

Othello

L'arbitre

1 Généralités

L'objectif de ce projet est de construire, à vous tous, un jeu d'Othello sur ordinateur, à l'aide d'OCaml. Votre dévoué enseignant a passé ses vacances à écrire (et réécrire) un logiciel, à le diviser et le modulariser. Cet Othello peut être joué à deux sur une machine ou sur deux machines ou encore contre une intelligence artificielle disposant de nombreux niveaux de difficulté. Vous avez maintenant sous la main toutes les briques nécessaires pour reconstruire le programme, documentation y compris. Tout, mais pas le code source.

Comment obtenir un Othello en état de marche ? Dans une invite de commande, allez dans le répertoire qui contient tous ces fichiers et tapez

```
make .cmd
```

La commande produira un fichier `reversi.exe`, sur lequel vous pouvez double-cliquer ou que vous pouvez lancer en ligne de commande. Si vous tapez

```
reversi.exe --help
```

un mini-manuel s'affichera, avec la liste des options de cet Othello.

Que devez-vous faire ? Dans l'état actuel des choses, le jeu entier a été programmé par votre tyrannique enseignant. Il s'agit d'une injustice à faire cesser aujourd'hui même, en remplaçant, pièce par pièce ce qu'il a fait.

Pour ce faire, choisissez un module, consultez sa documentation, fournie dans le fichier `index.html` du répertoire `doc`, supprimez le fichier `.cmo` correspondant, qui ne vous servira plus à rien et implantez ce module dans un fichier `.ml`.

Votre objectif est d'obtenir un fichier `.ml` tel que

```
make .cmd
```

produise un nouveau `reversi.exe` qui fonctionne aussi bien – voire mieux – que l'original.

À quoi avez-vous droit ? Vous avez 10h. Vous avez le droit à tous les documents, à Internet, à prendre des pauses, à discuter, à poser des questions. Par contre, vous n'avez pas le droit de faire faire votre travail par quelqu'un d'autre.

Vous avez le droit de travailler en binôme. Par contre, les notes seront individuelles. En d'autres termes, pour chaque fonction ou structure de données que vous écrivez, notez dans les commentaires (balise `@author`) le nom de la personne qui s'en est occupée. Seule cette personne recevra des points !

Comment serez-vous notés ? Pour chaque module, vous trouverez un barème. Pour avoir le maximum de points, il faut

- que le module compile
- que le module respecte son interface et s'intègre à un Othello fonctionnel
- que les difficultés que vous avez rencontrées et les limitations de vos algorithmes soient notés dans les commentaires
- que vos fonctions soient propres et lisibles – une fois que vous avez fait fonctionner quelque chose, n'hésitez pas à le retravailler pour améliorer la clarté

- que vos algorithmes soient documentés – ceci compte pour la moitié des points
- que vous ayez marqué votre nom devant chacune de vos fonctions écrites
- éviter de parler politique.

2 L'arbitre (environ 60 points)

Le module `Arbitre` (fichier `arbitre.ml`) définit les règles du jeu : dans quelles circonstances un joueur est autorisé à placer un pion sur une case, ce qui arrive si un joueur place un pion sur une case, etc.

Ce module est conçu de manière à éviter tous les effets de bord. En d'autres termes, les fonctions peuvent être appelées depuis n'importe quel module et à n'importe quel moment, sans crainte de modifier quoi que ce soit par inadvertance.

Dépendances Le module `Arbitre` utilise les fonctions et types définis dans les modules

- `Direction`
- `Encerclement` (pour les fonctions `coup_possible` et `retourneur`)
- `Plateau`

Vous êtes donc encouragés à consulter la documentation de ces modules et à vous servir de leur contenu. Vous n'avez pas à les réimplanter.

Utilisateurs Le module `Arbitre` est utilisé par les modules

- `Main` (pour déterminer qui joue en premier)
- `Evaluateur` (pour déterminer les scores)
- `Ia` (pour calculer des coups d'avance)
- `Controle` (pour jouer effectivement)
- `Client` (pour initialiser une partie réseau)
- `Serveur` (pour initialiser une partie réseau)
- `InterfaceTk` (pour afficher les coups possibles)
- `InterfaceGraphics` (pour afficher les coups possibles).

Suggestions

- Commencez par faire la liste des fonctions que vous devez implanter. En attendant de commencer chaque fonction, remplacez leur corps par

```
failwith "la fonction Arbitre.XXX n'est pas encore implantée"
```

- Commencez par implanter les fonctions, valeurs et types les plus simples et progressez en direction des constructions les plus compliquées. Voici une suggestion d'ordre. N'essayez pas de tout faire, ni même de tout comprendre, d'un coup.

1. `premier_joueur`
2. le type `resultat`
3. le type `score`
4. la fonction `score` (à l'aide de `Plateau.fold`)
5. `determiner_vainqueur` (à l'aide de `score`)

6. `coup_possible` (à l'aide de `Encerclement.chercher_encerclement`)
 7. `coups_possibles` (à l'aide de `coup_possible` et, si vous le souhaitez, de `Plateau.fold`)
 8. la fonction `determiner_resultat` (à l'aide de `coups_possibles`, cf. plus loin)
 9. la fonction `retourneur` (à l'aide de la fonction `Encerclement.chercher_encerclement_selon_direction` et de la fonction `Encerclement.placer_selon_direction`, cf. plus loin)
 10. `jouer` (à l'aide de `retourneur`, `Direction.fold` et `determiner_resultat`)
- Ce découpage utilise une fonction locale

```
val determiner_resultat : Plateau.t -> vient_de_jouer:Plateau.couleur ->
resultat
```

Le rôle de cette fonction est de déterminer si le joueur `vient_de_jouer` doit rejouer ou passer la main à l'autre joueur ou encore si la partie est terminée, selon l'algorithme suivant :

- Si l'adversaire de `vient_de_jouer` peut jouer, alors qu'il joue.
 - Si l'adversaire de `vient_de_jouer` ne peut pas jouer mais `vient_de_jouer` peut jouer, alors `vient_de_jouer` doit rejouer.
 - Si aucun des deux ne peut jouer, la partie est terminée.
- Ce découpage utilise une autre fonction locale

```
val retourneur : x:int -> y:int -> attaquant:Plateau.couleur -> Direction.t
-> Plateau.t -> Plateau.t
```

Le rôle de cette fonction est de retourner des pions selon une direction donnée. Elle doit être appelée une fois pour chaque `Direction` possible. Ainsi, `retourneur ~x:x ~y:y ~attaquant:j d p` va retourner sur le plateau `p` un certain nombre de pions de l'adversaire du joueur `j`. Les pions concernés sont ceux qui sont rencontrés en partant de la case de coordonnées `(x,y)` et en suivant la direction `(d)`.

Cette fonction s'arrête de retourner dès qu'elle rencontre une case vide ou contenant un pion de la couleur de `j`.

Faites un dessin, ça vous aidera.

BON COURAGE !