

Aide-mémoire JavaScript



Thomas Nardou

Client : Aurélie Curchod

Temps : 40 périodes

Table des matières

| | |
|---|----|
| 1. Introduction : | 3 |
| 2. Analyse : | 3 |
| 3. Explication du code : | 3 |
| 4. Les syntaxes : | 7 |
| 4.1 Les variables : | 7 |
| 4.2 Les Fonctions : | 8 |
| 4.3 Les Conditions : | 9 |
| 4.4 Les Boucles : | 11 |
| 5. Prendre tous les éléments d'un tableau : | 12 |
| 5.1 ".forEach()" : | 12 |
| 5.2 ".some()" : | 12 |
| 6. Les Classes : | 13 |
| 6.1 Utiliser les classes dans le "main" : | 13 |
| 6.2 Exporter les classes : | 13 |

1. Introduction :

Lors de ce projet il nous a été demandé de réaliser une réplique du célèbre jeu vidéo rétro : le "Snake". Certains éléments ont été imposés notamment

- L'utilisation de const et de let
- L'utilisation des classes "Snake" et "Apple"
- L'utilisation de fonction fléchée
- L'utilisation des modules "export" et "import"

2. Analyse :

Pour commencer nous nous sommes penchés sur les mouvements du serpent car nous ne savions pas comment faire en sorte que le corps suive la tête du serpent.

Par la suite nous en avons conclu que le moyen le plus facile serait qu'un carré représente une partie du serpent et que chaque partie se trouverait dans un tableau.

Pour régler le problème du mouvement nous avons décidé que la dernière partie du serpent serait supprimée puis une nouvelle partie du serpent serait ajoutée et prendrait la place de la tête.

3. Explication du code :

Vérifie s'il n'y a pas eu de collision entre le serpent et les bords du jeu :

```
function checkCollision() {
  allSnakes[allSnakes.length - 1].getCoorX() > 700 ||
  allSnakes[allSnakes.length - 1].getCoorY() > 700 ||
  allSnakes[allSnakes.length - 1].getCoorX() < 0 ||
  allSnakes[allSnakes.length - 1].getCoorY() < 0 ? isDead = true : isDead;
}
```

Vérifie si le serpent ne se cogne pas contre une partie de son corps.

```
function checkSnakeCollision() {
  !partSnakeSpawned ? allSnakes.forEach((element, index) => {
    index !== allSnakes.length - 1 && allSnakes[allSnakes.length - 1].getCoorX() === allSnakes[index].getCoorX() &&
    allSnakes[allSnakes.length - 1].getCoorY() === allSnakes[index].getCoorY()
    ? isDead = true : isDead;
  }) : undefined;
}
```

Regarde si le joueur n'est pas mort :



```
!isDead ? framNumber % FRAME == 0 ? gameDraw() : undefined : loseDraw();
```

S'il est mort :

- Affiche l'écran de défaite :



```
function loseDraw() {  
  let GameOverTitle = document.querySelector('.GameOver');  
  
  firstTime ? GameOverTitle.textContent = "GameOver !" : undefined;  
  
  setTimeout(function() {  
    GameOverTitle.style.fontSize = "30px";  
    GameOverTitle.textContent = "Vous allez etre rediriger"  
    firstTime = false  
  }, 2000);  
  
  setTimeout(function() {  
    window.location.replace("../index.html");  
  }, 5000);  
}
```

S'il est vivant :

- Dessine l'aire de jeu :



```
function gameDraw() {  
  
  ctx.fillStyle = 'black';  
  ctx.fillRect(0, 0, 800, 800);  
  
  drawSnake();  
  moveSnake();  
  partSnakSpawned = false;  
  
  hasBeenEat ? spawnApple() : undefined;  
  
  drawApple();  
  
  // Regarde si le serpent à mangé un pomme  
  allSnakes[allSnakes.length - 1].getCoorX() == apple.getCoorX() * 100 && allSnakes[allSnakes.length - 1].getCoorY()  
  == apple.getCoorY() * 100 ?  
  (addSnake(), score += 1, hasBeenEat = true) : undefined;  
}
```

Dessine le serpent et le fait bouger dans la direction choisie:

```
function drawSnake() {
  allSnakes.forEach((element) => {
    element == allSnakes[allSnakes.length - 1] ? ctx.fillStyle = 'red' : ctx.fillStyle = '#B44C43'
    ctx.fillRect(element.getCoorX(), element.getCoorY(), OBJECT_WIDTH, OBJECT_WIDTH);
  });
}

function moveSnake() {
  let y = allSnakes[allSnakes.length - 1].getCoorY();
  let x = allSnakes[allSnakes.length - 1].getCoorX();

  switch(direction) {
    case 'd':
      y += OBJECT_WIDTH;
      break;

    case 'u':
      y -= OBJECT_WIDTH;
      break;

    case 'r':
      x += OBJECT_WIDTH;
      break;

    case 'l':
      x -= OBJECT_WIDTH;
      break;
  }

  // Va Supprimer la dernière partie du serpent et va la mettre devant la tête
  direction == 'd' || direction == 'u' || direction == 'r' || direction == 'l' ? (allSnakes.push(new Snake(x,y)),
  allSnakes.shift()) : undefined;
}
```

Regarde si une pomme a été mangé :

- Si oui, fait apparaitre une nouvelle pomme

```
function spawnApple() {
  let randCooryY = 0; // Coordonnée X de la pomme
  let randCooryX = 0; // Coordonnée Y de la pomme

  let appleCanSpawn = true; // Indique si la pomme peut apparaitre ou pas


  do {
    // Génère un chiffre aléatoire entre 0 et 8
    randCooryY = Math.floor(Math.random() * 8);
    randCooryX = Math.floor(Math.random() * 8);

    // Regarde si les coordonnées de la pomme ne sont pas égal à une partie du serpent
    allSnakes.some((n1)=>(n1.getCoorX() === randCooryX * 100 && n1.getCoorY() === randCooryY * 100)) ? appleCanSpawn =
    false : appleCanSpawn = true;
  }
  while(!appleCanSpawn);

  // Crée la nouvelle pomme
  apple = new Apple(randCooryX, randCooryY);

  // dis que cette pomme n'a pas été mangé
  hasBeenEat = false;
}
```

Permet d'afficher la pomme :



```
function drawApple() {  
  ctx.fillStyle = 'orange'  
  ctx.fillRect(apple.getCoorX() * 100, apple.getCoorY() * 100, OBJECT_WIDTH, OBJECT_WIDTH);  
}
```

Regarde si le serpent mange une pomme




```
allSnakes[allSnakes.length - 1].getCoorX() == apple.getCoorX() * 100 &&  
allSnakes[allSnakes.length - 1].getCoorY() == apple.getCoorY() * 100 ?  
  (addSnake(), score += 1, hasBeenEat = true) : undefined;
```

- S'il en mange une : ajoute une nouvelle partie au serpent et ajoute un point et une nouvelle pomme:



```
function addSnake() {  
  allSnakes.push(new Snake(allSnakes[allSnakes.length - 1].getCoorX(), allSnakes[allSnakes.length - 1].getCoorY()))  
  partSnakSpawned = true  
}
```



```
score += 1, hasBeenEat = true
```

4. Les Tests :

| Nom | Description | Réussi ? |
|-------------------------------------|---|----------|
| Lancer une partie | Dans le Menu du jeu si l'utilisateur appuie sur "jouer" une partie se lance | Oui |
| Se déplacer | Si le joueur presse une touche fléchée dans le jeu, le serpent se déplacera en suivant la direction de la flèche. | Oui |
| Faire apparaitre une pomme | Dès que le joueur a commencé à bouger une pomme apparait au hasard sur l'aire de jeu | Oui |
| Faire apparaitre une nouvelle pomme | Dès que le joueur a mangé une pomme une autre apparait aléatoirement sur l'aire de jeu | Oui |
| Faire grandir le serpent | Une fois que le serpent a mangé une pomme la taille du serpent augmente de 1 | Oui |
| Augmenter le score | Une fois que le serpent a mangé une pomme son score augmente de 1 | Oui |
| Faire perdre le joueur | Si la tête du serpent touche une partie de son corp le jeu s'arrête un message de Game over s'affiche et l'utilisateur est redirigé vers le menu principale | Oui |
| Faire perdre le joueur | Si le joueur touche les bords du jeu le jeu s'arrête un message de Game over s'affiche et l'utilisateur est redirigé vers le menu principale | Oui |

5. Les syntaxes :

5.1 Les variables :

JavaScript est un langage non typé c'est-à-dire que les variables/fonction n'ont pas un type défini dès le départ contrairement au C# qui lui est typé. Une variable permet de stocker une valeur cette valeur peut être modifiée à tout moment dans le code. Pour déclarer une variable en JavaScript voici la syntaxe :



```
let maVariable = "Coucou";
```

Sur cette partie nous déclarons notre variable avec le préfix **"let"** qui signifie que l'on va déclarer une variable il est suivi par le nom de la variable dans notre exemple il s'agit de **"maVariable"** juste

après nous assignons une valeur à la variable (ceci n'est pas obligatoire) la valeur peut avoir plusieurs types :

- String (chaîne de caractère)
 - o Exemple : `"let monString = 'toto';"`
- Int (nombre entier)
 - o Exemple : `"let monInt = 19;"`
- Bool (booléen vrai ou faux)
 - o Exemple : `"let monBool = false;"`
 - o Exemple : `"let monBool = true;"`
- Array (le tableau peut contenir plusieurs types)
 - o Exemple : `"let monArray = ['John', 'Deo', 'Mercredi'];"`

Il est aussi possible de déclarer une variable avec le préfix **"var"** mais cela n'est pas recommandé car qui a presque la même fonction que le **"let"** la différence entre les deux est que le var ne se limite pas au blocs. Exemple :

- Avec **"var"** :

```
if (true) {  
  var test = true;  
}  
  
alert(test); // vrai, la variable existe après if
```

- Avec **"let"** :

```
if (true) {  
  let test = true;  
}  
  
alert(test); // ReferenceError: test is not defined
```

5.2 Les Fonctions

Voici comment déclarer une fonction classique



```
function myFunction(parameter1, parameter2) {  
  // your code  
}
```

D'abord on met le mot clé "**function**" qui signifie que l'on fait une fonction ensuite on met le nom de la fonction et cela est suivi par des parenthèses où l'on met les paramètres de la fonction (il peut ne pas en avoir). Il existe un autre moyen de déclarer une fonction qu'on appelle des fonctions fléchées :



```
let myFunction = (parameter1, parameter1) => {  
  // Your code  
};
```


Pour pouvoir appeler une fonction il suffit d'écrire ça dans le code :



```
myFunction( )
```

5.3 Les Conditions :

En JavaScript il existe plusieurs façons de faire des conditions mais voici la plus répandue :



```
if (condition) {  
  // Your code  
}
```

Il existe aussi le "**else**" et le "**else if**" qui sont lus si la condition d'un "**if**" n'est pas respectée qui s'écrivent respectivement comme ça :



```
else {  
    //Your code  
}
```



```
else if(condition){  
    //Your code  
}
```

Il existe une autre façon de faire des conditions dont voici la syntaxe :



```
condition ? /*Code if the condition = true*/ : /*Code if the condition = false*/
```

Il existe une dernière façon de faire une condition : il s'agit du "**switch**" celle-ci de mieux comparer une valeur avec plusieurs variantes. Voici la syntaxe :



```
switch ([value to compare]) {  
  
    case [condition1]: // = if  
        //Your Code  
        break;  
    case [condition2]: // = else if  
        //Your Code  
        break;  
    default: // = else  
        //Your Code  
}
```

5.4 Les Boucles :

En JavaScript il existe des boucles qui exécutent du code un certain nombre de fois tant qu'une condition est respectée. Il en existe trois :

- Les boucles "**for**" ces boucles sont utilisées quand on sait le nombre de fois que le code va être exécuté. Voici la syntaxe:



```
for (let i = 0; [condition]; i++) {  
  //Your Code  
}
```

- Les boucles "**do...while**" ces boucles sont utilisé quand le nombre de fois que le code est exécuté est incertains et le code sera forcément exécuté une fois. Voici la syntaxe :



```
do {  
  //Your Code  
}  
While( [condition] );
```

- Les boucles "**while**" quant à elles c'est ce que le code ne va pas s'exécuté si la condition est fausse dès le départ



```
While( [condition] ) {  
  //Your Code  
}
```

6. Prendre tous les éléments d'un tableau :

6.1 ".forEach()" :

Le ".forEach()" est une fonction fléchée directement intégrée à JavaScript celle si s'utilise uniquement avec un tableau. Elle permet d'exécuter du code pour chaque élément du tableau. Voici la syntaxe :




```
myArray.forEach((element, <index>) => {  
    //Your Code  
})
```

Le paramètre "**element**" représente l'élément du tableau celui-ci est un paramètre obligatoire contrairement à "**index**" qui lui n'est pas obligatoire, il représente l'index du tableau.

6.2 ".some()" :

Le ".**some()**" est aussi une fonction fléchée directement intégrée à JavaScript. Cette fonction passe en revue tous les éléments d'un tableau et vérifie si l'élément respecte une condition. Voici la syntaxe :



```
myArray.some((n1) => ([condition]));
```

7. Les Classes :

Avant de pouvoir utiliser/coder les classes il est vivement conseillé de créer un fichier par classes.

7.1 Utiliser les classes dans le "main" :


Pour importer la classe il suffit d'utiliser l'élément "**import**" dans le script principal il suffit de mettre cette ligne :



```
import {ClassName} from '{path}';
```

7.2 Exporter les classes :

Pour pouvoir réaliser l'étape du dessus il suffit d'utiliser l'élément "**export default**" pour pouvoir réaliser cette étape il faut d'abord exporter ça classe. Cela se réalise avec cette ligne :



```
export default class myClass {  
  constructor([parameters]) {  
    // Value  
  }  
  
  // Your Code  
}
```

8. Conclusion

Ce projet m'a permis d'apprendre le JavaScript avec un thème qui me parle et qui me plait beaucoup, le plus dur a été de savoir comment faire faire bouger correctement le serpent car même si j'avais la théorie du JavaScript je ne savais pas encore comment bien le faire bouger. Mais suite à une longue discussion avec mes camarades on a réussi à trouver une solution pour le faire bouger de manière optimisé.

Pour ce qui est de la documentation dans le Mémo je pense avoir mis tout ce que je trouvais vraiment important et que je risquais d'oublier.

Pour ce qui est des critères qui ont été demandé dans le cahier des charges je pense les avoir respectés. Je pense que si le projet était à refaire je le referais avec grand plaisir mais cependant j'essaierais d'améliorer mon autonomie car c'est encore quelque chose que je n'arrive pas à régler du moins à minimisé durant les projets.

Pour résumer le projet je trouve qu'il s'est bien passé, j'ai acquis pas mal compétence qui me seront utile plus tard.