide d'un ordinateur, on peut réussir à déduire la valeur de la probabilité d'occurrence de l'événement correspondant à différentes valeurs de n. Et on peut obtenir le graph suivant:



donc l'espérance de nombre d'action est :

$E(n) = \sum n * P(n) = 100.0000$

Enfin, nous avons répété l'expérience 1000 fois et compté la fréquence des n actions gagnées, tout en traçant un histogramme de fréquence.



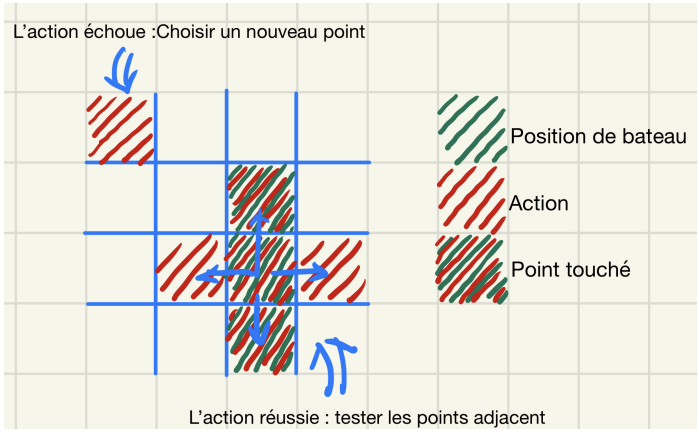
Nous avons comparé les résultats graphiques avec les prévisions que nous avions calculées et la réponse était conforme aux prévisions.

Version heuristique:

Dans le cadre de cette stratégie, nous sélectionnons toujours les points cibles de manière aléatoire comme précédemment, mais nous les divisons en deux cas différents：

- Si cette opération échoue, les nouvelles coordonnées sont à nouveau choisies au hasard.
- S'il réussit, les 4 points autour du point d'impact sont sélectionnés à tour de rôle et la direction du navire est jugée. Après cela, nous continuerons dans la direction du navire jusqu'à ce qu'il soit coulé.

Dans le shéma suivant, nous pouvons comprendre la stratégie de manière plus intuitive



En utilisant la même méthode, nous avons tracé l'histogramme de fréquence pour cette stratégie, et nous pouvons directement comparer que **Version heuristique** a moins d'actions.

Version probabiliste simplifiée:

Version Monte-Carlo: