TD - Functional Penalty

Programmation Fonctionnelle

Modalités de réalisation

- **Travail individuel ou en binôme.**
- Pour la qualité de votre apprentissage : **IA, non merci** !
- Le projet doit être réalisé progressivement en suivant les consignes fournies dans les supports de TD.
- Employez Git pour la réalisation et le rendu de votre travail.
- Avant de réaliser une étape, lisez intégralement les consignes de l'étape concernée.
- **Il ne s'agit pas d'un exercice de vitesse** : prenez le temps d'expérimenter, de tester et de refactoriser votre code.
- L'**auto-documentation** fait partie de l'exercice.
- En cas de problème, **n'hésitez-pas à me consulter**.

Critères d'évaluation

- **Respect des consignes**.
- **Qualité** du code : cohérence, nommage, faible couplage, respect des concepts de Programmation Fonctionnelle…
- **Quantité** de fonctionnalités implémentées,
- **Investissement** : autonomie, raisonnement, efforts fournis, capacités à surmonter les difficultés et force de proposition.
- **Comportement** professionnel : écoute, intérêt, participation, respect, assiduité et esprit collectif.

Modalités de rendu

- **Indiquez votre prénom et votre nom** dans un fichier **README.md** placé à la racine de votre projet.
- **Fournissez l'adresse de votre dépôt Git distant** (par mail ou message privé sur Discord). Si votre dépôt Git est privé, ajoutez ***shrp777*** ou ***alex@shrp.dev*** en tant que collaborateur du projet.
- **Rendu au plus tard le 19/12/2024 à 17h.**

Objectifs

- **Sélection d'une stack technologique compatible avec la Programmation Fonctionnelle** (langage typé statiquement : *TypeScript, OCaml, Haskell, Kotlin, Rust, Scala, Clojure…*) et mise en place des **outils de développement et de test** nécessaires à la réalisation du TD.

  Vous pouvez utiliser le projet suivant comme base de départ : https://github.com/shrp777/calculus-ratiocinator-starter-kit

- Familiarisation avec les concepts clés, les techniques et l'architecture de la Programmation Fonctionnelle.
- Familiarisation avec la **gestion d'état** avec une approche fonctionnelle.
- Emploi des **types** plutôt que des classes pour définir des structures de données.
- Production de **code déterministe donc testable**.
- Assurer l'absence d'effet de bord :
  - **en séparant la gestion des comportements et de l'état**,
  - en programmant des **fonctions pures et totales**,
  - en évitant les instructions impératives,
  - en privilégiant l'emploi de **listes** et les **méthodes fonctionnelles itératives** telles que *map, reduce, filter…*
  - en employant des **données immuables**,
  - en **gérant les erreurs sans générer d'exception**.

Logique

Une séance de tirs au but se déroule à la fin d'un match de football en cas de score de parité après les prolongations.

Dans une séance de tirs au but, chaque équipe effectue un tir, l'une après l'autre.

Le score de la séance de tirs au but est mis à jour après chaque tir :
si le tir au but est marqué, le score de l'équipe augmente d'une unité.

- **Scénario nominal** : l'équipe dont le score est le plus élevé après 5 tirs a gagné.
    - **Scénario alternatif 1** : si au cours de la séance des 5 tirs au but, une équipe dispose d'un avantage qui ne pourra pas être rattrapé par l'équipe adverse, la séance se termine immédiatement et la victoire lui est accordée (ex: 3/0, 4/1).
    - **Scénario alternatif 2** : en cas d'égalité après les 5 tirs au but, la séance se poursuit. Si l'une des 2 équipes a pris l'avantage après que chacune ait tiré, la victoire lui est accordée.

Travail à réaliser

- **Simulez le résultat de chaque tir au hasard** (pénalty marqué / pénalty manqué).
- Mettez à jour le score de la séance de tirs au but après chaque tir. Pour ce faire, employez **le concept d'état immuable**.
- Constituez un **historique du déroulé de la séance de tirs au but** permettant de retracer l'évolution du score.
- Mettez en place une fonction ***display*** permettant d'**afficher l'historique de la séance de tirs au but dans le terminal**, dans ce style :

```
Tir 1 : Score : 1/0 (Équipe A : +1 | Équipe B : 0)

Tir 2 : Score : 1/1 (Équipe A : 0  | Équipe B : +1)

Tir 3 : Score : 2/2 (Équipe A : +1 | Équipe B : +1)

Tir 4 : Score : 3/3 (Équipe A : +1 | Équipe B : +1)

Tir 5 : Score : 4/3 (Équipe A : +1 | Équipe B : 0)

Victoire : Équipe A (Score : 4/3)
```

- Faites en sorte que **votre programme se répète tant que la séance de tirs au but n'a pas permis de déterminer un vainqueur**. Pour ce faire, programmez une **fonction récursive**.
- Employez l'historique de la séance de tirs au but (obtenu ou généré) afin de tester votre programme en **rejouant une partie depuis le début ou à partir d'un moment choisi**.
- Implémentez des **tests unitaires permettant de valider chaque fonction de votre programme. Une approche Test-Driven Development est recommandée.**