

# Projet Web 2023 : Monster



Ce projet est à faire individuellement ou en binôme. Indiquer absolument votre nom (ou vos deux noms) en en-tête du fichier JavaScript. Les technologies attendues sont HTML, CSS et JavaScript; l'usage de jQuery est autorisé mais non indispensable. L'usage de copié-collé, plagiat, recours (même partiel) à des intelligences artificielles est proscrit.

L'objectif du projet est de programmer un simulateur de la vie d'un pauvre monstre qui passe son temps à dormir, courir, combattre, travailler et se nourrir. L'internaute doit surveiller son état et lui faire faire des actions qui lui font gagner ou perdre des points de vie ou de l'argent afin de le maintenir en vie.

## ⚠ DEUX versions à rendre

Votre archive finale devra être constituée de **deux** répertoires :

1. un répertoire **MonsterV1** correspondant aux quatre premières parties, et contenant exclusivement `monster.html`, `css/monster.css` (ces deux fichiers doivent être fournis sans modification), ainsi que `js/monster.js` (votre fichier) ;
2. un répertoire **MonsterV2** correspondant à la cinquième partie (beaucoup plus libre), et contenant ce que vous voulez. Ce second répertoire devra bien sûr contenir tous les fichiers (HTML, CSS, JS, images...) nécessaires à la bonne exécution de votre programme dans sa version améliorée.

## ⚠ JavaScript pur

Aucune modification des fichiers CSS et HTML fournis n'est autorisée au cours des quatre premières parties.



## EXERCICE

**Exercice 1** (Partie 1 : affichage du monstre).





**Question 1 :** Récupérer l'archive monster sous Moodle. Vérifiez que son contenu s'affiche correctement.

Créer le fichier `js/monster.js`. Dans ce fichier, créer les variables et fonctions de base relatives au monstre :


1. des variables globales décrivant l'état du monstre :

- `name` : le nom du monstre,
  - `life` : nombre de points de vie du monstre,
  - `money` : l'argent du monstre,
  - `awake` : booléen indiquant si le monstre est réveillé ou non (à initialiser à `true`);
2. la fonction `initMonstre(nom, vie, argent)` qui initialise l'état du monstre avec les valeurs reçues en paramètres;
  3. la fonction `afficheMonstre()` qui affiche toutes les propriétés du monstre sur une seule ligne dans la console.

 **Question 2 :** Faire un programme principal réalisant un appel à `initMonstre()` et à `afficheMonstre()` : une ligne dans la console doit maintenant s'afficher au rechargement de la page.

 **Question 3 :** Créer les fonctions de déroulement de l'application, et en particulier la déclaration des auditeurs associés aux événements produits par les actions de l'internaute sur l'interface. Cela comprend :

- la définition et l'initialisation de variables globales permettant de stocker les différents objets du DOM recevant des événements (une variable par bouton donc — certaines de ces variables ne seront cependant utilisées que dans les questions à venir, mais au moins elles seront toutes définies);
- la fonction `go()` qui initialise le monstre en appelant l'action `initMonstre()` puis enregistre la fonction `afficheMonstre()` en tant qu'auditeur de l'évènement « clic » sur le bouton `Show` correspondant.


 **Question 4 :** Enregistrer la fonction `go()` comme auditeur de l'évènement `load` de la fenêtre afin de lancer l'application. Cette déclaration devient ainsi l'unique ligne de votre nouveau programme principal (en dehors des définitions de variables globales). Vérifier que le bouton `Show` est bien fonctionnel.

■



## EXERCICE

**Exercice 2** (Partie 2 : affichage et log).

 **Question 1 :** Définir la fonction `logBoite(message)` qui permet d'ajouter un message dans la boîte `#actionbox` de l'interface (il est conseillé là aussi de définir et initialiser une variable globale qui garde le nœud du DOM en mémoire). Elle le fait en insérant un nouveau `<p>` comme premier fils, afin de décaler les anciens messages *vers le bas*.

 **Question 2 :** Améliorer la fonction `afficheMonstre()` afin qu'elle affiche non seulement un message dans la console (comme précédemment), mais en plus ajoute un message dans la boîte `#actionbox`, grâce à votre fonction `logBoite(message)`. Tester le bon affichage en cliquant sur le bouton `Show`.


 **Question 3 :** Définir une fonction `updateStatus()` qui affiche l'état du monstre dans la liste d'identifiant `statut` de l'interface (là aussi une nouvelle variable globale pointant vers le nœud du DOM est à définir); cette action doit remplacer le contenu existant.

■



## EXERCICE


**Exercice 3** (Partie 3 : le monstre entre en action).


 **Question 1 :** Compléter les actions que peut réaliser le monstre en implémentant les fonctions `courir()`, `se battre()`, `travailler()` et `manger()`, puis associer (au bon endroit) ces actions aux boutons correspondants :

- pour chaque méthode, le monstre doit être vivant, réveillé et disposer de suffisamment de points de vie ou d'argent pour l'action désirée;
- chaque méthode doit afficher un message dans la boîte `#actionbox` pour expliquer ce qu'il se passe (que l'action entreprise se réalise ou non — il faut dans ce cas en indiquer la raison), ainsi que le nouvel état du monstre dans la barre de statut (à sous-traiter à la bonne fonction donc).

Règles de jeu pour les actions :

- courir : perte de 1 point de vie;
- se battre : perte de 3 points de vie;
- travailler : perte de 1 point de vie et gain de 2 unités d'argent;
- manger : perte de 3 unités d'argent et gain de 2 points de vie.

 **Question 2 :** Construire la fonction `dormir()` qui endort le monstre et programme son réveil 5 secondes plus tard, en utilisant un temporisateur (*timer* : voir la fonction `setTimeout` dans la documentation JavaScript). Vérifiez que, lorsque le monstre dort, il ne peut pas courir, manger ni se battre. Le monstre gagne 1 point de vie à son réveil. Bien sûr, l'état affiché du monstre doit être affiché lorsqu'il s'endort et lorsqu'il se réveille.


 **Question 3 :** Construire une fonction `actionauhasard()` qui exécute une de ses actions au hasard (la fonction `Math.random()` sera sûrement utile). Indication : penser à construire un tableau d'auditeurs.


Au chargement de la page, mettre en place une exécution toutes les 7 secondes (définir une constante pour ce temps, en tête de votre programme JavaScript, pour qu'elle puisse facilement être modifiée à des fins de tests) de la fonction `actionauhasard()`; voir pour cela la fonction `setInterval()` dans la documentation JavaScript.




## EXERCICE

**Exercice 4** (Partie 4 : vie et mort).

 **Question 1 :** Écrire deux fonctions gérant la mort du monstre et sa résurrection; les associer comme auditeur aux boutons `Kill` (pour tuer le monstre, s'il est vivant) et `New life` (pour le ressusciter, uniquement s'il est mort).

 **Question 2 :** Faire en sorte que les boutons non pertinents à un instant donné soient désactivés. Par exemple, le bouton `Kill` doit être désactivé quand le monstre est mort. Ou encore, le bouton `Eat` doit être désactivé si le monstre est mort ou endormi.

 **Question 3 :** Changer l'apparence de la boîte `#monster` : la couleur doit varier progressivement de rouge (si le monstre a 1 point de vie) à vert (si 10 points de vie) puis à bleu (si  $\geq 20$  points de vie).

En outre, la largeur de la bordure doit être proportionnelle au nombre d'unités d'argent. ■

### ⚠ Notation

Ces quatre premières parties seront notées sur 15 points (sur un total de 20). Par conséquent, les points restants seront attribués en fonction de votre créativité (fonctionnalités, ergonomie, graphisme, animations...), comme décrit dans la dernière partie ci-dessous.



## E X E R C I C E

### Exercice 5 (Partie 5 : créativité).



**Question 1 :** Améliorer votre projet de la façon qui vous plaît. La notation dépendra de votre créativité, de l'ajout de fonctionnalités, du graphisme, de l'ergonomie...

Toute modification des fichiers HTML et CSS fournis est *autorisée* dans cette partie.

Quelques idées ci-dessous (dont aucune n'est obligatoire!).

- Inclure de nouvelles actions, par exemple :
  - manger du tofu : perte de 2 unités d'argent et gain de 5 points de vie;
  - prendre le RER : perte de 3 points de vie avec 80 % de probabilité (quand le RER roule); perte de 10 points de vie avec 20 % de probabilité (le RER est en panne);
  - jouer à la roulette russe : 1 chance sur 6 de mourir;
  - aller à la scolarité : fonctionnement similaire à dormir mais en plus long (pendant 10 secondes), et perte d'un point de vie au bout.
- modifier/améliorer l'ergonomie, en changeant la disposition des boutons/logs/etc. dans la page;
- remplacer les boutons par des images; penser à l'opacité ou toute autre solution pour indiquer quand ils sont désactivés;
- remplacer la représentation du monstre par une ou plusieurs images;
- soigner les effets CSS : transparence, animation...;
- afficher de façon visuelle (ex : compte à rebours) un temporisateur lorsque le monstre dort, montrant le temps restant; idem pour le compte à rebours avant la prochaine action automatique aléatoire;
- proposer à l'internaute d'entrer le nom du monstre lors de la création et/ou sa résurrection;
- changer la représentation graphique de l'argent (par exemple un nombre de pièces proportionnel à la quantité d'argent du monstre);
- changer la représentation graphique des points de vie;
- proposer une première page permettant à l'internaute de choisir un monstre (nom et image prédéfinis);
- etc.



Source : <https://commons.wikimedia.org/wiki/File:Creative-Tail-Halloween-monster.svg>

Version : 13 février 2023