PROJET 1 : BATAILLE NAVALE

1 . MODÉLISATION ET FONCTIONS SIMPLES

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 pour chacun des joueurs , l'objectif du joueur est de couler tous les bateaux de l'adversaire en un nombre minimum de coups.
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 valeur de la case est 0); si la case est occupée par une partie d'un bateau, la réponse est “touché” (on met la valeur de la case à -1) ; 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.Le gagnant est donc le joueur ayant fait couler tous les bateaux en un nombre minimal de coups.

Pour se faire , nous avons créé la classe suivante:

Classe Grille : représente notre grille et contient la matrice dont chaque case contient un 3-uplet tel que : le premier élément est à 0: si case vide ,à -1 : si touché et égale à l’identifiant du bateau sinon ; le deuxième élément c’est la direction et le troisième contient l'entier i qui représente la i éme case du bateau placé dans cette case . Cette classe est définie par l’ensemble des méthodes suivantes (qui sont bien documentées dans le code source):
`peut_placer(self, bateau, position, direction) , place(self, bateau, position, direction), place_alea(self, bateau) , affiche(self),eq (grilleA, grilleB) ,`

```
genere_grille(),nb_places(self, bateaux ) ,
eq_grille_donnee_grille_genere(self) ,
genere_grille_de_bateau(bateaux) ,
approximer_nb_grille(self,bateaux)
```

Dictionnaire des bateaux : comme variable globale contenant l'identifiant de chaque bateau associé au nombre de cases qu'il occupe

---

2 . COMBINATOIRE DU JEU

Dans cette partie, nous nous intéressons au nombre de grilles possibles avec les différentes positions des 5 bateaux afin d'appréhender la combinatoire du jeu.

2.1 Borne supérieure

Afin de calculer la borne supérieure du nombre de configurations possibles pour la liste complète de bateaux sur la grille de taille 10, on suppose que les bateaux peuvent se superposer (ce qui n'est pas le cas dans le vrai jeu).Donc, pour chaque bateau on essayera de trouver le nombre de dispositions possibles que ça soit horizontal ou vertical sur une grille vide.

Pour calculer ce nombre on remarque que sur une ligne ou une colonne il y a :

$$N_{nombre\ de\ case\ dans\ une\ ligne} - (Taille(bateau) - 1)$$

possibilités pour placer un bateau, et comme il y a 10 lignes et 10 colonnes donc on multiplie ça par 2*10 ce qui donnera :

$$2 * N_{nombre\ de\ ligne}(N_{nombre\ de\ case\ dans\ une\ ligne} - (Taille(bateau) - 1))$$

possibilités pour chaque bateau.

On obtient donc à la fin la relation suivante pour calculer la borne supérieure :

$$\prod_{chaque\ bde\ bateaux} 2 * N_{nombre\ de\ ligne}(N_{nombre\ de\ case\ dans\ une\ ligne} - (Taille(b) - 1))$$

Bateau	Nombre de dispositions
Porte-avions	2*10*(10-(5-1)) = 120
Croiseur	2*10*(10-(4-1)) = 140
Contre-torpilleurs	2*10*(10-(4-1)) = 160
Sous-marin	2*10*(10-(3-1)) = 160
Torpilleur	2*10*(10-(2-1)) = 180
Borne supérieure	77 414 400 000

2.2 Nombre de façons de placer un bateau

La fonction qui permet de calculer le nombre de façons de placer un bateau donné sur une grille vide est implémentée dans la classe Grille : `denombrement_bateau(self, bateau)`.

En comparant la valeur retournée par cette méthode avec le calcul théorique fait dans la première question, on remarque que les résultats sont identiques. Donc on peut conclure que le raisonnement appliqué dans la question précédente est correct.

2.3 Nombre de façons de placer une liste de bateaux

Pour calculer le nombre de façon de placer une liste de bateaux sur une grille vide, on a implémenté dans la classe Grille la méthode `nb_places(self, bateaux.` Cette méthode prend en paramètres une liste de bateaux et calcule de manière récursive le nombre de façon de placer une liste de bateaux, en testant la possibilité de placer chaque bateau sur toutes les cases de la grille que ça soit horizontalement ou verticalement et en prenant en considération les bateaux déjà placés d'une façon à ne pas avoir des bateaux superposés.

En testant cette fonction sur des listes de 1, 2, 3 bateaux on aura les résultats suivant :

1 Porte-Avion: 120 dispositions

1 Porte-Avion + 1 Croiseur: 14400 dispositions

1 Porte-Avion + 1 Croiseur + 1 Contre-Torpilleur: 1850736 dispositions

Il n'est pas possible de calculer avec cette fonction le nombre de grilles possibles avec une liste de 5 bateaux car sa complexité est exponentielle. Donc pour 5 bateaux la fonction tournera pendant une longue durée pour calculer toutes les configurations possibles une par une.

2.4 Probabilité de tirer une grille donnée

En considérant toutes les grilles équiprobables, la probabilité pour tirer une grille sera égale à 1/N (N est le nombre total de grilles possibles)

2.5 Approximation du nombre total de grilles

Pour cette question, on a implémenté la fonction :
`approximer_nb_grille(self,bateaux)` qui prend en paramètre la liste des bateaux de la grille et génère aléatoirement une deuxième grille avec cette liste de bateaux jusqu'à l'obtention d'une grille équivalente, et tant que la grille obtenue n'est pas égale à notre grille on incrémente le compteur qui sera retourné à la fin. Cette méthode n'est pas une bonne manière de procéder pour la liste complète de bateaux car elle prend beaucoup de temps afin de s'executer.

2.6

Pour approximer plus justement le nombre de configurations, on doit prendre en compte la superposition des bateaux: il faudrait réduire le nombre de cases disponibles en fonction de la taille du bateau placé précédemment.

---

3 . MODÉLISATION PROBABILISTE DU JEU

Dans cette partie , nous allons introduire les classes suivantes :

Classe Bataille : qui est définie par une grille générée aléatoirement, un dictionnaire de bateaux coulés , ainsi que les méthodes (qui sont bien documentées dans le code source) :
`nb_case(self,bateau), joue(self, position), victoire (self),reset (self)`

Classe Joueur : qui est définie par une grille spécifique au joueur , le nombre de coups obtenu par le joueur à la fin de la bataille ,ainsi que les méthodes suivantes (qui sont bien documentées dans le code source): `reset(self), joue_aleatoire(self, bataille), joue_heuristique(self,bataille), joue_simplifie(self, bataille), proba_grille(self,bataille)`

VERSION ALÉATOIRE :

Espérance : On suppose que la probabilité de trouver une case d'un bateau est indépendante par rapport aux cases déjà trouvées, on a donc une espérance égale à $\sum Cne \diagup Crt$
où:

Cne : Nombre de Cases non explorées.
Crt : Nombre de Cases restantes à trouver.
En faisant la somme, on tombe sur une espérance de 100 coups.
L'espérance expérimentale étant de : 95 tours, on conclut que l'hypothèse de l'indépendance était fausse.

Distribution : Pour calculer la distribution, on répète la partie plusieurs fois (1000 dans ce cas , vous trouverez le code correspondant bien documenté), on obtient alors une fréquence du nombre de coups pour la partie, voici le graphique correspondant:

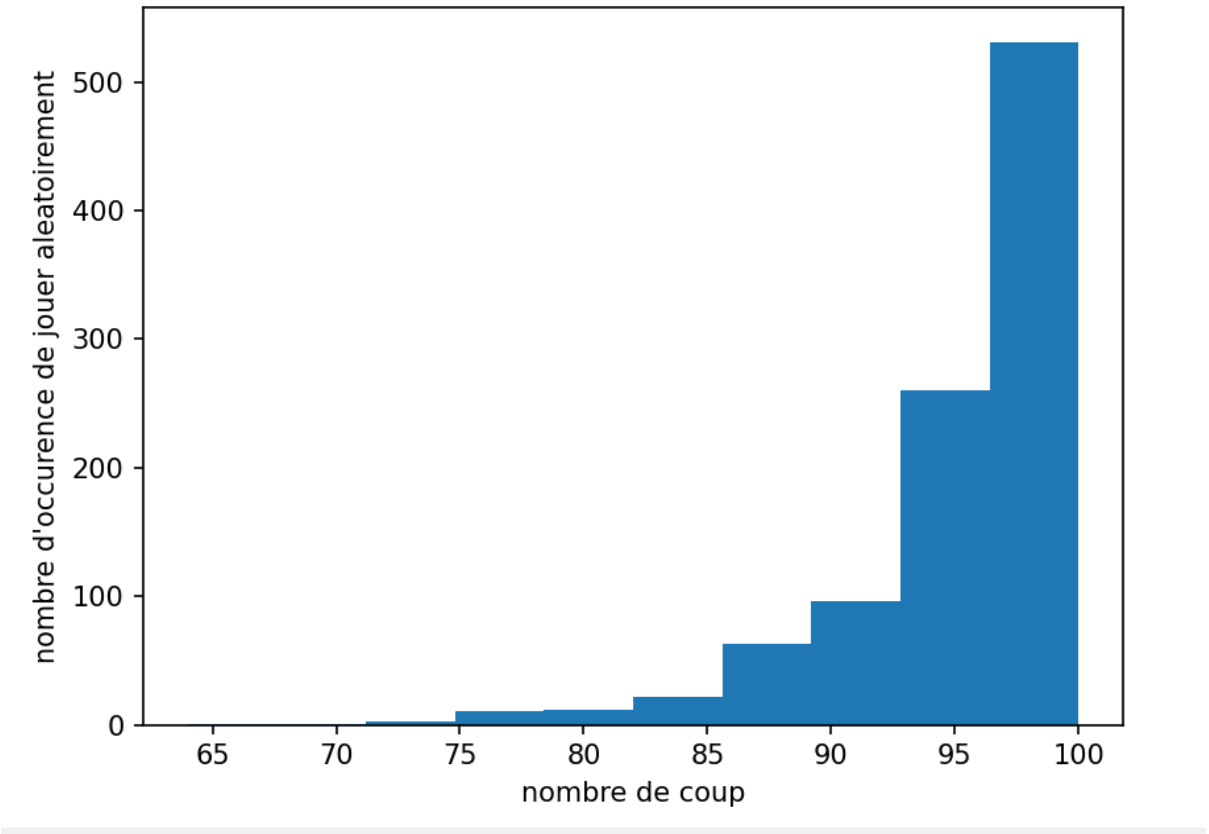


Figure 1 : Distribution de de nombre de coup selon le nombre d'occurence que le joueur a joué avec la version aléatoire

VERSION HEURISTIQUE :

Dans cette méthode, on parcourt aléatoirement les cases jusqu'à trouver une case qui contient un bateau (en éliminant à chaque fois les cases déjà utilisées), alors on explore les cases adjacentes (car il est plus probable d'en trouver dans l'une des cases adjacentes) .

Espérance :expérimentale : 81 , la probabilité de trouver un bateau dans une case n'est plus indépendante cette fois-ci .
L'amélioration est bien apparente par rapport à la version aléatoire , car ici on utilise le fait que si on trouve un bateau dans une case donnée alors il se trouve sûrement dans l'une des cases qui lui sont adjacentes

Distribution : Voici le graphique de distribution pour 1000 parties du jeu heuristique :

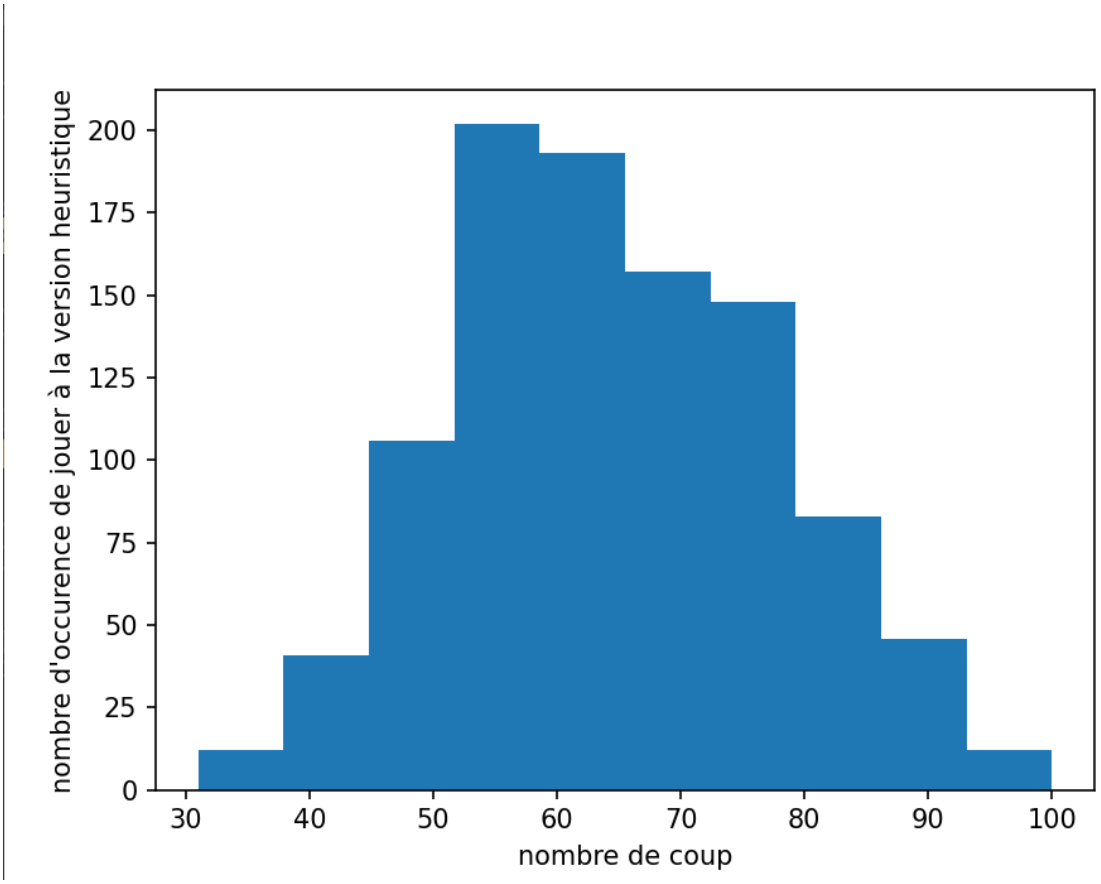


Figure 2 : Distribution de de nombre de coup selon le nombre d'occurence que le joueur a joué avec la version heuristique

VERSION PROBABILISTE SIMPLIFIÉE :

Dans cette méthode, on exploite le fait que l'on connaisse les cases déjà parcourues ,

les bateaux déjà coulés et on possède une matrice que l'on nommera : proba_grille dans notre cas ; qui contient dans chaque case le nombre de possibilités de la présence d'un des bateaux restant dans une case de la grille qui a les mêmes coordonnées , on peut donc connaître les positions qui contiennent probablement un bateau, ainsi que celles ou c'est impossible.

On procède de la manière suivante: on génère à chaque fois une position aléatoire non utilisée déjà ,on extrait alors la probabilité correspondante à cette position de la matrice proba_grille et on la compare avec tout le reste des cases de la matrice proba_grille pour trouver la position qui correspond à la plus grande probabilité de trouver le bateau dans à cette position de la matrice , on met à jour la position et on teste pour cette nouvelle position avec la méthode `joue(self, position)`

Espérance : expérimentale 67
Cette implémentation est plus rapide que l'aléatoire et l'heuristique

Distribution : Voici le graphique de distribution pour 1000 parties:

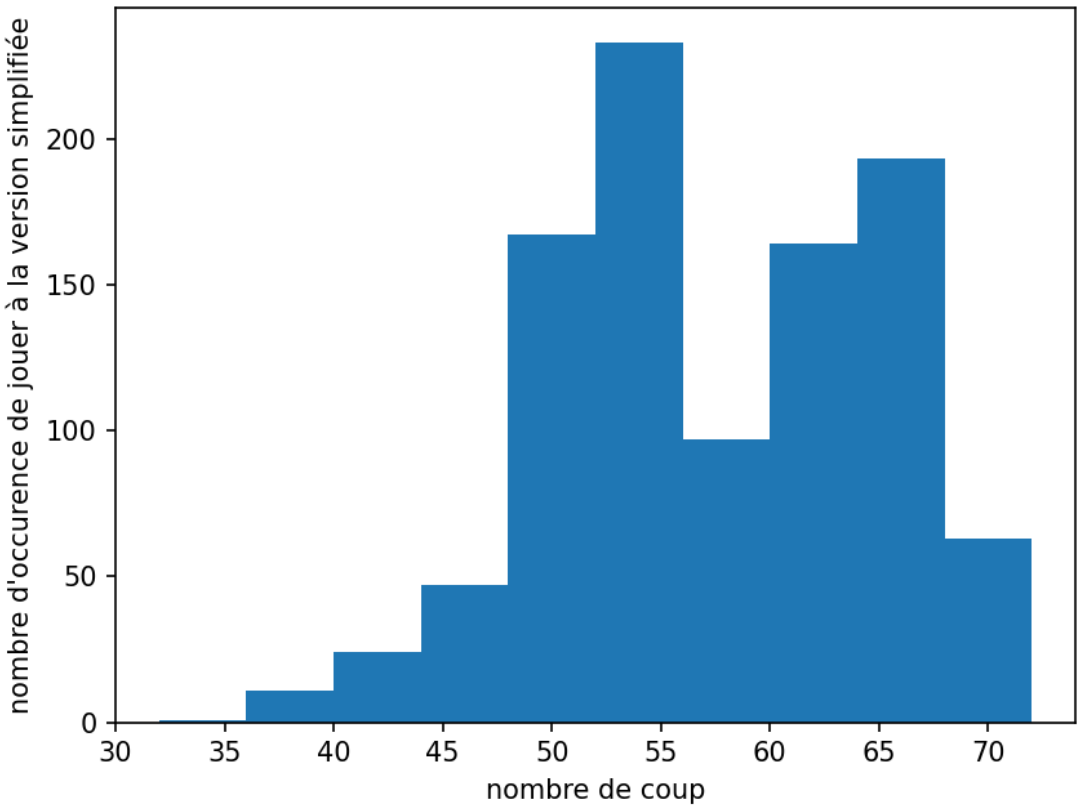


Figure 3 : Distribution de de nombre de coup selon le nombre d'occurence que le joueur a joué avec la version probabiliste simplifiée

4 . SENSEUR IMPARFAIT : A LA RECHERCHE DE L'USS SCORPION

4.1 La formulation en termes de probabilité de Ps

Ps est la probabilité de la détection de l'objet par le senseur dans une région sachant qu'il se trouve dans cette région.
Ps = P(Zi=1 | Yi=1)

4.2 La loi de Yi et Zi|Yi

-Yi suit une loi de Bernoulli de paramètre Πi

P(Yi = 1) = Πi et P(Yi = 0) = 1 - Πi

-La loi de Zi|Yi :

P(Zi=0 | Yi=0)= 1 P(Zi=0 | Yi=1)= 1-Ps
P(Zi=1 | Yi=0)= 0 P(Zi=1 | Yi=1)= Ps

Donc on peut déduire que P(Zi=y | Yi=0) tel que y ϵ {0,1} suit une loi de Bernoulli de paramètre 0 P(Zi=y | Yi=0)= 1 - y
et P(Zi=y | Yi=1) tel que y ϵ {0,1} suit une loi de Bernoulli de paramètre Ps

P(Zi=y | Yi=1)= $(1 - Ps)^{1-y} * Ps^{y}$

4.3 Probabilité d'un événement

On s'intéresse dans cette question au cas où le sous-marin se trouve en case k et un sondage est effectué à cette case mais ne détecte pas le sous-marin. Donc, la probabilité de cet évènement est :
$P(Zk = 0 \mid Yk = 1)$ = 1-Ps

4.4 Mise à jour de Πk

Si le senseur n'a rien détecté dans la case k alors on met à jour Πk .

Le nouveau Πk de cette case sera égale à P(Yk =1 | Zk =0) = $\frac{P(Yk=1 \cap Zk=0)}{P(Zk=0)}$

On a :
$P(Yk = 1 \cap Zk = 0)$ =$P(Zk = 0 \mid Yk = 1)$ * P(Yk=1) = (1-Ps)Πk
Il nous reste juste à trouver l'expression de $P(Zk = 0)$:
$P(Zk = 0)$ = P(Zk =0 | Yk =0) * P(Yk=0) + P(Zk =0 | Yk =1) * P(Yk=1)
= 1 * (1 - Πk) + (1-Ps) * Πk
= 1 - Πk +Πk - PsΠk
= 1 - PsΠk

Donc si le senseur ne détecte rien alors $\Pi knew = \frac{(1-Ps)\Pi k}{1- Ps\Pi k}$

Si le senseur n'a rien détecté dans la case k alors on met à jour Πi de la case i tel que i $\neq k$

Le nouveau Πi de cette case sera égale à P(Yi =1 | Zk =0) i $\neq k = \frac{P(Yi= 1 \cap Zk = 0)}{P(Zk = 0)}$

$= \frac{P(Zk =0 \mid Yi =1) * P(Yi=1)}{P(Zk = 0)}$

$P(Zk = 0 \mid Yi = 1)$ est la probabilité de l'évènement où le senseur ne détecte rien dans la case k sachant que le sous-marin se trouve dans la case i, cette probabilité vaut toujours 1 car si le sous-marin se trouve dans une case le senseur ne détecte toujours rien dans une autre case.

Et on sait que $P(Yi = 1) = \Pi i$ et $P(Zk = 0) =$ 1 - PsΠk

Donc P(Yi =1 | Zk =0) i $\neq k = \frac{\Pi i}{1 - Ps\Pi k}$

Donc $\Pi inew = \frac{\Pi i}{1- Ps\Pi k}$

Un algorithme pour rechercher l'objet perdu

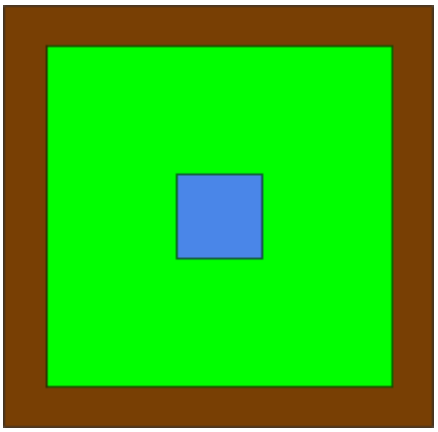
Tant que l'objet recherché n'a pas été trouver :

- On incrémente le nombre d'essais
- On récupère la case qui a la plus grande probabilité et on teste si l'objet se trouve dans cette case.
- Si on le trouve, on retourne le nombre d'essais.
- Sinon :
    - On met à jour les probabilité de toutes les cases

Pour tester cet algorithme on a créer deux grille de taille 10*10.

La grille 1 :

La première grille qu'on a créé est divisée sur 3 zones comme le montre le schéma suivant :



Avec:

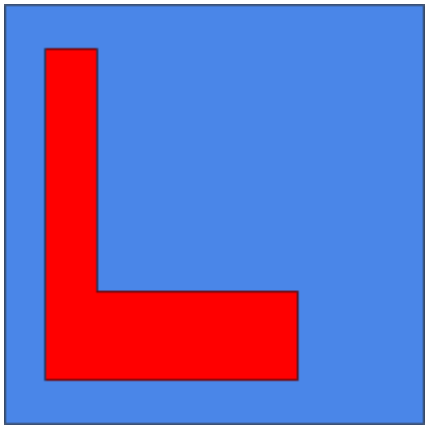
- 60% de chance que l'objet se trouve à dans la zone marron.
- 30% de chance que l'objet se trouve à dans la zone verte.
- 10% de chance que l'objet se trouve à dans la zone bleue.

A la création de la grille on initialise :

- Toutes les cases de la zone marron à 1/60.
- Toutes les cases de la zone verte à 1/200.
- Toutes les cases de la zone bleue à 1/40

La grille 2 :

La deuxième grille qu'on a créé est divisée sur 2 zones comme le montre le schéma suivant :



Avec:

- 80% de chance que l'objet se trouve à dans la zone rouge.
- 20% de chance que l'objet se trouve à dans la zone bleue.

A la création de la grille on initialise :

- Toutes les cases de la zone rouge à 2/45.
- Toutes les cases de la zone bleue à 1/410