P-Bulle – Snake

Thibaud Racine – CID2A
ETML Sébeillon
2023 - 2024
40 P

Table des matières

INTRODUCTION

1.1 Projet

Le Projet P-Bulles nous demande de faire un réplica du célèbre jeu Snake à l'aide de Java Script afin de nous familisariser avec ce language. N'ayant aucune connaissances en Java Script, j'ai décidé de suivre un tutoriel sur Youtube pour ce projet. Le Lien de la vidéo se trouve dans la webographie du rapport de projet.

1.2 Const, Let & Var

L'utilisation de **const** et **let** par rapport à **var** en JavaScript offre une meilleure gestion de la portée des variables.

const et **let** donne un meilleur contrôle sur le comportement des variables, réduisent les erreurs liées à la portée et au hoisting.

CODE SNAKE

1.3 Let

running :
Cette variable booléenne indique si le jeu est en cours d'exécution **true** ou s'il est terminé **false**.
Elle est utilisée pour contrôler la boucle principale du jeu et déterminer si le jeu doit continuer à être mis à jour.

xVelocity et yVelocity :
Ces variables représentent les composantes de la vélocité du serpent sur l'axe X et Y.
Elles déterminent la direction dans laquelle le serpent se déplace à chaque tick du jeu.
Ces valeurs peuvent changer en réponse aux commandes du joueur.

Score :
Cette variable conserve le score du joueur dans le jeu.
Elle est mise à jour chaque fois que le serpent mange une pomme.

Apple :
Cette variable représente l'instance de la classe **Apple** dans le jeu.
Elle est utilisée pour gérer la position et le rendu de la pomme.

Snake :
Cette variable représente l'instance de la classe **Snake** dans le jeu.
Elle est utilisée pour gérer le corps du serpent, ses mouvements et son rendu.

1.4 Const

Définit des valeurs qui restent constantes tout au long du jeu.

gameBoard :
Représente le canevas HTML sur lequel le jeu Snake est rendu.

ctx :
Le contexte graphique du canevas **gameBoard**. Il est utilisé pour dessiner des éléments graphiques sur le canevas.

scoreText et **scoreValue :**
Représentent des éléments HTML utilisés pour afficher le texte "Score" et la valeur du score dans l'interface utilisateur du jeu.

resetBtn :
Représente le bouton HTML utilisé pour réinitialiser le jeu après la partie terminée.

gameWidth et **gameHeight :**
Représentent la largeur et la hauteur du canevas de jeu.
boardBackground ,**snakeColor**, **snakeBorder**, et **foodColor** :
Représentent des couleurs utilisées dans le rendu graphique du jeu.
boardBackground est la couleur de fond du canevas.
snakeColor est la couleur du serpent.
snakeBorder est la couleur de la bordure du serpent.
foodColor est la couleur de la pomme.

unitSize :
Représente la taille d'une unité dans le jeu. Elle est utilisée pour déterminer la taille du serpent, la taille de la pomme, etc.

Ces constantes sont utilisées tout au long du script pour définir des paramètres tels que la taille du canevas, les couleurs, les éléments HTML et d'autres valeurs fixes qui ne changent pas pendant l'exécution du jeu.

CLASSES

1.5 Apple.js

Cette classe en JavaScript, nommée "Apple" encapsule les fonctionnalités liées à la gestion de la pomme dans le jeu, comme l'initialisation, la génération aléatoire de position, le draw Fruit, et le log position.

Constructeur (Ligne 11) :
Le Constructeur initialise la taille de la pomme avec **unitSize**, la largeur du jeu avec **gameWidth**, et la hauteur du jeu avec **gameHeight**.
La position de la pomme est définie de manière aléatoire en appelant la fonction createRandomPosition

```
export default class Apple {
  // Constructeur pour initialiser l'objet Apple
  constructor(unitSize, gameWidth, gameHeight) {
    this.unitSize = unitSize; // Taille de la pomme
    this.gameWidth = gameWidth; // Largeur du jeu
    this.gameHeight = gameHeight; // Hauteur du jeu
    this.position = this.createRandomPosition(); // Position initiale aléatoire de
la pomme
  }
```

Méthode logPosition (Ligne 19) :
Affiche la position actuelle de la pomme dans la console.

```
  // Log la position des pommes
  logPosition() {
    console.log(
      `Position de la pomme = X: ${this.position.x} | Y: ${this.position.y}`
    );
  }
```

Méthode createRandomPosition (Ligne 24) :
Génère une position aléatoire pour la pomme à l'intérieur des limites du jeu et Utilise la fonction **randomNumber(min, max)** pour déterminer les coordonnées X et Y de la pomme.

```
  // Crée une position aléatoire pour la pomme dans les limites du jeu
  createRandomPosition() {
    const x = this.randomNumber(0, this.gameWidth - this.unitSize);
    const y = this.randomNumber(0, this.gameHeight - this.unitSize);
    return { x, y };
  }
```

Méthode draw(ctx, color) (Ligne 31) :
Dessine la pomme graphiquement avec la couleur spécifiée.
Utilise les coordonnées de la pomme et sa taille pour dessiner un rectangle qui représente la pomme.
Appelle la méthode **logPosition** pour afficher la position de la pomme.

```
// Méthode pour dessiner la pomme
 draw(ctx, color) {
   ctx.fillStyle = color; // Définit la couleur pour la pomme
   ctx.fillRect(
     this.position.x,
     this.position.y,
     this.unitSize,
     this.unitSize
   );
   apple.logPosition(); //Log la position de la nouvelle pomme
 }
```

Méthode randomNumber(min, max) (Ligne 38) :
Génère un nombre aléatoire dans la plage spécifiée (min, max).

```
 // Génère un nombre aléatoire
 randomNumber(min, max) {
   // Génère un nombre aléatoire pour obtenir une position alignée sur la grille
   return (
     Math.round((Math.random() * (max - min) + min) / this.unitSize) *
     this.unitSize
   );
 }
```

1.6 Snake.js

Cette classe en JavaScript, appelée "Snake" encapsule les fonctionnalités liées à la gestion du Snake dans le jeu, y compris l'initialisation, le déplacement, le dessin sur le contexte graphique, et éventuellement la logique pour manger la nourriture.

Constructeur (Ligne 11) :
Taille de chaque carré dans le Snake **unitSize** et crée le corps initial du serpent en appelant la méthode **createInitialSnake(initialLength)**.

```
export default class Snake {
  // Constructeur pour initialiser l'objet Snake
  constructor(unitSize, initialLength) {
    this.unitSize = unitSize; // Taille de chaque carré dans le Snake
    this.snakeBody = this.createInitialSnake(initialLength); // Initialise le corps
du serpent
  }
```

Méthode createInitialSnake(initialLength) (Ligne 16) :
Crée le corps initial du serpent.
La longueur initiale du serpent est spécifiée par le paramètre **initialLength**.

```
  // Crée le corps initial du Snake
  createInitialSnake(initialLength) {
    const snake = [];
    for (let i = 0; i < initialLength; i++) {
      snake.push({ x: this.unitSize * (initialLength - i - 1), y: 0 });
    }
    return snake;
  }
```

Méthode move(xVelocity, yVelocity) (Ligne 24) :
Déplace le serpent en mettant à jour les coordonnées de chaque partie du corps avec **xVelocity** et **yVelocity**.
Ajoute une nouvelle tête à l'avant du corps du serpent.

```
  // Déplace le serpent en mettant à jour ses coordonnées en fonction de la
vélocité
  move(xVelocity, yVelocity) {
    const head = {
      x: this.snakeBody[0].x + xVelocity,
      y: this.snakeBody[0].y + yVelocity,
    };
    // Ajoute la nouvelle tête à l'avant du corps du serpent
    this.snakeBody.unshift(head);
  }
```

Méthode draw(ctx, snakeColor, snakeBorder) (Ligne 31) :
Dessine le serpent sur le contexte graphique (`ctx`).
Chaque partie du corps du serpent est dessinée avec une couleur différente basée sur la teinte (`hue`).
La bordure du serpent est également définie avec une couleur spécifiée (`snakeBorder`).

```
  // Dessine le serpent sur le jeu
  draw(ctx, snakeColor, snakeBorder) {
    let hue = 0; // Initialise la teinte du Snake
    this.snakeBody.forEach((snakePart, index) => {
      // Change les couleurs du serpent
      hue = (hue + 5) % 360; // Change le gradient de 3 du snake à chaque fruit
mangé
      const color = `hsl(${hue}, 100%, 50%)`;
      ctx.fillStyle = color;
      ctx.strokeStyle = snakeBorder;
      ctx.fillRect(snakePart.x, snakePart.y, this.unitSize, this.unitSize);
      ctx.strokeRect(snakePart.x, snakePart.y, this.unitSize, this.unitSize);
    });
  }
```

1.7 Main.js

Ce script JavaScript, appelé "Main.js" gère les aspects du jeu Snake, de l'initialisation à la logique de jeu, en passant par la gestion des événements utilisateur et la mise à jour constante de l'affichage. Constitue le moteur du jeu Snake.

Constantes :
Définit plusieurs constantes, comme les éléments du **gameBoard**, **scoreText**, **scoreValue**, **resetBtn**, les couleurs du jeu etc.

Variables :
Déclare les variables nécessaires pour le fonctionnement du jeu, comme l'état du jeu **running**, les vélocités du serpent (**xVelocity** et **yVelocity**), le score, les instances de **apple** et du **snake**.

Event Listeners :
Écoute les événements du clavier **keydown** pour détecter les changements de direction du serpent.
Écoute l'événement du bouton de réinitialisation **resetBtn** pour relancer le jeu.

Fonction gameStart :
Appelle la fonction **initializeGame**.
Démarre la boucle de jeu avec la fonction **nextTick**.

Fonction initializeGame (Ligne 47) :
Importe les modules des classes **Apple** et **Snake**.
Initialise les instances de la pomme **apple** et du serpent **snake**.

Fonction nextTick :
Gère la logique du jeu dans une boucle. Efface le plateau de jeu, dessine la pomme, déplace et dessine le serpent, et vérifie si la partie est terminée.
Les Fonctions utilitaires comme **clearGame**, **createFruit**, **drawFruit**, **moveSnake**, **drawSnake**, **changeDirection**, setDirection, **checkGameOver**, **displayGameOver**, **resetGame**
Effectuent différentes tâches, telles que la gestion de l'affichage, le déplacement du serpent et la gestion du game over.

```
function nextTick() {
  if (running) {
    // Si le jeu tourne
    setTimeout(() => {
      clearGame();
      drawFruit();
      moveSnake();
      drawSnake();
      checkGameOver();
      nextTick();
    }, 75);
  } else {
    displayGameOver();
  }
}
```

MODULES

1.8 Import :

Import est utilisé pour inclure des fonctionnalités d'un module dans un autre module. On peut importer des éléments individuels ou la valeur par défaut d'un module.

1.9 Export :

Export est utilisé pour rendre accessible une variable, une fonction ou une classe d'un module à d'autres modules.

Les modules sont largement utilisés pour organiser le code de manière modulaire et organisée dans les applications JavaScript. Ces modules permettent de connecter **Snake.js** et **Apple.js** à **Main.js**

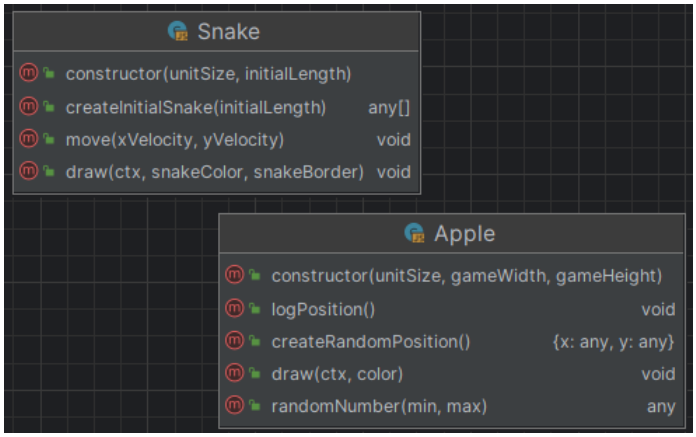
CONCEPT DE PROGRAMATION

1.10 Génération aléatoire :

Création aléatoire de la position de la pomme :

- La Méthode « **createRandomPosition** » de l'objet **Apple** est responsable de générer une position aléatoire pour la pomme dans le canvas.
- Elle utilise aussi **unitSize** qui sera la taille de la pomme, **gameWidth** comme largeur du canvas de jeu et **gameHeight** comme hauteur de jeu pour garantir que la pomme apparaisse à l'intérieur du canvas de jeu

1.11 Diagrammes de Classes



1.12 Opérateur Rest

L'opérateur **rest** est utilisé dans des fonctions pour collecter un nombre d'arguments qui varient sous la forme d'un tableau. Il permet à une fonction de recevoir un nombre indéfini d'arguments et de les traiter sous la forme d'un tableau.
L'opérateur rest est très utile lorsque qu'on a besoin de traiter un nombre variable d'arguments sans connaître à l'avance combien il y en aura.
Dans le cas de mon programme, je n'ai pas utilisé l'opérateur rest car je n'en voyais pas l'utilité.

1.13 Gestion Mémoire:

La position de la pomme est stockée dans la propriété "**Position**" de l'objet "**Apple**"

1.14 Commit Logs

Cette fonction JavaScript, logPosition(), affiche dans la console la position d'une pomme en utilisant les coordonnées X et Y stockées dans l'objet auquel elle est liée.

```
logPosition() {
        console.log(`Position de la pomme = X: ${this.position.x}
| Y: ${this.position.y}`);
    }
```

AUTRES

1.15 Webographie

Tuto Youtube: https://www.youtube.com/watch?v=Je0B3nHhKmM