

Othello

Interface graphique avec LablTk

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 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.

Interface graphique (20 à 40 points selon les fonctionnalités)

Le module `InterfaceTk` (fichier `interfaceGraphics.ml`) définit une interface graphique à l'aide de `LablTk`. Cette interface graphique ne connaît rien des règles du jeu, pas même le nombre de cases sur le plateau.

Les règles du jeu sont fournies par les modules `Arbitre` et `Plateau`. L'interactivité est fournie par une fonction de `controle`, tirée du module `Controle`.

Dépendances Le module `InterfaceTk` dépend de

- `Plateau`, pour connaître la largeur et la hauteur du plateau, pour manipuler les couleurs des pions...
- `Arbitre`, pour déterminer s'il est légitime de jouer sur une case
- `Vue`, pour fournir cette interface sous la forme d'une `Vue`

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 `InterfaceTk` est utilisé par le module `Main`, pour lancer la partie.

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 InterfaceTk.XXX n'est pas encore implantée"
```

ou

```
print_endline "la fonction InterfaceTk.XXX n'est pas encore implantée"
```

selon si vous voulez que le programme s'arrête en cas d'erreur ou continue en affichant juste un message d'erreur.

- Commencez par faire afficher le plateau. Lorsque cela fonctionnera, vous pourrez ajouter progressivement les fonctionnalités, à l'aide de l'argument `~command` de `Button.create`.

- Lorsque cela fonctionnera, ajoutez progressivement des fonctionnalités avancées (verrouiller l'interface graphique lorsque ce n'est pas au joueur de jouer, faire afficher automatiquement toutes les cases sur lequel le joueur actuel peut jouer, changer le curseur de la souris en fonction du joueur en cours, etc.)
- Pour le type `t`, le plus simple est d'utiliser quelque chose comme

```
type t = {
  tk      : Widget.frame Widget.widget; (*La fenêtre créée par Tk.openTk*)
  cases  : Widget.button Widget.widget array array; (*Les boutons
représentant les cases*)
  mutable interface_active: bool;
  mutable plateau: Plateau.t;
  mutable joueur_actuel: Plateau.couleur;
}
```

Vous utiliserez `cases` pour changer le contenu d'une case, `plateau` pour contenir le plateau actuel, `joueur_actuel` pour retenir quel joueur doit maintenant jouer.

Si vous ajoutez des fonctionnalités avancées, vous aurez probablement besoin d'ajouter des champs à ce type.

- La définition des couleurs est globalement impossible à trouver dans la documentation de LablTk. Pour manipuler les couleurs, utilisez donc les constantes suivantes :

```
let vert = 'Color "#00FF00"
let blanc= 'Color "#FFFFFF"
let noir = 'Color "#000000"
let rouge= 'Color "#FF0000"
let bleu = 'Color "#0000FF"
```

BON COURAGE !