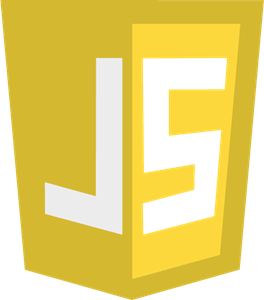
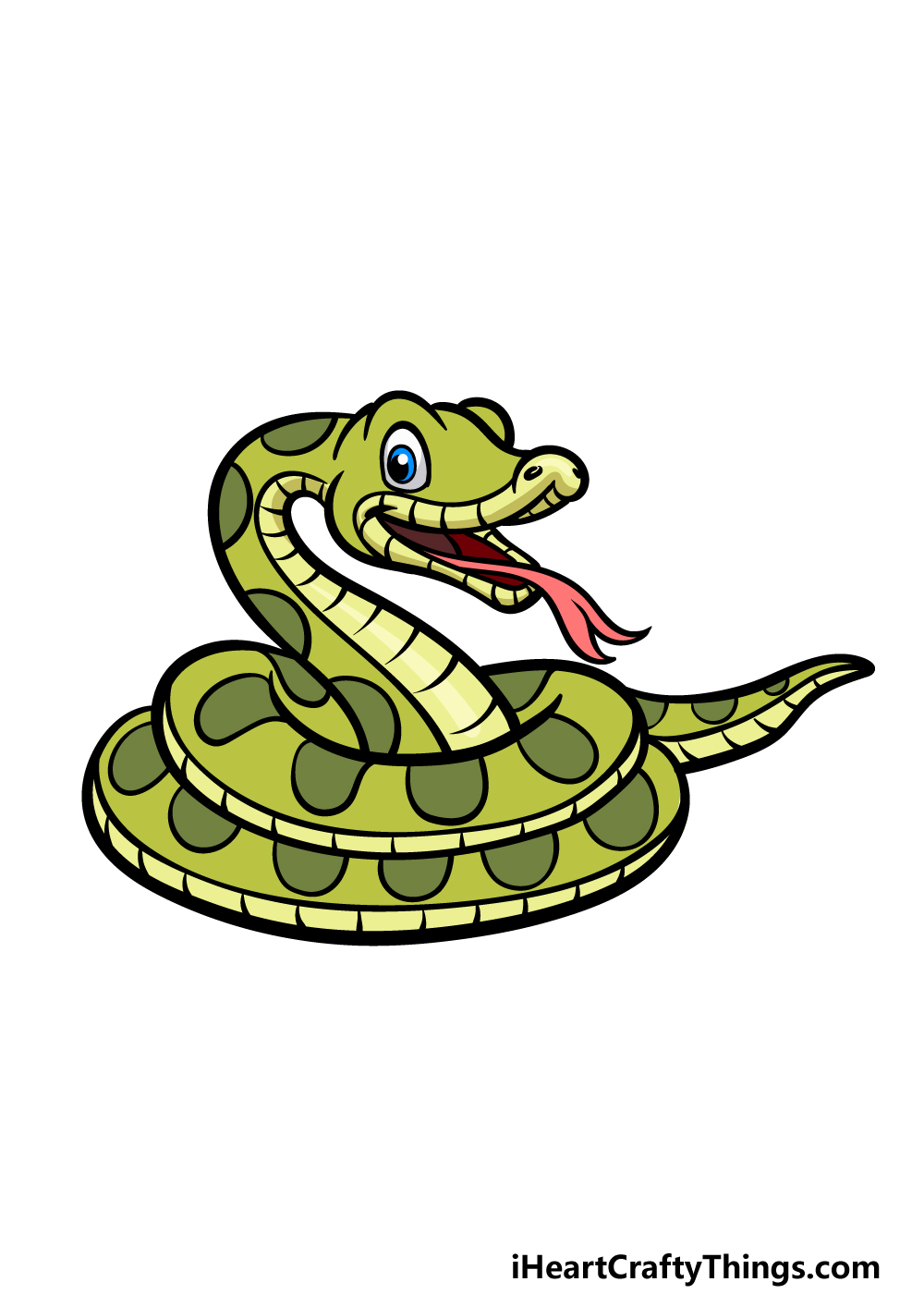
Support de cours

Snake - P\_Bulles

Cyril Napoleone





# Introduction

Avant de commencer le code concrètement j’ai dû faire plusieurs étapes préalables que voici (L’entièreté de ce support de cours se déroule dans l’ordre chronologique de mon travail):

* Installation VM linux
* Mise en place Node.js
* Arrangements pour le bon fonctionnement du programme (live server)

## VM linux

Le professeur nous a fourni une machine virtuelle déjà prête (quelques modifications nécessaires pour que l’import fonctionne) puis j’ai installé le tout sur mon SSD pour pouvoir l’utiliser n’importe où.

La VM a déjà Visual Studio Code d’installé et est entièrement configurée.

## Installation Node.js

J’ai installé Node.js sur linux via le terminal, nous (la classe) avons eu un souci lié au fait que la marche à suivre datait d’autres versions. Nous avons utilisé cette référence pour l’installer avec succès : <https://blog.crowncloud.net/post/installing-node-js-on-ubuntu-23-10/>

## Arrangements additionnel

Il y’a eu un autre souci qui a rendu le commencement du concret plus long que prévu, le fichier html ne prenait pas en compte le fichier js, j’ai réglé le problème en utilisant l’extension « live server » qui permet d’héberger localement et temporairement son projet. Mais nous avons découvert que plusieurs résolutions du problème étaient possibles (dossiers partagés, vite, etc..).

# Familiarisation avec JavaScript

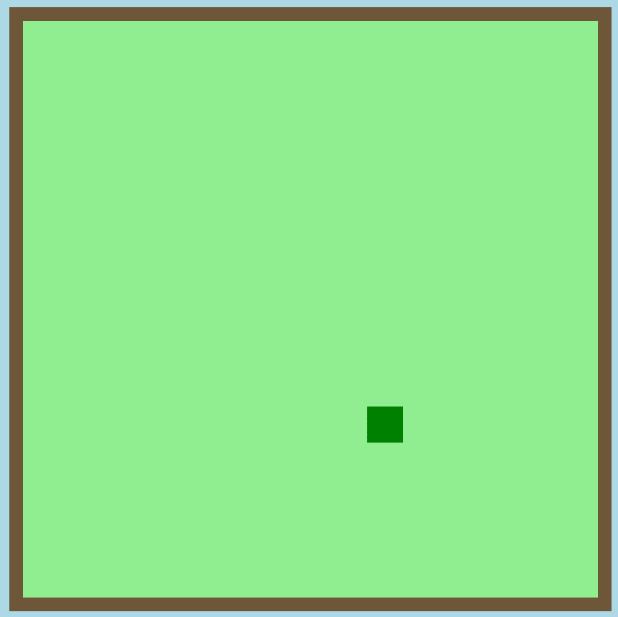
## Familiarisation seul

Étant donné que je n’ai jamais utilisé JavaScript avant ce projet et que nous avons pas de

théorie en parallèle à ce cours, j’ai décidé de me familiariser seul avec le langage avant de me renseigner en ligne.

J’ai donc touché au réglages du code que nous avons par défaut, j’ai compris comment il marchait et à quoi il servait puis comment afficher des choses, les variables, un peu de style et quelques fonctionnalités.

Je suis très vite arrivé à un résultat car j’ai réussi à faire un carré qui se déplace seul et qui s’il atteint la limite du terrain, apparait de l’autre côté de celui-ci (et cela dans tous les sens). Sans avoir encore utilisé internet.



## Documentation internet

Le point à partir du quel j’ai senti nécessaire de commencer à me documenter sur des forums, sites, etc… est quand j’ai voulu pouvoir contrôler mon carré (qui se déplaçait déjà) avec mes flèches plutôt que de devoir changer le code manuellement pour voir des changements de directions.

<https://stackoverflow.com/questions/5597060/detecting-arrow-key-presses-in-javascript>

J’ai donc implémenter ce que j’ai appris (lien ci-dessus) et maintenant je peux me déplacer comme je veux sur mon terrain avec les limites fonctionnelles (quand je touche une flèche, elle change la variable direction qui elle est utilisé dans le déplacement. Par ex : si direction = 1 alors le carré va à droite jusqu’à changement de la variable direction)

Maintenant que j’ai déjà pris mes marques avec ce langage je peux commencer à faire les parties + complexe du code (système serpent, points, pommes, etc…) et puis l’utilisation des classes.

# Code

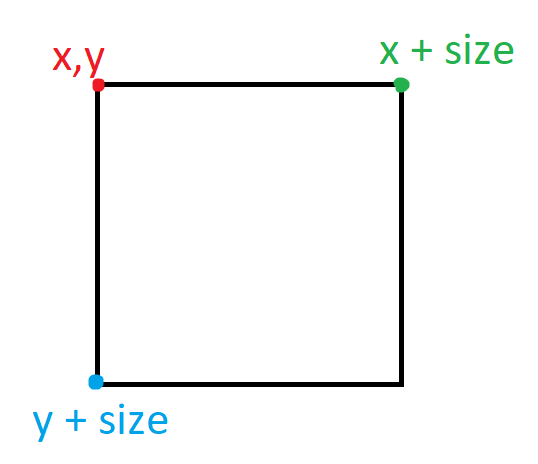
Avant d’ajouter des fonctionnalités j’ai bien pris en considération le CDC. J’ai donc, avant tout, fait en sorte de comprendre (via <https://www.w3schools.com/js/js_classes.asp> ) et utiliser les classes pour pouvoir y mettre mon serpent et pour que je me familiarise déjà avec ce concept avant qu’il soit trop tard (car en C# POO j’ai voulu tout convertir en orienté objets après avoir fini le code sur une page, ce qui a été une erreur car j’ai eu beaucoup de mal).

J’ai réussi sans trop de soucis puis me suis penché sur le fonctionnement de l’aléatoire en JavaScript car il est un peu différent du C#.

<https://www.w3schools.com/js/js_random.asp>

Grâce à cette page, j’ai pu comprendre son fonctionnement et l’utiliser pour l’apparition de ma pomme, qui d’ailleurs m’a posé un souci après ça.

Le problème était que je voulais lui attribuer une nouvelle position quand mon carré la touche (manger la pomme) sauf que mes détections de collisions ne marchaient pas. J’ai donc fais un schéma rapide sur Paint pour mieux visualiser ce qui devait être pris en compte.



Car effectivement, ma position x et y représente seulement le coin en haut à gauche du carré, donc tout le reste et une combinaison de variables (position et taille du carré). Grâce à ça j’ai pu créer un if comportant 4 conditions :

if (snake.x < apple.x + apple.size && snake.x + snake.size > apple.x && snake.y < apple.y + apple.size && snake.y + snake.size > apple.y)

Maintenant, dès que mon carré touche la pomme, cela est détecté, elle réapparait autre part aléatoirement et le score est incrémenté.

Après une discussion avec monsieur Cheneaux sur le système actuel de mon jeu, j’ai décidé de changer l’aspect « mouvements libres » de mon Snake, car oui je pouvais bouger sans effet de « quadrillage », c’est-à-dire le serpent et la pomme ne s’emboitaient pas parfaitement par exemple.

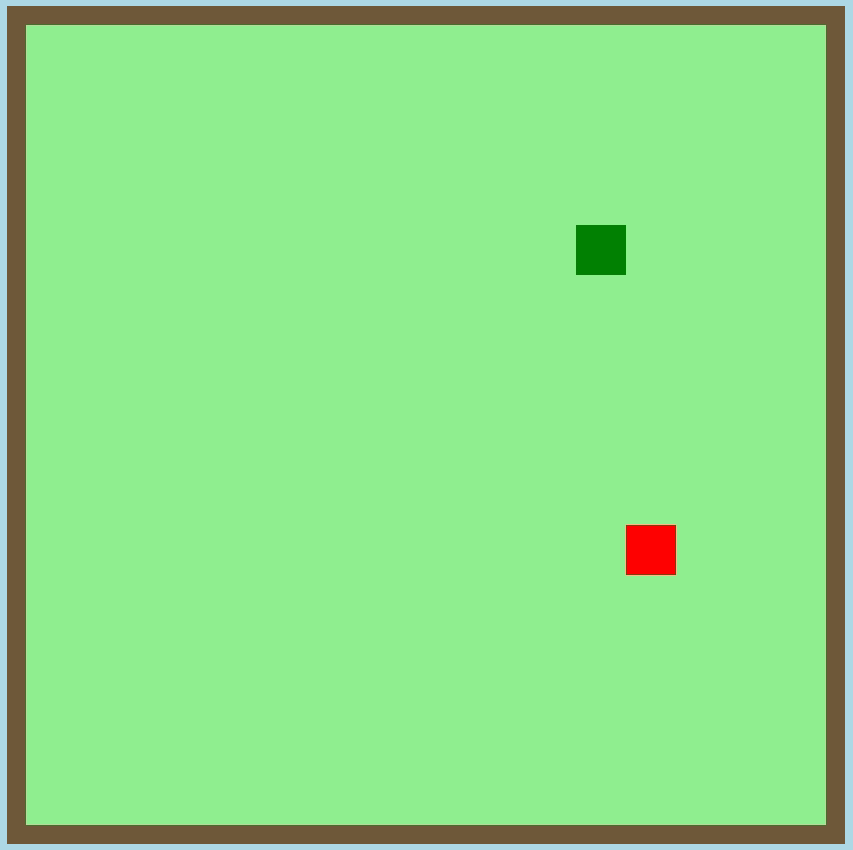
Et au lieu de régler cet aspect plus tard comme proposé par le prof, j’ai préféré faire ça tout de suite.

J’ai simplement trouvé un équivalent du Thread.Sleep (du C#) en JavaScript via [cette page](https://www.sitepoint.com/delay-sleep-pause-wait/#:~:text=intrusive%20sleep%20method%3A-,function%20sleep(ms)%20%7B%20return%20new%20Promise(resolve%20%3D,%2C%20then%20log%20%E2%80%9CWorld!%E2%80%9D), que j’ai ajouté à mon code juste avant chaque mouvement\* (que j’ai incrémenté pour faire des « bons » plus gros ).

Puis, pour que la pomme apparaisse toujours à un endroit qui concorde aux déplacements de 50 par 50 du serpent, j’ai repris un système (avec quelques adaptations) que j’avais fait en collaboration avec Yohan pour son code à lui mais que je n’avais pas implémenté au miens.

Il consiste à utiliser un tableau de valeurs (dans mon cas, les multiples de 50, de 0 à 750 (limites terrain)) et piocher aléatoirement une case du tableau comme position x ou y. J’ai adapté ce principe pour en faire des fonctions qui retournent donc une position aléatoire quand je veux, ce qui a rendu cette partie plutôt simple.

\*Au final, j’ai supprimé la fonction sleep que j’avais créée car j’ai découvert et utilisé la fonction global « [setTimeout](https://developer.mozilla.org/en-US/docs/Web/API/setTimeout)» parce que elle est incorporée de base et est plus simple.

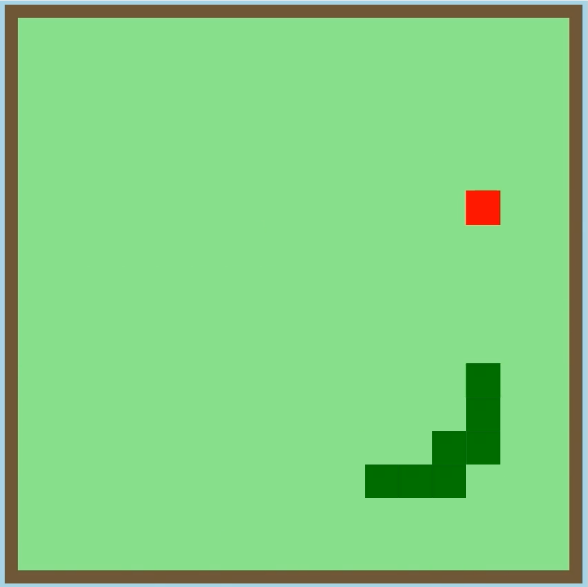


Le carré rouge (la pomme) réapparait à une nouvelle position aléatoire dès qu’elle est touchée par le carré vert (le serpent). Le score est aussi incrémenté de 1 à chaque pomme mangée (score pas encore affiché).

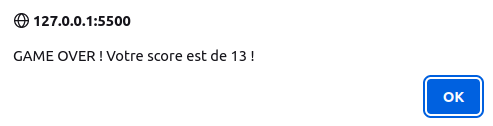
Après ça je me suis attaqué au système d’agrandissement du serpent.

Un système de tableau est ce qui me paraissaient le plus logique et simple, j’ai donc ajouté un tableau d’objets « segments » aux paramètres de ma classe Snake, les objets de ce tableau ont 2 propriétés qui sont x et y, pour avoir la position sur le terrain de chaque partie du serpent.

Ensuite pour ce qui est de l’agrandissement de celui-ci, le système consiste en ceci : Le serpent ajoute un segment au début du tableau (tête du serpent) à chaque avancement, et pour que il garde sa longueur, il supprime la dernière case du tableau sauf si la fonction Eat() est reconnue, ce qui assure qu’il s’agrandit bien seulement au contact d’une pomme.



Comme dit plus tôt dans ce support ([ici](#_Familiarisation_seul)), lorsque le serpent atteint la limite du terrain, il réapparait de l’autre côté de celui-ci, ce qui est possible dans certains jeu Snake mais n’est pas très fidèle à sa version originale et surtout à l’exemple donné dans les spécifications du projet. Alors j’ai simplement remplacé le résultat de la condition de collision aux limites du terrain par une alerte game over avec le score atteint (ci-dessous) suivi d’un arrêt du programme.

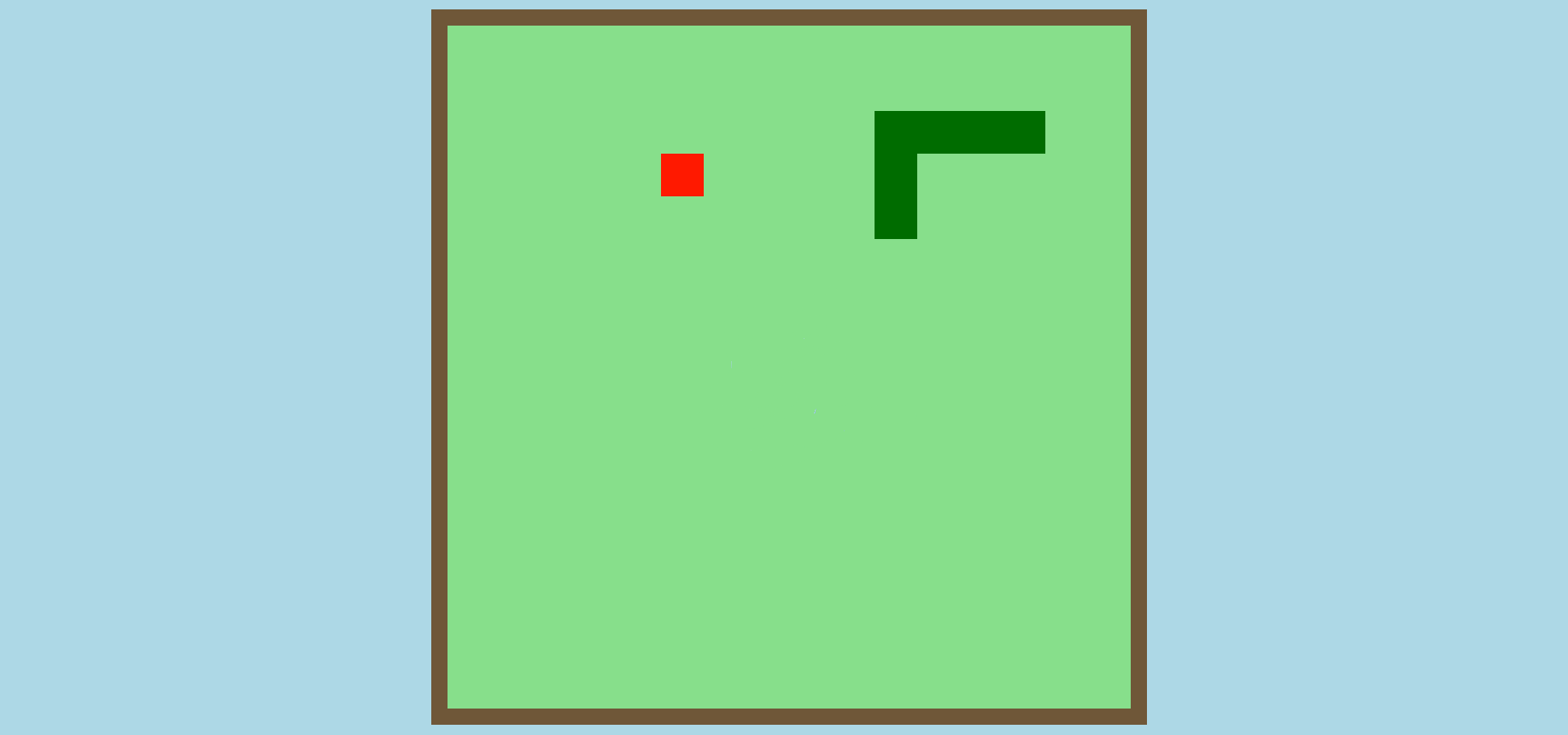


Maintenant que les fonctionnalités « principales » du jeu sont faites, j’ai pris un peu de temps à rendre mon jeu plus esthétique.

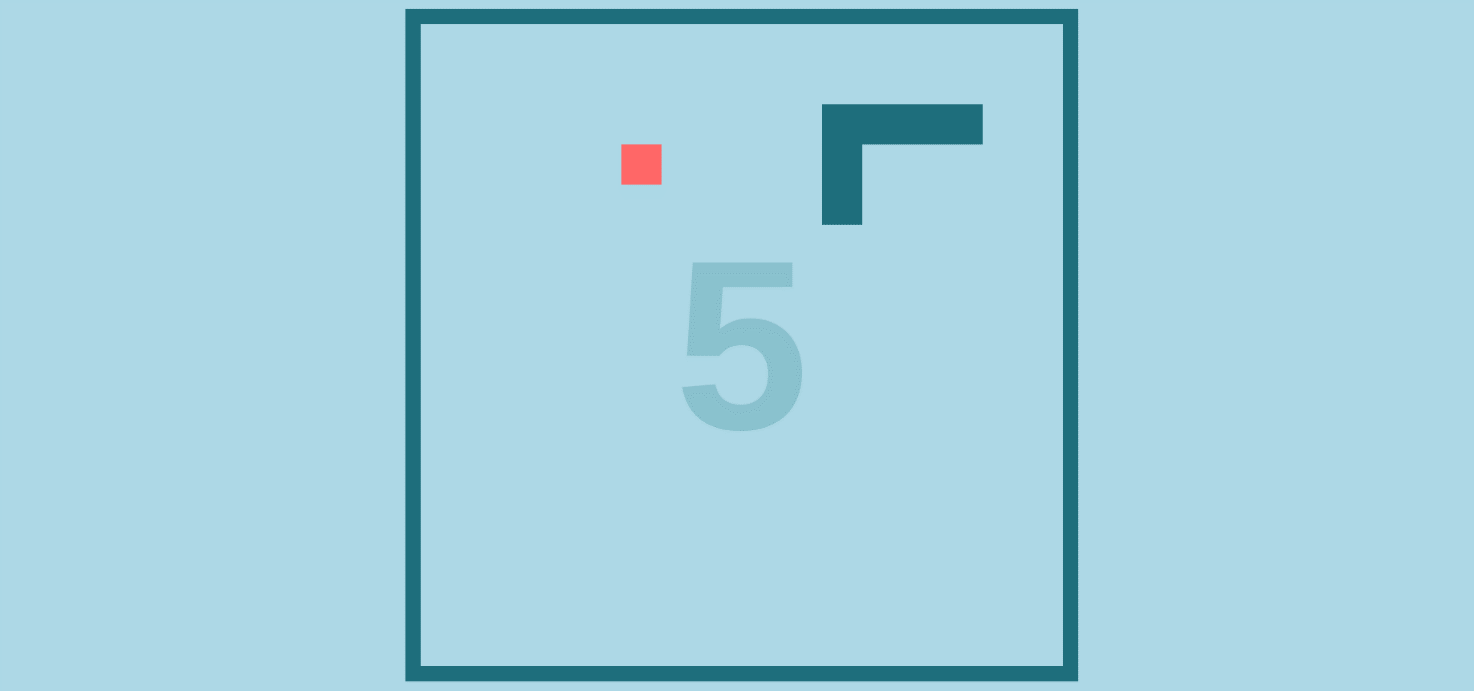
Pour ma couleur de thème, j’ai sélectionné la couleur que j’avais déjà choisie pour le fond au début du jeu (simplement « lightblue » qui revient à rgb(173, 216, 230)). À partir de cette couleur, j’ai choisie comme bordure une version plus sombre de ce bleu, que j’ai fini par utiliser pour le serpent aussi, pour finir j’ai mis mes pommes en un rouge pastel qui va avec le bleu ce qui donne vraiment ce thème singulier à mon jeu et le rend plus moderne et esthétique comparé à ma version de base.

J’en ai profité pour enfin affiché le score en temps réel sur le jeu. J’ai créé un paragraphe avec l’id « score » sur mon code html qui est relié à mon javascript via un getElement(). Un peu de style dans le css et maintenant le score et affiché en faible opacité au milieu du terrain.

**Avant**



**Après**



Plus j’avance dans mon code, plus il commence à se faire long. Bien que je fasse attention à qu’il soit bien organisé (classes, fonctions, variables, etc..), il devient tout de même lourd à lire et à travailler avec. Je me suis donc lancé dans le scindement de cette même page en 3 différentes (main, snake et apple). Je me suis alors renseigné sur l’import et l’export de données (principalement [ici](https://bobbyhadz.com/blog/javascript-import-class-from-another-file)) puis comment l’adapter à ma situation avec mes classes, mes fonctions et mes variables.

Mon code est maintenant bien plus lisible est léger.



Pour faire en sorte qu’une pomme ne puisse pas apparaître sur une case que le serpent occupe, j’ai cherché sur internet mais ai fini par m’aider de ChatGPT car manque de réponses satisfaisantes ailleurs. J’ai découvert le [.some()](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array/some) et en ai conclu que cette méthode fonctionnera pour ma situation.

J’ai alors fais un système qui réitère la génération de position tant que le nombre est égal à une position d’un segment du serpent ce qui a fonctionné (a finalement pris + de temps que prévu).

Après avoir fini la fonctionnalité ci-dessus, j’ai eu l’idée d’utilisée la méthode que je venais de découvrir (.some()) pour faire mon système de collision du serpent avec lui-même. Car je peux utiliser le même principe avec quelques adaptations : si la position de la tête est égale à n’importe quelle position de segment du serpent, alors considérer que le serpent s’est touché lui-même.

J’ai fait une boucle for qui vérifie la position de chaque segment SAUF le segment 0 (la tête) car évidemment elle sera toujours égale à elle-même.