UNIVERSITÉ DE LORRAINE | IUT nancy Charlemagne

DUT Informatique
IHM

Javascript : DOM et événements
Projet Monster

L'objectif du TP 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'utilisateur 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.

Préliminaires

Récupérer l'archive `monster` sous Arche. Vérifiez que son contenu s'affiche correctement.

1 Étape 1

Dans le fichier javascript :

1. Créer les variables et fonctions de base relatives au monstre :
   a) des variables globales décrivant l'état du monstre :
      — `name` (ou « `nom` » si cela mène, par construction, à des effets de bord embêtants avec la propriété `name` présente dans les fonctions) : le nom du monstre,
      — `life` : nombre de points de vie du monstre,
      — `money` : l'argent du monstre,
      — `awake` : `true` ou `false`, indique si le monstre est réveillé ou non (à initialiser à `true`),
   b) la fonction `init(n, l, m)` qui initialise l'état du monstre avec les valeurs reçues en paramètres,
   c) la fonction `showme()` qui affiche toutes les propriétés du monstre sur une seule ligne dans une alerte.

   Faire un programme principal réalisant un appel à `init()` et à `showme()` : une alerte doit maintenant s'afficher au rechargement de la page.
2. Créer les fonctions de déroulement de l'application, et en particulier la déclaration des *handlers* associés aux événements produits par les actions de l'utilisateur sur l'interface. Cela comprend :
   a) 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),
   b) la fonction `go()` qui initialise le monstre en appelant l'action `init()` puis enregistre la fonction `showme()` en tant que *handler* de l'événement `click` sur le bouton correspondant.
3. Enregistrer cette fonction `go` comme *handler* de l'évènement `load` de `window` 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…

2 Étape 2

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

2. Définir la fonction `displayStatus(life, money, awake)` qui permet d'afficher l'état du monstre reçu en paramètre dans la liste `.status` (là aussi une nouvelle variable globale est à définir…) de l'interface (ce qui *remplace* donc le contenu précédent).
3. Reprogrammer la fonction `showme()` afin qu'elle n'utilise plus une alerte mais la fonction `log()` (dans un premier temps, à tester) et ensuite la fonction `displayStatus()` (dans un second temps, à tester aussi). Lorsque les 2 tests sont réalisés, laisser juste un appel à `log()` à l'intérieur de la fonction `showme()`.

❸ Étape 3

1. Compléter les actions que peut réaliser le monstre en contruisant les fonctions `run`, `fight`, `work` et `eat`, 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 l'*action box*) pour expliquer ce qui se passe (que l'action entreprise **se réalise ou pas** — 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),
   — `run` : perte de 1 point de vie,
   — `fight` : perte de 3 points de vie,
   — `work` : perte d'1 point de vie et gain de 2 unités d'argent,
   — `eat` : perte de 3 unités d'argent et gain de 2 points de vie.
2. Construire la fonction `sleep` qui endort le monstre et programme son réveil 7 s[1] plus tard, en utilisant un *timer* : voir la fonction `setTimeOut` dans la référence JS. Vérifiez que lorsque le montre dort, il ne peut pas courir, manger ni combattre. 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.

❹ Étape 4

1. Construire une fonction `hasard()` qui exécute une des actions au hasard (voir `Math.random()`). On donne une indication à ce sujet : penser à construire un tableau de *handlers*…

   Au chargement de la page, mettre en place une exécution toutes les 12 s[2] de la fonction `hasard()` (voir pour cela la fonction `setInterval()` dans la référence JS).
2. Programmer et associer les actions correspondant aux boutons *Kill* (pour tuer le monstre) et *New life* (pour lui redonner une nouvelle vie, **uniquement** s'il est déjà mort).
3. Modifier la fonction `displayStatus()` pour ajouter un effet visuel lié à l'état du monstre :
   — faire varier la couleur de la boîte `#monster` en fonction du nombre de points de vie (par ex. < 5 : rouge, < 10 : orange, < 15 : bleu, > 20 : vert etc.
   — faire varier l'épaisseur de la bordure de la boîte `#monster` en fonction de la quantité d'argent possédée par le monstre.
4. Apporter des améliorations ergonomiques à votre application :
   — certaines peuvent être purement esthétiques (en les implantant au bon endroit, *i.e.* principalement dans le CSS),
   — autant que possible, imaginer des codes visuels (comme ceux qui vous sont proposés ci-dessus — couleur de boîte et épaisseur de boîte) qui permettent de rendre compte de l'état dans lequel se trouve le monstre à un moment donné.

[1] Définissez donc une constante au début de votre fichier pour cette valeur : il sera plus facile de tester votre projet, autant pour vous que pour nous.

[2] Définissez donc…`[snip]`