Tournaments

Ajouts personnels

Notre ajout personnel est l'arbre de tournoi en lui-même, qui est généré à partir du nombre de participants pour un tournoi donné. Le raisonnement est le suivant⁽¹⁾.

Soit N le nombre de participants pour un tournoi donné. On l'exprime en fonction de la puissance de 2 inférieure la plus proche et du reste que l'on note a.

$$N = 2^n + a$$

On fait rentrer 2a joueurs au premier tour, avant de faire rentrer les N-2a autres joueurs au deuxième tour.

Le nombre de tours (rounds), noté R, équivaut à la partie entière par excès du logarithme en base 2 du nombre de participants N.

$$R = \lceil \log_2(N) \rceil$$

Exemple

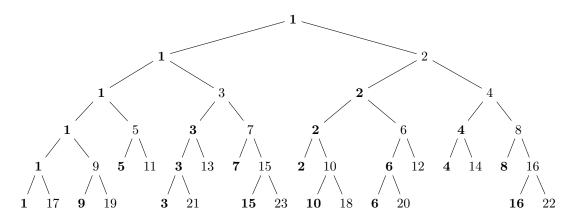
N = 23, donc $N = 2^4 + 7$, soit a = 7.

On fait rentrer 2a=14 joueurs au premier tour, il reste donc N-2a=23-14=9 joueurs qui rentreront au deuxième tour.

À l'issue du premier tour, il reste a=7 joueurs. Au deuxième tour, 9+7=16 joueurs s'affronteront. Le nombre de tours équivaut à $R=\lceil \log_2(23)\rceil=5$ (car $\log_2(23)\approx 4.52$).

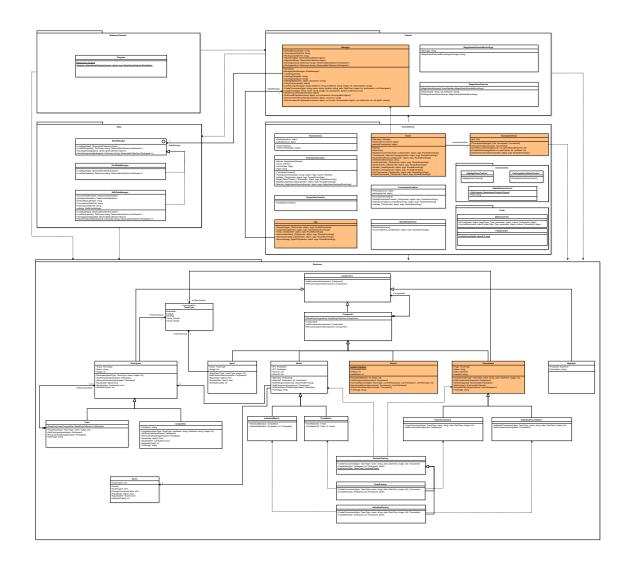
Illustration de l'exemple

Le premier nœud ne compte pas, il faut 2 participants pour former un match et un tour est formé d'un (ou de) match(es).



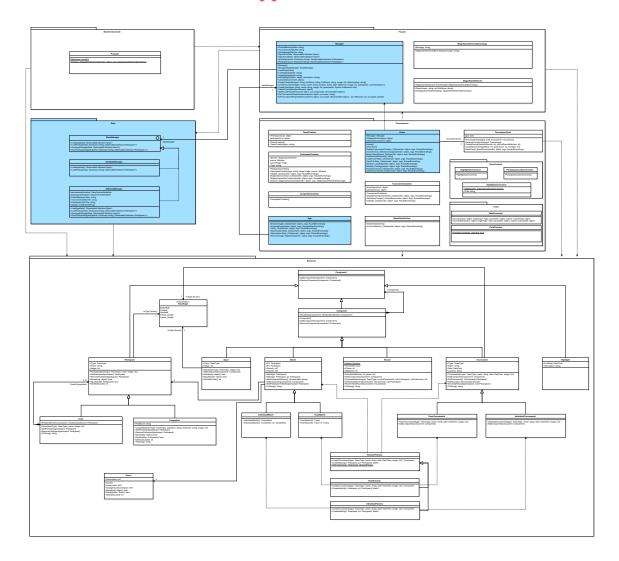
Avec une répartition quelconque, on voit qu'on arrive bien à 5 tours, avec 7 matches au tout premier tour.

 $^{{}^{(1)}}Merci\ \grave{a}\ M.\ Wohrer\ pour\ l'aide\ qu'il\ nous\ a\ apport\'ee\ pour\ trouver\ l'algorithme\ le\ plus\ \'egalitaire.$



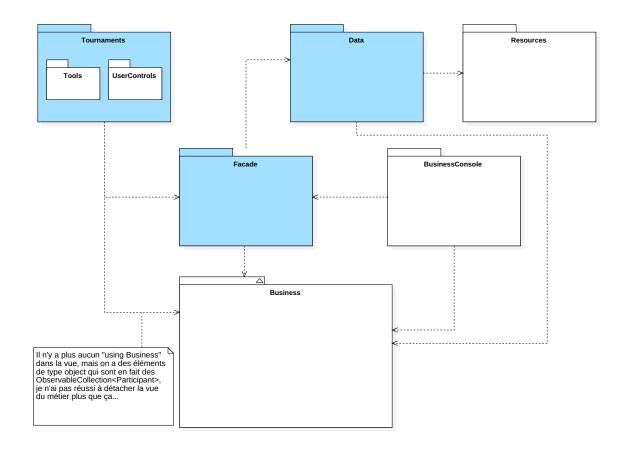
Home fait appel, au moment de la sélection d'un tournoi, au constructeur de TournamentView en effaçant le tableau déjà affiché s'il y a besoin. L'implémentation de cet algorithme se fait à la fois dans TournamentView, et dans Tournament. En réalité, le Manager (dont l'instance est stockée dans App) va se servir de Tournament et les passer à TournamentView pour qu'il puisse générer le bon nombre de lignes et colonnes de la grille passée en paramètre de son constructeur, l'arbre en lui-même et gérer l'alignement des matches par rapport au tour précédent (sauf pour les matches préliminaires). Round est utilisé, quant à lui, pour instancier le bon nombre de matches par tour. On a ainsi un arbre généré dynamiquement, ce qui nous a obligés à faire du binding en C#.

Persistence au sein de l'application



Nous avons en bleu les classes auxquelles le client a accès pour charger ou sauvegarder ses données. Toutes les classes métier à l'exception des fabriques sont sérialisées.

Le client a le choix de lancer le chargement ou la sauvegarde depuis un dossier s'il n'a pas choisi de le faire au lancement avec la boîte de dialogue ou à la fermeture de l'application. À ce moment il se situe dans Home. Cette classe va récupérer le dossier sélectionner et le passer au Manager, qui est stocké dans App qui va s'occuper de définir le répertoire courant avant de lancer le chargement ou la sauvegarde des tournois ainsi que des participants avec le IDataManager approprié, dans notre cas, le XMLDataManager.



On voit donc maintenant que la vue fait appel à la façade qui délègue ensuite le chargement ou la sauvegarde à Data, qui est responsable des données de l'application.