# Page explicative de notre implémentation d’Othello

## Structure générale

Nous avons découpé notre application suivant le modèle MVC.

La classe MainWindow est le point d’entrée de notre programme et joue le rôle de la vue. Elle initialise les composants graphiques et effectue le binding des éléments dynamiques.

La classe Engine joue le rôle de contrôleur. Elle s’occupe du déroulement du jeu et s’occupe de la gestion de toutes les fonctionnalités offertes par la vue.  
  
La classe Board représente le modèle. Elle actualise l’état du plateau de jeu en fonction des coups joués. Le plateau est stocké sous la forme d’un tableau de deux dimensions.

## Points spécifiques

* Documentation exhaustive de la partie métier.
* Le sauvegarde est réalisée en sérialisant une instance de la classe SavableEngine dans un fichier XML. Cette opération est faite dans un thread anonyme. La création de la classe SavableEngine permet de contrôler et modifier facilement les données sauvegardées.
* La fonction getNextMove de l’interface IPlayable utilise un délégué qui appel la bonne stratégie en fonction du niveau de l’IA demandé. L’implémentation des stratégies est faite dans la classe IA. Cette approche permet d’ajouter simplement de nouvelles stratégies et décharge la classe Engine du code associé.
* Afin de mettre à jour les chronomètres de l’affichage graphique. Un thread Windows met à jour la propriété du chronomètre du contrôleur toutes les dixièmes de seconde.
* La classe Board possède une méthode GetSquareIndices(int value) qui permet de récupérer les indices représentant les cases dont la valeur est donnée en paramètre. Le renvoie ce fait dans une liste de tuple. Cette approche permet d’utiliser une boucle foreach pour parcourir le plateau de jeu. Cela facilite la lecture et l’écriture du code.
* La couleur du joueur ainsi que la couleur du pion d’une case sont représentés par des constantes locales (BLACK=1). Cette approche est plus lisible que des constantes globales (player = MesConstantes.BLACK) ou une valeur simple(player = 1). L’utilisation d’une énumération n’est pas pratique lors des comparaisons  
  if((int)Player.Black == (int)Square.Black)). De plus, la couleur de l’adversaire est simplement obtenue par player = -player.
* La mise à jour des éléments de la vue est exclusivement effectuée par des appels partant d’une modification d’une des propriétés de la classe Engine ou de la classe Board. Ainsi la partie logique et la partie graphique ne se mélangent pas.
* Le Undu est fonctionnel et est préservé lorsque la partie chargée depuis une sauvegarde.
* Le jeu en ligne fonctionne mais il crash si un joueur joue alors que ce n’est pas son tour.  
  Un backgroundWorker écoute la réponse de l’adversaire. Lorsqu’une réponse est obtenue, le backgroundWorker appelle la fonction PlayMove avec les paramètres récupérer sur le réseau.  
  La majorité du travail réseau est encapsulé dans la classe Network.
* L’encapsulation de la classe Engine est faite de sorte à empêcher une classe externe d’appeler des méthodes ou des propriétés qui pourraient altérer le déroulement de la partie.