Leçon 8.2 – HTML5 – Méthodologie

Jeu des bulles - Développement pas-à-pas

J.-Y. Plantec – v1 10/2017

Ce document a pour objectif de vous aider à réaliser le développement du jeu des bulles. Le code complet du jeu n'est pas donné : il vous faudra écrire et tester plusieurs parties en vous aidant du travail de conception réalisé précédemment et en suivant scrupuleusement les étapes proposées.

De loin en loin, vous trouverez des copies d'écran qui vous permettront de vérifier votre progression.

1 – Code HTML

On commence par le fichier HTML que l'on construit au sens du cahier des charges fonctionnel et de la partie consacrée à la liste des écrans et à leur contenu potentiel. Cette partie de code est donnée.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8"/>
<title>BubbleHunt</title>
<link rel="stylesheet" media="screen" href="./css/styles.css" type="text/css"/>
<script src="./js/jquery.js"></script>
<script src="./js/code.js"></script>
</head>
<body>
<div id="ecranAccueil">
        <div id="titre">BubbleHunt</div>
        <div id="image">(ici une image du jeu)</div>
        <div id="regles"><p>Déplacez la bulle bleue avec la souris pour attraper les seules bulles vertes
        <div id="boutonJouer"><input type="submit" value="Jouer"></div>
</div>
<div id="ecranJeu">
        <div id="score">(score)</div>
        <div id="niveau">(niveau)</div>
        <div id="vies">(vies)</div>
        <div id="temps">(temps)</div>
        <div id="animation">
            <canvas id="dessin" width="450" height="300">
                Texte pour les navigateurs qui ne supportent pas canvas
            </canvas>
        </div>

        <div id="boutonQuitter"><input type="submit" value="Quitter"></div>
</div>
<div id="ecranBilan">
        <div id="scoreFinal">(score)</div>
        <div id="niveauFinal">(niveau)</div>
        <div id="boutonAccueil"><input type="submit" value="Accueil"></div>
        <div id="boutonRejouer"><input type="submit" value="Rejouer"></div>
</div>
<footer>BubbleHunt v1.0</footer>
</body>
</html>
```

2 – Code CSS

Concernant le code CSS, le fichier s'appelle styles.js. Pour commencer à écrire ce fichier CSS, on s'appuie sur les éléments généraux des spécifications techniques détaillées. On définit ainsi :

- le fond de page gris clair #ccc
- le style du pied de page : Arial, noir, 12 pts
- le texte de base des écrans : Arial, noir, 12 pts
- pour l'écran d'accueil
  - le titre : Arial, rouge, 16 pts
  - le texte expliquant les règles : Arial, noir, 12 pts
  - une image du jeu 450px *300px
- Pour l'écran de jeu :
  - le texte pour le score et le nombre de vies : à nouveau Arial, noir, 12 pts
  - un élément de type canvas 450*500 à l'intérieur duquel évolueront les bulles de couleurs sur un fond, par exemple blanc : #fff
- Pour l'écran de bilan, il y aura :
  - le texte du bilan : à nouveau Arial, noir, 12 pts

```
body {
    background-color: #ccc;
}
* {
  font-family: "Arial", sans-serif;
  font-size: 12pt;
  color: #000;
}
#titre {
  font: bold 16pt sans-serif;
  color: red;
}
footer {
}
```

Vous mettrez à jour ce fichier CSS lorsque vous positionnerez les boutons et définirez le canvas.

3 – Code JavaScript

Concernant le code JavaScript

- ici, on a choisi de mettre en oeuvre jQuery, mais ce n'est pas une obligation ;
- le fichier js s'appelle code.js

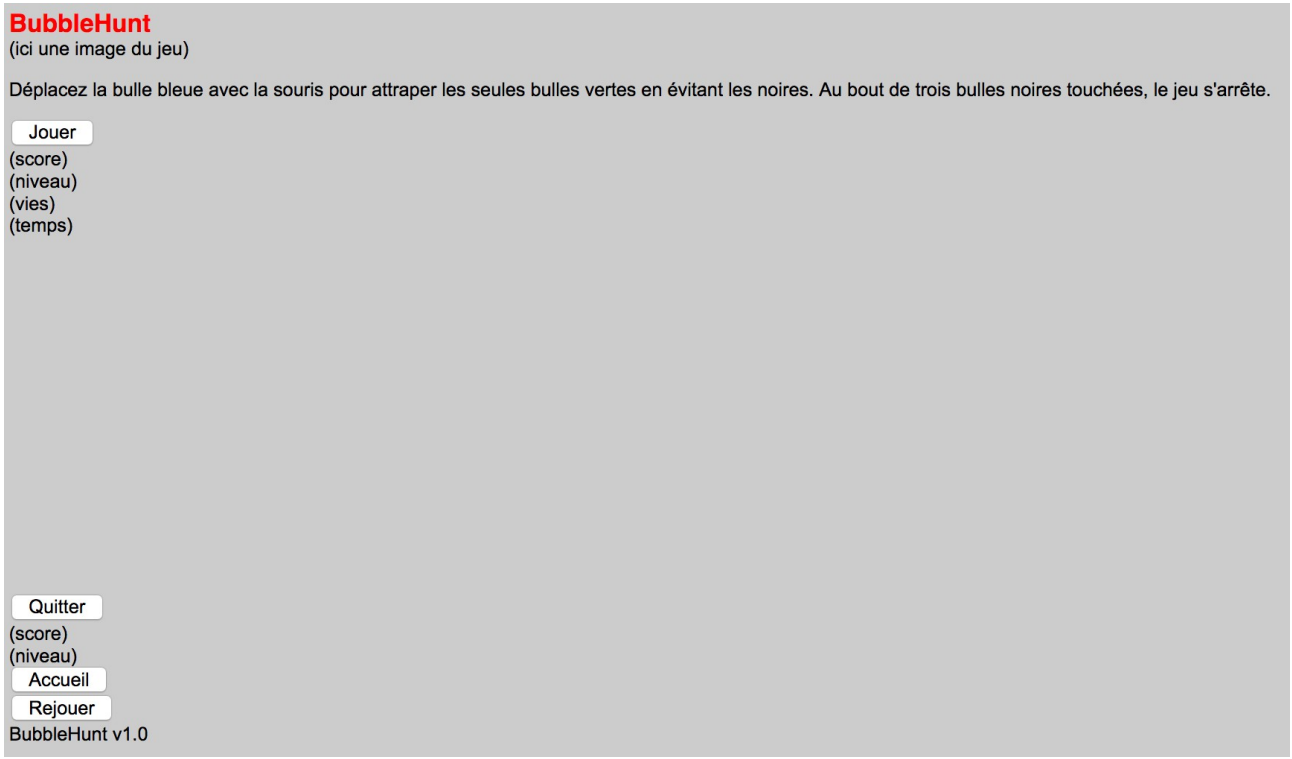
Structure générale

On commence à écrire la structure générale du code :

```
$(function() {
        init();
});

function init(){

    // structure

    // paramètres (données non modifiables et non modifiées par le jeu) - variables globales

    // initialisation des variables

    // gestionnaires

    // moteur de règles

    // lancement : affichage de la page d'accueil

}
```

Pour ceux qui n'auraient pas choisi jQuery, il faut bien entendu remplacer $(function(){...}) ; par document.ready...

À ce stade, on peut tester ce travail pour voir si aucune erreur n'est levée. Bien évidemment, pour l'instant, tous les écrans s'affichent ; et vous n'avez pas encore d'image du jeu.



Puis, on construit ce code étape par étape comme ci-dessous et en suivant les spécifications techniques détaillées.

Étape 1 : structure

Nous n'avons pas ici à créer par JavaScript les éléments de structure de chaque écran, car ils sont déjà définis dans le fichier HTML.

Étape 2 : données

On définit ici les paramètres du jeu :

- l'intervalle entre deux rafraîchissements en millisecondes : intervalleRafraichissement
- l'intervalle entre deux nouvelles bulles en millisecondes : intervalleNouvelleBulle
- le nombre de bulles noires pour une bulle verte : proportionBullesNoires
- le temps de jeu en millisecondes : tempsLimite
- la liste des couleurs, définie ici dans un dictionnaire JavaScript : listeCouleurs = {"B": "#0000ff","V": "#00c000","N": "#555"} ;
- l'intervalle entre deux changements de vitesse en millisecondes (changement de niveau) : intervalleChangementVitesse
- l'incrément de vitesse à chaque changement de niveau : incrementVitesse
- le rayon de la bulle bleue : rayonBulleBleue

A vous de donner des valeurs à ces paramètres.

Étape 3 : variables

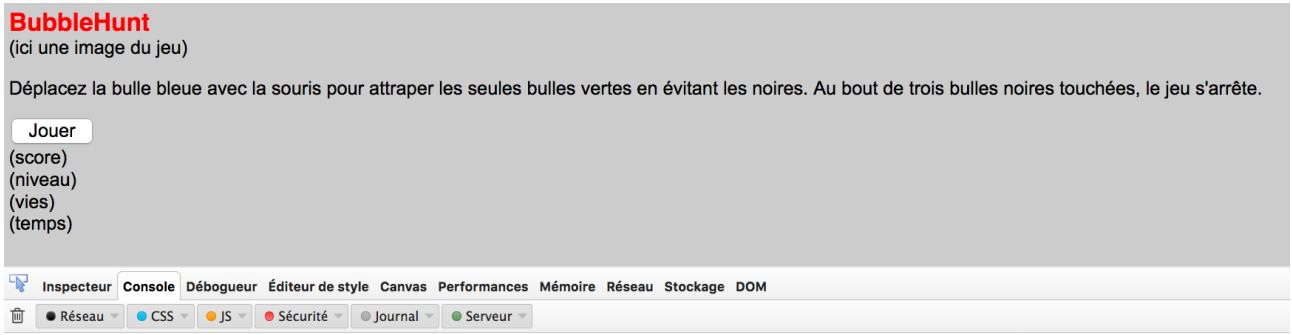
C'est à cet endroit que l'on définit la variable globale monCanvas (à l'aide du code déjà utilisé plusieurs fois dans les exercices.

```
   monCanvas =
```

C'est également ici que l'on définit et que l'on initialise les variables. Lors de l'étape de conception, nous avons vu que la réinitialisation du jeu intervenait à plusieurs endroits, d'où l'intérêt de construire une fonction de réinitialisation du jeu dont le code est donné ci-dessous.

```
function reinitialisation() {
    tempsJeu = 0;
    ecranCourant = null;
    niveauCourant = 1;
    score = 0;
    // position de la souris
    xSourisCanvas = monCanvas.width/2;
    ySourisCanvas = monCanvas.height/2;
    // liste des bulles
    listeBulles = [];
    // nombre total de bulles (sans la bulle bleue)
    nbBulles = 0;
    // vitesse initiale des bulles en pixels par seconde
    vitesse = 10;
    // nombre initial de vie
    nombreVies = 3;
}
```

On teste ces ajouts à l'aide de la console de débogage du navigateur.



Étape 4 : lancement

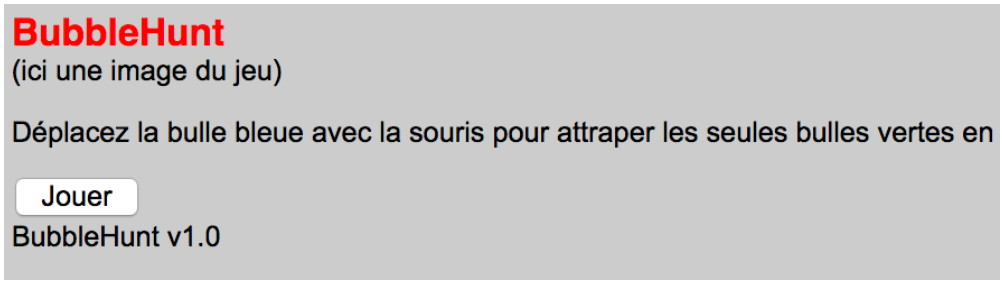
Au chargement du jeu

L'écran d'accueil s'affiche au chargement (et masque les autres, bien entendu). Les instructions sont écrites dans la partie « lancement » de la fonction init(). On peut créer une fonction afficheEcranAccueil() qui

- met à jour la variable ecranCourant (ecranCourant="accueil";),
- affiche l'écran #accueil,
- masque les autres écrans.

```
function afficheEcranAccueil(){...7 lines }
```

C'est à vous ! On teste. Seul l'écran d'accueil est visible.



Étape 5 : gestionnaires et règles

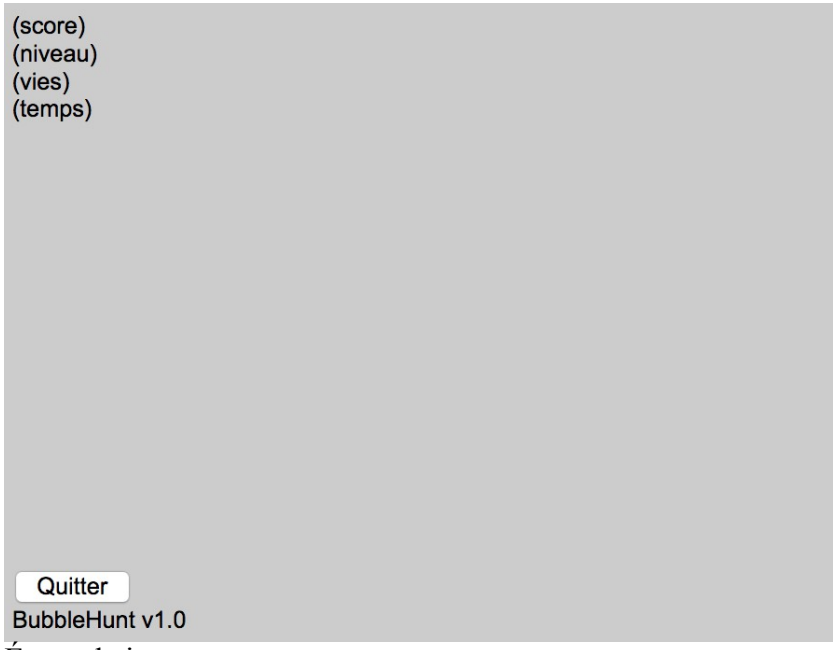
Écran d'accueil

On s'occupe des règles et fonctionnalités de l'écran d'accueil.
Seule fonctionnalité de cet écran : si on clique sur le bouton #boutonJeu alors on passe à l'écran #jeu. Dans la partie conception, nous avons choisi d'installer un gestionnaire sur le bouton #boutonJeu associé à une fonction afficheEcranAccueil() qui

- met à jour la variable ecranCourant,
- affiche l'écran de jeu,
- masque les autres écrans.

C'est à vous !

On teste :



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.

On s'occupe des règles de l'écran de jeu. Cet écran et ses éléments sont soumis à plusieurs règles et fonctionnalités. Première règle :

- Sans condition, les bulles descendent

Selon les spécifications détaillées, il suffit, maintenant que le code HTML du canvas a été créé :

- d'introduire une méthode setInterval() qui lance à intervalles réguliers une fonction regles(),

```
// moteur de règles
inter = setInterval(regles, intervalleRafraichissement);
```

- de définir la fonction regles() qui vérifie que l'on se trouve bien dans l'écran de jeu et lance une fonction générique qui va dessiner les bulles que l'on nomme ici animer().

```
function regles(){
    if (ecranCourant === "ecranJeu"){
        // animation
        animer();
    }
}
```

Que fait cette fonction animer() ? Reprenons le document de spécifications.

1. Elle va incrémenter le temps de jeu ;
2. Elle va à intervalles réguliers créer une nouvelle bulle – fonction creeBulle() – c'est-à-dire ajouter un enregistrement au tableau listeBulles ; après un certain nombre de bulles noires, elle créera une bulle verte ; choix de conception : une bulle est représentée par un tableau contenant :
   - la position selon x
   - la position selon y
   - la couleur
   - le rayon
   - un attribut précisant si la bulle est visible ou pas

3. grâce à une boucle sur l'ensemble des bulles contenues dans le tableau listeBulles, dessiner toutes les bulles contenues à un instant t dans le tableau des bulles – fonction dessineBulle() ;
4. dessiner une bulle particulière – la bulle bleue – précisément à l'endroit du pointeur de la souris ; la position de la souris est obtenue en plaçant un gestionnaire/écouteur mousemove sur le canvas qui lance une fonction positionSouris() permettant de calculer cette position en x et en y (ce gestionnaire est inséré dans la partie « gestionnaires » du code de la fonction init(), la fonction positionSouris() étant très similaire à celle étudiée dans les exercices précédents.

```
// interactivité sur le canvas
monCanvas.addEventListener("mousemove", positionSouris, false);
```

5. mettre à jour l'affichage : score, nombre de vies, niveau, temps de jeu.

Voici une bonne partie du code de la fonction animer que nous allons commenter.

```
function animer() {

    // 1- temps de jeu
   tempsJeu = tempsJeu + intervalleRafraichissement;

    // 2 - création des bulles N et V - test sur le temps
    if (tempsJeu % intervalleNouvelleBulle === 0){
        // création d'une nouvelle bulle -> test s'il est temps de créer une verte
        if(listeBulles.length % proportionBullesNoires === 0 && listeBulles.length !== 0) {
            creeBulle("V");
        } else {
            creeBulle("N");
        }
    }
    // 3 dessin des bulles
    ctx.clearRect(0,0, monCanvas.width,monCanvas.height);
    for (var j=0; j<listeBulles.length; j++){
        var bulle = listeBulles[j];
        dessineBulle(bulle, j);
    }
    // 4 - dessin de la bulle bleue (ne fait pas partie de la liste
    dessineBulle([xSourisCanvas , ySourisCanvas, "B", rayonBulleBleue, true], null);

    // 5 - affichages...

}
```

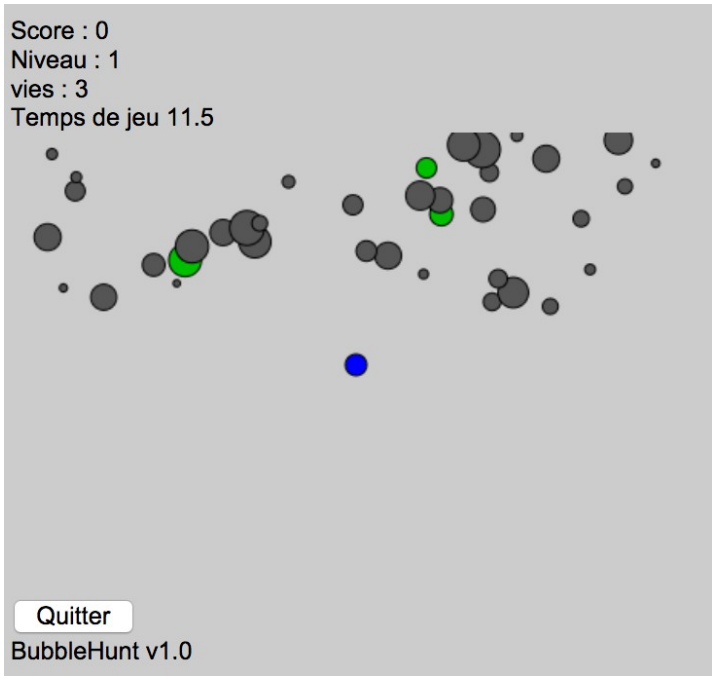
- l'incrémentation du temps de jeu est évidente
- pour créer à intervalles réguliers (tous les intervalleNouvelleBulle) une nouvelle bulle, on utilise la fonction JavaScript modulo : si le reste de la division de tempsJeu par intervalleNouvelleBulle est nul, on lance la création d'une bulle ;
- même principe pour la création à intervalles réguliers d'une bulle verte ;
- le dessin d'une bulle implique la fonction dessineBulle() qui reçoit en paramètre un tableau rassemblant les caractéristiques de la bulle (position selon x, position selon y, couleur, rayon, un attribut précisant s'il est est visible ou pas) et l'identifiant de cet enregistrement (nous verrons plus bas pourquoi) ;
- le dessin d'une bulle bleue implique la même fonction dessineBulle() avec ce dernier argument à null ;

Votre travail

1. définir la fonction creeBulle() – indications :
   - argument : la couleur sous forme d'une lettre
   - actions :
     - création d'un tableau [position selon x en px, position selon y en px, couleur sous forme d'une lettre, rayon en px, un booléen pour la visibilité]
     - ajout de ce tableau dans le tableau listeBulles
   - retour : rien
2. définir la fonction dessineBulle()
   - argument : une bulle sous forme de tableau
   - action : dessin de la bulle dans le canvas
   - retour : rien
3. compléter l'affichage du score, du temps de jeu, etc.

Quelques conseils :

- Soyez vigilants dans dessineBalle() à bien récupérer les bonnes informations pour dessiner la balle : couleur, taille, vitesse, position selon x.
- Soyez vigilants sur la portée des variables utilisées.
- N'hésitez pas à utiliser le débogueur !
- Pour tester, on peut aussi mettre en commentaire le setInterval et lancer une seule fois la fonction Animer() pour vérifier les instructions.



Conformément au cahier de spécifications, pour éviter dessiner des bulles qui présenteraient un y supérieur à la hauteur du canvas, il suffit d'introduire un test supplémentaire dans la partie 2 (dessin des bulles). A vous !

Règle suivante :

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

Nous avons choisi d'effectuer ce test à l'intérieur de la fonction dessineBulle(). Attention, ce test ne doit être effectué que pour les bulles du tableau listeBulle(), or la fonction dessineBulle() est également utilisée par la bulle bleue : c'est la raison de l'existence du second argement de dessineBulle() qui est mis à null au moment du dessin de la bulle bleue. L'algorithme est le suivant :

SI la bulle courante (verte ou noire) touche la bulle bleue ET la bulle courante n'est pas la bulle bleue

ALORS

SI c'est une bulle noire

ALORS

décrémenter le nombre de vies

FIN SI

SINON (bulle verte)

ALORS

incrémenter le score

FIN SI

dans tous les cas, disparition (mettre l'argument visible à false)

FIN SI

A vous de jouer ! Et de tester. Si arrivez juste qu'ici, bravo ! L'essentiel est fait. Ce point était le point dur du développement. La suite est plus facile.

Règle suivante :

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

La mise en œuvre est simple : au sein de la fonction animer(), à intervalles réguliers et grâce à la fonction modulo on fait évoluer la vitesse :

```
if (tempsJeu % intervalleChangementVitesse === 0){
    // changement de niveau
    vitesse = vitesse + incrementVitesse;
    niveauCourant++;
}
```

Règle suivante :

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

La mise en œuvre est simple : au début de la fonction animer(), on teste le temps de jeu et, le cas échéant, on lance une afficheEcranBilan() qui masquera l'écran de jeu et affichera l'écran de bilan.

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 en lançant la fonction de réinitialisation.

C'est à vous !

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.

On s'occupe des fonctionnalités de l'écran de bilan :
- si on clique sur le bouton # boutonRejouer alors on repasse à l'écran #ecranJeu ;
- si on clique sur le bouton # boutonAccueil alors on repasse à l'écran #ecranAccueil.

Ces fonctionnalités sont faciles à mettre en oeuvre :
- un gestionnaire sur le bouton #boutonRejouer ; le clic lance
  - une réinitialisation
  - la fonction afficheEcranJeu()
- un autre sur le bouton #boutonAccueil ; le clic lance
  - une réinitialisation
  - la fonction afficheEcranAccueil()

C'est à vous !

4 – Tests

Voilà. Vous êtes arrivés au bout ! À ce stade, pourquoi ne pas valider le code qui a été écrit ?
- HTML : http://validator.w3.org
- CSS : http://jigsaw.w3.org/css-validator/
- JavaScript : http://www.javascriptlint.com/online_lint.php

5 – Jouabilité

À ce stade le jeu est simplement fonctionnel. Avant de le publier, il faut définir un jeu de données/paramètres tels que le jeu puisse être jouable/intéressant. À vous de définir ces paramètres :

- l'intervalle entre deux rafraîchissements en millisecondes
- l'intervalle entre deux nouvelles bulles en millisecondes
- le nombre de noires pour une bulle verte
- l'intervalle entre deux changements de vitesse en millisecondes (changement de niveau)
- l'incrément de vitesse à chaque changement de niveau

Je vous laisse échanger sur ce point sur le forum.

6 – Initiative personnelle

Parmi les petits développements possibles (mais, encore une fois pas obligatoires) :

- l'introduction de la gravité : dans la fonction dessineBulle(), modifier en ce sens l'instruction

```
var y = bulle[1]+vitesse*intervalleRafraichissement/1000;
```