M3104 – Programmation Web côté serveur (2018/2019)

TDm 9

Le code source de votre application et un rapport seront à rendre sur la plateforme UMTice. Le contenu du rapport est précisé à la fin du sujet.

NB : ce TP est prévu sur 3 séances. L'archive est à rendre au plus tard 2 semaines après la dernière séance.

Partie 1 : Description générale

Lors de ce TD / mini-projet, vous allez concevoir et développer une application Node.js permettant à des applications de communiquer avec un « serveur », qui compilera les infos transmises par tous les clients et leur redistribuera des « informations ».

Pour ce faire, vous utiliserez l'environnement « Node.js portable » disponible à l'adresse suivante : https://nodejs.org/en/download/ (**Version Windows Binary**). Les fichiers .js sont à déposer dans un dossier de votre choix (/Data, par exemple, ou un sous-répertoire de Data). Ils s'exécutent **en ligne de commande**. Exemple : `node.exe Data\index.js`

Le principe de l'application que vous devez concevoir et implémenter est le suivant :

- chaque client peut communiquer avec le serveur afin de lui **envoyer des données** (position géo-localisée, réussite d'un action, etc.). Cette transmission des données se fait en appelant une URL spécifique et en lui transmettant des **données en POST ou GET**
- le serveur **consolide et stocke** les données reçues (stockage volatile, i.e. dans **des variables locales**, ce qui induit un retour à l'état initial à chaque redémarrage du serveur, ou dans des fichiers **JSON**)
- les clients **interrogent** le serveur (appel d'une URL spécifique) afin de **recevoir** les données consolidées. Les données transmises par le serveur sont au **format JSON**.

Partie 2 : Instructions pour le rendu

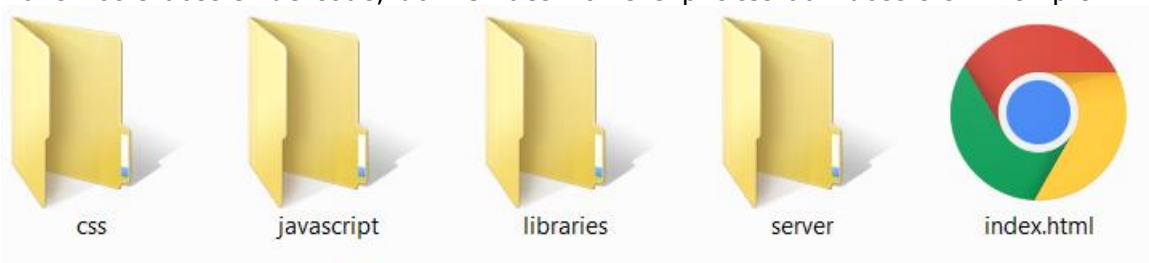
Archive

Votre archive devra être de la forme suivante :

- **Nom_Prenom_tpnodejs.zip** (Exemple : `Couland_Quentin_tpnodejs.zip`). Vérifiez le format de l'archive (je veux du **.zip** et pas autre chose). Si ce format n'est pas respecté, pénalité

- A la racine de l'archive : le rapport, puis le dossier contenant votre code (ce dossier peut (et je dirais même doit) contenir d'autres sous-dossiers, en fonction de comment votre code est organisé)

- Dans votre dossier de code, donnez des noms explicites aux dossiers. Exemple :



Conseil : téléchargez les bibliothèques que vous utilisez, plutôt que de les importer d'internet (jQuery, bootstrap, etc.)

- Idem pour les noms de fichier/fonctions/variables/etc.

Rapport

Le rapport devra impérativement contenir (dans l'ordre suivant) :

- La version de Node.js utilisée (prenez la **10.13.0** de préférence)

- Une description du projet et, si votre application a un thème, une petite présentation du thème en question

- Une explication succincte de la hiérarchie de votre archives (un petit schéma avec les dossiers, une explication de la logique de découpage de votre projet si elle est spéciale, etc.)

- Ecrivez (et soyez honnête) ce que vous avez fait ou non. Si vous avez commencé à faire quelque-chose et que ça ne fonctionne pas, précisez-le aussi

- Pour chaque variable globale de votre code : son nom, ainsi que son utilité

- Pour chaque fonctionnalité de votre code : la fonction associée, les paramètres d'entrées et le type attendu (soyez explicite sur le nom, ou alors précisez ce que c'est), la sortie de la fonction et son type (s'il y en a une)

Exemple 1 : **objet**

Nom : *player*
Utilité : L'objet qui contient les données du joueur

Exemple 2 : **fonctionnalités**

Fonctionnalité : Aller dans une direction
Nom de la fonction : *moveToDirection(direction)*
Description : Permet de se déplacer dans le monde
Paramètres : *direction*: la direction dans laquelle le joueur va bouger
Sorties : Le succès ou non du déplacement
Logique : Aucune

Note : dans cet exemple, il est inutile de détailler la logique de la fonction, car elle est simple et le nom est explicite.

- Pour chaque fonction du serveur Node.js, un exemple d'URL valide, ainsi que le bouton ou la fonction qui permet d'appeler cette URL

Exemple 1 : **URL**

Fonctionnalité : sauvegarder le nom du joueur
URL exemple : http://127.0.0.1:8080/save?name=slt_sava
Appel : fonction *sauvegarder_pseudo_joueur* (bouton **Sauvegarder**, en bas de la page **index.html**)

Vous êtes bien sûr libre de rajouter des choses (du contenu, des images, des dessins, des compliments, etc.), mais il faut au minimum cette base. **Le rapport comptera dans la note finale**.

Conseils en vrac :

- Ctrl + Shift + i : ouvre le panel d'outils du développeur, qui contient entre autre un onglet Console, utile pour voir les messages d'erreurs
- Si vous rechargez la page, le contenu de la console disparaît. Pas pratique pour débugger quand on doit rafraîchir la page. Pour éviter cela, Console Settings -> Preserve log
- Cette console permet également de taper du JS dedans, ce qui permet de faire des tests à la volée
- Si vous avez besoin d'aide, n'hésitez pas à m'appeler. Ce n'est pas parce-que c'est noté que je ne vous aiderai pas

Partie 2 : Description détaillée du sujet proposé

Connaissez-vous les **RPG** ?

Les **R**ole **P**laying **G**ames (**J**eux **d**e **R**ôles en français) sont des jeux où, généralement, vous dirigez un personnage que vous aurez préalablement créé, pour lui faire vivre moult aventures afin d'amasser des richesses, de la gloire, du pouvoir, etc.

Exemples : Fallout 1 & 2, Diablo, Dark Souls, Golden Sun, South Park: The Stick of Truth, Undertale, etc.

Dans ce TP, il vous sera demandé de réaliser un **RPG** (sur le thème de votre choix), avec une architecture client/serveur (oui alors par contre faut rester calme, ça ne sera probablement pas aussi bien que ceux cités au-dessus). Les noms de fonction, fichier, variable, etc. sont données à titre indicatif, libre à vous d'en changer également.

Question 1.1 : Créez un fichier **Javascript** dans lequel vous déclarerez un objet représentant le *joueur*, qui aura au moins les caractéristiques suivantes : *nom*, *points de vie (hp), force*, *agilité*, *intelligence*.

Question 1.2 : Créez une page **HTML** contenant à minima :

- **a)** Des emplacement pour rentrer le *nom*, les *points de vie*, la *force*, *l'agilité* et *l'intelligence* du personnage
- **b)** Un bouton qui, lorsqu'il est cliqué, permet de créer le personnage en **Javascript**
- **c)** Un emplacement pour afficher les caractéristiques du personnage

Vous avez donc maintenant un personnage, prêt à partir à l'aventure. Cependant, sans moyen de se déplacer, difficile de continuer.

Question 2.1 : Ajoutez un moyen de :

- **a)** **Javascript** : se déplacer dans une direction choisie (commençons par *Nord*, *Sud*, *Est*, *Ouest*)
- **b)** **HTML** : appeler la ou les fonction(s) de déplacement

Formidable, votre personnage se déplace ! Malheureusement, il n'y a pas de moyen de le savoir, vu que rien ne s'affiche à l'écran.

Question 2.2 : Ajoutez un emplacement sur la page **HTML** qui affiche la direction préalablement choisie.

Ooooooook, maintenant on sait où on va… Mais on ne sait pas d'où on vient (c'est beau, hein ?). Il serait judicieux de pouvoir garder en mémoire la position actuelle dans le monde.

Question 2.3 : Ajouter à votre classe de personnage un moyen de garder en mémoire la *position*, et modifier votre script Javascript en conséquence.

Maintenant que vous pouvez parcourir le monde, vous vous rendez compte qu'il est bien vide. C'est là qu'entrent en scène les monstres (et les trésors, accessoirement). Avant d'ajouter toutes ces jolies petites choses à votre jeu, une petite modification s'impose.

Question 3.1 : Ajoutez à votre classe de personnage un *inventaire*. On pourra, par exemple, avoir un attribut arme, un autre armure, etc., ou alors utiliser un tableau.

Question 3.2 : Créez maintenant une classe pour les *armes* (*nom*, *dégâts*), pour les *armures* (*nom*, *valeur d'armure*), et pour les *monstres* (*nom*, *dégâts*).

Question 3.3 : Maintenant, ajoutez au déplacement une fonction qui, aléatoirement, déclenche un évènement (ou non), par exemple un combat, la découverte d'un trésor, la perte de points de vie à cause d'un piège, etc.

Su-per, vous avez maintenant un mini-RPG.

Le problème actuel de ce jeu est qu'il est très facile de tricher. En effet, rien ne vous empêche d'ouvrir la console du navigateur et de vous mettre de l'équipement surpuissant, de mettre vos caractéristiques à des valeurs gigantesques, et ainsi de suite. C'est là que va entrer en jeu le **Node.js**. En effet, vous allez déplacer vos fonctions et variables / objets " critiques " vers le serveur. Cela devrait empêcher les joueurs de manipuler les variables. De plus, pour l'instant, chaque fois que vous quittez la page ou que vous tuez le serveur (bande de monstres), le joueur est effacé. Il serait bien d'avoir une persistance des personnages d'une session à l'autre.

Question 4.1 : Créez maintenant un serveur en **Node.js**. La communication avec ce serveur se fera à l'aide de **requêtes GET**, lancées par le code (côté client) en Javascript. Pour commencer, le serveur disposera d'une URL /save, permettant de sauvegarder les informations du jeu que vous jugerez utile de conserver.

Ces données **devront être sauvegardées dans un/des fichier(s) au format JSON** sur le serveur.

ATTENTION : si votre serveur Node.js est lancé sur la même machine où votre code Javascript s'exécute, le serveur renverra probablement une erreur du type

« `No 'Access-Control-Allow-Origin' header is present on the requested resource` ».

Cette erreur est due au fait que la requête cherche à accéder à des données sur un autre serveur (plus de détail : https://en.wikipedia.org/wiki/Same-origin_policy).

Pour pallier ce problème, il faut que votre code Node.js contienne la ligne suivante avant la fermeture du flux de réponse :

```
response.setHeader("Access-Control-Allow-Origin","*")
```

Ainsi, le serveur spécifie que le partage de ressources de différentes origines est autorisée.

Question 4.2 : On veut, logiquement, implémenter le chargement des données (toujours par le serveur), à l'aide de l'URL /load. Cette fois, allons y plus progressivement (parce-que c'est quand même moins facile que la sauvegarde).

a) Récupérez les données du serveur dans votre script, puis vérifiez leur type. Faites en sorte que la donnée récupérée soit du même type que les tableaux ou variables qui contiennent vos valeurs.

b) Assignez les valeurs récupérées du serveur dans les variables Javascript.

Question 4.2 : Déplacez la création du personnage, des monstres, des objets (armes, armures, etc.) vers un serveur **Node.js**. L'appel à la fonction de **Javascript** fera donc elle-même un appel à une fonction **Node.js**.

Question 4.4 : Faites de même avec la fonction de génération d'évènement aléatoire. Affectez également un coefficient de difficulté en fonction de la direction choisie (exemple : *Nord* -> **x2** *points de vie* et *dégâts* des *monstres*, mais également **x2** *récompenses* / *Sud* -> **x0.5**, etc.). Ainsi, il y a utilité à avoir plusieurs directions disponible pour le déplacement.

Quand vous avez fini cette partie : **FAITES UNE COPIE DE VOTRE CODE**. La suite concerne des améliorations, mais il faut que la base (le code réalisé à la suite des 7 questions) soit disponible sans autre modification. Faites un dossier « base », par exemple, et un dossier « améliorations », qui seront tous les deux à la racine de votre archive.

Améliorations

Vous êtes arrivé ici ? Bravo.

Les améliorations ici sont des suggestions, qui ajouterons des points à votre note finale. L'ordre n'est pas important : certains sont faciles et rapides, d'autres ne sont que des améliorations esthétiques, à vous de voir ce que vous voulez faire. Comme indiqué précédemment, si vous arrivez à cette partie, **faites un autre projet séparé**.

Petite précision : n'ignorez pas une partie des questions du dessus en vous disant que vous rattraperez le manque de points avec les questions bonus, ça ne marche pas comme ça ☺

1) Pour l'instant, tout se passe en mode texte. Il serait sympa d'avoir une carte (en ASCII, par exemple), représentant le monde et le personnage se déplaçant.
2) Si vous avez, par exemple, 4 fonctions différentes pour le déplacement, essayer de n'en utiliser qu'une à la place : en un mot, factorisez votre code.
3) Développez l'aspect RPG de votre jeu, en ajoutant des monstres, des objets, des surprises, etc.
4) Votre code est fonctionnel, mais c'est probablement **MOCHE** à regarder. Rendez tout ça plus joli, en utilisant par exemple Bootstrap.js, qui fournit (entre autre) un CSS personnalisable.
5) Implémentez un chargement des objets à partir de fichiers externes (JSON, ou même un format personnalisé si ça vous chante). Ainsi, il sera facile de customiser votre jeu, même pour une personne extérieure (système de mods).
6) Faites en sorte de pouvoir sauvegarder (et charger) des sauvegarde de différents joueurs, par exemple en permettant au joueur de sauvegarder avec son pseudo.
7) Implémentez un leaderboard (en Node.js), avec les valeurs que vous désirez (par exemple le cumul des richesses, les stats de chaque joueur, etc.)
8) Vous êtes bien sûr libres de faire d'autres modifications ; cependant, n'oubliez pas de préciser ce que vous avez fait de différent / en plus, afin que je puisse regarder ça. Il serait dommage que ça ne vous rajoute pas de points. Si vous n'êtes pas sûr si ce que vous faite vous apportera effectivement des points, vous pouvez me demander avant de le faire.

1) L'application que vous avez développée aurait pu reposer sur un framework permettant de simplifier la création du squelette de votre application et éviter de recréer les mécanismes de base. Au regard de l'application que vous devez développer, le micro-framework « Express » est particulièrement. Il vous est donc possible de redévelopper la partie serveur à l'aide de ce framework.

2) Il existe un langage, le Typescript, qui est un superset de Javascript. Il permet, entre autre, de disposer d'un typage statique (optionnel), de créer des classes et interfaces, import de modules, etc. Vous pouvez utiliser ce langage afin de (re-) coder votre projet.