3i005 – Projet Bataille Navale

Bastien COUPARD & Ahmed MELLITI

I) Modélisation

Nous avons implémenté nos bateaux en tant que tableaux d'une taille spécifique et dont les valeurs correspondent à un identifiant unique ce qui nous permet de les distinguer facilement sur la grille et les placer plus facilement. (Ex : Porte-avion se définit comme un tableau de 5 cases de 1) La grille est une matrice 10 par 10.

Peut_placer :

Nous avons implémenté une fonction peut_placer prenant en argument une grille de bateaux de taille 10x10, un bateau dont on va vérifier la possibilité de placement sur cette grille, la position à laquelle on teste le placement, ainsi qu'une direction correspondant à l'orientation du bateau (horizontale et verticale).

Dans un premier temps cette fonction vérifie l'orientation du bateau avec une simple condition **if**. Ensuite, nous parcourons notre grille à l'aide d'une double boucle imbriquée jusqu'à la position passée en paramètre. On vérifie que l'ordonnée ou l'abscisse (selon l'orientation) sommée à la taille du bateau ne fasse pas un dépassement dans la grille. Si jamais les cases ciblées sont indisponibles, la fonction retourne False, True sinon.

Place :

Notre fonction Place copie la grille donnée en paramètre, vérifie qu'on peut placer le bateau grâce à la fonction précédente puis, selon la direction donnée en paramètre, remplit les cases ciblées par le premier indice du bateau car les valeurs des cases font office d'identifiant**.** La fonction retourne la nouvelle grille.

Delete :
De même, la fonction **delete** fait l'opération inverse et met des zéros à la place. Cela servira lors du dénombrement des placements possibles des bateaux.

Place_alea :

Cette fonction a pour but de placer un bateau en tirant aléatoirement ses coordonnées et sa direction. Pour ce faire nous utilisons la fonction **randint** pour tirer deux entiers entre 0 et 9 qui correspondront aux coordonnées voulues. De plus, nous faisons un tirage aléatoire **random** pour avoir un flottant entre 0 et 1. Si ce flottant est supérieur à 0,5, on effectue un placement vertical, horizontal sinon. On fait ensuite appelle à la fonction **place.**

Affiche :
Simple fonction d'affichage.

Eq :
Comparaison des deux grilles passés en paramètres, et retourne le booléen résultant de cette comparaison.

Genere_grille :

Génère une grille vide et place aléatoirement les bateaux en suivant l'ordre établi dans la fonction (porte-avion, puis croiseur, puis sous-marin, etc.)

II) Combinatoire du jeu

1) Pour la borne supérieur, nous calculons le nombre de position valides pour le premier bateau. Ensuite on ajoute les bateaux un à un et on calcule le nombre de possibilités pour chacun.

2)

1) Porte-avion : bateau de 5 cases, 100 cases disponibles

⇨ 6 possibilités par lignes x 10 lignes
⇨ 6 possibilités par colonnes x 10 colonnes
⇨ 120 placements possibles pour le premier bateau.

2) Croiseur : bateau de 4 cases, 95 cases disponibles
Si on le place sur la même ligne que Porte-avion, il reste 5 cases disponibles sur cette ligne, soit deux placements possibles

⇨ 7 possibilités par lignes x 9 lignes vides
⇨ 7 possibilités par colonnes x 9 colonnes vides
⇨ Colonnes (63 +2) + Lignes (63+2) =130 placements possibles pour le second bateau.

3) Contre-Torpilleur : bateau de 3 cases, 91 cases disponibles
Il n'y a plus de place sur la même ligne que 1) et 2), ce qui laisse 9 lignes restantes.

⇨ 8 possibilités par lignes x 9 lignes
⇨ 7 possibilités par colonnes x 9 colonnes
⇨ 8 possibilités par colonnes x 1 colonne
⇨ 63+8+8x9= 143 possibilités

4) Sous-Marin : bateau de 3 cases, 88 cases disponibles
Il n'y a plus de place sur la même ligne que 1) et 2)
7 cases disponibles sur celle de 3)

⇨ 5 possibilités sur cette ligne
⇨ 8 possibilités par ligne x 8 lignes
⇨ 7 possibilités par colonnes x 9 colonnes
⇨ 5 possibilités par colonnes x 1 colonne
⇨ 3) enlève 6 possibilités
⇨ 64+5+63+5-6= 137 possibilités

5) Torpilleur : bateau de 2 cases, 85 cases disponibles
   Il n'y a plus de place sur la même ligne que 1) et 2)
   7 cases disponibles sur celle de 3) et 4)
   8 lignes de 10 cases

⇨ 3 possibilités par lignes x 1 ligne
⇨ 9 possibilités par lignes x 8 lignes
⇨ 3 possibilités par colonnes x 1 colonnes
⇨ 8 possibilités par colonnes x 9 colonnes
⇨ 72+3+72+3= 150 possibilités

Donc sur une grille de taille 10, il y a 120 x 130 x 143 x 137 x 150 = 45 842 940 000 possibilités.

3)

Pour le placement d'un seul bateau, on lance la fonction peut_placer sur toutes les cases du tableau et on incrémente un compteur à chaque fois qu'on récupère un True en sortie de peut_placer.
Soit une ligne de taille l et un bateau de taille n.
Il y a $(l-n)+1$ possibilités de placer ce bateau sur cette ligne.
Idem pour les colonnes. Ainsi, sachant que dans notre grille, le nombre de lignes est égale au nombre de colonnes, il y a $2 * [(l-n)+1]$ possibilités pour un bateau sur l'ensemble de la grille. Donc, dans une grille 10*10, il y a : $(2 * 10) * [(l-n)+1]$ placements possibles.

4)

On a donc 45 842 940 000 grilles possibles, et on en tire une seule. On a donc : $\frac{1}{45\,842\,940\,000}$ chance de tirer une grille donnée.

On a déjà une fonction qui génère des grilles aléatoirement. On va donc récupérer une grille en paramètre de notre fonction, et incrémenter un compteur dans une boucle tant que notre grille générée n'est pas équivalente à la grille passée en paramètre.

5)

Pour cet algorithme, nous avons décidé de prendre en paramètre une grille ainsi qu'une liste de bateaux. Pour des raisons d'ergonomie, nous avons opté pour la méthode récursive. Nous copions la grille donnée en paramètre et testons toutes les positions pour le placement du premier bateau de la liste, à chaque position valide. Nous avons copié la grille avec le premier bateau placé dessus. Nous incrémentons un compteur de façon récursive en utilisant comme paramètre la nouvelle grille avec le premier bateau de la liste placé, ainsi que la liste de bateau en enlevant le bateau à l'indice 0 dans notre liste de bateaux. Ceci étant fait, nous enlevons le bateau pour tester les positions suivantes. Ainsi, le compteur qui est retourné correspond bien au nombre de possibilités de placements pour une liste de n bateaux.

Cependant, notre algorithme n'est viable que pour une liste de 3 bateaux. Au-delà le temps d'exécution est trop important.

Pour optimiser notre algorithme, nous aurions pu améliorer la fonction peut_placer avec une seule boucle allant de 0 à 100 pour parcourir toute la grille.

III) Modélisation probabiliste du jeu

On crée bien la classe Bataille, qui contient une grille générée aléatoirement. La fonction **joue** teste si la position passée en paramètre correspond bien à une position occupée par un bateau. On renvoi True si on touche un bateau et on modifie la position sur la grille pour visualiser un bateau endommagé. Dans le cas contraire, on revoit uniquement False.
Cette classe comporte également une méthode victoire, qui parcourt la grille et vérifie si il reste une case autre que vide ou bateau endommagé. Et enfin, une fonction reset qui remplace la grille par une nouvelle générée aléatoirement.

1)

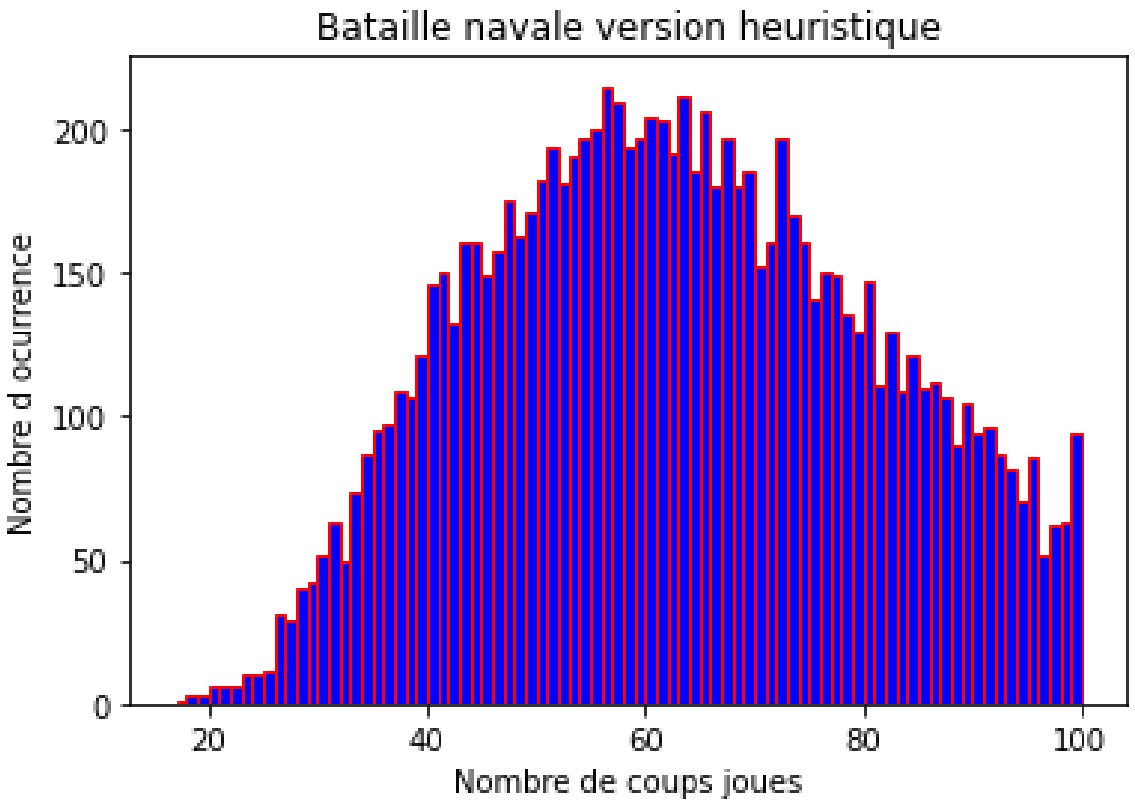
Pour la version aléatoire, nous utilisons une liste de positions que nous remplirons avec les positions tentés. Cela nous permettra de compter le nombre de tentative à la fin. On tire des coordonnées aléatoirement à l'aide de la fonction **randint**, et on utilise la fonction **joue** de la classe bataille qui aura été initialisé précédemment. Pour éviter de tirer deux fois les mêmes coordonnées, on vérifie que la position résultant du tirage aléatoire ne se trouve pas déjà dans la liste des positions déjà tentés. On retourne la taille de la liste à laquelle on retire 1 (le premier élément qui est vide est superflu).

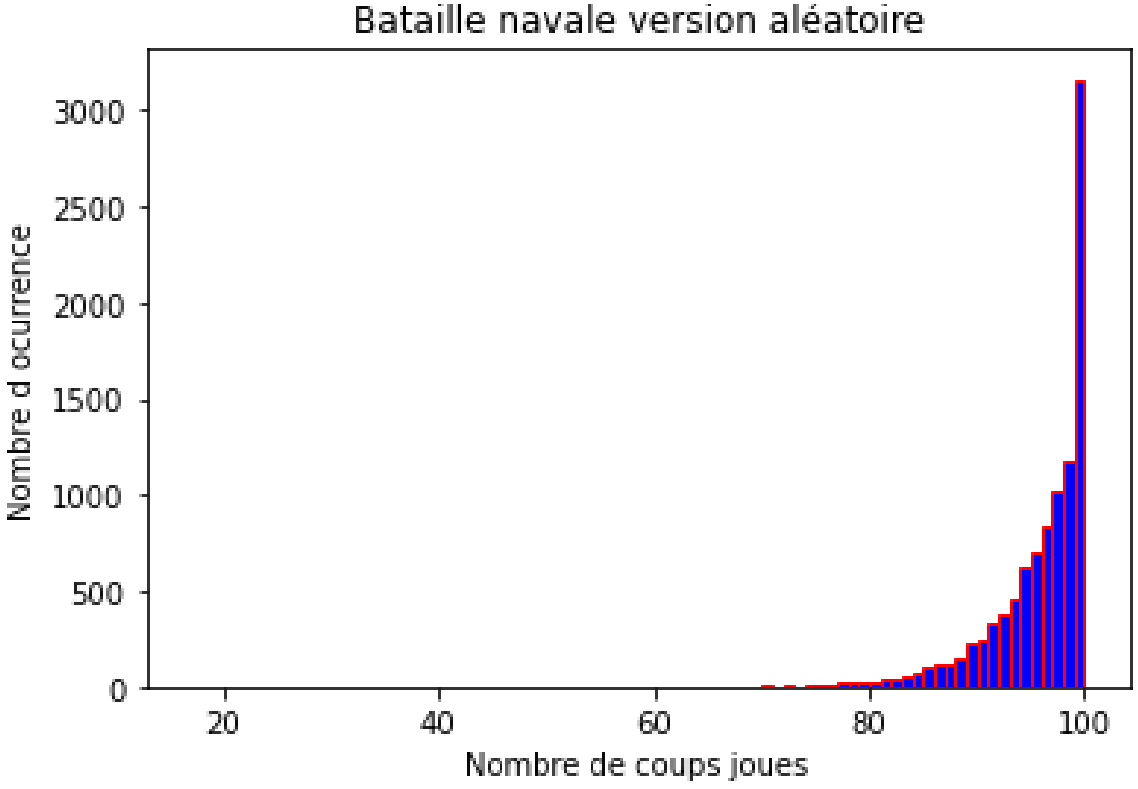
2)

Pour la version heuristique nous avons décidé de séparer le processus en 2 fonctions. Une première fonction qui tire aléatoirement des coordonnées dans une liste des coordonnées possibles construite précédemment dans une fonction annexe (**build_list**), une fois qu'un bateau est touché par cette fonction, elle va vérifier toute les cases connexes à celle-ci pour déterminer le sens (horizontal ou vertical) et l'orientation (droite ou gauche / haut ou bas). Une fois tous ces paramètres déterminées, ils sont transmis à la seconde fonction qui va simplement enchainer les tentatives jusqu'à avoir complètement détruit le bateau repéré précédemment.

Cette version, plus efficace que la précédente, nous permet de faire un grand nombre de simulation (jusqu'à 10,000 réussi, nous n'avons pas tenté au-delà pour des raisons écologiques). Grâce à cela, nous avons pu présenter nos résultats sous forme de graphique, avec en abscisses le nombre de coups joués, et en ordonnés, le nombre d'occurrences de ce nombre parmi le nombre total de tentatives.

Vous trouverez ci-dessous les graphiques résultants des 10,000 simulations effectués pour les deux versions.


Bataille navale version heuristique
Nombre d ocurrence
0
50
100
150
200
20
40
60
80
100
Nombre de coups joues


Bataille navale version aléatoire
Nombre d ocurrence
0
500
1000
1500
2000
2500
3000
20
40
60
80
100
Nombre de coups joues