TP : m.O.G. (minimalist Online Game)

IMR3/LSI3 :RFM

Version du : 22 octobre 2014

Objectifs *A l'issue de cette séance de TP, vous serez capable de*
- *décrire le mécanisme de modules de nodeJS*
- *mettre en place un serveur http avec nodeJS*
- *mettre en place un serveur websocket avec nodeJS*
- *créer et lire le format JSON en javascript*
- *récupérer le flux d'une webcam*
- *récupérer le flux d'un gamepad*

Vous disposerez des outils pour prolonger le TP : WebCam et GamePad. Le délivrable individuel est :
- *au moins 3 archives de code source commenté (une archive par run).*
- *compte rendu de réalisation*

Date de Rendu : fin Décembre, pour la période de Noël.

1 But du TP

1.1 Cas d'usage

Dans cette séance de Travaux Pratiques, vous allez implémenter une application de Jeu partagé.
Un joueur A va pouvoir défier un concurent B à un jeu simple de "Stock Car" :

1. le joueur B se connecte à un serveur nodeJS pour se déclarer. A cette occasion, il ouvre une WebSocket avec le serveur dans laquelle il va pouvoir y envoyer les commandes issues de son gamepad. Il ouvre également un flux RTC dans lequel il enverra l'image de sa webcam.
2. Le gamePad permet de déplacer une voiture grâce au pavé gauche (ou joystic) dans un canevas (l'arène). Le retour de la webCam est là pour avoir un retour de son adversaire.
3. Le joueur A, voyant que B est présent, le défi et entre dans l'arène. Sur l'écran de chaque joueur, les deux voitures sont présentes, ainsi que le flux vidéo de l'adversaire.
4. Le premier joueur qui embouti l'autre a gagné. Cet événement déclenche une prise de photo chez les deux joueurs, pris sur le vif de la victoire et de la défaite. La photo vient remplacer la photo de profil des joueurs.

Vous allez développer cette application en plusieurs run :
- Premier run : Faire un jeu en local.
- Deuxième run : Porter le jeu online
- Troisième run : Ajouter la fonctionalité de partage de webCam.

1.2 Matériel fourni

- un "Kit Multimédia", c'est à dire un ensemble gamePad + webCam
- une application minimale de reconnaissance de gamePad (HTML5/JS).
- une application minimale de chat (WebSocket).
- une application minimale d'utilisation de la webCam (HTML5/JS).
- une application avancée de chat vidéo.

1.2.1 application minimale de reconnaissance de gamePad

Le fichier `testGamePad.html` a pour but de repérer la connexion d'un gamepad, de lister ses capacités (ses boutons) et enfin de recupérer des évenements sur ses boutons. Le code est commenté pour que vous puissiez le faire évoluer.

1.2.2 application minimale de chat

NodeJS est installé sur les machines de TP. NodeJs est en fait le moteur (javascript) V8 de Google. Il permet donc d'interpréter du Javascript en dehors d'un navigateur. Il est accompagné d'un ensemble de modules qui permettent de s'interfacer avec le système d'exploitation et en particulier avec le système de fichier et les ports de communication. En particulier, il est possible de rédiger rapidement un serveur http. L'intérêt de nodeJS est que son execution est parallèle (merci la programmation évenementielle) : il peut servir un très grand nombre de requêtes simultanément sans que le développeur ai à se soucier de concurrence d'accès.

Une application de chat utilisant nodeJS et les websockets vous est fournie. Elle a été crée par [http://ejohn.org]. Vous disposez donc de 3 fichiers :

`chat-server.js` C'est le serveur. C'est en fait un serveur http (cf `require('http')`) qui embarque un serveur websocket (cf `require('websocket')`). Remarquez que le serveur http est instancié en passant une call-back dont le rôle est de traiter les requêtes et les réponses (http). Ici, cette callback est vide (voir http ://nodejs.developpez.com/tutoriels/javascript/node-js-livre-debutant/). On instancie alors un serveur websocket avec pour paramètre le serveur http créé. En effet, le protocole ws (websocket) est une sur-couche de http. Cette encpsulation est donc assez logique. La logique métier est donc concentrée dans le traitement des requêtes au serveur websocket. A la première requête issue d'un client, nous choisissons d'ouvrir la socket (objet `connection`) et d'envoyer l'historique sous la forme d'un objet JSON. La requête suivante est alors interprétée par `connection` comme la déclaration du nom d'utilisateur du client et les suivants, comme des messages. Les messages doivent être renvoyés (sous le format JSON) sur la socket après qu'on leur ai ajouté la date de publication et le nom de l'auteur. Le message est aussi ajouté à l'historique, pour les clients qui viendraient à ce connecter (remarquez que l'historique est limité à 100 messages). On observe que le serveur émet sur la socket des messages dont le type est différencié par le format (des messages "color", "history" ou "message").

`chat-frontEnd.html` Le script client est très simple : il ne comporte qu'une zone de saisie (pour l'identifiant puis pour les messages) et une div d'affichage pour l'historique. Le code important est en fait dans le fichier de script attaché.

`chat-frontEnd.js` Ce fichier de script est interprété par le navigateur. Remarquez qu'il utilise la notation jQuery afin d'accéder facilement à la zone de texte et à la div. Il va chercher à établir une connexion websocket avec le serveur (attention à la correspondance des URLs et du port). Une fois la connexion établie, la zone de texte est rendue accessible. A la réception de message, un switch sur la nature de celui-ci est implémenté : s'agit-il d'un message "color", "history" ou "message" ? Enfin, une partie du code est dédiée à la capture de clavier : la touche entrée dans la zone de texte déclenche l'envoie d'un message vers le serveur.

modules node Ce répertoire contient les modules additionnels à nodeJS permettant l'établissement de websocket. Vous n'avez pas à y toucher.

Dans un premier temps, le serveur est lancé par la commande `node chat-server.js`. Les pages client (`chat-frontEnd.html` utilisant `chat-frontEnd.js`) peuvent être lancé de n'importe quelle machine ou bien servie (attention à la correspondance des URLs). L'utilisateur, lorsqu'il appelle la page client, établit une connexion avec le serveur qui l'accepte (sans test). Son premier message au serveur est la déclaration de son identifiant. En réponse, le serveur lui envoie l'historique des messages et une couleur qui l'identifie. Les messages suivants vers le serveur sont forcément des messages de chats (intégrés à l'historique et diffusés sur la web socket).

1.2.3 une application minimale d'utilisation de la webCam

Le fichier `testWebCam.html` a pour but d'ouvrir le flux de la webcam (après sollicitation de l'utilisateur) et de la renvoyer sur une balise vidéo. Le code est commenté pour sa compréhension.

1.2.4 application avancée de chat vidéo

L'archive fournie est tirée de `http://blog.felixhagspiel.de/`. et de

`https://github.com/felixhagspiel/webrtc-tutorial/tree/development`

Le serveur (nodeJS) ressemble beaucoup au premier...c'est normal, ils viennent de la même personne ! Allez sur le Blog, au post `create-your-own-videochat-application-with-html-and-javascript` pour un éclairage sur le code.

2 Votre travail

2.1 Premier Run : Faire un jeu en local

- Vous partez de `testGamePad.html`.
- L'idée est d'ajouter un canevas et un objet "voiture" (vous pouvez vous passer d'objet dans un premier temps et redessiner l'arène et la voiture dans une mêm fonction).
- Le pilotage de la voiture se fait dans la boucle de jeu.
- La voiture est lancée à fond les balons, vous ne pouvez que la diriger à droite, à gauche, la ralentir un peu (flêche bas) ou accélérer un peu (flêche haut) (du pavé gauche / joystic).
- Si vous rencontrez le bord du canevas, vous perdez.
- Faites tester votre jeu par un collègue.

2.2 Deuxième run : Porter le jeu online

- Vous partez de l'application minimale de chat. Vous allez déporter une partie du métier sur le serveur : il récupère les commandes et cacul la position de la voiture puis renvoie ce qui doit être affiché sur votre canvas.
- D'abord, créez de nouveaux messages permettant d'envoyer les signaux de votre gamePad. N'envoyez pas la position de votre voiture, pour éviter la triche.
- Ensuite, transposez la logique de jeu sur le serveur nodeJS : est ce que votre voiture rentre en collision avec le bord de l'arène ? Test : le comportement client doit être le même qu'à l'issus du premier run.
- Proposez un mécanisme permettant de stocker un profil utilisateur côté serveur : l'utilisateur rempli son profil à la connexion. Pas la peine de le persister.
- Puis, créez de nouveaux messages permettant de rejoindre une partie.
- Enfin, modifez la logique métier sur le serveur pour gérer les collisions de voitures entre elles.
- Faites tester votre jeu par deux collègues.

2.3 Troisième run : Ajouter la fonctionalité de partage de webCam

- Inspirez vous de l'application de chat vidéo pour ajouter le flux de webCam.
- Ajoutez la fonctionalité de partage de flux vidéo dans votre client de jeu. Vous pourrez stocker les informations nécessaire pour la PeerConnection dans le descriptif du `client` du serveur NodeJS.
- Lors de la collision, une photo est prise et vient écraser les photos des profils des utilisateurs.

2.4 Fonctionalités supplémentaires

Dans l'état actuel de votre jeu, le serveur de jeu fait aussi office de serveur de présence.

- Demandez gentiment à un LSI3 de vous passer le code (JEE) de son serveur de présence (service REST).
- Modifiez le pour stocker un profil utilisateur et une URL de PeerConnection (si joueur online).
- Modifiez votre serveur de Jeu pour qu'il accède au service de présence et débarassez le des fonctionalités de présence.
- Partagez avec des collègues le même serveur de présence.