P\_BULLES

Snake

Christopher Ristic

Ristic Christopher – FID2

ETML / 1004 LAUSANNEVennes 12 1004 Lausanne

Table des matières

[1 Introduction 3](#_Toc207011091)

[1.1 Objectif 3](#_Toc207011092)

[2 Installation 3](#_Toc207011093)

[2.1 Versions des outils utilisés 3](#_Toc207011094)

[3 Accéder à la page du jeu 4](#_Toc207011095)

[4 Conclusion 4](#_Toc207011096)

[4.1 Conclusion personnelle 4](#_Toc207011097)

# 

# Introduction

Ce projet consiste à réaliser un réplica du jeu classique du serpent en utilisant **JavaScript**.

Le jeu doit être dessiné sur un **canevas HTML** (<canvas>)

Le code **JavaScript** utilise la syntaxe des modules **ES** (ECMAScript Modules, ESM) pour organiser le code en plusieurs fichiers avec des responsabilités distinctes.

Le serveur de développement est géré par **Vite**, un outil moderne qui permet un rechargement rapide et une configuration minimale.

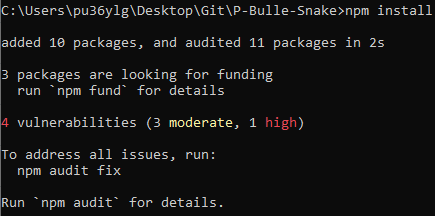
## Objectif

Le but du jeu est simple :

* Contrôler un serpent qui se déplace sur une grille
* Manger de la nourriture (des pommes) pour grandir
* Eviter de se heurter aux murs ou à son propre corps.
* Le score augmente à chaque fois que le serpent mange de la nourriture.

# Installation

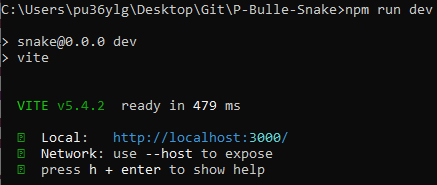
On a besoin d’un environnement d’exécution Javascript du côté serveur. Pour cela, il faut installer **Node.js** puis, depuis le dossier racine du jeu, On va taper la commande **npm install** dans **l’invite des commandes** :

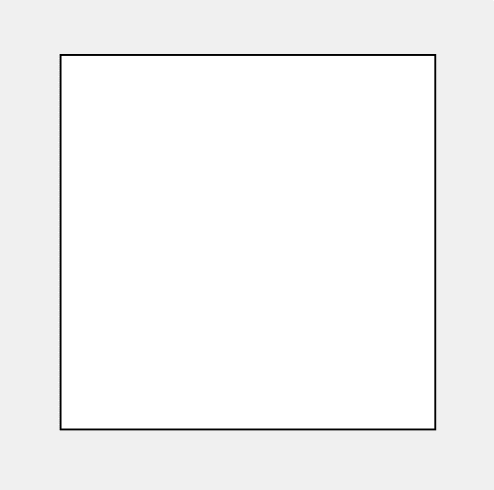


## Versions des outils utilisés

|  |  |
| --- | --- |
| **Outil** | **Version** |
| Node.js | 20.17.0 |
| Gestionnaire de paquets (npm) | 10.8.2 |
| Vite | 5.4.2 |

# Accéder à la page du jeu

Tout d’abord il faut lancer le serveur avec la commande **npm run dev** dans l’invite des commandes depuis la racine du jeu.

Après avoir démarré le serveur, on accède à la page du jeu via le lien <http://localhost:3000/>  
  
Premier affichage visuel du jeu sur la page web :

# Affichage du score

Pour afficher le score, on utilise le type **CanvasRenderingContext2D** qui est une interface permettant de dessiner sur le canvas. Le **canvas** est le rectangle blanc dans lequel on pourra jouer.Le score doit être affiché en noir avec une police Arial de 20px.

# Gestion des constantes

Une organisation claire et centralisée des constantes a été essentielle pour garantir la maintenabilité et la lisibilité du code. Cela évite aussi les fautes de frappes éventuelles.

Le but étant de rassembler toutes les constantes utilisées dans le projet dans un fichier unique, afin d’éviter la duplication et de faciliter les modifications futures.

# Snake

**Snake.js** définit la logique de base du serpent dans le jeu Snake. Il comprend les fonctions pour initialiser la position du serpent, gérer son déplacement selon une direction donnée et l’afficher correctement sur le canvas. Cette bibliothèque permet ainsi de centraliser la gestion visuelle et fonctionnelle du serpent, en assurant clarté et réutilisabilité du code.

# Collision et Système de layer

Le module **collision.js** gère les collisions dans le jeu Snake, séparant celles avec le corps du serpent et celles avec les murs ou obstacles internes. Les fonctions utilisent les layers (SNAKE et WALL) pour détecter précisément les collisions et box pour tenir compte de la taille des segments. Grâce à cette approche, le code reste lisible, modulable et facilement extensible à d’autres types d’entités.

Le **layering** est une technique permettant de séparer les éléments d’un jeu en différentes couches logiques pour gérer plus facilement les interactions et collisions.

Dans le projet Snake, les layers peuvent représenter la pomme, les murs ou des bonus, chaque entité étant associée à un masque de bits.

## Bitmask

L’utilisation d’un **bitmask** permet de combiner plusieurs couches dans une seule variable avec l’opérateur **|** (OR bit à bit) et de tester rapidement les collisions avec l’opérateur **&** (AND bit à bit), tandis qu’une approche à **valeurs simples** ne permet pas cette combinaison et nécessite des conditions multiples. Le bitmask offre ainsi plus de flexibilité et de performance, tandis que les valeurs simples restent plus lisibles pour des jeux très simples.

# Food

Nous avons intégré le **layering** pour la nourriture en utilisant la **constante** LAYERS.FOOD, permettant de distinguer visuellement et logiquement la nourriture des autres entités du jeu.

La **fonction** **generateFood** place aléatoirement la nourriture sur la grille tout en respectant les dimensions du canvas et la taille des cases (box), et chaque nourriture générée possède le layer FOOD.

La **fonction** **drawFood** dessine la nourriture sur le canvas à sa position actuelle, en utilisant box pour sa taille.

Dans **main.js**, lorsque la tête du serpent touche une case avec layer FOOD, le score est incrémenté, le serpent grandit et une nouvelle nourriture est générée, garantissant que le layering est pris en compte pour toutes les interactions.

# Utilisation de l’IA

Aide au niveau de la syntaxe, par exemple sur les méthodes d’utilisation de l’interface CanvasRenderingContext2D. Aide sur la rédaction, commentaires, et sur les calculs.

# Conclusion

## Conclusion personnelle