Projet renforcement module JS / TS

Objectif : Concevoir une **application interactive** (jeu, outil, simulation, etc.) en utilisant **obligatoirement** les notions abordées pendant les 2 jours.

Vous avez une **liberté totale sur le thème** (aventure, puzzle, gestion, éducatif, artistique, etc.), mais votre projet doit respecter les **contraintes techniques** ci-dessous.

Livrable : Un dépôt GitHub avec le code source, un `README.md` explicatif, et une démo fonctionnelle.

Contraintes Techniques Obligatoires

Votre projet **doit** utiliser les éléments suivants (au moins une fois chacun) :

1. ES6+ et Fondamentaux Avancés

- **`let`/`const`** : Déclaration de variables (ex: `const tableau = [...]`).
- **Fonctions fléchées** : Au moins une fonction fléchée (ex: `const maFonction = () => {...}`).
- **Template literals** : Pour afficher du texte dynamique (ex: `` `${variable}` ``).
- **Destructuration** : Extraire des propriétés d'un objet ou tableau (ex: `const { nom, note } = etudiant`).
- **Spread operator** : Copier ou fusionner des tableaux/objets (ex: `[...tableau, nouvelElement]`).
- **Modules** : Séparer le code en fichiers avec `import`/`export`.
- **Closures** : Une fonction qui "mémorise" son environnement (ex: compteur, cache).
- **`this` et contexte** : Utiliser `bind`, `call`, ou `apply` pour manipuler `this`.

2. DOM et Événements

- **Manipulation du DOM** : Créer/modifier des éléments HTML (ex: `document.createElement`, `appendChild`).
- **Gestion d'événements** : Écouter des clics, touches clavier, ou mouvements de souris (ex: `addEventListener`).
- **Modification de styles** : Changer dynamiquement le CSS (ex: `element.style.width = "100px"`).
- **Animations ou interactions** : Utiliser `setInterval`, `requestAnimationFrame`, ou `setTimeout`.

3. Asynchronisme

- **Promesses** : Créer et utiliser une promesse (ex: `new Promise` + `.then()`/`.catch()`).
- **`async`/`await`** : Remplacer `.then()` par `async/await` pour au moins une opération asynchrone.

- **Gestion d'erreurs** : Utiliser `try/catch` avec des promesses ou `async/await`.
- **Opérations parallèles** : Utiliser `Promise.all` ou `Promise.race`.

4. TypeScript

- **Interfaces** : Définir au moins 2 interfaces (ex: `interface Utilisateur { ... }`).
- **Typage strict** : Typer les variables, paramètres de fonction, et retours de fonction.
- **Optional chaining** : Accéder à une propriété imbriquée en sécurité (ex: `objet?.propriete?.sousPropriete`).
- **Types unions ou optionnels** : Utiliser `|` ou `?` (ex: `propriete?: string`).

5. Architecture et Bonnes Pratiques

- **Séparation des responsabilités** : Organiser le code en modules/logiques distincts (ex: `core/`, `utils/`).
- **Gestion d'état** : Stocker et mettre à jour des données (ex: score, position, inventaire).
- **Commentaires et documentation** : Expliquer les fonctions complexes et les choix techniques.

<u>Notation :</u>

- contraintes techniques (Utilisation de let/const, fonctions fléchées, destructuration, modules, Promesses + async/await + gestion d'erreurs, Interfaces TypeScript et typage strict, Manipulation du DOM + événements) = **5 PTS**
- architecture modulaire (propre et bien pensée) = **2 PTS**
- fonctionnalités et evènements (déplacement, collision, scoring, interface de lancement et de fin) = **3 PTS**
- originalité et fluidité = **2 PTS**
- explication et justification à l'oral des points techniques et de la structure = **8 PTS**