TME 2 : JavaScript

1 Échauffement

Q qz0.1 Event Loop : observez ce que fait ce code et interprétez

```
fs = require('fs')

// Attends ms millisecondes (non asynchrone)
function wait(ms) {
    var start = +(new Date());
    while (new Date() - start < ms);
    console.log("Wait is over....")
}

// Planifie un message dans 100ms
setTimeout(() => {
    console.log("Message 1")
}, 100)

wait(1000)

// Planifie un nouveau message dans 100ms
setTimeout(() => {
    console.log('Message 2')
}, 100)

// Appel synchrone
let s = fs.readFileSync(__filename)
console.log("Sync file %s", __filename)

// Appel asynchrone
fs.readFile(__filename, () => {
    console.log("Async file read")
})
```

2 Jeu du pendu

Nous allons développer un jeu du pendu. Le but est de trouver un mot en proposant les lettres qui le compose en se trompant moins d'un certain nombre de fois (ex. moins de 5 lettres proposées qui n'appartiennent pas au mot). Nous allons également utiliser un temps limité (ex. 10 secondes). Merci de vous référer à la fin du document pour une liste de pointeurs vers la documentation JavaScript.
Avant de commencer, vous allez créer le répertoire qui contiendra le projet.

```
mkdir technoweb-tme2-pendu
cd technoweb-tme2-pendu
npm init
```

Vous répondrez aux différentes questions et vérifirez que le fichier `package.json` a bien été créé.
Téléchargez le fichier "dico.txt" et placez dans le répertoire principal du projet.
Vous placerez les fichiers dans le répertoire `src` avec les fichiers suivants :

— Le fichier `package.json` créé par `npm init` contenant la liste des dépendences et les informations sur le projet Node.JS.

- Le fichier principal de l'application sera le fichier `src/app.js` que vous pourrez exécuter en tapant la commande `node src/app.js`.
- Le fichier permettant la lecture du dictionnaire de mots `src/dico.js`
- Le fichier contenant le code du pendu `src/pendu.js` (et éventuellement `src/ticking_pendu.js` i vous implémentez la version avec temps limité).
- Le fichier permettant de tester le code de lecture du dictionnaire `test/testdico.js`

Attention : CommonJS vs ES Module Comme vu en cours, il y a actuellement deux façons de déclarer des exportations et importations de symboles. Par défaut, Node.JS utilise le standard CommonJS (avec `module.exports` et `require`). Si vous souhaitez utiliser le standard ESM (`export` et `import`), il faut renommer tous les fichiers en utilisant l'extension `.mjs` au lieu de `.js`.
Afin de permettre d'interrompre le programme avec la combinaison de touches control-C, placez ce code au début du fichier `src/app.js` :

```
// On s'assure que control-C nous permet d'interrompre
// l'exécution du programme
const readline = require('readline');
readline.emitKeypressEvents(process.stdin);
process.stdin.setRawMode(true);
process.stdin.on('keypress', (_str, key) => {
    if (key.ctrl && key.name == "c") {
        console.log("Quitting with control-c");
        process.exit();
    }
});
```

2.1 Lecture du dictionnaire

Le code de lecture du dictionnaire sera écrit dans le fichier `src/dico.js`.

Q qz0.2 Lire le dictionnaire `dico.txt`. Vous écrirez une fonction `litSync(filepath, minlength)` qui renvoie une liste de mots (tableau) contenus dans le fichier `filepath`. Vous pouvez filtrer les mots pour ne pas inclure les mots trop courts (inférieur à `minlength` caractères).

Q qz0.3 Version asynchrone (Promise) : lire le dictionnaire `dico.txt`. Vous écrirez une fonction `lit(filepath, minlength)` qui renvoie une promesse (Promise) d'une liste de mots (tableau). Vous travaillerez donc de manière asynchrone (on peut aussi le faire de manière synchrone, mais autant vous habituer).

Q qz0.4 Version asynchrone (async) : Vous écrirez une fonction `async litAsync(filepath, minlength)` qui renvoie une promesse (Promise) d'une liste de mots (tableau). Vous utiliserez le sous-module `fs.promises` pour lire le fichier.

Tests

Afin de tester vos trois fonctions, vous copierez le code suivant dans un fichier nommé `test/testdico.js`.

```
const assert = require("assert")
const dico = require("../src/dico.js")
const path = require('path')
const dicopath = path.join(path.dirname(__dirname), "dico.txt")

describe("Lire un dictionnaire", () => {
    it("sync", () => {
        let words = dico.litSync(dicopath, 5)
        assert.strictEqual(words[0], "ANGLE")
        assert.strictEqual(words[20], "MEUBLE")
    })

    it("promise", async () => {
        let words = await dico.lit(dicopath, 5)
```

```
        assert.strictEqual(words[0], "ANGLE")
        assert.strictEqual(words[20], "MEUBLE")
    })

    it("async", async () => {
        let words = await dico.litAsync(dicopath, 5)
        assert.strictEqual(words[0], "ANGLE")
        assert.strictEqual(words[20], "MEUBLE")
    })
})
```

Vous ajouterez le module `mocha` avec la commande `npm add -D mocha` (l'option `-D` indique que le module est utile pour le développement mais pas pour l'exécution). Vous modifierez le `package.json` en ajoutant la commande pour le test.

```
...
    "scripts": {
        "test": "mocha test"
    },
...
```

Finalement, vous exécuterez le test en tapant la commande `npm run test`. Si des erreurs se produisent, il faudra corriger votre code de lecture du dictionnaire en conséquence !

2.2 Classe Pendu

La classe `Pendu` sera définie dans un fichier `src/pendu.js`.

Q qz0.5 Créer une classe `Pendu` avec comme constructeur qui prendra deux paramètres, le mot à trouver et le nombre d'erreurs maximum. Afin de spécifier un nombre par défaut, on peut utiliser l'instruction suivante dans le constructeur :

```
// Permet d'initialiser à 5 si max_errors
// n'est pas transmis au constructeur
this.max_errors = max_errors || 5
```

. Elle initialisera les variables d'instance (`this.IDENTIFIANT`) suivantes :

- Le nombre d'erreurs courant (`errors`) avec comme valeur intiale zéro
- Le nombre de lettres trouvées (`foundcount`) avec comme valeur intiale zéro
- Un tableau contenant "_" autant de fois qu'il y a de lettre : cela permettra de savoir quelles lettres ont été trouvées ("_" signifiant non trouvée)

Q qz0.6 Créer une méthode `show()` qui affiche l'état actuel du jeu (nombre d'erreurs et lettre trouvées ou "_"). Testez l'objet depuis app.js en exécutant le code `new Pendu("POMME", 5).show()` ; vous devriez voir apparaitre

```
Erreurs: 0/5
-----
```

Note : Vous pouvez utiliser le code suivant pour effacer l'écran afin de rendre l'affichage plus lisible lors des parties (`readline` est un module Node.js) :

```
readline.cursorTo(process.stdout, 0,0)
readline.clearScreenDown(process.stdout)
```

Q qz0.7 Créez la méthode `keypressed(c)` où `c` est une lettre (string). Cette méthode met à jour les différentes variables d'instance correspondantes, et renvoie 1 si la partie est gagnée, 0 si perdue, et `undefined` sinon (pas de `return`).
Testez l'objet depuis app.js en exécutant le code vous devriez voir apparaitre

```
const pendu = new Pendu("POMME", 5)
console.log(pendu.keypressed('o'))
pendu.show()
console.log(pendu.keypressed('r'))
pendu.show()
```

La sortie attendue est :

```
r = undefined
Erreurs: 0/5
_O___
r = undefined
Erreurs: 1/5
_O___
```

2.3 Début du jeu

Q qz0.8 Ajoutez une méthode `play` à votre classe `Pendu` afin de débuter la partie. Cette méthode renvoie une promesse qui est réalisée lorsque la partie est gagnée et rejetée sinon. Afin de pouvoir réagir au clavier, vous vous inspirerez du code ci-dessous qui permet d'afficher chaque lettre.

```
const readline = require('readline');
readline.emitKeypressEvents(process.stdin);
process.stdin.setRawMode(true);

const f = (str, key) => {
  if (key.ctrl && key.name === 'c') {
    process.stdin.pause()
  } else {
    console.log(`You pressed the "${str}" key`);
    console.log()
    console.log(key)
    console.log()
  }
}

process.stdin.on('keypress', f);

// Ne pas oublier de faire process.stdin.removeListener("keypress", f)
// afin de "nettoyer le handler"
```

Q qz0.9 Dans `src/app.js`, vous allez ajouter le code qui lit le dictionnaire en appelant `lit`, choisit un mot au hasard (en utiliser la librairie `random` que vous installerez avec `npm add random`) puis appelle `pendu.play()`.

2.4 Ajout d'un temps limité

Nous allons maintenant rendre plus dur le jeu en donnant un temps limité au joueur.

Q qz0.10 Créer un fichier `src/ticking_pendu.js` qui définit une classe `TickingPendu` en héritant de `Pendu`. Puis vous

1. Définirez un constructeur qui prend en plus des autres paramètres le nombre de seconde.
2. Étendrez (utilisez `super`) la méthode `show` pour qu'elle affiche le temps restant ;
3. Redéfinirez la méthode `play` pour prendre en compte le temps qui passe (utilisez la fonction avec callback `setInterval`). Pour pouvoir gérer le fait de perdre une partie soit faute de temps, soit à cause des erreurs, pous utiliserez la librairie Node.JS `events`, et en particulier la classe `EventEmitter` qui permet de définir un canal de communication qui servira à transmettre le fait que la partie a été gagnée ou perdue.

2.5 Bonus : ajouter du typage : Typescript

Renommez tous les fichiers ".js" en ".ts" et créez un fichier `tsconfig.json` contenant le JSON suivant :

```
{
  "compilerOptions": {
    "target": "es2017",
    "module": "commonjs",
    "rootDir": "src/",
    "removeComments": false,
    "strict": true,
    "moduleResolution": "node",
  }
}
```

Ensuite, installez les modules nécessaires à la transpilation (typescript pour la transpilation, et `@types/...` pour les fichiers contenant des informations de typages sur les modules importés) :

```
npm add -D typescript @types/node @types/random
```

Pour transpiler et exécuter, tapez

```
    npx tsc && node dist/app.js
```

Une autre option, pour accélerer la compilation, et d'utiliser `npx tsc -w` dans un terminal : ceci compilera on continu. Notez qu'avec un éditeur moderne, les erreurs vous seront signalées directement dans l'éditeur !
Maintenant, plongez-vous dans la documentation TypeScript...

Pointeurs utiles

JavaScript

Vous trouverez la documentations pour :

- Générale
- Les chaînes de caractères (en particulier, `.split()`, `.length` et `.trim()`) ;
- Les tableaux (en particulier `.filter()`, `.join()`)
- Les promesses (Promise) et Les fonctions async(hrones)
- Gestion du temps avec setInterval

Node.JS

- Le répertoire courant dans un script Node.JS est donné par `__dirname` ;
- Fonctions permettant d'accéder aux fichiers : module fs et fs.promises de Node.JS
- Fonctions permettant de travailler avec les chemins de fichier : module path de Node.JS
- Pour gérer des canaux de communcations, utilisez EventEmitter.