Combat de personnages

On va faire un exercice standard de la POO : créer une classe personnage, et faire se battre deux personnages entre eux. Dans cette partie, il n'y a rien à faire dans le DOM, tout se fait dans la console, afin de ne travailler que l'aspect POO et JS.
J'ai volontairement poussé à l'excès le détail des choses à faire pour que tout le monde avance correctement et étape par étape.

Classe, attributs et méthodes

Partie 1 :

- créez une classe Personnage
- donnez-lui un attribut nom avec une chaîne vide comme valeur par défaut
- donnez-lui un attribut vie avec zéro comme valeur par défaut
- donnez-lui un attribut attaque avec zéro comme valeur par défaut
- donnez-lui un attribut defense avec zéro comme valeur par défaut

Partie 2 :

- créez le constructeur
- donnez-lui un paramètre "nom"
- si le nom est renseigné lors de l'appel du constructeur, assignez la valeur du paramètre à la propriété "nom" de l'objet courant, puis affichez un console.log qui affiche "Nouveau personnage NOM créé !"
- sinon affichez un message d'erreur

Partie 3 :

- après votre classe (mais dans le même fichier), créez un nouveau personnage dans une variable perso1
- vérifiez que le message d'erreur de votre constructeur fonctionne bien

Partie 4 :

- rajoutez un attribut "existe" à votre classe, avec false comme valeur par défaut
- dans votre constructeur, passez votre propriété existe de l'objet courant à true si le personnage a correctement été créé
- refaites des tests sur votre objet perso1 (en l'affichant simplement dans la console) pour constater que la propriété existe est bien à true ou à false selon que notre personnage est correctement créé ou pas

Partie 5 :

- via votre objet perso1, donnez la valeur 50 aux propriétés vie, attaque et defense
- constatez que ça marche via la console, en affichant simplement votre objet

- créez une méthode info qui affiche UN console log affichant le nom, la vie, l'attaque et la défense du personnage
- utilisez votre méthode info sur votre perso1 pour afficher les infos

Partie 6 :

- créez un deuxième personnage perso2. N'oubliez pas de lui donner un nom
- assignez-lui des valeurs numériques pour vie, attaque et defense
- affichez ses infos

Partie 7 :

- créez une nouvelle méthode attaquer()
- donnez-lui un paramètre "defenseur"
- affichez un console.log "Nouvelle attaque de NOMATTAQUANT sur NOMDEFENSEUR !!"
- si l'attaque de l'attaquant est supérieure à la défense du défenseur : retirer 10 points de vie au défenseur
- si l'attaque de l'attaquant est égale à la défense du défenseur : retirer 5 points de vie au défenseur
- si l'attaque de l'attaquant est inférieure à la défense du défenseur : retirer 5 points de vie à l'attaquant
- à la fin de votre méthode attaquer (à l'intérieur), afficher les infos de l'attaquant et du défenseur via la méthode info()
- dans votre script, faites attaquer perso2 par perso1 en utilisant la méthode attaquer() avec le bon paramètre
- changer les valeurs d'initialisation de vos personnages pour générer plusieurs cas de figures, et constatez que ça marche

Partie 8 :

- dans votre méthode attaquer()
- si la vie de l'attaquant est inférieur ou égale à zéro, afficher un console.error "Le personnage NOM_ATTAQUANT est dead.", et mettez la propriété "existe" de l'attaquant à false
- si la vie du défenseur est inférieur ou égale à zéro, afficher un console.error "Le personnage NOM_DEFENSEUR est dead.", et mettez la propriété "existe" du défenseur à false

Partie 9 :

- dans votre classe, définissez une méthode statique nombreAleatoire, qui retourne un nombre aléatoire entre 20 et 100
- définissez les caractéristiques vie, attaque et défense de vos personnages en utilisant la méthode nombreAleatoire
- relancer le script plusieurs fois pour vérifier que ça marche, et que les deux personnages peuvent gagner chacun leur tour

Partie 10 :

- dans votre script d'initialisation, mettez tout en commentaire
- créez une constante nbrJoueur, initialisez-la à 2
- créez un tableau players initialisé à vide
- faites une boucle qui va :
    - demander de saisir un nom
    - créer un nouveau Personnage avec le nouveau nom
    - donner des valeurs aléatoires pour la vie, l'attaque et la défense
    - afficher les infos du personnage
    - insérer le personnage dans le tableau players

Partie 11 : (pour les vrais)

- dans votre script, faire une fonction run() qui va se faire affronter les joueurs présents dans le tableau player
- ce sera une fonction récursive
- il y aura pas mal de boucles
- afficher le gagnant à la fin
- changer votre constante nbrJoueur à 3 une fois que ça marche

=> Je ne vous ai pas détaillé cette partie, car c'est vraiment de l'algo "pur", et pas de la POO. Je serai là pour vous aiguiller si nécessaire. Prenez le temps d'y aller pas à pas pour faire une fonction qui marche "dans tous les cas de figures". N'oubliez pas que pour se battre, un personnage doit avoir de la vie, et donc "exister".

Visibilité, get & set, statique

Partie 1 :

- passer tous vos attributs (existe, nom, vie, attaque, défense) en visibilité privé
- rajouter un underscore devant le nom de chaque attribut
- faire un getter et un setter pour chacun de vos attributs. Les getters retournent la valeur de la propriété de l'objet courant. Les setters assignent une nouvelle valeur à la propriété de l'objet courant.
- vérifier que votre script fonctionne comme avant

Partie 2 :

- faites une recherche sur Internet, et rajoutez-moi des couleurs dans vos console.log (couleurs logiques : mort = rouge, victoire = vert, etc.....)
- enlever tous les console.warn et console.error, et ne mettez que des console.log avec des couleurs

Partie 3 :

- passer la méthode nombreAleatoire() en static
- déplacer les instructions qui donnent de la vie, de l'attaque, et de la défense à vos personnages à l'intérieur du constructeur au même endroit où l'on définit le nom et "existe"
- modifier toute cette partie pour que votre classe refonctionne normalement

Partie 4 :

- dans votre méthode attaquer, supprimer les if (les deux) qui vérifient si les personnages sont morts (if(vie <= 0))
- dans votre setter vie(), reporter la règle : si vie <= 0, alors existe = false et vie = 0 et on affiche le message de la mort d'un perso (console.log)
- vérifier que votre script fonctionne toujours
- constater que les vies ne sont plus négatives dans les messages d'information
- constater qu'on a plus besoin de vérifier la vie de l'attaque ET la vie du défenseur, cela se fait automatiquement dès qu'on touche à la propriété vie d'un objet. Pratique

Partie 5 :

Pas le temps d'écrire l'énoncé. Créez-moi une classe Match dans le même fichier que personnage. Mettez tout votre script d'initialisation (demander le nom des perso, fonction run, etc...) à l'intérieur de la classe Match. Débrouillez-vous pour que tout marche correctement. Le code sera sensiblement le même, à quelques différences près. L'idée, c'est d'avoir une classe qui gère le match, et une classe qui gère les personnages. Appelez-moi si vous ne comprenez pas ce que je vous demande.

Classe Match

Partie 1 :

Dans la classe Match

- vérifier que les joueurs ne peuvent partager un même nom (faire ça uniquement dans la classe Match). Pour cela, créer une méthode verifieNomPersonnage dans votre classe Match, qui se chargera de dire si un nom est déjà existant parmi les joueurs enregistrés
- rajouter la notion de round dans votre classe Match. L'idée, c'est que l'on affiche "Round 1, round 2, round 3" à chaque nouveau combat (à chaque nouveau tour de jeu)
- créer un attribut winner. Créer une méthode win(). En cas de victoire, mettre l'objet du personnage qui a gagné dans la propriété winner, puis appeler la méthode win
- dans la méthode win, afficher le nom du gagnant, puis afficher ses informations via la méthode info()

Héritage

Partie 1 : classe CRS

- créer une classe CRS, qui hérite de Personnage
- créer un constructeur qui prends le paramètre nom
- dans votre constructeur, appelez le constructeur de la classe personnage, avec le paramètre nom
- donner +5 à l'attaque
- donner +5 à la défense
- donner -5 à la vie
- dans votre classe Personnage, rajouter un attribut type, avec la visibilité protected et une chaîne vide par défaut
- dans votre constructeur CRS, donner la valeur type de votre objet courant à "CRS"
- dans votre méthode info(), rajouter le type du personnage à coté du nom. Faites de même avec le console.log de la méthode attaquer(), et celui du setter vie().
- dans votre constructeur de Match, remplacer "new Personnage()" par "new CRS()" afin de faire des tests

Partie 2 :

- modifier votre setter attaque, afin que la valeur de attaque ne puisse pas dépasser les 100 et ne puisse pas être inférieure à 20
- modifier votre setter defense, afin que la valeur de defense ne puisse pas dépasser les 100 et ne puisse pas être inférieure à 20
- modifier votre setter vie, afin que la valeur de vie ne puisse pas dépasser les 100

Partie 3 : classe Gilet Jaune

- créer une classe GJ qui hérite de Personnage
- de la même manière qu'avec les CRS donner -5 à l'attaque, -5 à la défense, +5 à la vie, et "GJ" au type
- modifier le constructeur Match avec "new GJ" afin de faire des tests

Partie 4 : classe Match

- ne lancer la partie que si le nombre de joueurs est pair
- créer un attribut nextType, et donner lui la valeur 'crs' par défaut
- faire en sorte que l'initialisation des personnages crée un crs et un gilet jaune
- modifier votre méthode run(), afin qu'un CRS ne puisse attaquer qu'un GJ, et qu'un GJ ne puisse attaquer qu'un CRS (si a.type est différent de de b.type)

Partie 5 : classe CRS

- créer une méthode attaquer qui prends un paramètre gj
- à l'intérieur, appeler la méthode attaquer() de la classe personnage, avec le paramètre gj
- juste après, toujours dans votre méthode attaquer de la classe crs, créer une variable chance qui prendra un nombre au hasard entre 1 et 10
- dans la classe CRS, créer une méthode fumigène et une méthode canon à eau
- pour l'instant, afficher simplement dans des console log "attaque fumigène", et "attaque canon à eau", à l'intérieur
- dans votre méthode attaquer de la classe CRS : si chance est compris entre 7 et 9, exécuter la méthode fumigène. Si chance est ègale à 10, exécuter la méthode canon à eau. (60% de chance de ne rien faire, 30% de chance de fumigène, 10% de chance de canon à eau)
- vérifier que tout fonctionne à l'aide des console.log (chance, fumigènes, canon à eau)
- rajouter le paramètre gj dans vos appels de coup spéciaux (fumigènes et canon à eau)
- faites de même pour les méthodes elles-mêmes
- donner -5 vie au gj dans la méthode fumigène()
- donner -10 vie au gj dans la méthode canonAEau()
- reporter l'affichage des informations présentes dans la méthode attaquer de la classe Personnage à la fin de votre méthode attaquer de la classe CRS
- mettez des couleurs sympas au console.log de fumigene() et canonAEau()
- tester votre code. Statistiquement, pour l'instant les CRS l'emportent le plus souvent (car on a pas encore fait les coups spéciaux des GJ)

Partie 6 :

- Faites les coups spéciaux des GJ avec la même logique que les CRS
- faire une méthode caillassage qui enlève -5 point au crs
- faire une méthode mouvement de foule qui enlève -15 point au crs

Partie 7 : refactoring

- faites des beaux console.log pour chaque attaque afin que l'évolution des personnages et des actions soit bien lisible
- rajouter un console log dans le cas où il n'y a aucun coup spécial déclenché
- rajouter des console.log("\n") afin de faire des retours chariots (saut de ligne) dans votre console
- ajustez les modifications des propriétés dans les constructeurs crs et gj afin d'essayer d'avoir un résultat équitable (les gj gagnent autant de fois que les crs)
- Faites en sorte que le nom se génère automatiquement avec (type + i) (ex : GJ 1, CRS 1, GJ 2, CRS 2, etc...)

Voici un exemple avec ma console :

```
Nouvelle attaque de mp (GJ) sur ju (CRS) !!
mp (GJ) => Vie 62 | Attaque : 78 | Défense : 21
ju (CRS) => Vie 40 | Attaque : 57 | Défense : 41
CAILLASSAGE !!!
mp (GJ) => Vie 62 | Attaque : 78 | Défense : 21
ju (CRS) => Vie 25 | Attaque : 57 | Défense : 41


Nouvelle attaque de ju (CRS) sur ki (GJ) !!
ju (CRS) => Vie 25 | Attaque : 57 | Défense : 41
ki (GJ) => Vie 19 | Attaque : 59 | Défense : 39
ATTAQUE NORMALE (pas de coup spéciaux)
ju (CRS) => Vie 25 | Attaque : 57 | Défense : 41
ki (GJ) => Vie 9 | Attaque : 59 | Défense : 39

Nouvelle attaque de ju (CRS) sur mp (GJ) !!
ju (CRS) => Vie 25 | Attaque : 57 | Défense : 41
mp (GJ) => Vie 62 | Attaque : 78 | Défense : 21
CANON A EAU !!!
ju (CRS) => Vie 25 | Attaque : 57 | Défense : 41
mp (GJ) => Vie 42 | Attaque : 78 | Défense : 21
```

Interface graphique

Consigne : faire une maquette papier avant. Vous n'êtes pas obligé de faire la même interface que moi.

Essayer de faire une interface graphique sympa. Attention, c'est plus compliqué que ça n'y parait. Je n'ai pas eu le temps de finir, voici un exemple :