

# # Trolls et Châteaux



Ce rapport présente notre travail sur le projet "Troll". Il est divisé en trois parties: la **première partie** étant le plan d'action que nous avons élaboré en amont pour préparer le concours.

La **deuxième partie** présente notre travail et les analyses de nos expérimentations.

La **troisième partie** est notre compte rendu du TP noté du 26/06/2019.

---

## # Partie 1

### ## Introduction:

Nous sommes une équipe de 2 programmeurs qui allons travailler sur le jeu de "Troll et châteaux".

Pour cela, nous développerons une application qui permettra de simuler plusieurs parties (pouvant conduire à des statistiques) ainsi qu'une animation de partie.

Nous implémenterons aussi plusieurs stratégies: stratégie prudente, une ou plusieurs stratégies mixtes.

### ## Configurations

```
* 7 cases, 15 pierres : 15 | x | _ | _ | T | _ | _ | x | 15
* 7 cases, 30 pierres : 30 | x | _ | _ | T | _ | _ | x | 30
* 15 cases, 30 pierres :
30 | x | _ | _ | _ | _ | _ | T | _ | _ | _ | _ | _ | x | 30
* 15 cases, 50 pierres :
50 | x | _ | _ | _ | _ | _ | T | _ | _ | _ | _ | _ | x | 50
```

### ## Stratégies

#### ## Stratégie nulle

S1	J1			x				J2	S2	
14	1	-	-	T	-	-	1	14		
13	1	-	-	T	-	-	1	13		
				...						
0	1	-	-	T	-	-	1	0		

Dead end.

```
### Stratégie perdante
```

S1	J1			x			J2	S2
14	1	_	T	_	_	_	2	13
13	1	T	_	_	_	_	2	11
12	T	_	_	_	_	_	2	9

```
### Stratégie : late game
```

Lorsque le troll se trouve à une case, le joueur lance plus de pierres.

S1	J1			x			J2	S2
14	1	_	T	_	_	_	2	13
13	1	T	_	_	_	_	2	11
10	3	_	T	_	_	_	2	9
7	3	_	_	T	_	_	2	7
4	3	_	_	_	T	_	2	5
1	3	_	_	_	_	T	2	3
0	1	_	_	_	T	_	2	1
0	0	_	_	T	_	_	*1	0

Match nul.

> Très risqué.

```
### Stratégie : jet de pierres en fonction du nombre de cases
```

Il y a 3 cases entre le château adverse et le troll :  $15 / 3 = 5$ .

Donc il faudrait lancer 5 pierres à chaque tour.

Tentative de contre en jouant un nombre supérieur :

S1	J1			x			J2	S2
9	6	_	_	_	T	_	5	10
3	6	_	_	_	_	T	5	5
0	3	_	_	_	T	_	5	0
0	0	_	_	_	_	_	T	0

Victoire.

Tentative de contre avec la stratégie late game :

S1	J1			x			J2	S2
14	1	_	T	_	_	_	5	10
13	1	T	_	_	_	_	5	5
7	6	_	T	_	_	_	5	0
0	*7	_	_	_	_	_	T	0

Victoire.

> Revient à la stratégie nulle si utilisée des deux côtés.

### Stratégie : défense

Le but est de maintenir le troll à x+1 ou x-1 case jusqu'à la fin.

Pour cela, il faut :

- \* Prendre l'avantage au premier tour (si on se réfère à la stratégie précédente, ce serait 5)
- \* Réaliser des jets de pierres égaux à ceux de l'adversaire

Prédire le jet de pierre de l'adversaire est assez compliqué.

À la place, on pourrait alterner entre un jet de 1 et un jet de rééquilibrage entre les tours (plutôt entre 2 et 3).

S1	J1			x			J2	S2
10	5	_	_	_	T	_	1	14
9	1	_	_	_	T	_	1	13
8	1	_	_	T	_	_	2	11
6	2	_	_	T	_	_	2	9
4	2	_	_	_	T	_	1	8
3	1	_	_	T	_	_	3	5
1	2	_	T	_	_	_	3	2
0	1	T	_	_	_	_	2	0
0	T	_	_	_	_	_	0	0

Jouer un grand nombre au départ contre un joueur prudent n'est pas une bonne stratégie. Il faudrait donc se concentrer sur les stratégies du début de jeu.

1er tour :

J1\J2	1	2	2	4	5	total
1	0/0	1/-1	2/-1	3/-1	4/-1	6
2	-1/1	0/0	1/-1	2/-1	3/-1	3
3	-2/1	-1/1	0/0	1/-1	2/-1	0
4	-3/1	-2/1	-1/1	0/0	1/-1	-3
5	-4/1	-3/1	-2/1	-1/1	0/0	-6

2e tour :

		l'adversaire joue				1		2		3		4		5		Total		
je joue	écart +	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	
1		0	1	2	3	0	1	2	3	1	2	3	4	2	3	4	5	
2		0	1	2	3	0	1	2	3	0	1	2	3	1	2	3	4	
3		-1	0	1	2	0	1	2	3	0	1	2	3	0	1	2	3	
4		-2	-1	0	1	-1	0	1	2	0	1	2	3	0	1	2	3	
5		-3	-2	-1	0	-2	-1	0	1	-1	0	1	2	0	1	2	3	
% de garder l'avantage en fonction du coup joué et de l'écart.		15%				27%				39%				51%			63%	-
% de garder l'avantage en fonction de l'écart.	Somme écart 1 =	0%	Somme écart 2 =	25%	Somme écart 3 =	50%	Somme écart 4 =	75%						-			-	

L'avantage du stock de pierres joue aussi un rôle important.

S1   J1			x			J2   S2	
<hr/>							
14   1   _   T   _   _   _   2   13							
12   2   T   _   _   _   _   3   10							
2   10   _   T   _   _   _   1   9							
0   2   T   _   _   _   _   9   0							
0   T   _   _   _   _   _   0   0							

Déroulements possibles :

S1   J1			x			J2   S2	
<hr/>							
12   -   T   _   _   _   _   -   10							
2   10   T   _   _   _   _   10   0							
0   *2   _   _   T   _   _   0   0							

ou

S1   J1			x			J2   S2	
<hr/>							
12   -   T   _   _   _   _   -   10							
6   T   _   _   _   _   _   10   0							

Le joueur qui creuse l'écart de stock dès le début de la partie pourra prendre l'avantage sur un coup futur.

### ### Stratégie : s'adapter à la stratégie adverse

Cette stratégie se reposerait sur un historique des parties précédentes et évoluera donc au fil des parties.

Il peut être intéressant d'analyser les parties précédentes pour essayer de comprendre la stratégie adverse et essayer de la contrer.

### **## Simulations et statistiques**

Pour déterminer la meilleure stratégie, nous allons simuler plusieurs parties sur plusieurs catégories.

Les catégories correspondent aux configurations:

- \* 7 cases, 15 pierres
- \* 7 cases, 30 pierres
- \* 15 cases, 30 pierres
- \* 15 cases, 50 pierres

Chaque stratégie va affronter toutes les autres sur 4 configurations de jeu, définies par la longueur du chemin reliant les deux châteaux, et le stock initial de pierres dans chaque château. La simulation permettrait de simuler une IA qui affrontera une autre IA sur 1000 matchs par exemple.

- \* celle qui remporte le plus de matchs est déclarée vainqueur et gagne 3 points.
- \* En cas de match nul chacun gagne 1 point.
- \* Si coup invalide (c'est à dire un nombre de pierres négatif ou nul ou supérieur au stock de pierres restantes), alors disqualification ou alors -1 point.
- \* Si les deux stratégies proposent simultanément un coup invalide, elles sont toutes deux disqualifiées et perdent toutes deux 1 point.

L'IA qui aura le plus de points, toutes catégories confondues sera nommée meilleure IA du jeu.

Cette partie sera simulée dans l'application, permettant ainsi de simuler les 1000 parties. Notamment une interface où l'utilisateur peut saisir les paramètres de parties, le nombre de parties, etc.

Fin Partie 1

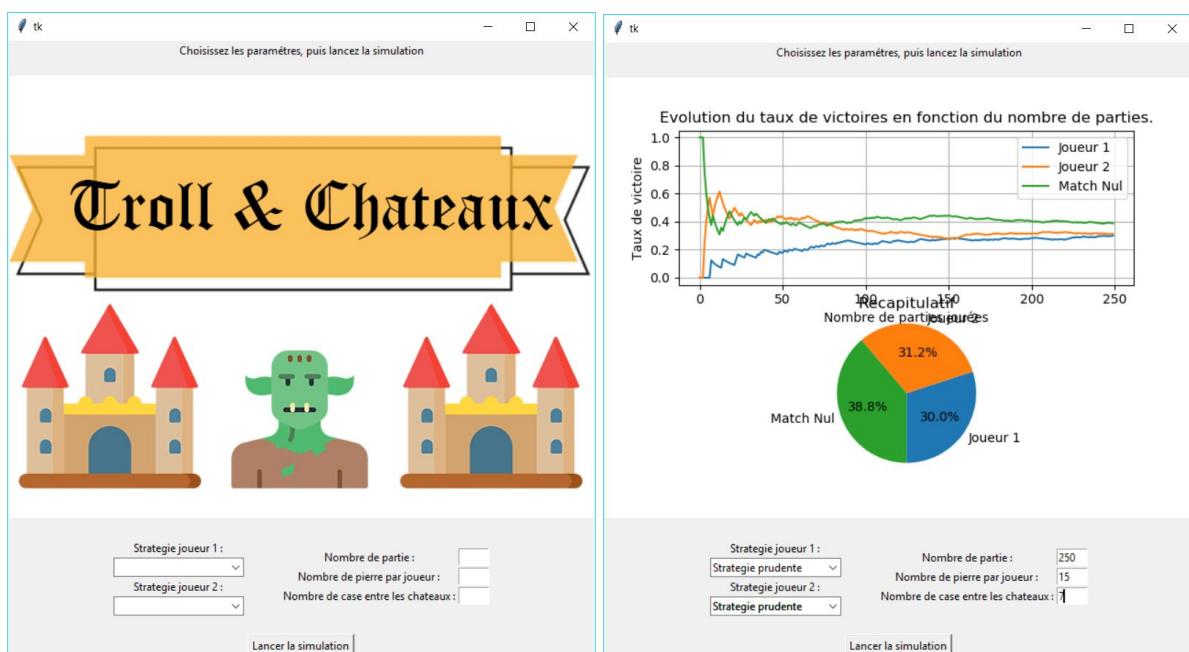
---

## # Partie 2

Cette deuxième partie implémente la stratégie prudente mixte vue en cours, et quelques stratégies basiques. Nous avons développé une interface graphique permettant de visualiser, avec graphiques à l'appui, l'issu de plusieurs parties entre deux stratégies.

### ## Interface graphique:

Afin de mieux présenter les résultats, nous avons mis en place une interface graphique permettant à l'utilisateur de visualiser les données sous forme graphique et de faire des simulations sur plusieurs parties.



### ### Librairies

Nous avons utilisé deux librairies: Matplotlib et Tkinter.

Matplotlib nous permet de tracer et de visualiser des données sous formes de graphiques.

Tkinter nous permet de créer des interfaces graphiques.

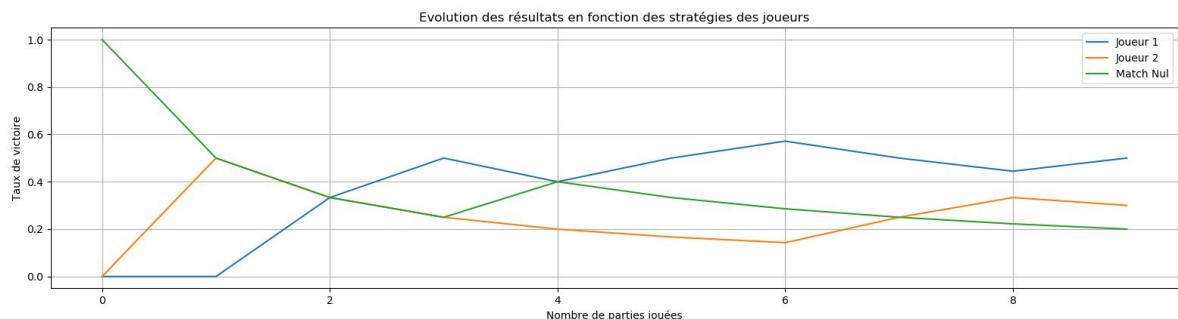
Nous avons utilisé ces deux librairies afin de proposer une interface user-friendly permettant à l'utilisateur de saisir les paramètres directement sur l'application (Choix de la stratégie du joueur 1 et du joueur 2, nombre de pierres, nombre de parties...).

### ### Paramètres et Graphiques

Au lancement de l'application, l'utilisateur doit saisir plusieurs paramètres :

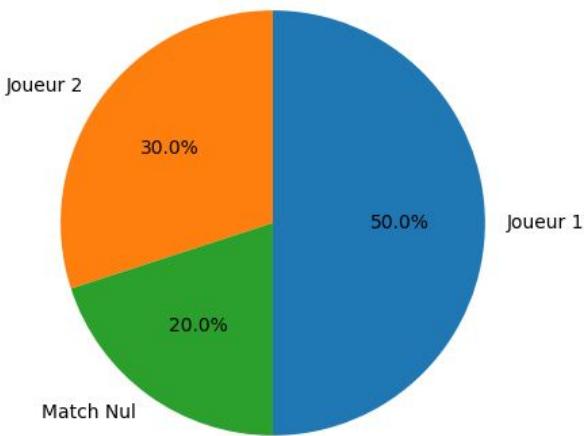
- La stratégie du joueur 1.
- La stratégie du joueur 2.
- Le nombre de parties qu'il souhaite simuler.
- Le nombre de pierres par joueur.
- Le nombre de case entre les châteaux.

Il suffit ensuite de cliquer sur "Lancer simulation" qui ouvrira une autre fenêtre affichant deux graphiques selon les paramètres qu'il a saisi.



Ce graphique représente le taux de parties gagnées par le joueur 1, le joueur 2, et le taux de matchs nuls au fur et à mesure des parties.

Recapitulatif



Ce graphique représente la moyenne des victoires pour les joueurs 1 et 2 pour N parties (ainsi que les matchs nuls).

L'intérêt de ce graphique est de visualiser rapidement l'issu de N parties entre deux stratégies.

### ## Stratégies:

Strategies.py :

Nous avons implémenté plusieurs stratégies, certaines très basiques, qui renvoient le même entier (renvoieCinq, renvoieQuatre...), d'autres aléatoires et la stratégie prudente.

Pour la stratégie prudente, on analyse le fichier pré-calculé correspondant au plateau actuel et en fonction du stock de pierres des joueurs et de la position du troll, on va chercher la solution optimale correspondante. On choisit alors le prochain coup joué parmi la distribution de probabilité appliquée sur le nombre de pierre à lancer.

Les configurations de parties ont par défaut un stock de 50 pierres et la taille du plateau est de 15 maximum. Pour ajouter de nouvelles configurations, il faut changer les valeurs N et M du fichier Configuration.py.

### **## PLNE:**

Configuration.py :

1. On crée un tableau qui énumère toutes les configurations possibles. Le tableau est de dimension  $N \times N$  où N est le stock maximal de pierres. La ligne et la colonne correspondent aux lancers possibles des deux joueurs. Chaque case du tableau comporte k cas où k est la position du troll sur le plateau (0 au centre).
2. Pour les cas de base on peut initialiser leur gain optimal (position du troll après confrontation du jet de pierres des joueurs) et en général, on initialise en fonction des paramètres donnés :
  - a. x le stock de pierres du joueur 1
  - b. y le stock de pierres du joueur 2
  - c. t la position du troll
  - d. m la taille du plateau
3. On calcule le gain optimal et la solution optimale lorsque toutes les configurations sont faites grâce à la résolution du PLNE avec la bibliothèque PuLP.
  - a. Dans le problème, gain optimal devient l'objectif à maximiser.
  - b. Pour le résoudre, on réalise une combinaison linéaire (cl) entre les gains optimaux ( $g_{opt}$ ) et leur indice qui viendra s'ajouter aux sous-contraintes ( $cl \geq g_{opt}$ ).
  - c. La dernière contrainte est que la somme de toutes les probabilités doit faire 1.
  - d. Après résolution du problème grâce à PuLP, on obtient ainsi le gain optimal et la solution optimale.
4. On sauvegarde les données de ces grandes simulations dans des fichiers (/data) pour pouvoir les étudier dans les stratégies.

Fin Partie 2

---

### # Partie 3

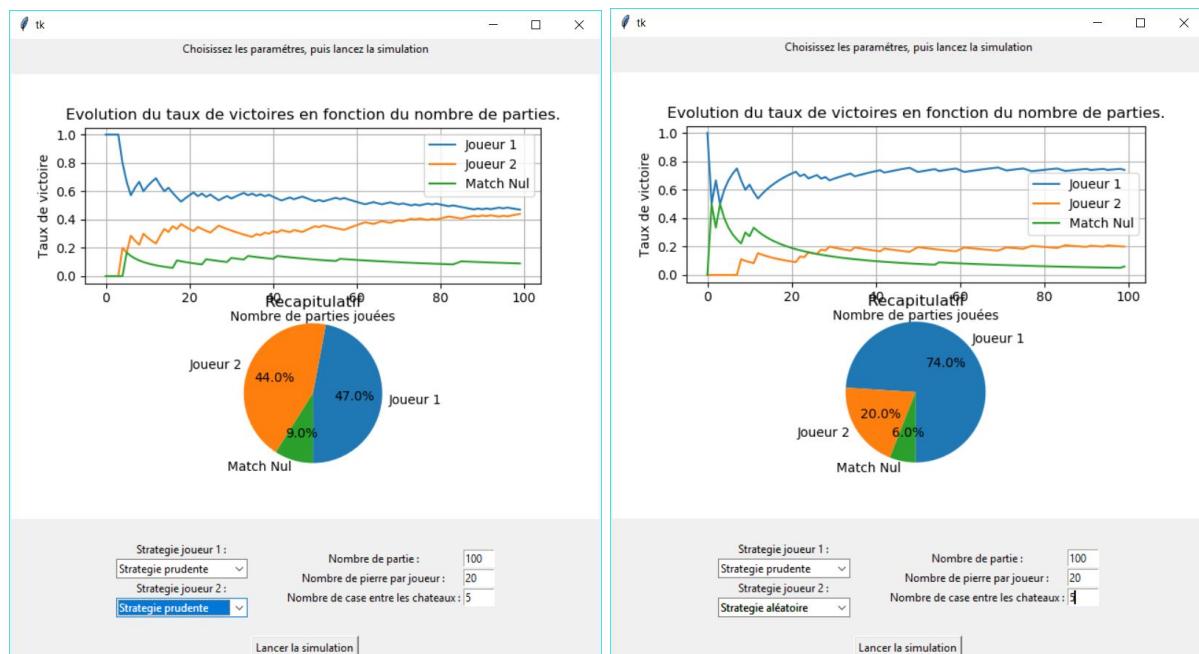
1. Supposons le joueur I prudent. Comment joue-t-il sur la configuration ( $n_1 = 20$ ;  $n_2 = 20$ ;  $t = 0$ ) ?

Nous avons lancé notre simulation en modifiant les paramètres du fichier Configuration.py. Le code nous retourne les résultats suivants:

```
g_opt:0.0,
s_opt:[0.025049335, 0.12495237, 0.065305469, 0.11085369, 0.0,
0.16495421, 0.03690677, 0.16593498, 0.050431658, 0.25561152, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

2. Précédemment,  $s_{opt}$  nous donne les probabilités du nombre de pierre à lancer pour la configuration  $n_1:20$ ,  $n_2:20$ ,  $t:0$ . Il ne jouera pas plus de la moitié de son stock.

Avec 25% de chance de lancer 10 pierres et de vider de moitié son stock et 0% de chance de lancer 5 pierres, ce qui semble bizarre.



En simulant cette configuration sur 100 parties avec notre interface en mettant les joueurs 1 et 2 en stratégie prudente (image 1), on remarque que le joueur 1 a un taux de parties gagnées légèrement supérieur à celle du joueur 2. Mais on peut dire que les deux joueurs ont à peu près la même chance de gagner.

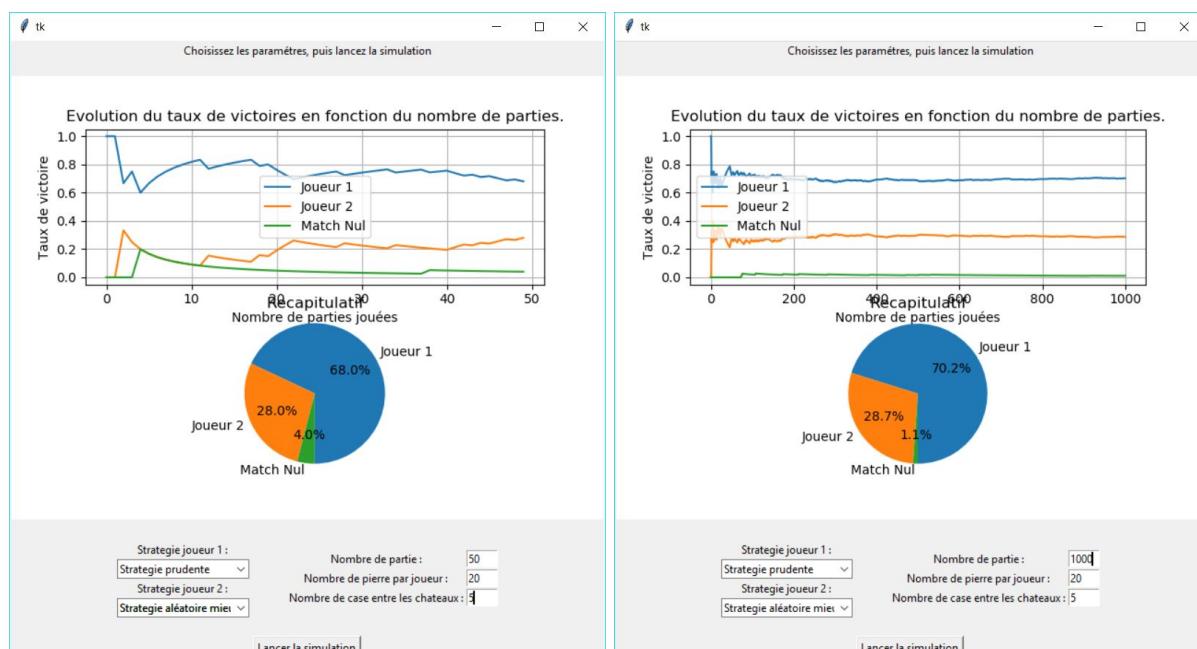
Cependant si le joueur 2 joue avec une stratégie aléatoire qui consiste à renvoyer un nombre de pierres compris entre 1 et le stock restant, on remarque que le taux de victoire du joueur 1 devient largement inférieur (image 2). Le joueur 1 a donc intérêt à jouer une stratégie prudente quelle que soit la stratégie du joueur 2. Il existe des limites comme si on confronte deux stratégies prudentes, le taux de victoire de chaque joueur sera à peu près équivalent. Elle est donc peu efficace contre des stratégies similaires ou plus évoluées.

Probabilité de lancer Configuration	1 Pierre	2 Pierres	3 Pierres	4 Pierres	5-15 Pierres
15 Pierres	0	46.12 %	4.89 %	48.99 %	0

Le tableau ci-dessus indique les probabilités du nombre de pierres à lancer pour une configuration de 15 pierres. Si le joueur 2 connaît la stratégie du joueur 1 ainsi que ses probabilités de jets, il peut voir que le joueur 1 a de grandes chances de lancer 2 ou 4 pierres et peut donc décider par exemple de jeter 2 pierres afin de faire match nul ou d'épuiser son adversaire de 2 pierres de plus.

3. Pour cette question, nous avons ajouté une nouvelle stratégie dans notre fichier Stratégies.py:

```
def renvoieAleaMieux(partie=None, partiesPrecedentes=None):
    return random.randint(1, (partie.stockDroite/2))
```



On remarque que la stratégie prudente du joueur 1 domine la stratégie aléatoire qui consiste à lancer un nombre de pierres choisi aléatoirement entre 1 et  $\lceil n/2 \rceil$ .

On a simulé avec un nombre de parties à 50 et 1000.

Plus le nombre de parties est important, plus le joueur 1 a un taux de victoire élevé. Le pourcentage de match nul diminue aussi. Le joueur 1 a donc intérêt à jouer sa stratégie prudente face à cette stratégie aléatoire du joueur 2.

Le gain du joueur I (jouant sa stratégie prudente) sur la configuration (15; 15; 0) est:

```
g_opt:0.0,
s_opt:[0.094513635, 0.1773064, 0.05050073, 0.1390776, 0.14054442,
0.15517793, 0.24287929, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
```

4.

(a) Pour cette question, nous avons aussi ajouté une nouvelle stratégie dans notre fichier Stratégies.py:

```
def renvoieAleaImpair(partie=None, partiesPrecedentes=None):
    return random.randrange(1,int(partie.stockDroite),2)
```

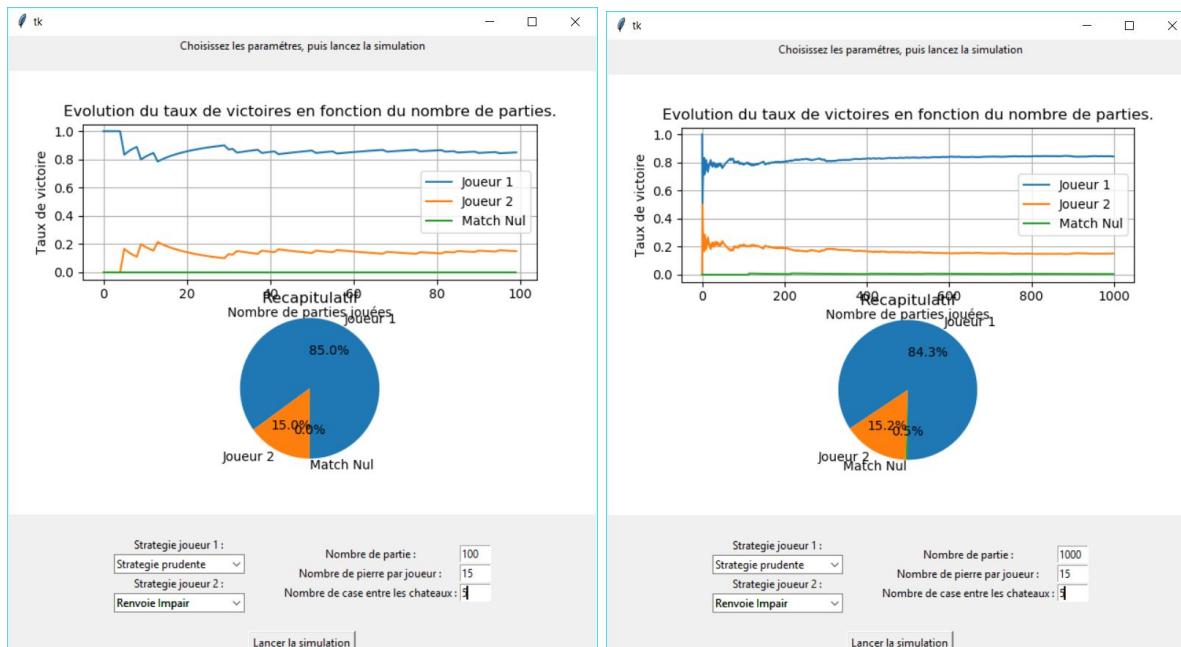
On peut regarder l'état de la partie en mettant la variable `affichageTexte` à `True` à la ligne 10 du fichier chart.p:

```
resultats.append(troll.jouerPartie(nb_case, nb_pierre, strat1, strat2,
affichageTexte=True))
```

Le joueur 2 renvoie donc bien à chaque partie un nombre impair de pierres:

[[15] \_ \_ [15]  
8 11  
[ 7] \_ [4]  
4 3  
[ 3] \_ [1]  
3 1  
[ 0] \_ [0]]

Victoire du joueur de gauche !



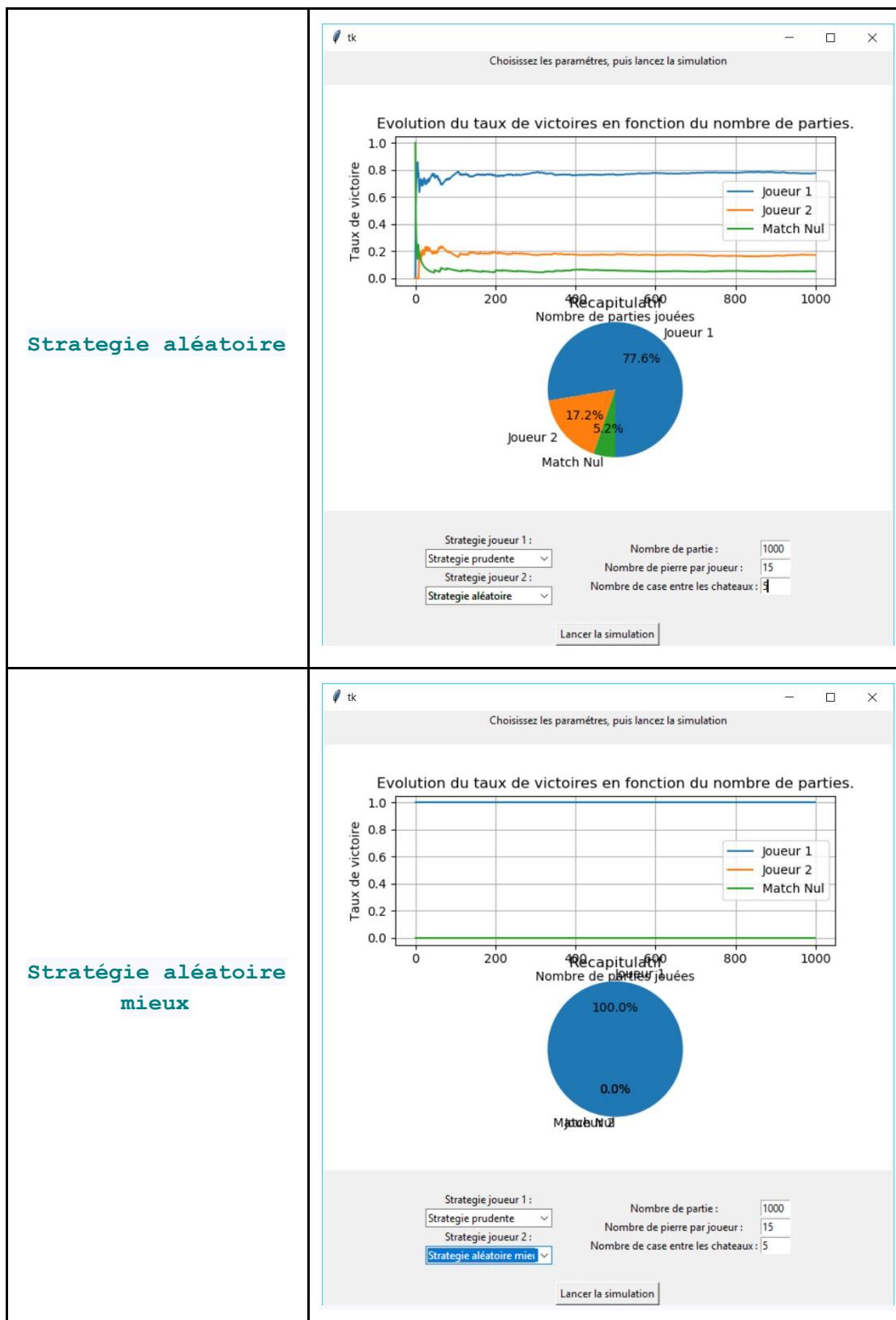
On remarque que la stratégie prudente du joueur 1 domine la stratégie aléatoire qui consiste à lancer un nombre de pierres impair.

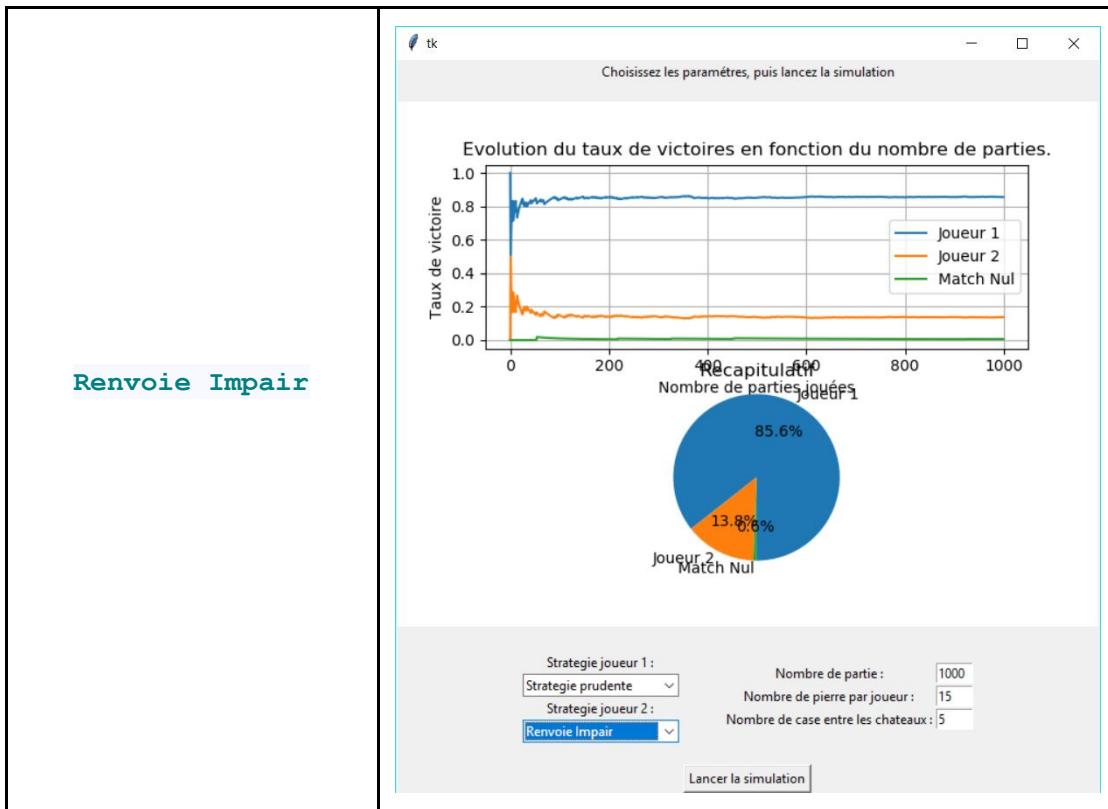
On a simulé avec un nombre de parties à 100 et 1000.

Le nombre de parties n'influe pas sur le taux de victoire du joueur 1. Le pourcentage de match nul augmente très peu aussi. Le joueur 1 a donc intérêt à jouer sa stratégie prudente face à cette stratégie aléatoire du joueur 2.

(b) Pour cette question, on a modifié la fonction `tourDeJeu` dans la fonction `troll.py`.

En testant cette fonctionnalité avec le joueur 1 en stratégie prudente et le joueur 2 avec différentes stratégies:





On remarque donc que la stratégie prudente du joueur 1 reste dominante par rapport aux autres stratégies et comportement du joueur 2. L'issu de plusieurs partie tend donc vers une victoire du joueur 1.

```
(c) p = 0.8
pNombreDroite = []
for i in range(0, nombreDroite+1):
    coeffBinomial =
        math.factorial(nombreDroite) / (math.factorial(nombreDroite-i)*math
        .factorial(i))
    pNombreDroite.append(coeffBinomial * (p ** i) * ((1 - p) **
        (nombreDroite-i)))
nombreDroite = numpy.random.choice(numpy.arange(nombreDroite+1),
1, p=pNombreDroite)
```

Sur les simulations avec  $p = 0,8$  où le joueur 1 ne renvoie que 4 et le joueur 2 que 5, le joueur 1 a autant de chance de gagner que le joueur 2.

