Algo & Prog, avec Python
(L1-Sciences)
TP n° 11, Automne 2016

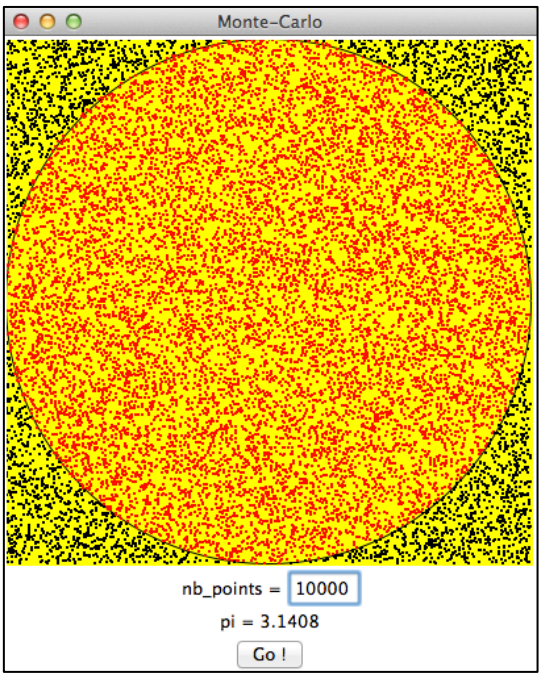
Une méthode de Monte-Carlo illustrée par une interface graphique

Exercice 11.1 Nous nous proposons d'effectuer une **expérience de Monte-Carlo**. Elle consiste à *calculer une valeur numérique en utilisant des procédés aléatoires, c'est-à-dire des techniques probabilistes* (Wikipedia[1]).

a) On considère un cercle de rayon `R = 200` tangent à un carré. Quelle est la probabilité `p` qu'un point tiré au hasard dans le carré tombe à l'intérieur du cercle ? Faites un calcul purement géométrique.

b) Une autre manière de calculer cette probabilité consiste à tirer au hasard `N = 5000` points dans le carré et à compter le nombre `S` (comme *succès*) de points tombés dans le cercle. En déduire une formule permettant de calculer une valeur approchée de π.

c) Vous allez développer une interface graphique comme ci-dessous, avec un canvas jaune 400 x 400, une zone éditable (*Entry*) contenant le nombre N de points, et un bouton *Go!* pour lancer les N points dans le canvas, en les affichant en noir s'ils tombent à l'exterieur du cercle et en rouge s'ils tombent dans le cercle. Un compteur vous permettra à la fin de savoir combien sont tombés dans le cercle. Vous pourrez alors afficher en bas (dans un *Label*) le résultat de cette expérience de Monte-Carlo, c'est-à-dire la valeur approchée de π. A vous !



[1] `http://fr.wikipedia.org/wiki/Methode_de_Monte-Carlo`

Algo & Prog, avec Python
(L1-Sciences)
TD n° 11, Automne 2016

Utilisation de tkinter pour construire une ANIMATION

• Nous souhaitons programmer la simulation d'une balle lachée à la verticale dans un champ de pesanteur. Nous optons donc pour un canvas tubulaire jaune de largeur `100` et de hauteur `400`. La position initiale de la balle sera `x = 50` et `y = 100`. Immergée dans un champ de gravitation dirigé vers le bas, la vitesse verticale `dy` de la balle va changer, sa vitesse horizontale étant constante (ici nulle). La vitesse initiale `dy` est nulle. La balle est un disque rouge de rayon `r = 20`.

Exercice 11.1 Programmez déjà toutes ces données. Pour empêcher l'utilisateur de modifier la taille de la fenêtre, vous chercherez comment utiliser la méthode `resizable` sur la fenêtre `root`.

• Le principe d'une telle animation est le suivant. On utilise une **horloge** qui déclenchera une action par exemple toutes les 10 ms. A chaque top d'horloge la balle va donc se déplacer. Deux tops d'horloge étant séparés de 10 ms, le mouvement sera suffisamment fluide.

Exercice 11.2 Nous programmons donc en réalité un dessin animé. A combien d'images/sec ?

• Sans entrer dans les détails, il existe une méthode `after` qui demande à un canvas d'exécuter une fonction d'arité 0 toutes les x millisecondes, par exemple :

```
canvas.after(10,animation)
```

où `animation()` sera notre fonction chargée de calculer la position suivante de la balle puis de la dessiner. Vous notez qu'elle n'a pas de paramètres, et doit modifier les variables `x`,`y` etc. Donc il faudra utiliser en début de cette fonction une déclaration `global` pour prévenir Python que les variables susceptibles d'être modifiées sont globales !
Il ne s'agit pas de faire de la physique et de résoudre des équations différentielles[2]. Notre perception infinitésimale du vecteur vitesse consiste à dire que si la balle est en (x,y) avec une vitesse (dx,dy) à un certain top d'horloge, sa position au top d'horloge suivant - 10 ms plus tard - sera (x+dx,y+dy). On avance donc d'un petit vecteur vitesse. Mais trois problèmes surgissent.

a) la vitesse dx ne joue pas, mais la composante dy est augmentée d'une constante gravitationnelle `GRAV` à chaque top d'horloge. Ou diminuée si elle est en train de remonter après rebond au sol. L'accélération de la pesanteur est constante. Nous prendrons `GRAV = 0.5` puisque nous sommes nommés Maîtres des Unités.

[2] En réalité c'est bien ce que l'on fait mais de manière numérique, très précisément en utilisant sans le savoir la méthode d'Euler.

b) Il faudra tenir compte du rebond au sol, donc se poser la question : la balle est-elle en train de s'enfoncer dans le sol, auquel cas il faudra l'empêcher et inverser son vecteur vitesse. Il faudra aussi tenir compte du frottement au sol et de la perte d'énergie cinétique. Attention...

c) Si tout est bien programmé le mouvement avec rebond est perpétuel puisque l'énergie est conservée. Elle ne le sera pas s'il y a frottement au sol avec production de chaleur. Nous introduisons un coefficient multiplicatif FROT dans [0,1] qui fera baisser la vitesse à chaque rebond. Si FROT = 1 le choc est *élastique* (mouvement perpétuel), et si FROT = 0, la balle reste collée au sol (choc *mou*) comme un chewing-gum. Vous testerez avec FROT = 0.9.

d) Enfin, même avec frottement il faudra décider que la balle est à l'arrêt même si ce n'est pas vrai et que sa vitesse au sol est < 2 par exemple, sinon le mouvement serait imperceptiblement perpétuel...

Exercice 11.3 Voilà, il n'y a plus qu'à relever le défi, donc essentiellement à programmer en une douzaine de lignes la fonction `animation` !

N.B. Nous n'aurons pas le temps de le faire, mais il serait intéressant de reprogrammer cette simulation dans un style plus objet, en définissant une classe Balle...