

# Tipe : Stratégies du jeu de plateau Risk

---

Yifan Wang

# Sommaire

Présentation du jeu

Simulation

Chaînes de Markov

Vérification et amélioration de stratégies

Annexes

## Présentation du jeu

---

# Règles du jeu

- En confrontations.
- Lancers de dés.
- Le plus grand chiffre l'emporte.

## Exemple de duel

- L'attaquant lance trois dés et obtient : 6, 4 et 1.



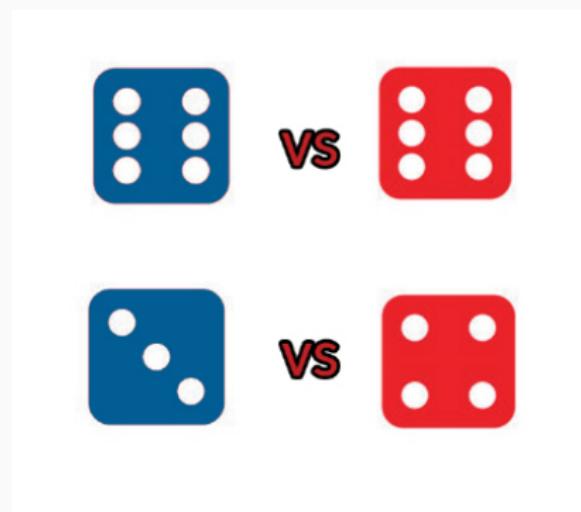
## Exemple de duel

- L'attaquant lance trois dés et obtient : 6, 4 et 1.
- Le défenseur en lance deux et obtient : 6 et 3.



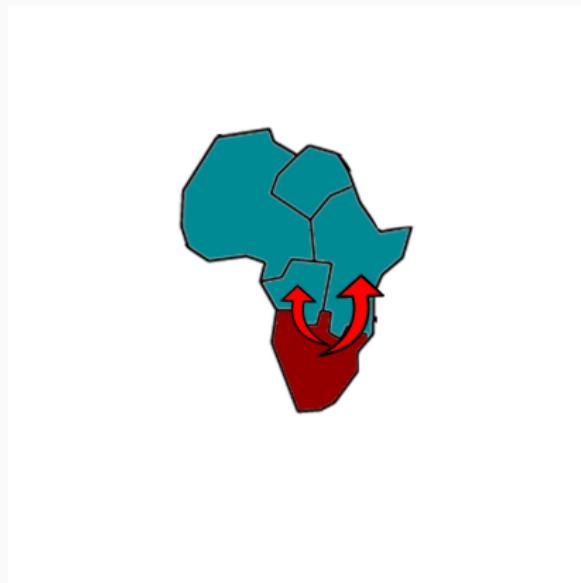
## Exemple de duel

- L'attaquant lance trois dés et obtient : 6, 4 et 1.
- Le défenseur en lance deux et obtient : 6 et 3.
- Alors le 6 du défenseur s'oppose au 6 de l'attaquant, et le 3 du défenseur au 4 de l'attaquant.
- Les deux perdent respectivement 1 armée.



## Simplification du modèle.

- La bataille se finit par l'élimination totale d'une armée.
- On étudiera une modélisation unidimensionnelle.



**Figure 1:** Exemple de progression

# Problématiques

## Problématiques:

- Comment peut-on calculer les probabilités primaires liées aux duels directs ?
- Dans le cas où une attaque se conclut forcément par l'élimination totale d'une armée, comment peut-on attaquer/défendre de manière optimale plusieurs territoires consécutifs ?

## Simulation

---

# Tableau des probabilités trouvées sur 10 000 essais pour le cas du 3vs2

---

## Situation de 3v2

---

L'attaquant gagne les deux duels	3690/10000	36.9
Le défenseur gagne les deux	2926/10000	29.26
Chacun en gagne 1	3384/10000	33.84

**Table 1:** Pour 10 000 simulations

# Chaînes de Markov

---

## Modèle de MARKOV

- Notons  $a, d$  respectivement le nombre d'attaquants et de défenseurs.
- Chaque tour d'une bataille sera représenté par un couple  $(a, d)$ .
- Une bataille se finit seulement si  $a = 0$  ou  $d = 0$ .
- Les **états absorbants**, c'est-à-dire, ceux marquant la fin d'une bataille seront de la forme  $(0, i)$   $(0, j)$   $\forall(i, j) \in \llbracket 1; a \rrbracket * \llbracket d; 0 \rrbracket$ .

Soit  $X_n = (a_n, d_n)$  l'état représentant le  $n$ -ème tour de la bataille.  
La probabilité de parvenir à cet état sachant les  $n - 1$  tours  
précédents suit la propriété de Markov

### Théorème : Propriété de Markov

$$\mathbb{P}(X_n = (a_n, d_n) | X_{n-1}, \dots, X_0) = \mathbb{P}(X_n = (a_n, d_n) | X_{n-1})$$

- Les **états de transition** sont ordonnés dans l'ordre suivant  
 $(1, 1) \dots (1, d), (2, 1) \dots (2, d) \dots \dots (a, d)$  et les absorbants  
 $(0, 1) \dots (0, d) \dots \dots (a, 0)$

$$\begin{pmatrix} P & S \\ 0 & I_{a+d} \end{pmatrix}$$

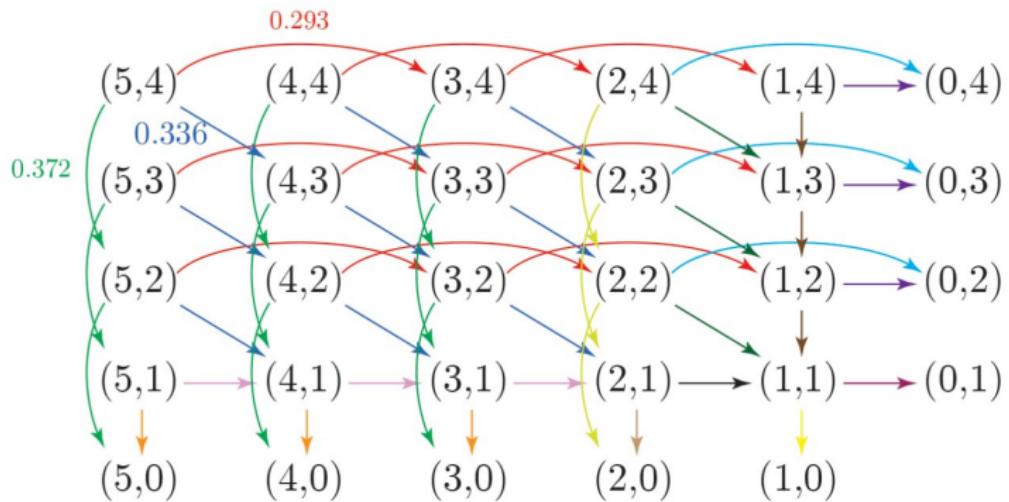
Avec  $P$  la matrice de taille  $ad * ad$  désignant la matrice pour passer d'un état de transition à un autre  $S$  la matrice de taille  $ad$ ,  $a + d$  la matrice pour aller d'un état de transition à un état d'absorption.

## Matrice où l'on commence en 2v2

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0.59 & 0 & 0.41 & 0 \\ 0.245 & 0 & 0 & 0 & 0 & 0.745 & 0 & 0 \\ 0.43 & 0 & 0 & 0 & 0 & 0 & 0 & 0.57 \\ 0.324 & 0 & 0 & 0 & 0 & 0.448 & 0 & 0.227 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

**Figure 2:** Matrice de transition pour le 2v2

# Répétition et récursivité

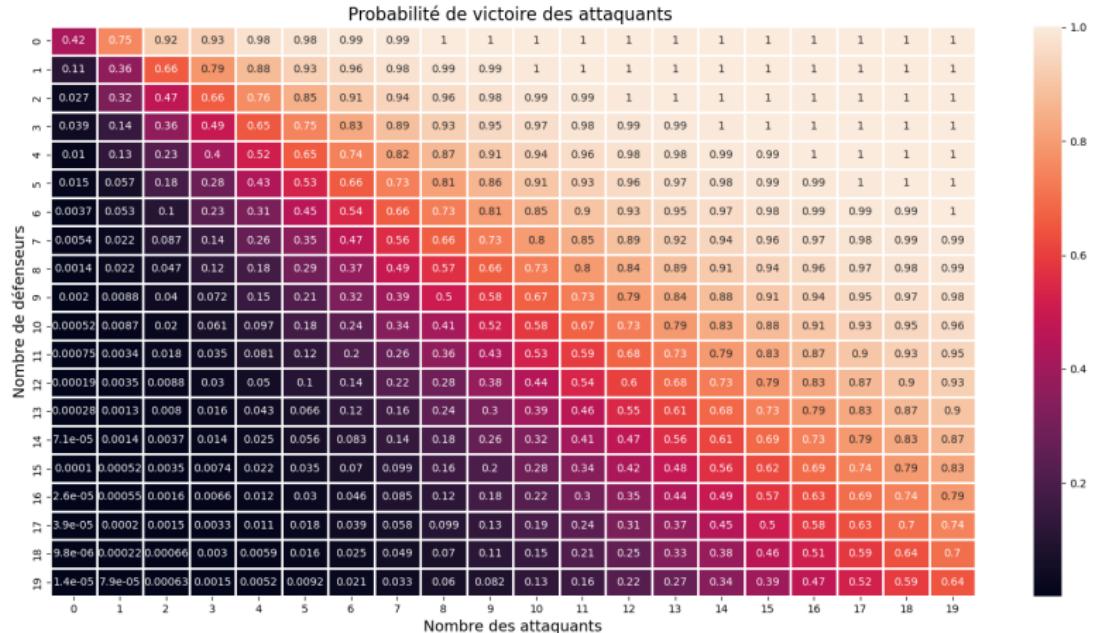


**Figure 3:** Etats de Markov lorsque l'attaquant commence avec 5 troupes et le défenseur 4

## Comparaison expérimentale et théorique

i	j	événement	$\mathbb{P}_{ijk}$	estimation	résultat réel
1	1	le défenseur perd 0	$\mathbb{P}_{110}$	0.554	$\frac{21}{36}$
1	1	perd 1	$\mathbb{P}_{111}$	0.446	$\frac{15}{36}$
1	2	perd 0	$\mathbb{P}_{120}$	0.76	$\frac{161}{216}$
1	2	perd 1	$\mathbb{P}_{121}$	0.24	$\frac{55}{216}$
2	1	perd 0	$\mathbb{P}_{210}$	0.392	$\frac{91}{216}$
2	1	perd 1	$\mathbb{P}_{210}$	0.608	$\frac{125}{216}$
2	2	perd 0	$\mathbb{P}_{220}$	0.4451	$\frac{581}{1296}$
2	2	perd 1	$\mathbb{P}_{221}$	0.3226	$\frac{420}{1296}$
2	2	perd 2	$\mathbb{P}_{222}$	0.2323	$\frac{295}{1296}$
3	1	perd 0	$\mathbb{P}_{310}$	0.6596	$\frac{441}{1296}$
3	1	perd 1	$\mathbb{P}_{311}$	0.3404	$\frac{855}{1296}$
3	2	perd 0	$\mathbb{P}_{320}$	0.2883	$\frac{2275}{7776}$
3	2	perd 1	$\mathbb{P}_{321}$	0.3331	$\frac{2611}{7776}$
3	2	perd 2	$\mathbb{P}_{322}$	0.3786	$\frac{2890}{7776}$

# Graphe de chaleur

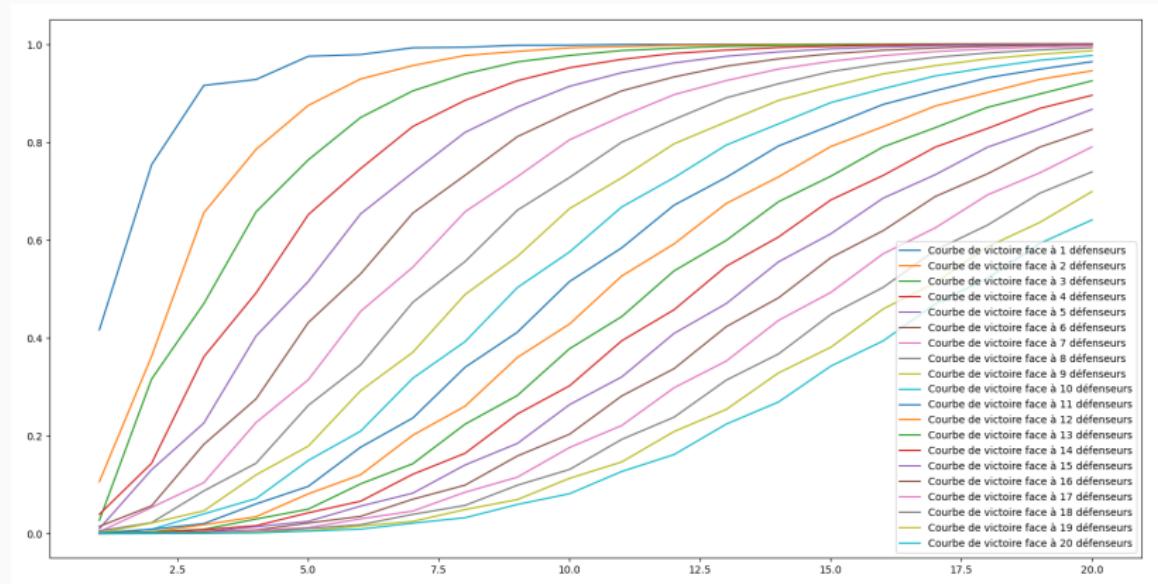


**Figure 4:** Graphe de chaleur

# Comment mieux aborder un duel

## Résultat préliminaire

Au delà de 5 troupes, l'attaquant doit favoriser l'agressivité.



**Figure 5:** Pourcentage de victoire face à un nombre donné de défenseurs

## **Vérification et amélioration de stratégies**

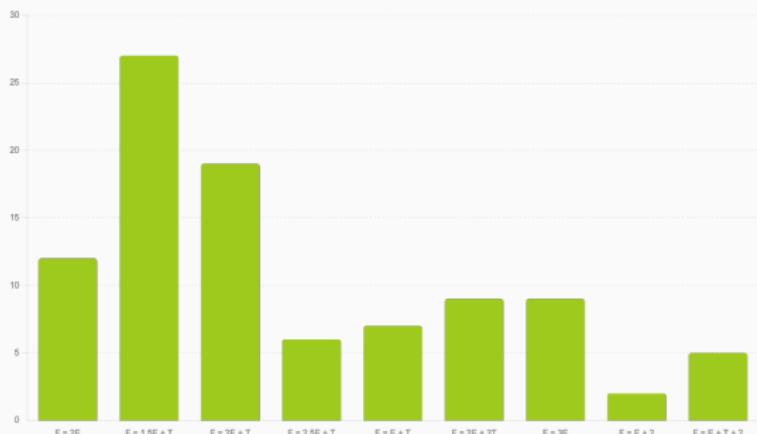
---

# Comment optimiser une offensive ?

D'après les sondages, l'attaque suivante est la plus efficace :

## Conjecture de la communauté

$A = 1.5 * D + T$  avec  $T$  le nombre de territoires,  $D$  le nombre de défenseurs



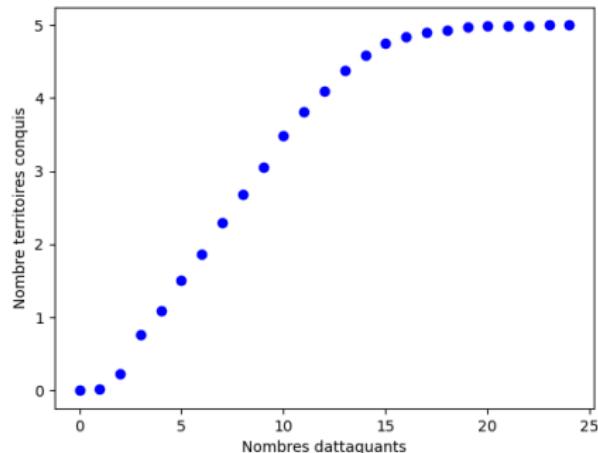
**Figure 6:** Sondage des formules offensives les plus populaires se basant sur 827 votes.

# Confrontations consécutives

```
1 def Simulationdeplateau(A,D):
2     territoiresconquis = 0
3     DefensePerdu = 0
4     for i in range(len(D)):
5         A,B = nbtroupesrestantes(A,D[i])
6         if A==0:
7             return territoiresconquis,territoiresconquis,
8             DefensePerdu
9         if B == 0 and i +1== len(D):
10            return A + len(D), len(D), DefensePerdu + D[i]
11         if B == 0 and i < len(D):
12             A = A - 1
13         # On dépose une troupe sur le territoire
14         #en plus de celles perdues car on peut plus l'utiliser
15         territoiresconquis += 1
16         DefensePerdu+= D[i]
17     return A + territoiresconquis,territoiresconquis, DefensePerdu
```

**Python** : Programme simulant une confrontation sur plusieurs territoires

# La meilleure attaque c'est la défense



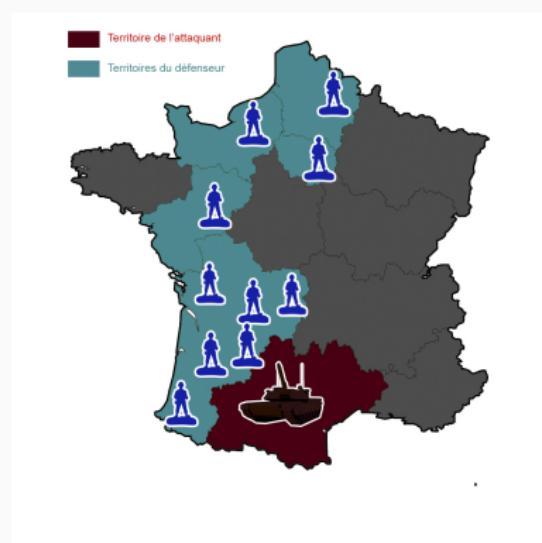
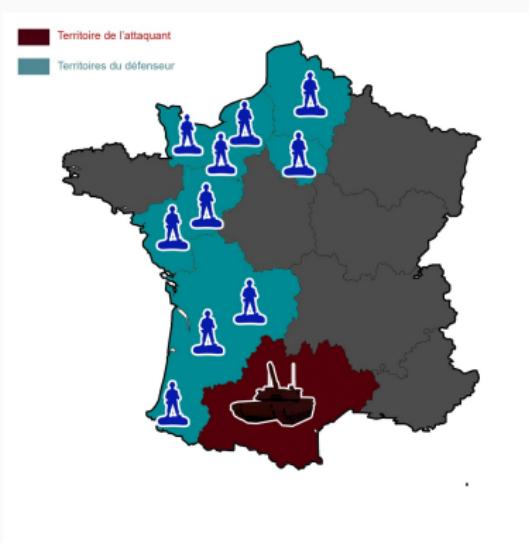
- Quasi-linéaire : les stratégies offensives ne sont pas décisives.
- In fine, les stratégies défensives pourraient déterminer l'issue du jeu!

**Figure 7:** Nombre de territoires conquis selon le nombre d'attaquants déployés

Considérons l'exemple suivant :

- L'attaquant possède un nombre donné de troupes.
- Le défenseur étend 10 troupes sur 5 territoires.

Plusieurs répartitions, c'est-à-dire de manières de placer les troupes défensives sont alors possibles.



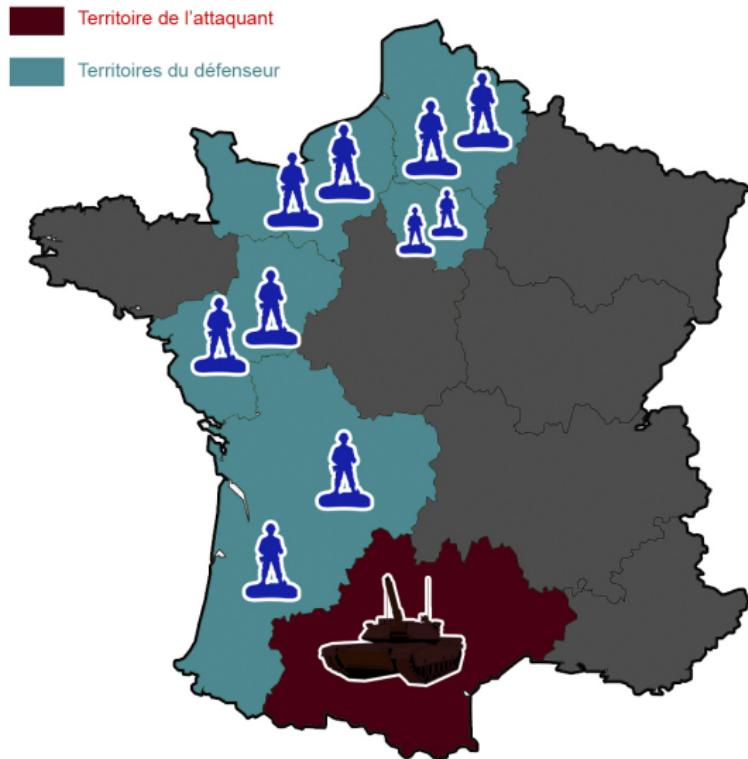
## Equivalence de certaines configurations

Dans le cas où il y a 5 territoires et 10 défenseurs on trouve, grâce à la **formule des combinaisons avec répétition**, qu'il existe 126 configurations possibles.

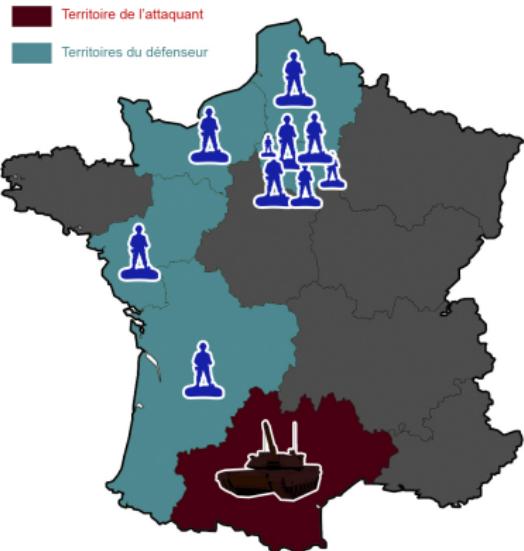
On vient donc à considérer certains scénarios comme équivalents afin de minimiser les calculs.

On retiendra notamment trois grandes configurations.

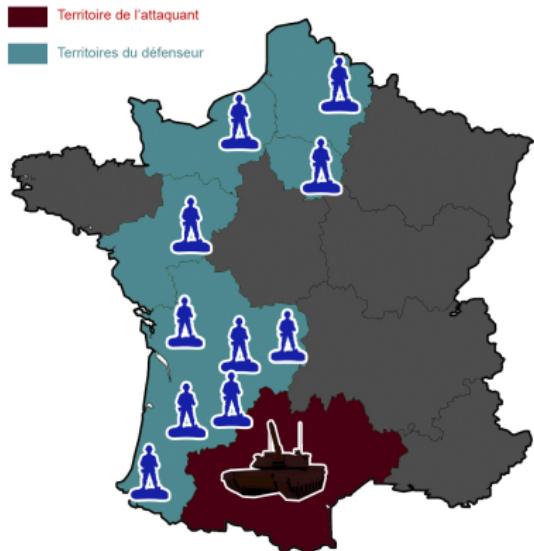
## Configuration 2-2



# Configurations égoïstes

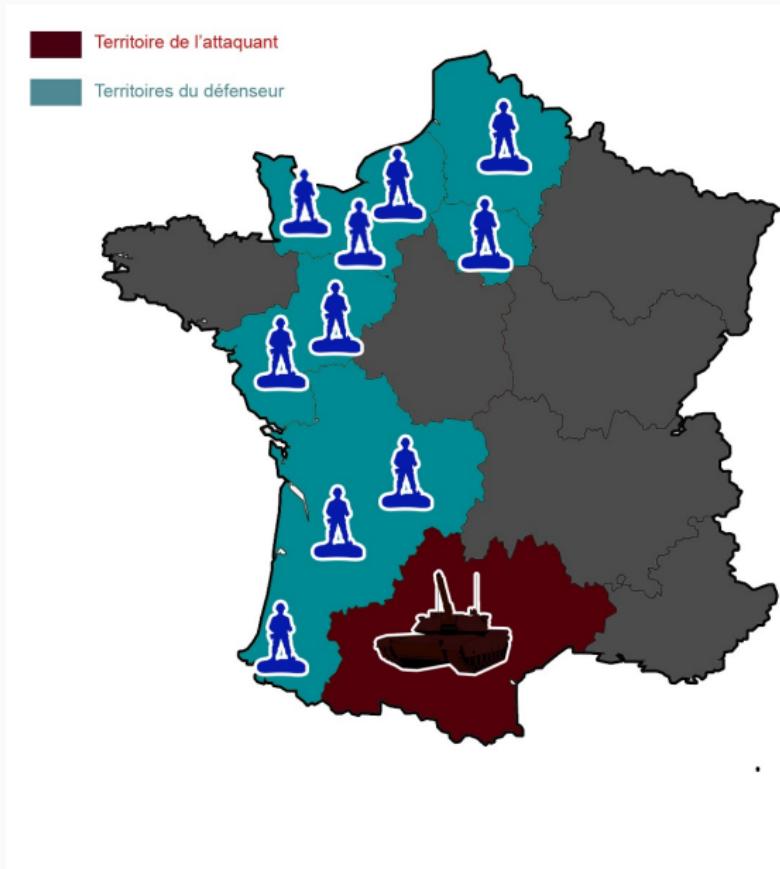


**Figure 8:** Paris à tout prix !



**Figure 9:** Tous au front !

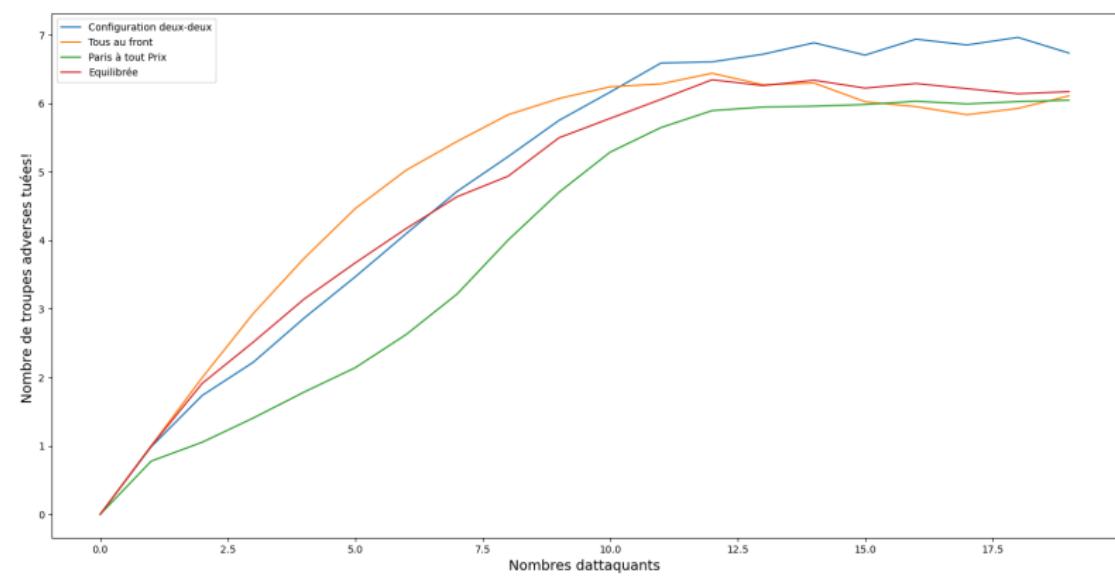
# Configuration dite équilibrée



# Représentation sur python et utilisation de notre programme

```
1 #Création des configurations possibles de défenses
2 Dd=[2,2,2,2,2] #Territoires
3 Patp=[1,1,1,1,6]
4 TAF=[6,1,1,1,1]
5 Eq=[3,2,1,1,3]
6 Att=[i for i in range(20)]
7 Deuxdeux=[0 for i in range(20)]
8 # Initialisation des résultats selon i le nombre d'attaquants
9 Touspourun=[0 for i in range(20)]
10 Equilibre=[0 for i in range(20)]
11 Parisatoutprix=[0 for i in range(20)]
12 for i in range(1000):
13     for j in range(20):
14         Deuxdeux[j] += j - Simulationdeplateau(j,Da)[0]
15         # Nb de Troupes offensives tuées
16         Touspourun[j] += j-Simulationdeplateau(j,G)[0]
```

# Résultats selon les configurations



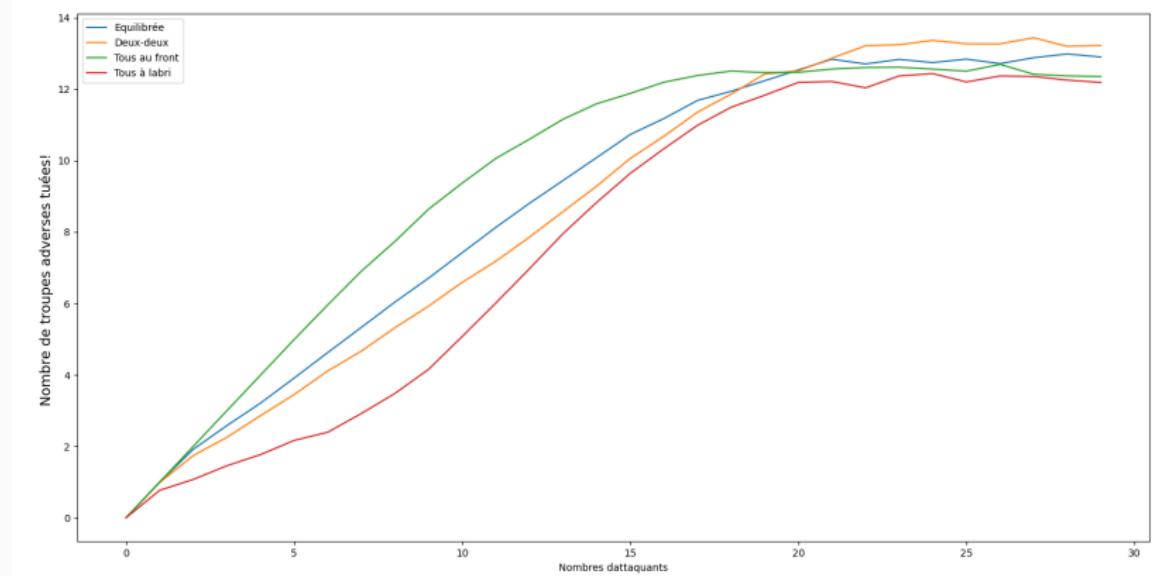
**Figure 11:** Eviter Paris à tout prix !

## Extension des résultats à une autre situation

En considérant 6 territoires consécutifs et 18 troupes, on peut accentuer les écarts entre les différentes configurations.

- **Deux-deux** = [2,2,2,2,2,6]
- **Equilibrée** = [3,3,3,3,3,3]
- **Front** =[13,1,1,1,1,1]

# Résultats sur 8 territoires



**Figure 12:** Nombre de troupes éliminées selon la configuration

# La défense optimale ?

## Conjecture

Face à un nombre d'opposants équivalent ou supérieur, le défenseur a tout intérêt à étendre ses troupes en groupe de deux afin de maximiser les pertes adverses.

## Les limites de notre étude

- Caractère unidimensionnel.
- Alliances.
- Le retrait possible de l'attaquant.

# Jouer pour s'amuser



# Fin

Merci de m'avoir écouté !

## **Annexes**

---

## Annexes

dés lancers	1	2	3	4	5	6			
1	1/6	1/6	1/6	1/6	1/6	1/6			
	$\mathbb{P}(Y^1 = k)$	1/36	3/36	5/36	7/36	9/36	11/36		
2		$\mathbb{P}(Y^2 = k)$	11/36	9/36	7/36	5/36	3/36	1/36	
		$\mathbb{P}(Y^1 = k)$	1/216	7/216	19/216	37/216	61/216	91/216	
3			$\mathbb{P}(Y^2 = k)$	16/216	40/216	52/216	52/216	40/216	16/216

**Tableau:** Probabilités qu'un chiffre soit le 1er/2ème plus grand

# Annexe

```
1 def deuxiememaximum(lst):
2     if len(lst) < 2:
3         return None
4     # Pas de deuxième maximum si la liste a moins de 2 éléments
5     # Trouver le maximum initial
6     max1 = max(lst)
7     # Supprimer le maximum initial de la liste
8     lst.remove(max1)
9     # Trouver le deuxième maximum dans la liste modifiée
10    max2 = max(lst)
11    return max2
12
13
14
```

1/1

---

**Python** : Fonction pour le deuxième maximum

# Annexe

```
1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 #Situation de 1vs1
6 def simulationunversusun(n):
7     Att = 0 # Compteur du nombre de victoires offensives
8     Def=0
9     for i in range(n): #n-simulations
10         a=random.randint(1,6)
11         d=random.randint(1,6)
12         if d > a:
13             Att+=1
14         else:
15             Def+=1
16     return (Att/n, Def/n)
17     # Renvoie le pourcentage de victoire de chaque côté
18
19
```

**Python :n-simulations d'une confrontation "un versus un"**

```
1 #Situation de 1vs2
2 def simulationunversusdeux(n):
3     Att=0
4     Def=0
5     for i in range(n):
6         de1=random.randint(1,6) #dé offensif
7         de2=random.randint(1,6)
8         de3=random.randint(1,6)
9         if de1 > de2 and de1>de3:
10             Att+=1
11         else:
12             Def +=1
13     return Att/n, Def/n
14
```

1/1

**Python :n-simulations d'une confrontation "un versus deux"**

# Annexe

```
1 #Simulation de 2v1
2 def simulationdeuxversusun(n):
3     Att = 0
4     Def = 0
5     #le défenseur gagne un duel et l'attaquant aussi
6     for i in range(n): # n-simulations
7         de1= random.randint(1,6) # De offensif 1
8         de2=random.randint(1,6) #De offensif 2
9         de3=random.randint(1,6)
10        if de3>=de1 and de3>=de2:
11            Def+=1
12        if de1>de3 or de2>de3:
13            Att+=1
14    return Def/n, Att/n
15
16
17
```

```
1 #Situation de 2v2
2 def simulationdeuxversusdeux(n):
3     Att = 0#Compteur du nombre de double-duels offensifs gagnés
4     Def = 0
5     Egalite = 0
6     #le défenseur gagne un duel et l'attaquant aussi
7     for i in range(n):
8         de1= random.randint(1,6)
9         de2=random.randint(1,6)
10        de3=random.randint(1,6)
11        de4=random.randint(1,6)
12        Attaque = [de1, de2]
13        Defense = [de3,de4]
14        AttaqueMax= max(de1,de2)
15        DefenseMax= max(de3, de4)
16        Attaque2 = deuxiememaximum(Attaque)
17        Defense2=deuxiememaximum(Defense)
18        if AttaqueMax>DefenseMax and Attaque2>Defense2:
19            Att+=1
20        if AttaqueMax>DefenseMax and Defense2>=Attaque2:
21            or DefenseMax>=AttaqueMax and Defense2<=Attaque2:
22            Egalite +=1
23        if DefenseMax>=AttaqueMax and Defense2>=Attaque2:
24            Def+=1
25    return Egalite/n, Def/n, Att/n
```

**Python :n-simulations d'une confrontation de "deux versus un"**

**Python :n-simulations d'une confrontation de "deux versus deux"**

## Annexe

```
1 #Le code suivant permettra de simuler n fois la situation de 3vs2
2 def simulationtroisdeux(n):
3     Att = 0
4     Egalite = 0
5     Def = 0
6     for i in range(n):
7         a=random.randint(1,6)
8         b=random.randint(1,6)
9         c=random.randint(1,6)
10        d=random.randint(1,6)
11        e=random.randint(1,6)
12        Attaque=[a,b,c]
13        Defense=[d,e]
14        AttaqueMax = max(a,b,c)
15        Attaque2 = deuxiememaximum(L)
16        DefenseMax= max(d,e)
17        Defense2=deuxiememaximum(L2)
18        if AttaqueMax>DefenseMax and Attaque2>Defense2:
19            Att+=1
20        if AttaqueMax>DefenseMax and Defense2>=Attaque2 or
21        DefenseMax>=AttaqueMax and Defense2<Attaque2:
22            Egalite+=1
23        if DefenseMax>=AttaqueMax and Defense2>=Attaque2:
24            Def+=1
25    return Egalite/n, Def/n, Att/n
```

## Annexe

Notons

- $X_1$  la variable aléatoire du plus grand dé défensif
- $X_2$  celle du deuxième maximum défensif
- $Y_1$  la variable aléatoire qui donne le plus grand dé offensif tiré
- $Y_2$  donnant le deuxième maximum offensif

On note  $\mathbb{P}_{ijk}$  la probabilité que pour i dés offensifs, j défensifs que le défenseur perde k troupes.

## Annexe

### Calculs pour les premiers cas

$$\mathbb{P}_{110} = P(X_1 \geq Y_1) = \sum_{y_1=1}^6 P(X_1 \geq y_1)P(Y_1 = y_1)$$

$$\mathbb{P}_{110} = \sum_{y_1=1}^6 \sum_{x_1=y_1}^6 P(X_1 = x_1)P(Y_1 = y_1) = \frac{21}{36}$$

$$\mathbb{P}_{111} = P(Y_1 > X_1) \quad \mathbb{P}_{111} = \sum_{x_1=1}^6 P(Y_1 > x_1)P(X_1 = x_1)$$

$$= \sum_{x_1=1}^6 \sum_{y_1=x_1+1}^6 P(Y_1 = y_1)P(X_1 = x_1) = \frac{15}{36}$$

$$\mathbb{P}_{120} = P(X_1 \geq Y_1) \quad \mathbb{P}_{120} = \sum_{y_1=1}^6 P(X_1 \geq y_1)P(Y_1 = y_1)$$

$$= \sum_{y_1=1}^6 \sum_{x_1=y_1}^6 P(X_1 = x_1)P(Y_1 = y_1) = \frac{161}{216} \quad \mathbb{P}_{121} = 1 - \mathbb{P}_{110} = \frac{91}{216}$$

$$\mathbb{P}_{210} = P(X_1 \geq Y_1, X_1 \geq Y_2)$$

$$\mathbb{P}_{210} = \sum_{y_1=1}^6 \sum_{y_2=1}^6 P(X_1 \geq y_1)P(X_1 \geq y_2)P(Y_1 = y_1, Y_2 = y_2)$$

$$= \frac{91}{216}$$

$$\mathbb{P}_{211} = 1 - \mathbb{P}_{210} = \frac{125}{216}$$

## Annexe

### Calculs pour deux défenseurs/attaquants

$$\mathbb{P}_{220} = P(X_1 \geq Y_1, X_1 \geq Y_2)$$

$$\mathbb{P}_{220} = \sum_{y_1=1}^6 \sum_{y_2=1}^{y_1} P(X_1 \geq y_1) P(X_2 \geq y_2) P(Y_1 = y_1, Y_2 = y_2)$$

$$\mathbb{P}_{220} = \sum_{y_1=1}^6 \sum_{y_2=1}^{y_1} \sum_{x_1=y_1}^6 \sum_{x_2=y_2}^{x_1} P(X_1 = x_1, X_2 = x_2) P(Y_1 = y_1, Y_2 = y_2) = \frac{581}{1296}$$

$$\mathbb{P}_{222} = P(Y_1 > X_1, Y_2 > X_2)$$

$$\mathbb{P}_{222} = \sum_{x_1=1}^6 \sum_{x_2=1}^{x_1} P(Y_1 \geq x_1) P(Y_2 > x_2) P(X_1 = x_1, X_2 = x_2)$$

$$\mathbb{P}_{222} = \sum_{x_1=1}^6 \sum_{x_2=1}^{x_1} \sum_{y_1=x_1+1}^6 \sum_{y_2=x_2+1}^{y_1} P(X_1 = x_1, X_2 = x_2) P(Y_1 = y_1, Y_2 = y_2) = \frac{295}{1296}$$

$$\mathbb{P}_{221} = 1 - \mathbb{P}_{222} - \mathbb{P}_{220} = \frac{420}{1296}$$

## Annexe

### Calculs pour 3 attaquants et 1 défenseur

$$\mathbb{P}_{310} = P(X_1 \geq Y_1, X_1 \geq Y_2)$$

$$\begin{aligned}\mathbb{P}_{310} &= \sum_{y_1=1}^6 \sum_{y_2=1}^6 P(X_1 \geq y_1) P(X_1 \geq y_2) P(Y_1 = y_1, Y_2 = y_2) \\ &= \frac{441}{1296}\end{aligned}$$

$$\mathbb{P}_{311} = 1 - \mathbb{P}_{310}$$

## Annexe

### Calculs pour 3 attaquants et 2 défenseurs

$$\mathbb{P}_{320} = P(X_1 \geq Y_1, X_1 \geq Y_2)$$

$$\begin{aligned}\mathbb{P}_{320} &= \sum_{y_1=1}^6 \sum_{y_2=1}^{y_1} P(X_1 \geq y_1) P(X_2 \geq y_2) P(Y_1 = y_1, Y_2 = y_2) \\ &= \end{aligned}$$

$$\begin{aligned}&\sum_{y_1=1}^6 \sum_{y_2=1}^{y_1} \sum_{x_1=y_1}^6 \sum_{x_2=y_2}^{x_1} P(X_1 = x_1, X_2 = x_2) P(Y_1 = \\ &y_1, Y_2 = y_2) = \frac{2275}{7776} \end{aligned}$$

$$\mathbb{P}_{322} = P(Y_1 > X_1, Y_2 > X_2)$$

$$\begin{aligned}\mathbb{P}_{322} &= \sum_{x_1=1}^6 \sum_{x_2=1}^{x_1} P(Y_1 \geq x_1) P(Y_2 > x_2) P(X_1 = x_1, X_2 = x_2) \\ &= \end{aligned}$$

$$\begin{aligned}&\sum_{x_1=1}^6 \sum_{x_2=1}^{x_1} \sum_{y_1=x_1+1}^6 \sum_{y_2=x_2+1}^{y_1} P(X_1 = x_1, X_2 = x_2) P(Y_1 = \\ &y_1, Y_2 = y_2) = \frac{2890}{7776} \end{aligned}$$

$$\mathbb{P}_{321} = 1 - \mathbb{P}_{322} - \mathbb{P}_{320} = \frac{2611}{7776}$$

# Annexe

```
1 dico={}
2 def gagnant(A, B): #Chance de victoire totale pour A attaquant
3     if (A,B) in dico:
4         return dico[(A,B)]
5     if A <= 0:
6         dico[(A,B)] = 0
7         return 0
8     if B <=0:
9         dico[(A,B)] = 1
10        return 1
11    if A == 1 and B == 1:
12        dico[(A,B)] = 15/36
13        return 15/36
14    if A == 1 and B == 2:
15        dico[(A,B)] = 55/216 * (15/36)
16        return 55/216 * (15/36)
17    if A == 2 and B == 1:
18        dico[(A,B)] = 125/216 + 91/216 * (15/36)
19        return 125/216 + 91/216 * (15/36)
20    if A == 2 and B == 2:
21        dico[(A,B)] = (295/1296) + (420/1296) * (15/36)
22        return (295/1296) + (420/1296)*(15/36)
23    if A == 3 and B == 1:
24        dico[(A,B)] = 855/1296 + 441/1296 * 125/216 + 91/216*15/36*
25 441/1296
26        return 855/1296 + 441/1296 * 125/216 + 91/216*15/36* 441/1296
27    if A== 3 and B==2:
28        dico[(A,B)] = 2890/7776
29    if A == 1 and B ==3:
30        dico[(A,B)] = 0.02702
31        return 0.02702
32    else:
33        dico[(A,B)] = 2611/7776 * gagnant(A-1,B-1) + 2278/7776 *
34 gagnant(A-2,B) + 2890/7776 * gagnant(A,B-2)
35        return dico[(A,B)]
```

## Python : Programmation dynamique

# Annexe

```
1 #Situation de 1vs1
2 def unversusun(n):
3     NAlost = 0
4     NDlost=0
5     a=random.randint(1,6)
6     d=random.randint(1,6)
7     if d > a:
8         NDlost+=1
9     else:
10        NAlost+=1
11    return NAlost,NDlost
```

**Python** : Simulation qui donne le nombre de troupes perdues pour un duel " un versus un "

```
1 #Situation de 1vs2
2 def unversusdeux(n):
3     NDlost=0
4     NAlost=0
5     a=random.randint(1,6)
6     b= random.randint(1,6)
7
8     d=random.randint(1,6)
9     if a > b and a > d :
10        NDlost=1
11    else:
12        NAlost=1
13    return (NAlost, NDlost)
```

**Python** : Simulation qui donne le nombre de troupes perdues pour un duel " un versus deux "

# Annexe

```
1 #Situation de 2v2
2 def deuxversus2(n):
3     NDlost = 0
4     NAlost = 0
5     de1= random.randint(1,6)
6     de2=random.randint(1,6)
7     de3=random.randint(1,6)
8     de4=random.randint(1,6)
9     Attaque = [de1, de2]
10    Defense = [de3,de4]
11    AttaqueMax= max(de1,de2)
12    DefenseMax= max(de3, de4)
13    Attaque2 = find_second_maximum(Attaque)
14    Defense2=find_second_maximum(Defense)
15    if AttaqueMax>DefenseMax and Attaque2>Defense2:
16        NDlost=2
17        if AttaqueMax>DefenseMax and Defense2>=Attaque2 or
18        DefenseMax>=AttaqueMax and Defense2<Attaque2:
19            NAlost = 1
20            NDlost=1
21            if DefenseMax>=AttaqueMax and Defense2>=Attaque2:
22                NAlost=2
23        return (NAlost,NDlost)
24
```

```
1 def troisversusun(n):
2     NAlost = 0
3     NDlost = 0
4     a=random.randint(1,6)
5     b=random.randint(1,6)
6     c=random.randint(1,6)
7     d=random.randint(1,6)
8     if d >= a and d>=b and d>= c:
9         NAlost=1
10    else:
11        NDlost=1
12    return NAlost,NDlost
13
```

## Annexe

```
1 def troisversusdeux(n):
2     NAlost = 0
3     NDlost = 0
4     a=random.randint(1,6)
5     b=random.randint(1,6)
6     c=random.randint(1,6)
7     d=random.randint(1,6)
8     e=random.randint(1,6)
9     L=[a,b,c]
10    L2=[d,e]
11    AttaqueMax = max(a,b,c)
12    Attaque2 = find_second_maximum(L)
13    DefenseMax= max(d,e)
14    Defense2=find_second_maximum(L2)
15    if AttaqueMax>DefenseMax and Attaque2>Defense2:
16        NDlost =2
17        if AttaqueMax>DefenseMax and Defense2>=Attaque2
18    or DefenseMax>=AttaqueMax
19        and Defense2<Attaque2:
20            NAlost=1
21            NDlost = 1
22        if DefenseMax>=AttaqueMax and Defense2>=Attaque2:
23            NAlost=2
24    return (NAlost,NDlost)
```

## Annexe

```
1 def nbtroupesrestantes(A,D):
2     while A!=0 and D!=0:
3         if A >=3 and D >=2:
4             A,D= A-troisversusdeux(1)[0],D - troisversusdeux(1)[1]
5         if A >=3 and D == 1 :
6             A,D = A- troisversusun(1)[0], D -troisversusun(1)[1]
7         if D >= 2 and A == 2 :
8             A,D= A-deuxversus2(1)[0],D - deuxversus2(1)[1]
9         if A==1 and D >=2:
10            A,D= A-unversusdeux(1)[0],D - unversusdeux(1)[1]
11        if A==2 and D == 1:
12            A,D = A-deuxversusun(1)[0],D- deuxversusun(1)[1]
13        if A == 1 and D ==1:
14            A,D = A-unversusun(1)[0],D - unversusun(1)[1]
15    return (A,D)
16
```

# Annexe

```
1 #Création des configurations possibles de défenses
2 Dd=[2,2,2,2,2] #Territoires
3 Patp=[1,1,1,1,6]
4 TAF=[6,1,1,1,1]
5 Eq=[3,2,1,1,3]
6 Att=[i for i in range(20)]
7 Deuxdeux=[0 for i in range(20)]
8 # Initialisation des résultats selon i le nombre d'attaquants
9 Touspourun=[0 for i in range(20)]
10 Equilibre=[0 for i in range(20)]
11 Parisatoutprix=[0 for i in range(20)]
12 for i in range(1000):
13     for j in range(20):
14         Deuxdeux[j] += j - Simulationdeplateau(j,Da)[0]
15         # Nb de Troupes offensives tuées
16         Touspourun[j] += j-Simulationdeplateau(j,G)[0]
17         Equilibre[j] += j-Simulationdeplateau(j,K)[0]
18         Parisatoutprix[j] += j-Simulationdeplateau(j,L)[0]
19 for i in range(len(Deuxdeux)):
20     Deuxdeux[i] = Deuxdeux[i] /1000
21     Touspourun[i] = Touspourun[i] /1000
22     Equilibre[i] = Equilibre[i] /1000
23     Parisatoutprix[i] = Parisatoutprix[i] /1000
24 plt.plot(Att,Deuxdeux, label='Configuration deux-deux')
25 plt.plot(Att,Touspourun, label='Tous au front')
26 plt.plot(Att,Parisatoutprix, label='Paris à tout Prix')
27 plt.plot(Att,Equilibre,label='Equilibrée')
28 plt.legend()
29 axes = plt.gca()
30 axes.set_xlabel('Nombres d'attaquants', fontsize=14)
31 axes.set_ylabel('Nombre de troupes adverses tuées!', fontsize=14)
32 plt.show()
```

## Python : Simulation sur 5 territoires

# Annexe

```
1 Eq=[3,3,3,3,3,3]
2 Dd=[2,2,2,2,2,8]
3 Tpu=[13,1,1,1,1,1]
4 Upt=[1,1,1,1,1,13]
5 Deuxdeux=[0 for i in range(30)]
6 Touspourun=[0 for i in range(30)]
7 Equilibre=[0 for i in range(30)]
8 Unpourtous=[0 for i in range(30)]
9 W=[0 for i in range(20)]
10 Woippy =[0,0,0,0]
11 for i in range(1000):
12     for j in range(30):
13         Deuxdeux[j] += j - Simulationdeplateau(j,Eq)[0] # Nb de Troupes tuées
14         Touspourun[j] += j-Simulationdeplateau(j,Tpu)[0]
15         Equilibre[j] += j-Simulationdeplateau(j,Eq)[0]
16         Unpourtous[j]+=j-Simulationdeplateau(j,Upt)[0]
17         for i in range(len(Deuxdeux)):
18             Deuxdeux[i] = Deuxdeux[i] /1000
19             Touspourun[i] = Touspourun[i] /1000
20             Equilibre[i] = Equilibre[i] /1000
21             Unpourtous[i] = Unpourtous[i]/1000
22 A=[i for i in range(30)]
23 plt.plot(A,Deuxdeux, label='Equilibrée')
24 plt.plot(A,Touspourun, label='Deux-deux')
25 plt.plot(A,Equilibre, label='Tous au front')
26 plt.plot(A,Unpourtous, label='Tous à labri')
27 plt.legend()
28 axes = plt.gca()
29 axes.set_xlabel('Nombres dattaquants')
30 axes.set_ylabel('Nombre de troupes adverses tuées!', fontsize = 14)
31 plt.show()
```

## Python: Simulation sur 8 territoires

## Références

-  Baris Tan (1997)  
**Markov chains and the RISK board game**  
*Mathematics Magazine* 70:5, 349-357
-  Richard L.Bishop (1972)  
**Monopoly as a Markov Process**  
*Mathematics Magazine* 45(1):26-29
-  Abraham Neyman (2003)  
**Stochastic Games and Applications**
-  Hasbros  
**Règles du jeu Risk**
-  Ehsan Honary (2007)  
**Total Diplomacy: The Art of Winning RISK**