INF1 - Projet **Reversi** UNIVERSITÉ DE RENNES 1

Avant-propos

Avant de vous expliquer le projet, nous tenons à vous rappeler et souligner les quatre consignes suivantes :

- le projet doit être rendu (sur moodle) au plus tard pour le **vendredi 13 Décembre** à **23h 59** ;
- le **plagiat est interdit** : c'est un projet en binôme, vous ne devez pas copier du code d'un autre binôme ou d'internet. Toute triche sera reportée et donnera lieu à une sanction ;
- la **qualité** du code et des **commentaires** sera pris en compte pour la correction (indentation, noms de variables significatifs...) ;
- en particulier, pour chaque fonction, vous expliquerez brièvement votre algorithme ;
- testez vos fonctions dès que vous les avez écrites et n'attendez pas la fin du projet, sinon vous ne pourrez pas débogguer facilement.

L'archive pour commencer le projet est disponible sur moodle. Le projet doit être rendu sur moodle sous forme d'une archive projet java (exportez votre projet comme pour les TPs).

Le but de ce projet est d'implémenter le jeu de *Reversi* (aussi appelé *Othello*) en Java. Si vous ne connaissez pas le jeu, vous pouvez facilement le trouver en ligne, par exemple `https://cpt.toutimages.com/jeu_othello/`.

Vous pouvez commencer par faire quelques parties pour comprendre les règles. Notez qu'il existe plusieurs variantes que vous devrez implémenter. La règle 0 correspond à la règle classique de jeu. La règle 1 est une version alternative pour ce projet. Si besoin, vous pouvez vous référer au site de la fédération française d'othello :

`http://www.ffothello.org/`

Dans le reste du sujet, les pions du joueur 1 seront représentés par le symbole `X` et ceux du joueur 2 par le symbole `O`.

Dans ce projet, on ne fera pas d'interface graphique, tout se fera directement par de l'affichage dans le terminal/Eclipse.

Le sujet se compose de cinq parties. La première consiste à initialiser et afficher une grille de jeu. La deuxième vous permet de "charger/sauvegarder" une grille. La troisième vous guide pour un coup et ses effets sur le jeu. La quatrième lie les trois parties

précédentes pour vous faire implémenter le jeu. La dernière consiste à programmer un adversaire artificiel.

On utilisera un tableau bidimensionnel nommé `plateau` qui représentera la grille de jeu. En particulier, chaque case ne pourra prendre que trois valeurs :

— `plateau[i][j] = 0` si la case est vide ;
— `plateau[i][j] = 1` si la case est occupée par le premier joueur ;
— `plateau[i][j] = 2` si la case est occupée par le deuxième joueur.

De plus, il y aura deux manières de faire références à une case :

- la première consiste à donner la ligne et la colonne d'une case, on parlera alors de **coordonnées** de la case ;
- la deuxième consiste à numéroter les cases une à une depuis la première en haut à gauche jusqu'à la dernière en bas à droite et on parlera alors de **numéro** de la case.

Partie 1 : Initialisation du jeu, affichage

1.a] Dans votre programme, déclarer les variables globales `int[][] plateau`, `int taille`, `int joueur` et `boolean passe`. La variable `plateau` contiendra les pions, la variable `taille` la taille du plateau et la variable `joueur` le numéro du joueur qui doit placer le prochain pion. La variable `passe` indiquera si le joueur précédent a passé son tour.

1.b] Écrire une fonction `init(int taille, boolean regle)` qui retourne un booléen. Elle retournera `false` si `taille` n'est pas un nombre pair strictement positif. Cette fonction initialisera le tableau `plateau` en un tableau de `taille` lignes et `taille` colonnes et affectera la valeur de la variable locale `taille` à la variable globale `taille`. Elle placera aussi 4 pions dans le carré central de la façon suivante :

— pour `regle = false` :
```
X O
O X
```

— pour `regle = true` :
```
X O
X O
```

La variable `joueur` sera initialisée à 1 et la variable `passe` à `false`.

1.c] Écrire une fonction `caseCorrecte(int i, int j)` qui retourne un booléen indiquant si la case de coordonnées `(i,j)` existe dans la grille.

1.d] Écrire une fonction `caseCorrecte(int i)` qui prend en entrée un entier `i` et qui retourne un booléen indiquant si la case de numéro `i` existe dans la grille.

1.e] Écrire une fonction `conversion2D1D(int ligne, int colonne)` qui prend en entrée les coordonnées d'une case et qui retourne le numéro de la case correspondante.

```
 |A|B|C|D|E|F|G|H|
0|#| | |#| | | | |
1| | | | | |#| | |
2| | | | | | | | |
3| | | |X|O| | | |
4| | | |O|X| | | |
5| | | | | | | | |
6| | | | | | | | |
7| | | | | | | | |
```

FIGURE 1 – Affichage pour une grille définie avec la fonction `init(8, 0)` et `casesSurlignee` $= \{0, 3, 13\}$.

1.f] Écrire une fonction `conversion1DLigne(int numero)` qui prend en entrée le numéro d'une case et qui retourne l'indice de la ligne sur laquelle se trouve la case. De même, écrire une fonction `conversion1DColonne(int numero)` qui retourne l'indice de la colonne de la case.

1.g] Écrire une fonction d'affichage `affiche(int[] casesSurlignees)` qui prend en entrée un tableau de numéros de cases et qui affiche l'état courant du plateau. Les lignes et les colonnes seront numérotées respectivement par des nombres et des lettres. Les cases occupées par le joueur 1 seront affichées avec un X tandis que celles occupées par le joueur 2 seront représentées par un O. De plus, les cases dont le numéro apparaît dans `casesSurlignees` seront affichées avec le symbole #. La figure 1 vous donne un exemple d'affichage.

Note : dans la suite on considère que la grille ne dépasse pas la taille 52 et que donc chaque colonne peut être indicée par une lettre (minuscule ou majuscule).

1.h] Écrire une fonction d'affichage `affiche()` similaire à la question précédente mais qui ne prend pas d'entrée et donc n'affiche pas de cases surlignées. Cette fonction doit appeler la fonction de la question précédente.

1.i] Écrire une fonction `score()` qui affiche le score, c'est à dire le nombre de pions de chaque joueur. Cette fonction retournera également un entier :

- 1 si le joueur 1 a un score supérieur au joueur 2 ;
- 2 si le joueur 2 a un score supérieur au joueur 1 ;
- 0 si aucun des deux joueurs n'a l'avantage.

1.j] Écrire une fonction `sauvegarde()` qui retourne une copie de la variable `plateau`.

1.k] Écrire une fonction `charge(int[][] plateau, int joueur, boolean passe)` qui réinitialise les 4 variables globales `plateau`, `taille`, `joueur` et `passe` à partir des 3

variables locales `plateau`, `joueur` et `passe`.

Partie 2 : Jouer un coup

2.a] Écrire une fonction `verifierFormat(String input)` qui prend en entrée une chaîne de caractères et qui retourne un booléen indiquant si les 2 conditions suivantes sont vérifiées :

- les premiers caractères de la chaîne forment un nombre correspondant à l'indice d'une ligne de la grille ;
- le caractère suivant est le dernier caractère de chaîne et correspond à une colonne valide de la grille.

Par exemple "2A" est une entrée valide alors que "K26T2" n'en est pas une. En particulier, "2A" correspond à la case de coordonées $(2, 0)$.

2.b] Écrire une fonction `conversionChaineNumero(String input)` qui prend en entrée une chaîne vérifiant les deux conditions de la question précdente et qui retourne le numéro de la case correspondante. Par exemple, `conversionChaineNumero`("$2A$") doit retourner 16 pour un plateau de taille 8.

2.c] Écrire une fonction `contientPiece(int numero)` qui prend en entrée le numéro d'une case et qui retourne un booléen indiquant si une pièce est présente à la case indiquée.

2.d] Écrire une fonction `retournePiece(int numero)` qui prend en argument le numéro d'une case et qui change le propriétaire de la pièce de cette case, si une pièce est présente.

2.e] Écrire une fonction `retourneDir(int joueur, int numero, String direction)` qui retourne un tableau contenant le numéro de toutes les cases devant être retournées par ce coup dans la direction indiquée. Le String `direction` peut seulement prendre comme valeur : "N", "E", "S", "O", "NO", "NE", "SE", "SO" correspondant à Nord (les cases au-dessus dans la même colonne que la case donnée en argument), Est (les cases à droite dans la même ligne)... Attention, cette fonction ne change **aucune** pièce sur le plateau.

2.f] Écrire une fonction `possibleCoups()` qui renvoie un tableau contenant le numéro de toutes les cases où le joueur courant peut jouer.

2.g] Écrire une fonction `joueCoup(int numero)` qui prend en entrée le numéro d'une case et retourne un booléen indiquant si il s'agit d'un coup valide. Dans le cas où le coup est effectivement valide, le pion sera placé sur le plateau et toutes les pièces affectées seront être changées.

2.h] Écrire une fonction `aide(int version)` (pour l'instant, l'entrée ne sert à rien) qui affiche toutes les cases où le joueur courant peut placer un pion.

Partie 3 : Boucle de jeu

3.a] Écrire une fonction `jeuBoucle(boolean regle)` qui prend en entrée la règle selon laquelle les deux joueurs vont s'affronter et qui les fait jouer à tour de rôle. Si un joueur n'a pas de coup possible, il passe automatiquement son tour et si aucun des deux joueurs n'a de coup possible, la partie s'arrête.

Avant de jouer, le joueur peut enregistrer la partie en entrant "save", demander de l'aide en entrant "help" ou quitter la partie en entrant "exit". À chaque coup joué, votre fonction doit afficher :

- le plateau ;
- le score associé ;
- le joueur qui à la main.

Dans la règle `false` :

- le joueur courant ne peut pas choisir de passer son tour ;
- le vainqueur est celui qui a le plus de pièces à la fin de la partie.

Dans la règle `true` :

- un joueur peut choisir de passer son tour ;
- si les deux joueurs passent leur tour à la suite, la partie s'arrête ;
- si la différence de score entre les deux joueurs est inférieure à 5, la partie est déclarée nulle. Sinon, le joueur ayant le plus grand score gagne.

3.b] Écrire une fonction `menu()` qui vous permet de choisir différentes options de jeu :

- jouer avec la fonction `jeuBoucle` et la règle demandée ;
- charger une partie ;
- quitter le programme.

Dans votre fonction `main`, vous ne devez appeler que la fonction `menu`.

Note : les fonctions `jeu` et `menu` devront être modifiées dans l'exercice suivant pour prendre en compte l'intelligence artificielle (i.a.).

Partie 4 : Jouer contre l'ordinateur

Le but de cette partie est de vous faire coder un adversaire joué par l'ordinateur. On vous demande trois niveaux différents décrits dans les trois questions suivantes.

4.a] Créer une fonction `premierIA` qui joue au hasard parmi les coups possibles.

4.b] Créer une fonction `deuxiemeIA` qui joue pour :

1. terminer la partie en gagnant en 1 coup si possible ;
2. maximiser le nombre de pièces retournées par le coup joué.

4.c] Créer une fonction `troisiemeIA` qui joue mieux que l'IA précédente. Vous pouvez trouver des paradigmes de programmation avec une recherche rapide sur internet. Pour vous aider, voici quelque mots-clefs qui vous aideront à créer un meilleur adversaire :

1. "méthodes Monte-Carlo, jeu" ;
2. "algorithme Min-Max, jeu, arbre alpha-beta" ;
3. "algorithmes heuristiques pour des jeux".

4.d] Modifier en conséquence votre fonction `jeuBoucle` pour permettre à un joueur de jouer contre l'ordinateur avec une sélection de la difficulté. Cette nouvelle option de jeu doit être sélectionnable dans la fonction `menu`.

4.e] Modifier votre fonction `aide` pour que lorsque `version = 0`, elle donne toujours les coups possibles pour le joueur courant. Pour `version = i` avec $i \in \{1, 2, 3\}$, votre fonction aide doit donner au joueur le coup que jouerait la i-ème version de votre i.a.