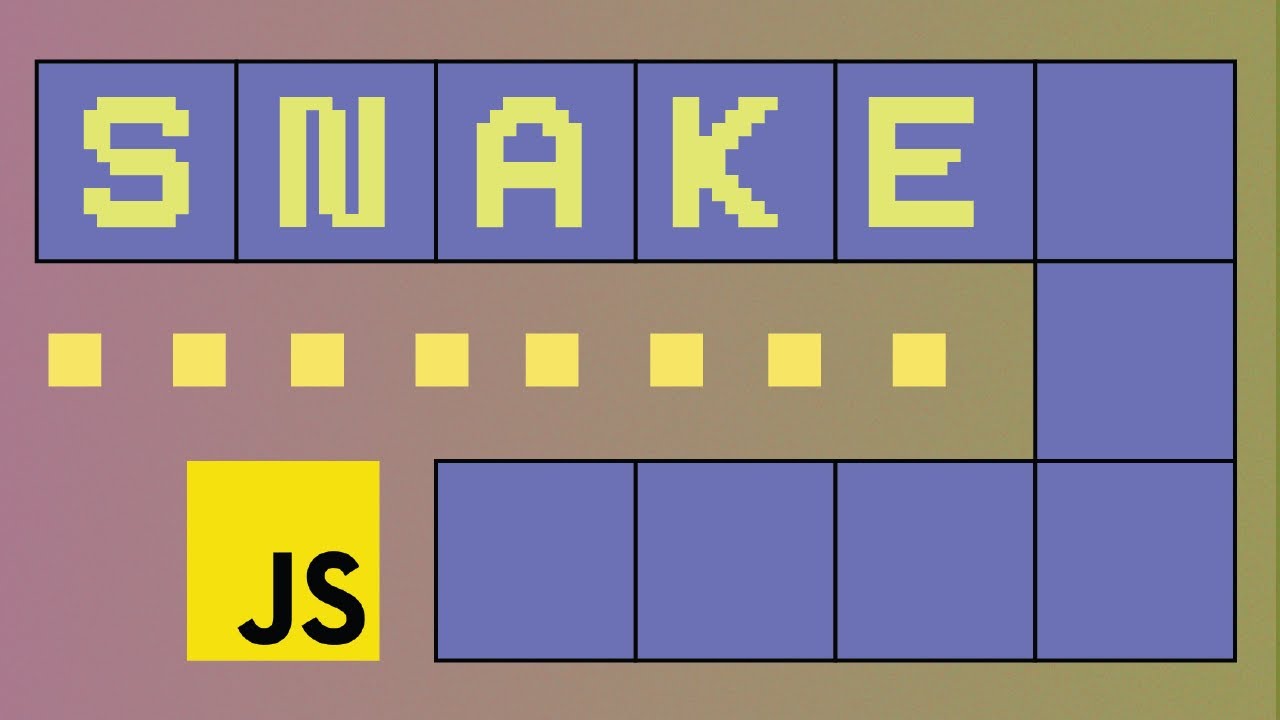
P\_Bulles\_Snake



Lucas Lordon MID2/GRP2D

Sébeillon – ETML

07.11.23 – 09.01.24

Aurélie Curchod

Table des matières

[Introduction 1](#_Toc153882462)

[Snake 1](#_Toc153882463)

[Introduction 1](#_Toc153882464)

[Résultat 1](#_Toc153882465)

[Logique du code 1](#_Toc153882466)

[Conclusion 1](#_Toc153882467)

[Self-Service Machine 1](#_Toc153882468)

[Introduction 1](#_Toc153882469)

[Résultat 1](#_Toc153882470)

[Logique du code 1](#_Toc153882471)

[Conclusion 1](#_Toc153882472)

[Conclusion 1](#_Toc153882473)

# Introduction

Le projet P\_Bulles a pour objectif l’apprentissage du JavaScript. Pour ce faire nous avons eu deux « exercices » : *Snake* et *SelfServiceMachine*, de plus un memo JS contenant la théorie que l’on a apprise durant ce projet doit être réalisé.

# Snake

## Introduction

Le projet Snake doit ressembler au GIF « abc.gif » disponible sous *P\_Bulles\_Snake\Doc\LiensRapport*.

Il consiste à crée une réplique du jeu Snake en JS avec les fonctionnalités suivantes :

* Mouvement du Snake, sauf direction opposée
* Collision (Bordure et Snake qui se rentre dedans)
* Apparition des pommes
* Manger les pommes (afin de grandir)
* Score (+1 à chaque pomme mangée)

## Analyse

Au début du cours j’ai mené une courte analyse afin de savoir comment j’allais coder mon snake.js, voici les points importants que j’en ai tiré :

**Langage** : JS, HTML, CSS, NODE

**Méthodes** : utilisation des classes (afin d’avoir un code propre et de mettre en place le multijoueur) et du canevas.

**Graphiquement** : Utilisation du code de base (canevas), chaque partie du serpent sera représenté par un carré de couleur dans le canevas. Afin de différentier les parties du snake la tête sera d’une couleur différente.

**Logiquement** :

**Snake :**

Le snake est une instance de la classe snake, chaque partie de son corps et enregistrer dans une liste via des cordonnée X et Y. Les caractéristiques générales sont définies lors de l’instanciation (couleur, taille, position de départ, etc.)

Le déplacement du snake ce fait en deux étape 1) une nouvelle tête ce crée dans la direction du snake au début de la liste des cordonnées 2) si le serpent a faim (isHungry == 0) la dernier ligne de la liste de cordonnée (.length) et supprimée.

**Apple :**

La pomme est une instance de la classe pomme sa position est enregistrer via des cordonnée X et Y.

C’est la pomme qui détecte si un « objet » de type snake est en contact avec elle dans ce cas elle augmente la variable « isHungry » du snake (de sa propre valeur (pomme)) et réapparait ailleurs.

## Résultat

A la fin du cours du à une modifiacation des classe le main ne fonctionnait plus donc j’ai deux version du code :

Mon résulta du Snake « de base » demandé est disponible sous *P\_Bulles\_Snake\Code\src\main.js*

Mon résulta du Snake « de base » demandé est disponible sous *P\_Bulles\_Snake\Code\src\main.js*

## Réalisation

Je fais le choix d’utiliser le code du snake a 4 comme exemple dans cette section car l’utilisation des classes y est plus parlée et que le code est très similaire à celui du mode de jeu solo.

### Création d’un snake

const snakeConfigs = [

  {

    color: SNAKE\_DEFAULT\_COLOR,

    bodyColor: SNAKE\_DEFAULT\_BODY\_COLOR,

    length: SNAKE\_DEFAULT\_LENGTH,

    posX: SNAKE\_DEFAULT\_POS\_X,

    posY: SNAKE\_DEFAULT\_POS\_Y,

    direction: SNAKE\_DEFAULT\_DIRECTION,

    hunger: SNAKE\_DEFAULT\_HUNGER,

    squareSize: GAME\_SQUARE\_SIZE,

    gridWidth: GAME\_GRID\_WITH\_SIZE,

    gridHeight: GAME\_GRID\_HEIGHT\_SIZE,

    score: SNAKE\_DEFAULT\_SCORE,

    canChangeDirection: SNAKE\_DEFAULT\_CAN\_CHANGE\_DIRECTION,

    downControl: SNAKE\_DEFAULT\_DOWN\_CONTROL,

    upControl: SNAKE\_DEFAULT\_UP\_CONTROL,

    leftControl: SNAKE\_DEFAULT\_LEFT\_CONTROL,

    rightControl: SNAKE\_DEFAULT\_RIGHT\_CONTROL,

    freez: false,

  },

snakeConfigs est la liste qui permet d’instancier les serpents selon la configuration que l’on veut

// Création des serpents en utilisant la configuration

let numberOfSnakes = 4;

numberOfSnakes est une variable essentiel a configurer il faut lui attribuer le nombre de serpent que l’on a créé dans snakeConfigs. Cette variable est réutilisée partout dans le code.

for (let i = 0; i < numberOfSnakes; i++) {

  let newSnake = new Snake(

    snakeConfigs[i].color,

    snakeConfigs[i].bodyColor,

    snakeConfigs[i].length,

    snakeConfigs[i].posX,

    snakeConfigs[i].posY,

    snakeConfigs[i].direction,

    snakeConfigs[i].hunger,

    snakeConfigs[i].squareSize,

    snakeConfigs[i].gridWidth,

    snakeConfigs[i].gridHeight,

    snakeConfigs[i].score,

    snakeConfigs[i].canChangeDirection,

    snakeConfigs[i].downControl,

    snakeConfigs[i].upControl,

    snakeConfigs[i].leftControl,

    snakeConfigs[i].rightControl,

    snakeConfigs[i].freez

  );

snakes.push(newSnake);

}

Cette boucle for permet de stocker tous les snake dans la même liste (snakes)

### Création d’une pomme

// Création de l'objet Apple

let apple = new Apple('orange', 0, 0, snakes, 1, GAME\_SQUARE\_SIZE, GAME\_GRID\_WITH\_SIZE, GAME\_GRID\_HEIGHT\_SIZE)

Instanciation d’une pomme

### Initialisation des snakes

// Initialisation des serpents

snakes.forEach(snake => {

  snake.initial();

});

Initialisation des serpents, à lieu avant la boucle de jeu.

### Initialisation de la pomme

      firstTime

          ? (apple.create(), firstTime = false)

Initialisation de la pomme, à lieu à la première exécution de la boucle de jeu.

    create() {

        this.spawn();

        this.draw();

    }

Détail de la méthode « create » qui permet de déplacer la pomme

    spawn() {

        let goodPos = true

        function isPositionValid(X, Y, snakeCoordinates) {

            return snakeCoordinates.some(coord => coord.PosY === Y && coord.PosX === X);

        }

        do {

            let X = Math.floor(Math.random() \* this.GAMEGRIDWITHSIZE);

            let Y = Math.floor(Math.random() \* this.GAMEGRIDHEIGHTSIZE);

            goodPos = this.snakes.every(snake => !isPositionValid(X, Y, snake.listeCordonnees));

            //definit la nouvelle position

            if (goodPos) {

                this.PosX = X;

                this.PosY = Y;

            }

        } while (!goodPos);

    }

La méthode spawn permet de déplacer les positions de la pomme de manière aléatoire sans quel ce déplace sur un serpent ou en dehors des bordures de jeu.

    draw() {

        ctx.fillStyle = this.color;

        ctx.fillRect(this.PosX \* this.GAMESQUARESIZE, this.PosY \* this.GAMESQUARESIZE, this.GAMESQUARESIZE, this.GAMESQUARESIZE);

    }

La méthode draw permet de dessiner la pomme dans le canva

### Programme principale

// Fonction pour gérer le jeu

const move = () => {

compareSnakeColision()

  for (let i = 0; i < numberOfSnakes; i++) {

    snakes[i].checkIntercection()

    snakes[i].canChangeDirection = true;

    snakes[i].update();

    apple.checkCollision(snakes[i]);

    allFreezeExceptOne() ?

      gameOver = true : gameOver = false;

  }

  !gameOver

  ? (CTX.fillStyle = GRID\_BACK\_GROUND\_COLOR,

      CTX.fillRect(0, 0, GRID\_WITH, GRID\_HEIGHT),

      firstTime

          ? (apple.create(), firstTime = false)

          : snakes.forEach(snake => snake.draw()),

      apple.draw(), cpttest++) : (cpttest !== 0 ? location.reload() : console.log("1"));

};

La première partie de la « boucle de jeu » move a pour but l’actualisation des serpents, la partie gameOver permet de gérer la première exécution de gameOver/move, la deuxième partie permet d’actualiser les serpents, la pomme, gérer la fin de partie.

### Comparer la collision des snakes entre eux

function compareSnakeColision() {

  for (let i = 0; i <= numberOfSnakes; i++) {

    for (let j = i + 1; j <= numberOfSnakes - 1; j++) {

      snakes[i].checkSnakeIntercection(snakes[i], snakes[j])

    }

  }

}

Cette fonction permet de faire une comparaison entre 2 serpent via la méthode checkSnakeIntercection en fesant attention d’éviter les comparaison double :

Sans optimisation :

0=0¦0=1¦0=2¦0=3¦1=0¦1=1¦1=2¦1=3¦2= 0¦2= 1¦2=2¦2=3¦3=0¦3=1¦3=2¦3=3

Avec optimisation :

0=1¦0=2¦0=3¦1=2¦1=3¦2=3

    // Vérifie les collisions entre deux serpents

    checkSnakeIntercection(snake1, snake2) {

        const [head1, ...body1] = snake1.listeCordonnees;

        const [head2, ...body2] = snake2.listeCordonnees;

        //tete du serpent 1 dans corps du serpent 2

        const head1CollidesWithSnake2 = body2.some(n => (n.PosX === head1.PosX && n.PosY === head1.PosY));

        //tete du serpent 2 dans corps du serpent 1

        const head2CollidesWithSnake1 = body1.some(n => (n.PosX === head2.PosX && n.PosY === head2.PosY));

        if (head1CollidesWithSnake2) {

            snake1.freez = true;

        }

        if (head2CollidesWithSnake1) {

            snake2.freez = true;

        }

        return head1CollidesWithSnake2 || head2CollidesWithSnake1;

    }

Méthode checkSnakeIntercection qui est utiliser dans la méthode compareSnakeColision.

### Collision seule ou avec bordure

    checkIntercection() {

        const [head, ...body] = this.listeCordonnees;

        //si le serpent ce rentre dedant ou si il rentre dans la bordure

        if (body.some((n1) => (n1.PosX === head.PosX && n1.PosY === head.PosY)) || (!this.checkBorder())) {

            this.freez = true;//il meurt

        }

    }

Méthode checkIntercection permet de gérer les colision d’un seul serpent avec lui-même ou les bordure.

    // Vérifie si le serpent atteint les bords du canvas

    checkBorder() {

        return (

            (this.listeCordonnees[0].PosX >= 0 && this.listeCordonnees[0].PosX < this.GAMEGRIDWITHSIZE) &&

            (this.listeCordonnees[0].PosY >= 0 && this.listeCordonnees[0].PosY < this.GAMEGRIDHEIGHTSIZE)

        );

    }

Méthode checkBorder utiliser dans la méthode checkIntercection

### Déplacement du snake

    // Déplace le serpent

    move() {

        this.listeCordonnees.unshift({ PosX: this.listeCordonnees[0].PosX, PosY: this.listeCordonnees[0].PosY }) //crée la nouvelle tete

        this.movehead(); //bouge la nouvelle tete en fonction de la direcion

        (this.isHungry === 0) ? this.listeCordonnees.pop() : undefined;//suprime la queu du serpent si il a faim (isHungry = 0)

    }

Méthode move qui permet de bouger un serpent dans la bonne direction

### Affichage

    // Dessine la pomme sur le canvas

    draw() {

        ctx.fillStyle = this.color;

        ctx.fillRect(this.PosX \* this.GAMESQUARESIZE, this.PosY \* this.GAMESQUARESIZE, this.GAMESQUARESIZE, this.GAMESQUARESIZE);

    }

Méthode draw de la pomme

    // Dessine le serpent sur le canvas (tete d'une couleur et corp d'une autre)

    draw() {

        //dessine le corp

        this.listeCordonnees.forEach(coord => {

            ctx.fillStyle = this.bodyColor;

            ctx.fillRect(coord.PosX \* this.GAMESQUARESIZE, coord.PosY \* this.GAMESQUARESIZE, this.GAMESQUARESIZE, this.GAMESQUARESIZE);

        });

        //dessine la tête

        ctx.fillStyle = this.color;

        ctx.fillRect(this.listeCordonnees[0].PosX \* this.GAMESQUARESIZE, this.listeCordonnees[0].PosY \* this.GAMESQUARESIZE, this.GAMESQUARESIZE, this.GAMESQUARESIZE);

    }

Méthode draw du serpent, qui dessine de couleur différente la tête de son corps.

## Conclusion

# Self-Service Machine

Je n’ai pas fini de coder le programme du self machine.

# Conclusion

# Utilisation d’IA

Personnellement je n’ai pas eu recours à des IA lors de la programmation car je pense quel sont positive dans le travail mais négative dans l’apprentissage, car il faut savoir se débrouiller seul.

Par contre lors des commentaires du code j’ai utilisé chatGPT.