

Développement :

Le célèbre jeu du serpent, popularisé grâce à une grande marque de téléphones mobiles.

Vous pouvez le tester ici : <http://serpent.lencodage.fr/>

SssnaKe



Cadre :

Il s'agit d'un TP proposé dans le cadre du cours " Apprendre JavaScript ", dans le cadre de la formation en ligne Udemy " Formation complète Developpeur Web ".

Support :

JavaScript, à l'aide de l'IDE NetBeans.

Contraintes / Organisation :

Le pas-à-pas détaillé du développement était proposé.

Difficultés rencontrées :

Pas de difficultés particulières.

Améliorations proposées :

En plus de ce qui était proposé dans le cours, j'ai ajouté un titre ainsi qu'une icône pour l'onglet du navigateur. Enfin, il nous était demandé d'ajouter une fonctionnalité : toutes les 5 pommes mangées, le serpent doit avancer plus vite. J'ai aussi choisi de rajouter l'affichage de la vitesse sur le Canvas.

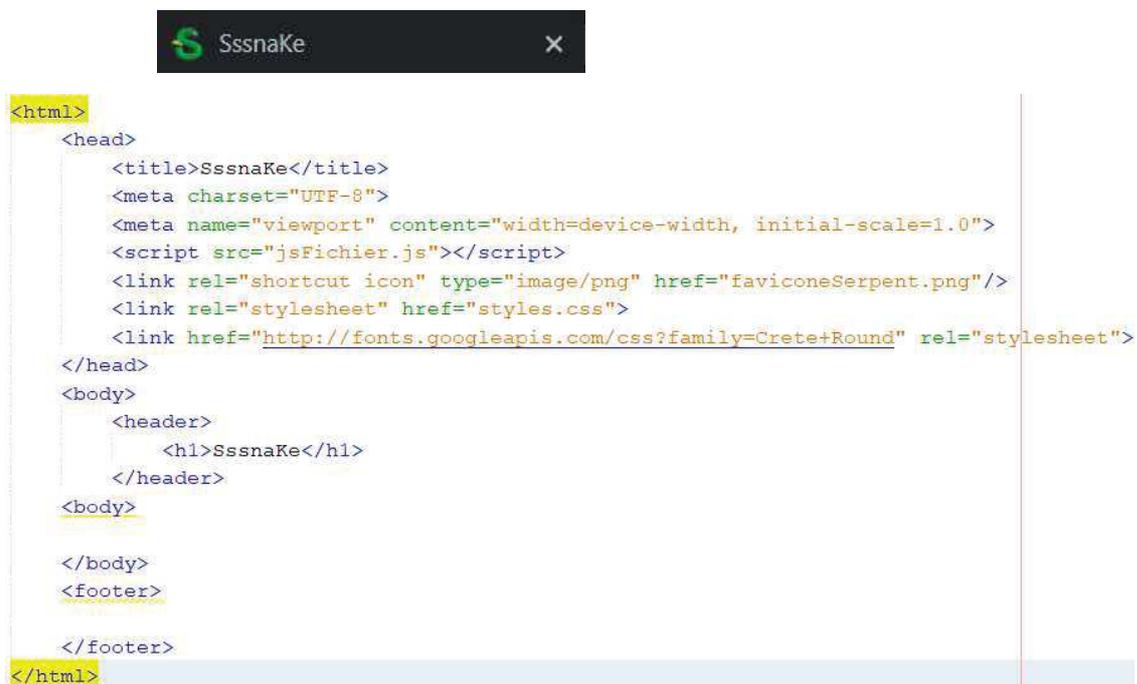
Description résumée du développement

Le développement s'est fait dans l'ordre suivant :

- création du Canvas
- rafraîchir le Canvas
- création du serpent
- diriger le serpent
- ajouter la pomme
- le serpent s'est-il pris un mur
- le serpent a-t-il mangé la pomme
- score et game over

1. Création du Canvas

Déjà, création d'une page HTML minimaliste. J'ai simplement placé ma petite icône serpent pour que ce soit joli



```
<html>
  <head>
    <title>SssnaKe</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <script src="jsFichier.js"></script>
    <link rel="shortcut icon" type="image/png" href="faviconeSerpent.png"/>
    <link rel="stylesheet" href="styles.css">
    <link href="http://fonts.googleapis.com/css?family=Crete+Round" rel="stylesheet">
  </head>
  <body>
    <header>
      <h1>SssnaKe</h1>
    </header>
  </body>
</body>
<footer>
</footer>
</html>
```

Et j'ai seulement mis un titre, ainsi bien sûr qu'un lien avec la page CSS pour la mise en page du titre, et surtout un lien avec la page JavaScript. C'est là que tout se passe en réalité, l'intégralité du code sera placée dans une fonction `window.onload()`, qui s'exécute lorsque la page html s'ouvre.

À l'initialisation du jeu, on crée le Canvas, avec quelques caractéristiques de style, qui est ajouté à la page grâce à `document.body.appendChild()`.

C'est ici qu'est initialisé un objet Snake (qu'on verra plus loin) ainsi qu'une pomme, et un score qui démarre à 0.

```
init(); // lancement du jeu

// Initialisation
function init() {
  var canvas = document.createElement('Canvas');
  canvas.width = canvasWidth;
  canvas.height = canvasHeight;
  canvas.style.border = "30px solid gray";
  canvas.style.margin = "50px auto";
  canvas.style.display = "block";
  canvas.style.backgroundColor = "#dcdcdc";
  document.body.appendChild(canvas);
  ctx = canvas.getContext('2d');
  snakee = new Snake([[6,4],[5,4],[4,4]],"right");
  applee = new Apple([10,10]);
  score = 0;
  refreshCanvas();
}
```

2. Rafraîchir le Canvas

Pour que le serpent avance, cela veut dire que l'affichage doit être rafraîchi, à certains intervalles de temps, à l'infini...ou jusqu'à ce qu'on perde.

Pour cela, on utilise la récursivité : une fonction *refreshCanvas()* est créée (et lancée à l'initialisation). Concrètement cette fonction : fait avancer le serpent (en réalité, modifie ses coordonnées, en fonction de la direction dans laquelle il se trouve), vérifie si on a perdu (heurté un mur ou heurté le corps du serpent), teste si le serpent a mangé une pomme (dans ce cas son score augment de 1 et sa vitesse peut augmenter, ainsi que la longueur de son corps), puis on dessine le score, la vitesse, le serpent et enfin la pomme.

Et pour finir, cette fonction s'appelle elle-même toutes les 500 millisecondes au début du jeu.

Voici la fin de cette fonction :

```
snakee.draw();
applee.draw();
timeOut = setTimeout(refreshCanvas,delay);
}
```

3. Création du serpent

Ce serpent a donc un corps (un tableau de coordonnées) et une direction en propriétés.

```
// Objet : Serpent
function Snake(body,direction) {
  this.body = body;
  this.direction = direction;
  this.ateApple = false;
  this.draw = function() {
    ctx.save();
    ctx.fillStyle = "#319e50";
    for (var i = 0; i < this.body.length; i++) {
      drawBlock(ctx, this.body[i]);
    }
    ctx.restore();
  };
};
```

4. Diriger le serpent

Pour cela, on a créer une méthode *advance()* qui utilise le switch pour modifier ses coordonnées.

```
this.advance = function() {
  var nextPosition = this.body[0].slice();
  switch(this.direction) {
    case "right" :
      nextPosition[0] += 1;
      break;
    case "left" :
      nextPosition[0] -= 1;
      break;
    case "up" :
      nextPosition[1] -= 1;
      break;
    case "down" :
      nextPosition[1] += 1;
      break;
    default :
      throw "Direction invalide";
  }
};
```

Par contre là on se contente d'ajouter une case à l'avant du serpent : il faut aussi bien sûr supprimer la dernière case, afin qu'il conserve la même taille, et donne l'illusion de se déplacer.

5. Ajouter la pomme

Là encore, c'est de la programmation objet : un objet apple qui a une certaine position, la méthode *draw()* pour la dessiner, la méthode *setNewPosition()* qui lui choisit une position aléatoire grâce à *random()*,

```
this.setNewPosition = function() {  
    var newX = Math.round(Math.random() * (widthInBlock - 1));  
    var newY = Math.round(Math.random() * (heightInBlock - 1));  
    this.position = [newX, newY];  
};
```

et enfin la méthode *isOnSnake()* afin de s'assurer que la pomme n'a pas été placée sur le serpent.

6. Le serpent s'est-il pris un mur ?

Où s'est-il cogné contre son propre corps ?

La fonction *checkCollision()* a été ajoutée à l'objet serpent afin de s'en assurer : il a suffit de vérifier que les coordonnées de la tête du serpent :

- étaient à l'intérieur du Canvas d'une part,
- et qu'elle ne se trouvait pas sur l'un des éléments du corps du serpent avec une boucle *for()*

```
var isNotBetweenVerticalWall = snakeY < minY || snakeY > maxY;  
  
if (isNotBetweenHorizontalWall || isNotBetweenVerticalWall) {  
    wallCollision = true;  
}  
  
for (var i = 0; i < rest.length; i++) {  
    if (snakeX === rest[i][0] && snakeY === rest[i][1]){  
        snakeCollision = true;  
    }  
}  
  
return wallCollision || snakeCollision;  
};
```

7. Le serpent a-t-il mangé la pomme ?

Voilà une question à ... 1 point. Très facile à vérifier puisqu'on a les coordonnées de la pomme, et celles de la tête du serpent :

```
this.isEatingApple = function(appleToEat) {  
    var head = this.body[0];  
    if (head[0] === appleToEat.position[0] && head[1] === appleToEat.position[1]) {  
        return true;  
    }  
    else {  
        return false;  
    }  
};
```

Du coup, maintenant qu'on sait dire si oui ou non il a mangé la pomme, on peut retourner dans *refreshCanvas()* pour écrire ce qui se passe dans ce cas : le score augmente, si le score est un multiple de 5, alors la vitesse augmente (j'ai plafonné pour que ça ne descende pas en-dessous de 1/10ème de seconde), et la pomme change de position :

```

if (snakee.isEatingApple(applee)) {
    snakee.ateApple = true;
    score += 1;
    if (score % 5 === 0 && delay > 100) {
        delay = delay - 100;
    }
    do {
        applee.setNewPosition();
    } while (applee.isOnSnake(snakee));
}

```

8. Score & Game Over

Arrivé là, il ne reste à faire que du dessin.

Puisque le jeu continue tant que le serpent n'est pas entré en collision avec un mur ou lui-même, de fait, dès qu'il y a en effet une collision le jeu s'arrête.



Il a donc fallu créer l'affichage des instructions, et c'est à ce moment là que j'ai ajouté l'affichage du score au milieu ainsi que celui de la vitesse du serpent en bas. Pour exemple, voici pour l'affichage du score :

```

// "Dessine" le score sur le fond
function drawScore() {
    ctx.save();
    ctx.font = "bold 200px sans-serif";
    ctx.fillStyle = "rgba(128,128,128,0.3)";
    ctx.textAlign = "center";
    ctx.textBaseline = "middle";
    var centreX = canvasWidth / 2;
    var centreY = canvasHeight / 2;
    ctx.fillText(score.toString(), centreX, centreY);
    ctx.restore();
}

```

Ensuite, pour relancer le jeu, la méthode *restart()* a été ainsi développée : on instancie un nouveau petit serpent, avec une pomme, le score est remis à 0, la vitesse est remise à 500 (variable de départ), et enfin on relance notre *refreshCanvas()* pour relancer le jeu.

```

// Relance le jeu
function restart() {
    snakee = new Snake([[6,4],[5,4],[4,4]], "right");
    applee = new Apple([10,10]);
    score = 0;
    clearTimeout(timeout);
    refreshCanvas();
}

```