Pablo Rauzy <pr@up8.edu>
https://pablo.rauzy.name/teaching/mp

Méthodologie de la programmation

TP 2 : Reversi

Dans ce TP :
— Utiliser la programmation objet (héritage, interface, encapsulation).

Exercice 0.

Démarrage et état des lieux du code fourni.

1. Pensez à organiser correctement votre espace de travail, par exemple tout ce qui se passe dans ce TP pourrait être dans `~/mp/tp2/`.
2. Récupérez les fichiers nécessaires depuis la page web du cours, ou directement en ligne de commande avec `wget https://pablo.rauzy.name/teaching/mp/05-tp-reversi.tgz`.
3. Une fois que vous avez extrait le dossier de l'archive (par exemple avec `tar xzf 05-tp-reversi.tgz`), renommez le répertoire en votre nom (avec la commande `mv mp_05-tp-reversi_files` *`votre-nom`*). Si vous ne le faites pas tout de suite, pensez à le faire avant de rendre votre TP.
4. L'objectif de ce TP est de vous faire manipuler un peu des concepts de la *programmation orientée objet*.

   Il vous est pour cela fourni le code d'un jeu de Reversi (on l'appelle aussi Othello) qui permet à deux joueur·es humain·es de s'affronter, en jouant à tour de rôle sur le même ordinateur.

   Nous allons commencer par explorer le code de ce jeu et son organisation, dans ce premier exercice. Dans les deux exercices suivant nous verrons comment écrire des IA naïves qui pourront très simplement être intégrées au code existant grâce à l'utilisation de la programmation orientée objet.
5. Le module principal du jeu est le fichier `reversi.py`.

   → Commencez par lancer le jeu et faire une partie avec votre voisin·e (mais ne perdez pas trop de temps !) pour être sûr·e que vous comprenez bien les règles.
6. Le code qui vous est fourni est composé de 4 modules :
   — `mat.py` gère le tapis de jeu et son état d'affichage ;
   — `logic.py` gère la logique du jeu et son état (règles, scores, etc) ;
   — `player.py` gère les interactions avec l'utilisateurice ;
   — `reversi.py` est le module principal, il contient le code qui fait appel aux autres modules pour faire tourner le programme : faire jouer les joueur·es tour à tour et synchroniser l'état logique et l'état d'affichage.

   Vous n'avez pour l'instant pas besoin d'entrer plus que ça dans les détails, vous êtes tout de même inviter à lire attentivement le code de chacun de ces fichiers et à essayer de le comprendre. Je vous invite également à renouveler l'opération dans quelques semaines, vous serez sûrement surpris·e !

   Ce que vous pouvez déjà remarquer, c'est que chaque module pratique l'*encapsulation* : son état interne n'est modifié depuis l'extérieur que via l'interface qu'il propose.

   Par exemple, l'objet qui représente le tapis de jeu ne permet pas à n'importe quel autre de dessiner une pièce n'importe où : il est nécessaire pour cela de passer par sa méthode `place_piece` qui va la dessiner dans une des cases de la grille de jeu.

   De même, l'objet qui représente l'état logique du jeu, il faut passer par la méthode `set_piece` pour placer une pièce, et cette méthode s'assure de la cohérence de l'état interne de la logique du jeu en mettant automatiquement à jour la liste des cases vides ainsi que les scores des joueurs.
7. Concentrons nous maintenant sur la classe `Player`.

   Elle contient :
   — un constructeur, qui prend un identifiant et un nom ;
   — un accesseur pour récupérer le nom ;
   — une méthode `play` qui fait faire un choix au joueur lors de son tour de jeu et qui renvoie une `Action`.

   Les `Action` peuvent être de type `QUIT` pour quitter le jeu ou de type `PLAY`, et dans ce cas elle a une valeur qui est la case dans laquelle jouer.

   Si on regarde dans la classe `Reversi` dans le module principal, on voit que la méthode `play` est appelé avec l'objet `self._logic` en argument, qui est une instance de la classe `Logic` qu'on a vu ci-dessus.

   Cet argument n'est pas utilisé car l'implémentation de la classe `Player` laisse une personne humaine qui est devant l'ordinateur prendre la décision du coup à jouer et celle-ci a visuellement accès à l'état du jeu. Cela nous sera en revanche utile pour écrire une IA.

Exercice 1.

Écriture d'une "intelligence" artificielle qui joue aléatoirement un coup légal.

1. → Commencez par créer un nouveau module `ia.py`.
2. Dedans nous allons créer une classe `Random` descendante de la classe `Player`. Elle *héritera* ainsi de son comportement : ses attributs et ses méthodes. Notre objectif va simplement être de *surcharger* la méthode `play` pour lui faire choisir un coup aléatoire parmi les coups autorisés.

   → Commencez par importer le module `random` de la bibliothèque standard de Python ainsi que les classe `Player` et `Action` du module `player` de notre projet.
3. → Déclarez la classe `Random` héritière de `Player` et sa méthode `play`.
4. Notre stratégie va être de lister l'ensemble des coups autorisés puis d'en choisir un aléatoirement. Pour ça, l'interface de la classe `Logic` (dont on reçoit l'instance du jeu en argument de la méthode `play`) nous fourni les méthodes suivantes :
   - `is_legal_move` qui étant donné l'id d'un joueur et une case de la grille renvoie `True` si c'est un coup autorisé et `False` sinon ;
   - `empty_cells` qui ne prend pas d'argument et renvoie la liste des cases vides.

   (a) → Déclarez une liste vide `legal`.

   (b) → Dans une boucle, pour chaque case vide, vérifiez si il s'agit d'un coup autorisé et si c'est le cas, ajoutez-le à la liste `legal` avec la méthode `append`.
5. La fonction `random.choice` prend en paramètre une liste et renvoie un élément aléatoire de cette liste.

   → Renvoyez une action de type `PLAY` aléatoire valide.
6. → Dans le module principal, importez votre classe `Random` du module `ia` de votre projet.
7. → Dans le constructeur de la classe `Reversi`, lors de l'initialisation des joueurs, remplacez l'une des deux instances de la classe `Player` par la classe `Random` (en lui donnant par exemple le nom "Aléa").
8. → Pourquoi avez-vous le droit de faire ça ?
9. → Testez votre jeu (si vous perdez contre cette IA, notez dans le rapport que je pense à vous enlever deux points à ce TP).

Exercice 2.

Écriture d'une "intelligence" artificielle gloutonne.

1. Cette fois-ci notre objectif est de faire une IA qui choisi systématique le coup qui lui rapporte immédiatement le plus de points (i.e., le coup qui va retourner le plus de pièce de l'adversaire).

   Pour cela, vous pouvez utiliser la méthode `get_flipped` de la classe `Logic` qui prend les mêmes arguments que `is_legal_move` mais qui à la place renvoie la liste des cases où des pièces vont être retournées si ce coup est joué.

   → Implémentez l'IA gloutonne dans une classe `Greedy` du module `ia`.
2. → Importez votre nouvelle IA dans le module principale et essayez de jouer contre elle et/ou de la faire jouer contre l'IA aléatoire.

   (Pour voir un peu ce qui se passe, vous pouvez rajoutez un délai artificielle par exemple d'une seconde à chaque coup d'une IA en mettant un appel à `pygame.time.wait(1000)` au début de sa méthode `play`, ce qui nécessite bien sûr d'avoir `import pygame` dans ce module).