# JAVASCRIPT - Exercice de Rattrapage

## Objectif

Revoir et s'exercer aux notions suivantes :

- Variables
- Fonctions
- Tableau
- Classe
- Objet
- Boucle
- Condition

## Fichiers fournis

Voici la liste des fichiers fournis :

- index.html (avec le html minimum pour exploiter le fichier script.js)
- script.js (possédant 2 tableaux conservés dans des constantes)

L'exercice sera fait entièrement au sein du fichier script.js.

## Contexte

Nous souhaitons créer les bases d'un jeu de combat de carte.

2 joueurs s'affronteront avec un deck constitué de 5 cartes générées aléatoirement. Le Vainqueur est le joueur qui remportera 3 duels.

Dans cet exercice, le jeu ne sera pas entièrement codé. Il sera uniquement codé la création aléatoire des cartes, et un joueur pourra voir les cartes de son deck.

### Conseil : Ne pas oublier de tester régulièrement le code écrit

**Auteur :** Yoann DEPRIESTER
**Relu, validé & visé par :**
☒ Jérôme CHRETIENNE
☒ Sophie POULAKOS
☒ Mathieu PARIS

**Date création :** 21-11-2023
**Date révision :** 21-11-2023

# JAVASCRIPT - Exercice de Rattrapage

## PARTIE 1

### Création d'une fonction utilitaire randomX()

1) Créer une fonction. Elle a pour nom randomX et prendra en paramètre une variable nommé number.

2) La fonction randomX va calculer un nombre aléatoire compris entre 0 et number, avec la formule : Math.random() * number

3) La fonction randomX va ensuite arrondir le nombre aléatoire généré précédemment, avec la fonction : Math.floor(). L'ensemble donne un nombre allant de 0 à number-1.

4) La fonction randomX va enfin retourner ce nombre, grâce à la commande return

### Création de la classe Card{}

Nous allons créer un class qui permettra de créer les cartes d'un deck, de montrer le profil d'une carte (fonction showCard()) et d'attaque une fois une carte ciblée (fonction attack(card)).

1) Créer la classe Card. Bien attention à ce que le nom commence bien par une majuscule (convention d'écriture).

2) Le constructeur de la classe Card aura 5 paramètres : name, power, thoughness, type, et strength.

3) Dans le constructeur, attribuer chaque paramètre à chaque propriété, selon l'exemple :

   this.name = name ;
   this.power = power ;
   …

4) Au sein de la classe Card, créer une fonction qui a pour nom showCard (attention, il ne faut pas utiliser le mot clé function, il y a juste besoin de mettre le nom de fonction, ses parenthèse et ses accolade).
5) Dans la fonction showCard(), faire qu'on affiche dans la console le profil d'un carte sous la forme :

```
-----------------------------------------------
    Nom : ${this.name}
    Puissance : ${this.power}
    Endurance : ${this.thoughness}
    Type : ${this.type}
    Fort contre : ${this.strength}
-----------------------------------------------
```

# JAVASCRIPT - Exercice de Rattrapage

6) Au sein de la classe Card, créer une fonction qui a pour nom attack(). Cette fonction aura un paramètre appelé card (son but sera de passer à la fonction un objet Card adverse qui combattra contre l'objet Card qui utilise la fonction attack()).

7) Dans la fonction attack(card), déclarer la variable dammage avec let (sans lui attribuer de valeur).

8) Dans la fonction attack(card), déclarer la variable attackPower avec let, et lui attribuer la valeur suivante : `this.power + randomX(6)` `(on fait la somme de la puissance de la carte qui utilise la fonction attack() et d'un nombre aléatoire allant de 0 à 5)`

9) Dans la fonction attack(card), utiliser une condition if, avec pour condition : si la force da carte attaquante (this.strength) est égale ( == ) au type de la carte adverse (card.type), alors on multiplie la variable attackPower par 1.5 et on affiche dans la console que la carte attaquante (this.name) a l'avantage contre la carte adverse (card.name).

10) Dans la fonction attack(card), utiliser une condition if, avec pour condition : si la variable attackPower est supérieur ( > ) à l'endurance de la carte adverse (card.thougness), alors on attribue à la variable dammage la valeur `randomX(6)+1` `(cela donne un nombre aléatoire allant de 1 à 6). SINON (else) on attribue à la variable dammage la valeur 1.`

11) Enfin, dans la fonction attack(card), soustraire à l'endurance de la carte adverse (card.thoughness) les dégâts subits (variable dammage). Puis afficher dans la console une phrase indiquant que la carte attaquante (this.name) inflige un nombre de point de dégâts égale à la variable dammage, à la carter adverse (card.name).

## Création de la classe Player{}

Nous allons créer une class qui permettra de définir un joueur, de lui créer un deck avec des cartes aléatoires, et de montrer son deck.

1) Créer la classe Player.

2) Le constructeur n'aura qu'un seul paramètre name.

3) Dans le constructeur, attribuer à la propriété this.name la valeur name, à la propriété this.deck un tableau vide ( [] ), et à la propriété this.score la valeur 0.

4) Créer une fonction createDeck, sans aucun paramètre. Elle va servir à créer un deck de 5 cartes générées aléatoirement.

5) Dans la fonction createDeck(), mettre ne place une boucle FOR classique. On initialise une variable i à 0 pour servir de compteur. La condition de sortie est i < 5 (pour faire 5 tours de boucles). Et à chaque tour on va incrémenter i de 1.

# JAVASCRIPT - Exercice de Rattrapage

6) Dans la boucle, initialiser une constante card. Lui attribuer dans le même temps un nouvel objet Card (new Card).

7) Pour générer le nouvel objet Card de manière aléatoire, on va utiliser notre fonction randomX() au sein des paramètres du nouvel objet lors de sa création. Mettre dans les paramètres et dans l'ordre :

   - un nom de carte piocher dans le tableau cardName. Pour cela, on appelle le tableau cardName, et on lui passe en index un nombre aléatoire allant de 0 à la taille du tableau -1 ( randomX(cardName.length) )

   - un puissance allant de 1 à 10 ( randomX(10)+1 )

   - une endurance allant de 1 à 10

   - un type de carte piocher dans le tableau cardType. On utilisera la même manière de faire que pour le nom de carte

   - un type contre lequel la carte est forte, piocher dans le tableau cardType

8) Toujours dans la boucle, afficher dans la console qu'un nouveau Challenger arrive. Puis montrer le profil de la nouvelle carte en appelant ( card.showCard() ). Et enfin, ajouter la nouvel carte au deck du joueur en utilisant la fonction push(card) sur this.deck

9) On va créer la fonction showDeck() qui va permettre d'afficher le profil de chaque carte du deck du joueur. Créer la fonction showDeck.

10) Dans cette fonction, afficher dans la console qui dira que l'on montre les cartes du deck du joueur en question ( this.name ).

11) Puis, mettre en place une boucle FOR…OF. Elle initialisera une variable card ( qui stockera à chaque tour de boucle une carte du deck), et la boucle s'exécutera sur le deck du joueur concerné ( this.deck ).

12) Dans cette boucle, on va appeler la fonction showCard() de la carte pour afficher son profil ( card.showCard() ).

# JAVASCRIPT - Exercice de Rattrapage

## Tester un Duel de Carte

On va maintenant tester notre code au sein du scénario suivant : 2 joueurs adverses et leur deck sont créés. On se fait affronter leur première carte sur plusieurs tours de jeu. Les cartes s'attaquent à tour de rôle. La première carte attaquante est tirée au sort, puis ce sera au tour de la carte adverse et ainsi de suite. Le duel se termine dès qu'une carte a réduit à zéro l'endurance de sa carte adverse.

1) Créer une constante jace à laquelle on assignera un nouvel objet Player (new Player) qui aura en paramètre le nom « Jace »

2) Créer une constante liliana à laquelle on assignera un nouvel objet Player qui aura pour nom « Liliana »

3) Créer le deck de Jace ( jace.createDeck() ), ainsi que le deck de Liliana.

4) Afficher le deck de Jace ( jace.showDeck() ), ainsi que le deck de Liliana.

5) On va lancer le duel de carte. Tout d'abord on va mettre en place un compteur de tour. Il faut donc initialiser une variable round en lui assignant la valeur 1.

6) Puis on doit tirer au sort la carte qui attaquera en première au premier tour. On va tirer à pile ou face grâce à la fonction randomX(2). Cependant, on sait que l'initiative va changer de camp à chaque tour de jeu. On est donc face à un phénomène cyclique. De fait, l'utilisation des modulos va nous être utile pour faire passer l'initiative d'une carte à l'autre

   En effet, le numéro du tour modulo 2 va nous donner comme résultat soit 0 (en cas de nombre pair), soit 1 (en cas de nombre impair). On pourra alors se servir de ce résultat en l'utilisant comme index dans un tableau contenant nos cartes. Lorsque l'index sera de 0 (numéro d'un tour pair % 2) ce sera la première carte qui va attaquer. Lorsque l'index sera 1 (numéro d'un tour impair % 2) ce sera la seconde carte qui va attaquer.

   Qui dit phénomène cyclique, dit aussi boucle. On devra donc en mettre une en place pour que les cartes s'attaquent à tour de rôle. On sortira de la boucle lorsque l'endurance de l'une des cartes atteint 0.

   Ceci étant établit, continuons le code :

7) Mettre en place le tableau de carte. Pour cela, initialiser une constante cardTab et lui assigner dans le même temps un tableau vide. Puis, mettre en place une condition if :

   - SI randomX(2) est égal à 0, alors on push() dans cardTab, dans l'ordre, la première carte de Jace ( jace.deck[0] ), puis la première carte de Liliana. ( note : on se trouve dans le cas où la carte de Liliana attaque en première, car le tour 1 % 2 donne un index de 1, ce qui est l'index de la carte de Liliana dans le tableau cardTab )

# JAVASCRIPT - Exercice de Rattrapage

- SINON on push() les cartes dans le sens inverse. ( note : on se trouve dans le cas où la carte de Jace attaque en première, en suivant la même logique que précédemment)

8) Afficher dans la console le nom de la carte qui attaque en première ( cardTab[1].name ).

9) Mettre en place une boucle qui va s'arrêter lorsque l'endurance d'une des cartes aura atteint 0. En d'autres termes, la boucle continue tant que l'endurance des 2 cartes est strictement supérieur à 0 ( cardTab[0].thoughness > 0 && cardTab[1].thougness > 0 ).

10) Au sein de la boucle, afficher dans la console le nom de la carte qui attaque ( cardTab[round %2].name ). Puis appeler sa fonction attack() en lui passant en paramètre la carte adverse ( cardTab[(round+1) % 2] ). Enfin, incrémenter le compteur round de 1.

11) Après la boucle précédente, afficher dans la console le nom de la carte qui a remporté le duel ( cardTab[(round – 1) % 2]  -> on a décrémenté le compteur round de 1 car il a été incrémenter à la fin de la boucle après que la carte attaquante remporte le duel).

## PARTIE 2

Dans la suite de l'exercice, il sera donné volontairement moins d'information.

### Mise en place d'un cimetière

1) Ajouter une propriété tomb, qui servira de cimetière à chaque joueur.

2) Créer une fonction qui permette de faire passer une carte du deck d'un joueur vers son cimetière (se renseigner sur la fonction splice qui peut être utile).

3) Créer une fonction qui affiche le cimetière d'un joueur.

### Mise en place d'un Match

1) Créer une class qui servira à lancer un match entre 2 joueurs. Cette classe a besoin d'une propriété pour le joueur 1, d'une propriété pour le joueur 2, et d'un nombre de manche à gagner pour remporter le match (avec 5 cartes dans chaque deck, il est conseillé de plafonner le score à 3).

2) Créer une fonction qui permette à 2 cartes de s'affronter, et qui ajuste le score de chaque joueur en conséquent.

3) Créer une fonction qui vérifie le score de chaque joueur et qui détermine le vainqueur du match.

# JAVASCRIPT - Exercice de Rattrapage

4) Créer une fonction qui vérifie le deck de chaque joueur et qui détermine le vainqueur du match. Pour cette condition de victoire, un joueur perd la partie lorsque toutes ses cartes sont aller au cimetière.

5) Lancer un Match entre 2 joueurs et afficher le vainqueur, avec comme condition de victoire le fait d'atteindre le score défini pour gagner. Chaque joueur pourra voir le deck adverse et choisir une carte de son deck avant chaque duel de carte.

6) Lancer un Match entre 2 joueurs et afficher le vainque, avec comme condition de victoire le fait que le joueur advers ne possède plus de carte dans son deck. Chaque joueur pourra voir le deck adverse et choisir une carte de son deck avant chaque duel de carte.

## PARTIE 3 – Bonus : Manipulation du DOM

Cette partie peut se révéler un peu plus complexe et demandera peut-être à modifier / adapter le script déjà écrit.

1) Créer une interface en HTML-CSS, et rendre le jeu jouable via cette interface (avec affichage d'information).