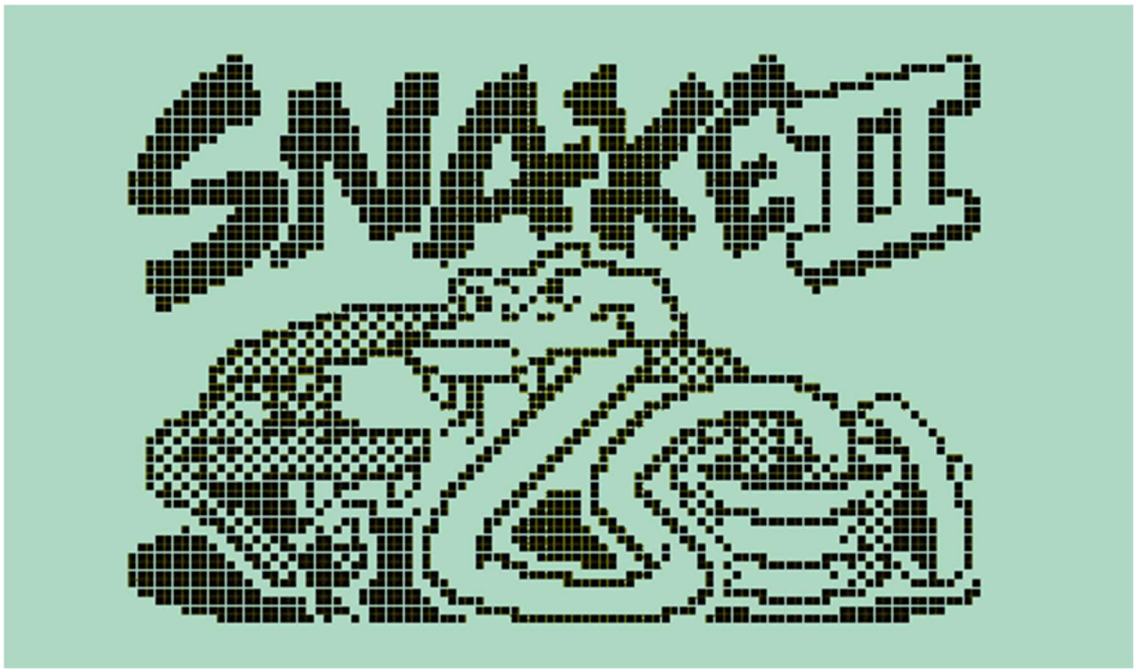
P-Bulles-Snake



Lebet Esteban – CID2A
Lausanne, ETML
40P
Chenaux Patrick

Table des matières

1. Introduction

Ce projet a été donné comme une introduction au JavaScript, le projet se tient sur une durée totale de 40 périodes. Le travail demandé est de créer une réplique du célèbre jeu « Snake », créé en 1976.

2. Const, Let et Var

Const et **Let** sont préférés à **Var** car ils permettent une meilleure gestion de la portée des variables

Var à un moins bon contrôle sur le comportement des variables, ce qui peut causer plus d'erreurs liées à la portée ou encore au hosting.

3. Classes

3.1. main.js

Fichier principal qui va se charger du jeu en général et de son bon fonctionnement. Vitesse du jeu (tick), vérifier si le jeu est fini ou encore montrer l'écran de GAME OVER.

3.2. apple.js

Cette classe/fichier se charge de tout ce qui concerne les pommes dans le jeu. De la génération, à la modélisation ou encore à l'aléatoire d'où apparaissent les pommes.

3.3. snake.js

Cette classe/fichier se charge de tout ce qui concerne le serpent dans le jeu. Notamment la modélisation du serpent, son point d'apparition initial, comment il doit se comporter s'il rentre en collision avec une pomme (quand il en mange une) et aussi ses contrôles.

4. Méthodes

4.1. snake.js

- **moveSnake**
  Cette méthode gère le déplacement du serpent dans le jeu. Elle créé sa tête et calcule les nouvelles coordonnées de la tête du serpent en fonction de la vélocité, si la tête rentre en collision avec une pomme le score est mis à jour, le serpent grandis en taille et une nouvelle pomme apparaît.
  .
- **drawSnake**
  Cette méthode gère le dessin du serpent sur le plateau de jeu. Elle créé des rectangles pour créer le corps du serpent et donne une couleur à ces rectangles.

- **changeDirection**
  Cette méthode est responsable de modifier la direction du serpent en réponse à une touche du clavier. Un switch est initialisé pour vérifier dans quel sens va le serpent et l'empêche de se rentrer dans lui-même sans faire exprès (serpent qui regarde vers le bas et qui peut tout de même aller en haut).

- **checkCollision**
  Cette méthode s'occupe de vérifier si le serpent rentre dans son propre corps ou s'il touche un des bords du plateau de jeu. Si c'est le cas le jeu s'arrête

- **resetGame**
  Cette méthode va remettre le score à 0, le remettre dans sa position d'apparition initiale et lui redonner sa taille par défaut (deux blocs).

4.2. apple.js

- **createFood**
  Cette méthode génère de manière aléatoire les positions X et Y de la pomme pour créer une position d'apparition à la pomme. La méthode s'occupe de générer les positions aléatoirement, d'attribuer ses positions à la pomme, et pour chaque position de stocker leur valeur pour pouvoir les logs dans la console.

- **drawFood**
  Cette méthode s'occupe de dessiner les pommes générer avec **createFood().**

5. Fonctions

5.1. main.js

- **gameStart**
  Cette fonction va initialiser tous les paramètres de début.

- **nextTick**
  Cette fonction permet de mettre à jour le jeu à des intervalles prédéfinis

- **clearBoard**
  Cette fonction va tout simplement effacer tout le contenu du gameBoard (plateau de jeu)

- **checkGameOver**
  Cette fonction va vérifier toutes les conditions qui peuvent mener à une fin de jeu.

- **displayGameOver**
  Cette fonction va se lancer si une des conditions de fin de jeu de **checkGameOver()** s'active et va afficher un texte de fin de jeu.

- **resetGame**
  Cette fonction va réinitialiser le les valeurs du jeu (score et serpent) et va relancer une partie.

6. Constructeurs

6.1. apple.js

```
    constructor(unitSize, gameWidth, gameHeight) {
        // Propriétés de la pomme
        this.unitSize = unitSize;
        this.gameWidth = gameWidth;
        this.gameHeight = gameHeight;
        this.foodX = 0;
        this.foodY = 0;
        this.foodNumber = 0;
    }
```

Ce constructeur initialise les propriétés de la pomme avec la unitSize, la largeur et la hauteur du gameBoard, ainsi que les coordonnées et le compteur de pommes pour pouvoir les utiliser dans un commit log.

6.2. snake.js

```
    constructor(unitSize, apple, snakeColor, snakeBorder, score, scoreText) {
        // Propriétés du serpent
        this.unitSize = unitSize;
        this.xVelocity = unitSize;
        this.yVelocity = 0;
        this.snake = [
            { x: unitSize, y: 0 },
            { x: 0, y: 0 }
        ];
        this.apple = apple;
        this.snakeColor = snakeColor;
        this.snakeBorder = snakeBorder;
        this.score = score; // Référence au score
        this.scoreText = scoreText;
    }
```

Ce constructeur initialise les propriétés du serpent avec la unitSize, la vitesse, les différentes parties du serpent, les couleurs, le score, l'élément HTML affichant le score et une référence à la pomme pour ensuite travailler sur comment le serpent va les manger.

7. Modules Import & Export

7.1. Import

Le module d'import en JavaScript permet de charger des variables, fonctions ou classes à partir d'un autre module. En utilisant la déclaration **import**, on spécifie quels éléments d'un module externe doivent être utilisés localement dans le module souhaité.

7.2. Export

Le module d'export en JavaScript permet de rendre des variables, fonctions ou classes disponibles pour être utilisées dans d'autres fichiers ou modules. En utilisant des déclarations d'exportation telles qu'export ou export default, permettant ainsi leur utilisation dans d'autres parties de l'application.

Dans mon cas, ces deux modules me permettent de connecter les fichiers **apple.js** et **snake.js** à **main.js**.

8. Opérateur Rest

L'opérateur Rest en JavaScript est représenté par trois points de suspension (...).
Cet opérateur est utilisé dans les fonctions pour rassembler les arguments restants sous la forme d'un tableau (Array). Il permet de gérer un nombre variable d'arguments lors de la définition d'une fonction.
Cependant dans mon projet je ne l'ai pas utilisé.

9. Fonctions fléchées

Les fonctions fléchées en JavaScript offrent une syntaxe concise, sont idéales pour les fonctions courtes, héritent du this du contexte englobant, et n'utilisent pas le mot-clé function.

J'ai utilisé les fonctions fléchées dans **createFood()** à cet endroit

```
createFood() {
    const randomFood = (min, max) => Math.round((Math.random() * (max - min) + min)
/ this.unitSize) * this.unitSize;
}
```

10. Concept programmation

10.1. Génération aléatoire

J'ai utilisé la génération aléatoire pour l'apparition des pommes, afin qu'elles apparaissent partout sur le plateau de jeu et donner un peu de challenge au joueur.

J'ai fait ça dans la méthode **createFood :**

```
createFood() {
    const randomFood = (min, max) => {
        const randNum = Math.round((Math.random() * (max - min) + min) / this.unitSize) *
        this.unitSize;
        return randNum;
    };
    this.foodX = randomFood(0, this.gameWidth - this.unitSize);
    this.foodY = randomFood(0, this.gameWidth - this.unitSize);
    console.log("Fruit " + this.foodNumber + ": X: " + this.foodX + " | Y: " + this.foodY);
    this.foodNumber += 1;
}
```

10.2. Gestion de mémoire

Le score s'incrémente à chaque pomme que le serpent mange.
La position de la pomme se stock dans une propriété.

11. Commit log

Le commit log que j'ai créé est un log de l'apparition des pommes. Il va écrire dans la console le combientième fruit il est, sa position x et sa position y.
Code :

```
console.log("Fruit " + this.foodNumber + ": X: " + this.foodX + " | Y: " + this.foodY);
```

12. Sources

N'ayant jamais fait de JavaScript auparavant, je me suis inspiré du tuto d'une personne sur YouTube du nom de « Bro Code » pour m'aider à la réalisation de ce Snake.
Lien de la vidéo

13. Conclusion personnelle

J'ai trouvé ce projet très intéressant, le fait de devoir découvrir un langage sur le tas et développer un jeu avec ce même langage a été une expérience que j'ai beaucoup apprécié !