Pourquoi avons-nous réalisé ces refactorings ?

Les refactorings réalisés abordent non seulement les points spécifiques mentionnés dans les TODO de la version originale, mais ils visent également à améliorer la clarté, la flexibilité, et la maintenabilité du code. En adoptant ces changements, le code devient plus aligné avec les principes de bonne pratique en développement logiciel, facilitant ainsi sa compréhension, son extension, et sa maintenance pour les développeurs actuels et futurs.

Ce que nous avons décidé de faire :

1. Centralisation des Chaînes de Caractères pour Faciliter les Traductions

Pourquoi ? La centralisation des chaînes dans `TennisTranslation` répond au TODO sur l'utilisation de chaînes littérales. Cela évite les chaînes en dur, facilitant les modifications et l'ajout de traductions.

2. Utilisation d'une Structure de Données pour les Scores

Pourquoi ? En passant à `self.player_scores`, on répond au TODO suggérant l'utilisation d'une structure de données pour les joueurs et leurs points. Cela améliore la clarté, réduit la complexité et facilite la gestion des scores.

3. Réduction de la Duplication du Code

Pourquoi ? En extrayant les fonctionnalités répétitives comme `_get_score_difference` et `_get_lead_player`, on répond au TODO sur la duplication de code. Cette approche diminue les risques d'erreurs et simplifie les modifications futures.

4. Remplacement des Nombres Magiques par des Constantes

Pourquoi ? Les modifications apportées utilisent des méthodes et des structures de données qui éliminent l'usage de nombres magiques (comme les scores 0, 1, 2, 3 transformés en termes de tennis dans `TennisTranslation`). Ceci répond au TODO sur les nombres magiques, rendant le code plus lisible et maintenable.

5. Simplification des Blocs Conditionnels

Pourquoi ? En réorganisant la logique de détermination des scores et des situations de jeu (comme le "Deuce" ou "Advantage"), on aborde le TODO concernant la complexité des méthodes. Cela simplifie la compréhension et la maintenance du code en décomposant la logique complexe en sous-parties plus gérables.

6. Gestion des Erreurs et Validation des Entrées

Pourquoi ? Le contrôle d'erreur ajouté lors de la mise à jour des scores répond au besoin implicite de validation des entrées, assurant que seuls les noms de joueurs valides sont traités. Ceci améliore la robustesse du code et sa capacité à gérer des situations inattendues.