# P\_Bulles DEV-PPT : Snake

## 

Table des matières

[P\_Bulles DEV-PPT : Snake 1](#_Toc212816030)

[1](#_Toc212816031)

[Introduction 3](#_Toc212816032)

[Étapes de développement 3](#_Toc212816033)

[Initialisation et déplacement du serpent 3](#_Toc212816034)

[Nourriture et croissance 4](#_Toc212816035)

[Gestion des collisions 4](#_Toc212816036)

[Score et chronomètre 4](#_Toc212816037)

[Gestion des contrôles et file d’attente des directions 5](#_Toc212816038)

[Boucle de jeu et rendu graphique 5](#_Toc212816039)

[Résultats et bilan 5](#_Toc212816040)

[Améliorations possibles 5](#_Toc212816041)

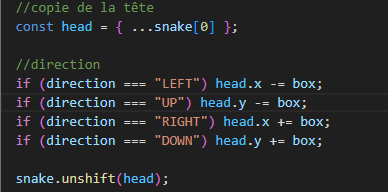
## Introduction

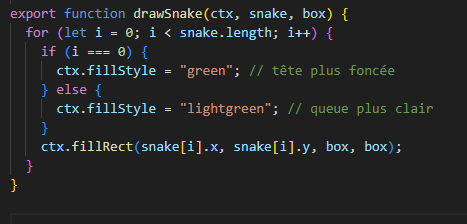
Le projet avait pour objectif de réaliser une version fonctionnelle du jeu Snake en JavaScript, avec affichage et interaction via un canevas HTML. La structure de fichiers était déjà en place (snake.js, food.js, controls.js, collision.js, score.js, main.js), et a été progressivement complétée afin d’implémenter toutes les fonctionnalités nécessaires : déplacement du serpent, gestion de la nourriture, collisions, score et amélioration des contrôles.

## Étapes de développement

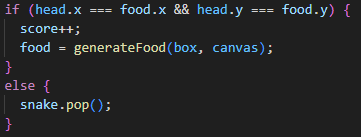
### Initialisation et déplacement du serpent

Le serpent est initialisé avec trois segments : une tête et deux éléments de queue. Chaque segment est représenté par des coordonnées x et y. Le déplacement est géré par la fonction moveSnake(), qui crée une nouvelle tête en fonction de la direction choisie et l’ajoute au début du tableau représentant le serpent :

Ce mécanisme permet de faire avancer la tête et de faire suivre le reste du corps.  
  
  
Gestion de la queue et distinction visuelle

Pour éviter que le serpent ne grandisse indéfiniment, la méthode snake.pop() a été ajoutée. Elle retire le dernier segment à chaque déplacement, créant l’illusion que la queue suit naturellement la tête.  
De plus, la tête a été distinguée visuellement grâce à une couleur verte plus foncée, tandis que les segments de la queue sont en vert clair.

### Nourriture et croissance

La nourriture est générée aléatoirement sur la grille grâce à generateFood(), puis affichée par drawFood(). Lorsqu’une collision entre la tête du serpent et la nourriture est détectée (lorsqu’ils partagent les mêmes cordonnées), snake.pop ne s’exécute pas et le serpent grandit, et une nouvelle pomme est générée  
  


Gestion des collisions  
Deux types de collisions ont été implémentés :

* Collision avec le corps :  
  La fonction checkCollision() parcourt le tableau du serpent (sauf la tête) et compare les coordonnées de chaque segment avec celles de la tête. Si une correspondance est trouvée, la fonction renvoie true, signifiant que le serpent s’est mordu la queue.
* Collision avec les murs :  
  La fonction checkWallCollision() détecte si la tête du serpent sort des limites du canevas. Si head.x ou head.y devient inférieur à 0 ou supérieur ou égal à la taille du canevas, la partie s’arrête.

En cas de collision, la fonction endGame() est appelée : elle efface le canevas, arrête la boucle du jeu et affiche le score ainsi que le temps écoulé.

### Score et chronomètre

Deux systèmes d’affichage ont été ajoutés pour rendre le jeu plus interactif.

Le score :  
Chaque fois que le serpent mange une pomme, la variable score augmente d’une unité. La fonction drawScore() affiche ce score dans le coin supérieur gauche du canevas.

Le chronomètre :  
Un minuteur a été ajouté pour afficher le temps de jeu en secondes grâce à startTimer(), pauseTimer(), resumeTimer() et stopTimer().  
Le temps est affiché en haut à droite avec drawTimer().

Ces ajouts permettent au joueur de suivre sa performance en temps réel.

Gestion des contrôles et file d’attente des directions  
Les commandes de direction (flèches du clavier) sont gérées par handleDirectionInput() et handleDirectionChange().  
Une file d’attente de directions (directionQueue) a été mise en place pour éviter les blocages lorsque le joueur appuie rapidement sur plusieurs touches. Cela permet une meilleure réactivité et évite que le serpent ne fasse demi-tour instantanément, ce qui entraînerait une collision immédiate.

De plus, la touche Espace permet de mettre le jeu en pause ou de le reprendre. Pendant la pause, un écran semi-transparent s’affiche avec l’inscription "Pause".

Boucle de jeu et rendu graphique  
La boucle principale, gérée par setInterval(draw, gameSpeed), appelle régulièrement la fonction draw().  
Cette fonction met à jour la direction du serpent, déplace la tête, gère la nourriture, détecte les collisions et redessine l’ensemble du jeu (serpent, pomme, score, temps).

Le canevas est effacé à chaque cycle avec ctx.clearRect(), assurant une animation fluide et continue.

Résultats et bilan  
Le jeu est entièrement fonctionnel, avec un serpent fluide et réactif, une gestion correcte des collisions, un score et un chronomètre visibles, ainsi qu’une mise en pause et une reprise sans bug.

Ce projet a permis de maîtriser plusieurs aspects fondamentaux du JavaScript, notamment la manipulation du DOM et des événements clavier, le dessin sur un canevas 2D, la gestion d’un état de jeu dynamique et la séparation logique du code en modules clairs et réutilisables.

Améliorations possibles  
Pour aller plus loin, plusieurs pistes d’amélioration peuvent être envisagées :

* Ajouter un menu d’accueil et un bouton "Rejouer" sans recharger la page.
* Intégrer un système de niveaux où la vitesse augmente progressivement.
* Enregistrer le meilleur score localement (localStorage).
* Ajouter des effets sonores pour la nourriture ou les collisions.
* Donner au serpent une apparence plus travaillée (arrondis, dégradés, sprite).