Journal de Bord du Refactoring de TennisGame

Introduction

Nous avons été confrontés au projet de refactoriser un morceau de code pour le jeu de tennis, `tennis1.py`, avec pour objectif d'en améliorer la lisibilité, la maintenabilité et la flexibilité. Le code original souffrait de plusieurs maux : utilisation de nombres magiques, chaînes de caractères littérales, et une duplication de code importante. Notre mission était de clarifier le code et de le préparer pour l'ajout facile de nouvelles fonctionnalités, comme la prise en charge de plusieurs langues.

Étape 1 : Analyse du code original

Le début de notre travail a consisté à analyser le code original pour identifier les principaux problèmes :

- Des variables et une logique de score éparpillées et mal structurées
- Une utilisation excessive de conditions imbriquées
- Une duplication significative de code, notamment dans la gestion des scores
- L'emploi de nombres magiques et de chaînes de caractères en dur

Étape 2 : Planification du refactoring

Suite à cette analyse, nous avons planifié nos actions de refactoring :

- Centraliser la gestion des joueurs et des scores au moyen de structures de données appropriées.
- Remplacer les nombres magiques et les chaînes littérales par des constantes et des énumérations pour plus de clarté.
- Simplifier la logique conditionnelle grâce à des méthodes plus spécifiques.
- Introduire un système de traduction pour faciliter la gestion des différentes langues.

Étape 3 : Refactoring

Introduction de TennisTranslation

Notre première grande modification a été l'ajout de la classe `TennisTranslation`, qui gère la traduction des termes de score dans différentes langues, rendant le code prêt pour l'internationalisation et améliorant sa flexibilité.

Refonte de TennisGame

La classe `TennisGame` a été entièrement remaniée pour utiliser un dictionnaire pour les scores des joueurs, ce qui a rendu le code plus intuitif et a réduit les risques d'erreurs lors de la manipulation des scores.

Simplification de la logique de score

Nous avons décomposé la logique complexe de détermination du score en plusieurs méthodes plus petites et plus claires, comme `_is_early_game_all`, `_is_deuce`, et `_get_regular_score`. Cela a considérablement simplifié la compréhension du flux logique.

Étape 4 : Tests et validation

Après le refactoring, les tests ont été cruciaux pour valider nos changements. Nous avons utilisé `golden_master_test.py` pour nous assurer que le comportement du jeu restait identique après nos modifications.

Golden Master Testing

Le test Golden Master a comparé les résultats des scores entre l'ancienne et la nouvelle version pour de nombreux scénarios de jeu, ce qui nous a permis d'identifier et de résoudre rapidement les divergences.

Test d'indépendance du nom des joueurs

Un test supplémentaire a vérifié que les scores restent cohérents quel que soit le nom des joueurs. Ce test a souligné l'importance d'une gestion adéquate des noms dans notre version refactorisée.