TME2:

Nous avons calculé à la main le nombre de façon possible de disposer chacun des bateaux sur une grille de 10x10.

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}.

La fonction **naïve** suivante nous permet donc de retrouver la même valeur que nos résultat théorique.

```
def nb_placer(type:int)->int:
    j = Bataille()
    ym,xm = j.dim
    nb : int = 0
    for y in range(ym):
        for x in range(xm):
            for dir in range(2):
                if j.peut_placer((y,x), type, dir):
                    nb += 1
    return nb

nb_tot: int = 1
for i in range(1,len(LEN_B)+1):
    nb_tot *= nb_placer(i)
```

Cette valeur cepandant inclue les tableaux où les bateaux peuvent se superposer ce qui n'est pas possible dans les règles du jeu. Pour avoir une valeur plus précise, nous devons ignorer tout ces plateaux. Manuellement ça prendra trop de temps donc voici une fonction pour le faire :

```
def nb_placerL(types:list[int])->int:
    j = Bataille()
    return _nb_placerL(j, j.plateau.copy(),types)

def _nb_placerL(jeu:Bataille, plateau : np.ndarray, types:list[int])->int:
    if not types:
        return 0

    jeu.plateau = plateau
```

```
    ym,xm = jeu.dim
    nb : int = 0
    for y in range(ym):
        for x in range(xm):
            for dir in range(2):
                if jeu.peut_placer((y,x), types[0], dir):
                    plateau = jeu.plateau.copy()
                    jeu.place((y,x), types[0], dir)
                    nb += _nb_placerL(jeu, jeu.plateau, types[1:])
                    jeu.plateau = plateau
                    if len(types) == 1:
                        nb +=1
                        print(nb)
    return nb
```

Cette fonction (*naïve toujours*) 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 (ce qui diviser le nombre de calcul par 4) cela prendrais toujours trop de temps, notre fonction se rapproche d'une complexité $O(3^n)$.

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
```

Essayons une autre approche, en supposons toutes les grilles équiprobable, on a donc : $P(\text{table}_n) = \frac{1}{N}$ avec $P(\text{table}_n)$ la chance de tirer une table aléatoire et N le nombre de table au totale.

Si on prend une fonction qui prend en paramètre une grille et génère des grilles aléatoirement jusqu'à ce que la grille générée soit la même que celle passé en paramètre. Alors dans ce cas le nombre de grille généré sera une approximation de nombre totale de tableau