ARCHITECTURE DU PROJET :

projet\_tournoi/

tournoi/ # Dossier principal du projet

models/ # Dossier de classes du modèle

\_init\_.py # Initialise le module comme un package

player.py # Classe Player (représente un joueur)

match.py # Classe Match (match entre 2 joueurs)

round.py # Classe Round (un round du tournoi)

tournament.py # Classe Tournament (gestion du tournoi)

data\_manager.py # Gestion save/dl des données

controllers/ # Dossier contenant les contrôleurs

\_init\_.py # Initialise le module comme un package

tournament\_controller.py # Contrôleur gérant le tournoi

views/ # Dossier contenant les vues (affichage)

\_init\_.py # Initialise le module comme un package

display.py # Gestion de l'affichage en console

data/ # Dossier de stockage des données JSON

\_init\_.py # Initialise le module comme un package

tournaments.json # Fichier stockant les tournois

players.json # Fichier stockant les joueurs

main.py # Point d'entrée principal du programme

README.md # Documentation du projet

DIAGRAMME DE CLASSES UML :

|  |  |  |  |
| --- | --- | --- | --- |
| **Classe** | **Attributs** | **Méthodes** | **Relations** |
| **Player** | - last\_name (str)  - first\_name (str)  - birth\_date (str)  - chess\_id (str)  - score (int) | + to\_dict() → Convertit en JSON  + from\_dict(cls, data) → Instancie un joueur à partir du JSON | - Est utilisé par Match et Tournament |
| **Match** | - player1 (Player)  - player2 (Player)  - score1 (int)  - score2 (int) | + to\_dict() → Convertit en JSON  + from\_dict(cls, data) → Instancie un match depuis JSON | - Associe **deux** Player  - Contenu dans Round |
| **Round** | - name (str)  - matches (list[Match])  - start\_date (str)  - end\_date (str) | + to\_dict() → Convertit en JSON  + from\_dict(cls, data) → Instancie un round depuis JSON | - Contient plusieurs Match  - Appartient à Tournament |
| **Tournament** | - name (str)  - location (str)  - start\_date (str)  - end\_date (str)  - players (list[Player])  - rounds (list[Round])  - notes (str) | + generate\_pairs() → Génère les paires de joueurs  + start\_new\_round() → Démarre un round  + save\_to\_json() → Sauvegarde en JSON  + load\_from\_json(cls) → Charge un tournoi depuis JSON | - Contient plusieurs Round  - Liste de Player |

**relations :**

* **Tournament** possède **plusieurs Round**.
* **Round** contient **une liste de Match**.
* **Match** met en opposition **deux Player**.
* **Chaque Player a un score** qui est mis à jour après un match.

**Points Clés du Projet**

**1️ Gestion des Joueurs**

**Possibilité d’ajouter de nouveaux joueurs avant le début d’un tournoi.  
Stockage des joueurs dans une structure bien définie avec :**

* **Nom, prénom, date de naissance, ID national, score  
  Mise à jour automatique du score après chaque match.  
  Conversion en JSON pour permettre la persistance des données.  
  Import/export des joueurs dans les fichiers JSON.**
  1. **Gestion des Tournois**
* Création d’un tournoi avec un nom, un lieu, une date de début et de fin.  
   Ajout dynamique des joueurs dans le tournoi.  
   **Sauvegarde et chargement** des tournois pour continuer une compétition plus tard.  
  Génération et gestion **des rounds et des matchs**.

1. **Gestion des Rounds et Matchs**

**Génération des matchs** pour chaque round.  
**Premier round aléatoire**, puis classement des joueurs selon leurs points pour les rounds suivants. Ajout automatique des scores dans chaque match.  
 Mise à jour du classement des joueurs après chaque round.

1. **Persistance des Données**

Sauvegarde des joueurs et des tournois dans **JSON** pour éviter de perdre les informations.  
 Lecture des fichiers existants pour continuer un tournoi sans recommencer.

**5 Génération des Paires et Classement**

**Premier round** : tirage **aléatoire** des matchs.  
 **Tours suivants** : appariement selon le **score total des joueurs**.  
 Éviter que **les mêmes joueurs s'affrontent plusieurs fois**.  
 Classement dynamique après chaque match.

1. **Interface et Interactions**

Exécution du programme via **un fichier main.py**.  
 Lecture et sauvegarde **automatique** des données.  
 Messages **clairs dans la console** pour suivre le déroulement des opérations.  
 Possibilité **de reprendre un tournoi enregistré** sans le recommencer.

**Interactions entre les Modules**

**Exemple d’un scénario d’exécution :**

1. L’utilisateur **crée un tournoi** et ajoute des joueurs.
2. Le programme **génère automatiquement les paires** pour le premier round.
3. **Les résultats des matchs sont entrés**, les scores sont mis à jour.
4. **Le tournoi continue** avec de nouveaux rounds, jusqu’à la fin.
5. À chaque étape, les données sont **sauvegardées** en JSON.
6. Le programme **peut être relancé plus tard** et continuer le tournoi.