La Bataille navale

Résumé :Étudier quelques propriétés de la bataille navale.

Chapitre 1 : Bataille navale simplifiée.

I.Les règles du jeu

Nous allons nous attacher à étudier le principe de la bataille navale et pour cela, il va falloir l'étudier par pallier de difficulté. Ceci implique que nous allons définir au début des règles simple puis ensuite, des règles de plus en plus complexes. Pour ce chapitre 1, on se placera dans un carré de dimension $n\times n$ où $n>2$ et où chaque ligne est composée de n carré de dimension 1, on nommera ces carrés les cases. Un bateau est un ensemble de 3 cases soit verticales, soit horizontales.

Remarque : Cette analyse de la bataille navale m'est venu en faisant le sujet des olympiades de l'académie de Nantes (2020).

II.Jeu optimal

II.0.Définition

Un jeu optimal est l'ensemble C de cases minimales où l'on doit apposer une croix pour que l'on soit sûr de toucher au moins une fois un bateau. Si l'on se trouve sur un carré de dimension $n\times n$, on note ce jeu optimal: $J(n)=card(C)$.

II.1.Les Multiples de 3.

X		
	X	
		X

Voici un carré de dimension 3×3 . On a posé les croix de telle façon que l'on ne puisse poser aucun bateau sur le carré sans qu'une de ces « parties » soient en contact avec une croix.
On note donc $J(3)=3$.

Démonstration : On raisonne par l'absurde en admettant que $J(3)=2$. On appelle C_n la colonne de rang n en comptant de gauche à droite et L_n la ligne de range n en comptant de haut en bas. Si on place deux croix sur 2 colonne différentes sur C_1 et C_2 (interchangeables) alors on pourra placer un bateau sur C_3 et le raisonnement est similaire pour L_n . Donc c'est $J(3)\neq2$, on en déduit que $J(3)=3$.

On peut essayer de généraliser pour tous les multiples de 3 :

6 x 6 : $J(6)=12$

		X			X
	X			X	
X			X		
		X			X
	X			X	
X			X		

9 x 9 : $J(9)=27$

		X			X			X
	X			X			X	
X			X			X		
		X			X			X
	X			X			X	
X			X			X		
		X			X			X
	X			X			X	
X			X			X		

Propriété : $\forall p\in\mathbb{N}^*$, on a : $\frac{J(3p)}{9p^2}=\frac{1}{3}$

Démonstration : On pose $n=3p$ la dimension du carré. Pour qu'il n'y est pas de bateaux qui puissent se poser sur le carré sans être touché il faut qu'il y est p croix par lignes ce qui fait $np=3p\times p=3p^2$ pour un carré de dimension $3p$. Or

$\frac{3p^2}{n\times n}=\frac{3p^2}{3p\times 3p}=\frac{3p^2}{9p^2}=\frac{1}{3}$

Théorème : $\forall p\in\mathbb{N}^*, J(3p)=3p^2$

Démonstration : La démonstration découle de la propriété surjacente, c'est à dire : $\frac{J(3p)}{9p^2}=\frac{1}{3}\Leftrightarrow J(3p)=3p^2$.

II.2 Les multiples de 4

		X	
	X		
X			X
		X	

On note donc $J(4)=5$

X		X			X		
	X			X			X
X			X			X	
		X			X		
	X			X			X
X			X			X	
		X			X		
	X			X			X

On note donc : $J(8)=21$.

Théorème : $\forall p\in\mathbb{N}^*,\ J(4p)=\lfloor\frac{16p^2}{3}\rfloor$.

Démonstration : On pose $n=4p$ la dimension du carré. De manière similaire à la démonstration précédente, on a $\frac{4p}{3}$ croix sur une ligne donc $\frac{16p^2}{3}$ pour tout le carré. Or, ce nombre doit être un nombre entier donc : $\lfloor\frac{16p^2}{3}\rfloor$.

II.3 Généralisation de la formule

Théorème : Pour tout entiers q,p tels que $q>2$ et $p\in\mathbb{N}^*$, on a : $J(pq)=\lfloor\frac{q^2p^2}{3}\rfloor$.

Démonstration : La démonstration est similaire aux deux précédentes.

Théorème : Soit la famille $(k_1,k_2,k_3,\ldots,k_n)$ de la taille des bateaux telle que $k_1<k_2<k_3<\ldots<k_n$. Alors $J(pq)=\lfloor\frac{q^2p^2}{k_1}\rfloor$.

Démonstration : La démonstration peut se résumer à $\forall k_1>2,\ k_1<k_2<k_3<\ldots<k_n\Leftrightarrow\lfloor\frac{q^2p^2}{k_1}\rfloor>\ldots>\lfloor\frac{q^2p^2}{k_n}\rfloor$. Autrement dit, si on considère un ensemble de bateaux avec des tailles variables. Si l'on ne considère pas le jeu équitable du plus petit bateau mais plutôt celui du i-ème, on pourra alors placer des bateaux de taille $k_{i-q},\forall q\in[\![1,i-1]\!]$.

II.4 Algorithmes sur Jeu optimale

On mettra en scène des algorithmes basiques permettant l'analyse du jeu optimal. Par soucis de généralité on codera ces algorithmes en langage naturel.

Calculer un jeu optimal :

```
Lire k,q,p
Afficher partEnt((q**2 * p**(2))/k) #partEnt pour partie Entière
```

Montrer la courbe d'évolution du jeu optimale :

```
Lire k,q,d #d est la dernière valeur souhaitée
x,y = [],[]
Pour q allant de 1 à d
     x.ajouter(q) #Fonction pour l'ajout d'éléments dans la liste
     y.ajouter(partEnt((q**2*p**(2))/k))
Fin Pour
Dessiner(x,y) #Fonction qui permet de dessiner un graphe
```

Remarque : Cet algorithme permet de mettre en évidence le fait que plus le nombre p croît, plus la probabilité de tomber sur un bateau dès le premier tour est faible.

III. Étude des structures

III.1. Étude de quelques cas particuliers

n=3 :

On sait que $J(3)=3$, donc il faut qu'on place 3 croix telles qu'on ne puisse plus placer nul part un bateau de longueur 3. On pose les croix de la manière suivante :

		X
	X	
X		

Remarque : Il existe d'autre manière de placer les croix pour qu'un bateau ne puisse se poser sur le carré sans être touché.

n=4:
On sait que $J(4)=5$. Donc on a :

		X	
	X		
X			X
		X	

n=5 :
On sait que $J(5)=8$. Donc a :

		X		
	X			X
X			X	
		X		
	X			X

On remarque que l'on a ces diagonales de croix qui sont un motif récurrent dans nos jeux optimaux. On va partir alors sur ce modèle.

III.2 Conception de l'algorithme

Pour modéliser la situation, on va utiliser les listes et plus précisément les listes de listes de mêmes longueurs, ce qui revient à utiliser des matrices carrés parfaitement adaptées pour stockées nos données. Nous coderons en langage naturel.

On va noter la liste principale C, la liste des colonnes et la liste secondaire (la liste contenue dans C), la listes des Lignes notée L.

On devrait avoir quelques choses de cette forme

$C=[L_{0,}L_{1,}L_{2,}L_{3,}...,L_{n-1}]$ où n est mesure du carré.

Voici le code correspondant :

```
Lire n
C = []
Pour v allant de 0 à n :
     L = []
     Pour w allant de 0 à n :
          L.ajouter(0)
     C.ajouter(L)
     Fin Pour
Fin pour
```

Remarque : Le code suivant sera sûrement très imparfait (au niveau de l'optimisation de la mémoire). Libre à vous de l'optimiser si c'est votre souhait.

Remarque : Nous allons avoir un programme qui permet de déterminer les positions des croix appartenant au jeu optimal. Ce programme est certes très amusant mais il trouve toutes ses applications s'il l'on conçoit un programme qu'il joue à la place d'un joueur. Cependant, il faut faire preuve d'imagination et ne pas concevoir notre programme comme un simple outil permettant de parcourir des listes prédéfinies. Principalement, nous allons introduire l'aléatoire. Celui-ci va rendre invulnérable notre algorithme aux stratégies de l'adversaire. Par exemple, si notre adversaire connaît l'ordre dans lequel parcours notre algorithme les listes prédéfinies, il peut alors optimiser son temps afin de gagner des coups d'avances. L'utilisation de l'aléatoire est donc requis pour optimiser la manière dont va jouer notre algorithme.

Voici le code correspondant pour construire le plan :

```
Lire n
i = 2
j = 0
Tant que (n+1)-i > 1 alors
     k = 0
     Pour k allant de 0 à n
          Si i-k ≥ 0 et n-1 ≥ k alors
               C[i-k][k] = « X »
          Sinon
               i += 1
               Stop
          Fin Si
     Fin Pour
     i += 2
Fin Tant que

Si C[0][n-1] == « X »  alors
     j = 3
Sinon
     Si C[0][n-2] == « X » alors
          j = 2
     Sinon
          j = 1
     Fin Si
Fin Si

Tant que (n+1)-j > 1 alors
     q = 0
     Pour q allant de 0 à n
          Si n-1-q ≥ 0 et n-1 ≥ j+q alors
               C[n-1-q][j+q] = « X »
          Sinon
               j += 1
               Stop
          Fin Si
     Fin Pour
     j += 2
Fin Tant que

Pour elt dans C
     Afficher elt
Fin Pour
```

Extraire les coordonnées des points du jeu optimale

```
coord1 = []
C = [[…],[…],…,[…]] #Voici notre carré (voir algorithme
précédent)
Pour k allant de 0 à n #n est la longueur de C notée aussi len(C)
     Pour l allant de 0 à n #même remarque
          Si C[k][l] = « X » alors
               coord1.ajouter([[k+1,l+1]])#Coordonnées jouables
          Sinon
               Passer
          Fin Si
     Fin Pour
Fin Pour
```

Remarque : La notion de coordonnées jouables est assez intuitive puisqu'il s'agit des coordonnées utilitaires notées de 0 à n-1 que l'on note simplement de 1 à n afin que cela soit plus naturel pour les joueurs.

Maintenant nous devons créer un algorithme qui parcourt de manière aléatoire cette liste prédéfinie. Pour cela, il faut un algorithme qui sort un nombre aléatoire différent à chaque fois. Nous pouvons imaginer l'algorithme suivant :

```
Définir fonction ListeAleatoire(cardC)
     L = []
     Tant que longueur(L)!= cardC
          a = alea(0,cardC)
          Si a est dans L alors
               Passer
          Sinon
               L.ajouter(a)
     Fin Tant que
     Retourne L
```

Remarque : CardC est simplement l'argument de la fonction qui représente la longueur de la liste des points du jeu optimal ou autrement dit J(qp)

Avec tous les éléments mis à votre disposition, vous pouvez assez aisément construire un programme qui joue de manière optimisé au jeu de la bataille navale. Nous essayerons de le mettre en ligne sur le Git-Hub !