TP 13
Dictionnaires et Piles – Sujet

0.1 SUPPRIMER LES DOUBLONS AVEC UN DICTIONNAIRE

0.2 DRONE

0.3 SOMMER LES ELEMENTS DUNE PILE

0.4 EXERCICES SUR LES PILES

Snakes and ladders : le jeu

Extrait du travail de T. Kovaltchouk - UPSTI

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.

Le plateau

- Le plateau comporte 100 cases numérotées de 1 à 100 en boustrophédon [*] : 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.

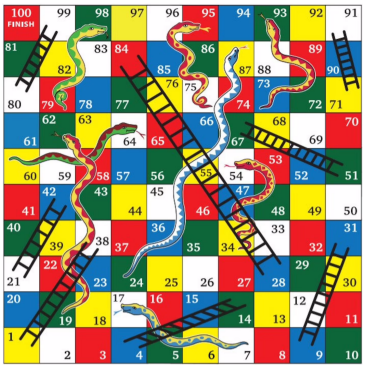

FIGURE 1 – Exemple d'un plateau de serpents et échelles

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).

[*] à la manière du bœuf traçant des sillons, avec alternance gauche-droite et droite-gauche

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.

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 : `cases[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.
```