Exercice 3 – Objets et classes en Python avec encapsulation

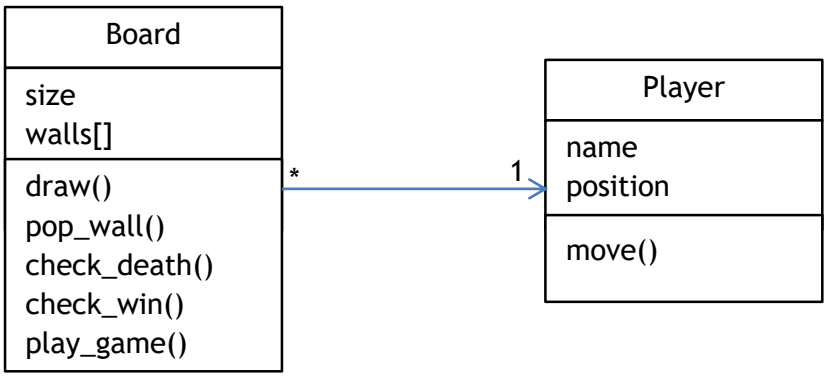
Voici un petit exercice un peu plus ludique. Il vous est demandé de le programmé en **respectant les principes d'encapsulation** vu à l'atelier.

Si vous rencontrez des difficultés, n'hésitez pas à demander de l'aide (à vos amis, profs de labo et théorie, google (mais soyez prudents et réfléchis)…). Il y a aussi le forum de moodle qui pourrait s'avérer utile.

Si vraiment, l'encapsulation vous pose problème, laissez les attributs publiques et implémentez le reste. Ça restera un bon exercice pour s'entrainer à l'orienté objet en Python.

Jeu du labyrinthe magique

Le jeu du labyrinthe magique est un petit jeu pour 1 joueur qui se joue sur un plateau carré. La taille de ce plateau peut-être variable mais elle est en général de 6x6. Le joueur commence au coin supérieur gauche et doit arriver dans le coin inférieur droit. Pourquoi ce labyrinthe est-il magique ? Parce-qu'à chaque mouvement que le joueur fait, un mur apparait sur une des cases du plateau. Si le joueur sort des limites du plateau ou se retrouve dans un mur, il perd.



Commencez par réfléchir à ce qu'implique ce diagramme en termes de classe et d'attributs. Notez bien la relation **unidirectionnelle**. En cas de besoin, n'hésitez pas à aller voir dans votre cours comment elles étaient traduites dans les classes python.

1. Représentation des cases

Une case sera représentée par un **tuple** (ligne,colonne) sachant que le (0,0) se trouve en haut à gauche.

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

2. Squelette

Pour commencer, écrivez pour chacune des deux classes leur constructeur en tenant compte des remarques suivantes :

Le constructeur de `Player` prend uniquement le **nom en paramètre**. La position du joueur sera automatiquement (0,0).

Le constructeur de `Board` prend **un joueur et une taille** (qui est de 6 par défaut) en **paramètre**. `walls`, qui retient la liste des murs présents sur le plateau (et sera donc une liste de tuple), est initialement vide.

Pensez à rendre vos attributs privés !

3. Sélecteurs et modificateurs

Écrivez les sélecteurs et modificateur de chaque attribut des deux classes.

4. Dessiner le plateau

Implémentez la méthode `draw` qui affiche le plateau de jeu. Par exemple :

```
. . . X . .
. . . . . .
. . . X . X
. . . X X .
X X . X X .
. . . . O .
```

Un point représente une case vide, une crois représente un mur et le cercle représente le joueur.

Aide : si vous avez du mal à vous lancer, commencez simplement par afficher un plateau avec uniquement des points mais qui dépend de la taille de `Board`. Une fois ceci fait, au moment d'afficher le point, regardez s'il n'y a pas un mur à cette place ou si ce n'est pas la case du joueur.

5. Faire bouger le joueur

Etape 1 : Déterminer les touches du joueur

Commencez par réfléchir aux touches (caractères) que le joueur utilisera pour se déplacer. Par exemple, zqsd, 5123, etc.

Définissez alors un dictionnaire `keyboard_key` qui associe une touche au tuple correspondant au mouvement.

Par exemple :

- *monter signifie faire -1 à la ligne ; le tuple correspondant est (-1,0)*
- *aller à droite correspond à faire +1à la colonne, donc le tuple sera (0,1)*

Donc, si j'utilise le 'z' pour monter, il y aura l'élément « 'z' :(-1,0) » dans le dictionnaire.

`keyboard_key` sera un attribut de la classe Player car tous les joueurs auront les mêmes touches.

Etape 2 : Ecrire la méthode `move`

La méthode `move` ne prend pas d'arguments. Dans un premier temps, elle demande au joueur d'entrer une touche et tant que la touche n'est pas un mouvement possible, elle redemande. Ensuite, la position du joueur est modifiée.

Pour cela, le dictionnaire `keyboard_key` va vous être utile.

6. Petit test

Dans le bloc `main`, créez un joueur et ensuite un plateau que vous affichez. Appelez la méthode move sur le joueur puis réaffichez le plateau.

Est-ce que tout fonctionne bien ?

7. Faire apparaitre des murs au hasard

Ecrivez la méthode `pop_wall` de la classe `Board` qui fait apparaitre un mur au hasard. Autrement dit, elle ajoute un tuple généré aléatoirement (dans les limites du terrain) à la liste des murs (`walls`).

Attention : si le mur existe déjà, aucun mur n'est ajouté.

Testez la méthode. Par exemple, après avoir créé le plateau, faites apparaitre 3 murs puis affichez le plateau.

8. Vérifier la victoire ou la défaite

Implémentez les méthodes `check_win` et `check_death`, chacune renvoie un booléen qui indique respectivement s'il y a une victoire ou une mort.

Rappel : il y a une victoire si le joueur se trouve dans le coin inférieur droit. Il y a une défaite si le joueur est hors limite ou si un mur se trouve sur la même case que lui.

9. Jouer (enfin presque)

Il ne vous manque plus que la méthode `play_game` qui permet de jouer une partie complète :

- Une partie commence avec l'affichage du plateau initial.
- Ensuite, tant que le joueur n'a pas gagné et n'est pas mort, celui-ci se déplace, puis, un mur apparait. Le plateau est alors réaffiché avant de passer au tour suivant.
- Quand la partie est finie, indiquez si le joueur a gagné ou perdu…

10. Jouez (vraiment ce coup-ci)

Pour pouvoir jouer, il vous suffit dès lors de créer un joueur et un plateau dans votre bloc `main` (en précisant les paramètres souhaités) et d'appeler la méthode `play_game` sur votre plateau.

Vérifiez que votre jeu fonctionne. Sortez du plateau, rentrez dans un mur, gagnez la part… Essayez avec des plateaux de différentes tailles…

Quelques questions

Voici quelques questions de réflexion. Vous pouvez simplement y répondre ou essayer d'implémentez ce qu'elles proposent.

1. Imaginons que l'on compte les points d'un joueur (par exemple, le joueur commence à 30 et perd un point par mouvement et s'il meurt, retombe à 0). Que faudrait-il changer ?

2. Imaginons, ensuite, que l'on veuille garder en mémoire le meilleur joueur (celui qui a eu le meilleur score). Comment faire ?

3. Si vous deviez implémenter une méthode pour que l'utilisateur choisisse ses touches, quels types de méthode utiliseriez-vous ?

Pour aller plus loin…

Maintenant que votre jeu est fait et fonctionne, vous pouvez l'améliorer de plein de manières différentes. Par exemple, empêcher qu'un mur n'apparaisse sur le joueur ou sur un autre mur…

Vous pouvez également faire des variantes du jeu : pouvoir aller en diagonale, mettre 2 joueurs, faire apparaitre des bonus…