TP1 : Jeu Pokémon avec Node.js

Réalisé par :

Hiba Moustaoudi

Oumaima Ait Rami

Objectifs de ce tp :

- Créer un mini-jeu Pokémon en utilisant Node.js et l'API PokéAPI.
- Permettre au joueur de choisir son propre Pokémon.
- Faire jouer le joueur contre un bot qui choisit ses mouvements aléatoirement.
- Prendre en compte l'accuracy des mouvements et les points de puissance (PP) restants.
- Afficher les dégâts causés et le vainqueur de la bataille.

Structure du code :

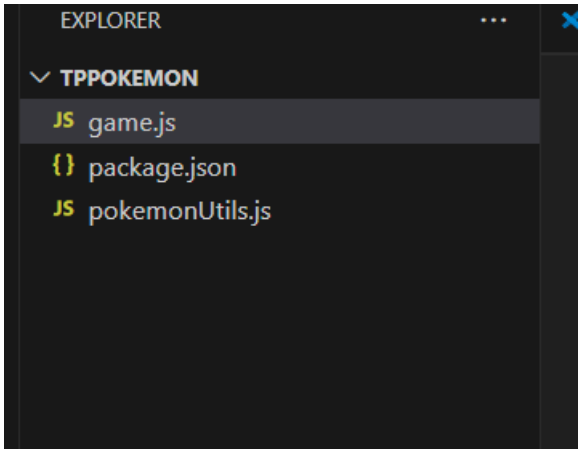
1. `pokemonUtils.js` : Cette fichier contient la fonction `getPokemonData` qui permet de récupérer les données d'un Pokémon donné depuis l'API PokéAPI. Il utilise le module `https` pour faire les requêtes.
2. `game.js` : Ce fichier principal contient toute la logique du jeu :
    a. La classe `Pokémon` qui représente un Pokémon avec ses attributs (nom, points de vie, mouvements).
    b. La fonction `startGame` qui gère le déroulement de la partie.
    c. La fonction `createPlayerPokemon` qui crée le Pokémon du joueur en lui demandant son choix.
    d. La fonction `createEnemyPokemon` qui crée aléatoirement le Pokémon adverse.
    e. Les fonctions `playerTurn` et `enemyTurn` qui simulent les tours de jeu.

1- Création d’un nouveau répertoire et initialisation d’un nouveau projet Node.js

```
PS C:\Users\MSI\Desktop\tpPokemon> npm init -y
Wrote to C:\Users\MSI\Desktop\tpPokemon\package.json:

{
  "name": "tppokemon",
  "version": "1.0.0",
  "main": "game.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Création du package.json :



1-Importation du module HTTPS :

```
const https = require('https');
```

Cette ligne importe le module HTTPS intégré de Node.js, qui permet de faire des requêtes HTTPS sécurisées.

2. Définition de la fonction asynchrone getPokemonData :

```
async function getPokemonData(pokemonName) {
  return new Promise((resolve, reject) => {
    const options = {
```

Cette fonction prend un nom de Pokémon en paramètre et retourne une Promise. Elle utilise une structure de Promise pour gérer l'opération asynchrone de récupération des données.

3. Configuration des options de la requête :

```
const options = {
  hostname: 'pokeapi.co',
  port: 443,
  path: `/api/v2/pokemon/${pokemonName.toLowerCase()}`,
  method: 'GET',
};
```

Ces options définissent les détails de la requête HTTPS, y compris l'hôte (pokeapi.co), le port (443 pour HTTPS), le chemin de l'API, et la méthode HTTP (GET).

4. Création et envoi de la requête :

```
const req = https.request(options, (res) => {
  let data = '';

  res.on('data', (chunk) => {
    data += chunk;
  });

  res.on('end', () => {
    if (res.statusCode === 200) {
      try {
        const parsedData = JSON.parse(data);
        resolve(parsedData);
      } catch (e) {
        console.error(`Error parsing JSON for ${pokemonName}: `, e);
        reject(e);
      }
    } else {
      console.error(`Error fetching data for ${pokemonName}: ${res.statusCode} - ${res.statusMessage}`);
      console.error('Request options:', options);
      console.error('Response data:', data);
      reject(`Error fetching data for ${pokemonName}: ${res.statusCode} - ${res.statusMessage}`);
    }
  });
});
```

   Ceci crée une requête HTTPS avec les options spécifiées et définit un callback pour gérer la réponse.

5. Gestion de la réponse :

```
const req = https.request(options, (res) => {
  let data = '';

  res.on('data', (chunk) => {
    data += chunk;
  });
```

Ces lignes accumulent les morceaux de données reçus dans la variable 'data'.

6. Traitement de la réponse complète :

```
    res.on('end', () => {
      if (res.statusCode === 200) {
        try {
          const parsedData = JSON.parse(data);
          resolve(parsedData);
        } catch (e) {
          console.error(`Error parsing JSON for ${pokemonName}: `, e);
          reject(e);
        }
      } else {
        console.error(`Error fetching data for ${pokemonName}: ${res.statusCode} - ${res.statusMessage}`);
        console.error('Request options:', options);
        console.error('Response data:', data);
        reject(`Error fetching data for ${pokemonName}: ${res.statusCode} - ${res.statusMessage}`);
      }
    });
  });
```

Une fois toutes les données reçues, ce code vérifie le code de statut de la réponse. Si c'est 200 (OK), il tente de parser les données JSON. En cas de succès, il résout la Promise avec les données parsées. Sinon, il rejette la Promise avec une erreur.

7. Gestion des erreurs de requête :

```
    req.on('error', (e) => {
      console.error(`Error fetching data for ${pokemonName}: `, e);
      reject(e);
    });

    req.end();
  });
}
```

Ce code gère les erreurs qui peuvent survenir pendant la requête elle-même.

8. Exportation de la fonction :

```
module.exports = { getPokemonData };
```

Cette ligne exporte la fonction getPokemonData pour qu'elle puisse être utilisée dans d'autres fichiers.

Ce code crée essentiellement une fonction réutilisable pour récupérer des données de Pokémon depuis l'API PokeAPI, en gérant les erreurs potentielles et en retournant les données sous forme de Promise.

Pour le deuxième fichier : game.js

1. Importation de la fonction getPokemonData :

```
const { getPokemonData } = require('./pokemonUtils');
```

Cette ligne importe la fonction getPokemonData du fichier pokemonUtils.js que nous avons expliqué précédemment.

2. Définition de la classe Pokemon :

```
class Pokemon {
  constructor(name, hp = 300, moves = []) {
    this.name = name;
    this.hp = hp;
    this.moves = moves;
  }

  attack(targetPokemon, moveIndex) {
    const move = this.moves[moveIndex];
    if (move.pp > 0) {
      if (Math.random() <= move.accuracy) {
        console.log(`${this.name} used ${move.name}!`);
        move.pp--;
        targetPokemon.hp -= move.power;
        console.log(`${targetPokemon.name} took ${move.power} damage!`);
      } else {
        console.log(`${move.name} missed!`);
      }
    } else {
      console.log(`${move.name} has no PP left!`);
    }
  }
}
```

Cette classe définit un Pokémon avec un nom, des points de vie (HP) et des mouvements. La méthode attack gère la logique d'attaque, prenant en compte la précision et les PP du mouvement.

3. Fonction startGame :

```
async function startGame() {
    try {
      const playerPokemonName = await askPlayerForPokemon();
      const playerPokemon = await createPlayerPokemon(playerPokemonName);
      const enemyPokemon = await createEnemyPokemon();

      while (playerPokemon.hp > 0 && enemyPokemon.hp > 0) {
        playerTurn(playerPokemon, enemyPokemon);
        if (enemyPokemon.hp > 0) {
          enemyTurn(enemyPokemon, playerPokemon);
        }
      }

      if (playerPokemon.hp <= 0) {
        console.log(`You lost! ${enemyPokemon.name} defeated ${playerPokemon.name}.`);
      } else {
        console.log(`You won! ${playerPokemon.name} defeated ${enemyPokemon.name}.`);
      }
    } catch (error) {
      console.error('Error starting the game:', error);
    }
  }
```

Cette fonction asynchrone gère le déroulement du jeu. Elle demande au joueur de choisir un Pokémon, crée les Pokémons du joueur et de l'ennemi, puis gère les tours de jeu jusqu'à ce qu'un Pokémon soit vaincu.

4. Fonction askPlayerForPokemon :

```
async function askPlayerForPokemon() {
  const readline = require('readline').createInterface({
    input: process.stdin,
    output: process.stdout
  });

  return new Promise((resolve) => {
    readline.question('Choose your Pokémon (e.g., Pikachu, Charmander, Bulbasaur): ', (pokemonName) => {
      readline.close();
      resolve(pokemonName.toLowerCase());
    });
  });
}
```

Cette fonction utilise le module readline pour demander au joueur de choisir un Pokémon via la console.

5. Fonctions createPlayerPokemon et createEnemyPokemon :

```
async function createPlayerPokemon(pokemonName) {
    const playerPokemonData = await getPokemonData(pokemonName);
    const playerMoves = playerPokemonData.moves.slice(0, 5).map(move => ({
      name: move.move.name,
      power: Math.floor(Math.random() * 50) + 1,
      accuracy: Math.random(),
      pp: 10
    }));
    return new Pokemon(pokemonName, 300, playerMoves);
  }
```

Ces fonctions créent les objets Pokémon pour le joueur et l'ennemi en utilisant les données de l'API.

6. Fonctions playerTurn et enemyTurn :

```
function enemyTurn(enemyPokemon, playerPokemon) {
  console.log(`${enemyPokemon.name}'s turn:`);
  const moveIndex = Math.floor(Math.random() * enemyPokemon.moves.length);
  enemyPokemon.attack(playerPokemon, moveIndex);
}
```

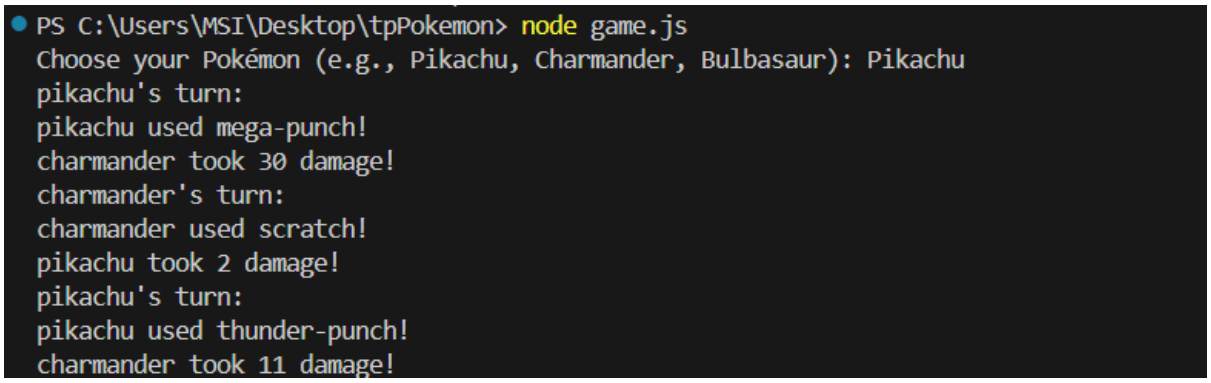
```
  function playerTurn(playerPokemon, enemyPokemon) {
    console.log(`${playerPokemon.name}'s turn:`);
    const moveIndex = Math.floor(Math.random() * playerPokemon.moves.length);
    playerPokemon.attack(enemyPokemon, moveIndex);
  }
```

Ces fonctions gèrent les tours de jeu pour le joueur et l'ennemi, choisissant aléatoirement un mouvement et effectuant l'attaque.

7. Lancement du jeu :

```
startGame();
```

Cette ligne lance le jeu en appelant la fonction startGame.


PS C:\Users\MSI\Desktop\tpPokemon> node game.js
Choose your Pokémon (e.g., Pikachu, Charmander, Bulbasaur): Pikachu
pikachu's turn:
pikachu used mega-punch!
charmander took 30 damage!
charmander's turn:
charmander used scratch!
pikachu took 2 damage!
pikachu's turn:
pikachu used thunder-punch!
charmander took 11 damage!