3. Le jeu fait jouer le joueur jusqu'à ce que le dernier obstacle ait été trouvé[5]. Faire jouer le joueur courant consiste à :

   a. Lui faire créer et choisir l'emplacement d'une particule s'il le souhaite,
   b. Faire avancer la particule créée jusqu'à ce qu'elle soit inactive, c'est-à-dire qu'elle soit prisonnière d'un obstacle prison ou qu'elle sorte du jeu. Cela revient à :
      1. Lui commander d'avancer d'un pas,
      2. Si sa nouvelle position est celle d'un obstacle, déclencher l'action associée à l'obstacle sur la particule,
      3. Si, après 2., la particule est toujours active, recommencer en 1.
   c. Demander au joueur ses hypothèses sur les positions et le type des obstacles, puis modifier son crédit en fonction de ses réponses.

3. Propositions et conseils de modélisation

Vous avez le choix de modéliser ce jeu de la manière que vous souhaitez dans le respect des règles énoncées ci-dessus. Voici cependant quelques éléments de réflexion que vous pouvez utiliser si vous avez des difficultés à aborder le problème.

Modélisation de l'« environnement »

En programmation objet, il est aisé de modéliser un objet isolé (cas d'une fraction par exemple), mais il est plus difficile de gérer les interactions entre plusieurs objets. Il y a donc souvent un « objet central » qui joue le rôle d'environnement pour les autres objets et permet ces interactions. Ici, c'est le jeu qui est central ; il regroupe une liste de joueurs « autour » du plateau de jeu, lequel contient les obstacles. On peut lui donner le rôle de piloter le déroulement du jeu vu précédemment, par exemple par une méthode `joue`. Pour que les informations que « contient » l'instance de jeu soient accessible facilement par toutes les autres instances, il est possible de doter la classe `Jeu` d'un attribut `jeuCourant` (attribut de classe introduit par `static`) qui référence l'instance de jeu en cours — ce qui implique qu'il n'y aura qu'un seul jeu possible à la fois.

Représentation du plateau et de l'avance d'une particule

Le plateau peut être vu comme un tableau à deux dimensions d'obstacles. Il n'est pas nécessaire de matérialiser la particule dans le tableau (de la « placer » dans le tableau). Il suffit que le jeu courant (encore lui) référence la particule active — qui contient l'information sur sa position dans le plateau et sa direction. L'avancée de la particule dans le plateau sera matérialisée par la modification de sa position dans le sens de sa direction.

Représentation des positions et des directions

Contrairement à ce qui a été fait en examen, pour des raisons de simplicité dans la gestion des particules et des obstacles, il peut être intéressant de créer...

- ✓ une classe `Direction` définie par deux entiers dans l'intervalle [-1,1][6]. Une instance de `Direction` peut alors porter par exemple un opérateur `getRotation` qui, à partir d'un paramètre entier spécifiant un certain nombre (entre 1 et 3) de quarts de tours dans le sens horaire (vers la droite), produit une nouvelle instance de `Direction` dans la nouvelle orientation.
- ✓ une classe `Position` comprenant un indice de colonne (de gauche à droite) et un indice de ligne (de haut en bas). On peut alors lui associer par exemple les méthodes suivantes :
  - `getSuivante` qui, à partir d'une direction en paramètre, produit une nouvelle instance de `Position` décalée d'une case dans le sens de la direction,
  - `isBord`, `isDedans`,... qui restitue `true` si la position est sur le bord du plateau, à l'intérieur du plateau,...
  - `getDir` qui restitue la direction à prendre quand une position est sur le bord du plateau (cas d'une particule qui est lancée du bord). Elle vaut par exemple `(1,0)` pour le bord gauche, `(0,-1)` pour le bord haut...

On peut aussi doter ces classes d'une méthode `isValide` qui vérifie que les valeurs prises par les attributs sont cohérentes (par exemple, dans l'intervalle [-1,1] pour les directions...).

Interface

Il n'y a pas de représentation graphique à faire dans le cadre de cette unité d'enseignement.
Il est nécessaire au minimum d'afficher les informations qui indiquent le déroulement du jeu.

Découpage en classes

En fonction de ce qui précède, pour modéliser le jeu, on peut distinguer les classes suivantes :

- ✓ `Position`, `Direction` dont les propriétés et les fonctionnalités ont été décrites ci-dessus,
- ✓ `Particule`. Une particule a une position, une direction et un poids. Une particule peut aussi être inactive si elle est hors du tableau ou piégée par un obstacle prison (voir ci-dessous). Dans ce cas, elle ne peut pas avancer.
- ✓ `Obstacle`. Un obstacle a un poids et exécute une action sur la particule qui arrive dans la case où il se situe lorsque ce poids est inférieur à celui de la particule. Pour pouvoir créer simplement de nouveaux types d'obstacles, le mieux est que chacun apparaisse sous forme d'une sous-classe de la classe obstacle (ici `Prison`, `Deviateur`, `Teleporteur`,...). Tous obstacles ont en commun d'effectuer une action sur la particule de poids supérieur au leur. La meilleure méthode pour exprimer qu'il existe une action à effectuer dans tous les cas mais qu'elle doit être spécifique à chaque type d'obstacle est de créer une méthode abstraite `action` (sans corps) dans la classe abstraite `Obstacle` puis de donner le corps de la méthode dans chaque sous-classe (voir la partie « 7. Abstraction » du cours). Pour effectuer leur action, les sous-classes de `Obstacle` peuvent avoir besoin d'attributs spécifiques. Par exemple, `Prison` doit référencer la particule prisonnière, `Deviateur` contient un nombre de quarts de tours à appliquer et `Teleporteur` la nouvelle position de la particule.
- ✓ `Joueur`, dont les instances sont définies par leur crédit et leur pseudo. Il peut lancer une particule et émettre une hypothèse sur la position d'un obstacle.
- ✓ `Jeu`. Les paramètres du jeu sont : la taille du plateau, le nombre d'obstacles, leur poids maximal ainsi que le crédit dont est doté initialement le joueur. L'instance de jeu référence aussi le joueur le plateau et la particule active. Elle contient au moins une méthode pour initialiser le jeu et la méthode `joue` qui pilote le déroulement du jeu. Pour éviter que cette méthode soit trop volumineuse, il est nécessaire de déléguer au maximum les fonctionnalités aux autres classes.

[5] Pour constater ce fait, il peut être utile de garder en mémoire le nombre d'obstacles présents dans le plateau et décrémenter ce nombre à chaque obstacle trouvé.

[6] C'est l'incrément à apporter à la position horizontale (-1 : déplacement d'une case à gauche, 1 : idem sur la droite, 0 : pas de déplacement). Idem verticalement (-1 : haut, 1 : bas, 0 : pas de déplacement).