Leçon 8.1.3 – HTML5 : Méthodologie

s1 ----------

s2 ----------

Intéressons-nous maintenant au code qui va

- créer la structure des écrans et leur contenu,
- permettre toutes les règles et fonctionnalités que nous avons identifiées.

Parlons structure. La première idée est ici de prévoir trois pages HTML différentes.

Cependant, l'écran de bilan a besoin d'afficher des informations liées à des événements qui ont eu lieu lors de l'écran de jeu, telles que le score et le niveau atteint. Il faudrait donc transférer ces informations de la page HTML de jeu à la page HTML de bilan. Or, dans le cadre de ce MOOC nous n'avons pas appris à faire cela.

Conclusion, l'écran de jeu et l'écran de bilan seront affichés par la même page HTML ; nous ferons disparaître ou apparaître les éléments en fonction de l'écran où nous trouvons.

Tant qu'à faire, intégrons l'écran d'accueil dans cette structure. Nous faisons donc le choix d'un seul fichier HTML contenant tous les objets. Ce qui est pratique du point de vue des fichiers CSS et JavaScript extérieurs : ils ne seront déclarés qu'une fois.

s3 ----------

À nouveau, une petite figure est utile.

Vous noterez que nous avons un conteneur div (ce pourrait être un conteneur HTML5 section) pour chaque écran ; le footer, commun à chaque écran, sera affiché tout le temps.

s4 ----------

Il nous faut maintenant envisager les choses du point de vue de la programmation en reprenant toutes les règles et fonctionnalités.

Imaginons le code JavaScript qui va permettre cela et construisons progressivement sa structure. Pour simplifier le propos, imaginons que l'ensemble de ce code se trouve dans un fichier code.js. Attention, nous sommes toujours dans la partie conception. Il ne s'agit pas ici d'écrire du code, mais d'imaginer les grandes parties de ce code JavaScript, de balayer notamment toutes les règles et fonctionnalités et pour chacune d'entre elles de se poser la question de savoir comment on va la mettre en place. Néanmoins, adoptons un principe important : commençons à donner des noms aux paramètres, variables et fonctions que nous identifions.

Quelle est la structure générale de ce code ? Je vous propose ici une structure générale adaptée au travail de conception que nous avons déjà commencé et qui doit pouvoir être mise en œuvre pour bon nombre d'animations et de jeux. Dans des cas plus complexes, elle doit pouvoir être utilisée sur des sous-parties. Voici sur un dessin cette structure, la première boîte est le fichier code.js.

Une fois que l'ensemble du code sera chargé (événement que l'on sait détecter, par exemple grâce à l'instruction window.onload) on lancera une fonction principale d'initialisation (pourquoi ne pas l'appeler init()). On la représente sur notre feuille par une nouvelle boîte. Voyons dans les grandes lignes, avant de détailler plus avant, les tâches dont se charge cette fonction init().

- D'abord elle va créer la structure HTML, c'est-à-dire définir tous les éléments potentiels identifiés précédemment écran par écran. Notons que tout ou partie de ces éléments peuvent être directement définis dans le fichier HTML.
- Ensuite, elle va définir les données, c'est-à-dire tous les paramètres dont le jeu à besoin et dont nous avons parlé dans la vidéo précédente.
- Puis, nous initialiserons les variables du jeu. Notons déjà que, comme il sera possible de rejouer, cette opération de réinitialisation des variables sera lancée une seconde fois, d'où l'idée de créer une fonction pour cette opération.
- Ensuite, il faut installer les gestionnaires qui permettront de mettre en œuvre les fonctionnalités.
- Enfin, il faut installer le mécanisme par lequel les règles vont pouvoir être mises en œuvre. J'appelle ici ce mécanisme « moteur de règles ».
- Tout est prêt. On affiche le premier écran : l'écran d'accueil.

En dessous, nous trouverons toutes les fonctions qu'il nous paraît pratique de créer, notamment pour ne pas répéter des parties de code. Ces fonctions seront progressivement identifiées.

Voici pour la structure générale. Reprenons ces tâches l'une après l'autre.

s5 ----------

Créer la structure HTML ne pose pas de problème particulier. Nous savons cibler un élément et modifier son contenu. Bien sûr si le contenu de ces éléments ne change pas au cours du jeu, comme le titre de l'écran d'accueil ou le contenu du footer vous pouvez les écrire directement dans le fichier HTML. Pour des raisons de lisibilité, il est possible de créer une fonction supplémentaire pour créer la structure de chaque écran. Attention, au moment du lancement, ces écrans de jeu et de bilan seront masqués.

Définir les paramètres du jeu – c'est-à-dire les données non modifiées au cours du jeu – consiste ici à définir par exemple la durée de jeu limite (notée tempsLimite), la couleur des bulles à éviter et celle des bulles à éviter (rassemblées un tableau listeCouleurs), etc. La liste précise de ces paramètres sera construite plus loin.

Initialiser les variables n'est pas forcément simple. Comme leur nom l'indique, les variables sont des

éléments qui vont changer tout au long du jeu ; les variables vont avoir un rôle important, notamment au niveau des règles et fonctionnalités. Ici, le temps de jeu (tempsJeu) a son importance puisqu'il est limité et puisque le niveau courant en dépend directement ; on pense également à une variable indiquant l'écran courant (notée ecranCourant), une autre pour le score (notée score) ; on pense au tableau – vide au départ – qui contiendra toutes les bulles au fur et mesure de leur création (noté listeBulle, à la vitesse initiale des bulles (notée vitesse), au nombre initial de vies (nombreVies), etc. La liste précise de ces variables sera construite plus loin au fur et mesure que nous avancerons dans la conception du jeu.

Occupons-nous des gestionnaires pour les fonctionnalités. Les gestionnaires consistent à installer des écouteurs et à définir les fonctions qui vont être lancées lors des événements correspondants : clic sur un bouton, etc.

Qu'en est-il des règles ? Installer le mécanisme pour gérer les règles est très simple. Il s'agit d'un simple setInterval() associé à une fonction que l'on appelle par exemple regles().

Cette fonction regles() va tester à intervalles réguliers toutes les règles qui ont été listées, ce qui nous impose de définir comme paramètre l'intervalle en millisecondes entre deux rafraîchissements : intervalleRafraichissement.

Comme nous l'avons vu, ces règles sont des conditions de type si alors. Dans des animations contenant beaucoup d'écrans, il peut y avoir plusieurs dizaines de règles : il est donc normal de tester dans cette fonction uniquement les règles de l'écran où le joueur se trouve. C'est la raison d'être de la variable ecranCourant.

Le lancement, enfin, consiste à afficher l'écran d'accueil.

Voyons maintenant, écran par écran, sur la base du travail fait précédemment,

- comment installer les éléments constitutifs de chaque écran au moment où il s'affiche ;
- comment installer les règles et les fonctionnalités qui sont de nature à modifier ces éléments, voire à changer d'écran.

s6 ----------

Dans la structure du fichier JavaScript, on fait donc ici un focus sur les règles et les fonctionnalités.

Intéressons-nous à l'écran d'accueil.

Au départ, tous les éléments potentiels s'affichent. Ceci se fait grâce à une fonction afficheEcranAccueil() qui affichera seulement l'écran d'accueil.

Il n'y a pas de règle pour cet écran.

Seule fonctionnalité de cet écran :

- si on clique sur le bouton #boutonJeu alors on passe à l'écran #jeu

Pour assurer cette fonctionnalité, on installe un gestionnaire sur le bouton #boutonJeu. Que fait ce gestionnaire ? En cas de clic il affiche l'écran de jeu. Ceci se fait grâce à une fonction afficheEcranJeu() qui masquera l'écran d'accueil et affichera l'écran de jeu.

s7 ----------

Intéressons-nous à l'écran de jeu. Au départ tous les éléments potentiels s'affichent : le score, le niveau, le nombre de vies, le temps de jeu ainsi que le bloc contenant l'animation.

Qu'avons-nous identifié comme règles ? Cet écran et ses éléments sont soumis à plusieurs règles et fonctionnalités. Considérons les deux premières :

- Sans condition, les bulles descendent
- Si on déplace la souris alors la bulle bleue suit le mouvement

Une parenthèse sur cette animation dans laquelle des bulles descendent. Nous avons réalisé une animation similaire à l'aide de la balise canvas. L'animation fonctionnait grâce à la méthode setInterval qui lançait à intervalles réguliers une fonction de dessin.

Quelle était la structure de ce code ?

Du point de vue HTML, on s'appuyait sur la balise canvas. Du point de vue JavaScript, on s'appuyait sur les éléments suivants :

- la méthode setInterval() qui lançait à intervalles réguliers la fonction animer(),

- la fonction animer() proprement dite qui s'appuyait sur une fonction générique de dessin des bulles que l'on renomme ici en dessineBulle(),

- la fonction dessineBulle() proprement dite.

Pour reproduire cette animation dans le jeu qui nous occupe, il suffit donc de :

- créer le code HTML du canvas, au moment de la création des écrans ou plus simplement, comme ce code ne changera pas, dans le HTML
- de lancer la fonction animer() à l'intérieur de la fonction regles() après avoir vérifié que l'on se trouvait dans l'écran de jeu.

Dans le précédent exercice, la liste des bulles était parfaitement définie. Dans ce jeu, nous choisissons d'avoir un nombre aléatoire de bulles. Une façon de procéder est de partir d'une liste de bulles vides et d'y ajouter de nouvelles bulles au fur et à mesure du déroulement du jeu, chaque bulle étant elle-même définie par un tableau de valeurs[1]. La fonction animer() va donc :

1. incrémenter le temps de jeu ;
2. à intervalles réguliers, créer une nouvelle bulle (ce qui nous impose de définir comme paramètre l'intervalle en millisecondes entre deux créations de bulles : intervalleNouvelleBulle) ; créer une bulle signifie ajouter un enregistrement à un tableau rassemblant toutes les bulles (appelons le listeBulles – il sera initialisé dans la fonction d'initialisation des variables) ; une nouvelle fonction creeBulle() va s'en charger ; cette nouvelle bulle est le plus souvent noire et verte de temps en temps. Ainsi la fonction animer

1 Nous pourrions utiliser des objets JavaScript, mais cette notion dépasse le cadre de ce MOOC

va

- après un certain nombre de bulles noires, créer une bulle verte (ce qui nous impose de définir comme paramètre le nombre de bulles noires après lequel une bulle verte est créée : proportionBullesNoires) ; nous allons bien entendu pouvoir utiliser la même fonction creeBulle() dont il suffit de passer en argument la couleur de la bulle ;

2. grâce à une boucle sur l'ensemble des bulles contenues dans le tableau, dessiner toutes les bulles contenues à un instant t dans le tableau des bulles ; dans notre précédent travail, la fonction dessineBulle() était lancée avec deux arguments : l'identifiant de la bulle et sa vitesse ; ici, la position de la bulle selon x, sa position selon y et son rayon sont aléatoires.
3. dessiner une bulle particulière – la bulle bleue – précisément à l'endroit du pointeur de la souris ; nous allons bien entendu pouvoir utiliser la même fonction dessineBulle ; afin de pouvoir dessiner cette bulle, nous avons besoin à chaque instant de la position de la souris ; ceci se fait en plaçant un gestionnaire/écouteur mousemove sur le canvas qui lance une fonction positionSouris() permettant de calculer cette position en x et en y
4. mettre à jour l'affichage : score, nombre de vies, niveau, temps de jeu.

Vous remarquerez que pour l'instant on dessine également des bulles qui présentent un y supérieur à la hauteur du canvas et qui sont donc non visibles. Il vous sera facile de corriger cela, si vous le souhaitez.

s8 ----------

Règle suivante :

- en cas de choc entre la bulle bleue et une bulle noire, cette dernière disparaît et le nombre de vies est diminué

Comment mettre en oeuvre cette règle ? Nous pourrions à la fin de la fonction animer() lancer une nouvelle boucle sur les bulles du tableau et tester, pour chaque bulle, s'il existe collision. Une meilleure façon de procéder (qui allège le processeur) consiste à faire ce test au moment du dessin de la bulle : si la position de la souris bleue est suffisamment (à vous de définir ce critère...) proche d'une autre bulle, on considère qu'il existe collision et cette bulle n'est pas dessinée ; comment faire pour qu'elle ne soit plus dessinée lors des rafraîchissements suivants ? Eh bien, tout simplement en ajoutant un attribut de plus à chaque bulle du tableau des bulles : un booléen qui définit si cette bulle doit être affichée ou pas.

Il restera à décrémenter le nombre de vies et à mettre à jour.

s9 ----------

Règle suivante :

- en cas de choc entre la bulle bleue et une bulle verte, cette dernière disparaît et le score est augmenté

Comment mettre en oeuvre cette règle ? De la même façon que la précédente, au sein de la fonction

animer() simplement ici c'est le score qui évolue.

Règle suivante :

- au bout d'un certain temps de jeu, le niveau change : la vitesse des bulles augmente.

Comment mettre en oeuvre cette règle ? A nouveau au sein de la fonction animer() : à intervalles réguliers, on fait évoluer la vitesse, ce qui nous impose de définir comme paramètre l'intervalle en millisecondes entre deux changements de niveaux – intervalleChangementVitesse – et l'incrément de vitesse – incrementVitesse.

Règle suivante :

- si le temps de jeu dépasse un certain nombre de secondes ou si le nombre de vies est nul alors on passe à l'écran de bilan (et on masque les autres écrans)

Comment mettre en oeuvre cette règle ? Au début de la fonction animer(), on teste le temps de jeu par rapport à un temps limite (défini comme paramètre : tempsLimite). Si le temps limite est dépassé, on passe à l'écran de bilan. Ceci se fait grâce à une fonction afficheEcranBilan() qui masquera l'écran de jeu et affichera l'écran de bilan.

s10 ----------

Une dernière fonctionnalité :

- si on clique sur le bouton #boutonQuitter alors on revient à l'écran d'accueil

Pour mettre en oeuvre cette fonctionnalité, on installe un gestionnaire de clic sur le bouton #boutonQuitter ; ce gestionnaire affiche l'écran d'accueil dans les mêmes conditions qu'au début, notamment en masquant les deux autres écrans. Il ne faut pas oublier de réinitialiser les variables du jeu :

- le temps de jeu : tempsJeu,
- l'écran courant : ecranCourant,
- le niveau courant : niveauCourant,
- le score,
- la position de la souris,
- le tableau de bulles : listeBulles,
- la vitesse initiale des bulles en pixels par seconde : vitesse,
- le nombre initial de vies : nombreVies.

s11 -----

Intéressons-nous à l'écran de bilan. Au départ tous les éléments potentiels s'affichent : les deux boutons #boutonRejouer et #boutonAccueil ainsi que le score obtenu et le niveau atteint.

Dans cet écran de bilan, deux fonctionnalités :

- si on clique sur le bouton #boutonRejouer alors on repasse à l'écran de jeu

Pour assurer cette fonctionnalité, on installe un gestionnaire sur le bouton #boutonRejouer ; que fait ce gestionnaire ? Il affiche certes l'écran #ecranJeu, mais avant il faut s'assurer que les variables sont réinitialisées. On voit ici que l'on peut introduire une fonction reinitialisation() qui s'occupe de cette fonctionnalité commune à plusieurs écrans.

- si on clique sur le bouton #boutonAccueil alors on passe à l'écran #accueil

Pour assurer cette fonctionnalité, on installe un gestionnaire sur le bouton #boutonAccueil ; que fait ce gestionnaire ? Il affiche certes l'écran #ecranAccueil, mais il faut également réinitialiser les variables.

s11 -----

Voilà, nous sommes arrivés au terme de l'étape de conception. Pour l'instant, ce travail est entièrement réalisé au brouillon sur une feuille A3. Je vous propose maintenant de rédiger le cahier de spécifications c'est-à-dire un document textuel qui possède la structure présentée précédemment et dans lequel vous pourrez insérer plusieurs scans des figures que vous avez dessinées.

Si j'ai détaillé ce processus consistant à balayer les écrans, les règles et les fonctionnalités jusqu'au bout, c'est pour tenter de vous montrer plusieurs choses :

- Ce n'est qu'en essayant d'imaginer comment une règle ou une fonctionnalité va être mise en oeuvre que l'on peut identifier quelles informations, quels attributs des objets que l'on manipule – ici les niveaux et les bulles – seront nécessaires lors de l'exécution du code, en d'autres termes quelle structure de données est nécessaire pour l'exécution.
- Çà et là différents choix sont possibles ; certaines options peuvent se révéler difficiles à gérer.
- Çà et là des écueils surgissent qu'il faut contourner.
- Vous avez vu comment les variables et le tableau des bulles ont évolué tout au long de la conception et vous comprenez qu'il est plus facile de faire ce travail d'adaptation en phase de conception à l'aide d'un crayon et d'une gomme, qu'en phase de développement où une partie importante du code a été écrite.
- Vous avez vu comment en prenant en compte les règles et fonctionnalités l'une après l'autre, telles qu'on les a identifiées initialement, les algorithmes – notamment celui de la descente des bulles –, évoluent.
- Toutes les variables importantes portent déjà un nom.

Bien évidemment, certaines choses peuvent avoir été oubliées dont on va se rendre compte lors de la première tentative d'exécution du code.

Ce dont je suis convaincu après de nombreuses années de développement de projets plus ou moins importants c'est que plus vous pousserez loin ce travail d'analyse, susceptible régulièrement de modifier la structure de données initiale, plus vous irez vite dans votre développement et plus vous réduirez les risques de mauvaise surprise en découvrant après avoir écrit la presque totalité du code un élément qui le remet complètement en cause.

s12 ----------

Une dernière chose dans ces spécifications techniques détaillées : l'architecture de fichiers.

L'architecture de fichiers est ici très simple. Elle se résume à :

- un fichier HTML : index.html,
- un dossier styles contenant le fichier styles.css,
- un dossier js contenant le fichier code.js et si l'on souhaite travailler avec jQuery le fichier jQuery.js,
- un dossier images contenant l'image affichée dans l'écran d'accueil.

s13 ----------

Cette partie nous a permis de détailler deux chapitres importants du cahier de spécifications :

- Le cahier des charges fonctionnel d'un jeu qui détaille
    - la liste des écrans et leur contenu potentiel
    - la liste des règles et des fonctionnalités
- Les spécifications techniques détaillées
    - les données importantes
    - éléments généraux
    - code
    - architecture de fichiers

Dans la prochaine partie, nous allons aborder la phase de développement et examiner rapidement quelques bonnes pratiques.

À tout de suite.