WebSocket

Installation de *node*

Requis : *npm* (node packet manager) & *node.js*

sudo apt-get update

sudo apt-get install npm (installe aussi node)

sudo npm install websocket

Utilisation de *node.js*

Créer un fichier pour le serveur, par exemple server.js

Pour l'exécuter : *node server.js*

Exemple de serveur :

```
var http = require('http')
var WebSocketServer = require('websocket').server
var server = http.createServer(function(request, response) {})
server.listen(18000, function() { })

// create the server
wsServer = new WebSocketServer({
    httpServer: server
})


wsServer.on('request', function(request) {
    //First connection handling
    var connection = request.accept(null, request.origin)



    connection.on('message', function(message) {
        //messages handling
    })

    connection.on('close', function(connection) {
        //disconnection handling
    })
})
```

NB : Le serveur websocket a besoin d'un serveur http pour fonctionner. Dans notre cas, le serveur écoute sur le port 18000, url : *ws://IP_MACHINE :18000*.

Le serveur peut envoyer des messages grâce à l'objet *connection* : *connection.sendUTF(« Coucou »)* envoie coucou au client de la connexion.

Toute la gestion se fait dans l'événement *message,* c'est grâce à cet événement qu'il faut implémenter le protocole et la logique de l'application.

Le serveur sert essentiellement d'intermédiaire entre les clients. Un client fait une mise à jour, il l'envoie au serveur. A son tour, le serveur la fait suivre aux autres clients.

Exemple de client :

```
//Request connection
var connection = new WebSocket('ws://127.0.0.1:18000')

connection.onopen = function () {
    //Connection accepted

}

connection.onmessage = function (message) {
    //message received
}
```

Logique générale de l'application

SERVEUR

Evénement : Un client se connecte

- Le serveur reçoit la connexion, il la conserve dans sa liste de connexions actives
- Il informe tous les clients déjà connectés de la présence d'un nouveau client
- Il informe le nouveau client de la présence de tous les clients connectés

Evénement : Un client se déconnecte

- Le serveur supprime le client de sa liste de connexions actives
- Il informe tous les clients connectés de la déconnexion d'un client donné

Evénement : Un client envoie un message ou une position

- Le serveur informe tous les clients du nouveau message ou de la modification de la position

CLIENT

Evénement : Le serveur confirme la connexion au client

- Le client crée un nouveau joueur

Evénement : Le serveur informe le client de la présence d'un nouveau joueur

- Le client enregistre le joueur dans sa liste des joueurs actifs et l'inclut dans la logique d'affichage et de mise à jour

Evénement : Le serveur informe le client d'un nouveau message ou d'un changement de position d'un joueur

- Le client déplace le joueur concerné au bon endroit ou affiche le message

Evénement : Le serveur informe le client de la déconnexion d'un joueur

- Le client enlève le joueur de la liste des joueurs actifs et l'exclut de la logique d'affichage et de mise à jour.

Travail pratique

Compléter le code source qui est disponible sur le site cyberlearn du cours.

Consignes

L'ossature du code source client et serveur sont disponibles. Ils contiennent des zones à compléter indiquées par :

```
////////////////////////////////////////
CODE TO INSERT HERE
////////////////////////////////////////
```

Les fichiers à compléter sont : « client.html » et « server.js ». Les deux autres fichiers ne sont pas à modifier (« client.js » et « GameManager.js »).

Pour réaliser ce travail, il est fortement conseillé de suivre la *logique générale de l'application*. Il est donc nécessaire de comprendre le code source disponible avant de le compléter