⚔️ Mon petit jeu RPG en JavaScript (Node.js) 👾

Pourquoi j'ai fait ce projet

J'avais envie de créer un jeu simple en ligne de commande pour apprendre à mieux utiliser JavaScript et m'entraîner à la programmation orientée objet (POO).
Plutôt que de faire un projet trop gros, je suis parti sur un **combat RPG tour par tour**, avec un seul héros qui affronte des ennemis à la suite.

Le but du jeu

Tu incarnes un personnage (le "Héros") et tu dois battre **le plus d'ennemis possible**.
Chaque ennemi est un peu plus fort que le précédent. Le jeu continue **jusqu'à ta mort**, et ton score correspond au nombre d'adversaires vaincus.

Ce que j'ai appris

En faisant ce projet, j'ai appris à :

- Créer des **classes** en JavaScript avec des méthodes
- Gérer un **état de jeu** (PV, score, etc.)
- Utiliser le module `readline` pour faire des **entrées utilisateur**
- Ajouter un peu d'aléatoire avec `Math.random()` (pour les coups critiques)
- Structurer mon code de manière claire, comme un vrai petit jeu

---

Comment le jeu est structuré

1. La classe `Personnage`

Chaque personnage a :

- un nom
- des points de vie (vie)
- une attaque
- une défense

Et des méthodes comme :

```
attaquer(cible) {
  let degats = Math.max(this.attaque - cible.defense, 1);
  // Coup critique
  if (Math.random() < 0.2) {
    degats *= 2;
    console.log("✨ Coup critique !");
  }
```

```
    cible.vie = Math.max(cible.vie - degats, 0);
  }
```

Comment ça se joue

Le jeu se passe dans le terminal :

1. Le joueur commence avec 100 PV.
2. À chaque tour, il peut choisir entre :
    - **attaquer** : pour infliger des dégâts à l'ennemi
    - **soigner** : pour regagner un peu de vie
3. L'ennemi attaque ensuite automatiquement.
4. Si le joueur gagne, un **nouvel ennemi plus fort** apparaît.
5. Si le joueur meurt, le jeu s'arrête et on affiche le **score final**.

🩹 Exemple de partie

```
⚔️ Le combat commence !
🚨 Tu affrontes Ennemi 1 !

Héros : 100 PV
Ennemi 1 : 70 PV
Score : 0

Que veux-tu faire ? (attaquer / soigner) : attaquer
Héros attaque Ennemi 1 et inflige 18 dégâts !
...
```

Pistes pour aller plus loin

Voici ce que je pourrais ajouter plus tard :

- Des **classes de personnages** (Guerrier, Mage...)
- Un **système d'inventaire** ou d'équipement
- Sauvegarder le **score** dans un fichier
- Une IA plus variée pour les ennemis (choix d'actions)

Conclusion

Ce projet m'a permis de mieux comprendre la logique d'un jeu tour par tour, tout en m'amusant avec du code.
C'est simple, mais super formateur, surtout si on débute en POO avec JavaScript.

C'est quoi une classe et un objet en JavaScript ?

Une classe

En JavaScript, une **classe** est un plan de construction pour créer des objets.
C'est comme un moule à gâteau : tu définis la forme (les attributs) et les actions (les méthodes).

Exemple :

```
class Personnage {
  constructor(nom, vie, attaque, defense) {
    this.nom = nom;
    this.vie = vie;
    this.attaque = attaque;
    this.defense = defense;
  }

  attaquer(cible) {
    // Action que le personnage peut faire
  }
}
```

Un objet

Un **objet** est une "instance" de la classe, c'est-à-dire un personnage réel dans le jeu :

```
const joueur = new Personnage("Héros", 100, 20, 5);
const gobelin = new Personnage("Gobelin", 80, 10, 2);
```

Chaque objet a ses propres **valeurs** pour `nom`, `vie`, etc.

Les éléments de syntaxe JavaScript utilisés

- `constructor`

C'est une méthode spéciale appelée automatiquement quand on crée un nouvel objet avec `new`.
Elle sert à **initialiser les attributs** de l'objet.

```
constructor(nom, vie, attaque, defense) {
  this.nom = nom;
  // ...
}
```

`this`

Le mot-clé `this` désigne **l'objet courant**.
Quand tu écris `this.vie`, tu accèdes à la vie du personnage en train d'exécuter le code.

Méthodes (fonctions dans la classe)

Ce sont les **actions** que l'objet peut faire.
Par exemple, attaquer un autre personnage :

```
attaquer(cible) {
  const degats = Math.max(this.attaque - cible.defense, 1);
  cible.vie -= degats;
}
```

`Math.max()` et `Math.random()`

- `Math.max(a, b)` → retourne la plus grande des deux valeurs. Utile pour éviter que les PV deviennent négatifs.
- `Math.random()` → retourne un nombre entre 0 et 1. Utilisé pour ajouter une **chance de coup critique** :

```
if (Math.random() < 0.2) {
  degats *= 2; // double les dégâts
}
```

`readline` pour lire les commandes utilisateur

```
const readline = require("readline");
const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.question("Action ? ", (reponse) => {
  // Traitement de la réponse
});
```

Conditions et boucles

- `if (condition)` permet de prendre des décisions
- `while (...)` ou `function tourDeJeu()` permet de répéter le jeu tant que le joueur est vivant

---

Pour conclure

Grâce à ce projet, j'ai pu :

- Créer des objets à partir d'une classe
- Donner des comportements à ces objets (attaquer, se soigner…)
- Utiliser la syntaxe `this`, `constructor`, `new`
- Structurer un vrai jeu interactif en ligne de commande