**PROJET P\_BULLE**



VELICKOVIC Mateja

école des métiers techniques lausanne

Création d’un réplica du célèbre jeu Snake en JavaScript

**Chef de projet :** CHENAUX Patrick

Sommaire  
VELICKOVIC Mateja

[1 Démarrage](#_Toc115727280)  1

[1.1 Initialisation du projet 1](#_Toc115727282)

[1.2 Installation des prérequis 1](#_Toc115727282)

[1 Explication](#_Toc115727280)  1

[2.1 Démarrage](#_Toc115727281) 1

[2.2 Terrain de jeu & Développement](#_Toc115727281) 2

[2.3 Objectif](#_Toc115727281) 3

[3 LOREM IPSUM](#_Toc115727280) 4-5-6

[3.1 Lorem ipsum 4](#_Toc115727281)

[3.2 Lorem ipsum 5](#_Toc115727282)

[3.3 Lorem ipsum 6](#_Toc115727282)

**PROJET P\_BULLE**

1. Initialisation du projet

Pour mener à bien ce projet, un environnement de codage correct est nécessaire, pour cela, notre dossier qui contient le code a été mis en localhost:5173 afin d’avoir un aperçu du JavaScript lorsqu’on programme, Visual Studio a été utilisé comme éditeur de texte avec une extension nommée Live Server afin d’avoir un rafraichissement en direct, tout le projet (sauf documentation) a été réalisé sur une machine virtuelle Ubuntu 22.04.3 LTS\* importée à l’aide d’un fichier OVA fournie par l’enseignant.  
*\*Versions d'Ubuntu qui obtiennent un traitement privilégié en matière de stabilité et de soutien, une version stable d'Ubuntu sur quatre est considérée LTS.*

1. **Installation des prérequis**

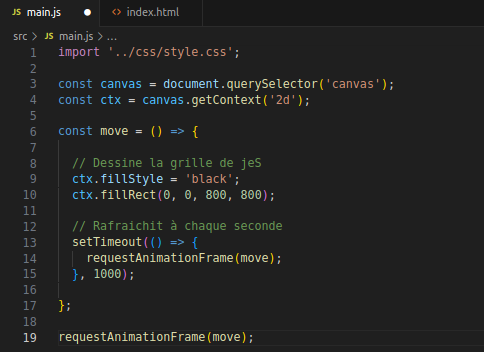
Afin de pouvoir mettre en local notre jeu, différentes manipulations ont dû être réalisées dans le terminal : en tout premier lieu il a fallu ouvrir l’invite de commandes en se plaçant dans le dossier de travail souhaité, ensuite, un **SUDO APT UPDATE** a été effectué afin de vérifier que tous les paquets installés sont bien à jour, une fois cela fait nous allons démarrer la mise en place de notre jeu en local.

Entrez ensuite cette commande afin d’installer Node Version directement depuis l’invite de commande **SUDO WGET -QO- HTTPS://RAW.GITHUBUSERCONTENT.COM/NVM-SH/NVM/V0.39.5/INSTALL.SH | BASH.**

Les deux premiers mots de cette commande nous indiquent que l’on souhaite installer le contenu à partir de l’URL spécifiée, avec la commande suivante nous allons installer **NPM**, qui est un gestionnaire de paquets pour l’environnement d’exécution en **JAVASCRIPT NODE.JS**: **SUDO APT INSTALL NPM –Y**.

Nous allons ensuite installer une version précise, qui est, la v21.1.0 de **NODE.JS**, pour connaître toutes les versions disponibles nous allons entrer cette ligne : **NVM LS-REMOTE**, et allons ensuite installer la version citée précédemment : **NVM INSTALL 21.1.0**.

L’avant dernière étape est l’installation de vite via : **NPM INSTALL VITE**, et puis finalement, nous allons lancer l’hébergement en local de notre programme en entrant, simplement ceci: **NPM RUN DEV**.

**2.1 Démarrage**

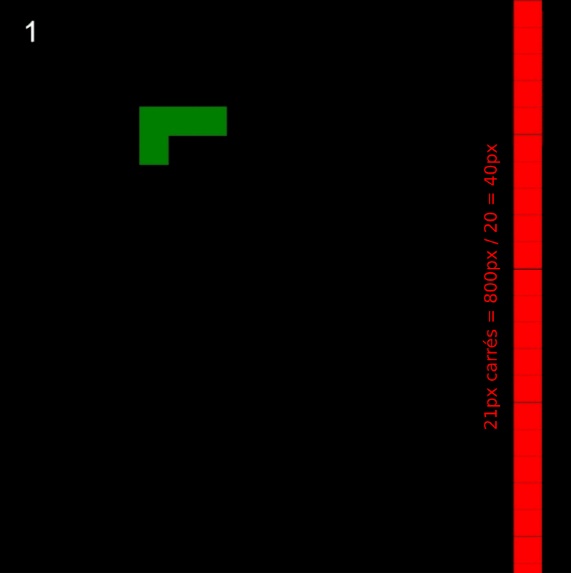
Une fois notre environnement de développement mis en place, nous pouvons désormais télécharger le dossier zip de base contenant l’arborescence de notre jeu du serpent afin de partir sur des bases.

Ce dernier contenant un fichier **HTML**, CSS ainsi qu’un fichier JS situé dans le dossier SRC, le fichier JS de base ressemblait à ceci :

Figure 1 : Fichier main.js de départ

**2.2 Terrain de jeu & Déplacement**

Pour initialiser le terrain de jeu, qui est de 800x800, il a fallu lui attribuer une couleur qui était dans notre cas, la couleur #000000 **(noire)**, ensuite nous avons placé un petit carré vert de taille 40x40 afin de représenter le serpent pour avoir une idée lors de la programmation de ce dernier.

Les tailles réelles du GIF donné ont été prises afin de reproduire au mieux le jeu, comme indiqué sur la Figure 2, les mesures de la taille de la pomme (identique au serpent) ont été prises afin de connaître leur taille en pixel et de pouvoir reproduire exactement la même forme sur la réplique.

21 carrés ont pu être posés en verticale ce qui nous amène à arrondir ces derniers à 20 pour avoir un résultat précis, donc : 800px **(tableau)** / 20 **(nbr de carrés)** = 40px.

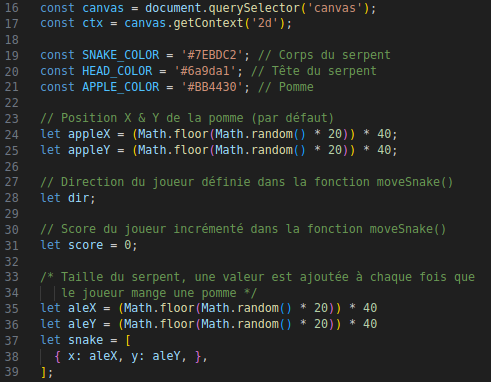
Les mesures prises ont donc été initialisées dans différentes variables retrouvables dans le code source.

Figure 2 : Calcul pour les dimensions de la pomme et du serpent en pixels.

L’étape suivante est le déplacement du serpent, pour ce faire, nous allons vérifier la touche que l’utilisateur aura pressé parmi les quatre flèches directionnelles, ensuite, selon la touche, nous allons attribuer une valeur à une variable **(1 pour haut, 2 pour bas, etc.)**, et puis désormais, nous allons créer une fonction qui déplace le serpent dans une direction précise en se basant sur la variable déclarée précédemment.

**2.3 Objectif**

**Déclaration LET & CONST**

****Pour la déclaration de variables, nous n’avons pas eu le droit d’utiliser VAR, mais uniquement LET et CONST, la différence entre le LET et le VAR est que la portée du VAR est globale contrairement à celle du LET qui est locale (limitée à celle du bloc dans laquelle elle est déclarée).

Comme montré sur la figure 3, la déclaration de ces dernières et organisée et commentée rapidement afin de pouvoir transmettre la signification des variables si quiconque utiliserait notre code.

Figure 3 : Voici un bout de code de mon main.js qui contient la déclaration des variables.

**Création de classes**

Pour la création des classes, je me suis aidé de deux sites internet en grande partie :

[**https://devdocs.io/javascript-classes**](https://devdocs.io/javascript-classes)[**https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes**](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Classes)

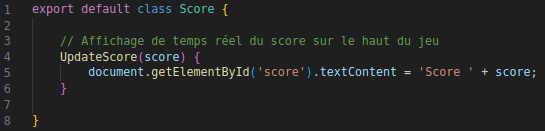
Dans mon cas, je vais créer trois classes, une première nommée Score (score.js), une seconde Snake (snake.js) et une dernière Apple (apple.js), ces dernières seront importées au fichier principal du programme et me serviront à avoir un fichier main.js beaucoup moins chargé ainsi que de répartir mon code sur différents fichiers pour mieux me retrouver lors d’une modification particulière à effectuer.

Figure 5 : Code présent dans la classe Score (score.js) avec en dessous l’importation effectuée dans le main.js.

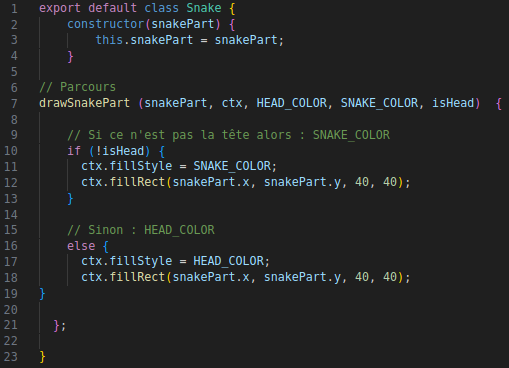


Figure 4 : Code présent dans la classe Snake (snake.js) avec en dessous l’importation effectuée dans le main.js.

**Fonctions fléchées**

En ce qui concerne la déclaration de fonctions, nous avons dû utiliser les fonctions fléchées car celles-ci ont une syntaxe plus courtes que les expressions de fonctions, pour utiliser ces fonctions, rien de très compliqué, il suffit d’indiquer le nom d’un argument (peut être vide si pas il n’y en a pas).

La fonction doit aussi être commentée selon les normes de codages (voir figure 6) afin de décrire les paramètres utilisés (si existants) et le but de la fonction.

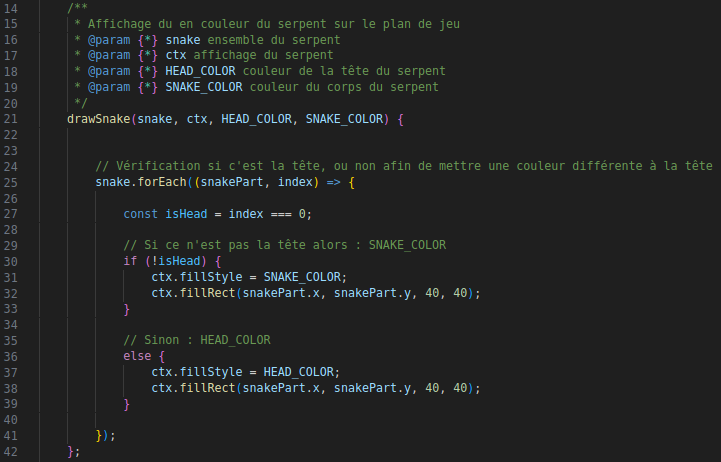


Figure 6 : Exemple d’annotation sur une fonction

**Modules d’importation / exportation**

Pour pouvoir utiliser les class précédemment créées dans d’autre fichiers, il a fallu exporter ainsi qu’importer ces dernières en utilisant les modules faits pour cette occasion, lorsqu’une classe est créée, on l’exporte directement lors de sa création en ajoutant **export default** devant son nom (voir figure 7), ce qui nous permettra d’utiliser le contenu de cette dernière dans le fichier qu’on souhaite.

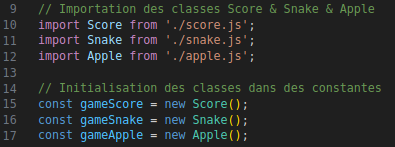
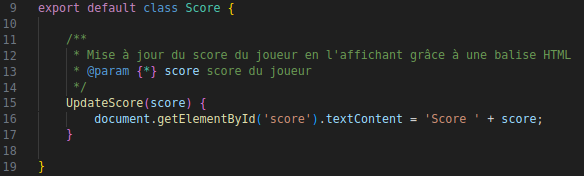
Dans notre situation, nous avons importé 3 classes différentes au cours de la programmation du jeu : Snake, Apple ainsi que Score, l’importation de ces dernières ce sont faites en utilisant **import NOM from ‘CHEMIN’** (voir figure 8) tout en haut du code, nous initialisons ensuite une constante avec comme valeur **new NOM() ;.**

Figure 8 : importation des classes exportées auparavant.

Figure 7 : exportation de la classe Score lors de sa création.

**Opérateur REST**

isa