# Influence des règles d’un jeu sur l’issue d’une partie

Villars Allan
N°29903

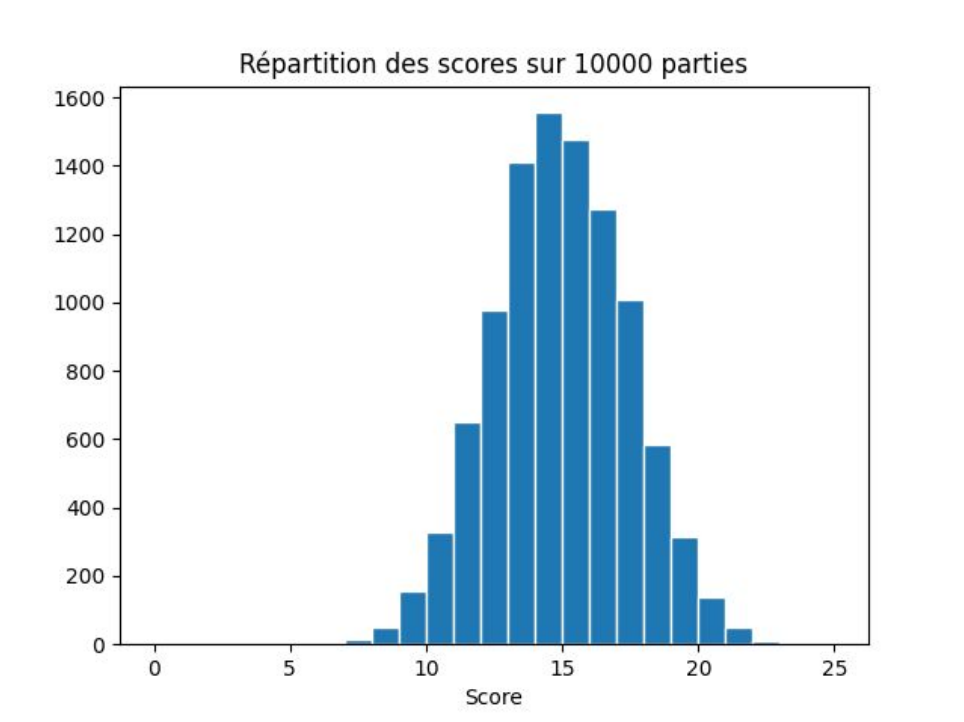
# Introduction

# Sommaire

# I.A. Simulation

- Simulation d'une partie de Hanabi
- Probabilités, espérances et écart-types
- Modèle approché

# I.A. Simulation




Exemple de simulation d’une partie dans les conditions standard avec nombre d’erreur illimité

# I.A. Simulation

Influence des paramètres

- Choix des cartes et nombre de cartes par mains
- Nombre de joueur
- Nombre de jetons erreur maximum
- Les proportions de cartes pour une couleur donnée

# I.A. Simulation

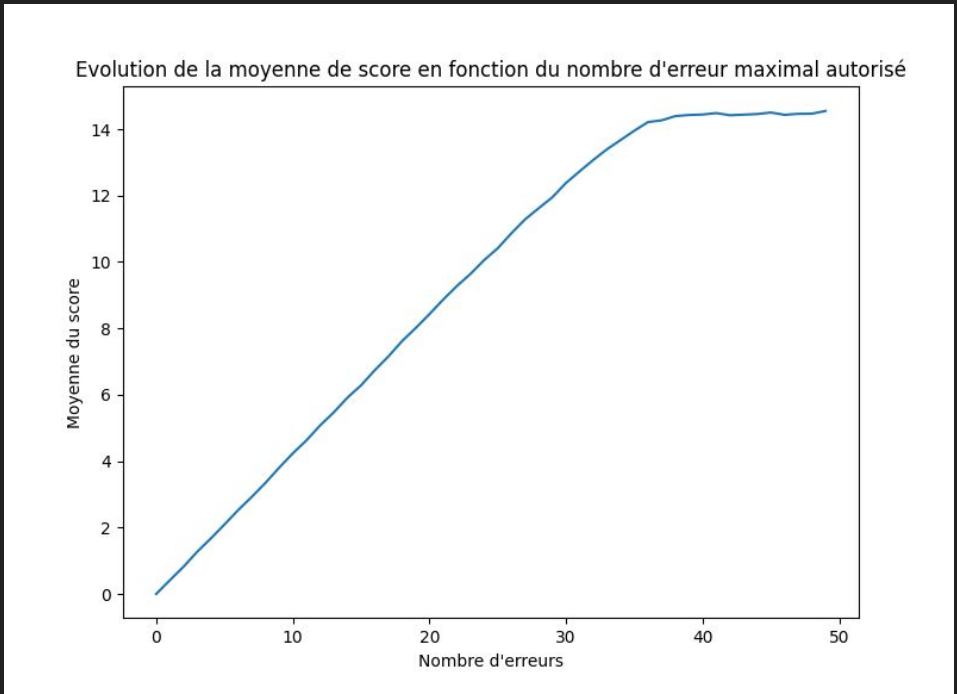
Calcul de l’espérance, de l’écart-type et de la médiane

$$\mathbb{E}[S] = \sum_{x \in S(\Omega)} x\mathbb{P}(S = x)$$

$$\mathbb{V}[S] = \sum_{x \in S(\Omega)} x^2\mathbb{P}(S = x) - \left( \sum_{x \in S(\Omega)} x\mathbb{P}(S = x) \right)^2$$

# I.B. Résultats

## Influence du nombre de jetons erreur maximum



# I.B. Résultats

## Influence du nombre de cartes par mains

```
Espérance :  14.455300000000001
Ecart-Type :  2.4619102156658736
```

```
Espérance :  14.445199999999996
Ecart-Type :  2.476367694830491
```

```
Espérance :  14.4627
Ecart-Type :  2.470710163090772
```

```
Espérance :  14.4551
Ecart-Type :  2.5099370490113966
```

```
Espérance :  14.443100000000003
Ecart-Type :  2.4784193329620154
```

```
Espérance :  14.457099999999999
Ecart-Type :  2.485550158415643
```

```
Espérance :  14.450000000000003
Ecart-Type :  2.494534024622624
```

```
Espérance :  14.458400000000001
Ecart-Type :  2.4457451706995075
```

```
Espérance :  14.519700000000002
Ecart-Type :  2.494115456429382
```

```
Espérance :  14.4458
Ecart-Type :  2.481826416170154
```

Pas d’influence

# I.B. Résultats

## Influence du choix des cartes

```
Espérance :  14.4401
Ecart-Type :  2.486566305168632
```

Uniforme

```
Espérance :  14.4394
Ecart-Type :  2.486710204265875
```

“Fixe”

```
Espérance :  14.492999999999999
Ecart-Type :  2.472559604943838
```

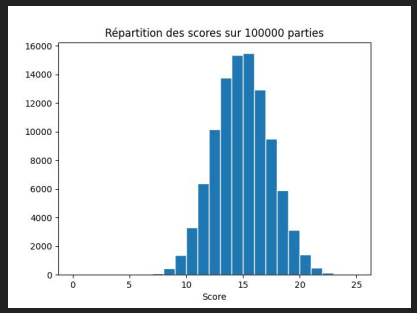
```
Espérance :  14.4488
Ecart-Type :  2.4752330314538105
```

Binomiale

Pas d’influence

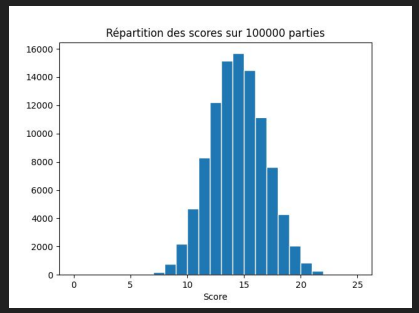
# I.B. Résultats

## Influence du nombre de joueurs



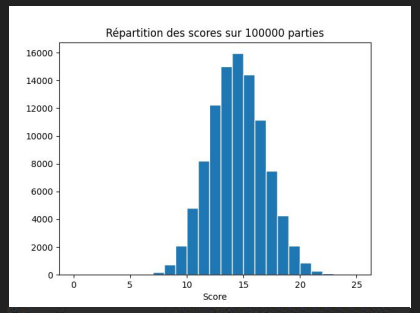

Espérance : 14.458519999999998
Ecart-Type : 2.4834813084861334

2 joueurs



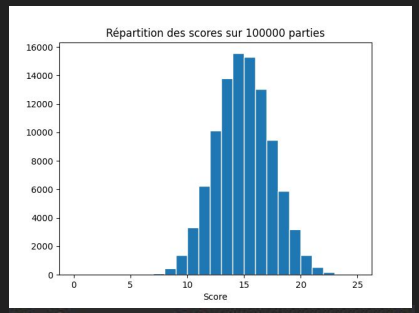

Espérance : 13.944089999999997
Ecart-Type : 2.454490593157792

3 joueurs




Espérance : 13.948730000000001
Ecart-Type : 2.4603457860837237

4 joueurs

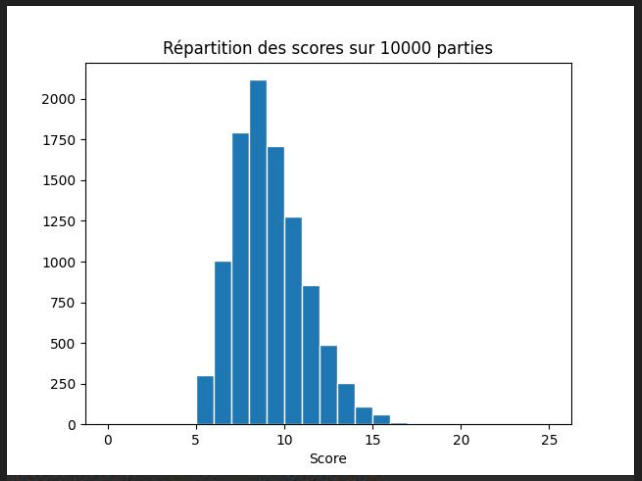



Espérance : 14.465640000000002
Ecart-Type : 2.480959368953862

5 joueurs

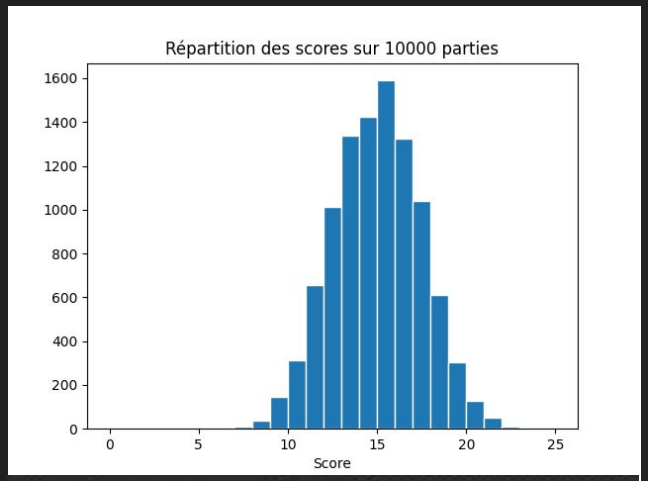
# I.B. Résultats

## Influence des configurations




```
Espérance :  8.6632
Ecart-Type :  2.0426859180990142
Médiane :  8
```

Configuration [1,1,1,1,6]

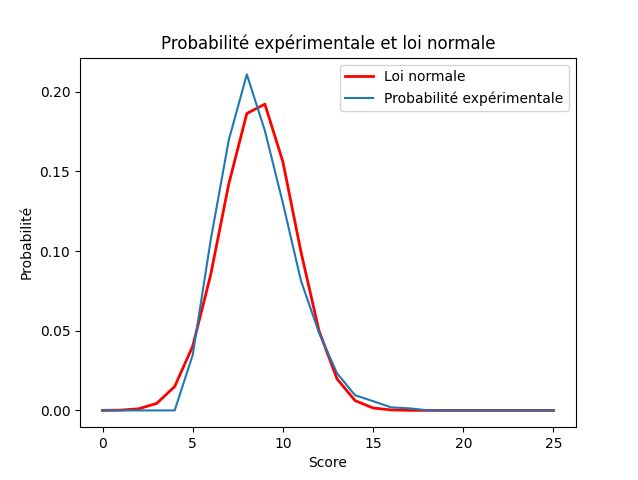



```
Espérance :  14.494400000000002
Ecart-Type :  2.47591773692098
Médiane :  15
```

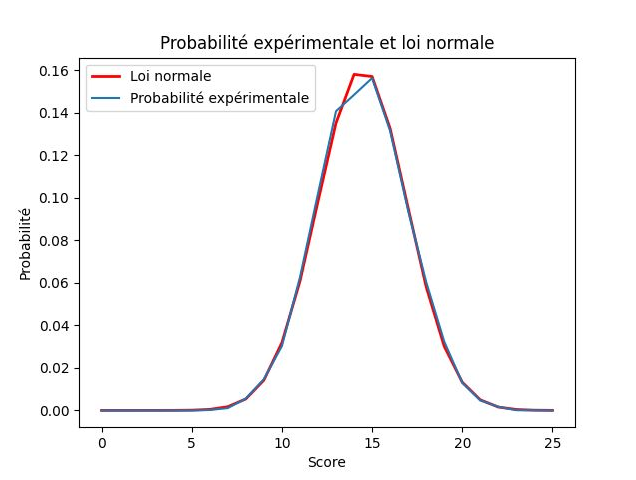
Configuration [3,2,2,2,1]

# I.B. Résultats

Allure en cloche -> Loi normale ?




Configuration [1,1,1,1,6]




Configuration [3,2,2,2,1]

Non adaptée à toutes les configurations

## I.B. Résultats

Hypothèses:

- Chaque joueur joue de manière aléatoire
- Les cartes sont triées uniformément
- Les parties sont indépendantes entre elles (cartes mélangées, indistinguables)

## I.B. Résultats

$\Omega$ = {ensemble des parties} ; $\mathbb{P}$ : probabilité uniforme

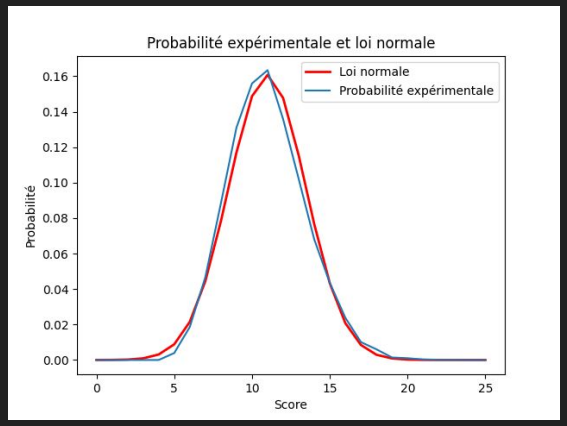
($\Omega$, P($\Omega$), $\mathbb{P}$) espace probabilisé

$X_c$ variable aléatoire du score d'une couleur

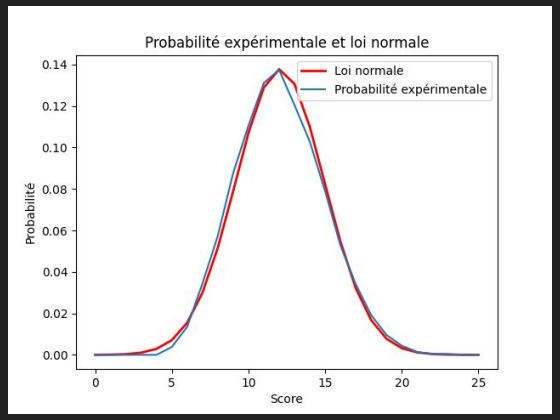
$S = X_w + X_g + X_b + X_r + X_y$ variable du score total

# I.B. Résultats

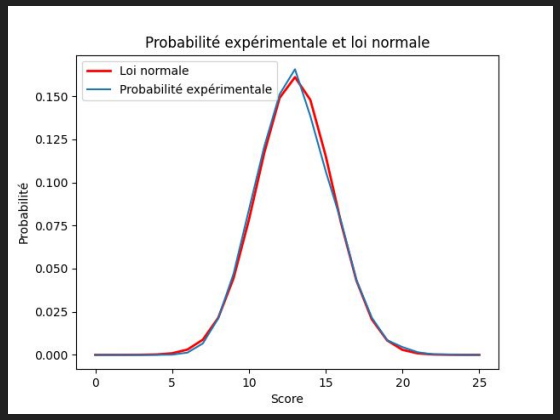
Cependant semble fonctionner pour une espérance >=11




Espérance = 11




Espérance = 12




Espérance = 13

Théorème Central Limite

# I.B. Résultats

## Stockage des différents paramètres pour une configuration

```
Configuration; Espérance; Ecart-type; Médiane; Score minimal; Score maximal, Probabilité de faire moins que 12, Probabilité de faire plus que 15
[1, 1, 1, 1, 6]; 8.6825; 2.0692; 8.0; 5.0; 19.0; 0.9482; 0.0023
[1, 1, 1, 2, 5]; 8.8501; 2.2411; 9.0; 5.0; 20.0; 0.9246; 0.0055
[1, 1, 1, 3, 4]; 8.9711; 2.3442; 9.0; 5.0; 21.0; 0.908; 0.0087
[1, 1, 1, 4, 3]; 9.0172; 2.3838; 9.0; 5.0; 20.0; 0.9013; 0.0103
[1, 1, 1, 5, 2]; 9.0813; 2.3854; 9.0; 5.0; 21.0; 0.8991; 0.0111
[1, 1, 1, 6, 1]; 9.0238; 2.3451; 9.0; 5.0; 20.0; 0.9064; 0.0093
[1, 1, 2, 1, 5]; 9.2944; 2.3733; 9.0; 5.0; 22.0; 0.8923; 0.0136
[1, 1, 2, 2, 4]; 9.5611; 2.5678; 9.0; 5.0; 21.0; 0.8517; 0.0265
[1, 1, 2, 3, 3]; 9.6597; 2.684; 9.0; 5.0; 22.0; 0.8298; 0.0348
[1, 1, 2, 4, 2]; 9.737; 2.7152; 9.0; 5.0; 21.0; 0.8207; 0.0389
[1, 1, 2, 5, 1]; 9.7649; 2.6863; 10.0; 5.0; 21.0; 0.8228; 0.0381
[1, 1, 3, 1, 4]; 9.6238; 2.5252; 9.0; 5.0; 20.0; 0.8537; 0.026
[1, 1, 3, 2, 3]; 9.9743; 2.7561; 10.0; 5.0; 23.0; 0.7987; 0.0494
[1, 1, 3, 3, 2]; 10.0585; 2.8091; 10.0; 5.0; 24.0; 0.7855; 0.056
[1, 1, 3, 4, 1]; 10.1702; 2.8083; 10.0; 5.0; 23.0; 0.7767; 0.0606
[1, 1, 4, 1, 3]; 9.8626; 2.5951; 10.0; 5.0; 21.0; 0.828; 0.0361
[1, 1, 4, 2, 2]; 10.2465; 2.7963; 10.0; 5.0; 22.0; 0.7717; 0.0631
```

# I.B. Résultats

Meilleure configuration ?

Différents critères:

- Meilleure espérance
- Meilleure médiane
- Probabilité d’être au dessus d’un seuil
- Probabilité de ne pas être en dessous d’un seuil

```
[3, 3, 2, 1, 1]; 14.749; 2.0994; 15.0; 7.0; 22.0; 0.1397; 0.5477
```

## II.A. Jetons indice et stratégie

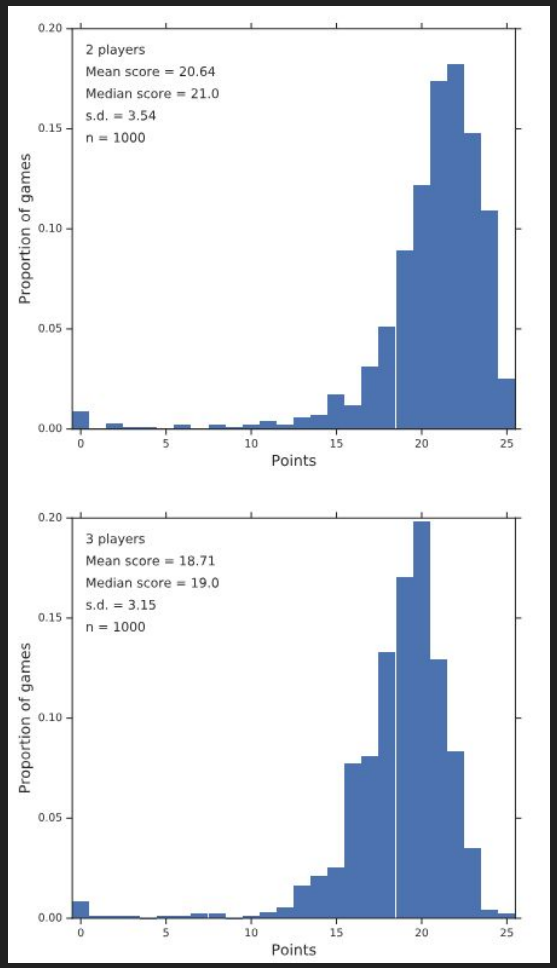
Différentes utilisations:

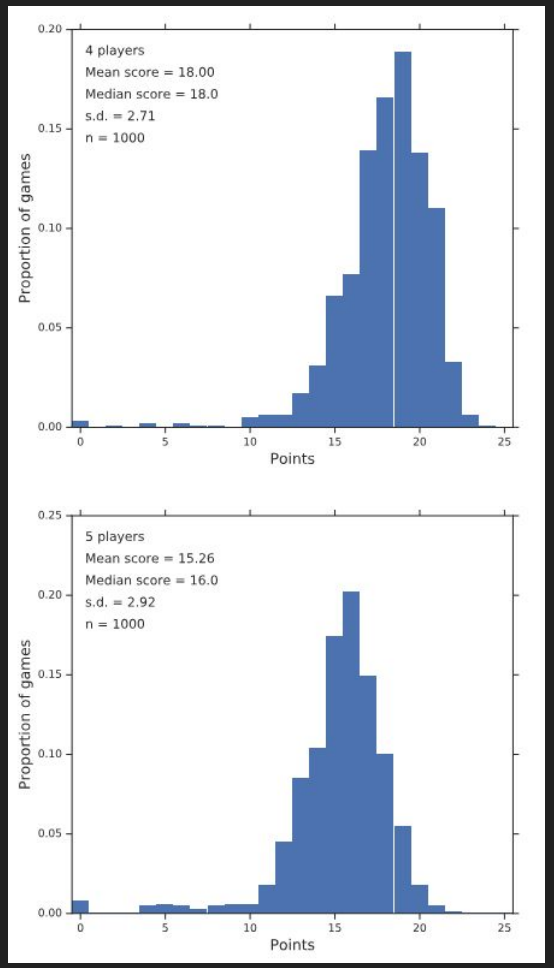
- Donner un maximum d’information sur le jeu
- Indiquer la position d’une carte à jouer
- Indiquer la position d’une carte à ne pas jouer

Le reste repose sur la déduction et de la mémorisation

# II.B. Résultats

## Simulations par utilisation de l'intelligence artificielle



# Conclusion

- Existence d’une meilleure configuration pour cette étude
- Stratégie non applicable, que dire de la configuration ?
- Méthodes plus efficaces, études à faire avec ces méthodes

# Annexe - Code

```python
import numpy.random as rd
import matplotlib.pyplot as plt
from scipy import integrate as intg
import numpy as np


# Code lié à la gestion du jeu
# Générateur des mains des joueurs au début de la partie
def gen_mains(nbr_joueurs, nbr_cartes_main, cartes):
    rd.shuffle(cartes)
    mains = {n_joueur: [(0, "") for _ in range(nbr_cartes_main)] for n_joueur in range(nbr_joueurs)}    # mains est un dictionnaire tel que mains[i][j] donne la j-ième carte de la main du i-ième joueur
    for i in range(nbr_joueurs):
        for j in range(nbr_cartes_main):
            mains[i][j] = cartes.pop()                                          # La distribution revient à remplir entièrement la main de chaque joueur chacun leur tour
    return mains


# Fonction permettant au joueur de placer une carte
def placer_carte(mains, num_joueur, indice_carte, jetons_erreur, cartes):
    v = mains[num_joueur][indice_carte][0]
    c = mains[num_joueur][indice_carte][1]
    if Status[c] == v-1:                                                        # Si la valeur de la carte choisie est plus grande de 1 que la plus grande actuelle pour la même couleur
        Status[c] = v                                                           # Alors on peut poser la carte
    else:
        jetons_erreur += 1                                                      # Sinon on a fait une erreur

    if len(cartes) != 0:                                                        # On tire une carte dans la main si cette dernière n'est pas vide pour remplacer celle posée
        mains[num_joueur][indice_carte] = cartes.pop()
    else:
        del(mains[num_joueur][indice_carte])                                    # Si elle est vide alors on a une carte en moins
    return jetons_erreur


```

# Annexe - Code

```python
# Fonction décidant de la manière dont le joueur choisi une carte parmi celles présentes dans sa main
def choix_de_carte(mains, num_joueur, loi):
    if loi == "fixe":
        return 0
    if loi == "uniforme":
        return rd.randint(0, len(mains[num_joueur]))
    if loi[:9] == "binomiale":
        p = float(Loi[10:])
        x = np.random.binomial(len(mains[num_joueur])-1, p, 1)
        return x[0]


# Générateur du jeu de carte en définissant les proportions des cartes (identiques entre les couleurs)
def gen_cartes(p1=3, p2=2, p3=2, p4=2, p5=1):       # Avec p_i étant le nombre d'occurrences de la valeur i pour une couleur donnée dans le jeu de carte
    assert p1+p2+p3+p4+p5 == 10
    proportion = [p1, p2, p3, p4, p5]
    couleur = ["w", "g", "b", "r", "y"]
    cartes = []

    for c in couleur:
        compteur = 0
        for p in proportion:
            compteur += 1
            for k in range(p):
                cartes.append((compteur, c))
    return cartes
```

# Annexe - Code

```python
# Fonction qui permet de jouer une partie
def partie(nbr_joueurs, nbr_cartes_mains, nbr_jetons_erreur_max, loi, cartes):
    mains = gen_mains(nbr_joueurs, nbr_cartes_mains, cartes)                        # On génère les mains et on initialise jetons, joueurs et tours
    jetons_erreur = 0
    num_joueur = 0
    tour = 0

    while len(mains[nbr_joueurs-1]) != 0 and jetons_erreur < nbr_jetons_erreur_max:         # On joue tant que le dernier joueur a des cartes et que le nombre de jetons erreur est acceptable
        jetons_erreur = placer_carte(mains, num_joueur, choix_de_carte(mains, num_joueur, loi), jetons_erreur, cartes)
        num_joueur = (num_joueur + 1) % nbr_joueurs
        tour += 1

    score = 0                       # On comptabilise le score total
    for i in Status.values():
        score += i

    return score, Status.values()


# ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
# Code lié au calcul statistique
# Calcul d'espérance
def esperance(lst):
    e = 0
    for i in range(len(lst)):
        e = e + i*lst[i]
    return e


# Calcul de variance
def variance(lst):
    v = 0
    for i in range(len(lst)):
        v = v + i*i*lst[i]
    return v - esperance(lst)**2
```

# Annexe - Code

```python
# Calcul d'écart-type
def ecart_type(lst):
    return np.sqrt(variance(lst))


# Cette fonction de médiane fonctionne en utilisant une liste donnant le nombre d'occurrences de chaque score sur N parties
def mediane(lst, n):
    s = 0
    m = 0
    while s < n/2:
        s += lst[m]
        m += 1
    return m-1


# Calcul de la fonction de distribution de la loi normale pour les paramètres mean et std
def loi_normale(mean, std, x):
    return 1/(std * np.sqrt(2 * np.pi)) * np.exp(-(x - mean)**2 / (2 * std**2))


# Calcul de probabilité de l'événement (a <= Score <= b) basé sur une distribution normale (pas d'intégrale car la distribution du score est une fonction en escalier)
def calcul_proba(a, b, esp, e_t):
    p = 0
    for i in range(a, b+1):
        p += loi_normale(esp, e_t, i)
    return p
```

# Annexe - Code

```python
# Calcul de probabilité de l'événement (a <= Score <= b) basé sur une distribution normale pour une configuration donnée
def calcul_proba_config(a, b, configuration: str):
    with open("D:\Prépa\TIPE_2024\Données_score_par_configurations.txt", "r") as data:
        ligne = str(data.readline()).split('; ')
        while ligne[0] != configuration:
            ligne = str(data.readline()).split('; ')

        esp = float(ligne[1])
        e_t = float(ligne[2])

    # def f(x):
        # return loi_normale(esp, e_t, x)
    # return intg.quad(f, a, b)

    return calcul_proba(a, b, esp, e_t)


```

# Annexe - Code

```
144 # ----------------------------------------------------------------------------------------------------------------------------------------------------------------
145 # Code lié à la mesure des scores pour les paramètres donnés
146 # Paramètres
147 N_parties = 10000
148 N_joueurs = 2
149 N_mains = 5
150 N_erreurs = 50
151 Loi = "uniforme"         # Si binomiale, indiquer le paramètre à la fin, ex: "binomiale_0.5"
152
153
154 # Histogramme
155 Cartes_init = gen_cartes(1, 1, 1, 1, 6)
156 X = []                                      # Liste prenant les scores à la fin de chaque partie
157 Occurrences = np.zeros(26)                  # Tableau qui compte le nombre d'occurrences de chaque score
158 OccCouleur = [np.zeros(6) for _ in range(5)]                 # Tableau qui compte le nombre d'occurrences de chaque score pour la couleur choisie au-dessus
159 for _ in range(N_parties):
160     # Status définit l'état de la partie en donnant le score par couleur à chaque tour de la partie
161     Status = {"w": 0,
162               "g": 0,
163               "b": 0,
164               "r": 0,
165               "y": 0}
166
167     Cartes_partie = [i for i in Cartes_init]
168     score, statusValues = partie(N_joueurs, N_mains, N_erreurs, Loi, Cartes_partie)
169     scoreCouleur = np.array([k for k in statusValues])
170
171     X.append(score)
172     Occurrences[score] = Occurrences[score] + 1
173     for i in range(5):
174         OccCouleur[i][scoreCouleur[i]] = OccCouleur[i][scoreCouleur[i]] + 1
175
176
177 Proba = Occurrences / N_parties                             # Tableau des probabilités qu'a un score d'être obtenu à l'issue d'une partie
178 ProbaCouleur = [OccCouleur[i]/N_parties for i in range(5)]  # Liste des probabilités pour chaque couleur
179
180 Esp = esperance(Proba)
181 E_t = ecart_type(Proba)
182 Med = mediane(Occurrences, N_parties)
```

# Annexe - Code

```
184  print("Espérance : ", Esp)
185  print("Ecart-Type : ", E_t)
186  print("Médiane : ", Med)
187  print('')
188
189  '''
190  # Calcul pour prouver la non-indépendance des scores des couleurs
191  idp = 1
192  for i in range(5):
193      idp = idp * ProbaCouleur[i][1]
194  print(idp)
195  '''
196
197  # Graphe de l'histogramme
198  count, bins, ignored = plt.hist(X, range=(0, 25), bins=25, edgecolor="white")
199  X_th = loi_normale(Esp, E_t, bins)
200  # plt.plot(bins, N_parties*X_th, linewidth=2, color="red")
201  plt.xlabel("Score")
202  plt.title("Répartition des scores sur " + str(N_parties) + " parties")
203  plt.show()
204
205  # Graphe des courbes de probabilité
206  plt.plot(bins, X_th, linewidth=2, color="red", label="Loi normale")
207  plt.xlabel("Score")
208  plt.ylabel("Probabilité")
209  plt.title("Probabilité expérimentale et loi normale")
210  plt.plot(bins, Proba, label="Probabilité expérimentale")
211  plt.legend()
212  plt.show()
213
214
```

# Annexe - Code

```
# Code pour tester toutes les configurations et stocker les résultats dans un fichier texte
with open("D:\Prépa\TIPE_2024\Données_score_par_configurations.txt", "w") as donnees:
    donnees.write("Configuration; Espérance; Ecart-type; Médiane; Score minimal; Score maximal, Probabilité de faire moins que 12, Probabilité de faire plus que 15\n")
    for i in range(1, 7):
        for j in range(1, min(7, 10 - i - 2)):
            for k in range(1, min(7, 10 - i - j - 1)):
                for l in range(1, min(7, 10 - i - j - k)):
                    m = 10 - i - j - k - l
                    Cartes_init = gen_cartes(i, j, k, l, m)
                    Es = []
                    Ec = []
                    Me = []
                    Max = []
                    Min = []
                    precision = 10
                    for _ in range(precision):
                        Occurrences = np.zeros(26)  # Tableau qui compte le nombre d'occurrences de chaque score
                        for __ in range(N_parties):
                            # Status définit l'état de la partie en donnant le score par couleur à chaque tour de la partie
                            Status = {"w": 0,
                                      "g": 0,
                                      "b": 0,
                                      "r": 0,
                                      "y": 0}

                            Cartes_partie = [p for p in Cartes_init]
                            score = partie(N_joueurs, N_mains, N_erreurs, Loi, Cartes_partie)[0]
                            Occurrences[score] = Occurrences[score] + 1

                        Proba = Occurrences / N_parties  # Tableau des probabilités qu'a un score d'être obtenu à l'issue d'une partie

                        # Écriture des résultats sous une chaîne de caractère en ne comptant que 4 décimales
                        Es.append(esperance(Proba))
                        Ec.append(ecart_type(Proba))
                        Me.append(mediane(Occurrences, N_parties))
```

# Annexe - Code

```
251                 # Recherche des scores minimaux et maximaux
252                 Score_min = 0
253                 while Occurrences[Score_min] == 0:      # Le premier score d'occurrence non nul est le score minimal
254                     Score_min += 1
255                 Min.append(Score_min)
256
257                 Score_max = 25
258                 while Occurrences[Score_max] == 0:      # Le dernier score d'occurrence non nulle est le score maximal
259                     Score_max -= 1
260                 Max.append(Score_max)
261
262             # Écriture des résultats dans un fichier .txt pour une utilisation future
263             es = sum(Es)/precision
264             ec = sum(Ec)/precision
265             me = str(round(sum(Me)/precision, 4))
266             mx = str(round(sum(Max)/precision, 4))
267             mn = str(round(sum(Min)/precision, 4))
268             config = str([i, j, k, l, m])
269             scmx = str(round(calcul_proba(15, 25, es, ec), 4))
270             scmn = str(round(calcul_proba(5, 12, es, ec), 4))
271             donnees.write('\n' + config + '; ' + str(round(es, 4)) + '; ' + str(round(ec, 4)) + '; ' + me + '; ' + mn + '; ' + mx + '; ' + scmn + '; ' + scmx + '\n')
272
273
```

# Annexe - Code

```
# Influence de N_erreurs
Moyennes = []
for k in range(50):
    N_erreurs = k
    Occurrences = np.zeros(26)
    for i in range(N_parties):
        Status = {"w": 0,
                  "g": 0,
                  "b": 0,
                  "r": 0,
                  "y": 0}

        Cartes_partie = [i for i in Cartes_init]

        score = partie(N_joueurs, N_mains, N_erreurs, Loi, Cartes_partie)[0]
        Occurrences[score] = Occurrences[score] + 1
    Proba = Occurrences / N_parties  # Tableau des probabilités qu'à un score d'être obtenu à l'issue d'une partie
    Esp = esperance(Proba)
    Moyennes.append(Esp)

Erreurs = [i for i in range(50)]

plt.plot(Erreurs, Moyennes)
plt.xlabel("Nombre d'erreurs")
plt.ylabel("Moyenne du score")
plt.title("Evolution de la moyenne de score en fonction du nombre d'erreur maximal autorisé")
plt.show()
```