Jeu de roulette

Simulation et probabilités

Vallet Gaëlle (IMAC2)

Cours de Mathématiques-Probabilités

Lancer l'application :

L'application est codée en langage python3 à l'aide de la librairie pyQt (pour la création de l'interface graphique), cette librairie doit être installé pour compiler et lancer le programme.
La fonction de tirage aléatoire sur [0,1] est la fonction `random()` (ou `random.random()`) de la bibliothèque random.

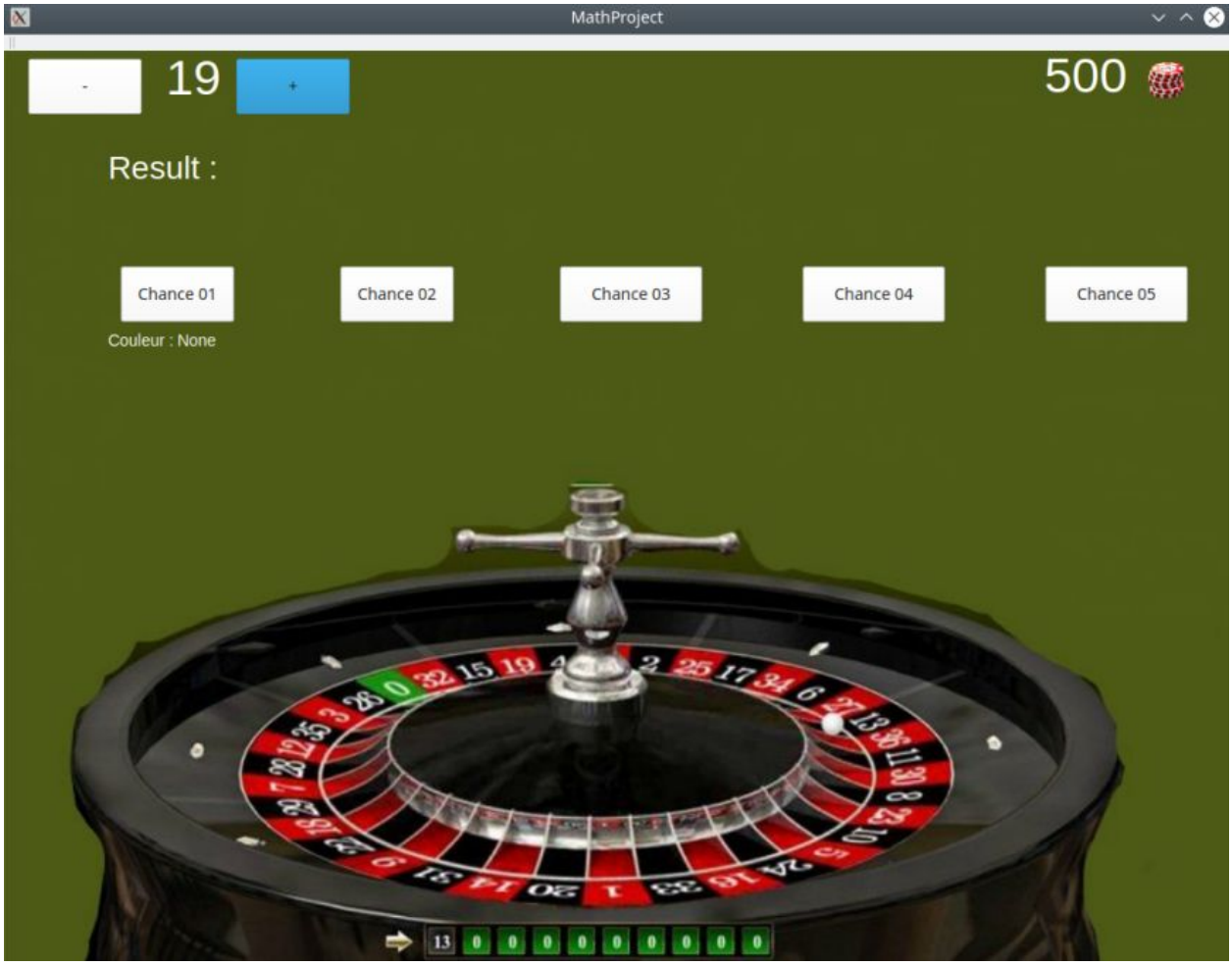
Le code du programme est disponible sur le dépôt git : https://github.com/gaellou/Maths_Casino

Sur machine linux :
Installer la librairie pyQt (?)
Ouvrir le dossier où se trouve le code dans un terminal et exécuter la ligne de commande :

```
python3 Main.py
```

Le fichier Main.py contient le fonctionnement du jeu et la partie affichage.
Le fichier Proba.py contient les fonctions de probabilité et de tirage.

Le jeu :



Ce jeu est un jeu de roulette.
En début de partie le joueur dispose de 500 jetons. Le joueur mise des jetons (10) et tente sa chance. Si la valeur tirée est celle sur laquelle il a parié, il remporte des jetons, sinon il perd sa mise.
Le joueur dispose de 5 mode de tirages aléatoires. La roue comporte 35 cases (de 0 à 34). Les cases paires sont rouges, les cases impaires sont noires et la case 0 est verte.

“**Chance 01**” correspond à un tirage selon une loi uniforme. La probabilité pour le joueur de gagner est de 1/35 à chaque tirage.
On obtient un tirage avec le code suivant :

```
int(random() * 34)
```

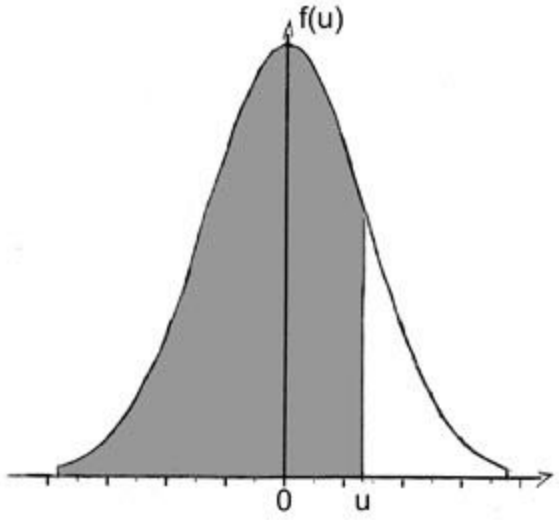
On génère un nombre aléatoire entre 0 et 1, on passe dans l’intervalle [0,34] par règle de propotionnalité, et on ne garde que sa partie entière (`int()`) pour obtenir un entier.

Un succès avec ce mode de tirage rapporte 150 jetons au joueur.

“**Chance 02**” correspond à un tirage selon une loi normale (ou de Gauss)

La loi normale dépend de 2 paramètres caractéristiques: sa moyenne m et son écart type σ . La moyenne correspond au milieu de la courbe en cloche, et l'écart-type mesure sa dispersion autour de cette moyenne. La courbe sera donc plus ou moins “aplatie” et plus ou moins étendue en largeur, mais la surface sous la courbe sera toujours égale à 1. La courbe ne donne pas une probabilité, mais une “densité de probabilité” et les probabilités seront lues en surface en dessous de la courbe.

La “loi normale centrée réduite” est une loi normale particulière de moyenne m=0 et d'écart-type $\sigma = 1$.
Il faut calculer la surface qui se trouve à gauche de la valeur u donnée, et on aura trouvé la probabilité pour que la variable aléatoire se trouve inférieure à u.



Cette surface a pour formule mathématique:

$$Prob\left(<u\right) = \int_{-\infty}^{u} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2} du$$

Pour calculer cette surface, j'ai utilisé la méthode des rectangles (pas la plus efficace mais suffisante dans notre cas) : la surface est décomposée en "tranches" verticales (2000 tranches par écart-type). Pour chaque tranche, on peut en calculer la hauteur puisqu'on a l'équation de la courbe, et on peut donc en calculer la surface puisqu'on a la largeur. On fera la somme de tous ces petits rectangles et on aura la surface recherchée.

Dans notre cas nous voulons faire l'inverse, trouver la valeure u correspondant à la probabilité p.
On va aussi utiliser des petites tranches verticales, à part qu'on ne sait pas d'avance combien il y en aura. En fait, on va calculer par approche successive.

- on commence par calculer la surface par petites tranches du=1/2000 (=2000 tranches par écart-type).

- à un moment, la surface dépasse la valeur cherchée, et donc on revient en arrière avec des tranches plus petites (moitié).

- ceci jusqu'à ce qu'on soit assez près de la surface donnée, et on a trouvé le "u" cherché.

Ce calcule correspond à la fonction xgaussred(p).
Pour effectuer un tirage aléatoire, on tire au hasard un nombre compris entre 0 et 1 avec random() du module random, et on considère cette valeur comme une probabilité gaussienne dont on veut avoir le u correspondant. On va donc naturellement utiliser la fonction précédente.
(cf fonction `hgauss(m,s,n)`)
Dans notre cas : m=19, s=5 et n=1.

En cas de succès, le joueur remporte 75 jetons.

“**Chance 03**” effectue un tirage selon une loi de poisson.
La loi de Poisson est une loi de probabilité associée à une variable aléatoire discrète.
La probabilité est donnée par la formule suivante:

$$P(k) = e^{-m} \times \frac{m^k}{k!}$$

Avec:

- e = 2.718281828459045
- m = paramètre caractéristique: c'est en même temps la moyenne et la variance de la loi.

On calcule la probabilité de la loi en appliquant directement la formule. J’ai fait le choix de ne pas utiliser d’algorithme pour la factoriel et de ne pas utiliser le calcule de puissance pour éviter les dépassement de capacité et ai préféré un calcul itératif avec une boucle sur k. On obtient la fonction suivante :

```
def poisson(k,m):
       p=e**(-m)
       for i in xrange(0,k):
                  p*=m/k
                  k-=1
       return p
```

Avec la fonction `hpoisson(m, nb=0)`, on génère un tirage aléatoire selon le principe suivant :

- on tire au hasard un nombre décimal compris entre 0 et 1 avec random(), et on le considère comme une probabilité cumulée.
- grâce au calcul de la probabilité cumulée de la loi de Poisson, on cherche à quel variable aléatoire k cela correspond

En cas de succès, le joueur remporte 100 jetons.

“**Chance 04**” correspond à un tirage selon une loi binomiale.
La formule donnant la probabilité cherchée est la suivante:

$$p\left(k\right) = C_n^k p^k \left(1\text{-}p\right)^{n-k}$$

Avec C_n^k = combinaison de n objets pris k à k.
Nous devons déjà est en mesure de calculer le coefficient binomial. J’ai utilisé un algorithme déjà étudier en cours de programmation qui évite le dépassement capacité lorsque k et n sont grands. On utilise les 2 boucles while pour que la valeur calculée à chaque boucle for ne dépasse jamais les capacités de calcul en nombres réels, et ceci, quelque soit le k et le n donné.
Nous avons alors la fonction `binom(k,n,p)` qui nous donne la probabilité d'avoir k réussites dans n événements indépendants, chaque évènement ayant une probabilité p de réussite.
On effectue la tirage aléatoire avec la fonction `hbinom(n,p,nb=0)`.

En cas de succès, le joueur remporte 120 jetons.

“**Chance 05**” correspond à un schéma de Bernoulli. Le joueur doit obtenir deux fois de suite la même couleur. S’il obtient 0, cela correspond à un succès.
On a donc une probabilité de 18/35 de succès à chaque tirage.
On la le code suivant :

```
p = 18/35
c = 0
piece = self.Pieces.text()
piece = int(piece) - 10
for i in range (0,1) :
    t=random()
    if t<p :
        c=c+1
if c == 2 :
    self.Error.setText("Gagné ! (+200)")
    pieces = int(piece)+200
else :
    self.Error.setText("Perdu... (-10)")
self.Pieces.setText(str(piece))
```

Le tirrage est réalisé dans la fonction `bernoulli(p)`
La variable c compte le nombre de succès. On réalise des deux tirages et on vérifie si on obtient deux succès.
Si le joueur obtient un succès aux deux tirages, il remporte 200 jetons.