TP 11

Dictionnaires

1 Snakes and ladders : le jeu

Extrait du travail de T. Kovaltchouk - UPSTI

1.1 Présentation du jeu

Le jeu *serpents et échelles* est un jeu de société où on espère monter les échelles en évitant de trébucher sur les serpents. Il provient d'Inde et est utilisé pour illustrer l'influence des vices et des vertus sur une vie.

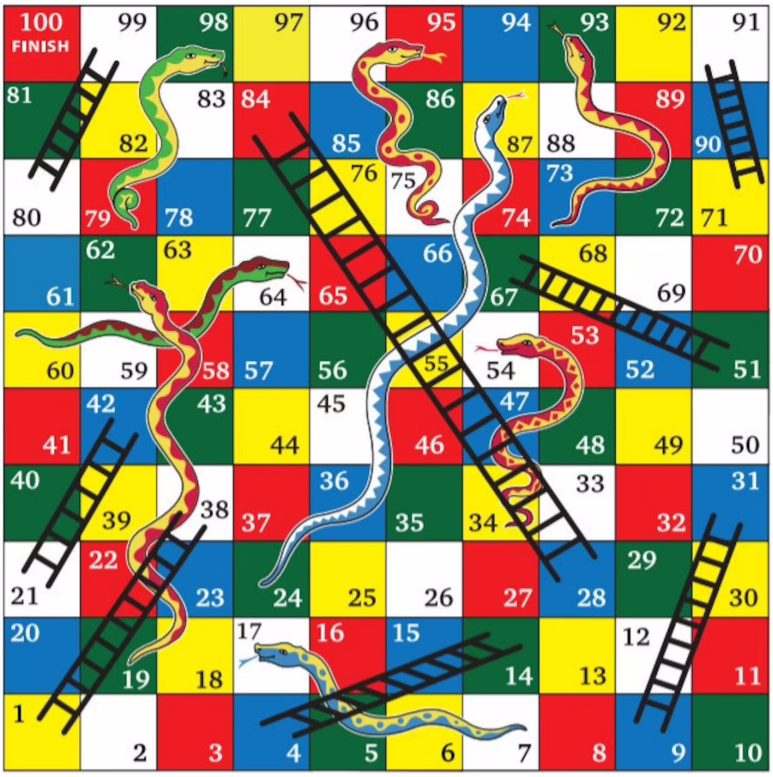


FIGURE 1 – Exemple d'un plateau de serpents et échelles

Le plateau

- Le plateau comporte 100 cases numérotées de 1 à 100 en boustrophédon [1] : le 1 est en bas à gauche et le 100 est en haut à gauche ;
- des serpents et échelles sont présents sur le plateau : les serpents font descendre un joueur de sa tête à sa queue, les échelles font monter un joueur du bas de l'échelle vers le haut.

1. à la manière du bœuf traçant des sillons, avec alternance gauche-droite et droite-gauche

Déroulement

- Chaque joueur a un pion sur le plateau. Plusieurs pions peuvent être sur une même case. Les joueurs lancent un dé à tour de rôle et ils avancent du nombre de cases marqués sur le dé. S'ils atterrissent sur un bas d'échelle ou une tête de serpent, ils vont directement à l'autre bout ;
- les joueurs commencent sur une case 0 hors du plateau : la première case où mettre leur pion correspond donc au premier lancer de dé ;
- le premier joueur à arriver sur la case 100 a gagné ;
- il existe 3 variantes quand la somme de la case actuelle et du dé dépasse 100 :
  - le rebond : on recule d'autant de cases qu'on dépasse ;
  - l'immobilisme : on n'avance pas du tout si on dépasse :
  - la fin rapide : on va à la case 100 quoi qu'il arrive.

On utilisera les notations suivantes pour les complexités : N_{cases}, le nombre de cases du plateau (100), et N_{SeE} la somme du nombre de serpents et du nombre d'échelle (16 dans notre exemple).

1.2 Simulation du jeu

Question 1 *Écrire une fonction* `lancerDe() -> int` *qui renvoie un nombre entier compris entre 1 et 6 en utilisant une fonction du module* `random`. *Vous pourrez vous aider des documentations en annexe.*

Les serpents et les échelles sont représentés par un dictionnaire `dSeE` tel que, pour une case de départ numérotée `i`, `dSeE[i]` donne le numéro de la case d'arrivée.

Avec l'exemple de la figure 1, on a :

```
dSeE = { 1: 38, 4: 14, 9: 31, 17: 7, 21: 42, 28: 84, 51: 67, 54: 34,
         62: 19, 64: 60, 71: 91, 80: 99, 87: 24, 93: 73, 95: 75, 98: 79}
```

Question 2 *Écrire la fonction* `caseFuture(case: int) -> int` *qui prend en argument le numéro de la case et qui renvoie le numéro de la case où va se trouver le joueur en atterrissant sur la case numérotée* `case`. *Par exemple,* `caseFuture(5)` *renvoie 5 (c'est un numéro de case stable),* `caseFuture(1)` *renvoie 38 (c'est un numéro de case avec échelle) et* `caseFuture(17)` *renvoie 7 (c'est un numéro de case avec une tête de serpent).*

Question 3 *Quelle est la complexité de cette fonction ?*

Question 4 *Écrire une fonction* `avanceCase(case: int, de: int, choix: str) -> int` *qui renvoie la case d'arrivée lorsqu'on part de la case* `case` *et qu'on a comme résultat au lancer du dé la valeur* `de`. *La variable* `choix` *est une chaine de caractère correspondant à la stratégie de fin différente :* `"r"` *pour le rebond,* `"i"` *pour l'immobilisme et* `"q"` *pour une fin rapide. .*

Question 5 *Écrire une fonction* `partie(choix: str) -> [int]` *qui lance une partie à un joueur et renvoie la liste successive des cases visitées sur le plateau. Elle commencera donc forcément par* `0` *et finira forcément par* `100`. *Le choix du mode de fin est en argument, de façon similaire à la question précédente.*

1.3 Plus court chemin

On souhaite, dans cette partie, utiliser un algorithme glouton pour trouver la partie la plus courte.

Question 6 *Écrire une fonction* `casesAccessibles(case: int) -> [int]` *qui renvoie la liste des 6 cases accessibles pour la case donnée en entrée. Vous utiliserez la fonction* `avanceCase` *de la question 4. La liste renvoyée* `cases` *doit avoir le codage suivant :* `case[i]` *doit correspondre à la case d'arrivée avec le résultat de dé* `i+1` *(donc la liste retournée doit toujours avoir une longueur de 6). On prendra l'option de fin rapide.*

Question 7 *Écrire une fonction* `meilleurChoix(case: int) -> int` *qui renvoie la meilleure case accessible depuis* `case`. *Il est interdit d'utiliser la fonction* `max` *dans cette question.*

L'algorithme glouton consistera à choisir la valeur du dé permettant de maximiser son déplacement à chaque coup.

Question 8 *Écrire une fonction* `partieGloutonne() -> [int]` *qui renvoie la liste des cases par lesquelles passe le pion dans l'algorithme glouton.*

Cette dernière fonction nous renvoie `[0, 38, 44, 50, 67, 91, 97, 100]`.

Annexe

Utilisation du module `random`

On vous donne les docstrings correspondant à deux fonctions du module `random` :

```
randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

Complexité des opérations sur les listes et dictionnaires

Principales opérations sur les listes

`n`, longueur de la liste `L`, `k`, un indice valide en négatif (`1` à `n`).

Opération	Moyen
Longueur (`len(L)`)	$O(1)$
Accès en lecture d'un élément	$O(1)$
Accès en écriture d'un élément	$O(1)$
Copie (`L.copy()` ou `L[:]`)	$O(n)$
Ajout (`L.append(elt)` ou `L+=[elt]`)	$O(1)$
Extension (`L1.extend(L2)` ou `L1+=L2`)	$O(n_2)$
Concaténation (`L1 + L2`)	$O(n_1+n_2)$
Test de présence (`elt in L`)	$O(n)$
Désempiler dernier (`L.pop()`)	$O(1)$
Désempiler autre (`L.pop(-k)`)	$O(k)$
Maximum ou minimum (`max(L)` et `min(L)`)	$O(n)$
Tri (`L.sort()` ou `sorted(L)`)	$O(n\log(n))$

Principales opérations sur les dictionnaires

`n`, longueur du dictionnaire `d`, `k`, une clé du dictionnaire.

Opération	Moyen
Longueur (`len(d)`)	$O(1)$
Accès en lecture d'un élément (`x = d[k]`)	$O(1)$
Accès en écriture d'un élément (`d[k] = x`)	$O(1)$
Copie (`d.copy()`)	$O(n)$
Ajout (`d[k] = x` la première fois)	$O(1)$
Test de présence (`k in d`)	$O(1)$
Retrait d'un élément (`del d[k]` ou `d.pop(k)`)	$O(1)$

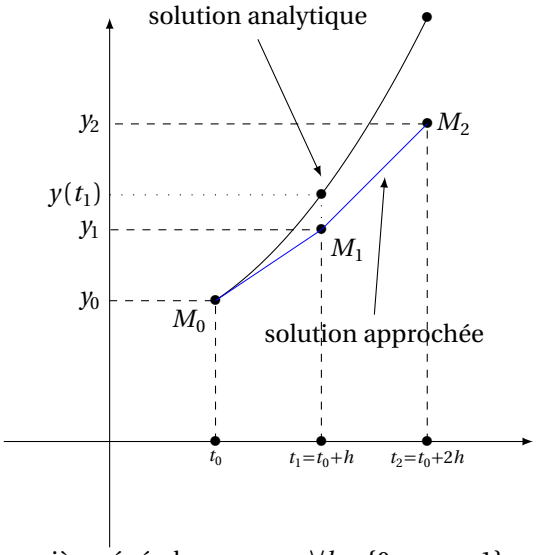
2 Principe de la méthode de la méthode d'Euler

Soit y l'unique solution de

$$\begin{cases} y'(t) = F(t, y(t)) \\ y(t_0) = y_0 \end{cases} \quad \text{avec } (t_0, y_0) \text{ fixé.}$$

On souhaite obtenir une approximation de la fonction y sur l'intervalle $[a, b]$.
Soit $n \in \mathbb{N}^*$.

- On découpe l'intervalle $[a, b]$ en n sous-segments de même longueur $h = \dfrac{b-a}{n}$.
- On pose : $t_0 = a \quad t_1 = t_0 + h \quad t_2 = t_0 + 2h \quad \ldots \quad t_k = t_0 + kh \quad t_n = t_0 + nh = b$. On part de t_0.
- On approche la portion de courbe entre t_0 et t_1 par la tangente à la courbe au point d'abscisse t_0.



Le point $M_1(t_1, y_1)$ appartient à la tangente à la courbe au point $M_0(t_0, y_0)$.
Alors, $y'(t_0) \approx \dfrac{y_1 - y_0}{h}$ D'où, $y_1 = y_0 + h\,y'(t_0)$.
Soit encore, $y_1 = y_0 + hF(t_0, y_0)$.
y_1 est une valeur approchée de la valeur exacte $y(t_1)$.

- De manière générale, on pose : $\forall k \in \{0, \ldots, n-1\}$, $y_{k+1} = y_k + hF(t_k, y_k)$. Ce qui amène à construire successivement les points M_k de coordonnées (t_k, y_k). La ligne polygonale reliant ces points est alors une approximation de la courbe représentative de la solution.

2.1 Programmation de la méthode

Question 9 *Considérant l'équation différentielle d'ordre 1 $y'(t) = F(t, y(t))$, écrire en python une fonction* `Euler(F, a, b, n)` *d'arguments la fonction F, les bornes a et b de l'intervalle d'étude, la condition initiale $y0$ et le nombre d'étapes n. Cette fonction renverra une liste de temps et une liste de valeurs approchées par la méthode d'Euler.*

Question 10 *Tracer la solution analytique et la solution approchée donnée par la méthode d'Euler pour l'équation différentielle :*

$$y'(t) + t\,y(t) = 0$$

avec $y(0) = 1$. On prendra $a = 0$, $b = 1$, $n = 100$.

2.2 Application

Dans cet exercice, on souhaite étudier une fonction $t \mapsto y(t)$ sur un intervalle $[a, b]$. La fonction y est solution de l'équation différentielle $y'' = -\dfrac{1}{y^2}$ avec les conditions initiales $y(a) = y0$ et $y'(a) = yp0$.

Question 11 *Mettre l'équation différentielle considérée sous forme d'un système de deux équations différentielles du premier ordre en introduisant une fonction auxilliaire $z(t) = y'(t)$.*

Question 12 *Compte tenu du système d'équations différentielles, comment exprimer $z(t+h)$ et $y(t+h)$ en fonction de h, $y(t)$ et $z(t)$ (ou h est la pas de la subdivision) ?*

On souhaite résoudre l'équation différentielle sur l'intervalle $[a, b]$ en utilisant N intervalles, soit $N+1$ points du segment $[a, b]$ (le premier vaut a, le dernier vaut b).

Question 13 *En supposant N, a et b préalablement définis dans le programme, écrire des lignes de code pour calculer le pas* `h` *ainsi que la liste* `les_t` *des $N+1$ instants équirépartis entre a et b.*

Question 14 *Écrire une fonction* `euler2(a,b,N,y0,yp0)` *qui calcule et renvoie les listes* `les_t`, `les_y` *et* `les_z` *correspondant aux différents instants et aux valeurs approximées par la* ***méthode d'Euler*** *des fonctions y et z à ces différents instants.*

On peut montrer (la justification n'est pas demandée ici) à l'aide de développements limités à l'ordre 2 que

$$y(t+h) - 2y(t) + y(t-h) = y''(t).h^2 + O(h^3).$$

Une méthode appelée **méthode de Verlet** consiste à négliger le terme en $O(h^3)$ et à écrire :

$$y(t+h) - 2y(t) + y(t-h) = y''(t).h^2.$$

Question 15 *Compte tenu de l'approximation de la méthode de Verlet et de l'équation différentielle, exprimer $y(t+h)$ en fonction de $y(t)$, $y(t-h)$ et h uniquement.*

Question 16 *Pourquoi ne peut-on pas utiliser cette expression pour le calcul du premier point (c'est-à-dire pour le calcul de y en $t_1 = a+h$) ?*

Question 17 *A l'aide d'un développement limité à l'ordre 2 et en négligeant le terme en $O(h^3)$, exprimer $y(a+h)$ en fonction de h, $y0$, $yp0$ et $y''(a)$.*

Question 18 *En tenant compte de l'équation différentielle, modifier l'expression précédente pour exprimer $y(a+h)$ en fonction de h, $y0$, $yp0$ uniquement.*

Question 19 *Ecrire une fonction* `verlet(a,b,N,y0,yp0)` *qui calcule et renvoie les listes* `les_t`, `les_y`, `les_z` *par la méthode de Verlet.*

Pour comparer les méthodes, on se propose de trouver une intégrale première du mouvement. Pour cela, on multiplie l'équation différentielle par y' et on prend une primitive.

Question 20 *Montrer que $E(t) = \frac{1}{2}y'(t)^2 - \frac{1}{y(t)}$ est une constante (indépendante de t).*

Question 21 *Ecrire les lignes de code permettant de tracer la courbe $E(t)$ obtenue avec la méthode d'Euler et celle obtenue avec la méthode de Verlet sur le même graphique ($a=0$, $b=20$, $y0=10$, $yp0=0.03$ et on se limitera à une graduation des ordonnées comprise entre -0.1 et -0.0975)*