Sorbonne Université **M1 Mathématiques**
2019–2020 **MU4MA056 : Programmation en C++**

TP noté 1 : Échelles et serpents

IMPORTANT : à l'issue du TP, chaque groupe envoie par email trois fichiers `echelleetserpent.hpp` , `echelleetserpent.cpp` et `test_echelleetserpent.cpp` . **Il est impératif de mettre en commentaire, dans tous les fichiers, les nom, prénom et n° d'étudiant de chacun des membres du groupe.**

Le jeu des échelles et serpents est un jeu qui se joue avec un plateau de jeu constitué de cases numérotées de 1 à N, un dé à 6 faces et autant de pions que de joueurs. Au départ, tous les pions des joueurs se trouvent sur la case de départ. Pour commencer, le premier joueur lance le dé et avance son pion du nombre indiqué sur le dé :

— si le pion arrive sur le bas d'une échelle, il monte dans la case où il y a le haut de l'échelle ;
— si le pion arrive sur la queue d'un serpent, il descend dans la case où il y a la tête du serpent ;
— si le pion arrive sur une case normale, il ne bouge pas ;
— si le pion arrive sur une case déjà occupée par un autre joueur, il retourne à la case départ.

Pour gagner, il faut dépasser en premier la dernière case. Les joueurs jouent l'un après l'autre, chacun leur tour, jusqu'à quand l'un d'entre eux ne gagne.

Implémentation en C++. Nous imposons la structure et la classe suivantes dans un fichier `echelleeteserpent.hpp` :

```
struct Player{
    int position;
    int n_step;
    Player():position(0), n_step(0){};
};

class SnakesAndLadders{
private:
    int n_players;
    int n_board;
    std::vector<Player> players;
    std::vector<int> board;
};
```

La structure `Player` décrit un joueur, en donnant sa position et le nombre de coups (lancés de dé) effectués par ce joueur (à chaque tour ce compteur est incrémenté de 1). La classe `SnakesAndLadders` contient la description d'un jeu :

— un champ `n_players` contenant le nombre de joueurs ;
— un champ `n_board` contenant la taille du plateau de jeu ;
— un champ `players` contenant les joueurs ;
— un champ `board` contenant le plateau, pour lequel nous allons adopter la convention suivante :
  - si `board[i]==i` : la case du plateau de jeu est une case normale ;
  - si `board[i]==j` avec `i<j` : il s'agit d'une échelle qui mène de la case `i` à la case `j` ;
  - si `board[i]==j` avec `i>j` : il s'agit d'un serpent qui mène de la case `i` à la case `j` .

Attention : **On veillera à étiqueter `const` toutes les méthodes et les arguments nécessaires.**

1. Recopier ce code dans un fichier `echelleeteserpent.hpp` et ajouter les `include` nécessaires. Préparer également un fichier `echelleeteserpent.cpp` avec les `include` nécessaires.

2. Écrire un constructeur par défaut qui crée un jeu vide.

3. Écrire un constructeur qui prend en argument 2 entiers `n` et `N` et qui crée un jeu avec `n` joueurs, un plateau avec `N` cases, avec 0 échelles et 0 serpents.

4. Écrire un constructeur qui prend en argument un flux d'entrée `std::istream &` vers un fichier de la forme suivante :

```
n_players
n_board
board_0 board_1 ... board_{n_board-1}
```

5. Ajouter un accesseur `get_position` qui prend en argument un entier `j` et renvoie la position du joueur `j` et un accesseur `get_n_step` qui prend en argument un entier `j` et renvoie le nombre de coups joués par le joueur `j` .

6. Ajouter deux méthodes `is_ladder` et `is_snake` qui prennent en argument un entier j et renvoient un booléen qui dit si la case `j` est occupée par le début d'une échelle ou par le début d'un serpent respectivement.

7. Surcharger l' opérateur `<<` de telles sortes qu'un jeu soit écrit avec le même format que le constructeur par flux.

8. Écrire un programme complet `test_echelleetserpent.cpp` qui fait les choses suivantes :

- on teste le constructeur par défaut, en affichant un jeu `S` vide ;
- on teste le constructeur avec 2 entiers, `n=1, N=100` , en affichant le jeu `S_trivial` ;
- on teste le constructeur par flux, en lisant le fichier `jeu.dat` et on affiche le jeu `S1` ;
- on teste la méthode `is_ladder` sur la $4^{\text{ème}}$ case (`j = 3`) du jeu `S1` , la bonne réponse est oui.

Attention : tant que vous n'obtenez pas les résultats attendus à la question précédente, il est inutile de continuer.

Nous rappelons que, en utilisant la bibliothèque `random` du standard C++11 et la bibliothèque `ctime` de la STL, nous pouvons créer et initialiser un générateur de nombres aléatoires

```
std::mt19937 gen(time(NULL));
```

et déclarer une loi uniforme discrète sur $\{i_1, i_1+1, \ldots, i_n-1, i_n\}$

```
std::uniform_int_distribution<int> U(i_1,i_n);
```

Une réalisation de cette loi, indépendante des précédentes, s'effectue via l'appel

```
int realisation = U(gen);
```

9. Ajouter la méthode `roll_die` qui prend en argument une référence vers un générateur de nombres aléatoires et renvoie un entier correspondant à un lancer de dé de 1 à 6.

10. Ajouter la méthode `one_step` qui prend en argument une référence vers un générateur de nombres aléatoires et un entier `j` et qui fait jouer un tour au joueur `j`, en suivant les étapes suivantes :

- on lance le dé ;
- le joueur `j` avance son pion du nombre indiqué sur le dé, en suivant les règles du jeu ;
- s'il dépasse la dernière case, il a gagné.

Cette méthode renvoie un booléen : `true` si le joueur a gagné, `false` sinon.

Pour vérifier si une case `pos` est occupée par un joueur, on pourra utiliser la fonction `std::any_of` de la bibliothèque `algorithm` sur le vecteur des joueurs. Le prototype de cette fonction est le suivant :

```
template <class InputIterator, class UnaryPredicate>
bool std::any_of (InputIterator first, InputIterator last, UnaryPredicate pred);
```

où `pred` est une fonction booléenne, qui dans ce cas pourra être déclarée en utilisant une λ-fonction, qui capture la variable `pos`, prend en argument un jouer `p` et renvoie le booléen qui dit si oui ou non la position de `p` est égale à `pos`.

11. Compléter le fichier `test_echelleetserpent.cpp` avec des tests sur le jeu `S_trivial` :

- on initialise un générateur de nombre aléatoires ;
- on lance le dé 100 fois et on calcule la moyenne empirique des réalisations (environ 3.5) ;
- on joue 3 fois en affichant la réalisation du dé (ajouter momentanément un affichage dans la méthode `one_step`, qu'on commentera par la suite), et vérifier que le joueur avance bien du nombre de cases attendues (appeler la méthode `get_position` sur le seul joueur du jeu).

Attention : tant que vous n'obtenez pas les réponses attendues à la question précédente, il est inutile de continuer.

12. Ajouter la méthode `game` qui prend en argument une référence vers un générateur de nombres aléatoires, qui joue une partie et qui renvoie l'entier correspondant au joueur gagnant.

13. Écrire une méthode `reset` qui permet de recommencer une partie, c'est-à-dire qui remet tous les pions sur la case de départ et qui remet à 0 les compteurs de coups des joueurs, tout en conservant le même plateau de jeu.

14. Dans le programme principal, calculer la moyenne et l'écart-type empiriques du nombre de coups nécessaires pour gagner une partie dans le jeu `S1` sur un échantillon de 10000 parties (il ne faudra pas oublier d'utiliser convenablement la méthode `reset`). Vous devriez obtenir deux valeurs proches de $\mu = 8.6$ et $\sigma = 6.8$.