Jeu de l'oie

L'objectif est l'implémentation du "jeu de l"oie" (*goose game*), bien connu des plus jeunes.

Le jeu
Ce jeu se présente sous la forme d'un plateau constitué dans la version traditionnelle de 63 cases numérotées à partir de 1 (plus une position de départ qui peut être considérée comme une 64ème case numérotée 0). Pour gagner et finir la partie, il faut qu'un joueur arrive exactement sur la case 63.

Les différents joueurs, en nombre quelconque, jouent chacun leur tour. Selon la case où il se trouve un joueur a ou non le droit de quitter la case où il se trouve. Si ce n'est pas le cas, il passe son tour. S'il est autorisé à jouer il lance 2 dés à 6 faces et avance son pion d'un nombre de cases égal à la somme des dés pour arriver sur sa case destination. Si son lancer de dés lui fait "dépasser" la dernière case, il revient en arrière du nombre de cases en excès. Par exemple, un joueur situé sur la case 57 fait 9 aux dés, il avance jusque la case 63 en dépensant 6 points et recule ensuite des 3 points restants pour arriver sur la case 60.

Une arrivée sur une case peut avoir plusieurs conséquences :

- elle déclenche un « rebond » :
  - si la case est une "case oie", le joueur double le score du dé et « rebondit » donc en avançant encore son pion d'autant de cases qu'indiqué par les dés ;
  - si la case est une "case de téléportation", le pion « rebondit » jusqu'à une autre case prédéfinie ;
  - pour toutes les autres cases, il ne se produit rien de particulier : le joueur reste sur la case, ce qui revient à considérer qu'il « rebondit » sur place,ou qu'il fait un « rebond de taille 0 ».
- dans tous les cas, si la case d'arrivée est déjà occupée, le pion du joueur remplace le pion déjà présent et ce dernier est placé sur la case précédemment occupée par le joueur. Il n'y a donc toujours qu'au plus un joueur par case, sauf pour la case de départ qui est un cas très particulier.
- certaines cases ne pourront être quittées que sous certaines conditions :
  - si la case est une "case piège", le joueur ne pourra plus la quitter tant qu'il restera dans cette case (c'est-à-dire jusqu'à ce qu'un autre joueur tombe également sur cette case et que la première situation décrite se produise) ;
  - si la case est une "case d'attente", le joueur ne peut quitter le case que pendant un nombre de tours prédéfini pour chaque case (sauf si un autre joueur arrive sur cette case et le renvoie donc vers sa case initiale, c'est au joueur arrivant de se retrouver en attente pour le nombre de tours initialement prévu pour la case) ;

On fait le choix que les « rebonds » dus à l'arrivée sur une case oie ou une case de téléportation ne se cascadent pas. C'est-à-dire que si le premier rebond fait arriver sur une case qui devrait elle-même produire un rebond, ce second rebond n'est pas appliqué.

Dans la *version originale* du jeu, les "cases oie" sont les cases 9, 18, 27, 36, 45 et 54 ; les "cases piège" sont les cases 31 (le puits) et 52 (prison) ; la case 19 est une "case d'attente" pour 2 tours ; les "cases de téléportation" sont les cases 6 (cheval) qui envoie en 12, 42 (labyrinthe) qui envoie en 30 et 58 (tête de mort) qui renvoie en 1.
Cependant par la suite il ne faudra pas se focaliser sur ces cases, on veut avoir la possibilité d'éventuellement personnaliser le jeu de l'oie implémenté en modifiant les effets de chacune des cases, voire même le nombre de cases du plateau.
Un exemple de partie est donné en annexe.

Mise en œuvre Nous nous intéressons donc maintenant à la conception objet de ce jeu. L'analyse a dégagé le besoin d'un certain nombre d'entités à modéliser :

▷ la classe `Game` : elle est caractérisée par un plateau et une liste de joueurs et permet de jouer une partie du jeu de l'oie, voici son diagramme UML :

Game
thePlayers : List<Player> # board : Board
≪constructor≫ + Game(board : Board) ≪methods≫ + addPlayer(p :Player) +play() ...

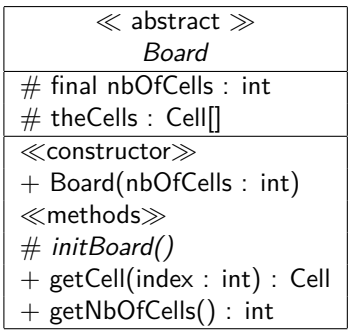
La méthode `play()` déroule une partie jusqu'à son terme quand il y en a un (une partie peut être infinie si tous les joueurs sont dans des "cases piège", on ne testera pas cette situation).

Jouer une partie consiste à faire jouer successivement chaque joueur selon les règles décrites ci-dessus.

CC BY NC SA

▷ on appelle `Cell` le type des cases du plateau de jeu.

▷ la classe `Board` est une **classe abstraite**. Elle permet de représenter le plateau de jeu contenant les différentes cases, en voici le diagramme UML :



Un plateau est constitué d'un tableau de *nbOfCells* cases, plus une case numérotée 0 : la case de départ. On a donc en tout *nbOfCells+1* cases.
Chaque case numéro i du plateau est rangée dans la case d'indice i du tableau `theCells`.
Le constructeur fait appel à la **méthode abstraite** `initBoard()`. C'est dans cette méthode que sont créées les différentes cases (`Cell`) qui constituent le plateau et sont définies dans l'attribut `theCells`.

Les sous-classes de `Board` peuvent donc personnaliser la configuration du plateau en fournissant leur implémentation de `initBoard()`.

En particulier, vous devrez créer une classe `ClassicalBoard` qui hérite de `Board`. Les instances de cette classe ont toujours 63 cases plus la case 0. De plus, la méthode `initBoard()` de cette classe initialise les objets `Cell` de `theCells` conformément à la description fournie plus haut de la *version originale* du jeu.

Q 1 . Donnez l'algorithme de la méthode `play` de la classe `Game`.
On ne cherchera pas à détecter les parties infinies (quand tous les joueurs sont dans des cases pièges).

Q 2 . En vous appuyant sur la réponse à la question précédente, construisez les diagrammes UML des entités nécessaires à la modélisation du jeu de l'oie et en particulier ce qui concerne la gestion des cases.

Q 3 . Proposez un code pour la classe abstraite `Board`.

Proposez ensuite le code pour sa sous-classe de `ClassicalBoard` correspond au plateau du jeu de base.

Q 4 . Comment proposez-vous de gérer la « case 0 » ?