Projet 4:

Chess Manager

Le but du projet:

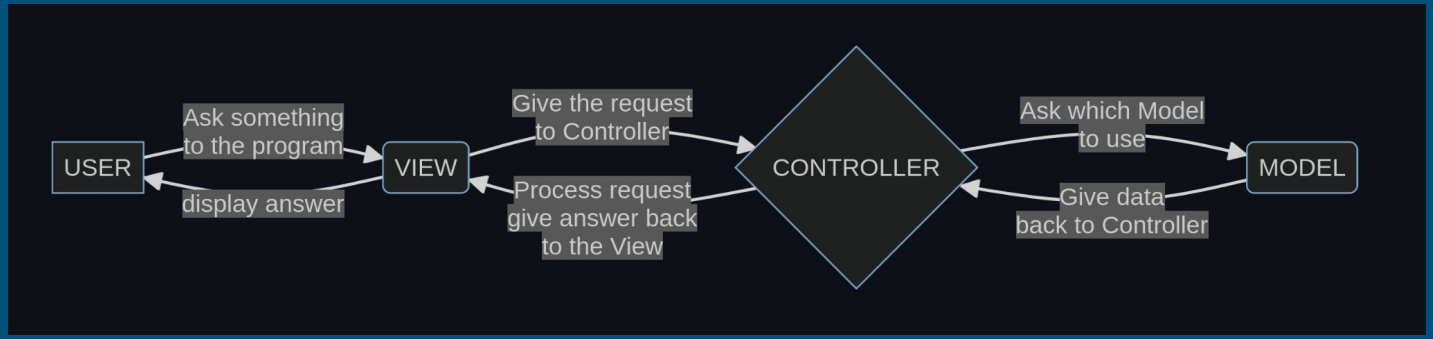
- Créer une application de gestion de tournois d'échec, permettant la sauvegarde dans, et le chargement depuis, une base de données.

- Mettre en œuvre la programmation orienté sur la base du modèle M.V.C

- La base de données sera traitée grâce à la librairie TinyDB.

Le Model M.V.C :

- Il s’agit d’une architecture de code , qui permet une meilleure organisation et une meilleure soutenabilité.
- Ainsi, le programme est divisé en trois parties. Chaque partie ayant une rôle précis, il est plus aisé d’intervenir sur le code, et de déterminer l’origine d’ éventuelles erreurs.



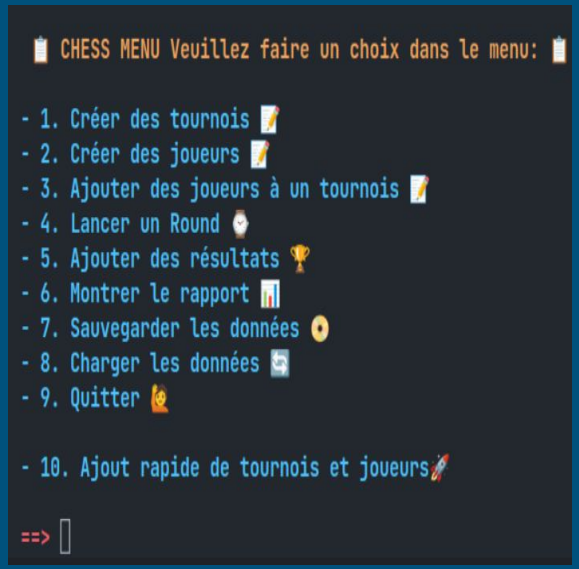
Le Model :

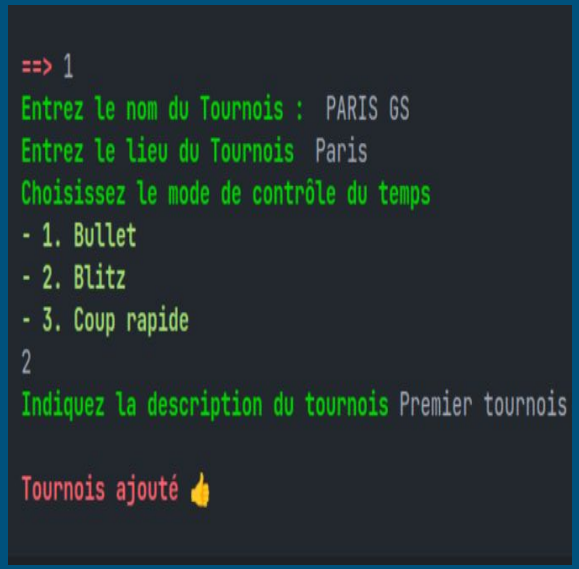
- C’est le Model qui fournit au programme les bases (Classes) au travers desquelles, le contrôleur devra traiter les requêtes de l’utilisateur.

```
class Tournament:
    def __init__(self, name: str, date: str, place: str, tours: list,
                 players: list, time_control: str, description: str,
                 player_score: dict, number_of_rounds: int = 4):
        self.name = name
        self.date = date
        self.place = place
        self.tours = tours
        self.players = players
        self.time_control = time_control
        self.description = description
        self.number_of_rounds = number_of_rounds
        self.player_score = player_score
```

La Vue :

- Elle est relative aux interactions utilisateur, c’est elle qui affiche les informations disponibles, et récupère les entrées utilisateur.





Le Contrôleur:

- C’est le liant entre le “Modèle” et la “Vue”. Sur la base du “Modèle” il va traiter les requêtes de la “Vue”. Le contrôleur crée, modifie, éventuellement supprime, des objets nécessaires au fonctionnement du programme..

```
def add_tournament(self):
    """This function get dict from tournament_view
    display_add_tournament_form() and instance a tournament in
    tournament_list
    """

    serialized_tournament = self.tournament_view \
        .display_add_tournament_form()

    self.tournament_list.append(Tournament(
        serialized_tournament["name"],
        serialized_tournament["date"],
        serialized_tournament["place"],
        serialized_tournament["tours"],
        serialized_tournament["players"],
        serialized_tournament["time_control"],
        serialized_tournament["description"],
        serialized_tournament["player_score"]))
```

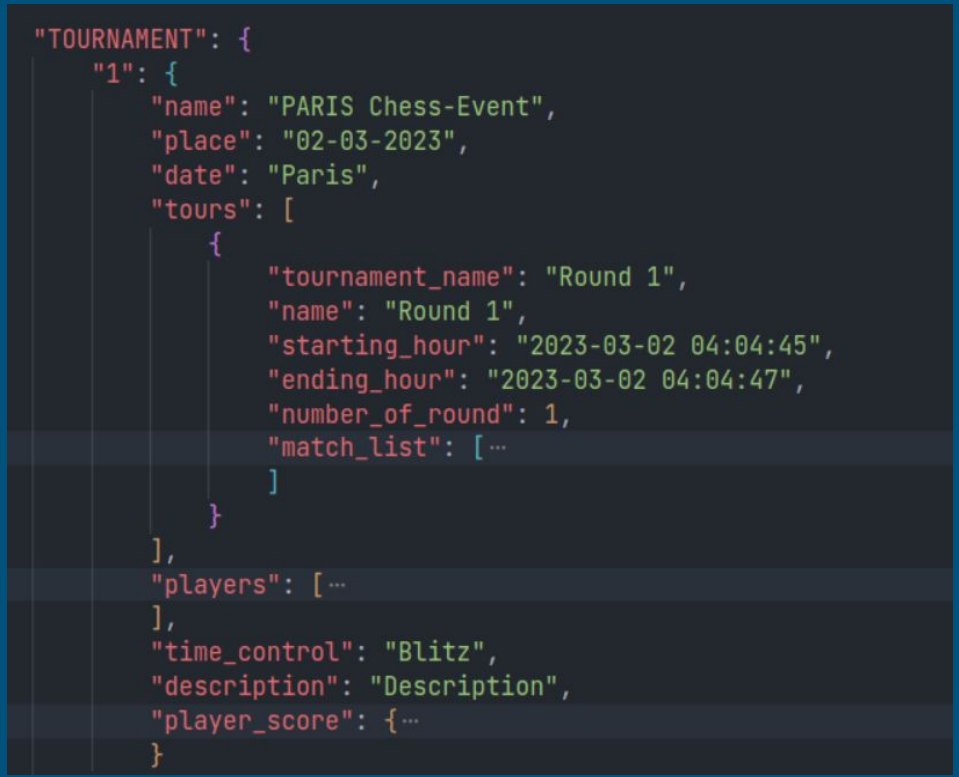
La base de données: TinyDB

- Légère, peu gourmande en ressource, fonctionnant sans dépendances, 100% Python, elle est adaptée aux “petits projets”, c’ est dans notre cas l’outil idéal.

- Elle va permettre de stocker les données du programme sous format .json , et nous serons capables grâce à cet outil, de sauvegarder et charger, donc utiliser, les données qui y sont stockées.

Fonctionnement :

- Notre base de donnée n’est autre qu’un dictionnaire dans lequel des “tables” sont créées, ici tournois et players. Les données données y sont stockées en respectant les “Model” de notre programme.



Sauvegarde :

- Cette fonction illustre la sauvegarde des données dans la table concernée, dans le respect du “Model” du tournoi.

```python
def serialised_tournament(self, tournament,table_tournament):
    """This function will be used in a loop…

    serialized_tournament = {
        "name": tournament.name,
        "place": tournament.place,
        "date": tournament.date,
        "tours": [],
        "players": [],
        "time_control": tournament.time_control,
        "description": tournament.description,
        "player_score": {}
    }

    table_tournament.insert(serialized_tournament)
```

Chargement :

- Cette fonction illustre le chargement des données dans la mémoire du programme, toujours en respectant le “Model” du tournoi.

```
def load_touranment(self):
    """This function will load tournament from database in controller…

    opener = open('db.json')

    data = json.load(opener)

    tournament_deserializer = []
    index = 0
    for _ in data["TOURNAMENT"]:
        index += 1
        tournament_deserializer.append(data["TOURNAMENT"][str(index)])

    for tournament in tournament_deserializer:

        if tournament["players"] ≠ [] and tournament["tours"] == []:
            self.tournament_list.append(
                Tournament(tournament["name"], tournament["date"],
                           tournament["place"], [],
                           self.load_players_in_tournament(tournament, data),
                           tournament["time_control"], tournament["description"],
                           self.load_players_tournament_p_score(tournament)))
```

Aller plus loin :

- Le programme étant opérationnel, et architecturé en M.V.C, il sera facile à l'avenir d'y implémenter de nouvelles fonctionnalités:

- Une interface graphique pourrait être mise en place.

- Un .json pourrait être généré pour chaque tournoi, et des comparaisons pourraient être traitées sous une autre format (.csv).

- Des mails pourraient être adressés aux participants du tournoi.