Hotel Othello

Architecture

L'entrée du programme est la classe `MainWindow`, elle contient des objets `TileButton` et un `OthelloGame`. `TileButton` est une spécialisation du composant WPF `Button`. Il contient en plus des coordonnées et un propriétaire (noir, blanc ou vide).
`OthelloGame` contient la logique et les algorithmes du jeu, cette classe peut être utilisée à part (voir OthelloText).
L'attribut `tiles` de la classe `OthelloGame` est un tableau 2D d'entiers, -1 pour les cases libres, 0 pour les cases blanches et 1 pour les noires.

Algorithmique

À chaque tour, `OthelloGame` appelle `computePossibleMoves`, cette méthode remplit le dictionnaire `possibleMoves`, qui est un élément crucial de notre algorithme. Il contient en tout temps la liste des coups possibles, et en plus de ça, à chaque coup est associé une liste des tuiles capturées par ce coup.
Nous avons remarqué que cela ne compliquait pas l'algorithme de détection des coups possibles, et que ce serait donc plus performant que de refaire tout un calcul lorsque le joueur choisira un mouvement. L'algo est 2-en-1, il détermine à la fois quels coups sont possibles, et quelles cases sont capturées par chaque coup. Ainsi, pour déterminer si un mouvement est valide ou non, il nous suffit de vérifier s'il correspond à une entrée du dictionnaire `possibleMoves`.

Interface graphique

Nous avons fait en sorte que l'interface soit la plus agréable possible pour les joueurs. Le plateau de jeu garde toujours un aspect carré grâce au data-binding de propriété. En effet, la hauteur maximale du plateau est liée (bindée) à la largeur de son container parent. Et sa largeur maximale est liée à sa propre hauteur.
Les informations du jeu, comme le nombre de points et le chronomètre de chaque joueur est aussi mis à jour par data-binding.
Un menu propose au joueurs de sauvegarder la partie, d'en charger une autre, de revenir en arrière et de mettre le jeu en pause.

Chronométrage

pour afficher le temps de réflexion d'un joueur, dans un premier temps, nous avons implémenté une solution naîve. Il s'agissait d'un Timer avec un interval de 100 millisecondes qui incrémentait un entier. L'entier était ensuite formatté

et affiché. Cela semblait bien marcher, mais après des tests comparartifs avec un vrai chronomètre, nous avons remarqué que cette solution n'était pas du tout précise (5 secondes d'écart sur une minute).
Pour remédier à cela, nous avons travaillé en temps "absolu", avec les classes `DateTime` et `TimeSpan`. L'idée est de calculer un laps de temps entre le début d'un tour et l'affichage.
Exemple : `deltaT = chrono + (DateTime.Now - ReferenceTime)`, ou `chrono` est le temps additionné des tours précédents du joueur, `DateTime.Now` est l'heure exacte du stytème et `ReferenceTime` est l'heure exacte qu'il était au début du tour. `deltaT` contient alors le temps de reflexion exact du joueur.

Sauvgarde

La sauvgarde se fait avec `JsonConvert` qui permet de sérialiser et déserialiser en format JSON. L'objet sérialisé est une instance de la classe `SavableBoard`, il contient la disposition de la grille, le joueur dont c'est le tour et les deux chronos.

OthelloText

Nous avons pensé qu'il était préférable de séparer complétement les parties métier et présentation de l'application. Pour garantir cela, nous avons d'abord développé toute la partie algorithmie (OthelloGame). Nous testions avec une application console, `OthelloTester`. Nous avons donc implémenté deux interfaces utilisateur, une graphique, et une en ligne de commande, qui utilisent toutes deux le même métier.