

CrÉation d’un jeu snake

Technique Web



03 fÉvrier 2020

Boinega Antonin, Dautheville Dylan

# Table des matières

[I. Table des matières 1](#_Toc31666340)

[II. Introduction 2](#_Toc31666341)

[III. Le principe d’un snake 2](#_Toc31666342)

[IV. Le modÈle MVC 3](#_Toc31666343)

[V. DÉroulement d’une partie 3](#_Toc31666344)

[1. Le dÉbut de la partie 3](#_Toc31666345)

[2. Le dÉroulement de la partie 4](#_Toc31666346)

[3. Fin de la partie 4](#_Toc31666347)

[VI. Choix dans le code 5](#_Toc31666348)

[VII. AmÉlioration possible 5](#_Toc31666349)

[VIII. Conclusion 5](#_Toc31666350)

# Introduction

Ce document traitera de la réalisation d’un jeu « Snake », à l’aide du langage Web JavaScript.

Il fait suite au cours de Monsieur Jean, « Technique WEB », qui concernait l’utilisation des langages WEB et plus précisément le langage JavaScript.

Ce TP à étais réalisé par Monsieur Boinega et Monsieur Dautheville qui ont travaillé en collaboration dans la réalisation de ce projet.

# Le principe d’un snake

D’après Wikipédia :

« Le Snake, de l'anglais signifiant « serpent », est un genre de jeu vidéo dans lequel le joueur dirige une ligne qui grandit et constitue ainsi elle-même un obstacle. Bien que le concept tire son origine du jeu vidéo d'arcade Blockade, il n'existe pas de version standard. Son concept simple l'a amené à être porté sur l'ensemble des plates-formes de jeu existantes sous des noms de clone. »

Le Snake est un jeu très simple et très rependu, ou le joueur incarne un serpent dont le but et de manger des fruits pour grandir (et faire monter le score), plus le serpent mange, plus il grandit et plus cela devient compliqué de rester en vie. En effet si le serpent se cogne la tête contre le mur ou qu’il se mange la queue, le serpent meurt et la partie est perdue.

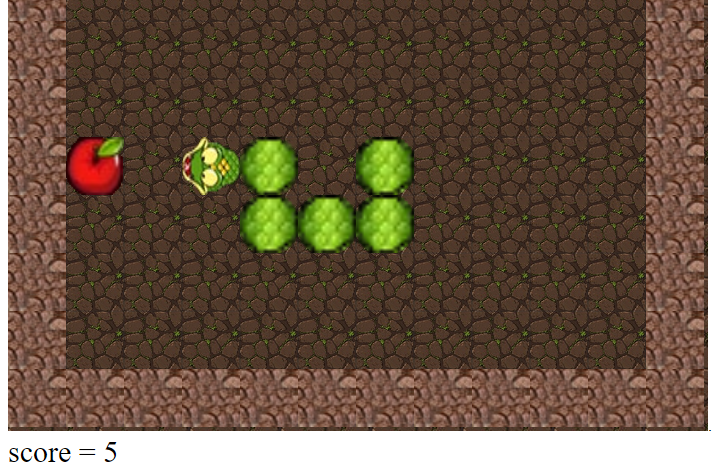


Figure 1 Capture d’écran de l'application

# Le modÈle MVC

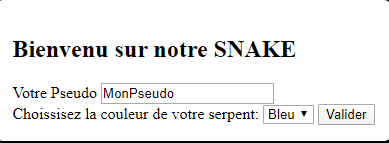
Pour la réalisation de ce TP, une des seules contraintes fonctionnelles était l’utilisation du modèle MVC (model view Controller). C’est une organisation très répandue qui consiste à répartir les classes selon leurs rôles, ainsi nous avons :

* Des classes Model, ce sont les classes métier, ils représentent l’objet et la data et seras appelé par le Controller, dans notre exemple nous avons deux classes Model, une représentante plusieurs objets tels que la grille, la position de la nourriture, la position de la tête, le premier élément de la queue et une autre classe Model Queue représentation un élément de la queue.
* Une View, qui représentent l’affichage, c’est dans cette classe on gère l’affichage de l’interface avec lequel le joueur devra interagir pour jouer.
* Une classe Controller, les classes Controller ont pour rôle de faire les appels aux deux classes citées précédemment (Model et View), et de gérer le déroulement de la partie, il réceptionnera les demandes de l’utilisateur (via des appuis sur les touches directionnelles du clavier) et les retransmettras dans son code pour faire respecter les demandes.

# DÉroulement d’une partie

## Le dÉbut de la partie

Au lancement de la page une modale s’ouvre et demande au joueur de rentrer un pseudo ainsi que de choisir la couleur de son serpent, c’est gérer grâce à une balise <dialog> et un peu de code JavaScript.



On peut ensuite cliquer sur valider pour faire démarrer le jeu.

## Le dÉroulement de la partie

Après ça on peut prendre le contrôle de notre Snake, chaque fruit mangé incrémente notre score de 1 et permet la génération d’un nouveau fruit à une position aléatoire.

Pour la gestion de ce trajet, on se sert d’un setInterval ( une fonction qui va permettre de rappeler la même fonction en boucle à chaque laps de temps donné ), et la fonction présente dans notre Controller qui s’appelle Déroulement.

La fonction déroulement commence par récupérer la touche directionnelle ( s’il y en a une ), qui à étais donné ensuite il fait appelle à la fonction changeDirection de la classe modelée afin de gérer la gestion du déplacement de notre fonction.

Cette fonction nous renvoie l’état de notre partie, on peut traiter ainsi les différents cas c’est-à-dire :

* Rien ne se passe et la partie continue normalement
* Le serpent mange un fruit, on génère donc un nouveau fruit et on ajoute une queue à notre serpent.
* Le serpent se mord la queue ou rencontre un mur, la partie se finit on enregistre le score et affiche le tableau de score.

## Fin de la partie

Quand la partie est terminée ont fait appelle à la fonction finDePartie de la classe controller ce qui va nous permettre dans un premier temps d’enregistrer le score au tableau des scores si celui-ci dépasse le 5ème meilleur score.

Ensuite on affiche une modale, on félicite l’utilisateur on lui affiche son score et on affiche un tableau des meilleurs scores.

NB : Le tableau des scores utilise une variable « localStorage », ce qui veut dire que c’est enregistré sur le navigateur du client, ainsi si on change de navigateur ou d’ordinateur, on perd le tableau des scores.

# Choix dans le code

Dans le code nous avons fait plusieurs choix, tout d’abord lorsque le serpent mange un fruit un nouvel élément découlant de ModelQueue est rattaché directement à la tête si le serpent avait déjà une queue le reste de la queue se rattache à ce nouvel élément.

À chaque frame la queue doit s’actualiser, la méthode que nous avons utilisée un plutôt optimisé puisque à part pour le premier élément de la queue qui doit s’adapter à la tête les autres éléments n’ont qu’a prendre la position de son élément précédent pour s’actualiser.

Nous avons décidé dans le Model de ne pas faire apparaitre la tête du serpent sur la grille pour ne pas a voir a gérer le cas de la tête de serpent qui prend la position d’un mur ou d’un fruit ou de sa queue même si c’est relativement facile à développer.

Également nous avons décidé de charger toutes les images aux préalables avant le lancement de l’application grâce à une fonction de callback.

Nous avons essayé de mettre en œuvre la plupart des pratiques que vous nous avez enseignées en cours

# AmÉlioration possible

Le premier point d’amélioration serait de dissocier les classes modèle, en effet avec la simplicité du logiciel que nous devions réaliser nous avons trouvé ça judicieux de regrouper l’ensemble applicatif de nos modèles au sein d’une même classe nommée modèle. Ce choix se justifiait par la simplicité du programme.

Cependant, pour faciliter l’amélioration du programme futur, le fait de tout séparer dans différents fichiers ( Snake.js, Grille.js, Fruit.js) voir même rajouter une structure de données pour le corps du Snake pourrait être un point d’amélioration primordiale.

Il faut également revoir complètement l’aspect graphique de notre application, qui n’est pas très belle et pas très en accord avec le jeu (les modales blanches devraient être vertes par exemple, et les skins devraient être visibles sur la modale de choix ).

# Conclusion

Ce projet nous a permis de pratique plus en profondeur le JavaScript, qui est une initiation pour nous, nous avons essayé de respecter l’ensemble du projet, mais nous regrettons également de ne pas avoir fait une architecture parfaite.

Dans l’ensemble nous sommes plutôt satisfaits du travail réalisé.