La Structure MVC Model-View-Controller

Pour développer ce projet, j'ai suivi le modèle de conception MVC (Modèle-Vue-Contrôleur), qui est un modèle d'architecture logicielle couramment utilisé pour concevoir des applications qui nécessitent une interface utilisateur. Ce modèle permet de séparer la logique de l'application en trois composants interconnectés : le Modèle, la Vue et le Contrôleur.

Voici comment j'ai appliqué le modèle MVC dans ce projet :

Modèle : Ces composants sont responsables de la logique métier de l'application, c'est-à-dire des opérations et des calculs qui sont effectués avec les données de l'application. Dans ce projet, les fichiers contenus dans le dossier /models/ correspondent aux modèles. Par exemple, player.py définit la classe Player et ses méthodes qui sont responsables de gérer les données relatives à un joueur.

Vue : Ces composants sont responsables de l'interface utilisateur et de la présentation des données à l'utilisateur. Dans ce projet, les fichiers contenus dans le dossier /views/ correspondent aux vues. Par exemple, main_menu.py définit la classe MainMenu qui affiche le menu principal de l'application à l'utilisateur.

Contrôleur : Ces composants sont responsables de la coordination entre le Modèle et la Vue. Ils reçoivent des entrées de l'utilisateur via la Vue, effectuent des opérations sur les données via le Modèle et mettent à jour la Vue avec les nouvelles données. Dans ce projet, les fichiers contenus dans le dossier /controller/ correspondent aux contrôleurs. Par exemple, player.py contient des fonctions qui gèrent la création et l'affichage des joueurs.

Voici un exemple de la manière dont ces composants interagissent dans le cadre de la création d'un nouveau joueur :

L'utilisateur sélectionne l'option "Créer un nouveau joueur" dans le menu principal (Vue).

Le contrôleur reçoit cette entrée et appelle la fonction create_player() dans player.py (Contrôleur).

Cette fonction crée une nouvelle instance de la classe Player avec les informations fournies par l'utilisateur (Modèle).

Le contrôleur enregistre le nouveau joueur dans la base de données en utilisant la fonction save_db() dans database.py (Contrôleur).

La vue est ensuite mise à jour pour afficher le nouveau joueur à l'utilisateur (Vue).

En suivant le modèle MVC, j'ai pu concevoir une application bien organisée et modulaire, où chaque composant a une responsabilité claire et où la logique métier est séparée de la présentation des données. Cela rend le code plus facile à comprendre, à tester et à maintenir.

Résumé de mon projet Terminé

Le Dossier /models

match.py

Le fichier match.py contient la définition de la classe Match qui modélise un match entre deux joueurs. Cette classe est utilisée pour stocker des informations sur le match, telles que les joueurs qui y participent, leurs scores, le gagnant du match et le nom du match.

La classe Match a plusieurs méthodes clés :

__init__ : Cette méthode initialise l'objet Match avec deux joueurs, leurs scores (initialement 0), le gagnant (initialement None) et le nom du match.

player_winner : Cette méthode prend en entrée le gagnant du match et renvoie une chaîne de caractères indiquant qui a gagné ou si le match a abouti à une égalité (ex aequo).

play_match : Cette méthode simule le match. Elle affiche un message à l'utilisateur, lui demandant de saisir le gagnant. Selon la réponse de l'utilisateur, elle met à jour les scores des joueurs et détermine le gagnant du match.

get_serialized_match : Cette méthode retourne une version sérialisée du match (un dictionnaire en python), qui peut être utilisée pour stocker ou transmettre les informations sur le match de manière plus compacte.

player.py

Le fichier player.py définit la classe Player, qui modélise un joueur dans votre projet. Un joueur est caractérisé par son nom, son prénom, sa date de naissance, son sexe, sa cote, son score total, le score qu'il a obtenu lors du tournoi, et une liste des joueurs avec lesquels il a déjà joué.

La classe Player a plusieurs méthodes clés :

__init__ : Cette méthode initialise un objet Player avec un nom, un prénom, une date de naissance, un sexe, une cote et un score total. Elle initialise également le score du tournoi à 0 et la liste des joueurs avec lesquels le joueur a déjà joué à une liste vide.

__str__ : Cette méthode retourne une chaîne de caractères représentant le joueur, sous la forme "prénom nom".

get_serialized_player : Cette méthode retourne une version sérialisée du joueur (un dictionnaire en python) qui peut être utilisée pour stocker ou transmettre les informations sur le joueur de manière plus compacte.

round.py
Enfin, le fichier round.py définit la classe Round, qui représente un tour du tournoi. Un tour est caractérisé par son nom, les paires de joueurs qui vont s'affronter pendant ce tour, les matchs qui vont avoir lieu pendant ce tour, la date de début du tour, et la date de fin du tour.

La classe Round a plusieurs méthodes clés :

__init__ : Cette méthode initialise un objet Round avec un nom, des paires de joueurs et une liste de matchs (qui est soit vide, soit générée à partir des paires de joueurs).

__str__ : Cette méthode retourne une chaîne de caractères représentant le tour, sous la forme "nom du tour".

create_matchs : Cette méthode crée les matchs pour le tour à partir des paires.OPENAI-LAST.EVENT.callback<|im_sep|>Le premier fichier est /models/match.py. Ce fichier définit la classe Match, qui représente un match entre deux joueurs. Chaque instance de Match contient deux instances de joueurs (player_1 et player_2), leurs scores respectifs, un éventuel gagnant, et un nom pour le match.

La méthode __init__ initialise ces attributs. Lorsqu'un nouvel objet Match est créé, cette méthode est appelée avec les instances des deux joueurs et le nom du match.

La méthode player_winner permet d'identifier le gagnant du match. Elle prend en paramètre le gagnant et renvoie une chaîne de caractères indiquant le nom du gagnant, ou si c'est une égalité.

play_match est une méthode qui met en œuvre la logique du jeu. Elle affiche des messages à l'utilisateur pour entrer le résultat du match et met à jour les scores des joueurs en conséquence. Elle met également à jour les scores du tournoi pour chaque joueur.

La méthode **get_serialized_match** est utilisée pour convertir l'objet Match en une forme qui peut être facilement stockée ou transmise. Elle renvoie un dictionnaire contenant toutes les informations importantes du match.

Le fichier est /models/tournament.py. Il définit la classe Tournament, qui représente un tournoi dans lequel les joueurs peuvent participer. Un tournoi est constitué de plusieurs rounds ou tours.

La classe Tournament a plusieurs attributs, notamment name (le nom du tournoi), location (le lieu où se déroule le tournoi), date (la date du tournoi), time_control (le type de contrôle de temps utilisé), players (la liste des joueurs participant au tournoi), rounds_number (le nombre total de tours dans le tournoi), description (une description optionnelle du tournoi) et list_round (la liste des rounds du tournoi). Ces attributs sont initialisés lors de la création d'une nouvelle instance de Tournament à l'aide de la méthode __init__.

La méthode __str__ renvoie une représentation sous forme de chaîne de caractères de l'objet Tournament. Elle retourne une chaîne formatée avec le nom du tournoi et la date.

La méthode create_round est utilisée pour créer un nouveau tour dans le tournoi. Elle utilise la méthode create_players_pairs pour créer les paires de joueurs pour le tour actuel. Elle crée ensuite un nouvel objet Round et l'ajoute à la liste des tours du tournoi.

La méthode create_players_pairs est utilisée pour créer les paires de joueurs pour un tour spécifique. Cette méthode utilise une logique complexe pour trier les joueurs en fonction de leur score total et de leur cote, puis génère des paires de joueurs en s'assurant qu'ils n'ont pas déjà joué l'un contre l'autre.

La méthode get_rankings est utilisée pour obtenir le classement des joueurs dans le tournoi. Le classement peut être basé soit sur le score du tournoi, soit sur le classement du joueur. Les joueurs sont triés en fonction du critère choisi et renvoyés sous forme de liste.

Enfin, la méthode get_serialized_tournament retourne une version sérialisée du tournoi, sous forme d'un dictionnaire. Cette méthode est utilisée pour convertir l'objet Tournament en une forme qui peut être facilement stockée ou transmise.

Le Dossier /views

Le fichier main_menu.py est le point d'entrée de votre application, gérant les interactions entre l'utilisateur et le programme via un système de menus. Le script utilise principalement le module colorama pour personnaliser l'affichage du texte dans le terminal.

La classe MainMenu est définie ici, héritant de la classe View. La méthode display_main_menu() est la fonction principale de cette classe, qui génère le menu principal et gère toutes les entrées utilisateur. Cela comprend la création de tournois, le chargement de tournois existants, la création de nouveaux joueurs et la gestion des listes (joueurs, tournois).

Lorsqu'une entrée est faite par l'utilisateur, elle est traitée par des blocs conditionnels if / elif pour diriger l'utilisateur vers le bon sous-menu. Les options incluent la création d'un nouveau tournoi, le chargement d'un tournoi existant, la création de nouveaux joueurs et l'affichage des listes de joueurs et de tournois.

Dans l'ensemble, ce fichier gère principalement l'interface utilisateur de l'application, offrant une série de menus et de sous-menus pour naviguer dans le programme. Il interagit avec plusieurs autres modules pour accéder aux données du tournoi, créer et gérer les joueurs, et jouer au tournoi lui-même.

Fichier : /views/player.py

Ce fichier est une partie du code d'une application de gestion de joueurs. Il définit deux classes : CreatePlayer et LoadPlayer, qui sont toutes deux des vues dans le modèle MVC (Model-View-Controller). Ces vues héritent de la classe View qui est probablement une classe générique pour les vues dans cette application.

La première classe, CreatePlayer, définit une méthode display_menu() qui recueille les informations sur un joueur via des entrées utilisateur. Ces informations comprennent le nom, le prénom, la date de naissance, le sexe et le classement du joueur. Chaque entrée est validée pour s'assurer qu'elle correspond à un type et/ou à un format spécifique. Par exemple, la date de naissance doit être au format DD/MM/YYYY, le sexe doit être soit H (Homme) ou F (Femme), et le classement doit être un nombre. Une fois que toutes les informations ont été collectées, la méthode affiche un message confirmant la création du joueur et renvoie un dictionnaire contenant les informations du joueur.

La deuxième classe, LoadPlayer, définit également une méthode display_menu() mais celle-ci prend un argument, nb_players_to_load, qui indique le nombre de joueurs à charger depuis une base de données. La base de données des joueurs est chargée et une boucle est lancée pour charger le nombre spécifié de joueurs. Pour chaque joueur, un menu de sélection est affiché pour permettre à l'utilisateur de choisir le joueur à charger. Si l'utilisateur sélectionne un joueur qui a déjà été chargé, un message est affiché et le nombre de joueurs à charger est augmenté. Une fois que tous les joueurs ont été chargés, la méthode renvoie une liste des joueurs chargés.

L'interaction avec l'utilisateur est améliorée grâce à l'utilisation de la bibliothèque colorama qui permet de modifier les couleurs du texte dans le terminal. Par exemple, les messages d'erreur sont affichés en rouge pour les rendre plus visibles.

Dans l'ensemble, ce fichier montre comment une application peut être structurée en utilisant le modèle MVC et comment les entrées utilisateur peuvent être validées pour s'assurer qu'elles sont correctes. Il montre également comment les données peuvent être chargées depuis une base de données et comment les interactions avec l'utilisateur peuvent être rendues plus conviviales grâce à l'utilisation de la couleur.

Fichier : /views/report.py

Ce fichier, report.py, appartient à la couche de vue dans une architecture MVC (Modèle-Vue-Contrôleur) d'un projet de logiciel pour un système de gestion de tournois d'échecs. Il fournit les interactions

nécessaires pour afficher différents types de rapports en lien avec les joueurs, les tournois et les tours de jeu.

init : Cette méthode est le constructeur de la classe Report. Elle charge les données des joueurs et des tournois depuis une base de données en utilisant la fonction load_db du module controller.database.

display_players_report : Cette méthode affiche un rapport détaillé d'un joueur. Il affiche les détails du joueur sélectionné comme le classement, le score total, le nom, le prénom, la date de naissance et le sexe. L'utilisateur peut choisir de revenir au menu précédent à tout moment.

display_tournaments_reports : Cette méthode affiche les détails des tournois. Elle présente des informations telles que le nom du tournoi, le lieu, la date, le type de contrôle du temps, le nombre de rounds et une description. De plus, elle permet à l'utilisateur de voir les détails des participants et des tours du tournoi sélectionné.

display_rounds : Cette méthode affiche les détails des rounds d'un tournoi donné. Elle donne des informations telles que le nom du round, le nombre de matchs, la date de début et la date de fin. L'utilisateur peut également voir les détails des matchs de chaque round.

sort_players : Cette méthode statique trie une liste de joueurs par classement ou par nom, selon le paramètre by_rank. Elle utilise la fonction intégrée sorted et itemgetter de Python pour accomplir cela.

L'architecture du code est conçue pour maintenir une séparation claire des préoccupations entre la logique de l'interface utilisateur, la logique métier et l'interaction avec la base de données. En outre, elle facilite la navigation de l'utilisateur à travers le système en utilisant des boucles et des interactions utilisateur claires. Les messages d'erreur sont affichés lorsque l'utilisateur entre une sélection non valide, améliorant ainsi la robustesse du logiciel.

Fichier : /views/tournament.py

Le fichier - /views/tournament.py est un module Python qui définit deux classes: CreateTournament et LoadTournament, chacune héritant de la classe View, qui est supposée être une classe de base pour la création et la gestion d'interfaces utilisateur.

La classe CreateTournament est responsable de la collecte d'informations auprès de l'utilisateur pour créer un nouveau tournoi.

La méthode **display_menu** dans cette classe est utilisée pour collecter les informations suivantes pour un nouveau tournoi :

LoadTournament: La classe LoadTournament est responsable du chargement d'un tournoi à partir de la base de données.

La méthode **display_menu** dans cette classe commence par charger tous les tournois de la base de données. Si aucun tournoi n'est trouvé, la méthode retourne False.

Si des tournois sont trouvés, la méthode présente à l'utilisateur une liste de tournois parmi lesquels il peut choisir. L'utilisateur doit entrer un nombre entier pour sélectionner un tournoi à partir de la liste.

La méthode **display_menu** retourne ensuite le tournoi sélectionné par l'utilisateur sous la forme d'un dictionnaire sérialisé.

Ces deux classes fournissent une interface pour la création et le chargement des tournois. Elles sont conçues pour être utilisées dans le cadre d'une application plus large qui gère des tournois d'échecs. Elles utilisent la classe View pour gérer l'interaction avec l'utilisateur, et la base de données pour stocker et récupérer les informations sur les tournois.

Fichier : /views/view.py

La classe View dans le fichier views/view.py sert de base pour créer des interfaces utilisateurs en ligne de commande pour ce projet. Cette classe contient deux méthodes statiques principales : get_user_entry et build_selection.

La méthode **get_user_entry** est utilisée pour récupérer l'entrée d'un utilisateur à partir de la ligne de commande. Elle prend en paramètre un message d'affichage, un message d'erreur, un type de valeur attendue et éventuellement une liste d'affirmations et une valeur par défaut.

Lorsque cette méthode est appelée, une boucle infinie commence, dans laquelle on demande à l'utilisateur d'entrer une valeur. Ensuite, la méthode vérifie le type de cette valeur. Si le type est numeric, la méthode vérifie que la valeur est numérique, la convertit en entier, puis la renvoie. Si le type est num_superior, elle vérifie que la valeur est numérique, convertit la valeur en entier, puis vérifie si cette valeur est supérieure ou égale à la valeur par défaut, puis renvoie la valeur. Si le type est string, la méthode vérifie que la valeur n'est pas numérique et la renvoie. Si le type est date, la méthode vérifie que la valeur est une date valide, puis la renvoie. Si le type est selection, la méthode vérifie que la valeur est contenue dans la liste d'affirmations, puis la renvoie.

La méthode **build_selection** est utilisée pour construire un message de sélection à partir d'un objet itérable. Cette méthode prend en paramètre un objet itérable, un message d'affichage et une liste d'affirmations. La méthode parcourt chaque élément de l'objet itérable et ajoute à son message d'affichage un texte formaté qui contient l'index de l'élément et la valeur du nom de l'élément. Ensuite, la méthode ajoute l'index de l'élément à la liste d'affirmations. Enfin, la méthode renvoie un dictionnaire contenant le message d'affichage et la liste d'affirmations.

La classe View sert de base pour les classes de vue spécifiques qui sont utilisées pour interagir avec l'utilisateur et pour récupérer et afficher les informations nécessaires pour l'application. Les classes de vue spécifiques peuvent hériter de la classe View et utiliser ou surcharger ses méthodes pour implémenter des fonctionnalités spécifiques.

Le Dossier /Controller

Fichier : /Controllers/database.py

Ce fichier est un composant crucial du projet Python. Il contient plusieurs fonctions permettant d'interagir avec une base de données via TinyDB, qui est un moteur de base de données léger et simple pour Python.

Il a des fonctions pour sauvegarder et mettre à jour des données dans la base de données. Par exemple, save_db prend un nom de base de données et des données sérialisées, puis insère les données dans la base de données. Tandis que update_db et update_player_rank sont utilisées pour mettre à jour un enregistrement existant dans la base de données.

Le fichier a aussi des fonctions pour charger des données depuis la base de données. Par exemple, load_db charge toutes les données de la base de données spécifiée et les retourne.

Il existe également des fonctions pour convertir des données sérialisées en objets spécifiques, comme load_player, load_tournament, load_rounds, et load_match. Ces fonctions prennent des données sérialisées et les convertissent en leurs objets respectifs (Player, Tournament, Round, Match).

Fichier : /Controllers/player.py

Ce fichier gère la création et la mise à jour des informations des joueurs.

Il a une fonction create_player qui crée un nouvel objet Joueur à partir des informations entrées par l'utilisateur. Cette fonction affiche un menu à l'utilisateur pour saisir les informations, crée un nouvel objet Player à partir de ces informations, sérialise l'objet Player et le sauvegarde dans la base de données.

Il a également une fonction update_rankings qui met à jour le classement d'un joueur. Selon un paramètre, cette fonction peut aussi ajouter le score du tournoi à son score total. Ensuite, elle sérialise l'objet Player et met à jour le classement du joueur dans la base de données.

En Conclusion, ces deux fichiers font partie intégrante de votre projet Python. Ils gèrent la sauvegarde, la mise à jour, et le chargement des données de la base de données, ainsi que la création et la mise à jour des informations des joueurs.

Le fichier "/Controllers/tournament.py", contient deux fonctions principales : create_tournament() et play_tournament().

La fonction create_tournament() commence par afficher un menu à l'utilisateur pour recueillir des informations sur le tournoi à créer, comme le nombre de joueurs. Ensuite, l'utilisateur est invité à choisir entre créer de nouveaux joueurs ou charger des joueurs existants. La fonction se poursuit en créant ou en chargeant les joueurs selon le choix de l'utilisateur. Une fois les joueurs prêts, un nouvel objet Tournament est créé avec les détails du tournoi et les joueurs, et ce tournoi est sauvegardé dans la base de données.

La fonction play_tournament() gère le déroulement d'un tournoi, y compris l'affichage du nom du tournoi, le décompte du nombre de rounds à jouer, la création de nouveaux rounds, et la fourniture d'un menu pour permettre à l'utilisateur de choisir parmi différentes options comme passer au round suivant, voir les classements, mettre à jour les classements, sauvegarder le tournoi ou quitter. La fonction se termine par la mise à jour du tournoi dans la base de données et le retour des classements finaux.

Le Dossier /data

Les fichiers JSON existants dans le dossier /data :

Le fichier "/data/players.json". est un fichier de données utilisé pour stocker les informations sur les joueurs. Le format JSON est un format de données largement utilisé pour le stockage et l'échange de données. Dans ce contexte, il est utilisé pour stocker des informations sur les joueurs telles que leur nom, leur classement, et leur score dans différents tournois.

Le 2em fichier est "/data/tournaments.json". Comme pour le fichier "players.json", il s'agit d'un fichier de données JSON utilisé pour stocker les informations sur les tournois. Il contient des détails tels que le nom du tournoi, la date, le lieu, le contrôle du temps, le nombre de rounds, et une liste des joueurs participants.

Le Dossier /utils

Le fichier /utils/check_date.py. Ce fichier contient une seule fonction check_date, qui vérifie si une date est dans un format correct en utilisant des expressions régulières (ou regex). Cette fonction est très utile pour garantir que les dates entrées par les utilisateurs sont valides et cohérentes tout au long du projet.

Le fichier /utils/timestamp.py. Ce fichier contient également une seule fonction get_timestamp, qui retourne l'horodatage actuel sous forme d'une chaîne de caractères. L'horodatage est généré en utilisant le module datetime et est formaté pour inclure le jour, le mois, l'année, l'heure et la minute. Cette fonction peut être utile pour marquer des événements ou des actions avec l'heure précise à laquelle ils se produisent.