**Projet 5 : Créez un jeu de plateau tour par tour en JS**

Je m’appelle Daryl LIVIO étudiant chez **OpenClassroom** en tant que développeur d’application front end en et mon rôle pour ce projet était de créer un jeu en JavaScript dans lequel 2 joueurs s’affrontent chacun leur tour.

**Pour une meilleure compréhension du projet, ma présentation se fera en trois parties.**

**La première partie**présentera la bibliothèque JQuery que j’ai étudié pour réaliser cette application.

**La deuxième partie,** nous pourrons tester les différentes situations du jeu.

**Enfin, la troisième partie** présentera le code sur **Virtual studio code**

**Qu’est-ce que JQuery ?**

JQuery est une bibliothèque qui permet de manipuler **les éléments HTML,**  
**les mises en forme en** CSS et **le JavaScript** en utilisant des instructions simples. La première version est lancée **en janvier 2006 par John Resig.**

**La devise** de jQuery est **« *Write less, do more* »,** ce qui signifie « Écrivez moins pour faire plus ».

JQuery permet d’écrire moins de code et, en écrivant moins de code, **nous faisons moins d’erreur**.

Pour utiliser JQuery, nous pouvons utiliser au choix **l'instruction jQuery() ou son alias $().**

**Présentation du jeu tour par tour**

**Je vais maintenant vous partager mon écran afin de vous présenter le jeu tour par tour**. **(Cliquez sur le bouton règle du jeu)**

Il fallait créer le jeu en 3 étapes :

**Étape 1 : génération de la carte**

* Générer aléatoirement la carte du jeu ou chaque case peut être **soit vide ou inaccessible. (ALLER VOIR)**

Générer 4 armes qui disposent d’un visuel et placé aléatoirement.  
L’arme par défaut qui équipe les joueurs doit infliger 10 points de dégâts.. **(ALLER VOIR)**

* **Le placement des deux joueurs** est lui aussi aléatoire. Ils ne doivent pas se toucher. **(ALLER VOIR)**

Pour tester le programme, je vais rafraichir plusieurs fois la page du jeu pour tester le respect de ces consignes.

### **Étape 2 : les mouvements**

A chaque tour, un joueur peut se déplacer d’une à trois cases (horizontalement ou verticalement) avant de terminer son tour. Il ne peut évidemment pas passer à travers un obstacle. **(ALLER VERIFIER)**

Si un joueur passe sur une case contenant une arme, il laisse son arme actuelle sur place et la remplace par la nouvelle. **(ALLER VERIFIER)**

### **Étape 3 : le combat !**

Si les joueurs se croisent sur des cases adjacentes (horizontalement ou verticalement), un combat à mort s’engage.

* Chacun attaque à son tour
* Les dégâts infligés dépendent de l’arme possédée par le joueur
* Le joueur peut choisir d’attaquer ou de se défendre contre le prochain coup
* Lorsque le joueur se défend, il encaisse 50% de dégâts en moins qu’en temps normal
* Dès que les points de vie d’un joueur (initialement à 100)
* Tombent à 0, celui-ci a perdu. Un message s’affiche et la partie est terminé

**La programmation dans Visual studio code**

**Nous allons maintenant passer à la troisième partie le code dans vscode :**

Pour la programmation, j’ai opté pour l’approche orienté objet.  
La POO considère l’application comme un **ensemble d'objets possédant des caractéristiques**.

J’ai donc réaliser un diagramme de classe pour commencer le projet dans les meilleures conditions.

**Diagramme de classe**

Le diagramme de classe permet de fournir une **représentation abstraite des objets de l’application** qui vont interagir ensemble les uns avec les autres pour réaliser les différents cas d'utilisation de l’application.

Il m’a permis d’obtenir les classes qui sont selon moi sont nécessaire au bon fonctionnement de l’application :

* **La classe Grille, la classe Joueurs, la classe Arme, La classe Case, La classe Jeu.**

Vous pouvez voir leurs méthodes et leurs attributs.

Ces différentes classes m’ont permis de séparer mon code en différent fichier JavaScript afin qu’il soit plus cohérent et optimisé.

(VOIR LE CODE)

**Le code est assez long donc je vais essayer présenter les éléments qui sont selon moi les plus importants et les plus pertinents pour ce programme**

J’ai essayé de faire une classe pour chaque élément de la grille. C’est-à-dire pour la classe ARME un **fichierJS arme**, pour la classe Joueur un **ficherJS joueur** etc.. Le dernier fichier.js **Jeu lui est un peu différent car il me permet charger toute l’application grâce à la fonction** readyfunction().

**ETAPE 1 : Génération de la carte**

Le fichier Grille.js est le plus important car c’est le moteur de l’application c’est lui qui met en relation tous les autres fichiers.JS. il me permet de **générer la carte, les armes, les obstacles et les joueurs.**

J’ai créé **la classe grille** qui comporte des propriétés et des méthodes.

* **Ma première action était de générer la grille.**   
  C’est à ce moment-là que j’ai vu la puissance de **JQuery** et toute son utilité. Au lieu de faire une centaine de balise **div** dans mon HTML pour générer les 100 cases de 10 cases de long et 10 de large. Avec JQUERY, J’ai pu avoir le même résultat en trois lignes de code.  
  J’ai utilisé 2 boucles **for pour** **x et pour y et j’ai récupérer l’élément du DOM (grilledujeu) via JQUERY** pour ajouter dans l’html les **100** **div** dynamiquement grâce à la propriété **append de jQuery** qui permet d’insérer du contenu à la fin d’un élément. Puis en écrivant du html dans une fichier **js** à l’aide **des backquotes**.
* **Pour générer les 15 obstacles,** j’ai procédé de la même façon en utilisant une boucle **for**.  
  Dedans, j’ai généré deux coordonnées aléatoires. Les coordonnées sont aléatoire grâce à la méthode **nbAleatoire** ou **Math.floor renvoie** **le plus grand entier qui est inférieur ou égal à un nombre** couplé à **Math.random** renvoie un nombre compris dans l'intervalle [0, 1 que je peux ensuite multiplier.   
  Puis à l’aide de JQuery j’ajoute la classe obstacle grâce à la **propriété addclass** (voir obstacle dans le css) dans les cases générées aléatoirement
* **Pour générer les armes,** j’ai dû faire **un nouveau fichier Arme.js.**J’ai créé une **classe arme** : dans le constructor, il y a le nom de l’arme et les dégâts qu’elle peut causer. Ensuite, il y a les différentes instances de la classe Arme (citez les armes)**.**
* **On retourne dans le fichier grille.js dans la méthode générer Arme**J’ai fait une boucle **for in** avec cette fois une condition **pour le point américain qui lui ne sera pas sur la grille au moment de générer la carte**. C’est l’arme par défaut qui fait 10 points de dégât et j’ajoute la classe **arme.nom.**
* **Pour générer les 2 joueurs,** j’ai é créer une classe joueur dans un autre fichier joueurs.js**.**La classe dispose dans le constructor : **nom, sante, arme** etc… et une méthode (la description du joueur).  
  Pour éviter que joueurs soient générés côte à côte, j’ai employé une méthode **coordonnesCoteAcote** qui stipule que si les joueurs sont générés cote a cote on régénère une coordonnée pour le **joueur2**

**ETAPE 2 : Les mouvements**

Pour l’étape 2 : les mouvements, j’utilise l’évènement **(onkeydown)** qui écoute les flèches directionnelles du clavier). (Haut, bas, droite et gauche.)

J’ai simplement créé une variable **compteur qui est égale à zéro** dans **le constructor** et une méthode qui permet d’incrémenter le compteur à chaque fois qu’un joueur se déplace d’une à trois cases. (Voir deplacementAutoriseeMax3cases)

Pour l’échange d’armes, j’ai créé une méthode (**echangerLesArmes)** avec pour condition :

Si la case comprenant une arme n’est pas nulle c’est-à-dire qu’il y a un des joueurs dessus. L’arme temporaire qui est au sol devient l’arme équipée par le joueur.

Ensuite Avec JQuery on retire l’arme équipée du joueur et on lui assigne la nouvelle arme avec les méthodes **remove** et **addclass**

**ETAPE 3 : LE COMBAT**

Pour l’étape 3 : j’ai réutilisé, la méthode cote à cote afin de détecter les deux joueurs et passer en mode combat avec la méthode **passageEnModeCombat().**

Je supprime la méthode **déplacementDuJoueur(). Les joueurs ne peuvent plus se déplacer** et je change de joueur avec la propriété **changementJoueur()**. Pour que l’autre joueur puisse commencer

La méthode **passageModecombat** appelle aussi une fonction anonyme qui envoie une alerte et montre les boutons attaque et défense.

Dans le fichier joueur.js nous avons les deux propriétés **attaque et défense  
  
Pour l’attaque et la défense,** j’ai créé une méthode **joueurEstblessé** que si la **défense est égale à true** cela signifie que le joueur est en mode défense les dégâts de l’arme équipée est divisé par deux.

En revanche si le joueur n’est pas en mode défense, Les dégât de l’arme sont totale.

**Avec les méthodes sedefend, et attaque** e je mets à jour les infos du combat en récupérant #infosAttaque et je crée un élément p pour faire apparaitre dynamiquement les infos du combat.

Pour la **fin de jeu**, si la sante d’un des joueurs est à 0 LE JOUEUR Adverse gagne la partie à l’aide de la fonction **finDejeu**() qui envoie un message de fin de jeu et propose une nouvelle partie.  
Tout l’application est chargé dans le fichierJS jeu grâce à la fonction readyFunction()

**Boucle for in :** permet d'itérer sur l'ensemble des propriétés d'un objet.

**Boucle for of :** permet d'itérer sur l'ensemble des valeurs d'un objet.

**continue :** ne stoppe pas entièrement l'exécution de la boucle.

**Do while :** crée une boucle qui exécute une instruction jusqu'à ce qu'une condition de test ne soit plus vérifiée

**Index Of :** renvoie le premier indice pour lequel on trouve un élément donné dans un tableau.