## *A-Survie du joueur 1*

1. On donne la liste en compréhension suivante :

```
import random as rd
Ldalles = [rd.randint(0, 1) for k in range(18)] #la liste des 18 étapes
```

2. On propose la fonction suivante :

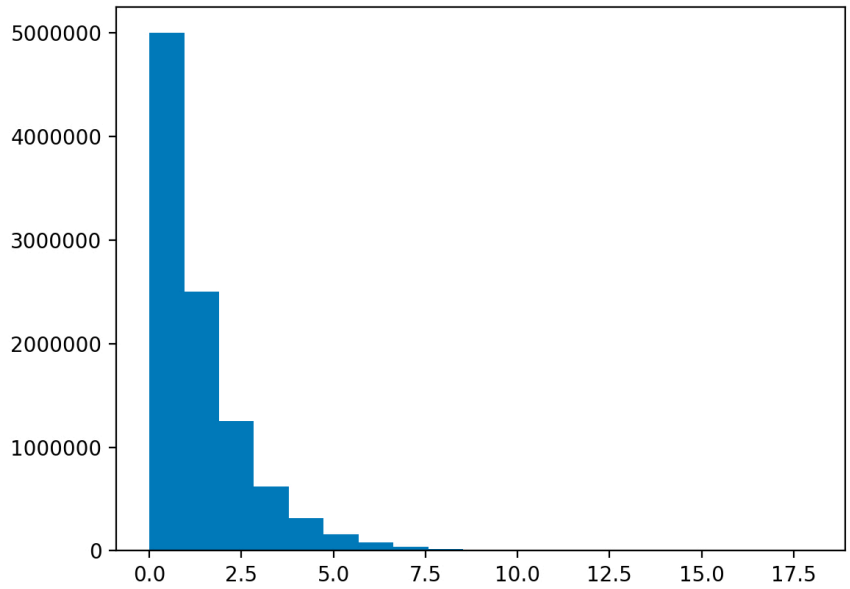
```
def J1():
    """renvoie le numéro de l'étape à laquelle tombe le joueur 1"""
    Ldalles = [rd.randint(0, 1) for k in range(18)] # choix des dalles
    etape = 0 # le numéro de l'étape
    while etape < 18 and rd.randint(0, 1) == Ldalles[etape]:
        # on teste si le pont est déja traversé (lorsque etape = 18)
        # et si le joueur a choisi le bon numéro de dalle pour pouvoir continuer la traversée
        etape = etape + 1
    return(etape)
```

3. On propose la fonction suivante :

```
import matplotlib.pyplot as plt

def HistoJ1(N):
    """histogramme des étapes où s'arrête le joueur 1 au cours de N jeux"""
    L = []
    for i in range(N):
        L.append(J1()) # on sauvegarde les résultats des N jeux dans la liste L
    plt.hist(L, range = (0, 18), bins = 19) # tracé de l'histogramme
    plt.show()
```

En la testant avec $N = 10^7$, voici l'histogramme que l'on obtient :




Il semblerait que le nombre de fois où le joueur 1 s'arrête à l'étape $i+1$ est la moitié de ce nombre à l'étape $i$, ce que nous allons justifier par la suite.

4. (a) On crée notre dictionnaire de la façon suivante :

```
def DicoJ1(N):
    """renvoie le dictionnaire donnant les étapes où s'arrête le joueur 1 lors de N jeux"""
    d = {} # dictionnaire vide
    for i in range(N):
        a = J1()
        if a in d: # si la clé est déjà présente
            d[a] = d[a] + 1 # on augmente le nombre de 1
        else:
            d[a] = 1 # sinon on crée la clé avec la valeur correspondante égale à 1
    return(d)
```

(b) Voici le dictionnaire que l'on obtient en répétant $10^7$ fois le jeu :

```
>>> DicoJ1(10000000)
{3: 624158, 0: 5000256, 1: 2499205, 4: 312458, 5: 156112, 2: 1252254, 7: 38814, 6: 78003, 8: 19337, 9:
    9656, 10: 4872, 11: 2415, 12: 1305, 14: 275, 13: 580, 18: 29, 16: 89, 15: 148, 17: 34}
```

On remarque que le joueur 1 s'est sauvé 29 fois. C'est conforme à ce que l'on peut intuiter car à chaque jeu le joueur 1 a une chance sur $2^{18}$ de se sauver. Ainsi en répétant 10 millions de fois le jeu, il se sauve en moyenne $\dfrac{10^7}{2^{18}}$ fois, ce qui vaut environ 38. On a bien un ordre de grandeur similaire.

*On remarque que les clés de ce dictionnaire ne sont pas ordonnées, c'est toujours le cas comme nous l'avons vu en cours.*

(c) On reprend le dictionnaire précédent obtenu avec $N = 10^7$ et l'on procède ainsi :

```
>>> d={3: 624158, 0: 5000256, 1: 2499205, 4: 312458, 5: 156112, 2: 1252254, 7: 38814, 6: 78003, 8: 19337,
    9: 9656, 10: 4872, 11: 2415, 12: 1305, 14: 275, 13: 580, 18: 29, 16: 89, 15: 148, 17: 34}

>>> {i : (d[i], d[i] / (10 ** 7)) for i in d}
{3: (624158, 0.0624158), 0: (5000256, 0.5000256), 1: (2499205, 0.2499205), 4: (312458, 0.0312458), 5:
    (156112, 0.0156112), 2: (1252254, 0.1252254), 7: (38814, 0.0038814), 6: (78003, 0.0078003), 8: (19337,
     0.0019337), 9: (9656, 0.0009656), 10: (4872, 0.0004872), 11: (2415, 0.0002415), 12: (1305,
    0.0001305), 14: (275, 2.75e-05), 13: (580, 5.8e-05), 18: (29, 2.9e-06), 16: (89, 8.9e-06), 15: (148,
    1.48e-05), 17: (34, 3.4e-06)}
```

(d) Il n'est pas indispensable d'utiliser un dictionnaire puisque les clés sont des nombres entiers entre 0 et 18. On aurait aussi pu sauvegarder les résultats dans une liste dont les indices seraient numérotés de 0 à 18.

(e) Au vu des résultats donnés dans le dictionnaire précédent, il semblerait que la probabilité de franchir avec succès l'étape 0 est $\dfrac{1}{2}$, la probabilité de franchir avec succès l'étape 1 est $\dfrac{1}{4}$ et ainsi de suite les probabilités étant divisées par 2 à chaque étape. En effet, si le joueur 1 franchit avec succès l'étape $i \in [\![0, 17]\!]$ alors, il a une chance sur deux de réussir également l'étape $i+1$, c'est pour cela que les probabilités sont divisées par 2 à chaque étape.

## *B-Survie des 16 joueurs*

1. On propose la fonction suivante :

```
import random as rd

def Jeu():
    ndc = 0
    # ndc représente le nombre d'étapes connues par les joueurs
    # soit parce qu'une dalle est brisée, soit parce qu'un joueur précédent a marché sur une dalle solide à cette étape-là
    d = [rd.randint(0, 1) for k in range(18)] # le choix des dalles
    joueur = 1 # le numéro du joueur qui s'élance
    while ndc < 18: # tant que toutes les dalles n'ont pas été découvertes
        test = rd.randint(0, 1) # le joueur choisit sur quelle dalle il va marcher
        while test == d[ndc]: # tant que le joueur trouve la bonne dalle, il continue son parcours
            test = rd.randint(0,1)
            ndc = ndc + 1 # une dalle de plus est découverte
            if ndc == 18: # si les 18 dalles sont découvertes, le joueur est sauvé
                return(joueur)
        ndc = ndc + 1 # si le joueur se trompe, cela permet de découvrir une étape
        joueur = joueur + 1 # joueur suivant
    return(joueur)
```

2. (a) Voici la fonction qui renvoie la moyenne :

```
def Moy(N):
    a = 0
    for i in range(N): # on répète N fois le jeu
        a = a + Jeu() # on somme les résultats obtenus et on fait la moyenne
    return(a / N)
```

(b) En testant la fonction avec $N = 10^6$, on obtient une moyenne proche de 10.

```
>>> Moy(1000000)
9.997595
```

(c) En moyenne un joueur découvre 2 dalles pour les joueurs suivants. En effet, on a la probabilité pour un joueur de découvrir exactement :

- 1 dalle : $\frac{1}{2}$ (il faut échouer à la dalle 1)
- 2 dalles : $\frac{1}{4}$ (il faut avoir franchit avec succès la dalle 1)
- 3 dalles : $\frac{1}{8}$ (il faut avoir franchit avec succès les dalles 1 et 2)

ainsi de suite jusqu'à :

- 18 dalles : $\frac{1}{2^{18}}$

Si l'on calcule l'espérance correspondante, c'est-à-dire le nombre moyen de dalles découvertes, nous obtenons donc :

$$\sum_{k=1}^{18} \frac{k}{2^k} \approx 2$$

Cette somme pouvant se calculer en Python. Si en moyenne, un joueur découvre 2 dalles alors les 9 premiers joueurs découvrent en moyenne les 18 dalles et le joueur 10 sera sauvé.

*Ce raisonnement manque de rigueur, nous aurons les outils pour préciser cela en fin d'année.*

3. (a) On propose la fonction suivante :

```
def Resultats(N):
    L = [0 for i in range(16)] # le nombre de succès pour chaque joueur
    for i in range(N): # on répète le jeu N fois
        a = Jeu()
        for j in range(a - 1, 16): # les joueurs à partir du a (d'indice a-1) sont sauvés
            L[j] += 1
    return(L)
```

(b) On obtient les valeurs suivantes :

```
>>> Resultats(1000000)
[2, 68, 636, 3732, 15279, 47915, 118768, 239775, 407010, 592666, 759560, 880767, 951556, 984504, 996225,
 999333]
```

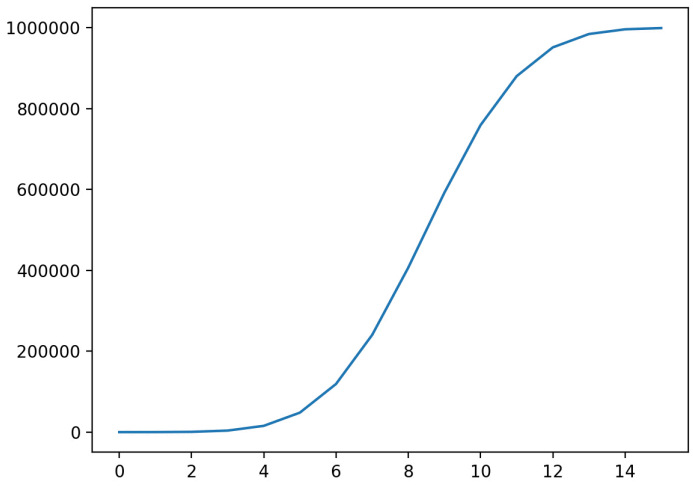
Conformément à l'intuition, le joueur 1 est sauvé très rarement et le joueur 16 est sauvé très souvent. Les valeurs obtenues sont croissantes.

(c) On utilise la fonction précédente pour faire le tracé :

```
import matplotlib.pyplot as plt

X = [k for k in range(16)] # les 16 joueurs
Y = Resultats(10 ** 6)
plt.plot(X, Y)
plt.show()
```

Voici le graphe obtenu :



4. (a) Pour avoir une probabilité, il suffit de diviser le résultat par le nombre de fois où l'on répète l'expérience :

```
def Resultats2(N):
    L = [0 for i in range(16)] # le nombre de succès pour chaque joueur
    for i in range(N): # on répète le jeu N fois
        a = Jeu()
        for j in range(a - 1, 16): # les joueurs à partir du a (d'indice a-1) sont sauvés
            L[j] += 1
    return([i / N for i in L]) # on divise toutes les valeurs par N pour avoir une probabilité
```

(b) Voici ce que l'on obtient avec $N = 10^6$ :

```
>>> Resultats2(10 ** 6)
[2e-06, 7.3e-05, 0.00063, 0.003668, 0.015077, 0.047644, 0.118493, 0.239821, 0.407582, 0.592829, 0.759419,
 0.880991, 0.951663, 0.984436, 0.996331, 0.999363]
```

Il semblerait que les joueurs 13, 14, 15 et 16 aient plus de 95% de réussite.

## *C-Généralisation*

1. On garde le principe des fonctions précédentes et on modifie uniquement les valeurs numériques 16 et 18 en $NJ$ et $ND$.

```
import random as rd

def JeuG(NJ, ND):
    ndc = 0
    # ndc représente le nombre d'étapes connues par les joueurs
    # soit parce qu'une dalle est brisée, soit parce qu'un joueur précédent a marché sur une dalle solide
     à cette étape-là
    d = [rd.randint(0, 1) for k in range(ND)] # le choix des dalles
    joueur = 1 # le numéro du joueur qui s'élance
    while ndc < ND: # tant que toutes les dalles n'ont pas été découvertes
        test = rd.randint(0, 1) # le joueur choisit sur quelle dalle il va marcher
        while test == d[ndc]: # tant que le joueur trouve la bonne dalle, il continue son parcours
            test = rd.randint(0,1)
            ndc = ndc + 1 # une dalle de plus est découverte
            if ndc == ND: # si les ND dalles sont découvertes, le joueur est sauvé
                return(joueur)
        ndc = ndc + 1 # si le joueur se trompe, cela permet de découvrir une étape
        joueur = joueur + 1 # joueur suivant
    return(joueur)


def Probas(N, NJ, ND):
    L = [0 for i in range(NJ)] # le nombre de succès pour chaque joueur
    for i in range(N): # on répète le jeu N fois
        a = JeuG(NJ, ND)
        for j in range(a - 1, NJ): # les joueurs à partir du a (d'indice a-1) sont sauvés
            L[j] += 1
    return([i / N for i in L]) # on divise toutes les valeurs par N pour avoir une probabilité
```

2. (a) Voici les résultats obtenus présentés sous la forme d'une liste.

```
>>> [Probas(10 ** 6, NJ, NJ) for NJ in range(1, 6)]
[[0.499516], [0.249665, 0.749638], [0.125396, 0.50085, 0.875083], [0.062888, 0.313442, 0.688021,
    0.937711], [0.031092, 0.18674, 0.499735, 0.811776, 0.969024]]
```

(b) On multiplie les valeurs obtenus par $2^{NJ}$ pour faire apparaitre les numérateurs recherchés. En reprenant la liste précédente :

```
>>> L = [[0.499516], [0.249665, 0.749638], [0.125396, 0.50085, 0.875083], [0.062888, 0.313442, 0.688021,
    0.937711], [0.031092, 0.18674, 0.499735, 0.811776, 0.969024]]

>>> [[L[i][k] * (2 ** (i + 1)) for k in range(i + 1)] for i in range(5)]
[[0.999032], [0.99866, 2.998552], [1.003168, 4.0068, 7.000664], [1.006208, 5.015072, 11.008336,
    15.003376], [0.994944, 5.97568, 15.99152, 25.976832, 31.008768]]
```

On obtient le tableau proposé à la question suivante en ajoutant des joueurs fictifs qui sont sauvés :

| | Joueur 0 | Joueur 1 | Joueur 2 | Joueur 3 | Joueur 4 | Joueur 5 | Joueur 6 |
|---|---|---|---|---|---|---|---|
| 1 dalle | 0 | 1/2 | 1 | 1 | 1 | 1 | 1 |
| 2 dalles | 0 | 1/4 | 3/4 | 1 | 1 | 1 | 1 |
| 3 dalles | 0 | 1/8 | 4/8 | 7/8 | 1 | 1 | 1 |
| 4 dalles | 0 | 1/16 | 5/16 | 11/16 | 15/16 | 1 | 1 |
| 5 dalles | 0 | 1/32 | 6/32 | 16/32 | 25/32 | 31/32 | 1 |

(c) On remarque que ce tableau est construit par un processus similaire à celui du triangle de Pascal. Chaque nombre est la moyenne de celui écrit au dessus avec celui écrit "au dessus à gauche". Par exemple :

$$\frac{11}{16} = \frac{\frac{4}{8} + \frac{7}{8}}{2}$$

*On peut trouver cela intuitif de faire cette moyenne mais la preuve rigoureuse n'est pas évidente.*

(d) D'après la remarque précédente sur la construction du tableau, on a le numérateur de la fraction qui correspond à la probabilité de survie du joueur $j$ lors d'une épreuve avec $i$ dalles qui vaut :

$$\sum_{k=0}^{j-1} \binom{i}{k}$$

La réponse demandée dans la question est donc :

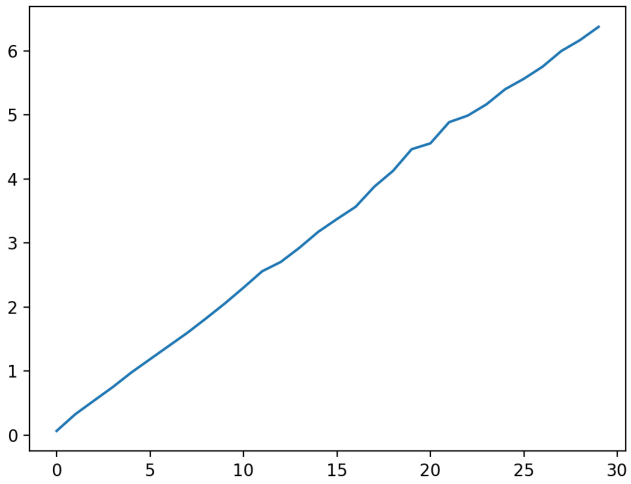
$$\sum_{k=0}^{455} \binom{1000}{k}$$

3.

```
import time as t
import matplotlib.pyplot as plt

def comp(N, Nmax): # Nmax est le nombre maximal de joueurs et de dalles dans le graphe considéré
    X = []
    for NJ in range(Nmax):
        a=t.time()
        Probas(N, NJ, NJ)
        X.append(t.time()-a) # on mesure le temps d'éxécution et on l'ajoute à notre liste
    plt.plot([NJ for NJ in range(Nmax)], X)
    plt.show() # on effectue le tracé
```

On teste ce programme jusqu'à $Nmax = 30$ pour obtenir le graphe suivant :




On remarque que la courbe est à peu près une droite, ce qui nous permet de conjecturer que la complexité est linéaire. Ceci n'est bien sûr pas une preuve, il faudrait analyser de plus près nos algorithmes pour le démontrer.