ETML

Rapport de Projet P-Bulle

CID2B

Mathis Botteau
12/12/2023

Table des matières

A. Introduction

Dans ce rapport, nous allons explorer notre projet P_Bulle, qui a pour but de créer le célèbre jeu du Snake en utilisant JavaScript tout en y apprenant ce nouveau langage. Nous commencerons par un aide-mémoire qui rassemblera les points essentiels du langage, comme les modules d'import-export et les types de définition des variables. Ensuite, nous regarderons le choix du design, avec les raisons de notre palette graphique.

Il y aura aussi une partie d'analyse technique, où nous verrons le rôle de chaque classe du programme et de chaque action liée à celles-ci. Pour finir nous aborderons une légère conclusion, avec les éventuels problèmes, les points à améliorer et de l'expérience personnel.

B. Aide-mémoire

Liens utiles :

Voici les liens utiles pour l'apprentissage du Javascript :

- Mdn web docs : https://developer.mozilla.org/fr/
- W3school : https://www.w3schools.com/

Variables :

- Var : Permet de déclarer une variable qui est accessible sur la globalité du code, cela peut causer des problèmes car la variable peut être utiliser dans tout le code. Il faudrait plutôt utiliser « let » que « var ».

- Const : Permet de déclarer une variable const qui ne pourra pas être réaffectée après son initialisation, cela permet d'assurer que sa valeur reste constante tout au long du programme.

- Let : Permet de déclarer une variable qui est accessible seulement à l'intérieur du bloc de code. Par exemple, si on a une variable « let test = 32 ; » et qu'elle se trouve dans un bloc if, elle sera uniquement accessible dans ce même if. En dehors de ce bloc, elle ne sera pas accessible, cela permet de restreindre l'utilisation de la variable à la portée du bloc dans lequel elle a été créée.

Classes :

- Création d'une classe : Pour créer il faut tout simplement noter le type de cette class et son nom comme ci-dessous :

```
- class Snake{
- }
```

Modules :

- Import : Permet d'importer des variables, fonctions ou des objets depuis une page externe, cela permet d'utiliser les variables et de communiquer entre les différentes classes du code.

```
- import Snake from '/src/snake';
```

- Export : Permet d'exporter les variables, les fonctions ou les objets que l'on souhaite pouvoir utiliser dans les autres classes.

```
export default Snake;
```

Paramètres:

En JavaScript, les paramètres doivent toujours être déclarés dans la signature de la fonction pour pouvoir utiliser les variables que l'on souhaite. Donc dès que l'on souhaite utiliser une variable dans une fonction ont est obliger de la déclarer en paramètre.

Canvas :

Le Canvas est un composant HTML qui permet d'afficher un rendu dynamique (de la couleur, des images) dans des scripts en Javascript.

- fillStyle : Permet de changer la couleur de canvas (background et texte)
- fillRect : Permet de définir la position et la taille du canvas
- fillText : Permet de définir un texte et sa position
- font : Permet de définir la police et la taille du texte
- drawImage : Permet d'afficher une image en lui donnant sa taille et sa position
- clearRect : Permet de clear le canvas en lui donne une taille et une position

Objets :

En JavaScript on peut instancier un objet directement dans une variable comme cela :

```
const headSnake = { x: 0, y: 0 };
```

Notre objet aura donc une variable x qui stockera sa position x et une autre qui stockera sa position y, si nous voulons utiliser une des deux variables on peut le faire comme cela :

```
headSnake.x
```

Functions :

- => : Les fonctions fléchées sont une syntaxe Javascript pour définir des fonctions. Cela permet de créer des fonctions de manières plus simple et permet de simplifier le « this ». Par exemple, si le this est égal à 43 dans une classe, il restera toujours égal à 43, ce qui simplifie la compréhension du code et évite des erreurs liées à des changements inattendus.

```
const move = () => {
```

- « : » « ? » : La fonction « ? » et « : » permet de remplacer l'instruction « if » et « else », on l'utilise sous cette forme-là : « condition ? valeurSiVrai : valeurSiFaux ». Si la condition est vraie, la variable prend la valeur valeurSiVrai sinon elle prend la valeur valeurSiFaux.

Fonctions Javascript :

Les fonctions en Javascript s'utilisent comme les méthodes en C#.

```
function maFonction() {
  // Code de la fonction
}
```

Tableau :

- … : L'opérateur rest, permet de décomposer un tableau en une liste d'éléments individuel, cela permet donc de pouvoir parcourir chaque élément du tableau de manière plus simple.

```
- Let table = [ 1, 2, 3, 4, 5, 6, 7] ;
- Let [first, second, ...rest] = table ;
```

- push : Permet d'ajouter un seul ou plusieurs éléments à la fin du tableau et de nous retourner sa nouvelle taille.

```
- snake.push({ x: headSnake.x, y: headSnake.y });
```

- pop : Permet de supprimer le dernier éléments du tableau et d'y retourner ce même éléments.

```
- snake.pop() ;
```

- shift : Permet de supprimer le premier éléments du tableau et d'y retourner ce même éléments.

```
- snake.shift() ;
```

- unshift : Permet d'ajouter un seul ou plusieurs éléments au début du tableau et de nous retourner sa nouvelle taille.

```
- snake.unshift({ x: headSnake.x, y: headSnake.y });
```

Opérateur :

- Math.floor : Permet d'arrondir un nombre aux nombre inférieur
- Math.random : Permet de donner un nombre entre un x et un y (x et y sont inclus)

```
- Math.floor(Math.random() * 20) * gridSize;
```

Ecouteur d'événements :

Les écouteurs d'événements permettent de détecter et de réagir à des actions de l'utilisateur.

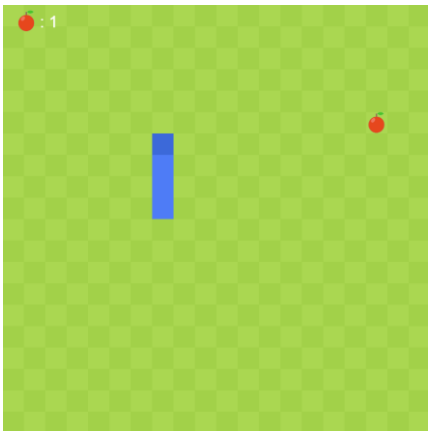
- addEventListener : Permet de détecter un événement de pression d'une touche du clavier.

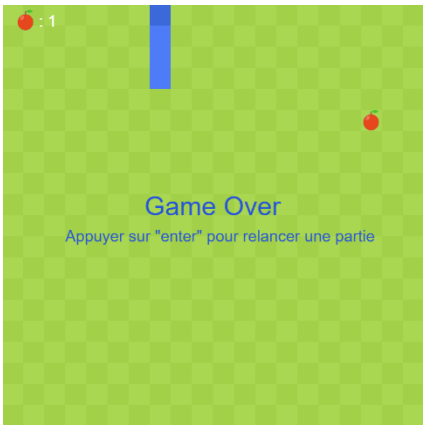
```
- document.addEventListener('keydown', function (event) {}
```

C. Design

Pour le design du jeu Snake, je me suis inspiré du Snake de Google. Notamment en ce qui concerne les couleurs et l'affichage de la pomme. Pour le menu, j'ai gardé une palette de couleurs similaire tout en ajoutant une police d'écriture joyeuse, pour garder une atmosphère amusante et joyeuse dans le jeu. Le but était de rendre une expérience agréable et divertissante pour les joueurs.

Voici le rendu du jeu :





D. Structure du code

En ce qui concerne la structure de mon code, j'ai opté pour une approche de séparation entre l'affichage et les actions du programme en créant plusieurs fichiers .js :

- **Main** : Cette partie agit comme le terrain de jeu, elle contient les déclarations de variables et les appels des méthodes à utiliser.
- **Snake** : Cette classe gère toutes les actions du serpent, elle stocke les informations et fonctionnalités associées à ses actions.
- **Apple** : Cette classe gère toutes les actions de la pomme, elle stocke les informations et fonctionnalités associées à ses actions.
- **Playground** : Cette classe est gère tous les affichages du programme, elle regroupe toutes les fonctionnalités liées à l'affichage.

Cette structure permet une meilleure organisation du code et une meilleure compréhension de ce que chaque classe fait et contient.

E. Analyse technique

Comme vu un peu plus haut, notre code est constitué de plusieurs classes et partie. Voici ci-dessous ce que chaque classes et partie effectue et de quoi elles sont constituées.

1. Main

Contexte :

Il contient toutes les appellent des méthodes, des déclarations des variables, tableaux et les function qui gèrent le fonctionnement du jeu.

Function :

- **Event :** Cette function permet d'écouter les touches de l'utilisateur, elle permet aussi qu'en fonction de la touche presser de gérer la direction à accorder au snake, de lancer le jeu depuis le menu d'accueil ou encore de relancer une partie.

```
document.addEventListener('keydown', function (event) {

  //Entrer des touches et de la direction à accorder (on ne peux pas accorder la direction si elle est op
  switch (event.key) {

    case 'w':
      if (direction != 'Down') {
        direction = 'Up'
      }
      break;

    case 's':
      if (direction != 'Up') {
        direction = 'Down'
      }
      break;

    case 'a':
      if (direction != 'Right') {
        direction = 'Left'
      }
      break;

    case 'd':
      if (direction != 'Left') {
        direction = 'Right'
      }
      break;

    case ' ':
      // Si on appuie sur espace et que le jeu n'est pas lancé on lance le jeu
      if (!gameStarted) {
        gameStarted = true;
      }

      // Si on appuie sur entrer et que le jeu est fini on relance le jeu
      if (gameOver || gameOver1) {
        // Relance le jeu
        location.reload();
      }
      break;
```

Méthodes :

- Dans notre main il y a plusieurs méthodes, il y a les méthodes qui s'occupent des actions et des fonctionnalités du jeu et les méthodes qui affichent**.** Ces deux types de méthodes sont séparé, en premier on appelle les méthodes actions et fonctionnalités et ensuite on appelle les méthodes d'affichage. Cela permet d'éviter les décalages et les problèmes entre le visuel et l'actions.

2. Playground

Contexte :

Elle contient toutes les méthodes qui s'occupe de l'affichage

Méthodes :

- **clearCanvas** : Efface tout le terrain de jeu

```
// Efface le canvas
clearCanvas() {
  // Clear toute la zone indiquée
  this.ctx.clearRect(0, 0, 800, 800)
}
```

- **drawSnake** : Dessine la tête du serpent et ensuite chaque partie de son corps

```
// Dessine le snake
drawSnake(snake, gridSize) {
  // Dessine la tete du serpent à une position définit
  this.ctx.fillStyle = '#3C69D9';
  this.ctx.fillRect(snake[0].x, snake[0].y, gridSize, gridSize);

  // Dessine le corps du serpent à une position définit
  //Boucle qui permet de parourir chaque partie du serpent sauf la tete
  for (let i = 1; i < snake.length; i++) {
    this.ctx.fillStyle = '#4E7CF6';
    this.ctx.fillRect(snake[i].x, snake[i].y, gridSize, gridSize);
  }

}
```

- **drawApple** : Affiche l'image de la pomme à une position aléatoire

```
// Dessine la pomme
drawApple(applePosition, gridSize, appleImage) {
  // Affiche l'image de la pomme à une position aléatoire
  this.ctx.drawImage(appleImage, applePosition.x, applePosition.y, gridSize, gridSize);
}
```

- **drawGameOver** : Affiche l'onglet Game-over avec le titre du menu et sur quelle touche il faut appuyer pour relancer le jeu

```
// Dessine game-over
drawGameOver() {
  // Titre
  this.ctx.fillStyle = '#2754dd ';
  this.ctx.font = '50px Arial';
  this.ctx.fillText('Game Over', 270, 400);
  // Texte
  this.ctx.font = '30px Arial';
  this.ctx.fillText('Appuyer sur "espace" pour relancer une partie', 120, 450);
}
```

- **drawScore** : Affiche le nombre de pomme manger de la partie avec une image de pomme

```
// Affiche le score
drawScore(scoreGame, appleImage, gridSize) {
  this.ctx.fillStyle = 'White';
  this.ctx.font = '30px Arial';
  this.ctx.drawImage(appleImage, 25, 10, gridSize, gridSize);
  this.ctx.fillText(': ' + scoreGame, 70, 42);
}
```

- **drawGrid** : Dessine le terrain de jeu et son quadrillage, pour afficher une cas sur deux d'une autre couleur on regarde tout simplement si la case est paire ou impaire et on alterne la couleur.

```
// Dessine la grille du jeu et le terrain
drawGrid(gridSize) {
  // Place la couleur sur chaque carreaux de la grille
  for (let x = 0; x < this.canvas.width; x += gridSize) {
    for (let y = 0; y < this.canvas.height; y += gridSize) {
      // On ajoute les couleurs différentes en fonction de si la case est paire ou impaire
      if ((x / gridSize + y / gridSize) % 2 == 0) {
        this.ctx.fillStyle = '#AAD751';
      }
      else {
        this.ctx.fillStyle = '#A2D149';
      }

      // Dessine la grille et le terrain
      this.ctx.fillRect(x, y, gridSize, gridSize);
    }
  }
}
```

- **drawMenu** : Affiche le menu d'accueil, le titre, le cadre et la touche sur laquelle il faut appuyer**.**

```
// Dessine le menu
drawMenu() {

  // Affichage du fond du menu
  this.ctx.fillStyle = '#A2D149';
  this.ctx.fillRect(0, 0, 800, 800);

  // Affichage du titre
  this.ctx.fillStyle = '#2754dd';
  this.ctx.font = '50px Broadway';
  this.ctx.fillText('Snake Game', 230, 100);

  // Affichage des contours

  // Bas
  this.ctx.fillStyle = 'white';
  this.ctx.fillRect(200, 125, 400, 4);

  // Haut
  this.ctx.fillStyle = 'white';
  this.ctx.fillRect(200, 40, 400, 4);

  // Affichage du texte
  this.ctx.fillStyle = '#2754dd';
  this.ctx.font = '40px Broadway';
  this.ctx.fillText('Appuyez sur espace pour jouer', 60, 400);

  // Affichage du texte ombre

  this.ctx.fillStyle = 'white';
  this.ctx.font = '40px Broadway';
  this.ctx.fillText('Appuyez sur espace pour jouer', 60, 398);
}
```

3. Snake

Contexte :

Elle contient toutes les actions du serpent et ces caractéristiques.

Constructor :

Notre snake à plusieurs caractéristiques qui sont définis dans le constructor les voici ci-dessous :

- **Speed** : La vitesse de déplacement du serpent
- **numBody** : Le nombre de partie de corps que le serpent aura par défault

Méthodes :

- **moveSnake** : Déplace la tête et chaque partie du corps selon la direction choisie, le principe est de sauvegarder à chaque étape du déplacement la position de la tête. Grace à cela chaque partie du corps peut suivre cette position. Ensuite chaque partie du corps sauvegarde sa position et la transmet à la partie derrière elle. Cela permet que chaque partie du corps suit le bon chemin sans sauter d'étape.

```
// Déplacement du snake
moveSnake(headSnake, direction, snake, gridSize) {
  //On stock la position de la tete du snake
  const positionHead = { x: headSnake.x, y: headSnake.y };

  //On déplace la tete du snake en fonction de la direction
  switch (direction) {
    case 'Up':
      headSnake.y -= gridSize;
      break;
    case 'Down':
      headSnake.y += gridSize;
      break;
    case 'Left':
      headSnake.x -= gridSize;
      break;
    case 'Right':
      headSnake.x += gridSize;
      break;
  }

  // Boucle qui parcours chaque partie du corps du snake sauf la tete
  for (let i = snake.length - 1; i > 0; i--) {
    // On sauvegarde la position de chaque partie du corps
    const updatePosition = { x: snake[i].x, y: snake[i].y };

    // On déplace chaque partie du corps à la position de la partie précédente
    snake[i].x = snake[i - 1].x;
    snake[i].y = snake[i - 1].y;
  }

  // On remet la postion de la tete à sa position actuel
  snake[0].x = positionHead.x;
  snake[0].y = positionHead.y;
}
```

- **toucheBorder** : Vérifie si la tête du serpent rentre en collision avec la bordure du terrain, dans ce cas on retourne que la partie est perdu.

```
// Vérifie si la tete du snake touche la bordure du terrain
toucheBorder(headSnake, canvas) {
  //On vérifie la position de la tete du snake par apport a la position de la bordure
  if (headSnake.x < 0 || headSnake.y < 0 || headSnake.x >= canvas.width || headSnake.y >= canvas.height) {
    return true;
  }
  return false;
}
```

- **toucheBody** : Vérifie si la tête du serpent rentre en collision avec son corps, dans ce cas on retourne que la partie est perdu.

```
// Vérifie si la tete du snake touche une partie de son corps
toucheBody(headSnake, snake) {
  //Boucle qui parcours chaque partie du corps
  for (let i = 0; i < snake.length; i++) {
    //On vérifie la position de chaque partie du corps par apport a la tete
    if (headSnake.x == snake[i].x && headSnake.y == snake[i].y) {
      return true;
    }
  }
  return false;
}
```

- **eatApple** : Vérifie si la tête du serpent touche la pomme, si c'est le cas on détecte cela comme si il mange la pomme et on lui ajoute une partie du corps à la fin.

```
// Vérifie si la tete du snake mange la pomme
eatApple(applePosition, headSnake, snake) {
  //On vérifie si la position de la tete est égal a la postion de la pomme
  if (headSnake.x == applePosition.x && headSnake.y === applePosition.y) {
    //Ajout d'une nouvelle partie au corps du snake
    snake.push({ x: headSnake.x, y: headSnake.y });
    return true;

  }
  return false;
}
```

- **updateScore** : Vérifie que si on mange une pomme cela nous ajoute +1 à notre score.

```
// Vérifie si la tete du snake mange la pomme
updateScore(applePosition, headSnake, scoreGame) {
  //On vérifie si la position de la tete est égal a la postion de la pomme
  if (headSnake.x == applePosition.x && headSnake.y === applePosition.y) {
    //Augmente le score
    scoreGame += 1;
  }
  return scoreGame;
}
```

4. Snake

Contexte :

Elle contient toutes les actions de la pomme

Méthodes :

- **randomPosition :** Définis des coordonnées x et y aléatoire à la pomme, cependant il y a quelque critères, la pomme doit être positionner dans une des case du terrain de jeu et elle doit apparaitre dans le terrain de jeu c'est pour cela que l'on fait x20 et gridSize

```
//Donne des coordonnées aléatoire a notre pomme
randomPosition(gridSize, applePosition) {
    applePosition.x = Math.floor(Math.random() * 20) * gridSize;
    applePosition.y = Math.floor(Math.random() * 20) * gridSize;

    return applePosition;
}
```

F. Utilisation Chatgpt

Dans ce projet, j'ai utilisé ChatGPT quelques fois, mais cela ne m'a pas été très utile. En effet, au début du projet, je ne connaissais pas du tout la syntaxe du JavaScript.

Je demandais donc à ChatGPT quelques exemples de syntaxe, mais il me fournissait des choses beaucoup plus compliquées ou alors des choses qui n'avaient rien à avoir. Cela m'a fait la même chose lorsque je lui demandais quels étaient les problèmes dans mon code ; il me sortait des choses complètement absurdes ou beaucoup trop compliquées.

J'ai donc arrêté de l'utiliser et me suis tourné vers W3Schools et MDN Web Docs, qui sont des sites web qui offrent la syntaxe JavaScript avec des explications et des exemples. ChatGPT ne m'a donc pas été utile car j'ai su trouver une meilleure aide ailleurs.

G. Test

Nom	Description	Résultat
Menu accueil	Menu d'accueil du jeu, le menu affiche sur quelle touche cliqué et une fois celle-ci appuyé cela lance le jeux.	OK
Game Over	Un onglet game over apparait en fin de partie et permet de relancer une partie en appuyant sur une touche définit	OK
Déplacement par défaut	Le snake se déplace en continue sur la droite au lancement du jeu.	OK
Déplacement gauche	Le snake se déplace en continue sur la gauche, quand on appuie sur la touche « a »	OK
Déplacement droite	Le snake se déplace en continue sur la droite, quand on appuie sur la touche « d »	OK
Déplacement haut	Le snake se déplace en continue vers le haut, quand on appuie sur la touche « w »	OK
Déplacement bas	Le snake se déplace en continue vers le bas, quand on appuie sur la touche « s »	OK
Collision bordure	Quand le snake touche la bordure du terrain de jeu, la partie s'arrete et ce met en mode game over	OK
Collision corps	Quand le snake touche une partie de son corps, la partie s'arrête et ce met en mode game over	OK
Mange pomme	Au moment où la tête du snake touche la pomme, celle-ci disparait et une nouvelle apparait à un autre endroit	OK
Augmente Score	Un score est visible et dès qu'une pomme est manger celui-ci augmente de +1	OK
Augmente snake	Au moment où le snake mange une pomme, une nouvelle partie de corps apparait à la toute fin de sa queue	OK

H. Conclusion

En conclusion, ce projet m'a appris la syntaxe JavaScript et l'utilisation de ses fonctions. Il m'a également permis de créer un aide-mémoire qui me sera certainement utile à l'avenir. J'ai également découvert de nouveaux sites web d'aide tels que W3Schools. Par ailleurs, il m'a fait réaliser que l'utilisation de ChatGPT n'est pas toujours bénéfique, car il ne résout pas toujours nos problèmes de manière optimale.

De plus, si ce projet devait être refait, je m'informerais davantage sur la manière d'utiliser les fonctions JavaScript pour réaliser le jeu Snake. Dans mon jeu actuel, je n'ai pas utilisé de forEach, de "?" ou encore de fonction fléchée.

I. Sources

- Mdn web docs : https://developer.mozilla.org/fr/
- W3school : https://www.w3schools.com/
- ChatGPT : https://chat.openai.com/