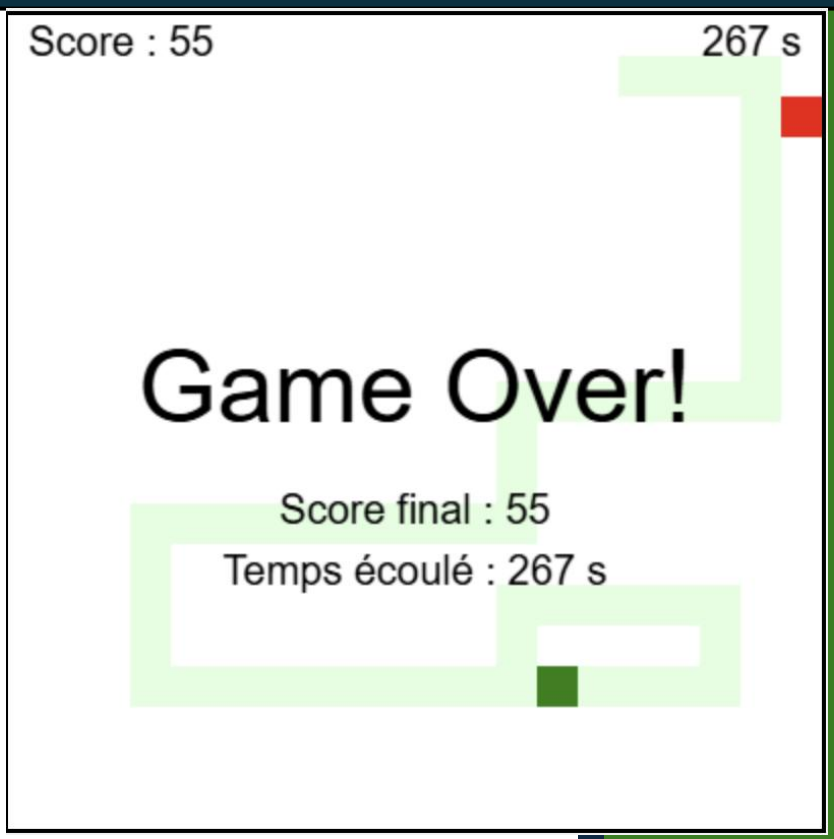
2025

# SNAKE GAME

Projet Personnel




Jessica GrisalesB

FID2

15/09/2025

# Table des matières

# Introduction

Lors du module P_Bulle un défi a été confié, la réalisation d'un classique et incontournable jeu du serpent. Ce projet devra être réalisé en utilisant le JavaScript natif pour le navigateur et en suivant les instructions données dans le cahier des charges.

Ce réplica devra être dessinée sur un canvas HTML, le css pour le style et le code JavaScript utilisera la syntaxe des modules ES (ECMAscript Modules, ESM). Les commentaires du code devront être écrit en JSCODE.

L'utilisation du serveur de développement sera gérée par Vite qui possède des atouts qui mettra en valeur la rapidité et une configuration minimale.

La structure du projet a été donné préalablement, cette mise en forme est imposée. Il n'est pas autorisé de la supprimer. Celle-ci compte avec une architecture modulaire qui facilite la compréhension grâce à sa clarté, à la distinction et séparation entre les diverses fonctionnalités.

## Planification

La structure de travail pour mener à bien la réalisation du jeu snake, se fera à l'aide d'un journal de travail qu'on doit veiller à qu'il soit à jour (il est mise à jour directement dans le fichier du module dans teams). De plus, l'outil GitHub sera également un outil indispensable pour suivre l'avancé du projet.

## Analyse de la structure modulaire

Premièrement avant de commencer, il est important de connaître et comprendre le rôle et la fonction de chaque module. Le projet possède le module snake qui sert à dessiner, initialiser, mettre en mouvement et dessiner le serpent. Puis, le module score est assez simple, il permet de compter et d'afficher le score. En ce qui concerne le module food, celui-ci est intéressant car il est chargé de dessiner et de générer la nourriture. Le module controls aura comme particularité donner la direction de mouvement au serpent grâce aux touches directionnelles du clavier. Et pour finir, le module collision, consiste à vérifier si le serpent a bien touché l'une des limites du carré du canvas ou son propre corps.

## Environnement de développement

Le jeu Snake sera lancé directement dans le navigateur web, afin de permettre son affichage de manière correcte, la configuration du serveur est capitale. Il faut bien qu'il soit sur **le port : 3000**.

Lors de l'installation et configuration de cet environnement, Une erreur empêchait de voir le jeu sur le navigateur. Le problème était lié au fait que l'exécution du fichier HTML dans visual code était ouvert directement avec **open with live server**. Il ne donnait pas alors directement accès au port 3000. Donc, une fois avoir vérifier que le server tournait bien à la racine du fichier, le lancement de ce dernier avec **npm run dev** montrait le lien http du jeu pour voir son affichage sur le navigateur.

La raison pour laquelle il n'est pas possible d'exécuter le jeu sans le lien http, est dû au modules EMAScript (ESM) (<script type="module">). Les navigateurs actuels bloquent le chargement des modules locaux pour des raisons de sécurité (CORS et origin null). C'est pourquoi il est conseillé d'utiliser le server de développement Vite.

Procédure de lancement (**Vite**) : Installation des dépenses avec **npm install**, lancement du server avec **npm run dev**, et pour finir http://localhost:3000/.

## Conception du projet

Premièrement la prise de connaissance du fichier HTML car l'information qui est représentée donne le point de départ.

```
<canvas id="gameCanvas" width="400" height="400"></canvas>
<script type="module" src="./src/main.js"></script>
```

### 1. Création du code du module snake et food :

Cette partie est fondamental pour réussir à afficher le premier carré de nourriture et la tête du serpent. En premier lieu, il faut créer la fonction initSnake qui sera le point de départ du serpent. Grâce au tableau d'objet {Array<{x: number, y: number}>} qui représente un carré du corps. Ensuite, la fonction drawSnake sert à enfin dessiner le serpent dans le canvas. Mais ce n'était pas si évident, car trouver le moyen d'assigner la couleur vert au carré (tête) à cause de la syntaxe spécifique a été difficile de mettre en place.

```
function drawSnake(ctx, snake, box) {
  // A compléter
  for (let i = 0; i < snake.length; i++) {
    ctx.fillStyle = (i === 0) ? "green" : "■#E0FFE0";
    ctx.fillRect(snake[i].x, snake[i].y, box, box);
  }
}
```

Pour la partie nourriture (food), Il fallait afficher un carré rouge cette fois-ci. Mais avec la spécificité que le carré de nourriture devait s'apparait aléatoirement dans le canvas. C'est pourquoi, l'option d'utiliser random est plutôt pertinente. Cependant, dans ce cas, réaliser la syntaxe adéquate pour que ce soit valide a également été compliqué. Maintenant, il ne manque plus qu'attribuer la couleur rouge et donner la position dans le canvas ainsi que la taille du carré grâce à cette fonction.

```
function drawFood(ctx, food, box) {
  // A compléter
  ctx.fillStyle = "red";
  ctx.fillRect(food.x, food.y, box, box);
}
```

Une fois que les modules food et snake sont prêts pour donner vie au jeu, il faudra utiliser la fonction draw dans le module main qui sera expliqué plus tard.

### 2. Création des directionnelles (controls)

Cette partie est fondamental pour permettre au serpent de changer de direction et que ce soit jouable. Il est donc important de créer la fonction handleDirectionChange(event, currentDirection). Cette partie est plus simple et courte à construire, toutefois la problématique de savoir et trouver qu'il fallait utiliser keyCode grâce à Event n'a pas été si évident. Une fois la variable let key = event.keyCode; mis en place, le reste du code était devenu plus simple à élaborer.

```
function handleDirectionChange(event, currentDirection) {

  let key = event.keyCode;
  if (key === 37 && currentDirection !== "RIGHT") return "LEFT";
  if (key === 38 && currentDirection !== "DOWN") return "UP";
  if (key === 39 && currentDirection !== "LEFT") return "RIGHT";
  if (key === 40 && currentDirection !== "UP") return "DOWN";
  return currentDirection;
}
```

### 3. Définition des collisions

**Condition pour le serpent :**

Etablir les conditions de collisions qui permettent de savoir ce qui va se passer lorsque le serpent touchera les bords du canvas ou son propre corps. Actuellement, sans une définition adéquate le serpent aura tendance à continuer son parcourt même s'il touche son propre corps. C'est pourquoi, qu'il faut créer une fonction conditionnelle comme checkCollision(head, snakeArray) pour que la partie s'arrête lorsque le serpent se touche lui-même.

```
function checkCollision(head, snakeArray) {
  // A compléter
  for (let i = 0; i < snakeArray.length; i++) {
    if (head.x === snakeArray[i].x && head.y === snakeArray[i].y) {
      return true;
    }
  }
  return false;
}
```

La difficulté lors de la création se cette condition, a été de comprendre qu'en JavaScript pour donner une égalité il faut utiliser un triple égale.

**Condition pour le carré de jeu:**

Maintenant, il faudrait aussi définir les conditions de collisions lorsque le serpent touche les bords du canvas, grâce à la fonction checkCollision(head, snakeArray). Cette fonction est importante, car pour l'instant le serpent ne détecte pas les bords et continue son chemin en dehors du canvas. L'élaboration de cette fonction a été compliquée car il fallait utiliser le .width et .height pour qu'il prenne en compte la longueur et largeur du canvas.

```
function checkWallCollision(head, canvas, box) {
  // A compléter
  return (
    head.x < 0 || head.y < 0 || head.x >= canvas.width || head.y >= canvas.height
  );

}
```

## 4. Mise en place du score et du temps dans le module score.js

Cette partie a été plutôt simple a réalisée, il s'agissait plus de la définition du style du point de vu graphique. Comme dans le cas du serpent et de la nourriture, l'usage de ctx suivie de la méthode qui donne la forme et la position sur le canvas. Dans ce cas, l'affichage du score sera en haut à gauche et pour le temps en haut à droite. Le style de police, la taille et la manière de les représenter dans le carré de jeu sont construits dans cette partie du projet.

```
function drawScore(ctx, score, timeSecond) {
  // 1. Affichage du Score (en haut à gauche)
  ctx.fillStyle = "black";
  ctx.font = "20px Arial";
  ctx.textAlign = "left";

  ctx.fillText("Score : " + score, 10, 20);

  // 2. Affichage du Temps (en haut à droite)
  ctx.textAlign = "right";

  // Le temps est affiché 10px avant le bord droit
  ctx.fillText(timeSecond + " s", ctx.canvas.width - 10, 20);
}
```

### 5. Import dans le main

Tous les fichiers exposés précédemment possède une instruction export qui va permettre d'être importé et réutilisé dans le main qui est le fichier central. Cette façon de faire, garde un ordre et une structure bien définie du projet. Maintenant le main aura accès à toutes les méthodes déjà créés.

```
import { initSnake, moveSnake, drawSnake } from "./snake.js";
import { generateFood, drawFood } from "./food.js";
import { handleDirectionChange } from "./controls.js";
import { checkCollision, checkWallCollision } from "./collision.js";
import { drawScore } from "./score.js";
```

Dans cette partie, il est fondamentale d'ajouter le document.getElementById(gameCanvas) et le canvas.getContext("2d") qui vont donner l'autorisation d'accéder au canvas dans le HTML et préparer un contexte de dessin 2D pour pouvoir dessiner dessus.

```
const canvas = document.getElementById("gameCanvas");
const ctx = canvas.getContext("2d");
```

**La fonction draw() :**

L'objectif est de pourvoir afficher et initier le jeu, pour cela la méthode draw() est principale. Mais avant de commencer avec cette méthode, il faut en premier lieu définir

les modalités initialisations du jeu avec la méthode starGame(). Elle fera la mise en place de comment va initialiser le jeux, comme la position de la tête du serpent, le score, l'affichage de game over et de pause. L'ajout de if(gameInterval) clearInterval(gameInterval) va vérifier si la boucle gameInterval est déjà en cours, si c'est le cas, elle l'arrête. Puis, (intervalTime) clearInterval(intervalTime); pour nettoyer le timer de l'ancienne partie.

En ce qui concerne le timer, la pause, game over et les intervalles, ont été ajouté et modifié vers la fin du projet.

```
function startGame() {
  snake = initSnake(box);
  food = generateFood(box, canvas);
  score = 0;
  gameOver = false;
  Paused = false;

  // Démarrer le chronomètre
  gameTime = Date.now();
  timeSecond = 0;

  if(gameInterval) clearInterval(gameInterval);
  if(intervalTime) clearInterval(intervalTime);

  draw();

  gameInterval = setInterval(draw, gameSpeed);
  intervalTime = setInterval(updateTime, 1000);
}
```

La fonction draw est la plus complexe à réaliser et celle qui a subit le plus de modifications pour permettre que tout puisse s'emboiter correctement et que ce soit jouable. Il fallait premièrement arriver à exécuter l'affichage de la tête du serpent, la nourriture et le score et sans doute le game over.

Pour ce faire, il a fallu faire appel aux fonctions drawSnake, drawFood, drawScore et checkCollion, etc. Pour commencer à tester la jouabilité et corriger les éventuelles erreurs ou bug. Toutefois, dans le cas de ce projet une erreur a été constaté.

**Détection des collision un carré avant :**

La fonction détectait la collision de la tête du serpent des bords du cadre canvas un carré avant, et lors de l'affichage game over et fin de la partie, cette fois-ci, la tête du serpent touchait bien les bords. En effet, le problème venait de l'ordre d'exécution dans

la fonction draw() avec une mauvaise utilisation de la fonction alert(). Ce qui ce passait était que le jeu s'arrêtait immédiatement après la détection de la collision et affichait alert (game over), et clearInterval empêchait le prochain cycle de jeu de dessiner. Mais, la position finale n'était jamais dessinée dans le cycle courant avant l'arrêt, figeant sur l'état valide précédent.

**Solution** adoptée, le bloc collision qui vérifie et arrête je jeu a été déplacé avant la logique de mise à jour du corps. Donc, alert("Game Over ! Score final : " + score); a été remplacé par déclaration de l'état : let gameOver = false; et drawGameOver() { … ctx.fillText("Game Over!", …); } et if (gameOver) { drawGameOver(); return; }

**Modification de Game Over :**

Après vérification que le jeu fonctionne et l'erreur déjà traité ne persiste pas, l'ajout de l'affichage Game Over écrit sur le canvas et ne plus sous forme d'alert comme fait précédemment, dans la fonction draw(). Cette fonction a été simple de réaliser car il s'agissait plus de donner un style et l'endroit à afficher, très similaire à ce qui a déjà été dans le cas du score et le compteur entre autre.

```
function drawGameOver() {
    ctx.fillStyle = "black";
    ctx.font = "50px Arial";
    ctx.textAlign = "center";
    ctx.fillText("Game Over!", canvas.width / 2, canvas.height / 2);

    ctx.font = "20px Arial";
    ctx.fillText("Score final : " + score, canvas.width / 2, canvas.height / 2 + 50);
    // Affichage du temps final
    ctx.fillText("Temps écoulé : " + timeSecond + " s", canvas.width / 2, canvas.height / 2 + 80);
}
```

**Création du Chronomètre (Timer) :**

Dan le cas du timer, l'initialisation de trois variables différentes a été nécessaire, comme gameTime, timeSecond et intervalTime. Ici, une erreur a été commise en utilisant au début la même intervalle gameInterval que dans le cas du serpent. Cette pratique n'a pas était retenu car des bugs ont surgit.

Maintenant, c'est la nouvelle fonction updateTime() qui sera chargé de calculer la différence entre le temps actuelle (Date.now()) et l'heure de départ (gameTime) et bien sûr de mettre à jour le compteur timeSecond.

```
function updateTime() {
  if (Paused || gameOver)

    return;

  // Calculer le temps écoulé depuis le début du jeu en secondes
  const now = Date.now();
  timeSecond = Math.floor((now - gameTime) / 1000);
}
```

Lors du démarrage (startGame()), la variable timeSecond sera initialisé à 0. Le temps sera également figé quand on fait pause et repart d'où il était avec la reprise de la partie. Mais, il a fallu spécifié que le timer devait s'arrêter complétement quand la collision

```
function startGame() {
  snake = initSnake(box);
  food = generateFood(box, canvas);
  score = 0;
  gameOver = false;
  Paused = false;

  // Démarrer le chronomètre
  gameTime = Date.now();
  timeSecond = 0;
```

aura lieu. C'est pourquoi, pour un bon fonctionnement, il y a eu un ajout de timesecond dans la fonction drawGameOver().

```
  // Affichage du temps final
  ctx.fillText("Temps écoulé : " + timeSecond + " s", canvas.width / 2, canvas.height / 2 + 80);
}
```

**Mise en place de l'option pause :**

Toujours dans le main, la première chose a été de créer une variable qui sera utilisée dans la nouvelle fonction pauseTouch(). En réalité, la variable paused sera définit à false lorsque le jeux sera actif, mais du moment où l'on appuie sur la touche espace l'interval gameInterval et timerInterval se fige et appel ensuite la fonction draw pour afficher

"PAUSE" sur l'écran, et vise versa pour reprendre la partie.

```
function pauseTouch() {
  if (gameOver) return;

  Paused = !Paused;

  if (Paused) {
    clearInterval(gameInterval); // SUSPENDRE L'INTERVALLE du mouvement
    clearInterval(intervalTime); // SUSPENDRE L'INTERVALLE du chronomètre
    draw(); // Dessine une dernière fois pour afficher le message PAUSE
  } else {
    // Reprendre le chronomètre en ajustant le temps de départ
    gameTime = Date.now() - (timeSecond * 1000);

    // Redémarrer les intervalles
    gameInterval = setInterval(draw, gameSpeed);
    intervalTime = setInterval(updateTime, 1000);
  }
}
```

Afin de créer cette fonctionnalité, le plus important est de modifier le document.addEventListener on ajoutant cette condition ci-dessous :

```
document.addEventListener("keydown", (event) => {
  // 32 est le code de la touche Espace
  if (event.keyCode === 32) {
    pauseTouch(); // Appelle la fonction de pause/reprise
  } else if (!Paused && !gameOver) {
    // Ne change la direction que si le jeu n'est ni en pause ni terminé
    direction = handleDirectionChange(event, direction);
  }
});
```

En suite, il a fallu préciser que la pause devait aussi inclure l'état du serpent de la nourriture, tu temps et du score.

```
function draw() {

  ctx.clearRect(0, 0, canvas.width, canvas.height);

  // 1. GESTION DE LA PAUSE : Si le jeu est en pause, on affiche le message et on sort.
  if (Paused) {
      drawSnake(ctx, snake, box);
      drawFood(ctx, food, box);
      drawScore(ctx, score, timeSecond); // Doit passer le temps
      drawPause();
      return;
  }
```

Puis, la mise en forme du style et la manière dont la pause s'affichera sur l'écran :

```
function drawPause() {
    ctx.fillStyle = "rgba(0, 0, 0, 0.1)";
    ctx.fillRect(0, 0, canvas.width, canvas.height);

    ctx.fillStyle = "white";
    ctx.font = "50px Arial";
    ctx.textAlign = "center";
    ctx.fillText("PAUSE", canvas.width / 2, canvas.height / 2);
}
```

## Commentaires JSCODE

Pour ce projet, les commentaire devaient respecter cette syntaxe. Il y avait déjà la plus grosse partie faite, car le projet n'était pas vide et avait au départ une structure à respecter. En effet, les modules comptaient avec un style de commentaire uniforme et structuré inspiré JSCODE. Ce dernier permet de décrire les fonctions, les paramètres et les valeurs de retour de manières standardisée. Les commentaire inline expliquent le fonctionnement étape par étape du code, ce qui améliore la lisibilité et la maintenabilité.

## Conclusion

Dans cette section je vais donner un avis plus personnel. J'ai pris du plaisir à réaliser ce projet, cependant je considère que j'ai dû apprendre seule car les cours ou supports était inexistants. Ce que je trouve dommage, du fait que c'était la première fois qu'on voyait du javascript dans la formation. Il est agréable d'avoir une certaine autonomie pour la réalisation du projet et de l'apprentissage de ce langage. Mais, le manque de renforcement dans les bases pour s'assurer de bien comprendre pas prise en compte.

## Usage de l'IA

L'intelligence artificielle a été utilisée pour comprendre comment fonctionne le javascript car aucune théorie ou support nous a été fourni. Il y a beaucoup d'informations sur internet et spécialement sur le site javascript.info qui enseigne les bases avec quelques exemples. Toutefois, pour ma part ce n'est pas assez et j'ai décidé d'utiliser l'IA pour quelle me génère des exemples plus parlant et de me les expliquer. Sans cet outil, il aurait été vraiment difficile de finir ce projet en si peu de temps.