2025

Jessica GrisalesB

FID2

15/09/2025

SNAKE GAME

Projet Personnel

****

Table des matières

[Introduction 2](#_Toc210638211)

[Planification 3](#_Toc210638212)

[Analyse de la structure modulaire 3](#_Toc210638213)

[Environnement de développement 3](#_Toc210638214)

[Conception du projet 4](#_Toc210638215)

[Accessibilité 4](#_Toc210638216)

[Conclusion 4](#_Toc210638217)

[Usagede l’IA 4](#_Toc210638218)

## Introduction

Lors du module P\_Bulle un défi nous 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). 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 imposé. Il n’est pas autorisé à la supprimer. Celle-ci compte avec architecture modulaire qui facilite la compréhension de chaque élément et donne un ordre car chaque fonctionnalité est distinctement déparée.

## Planification

La structure de travail pour mener à bien la réalisation du jeux snake, se fera à l’aide d’un journal de travail qu’on doit veiller à qu’il soit à jour. 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 afficher le score. En ce qui concerne le module food, celui-ci est intéressant car il est charger 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.

## Environnement de développement

Le jeux 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.

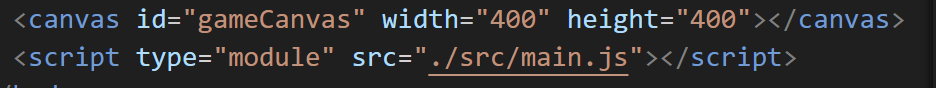
Lors de l’installation et configuration de cet environnement, Une erreur empêchait de voir le jeux sur le navigateur. Le problème était lié au fait que l’exécution du fichier HTML directement avec open with live server ne donner pas directement accès au port 3000. Donc, une fois vérification que le server tournait bien à la racine du fichier, le lancement de ce dernier avec npm run dev montrait le lien http du jeux pour voir son affichage sur le navigateur.

La raison pour laquelle il n’est pas possible d’éxécuter le jeux 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.



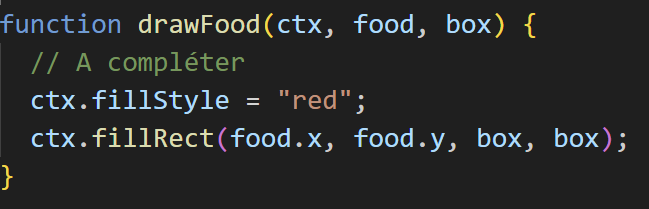
1. **Création du code du module snake et food :**

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) dû à la syntaxe spécifique a été difficile de mettre en place.

Une image contenant texte, capture d’écran, Police

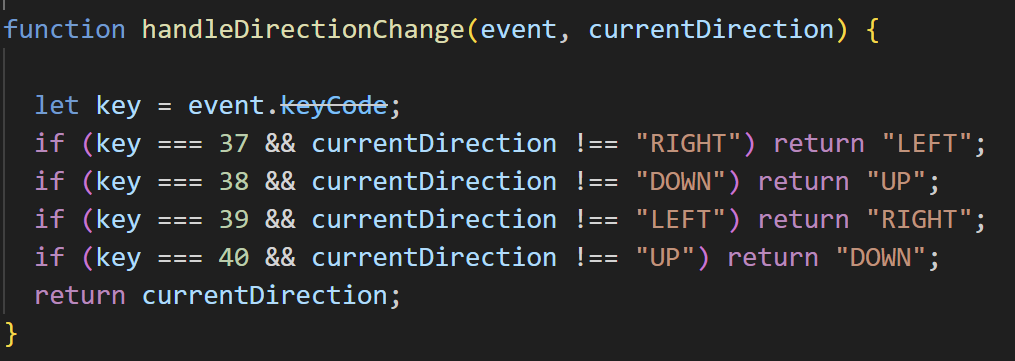
Le contenu généré par l’IA peut être incorrect.

En ce qui concerne 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 difficile. 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.



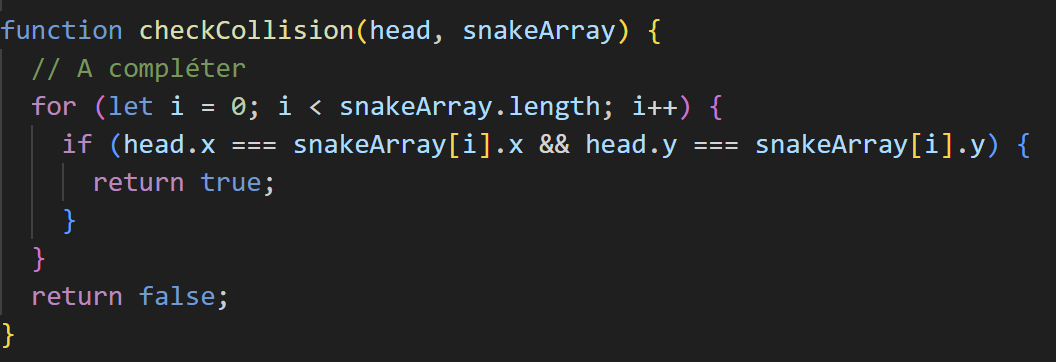
Une fois que les modules food et snake sont prêts pour donner vie au jeux, il faut compter avec la fonction draw dans le module main qui sera expliqué plus tard.

1. **Création des directionnelles (controls)**

Cette partie est fondamental pour changer le serpent 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. 

1. **Définition des collisions**

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 alors qu’il faut créer une fonction conditionnelle comme checkCollision(head, snakeArray) pour que la partie s’arrête lorsque le serpent se touche lui-même.



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.

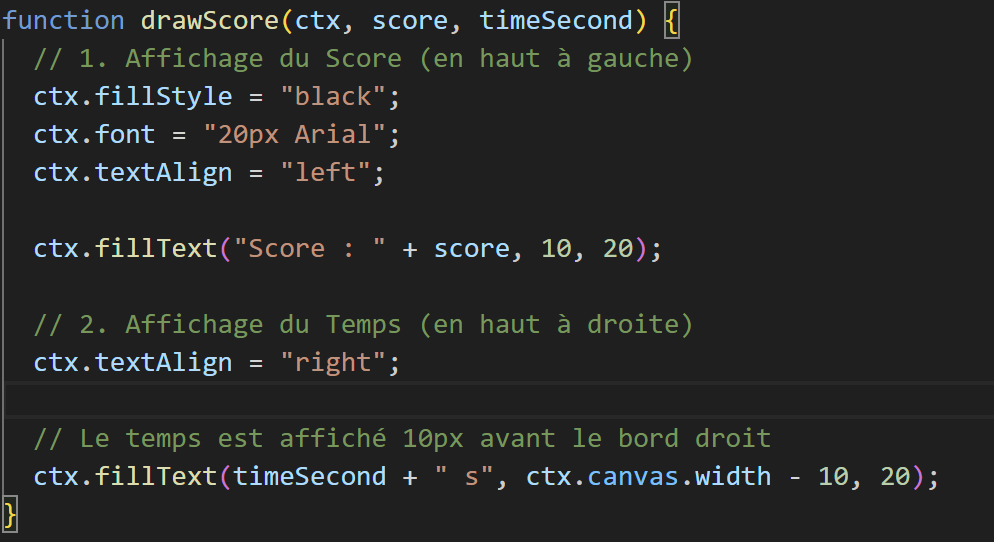
Maintenant, il faudrait aussi définir les conditions de collisions lorsque le serpent touche les bords du canvas. Grâce à la fonction checkCollision(head, snakeArray). Cette fonction est importante car pour l’instant le serpent ne détecte pas les bords et continue son chemin en dehors du canvas. L’élaboration 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.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

1. **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.



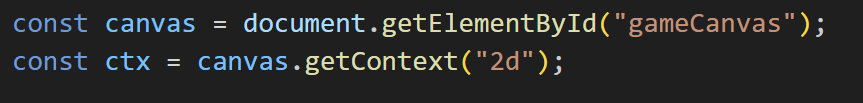
1. **Connecter tout ces méthodes dans le main grâce à import**

Tous les fichiers exposés précédemment possède une instruction export qui va permettre d’être importer et réutiliser 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.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

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.



L’objectif est de pourvoir afficher et initier le jeux, pour cela la méthode draw() est principale. Mais avant de commencer avec cette méthode, il faut en premier lieu définir les modalités initialisations du jeux 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.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

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.

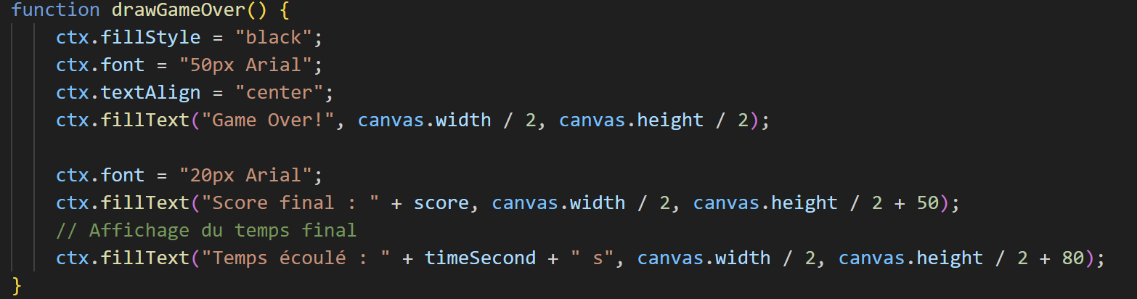
Pour ce faire, il a fallu faire appel aux fonctions drawSnake, drawFood, drawScore et checkCollion, etc. Pour commencer à tester la jouabilité et corriger les éventuelles erreurs ou bug. Toutefois, dans le cas de ce projet une erreur a été constaté.

**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 éltait 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 jeux 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; }

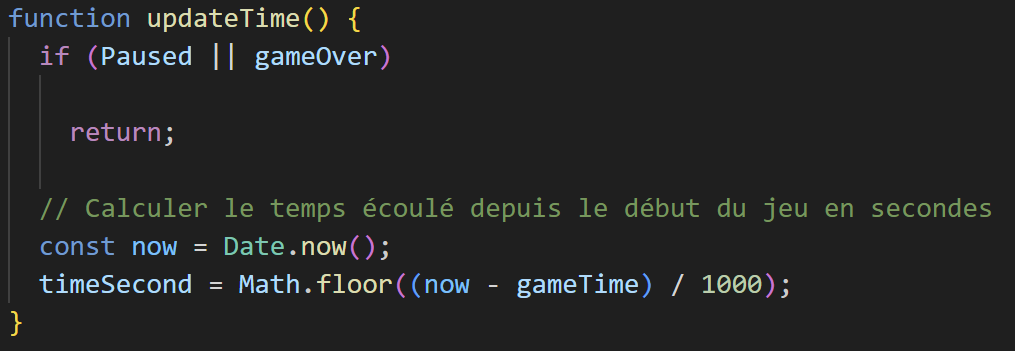
**Modification de Game Over :**

Après vérification que le jeux 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().

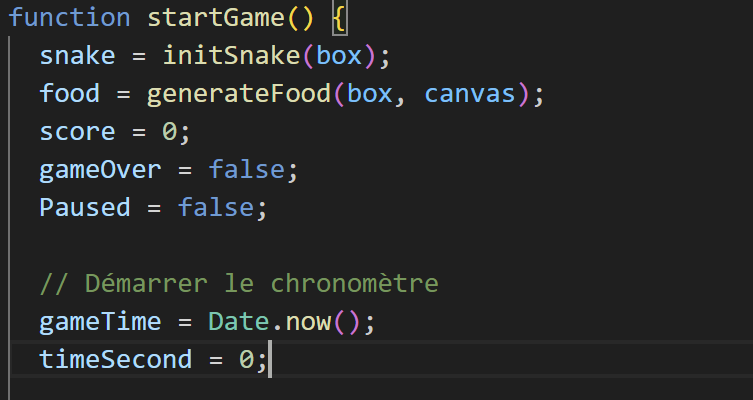
**Création du Chronomètre (Timer) :**

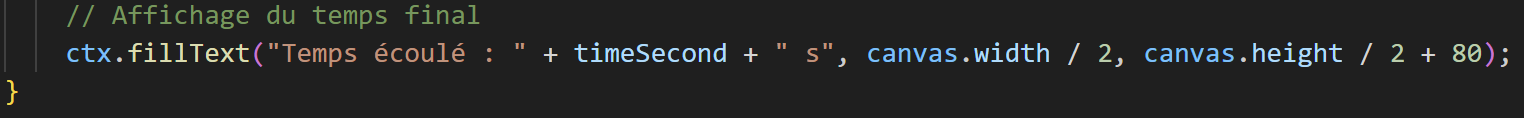
Dan le cas du timer, l’initialisation de trois variables différentes a été nécessaire, comme gameTime, timeSecond et intervalTime. Ici, une erreur a été commise en utilisant au début la même intervalle gameInterval que dans le cas du serpent. Cette pratique n’a pas était retenu car des bugs ont surgit.

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.

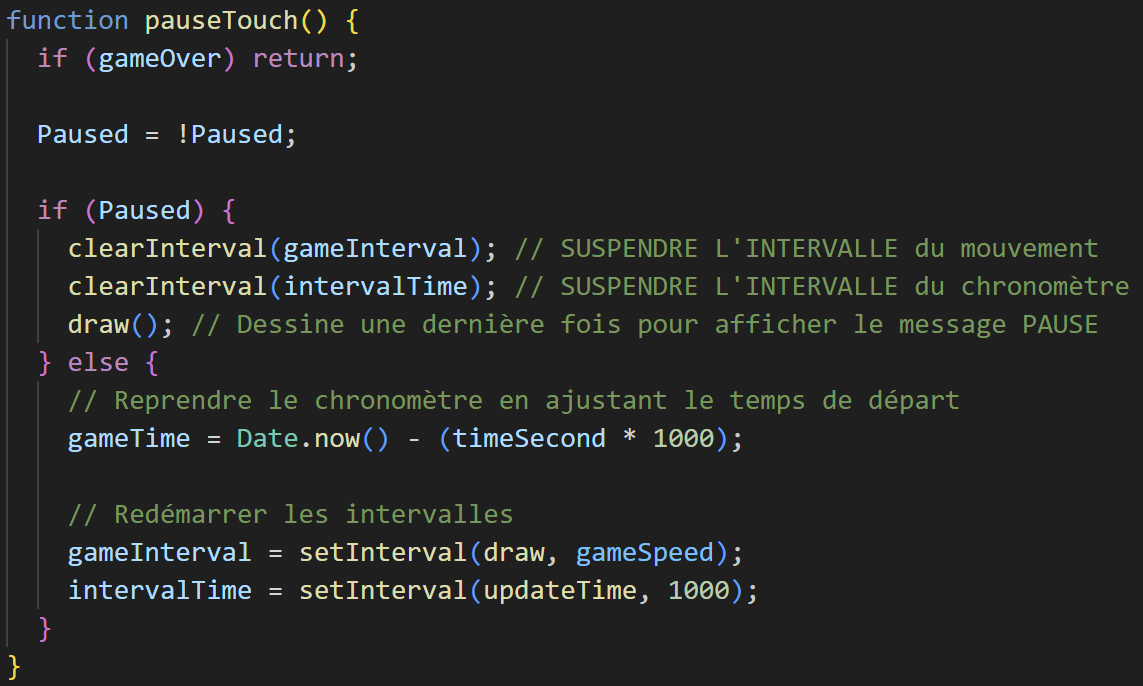


Lors du démarrage (startGame()), la variable timeSecond sera initialisé à 0. Le temps sera également figé quand on fait pause et repart d’où il était avec la reprise de la partie. Mais, il a fallu spécifié que le timer devait s’arrêter complétement quand la collision

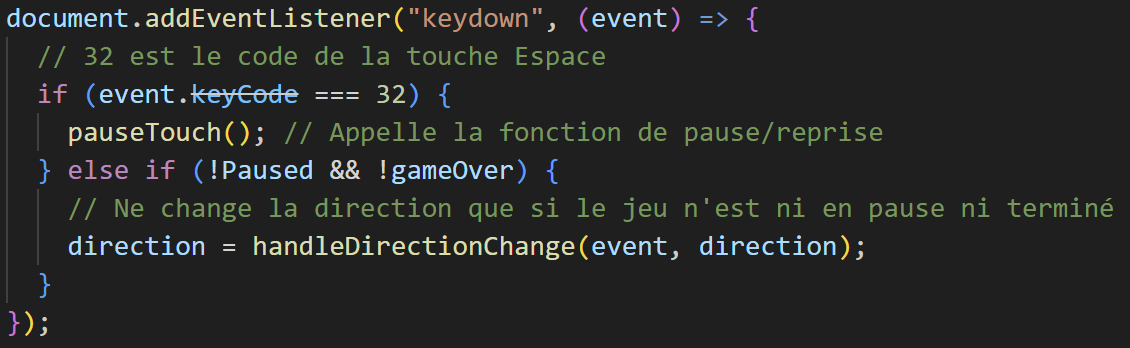
aura lieu. C’est pourquoi, pour un bon fonctionnement, il y a eu un ajout de timesecond dans la fonction drawGameOver().



**Mise en place de l’option pause lorsqu’on appuie sur la touche espace :**

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.

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

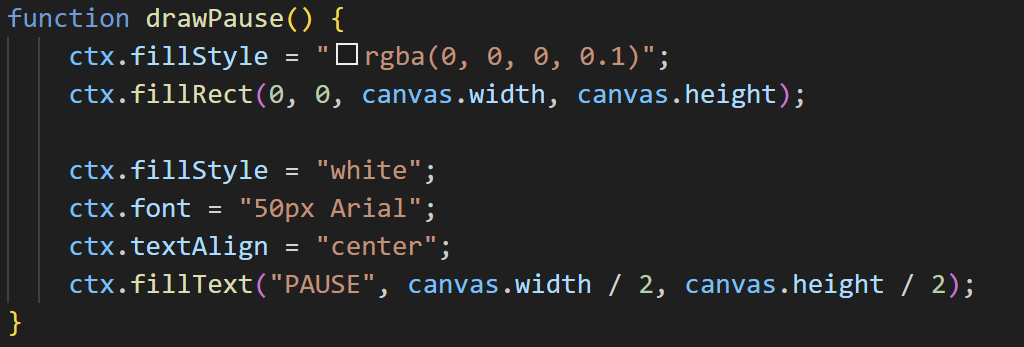


En suite, il a fallu préciser que la pause devait aussi inclure l’état du serpent de la nourriture, tu temps et du score.

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

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



## Accessibilité

## Conclusion

## Usagede l’IA