PHP

Bertrand Estellon

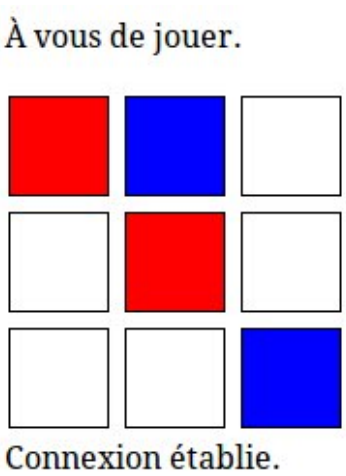
Aix-Marseille Université

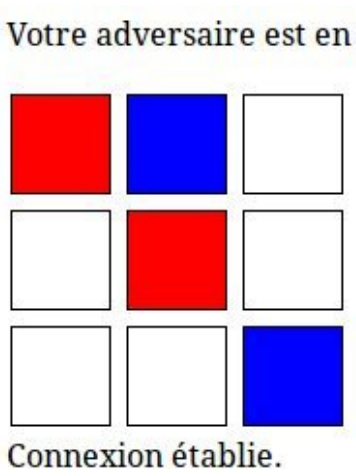
13 mars 2012

Problématique

Problématique : Nous souhaitons réaliser un jeu de morpion en réseau.

- Les clients se connectent au jeu ;
- Les clients jouent chacun leur tour ;
- Les coups d'un joueur sont répercutés sur la grille de l'autre joueur ;
- Le serveur organise la partie (débute la partie, décide le gagnant...) ;





Problématique

Problème : En PHP, chaque requête du client est traitée indépendamment.

Il n'y a donc pas de :

- Processus persistant et de donnée en mémoire persistante
  ⇒ Difficulté pour conserver l'état courant de la partie
  (*i.e.* orchestrer plusieurs clients)
- Connexion persistante (le client demande et le serveur répond)
  ⇒ Difficulté pour envoyer des événements aux clients.

Solution :

- Côté client : WebSocket de HTML 5
- Côté serveur : Java et Jetty (serveur HTTP et moteur de servlet libre)

Autres solutions :

- AJAX et l'approche Comet
- Socket.IO (développement du client et du serveur en JavaScript)
- ...

WebSocket

Aix*Marseille université

Objectif (Wikipedia) : Obtenir un canal de communication bidirectionnel et full-duplex sur un socket TCP pour les navigateurs et les serveurs web.

Demande de connexion du client et "handshake" :

```
GET /ws HTTP/1.1
Host: pmx
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Version: 6
Sec-WebSocket-Origin: http://pmx
Sec-WebSocket-Extensions: deflate-stream
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
```

Réponse du serveur :

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

Client – WebSocket et JavaScript

Aix*Marseille université

Demande d'ouverture de la connexion avec le serveur :

```
if ('WebSocket' in window)
        ws = new WebSocket('ws://toto.com:8081');
else if ('MozWebSocket' in window)
        ws = new MozWebSocket('ws://toto.com:8081');
else alert("Pas de WebSocket");
```

Mise en place des "callbacks" :

```
ws.onopen = onOpen;
ws.onclose = onClose;
ws.onerror = onError;
ws.onmessage = onMessage;
```

Exemples de "callbacks" :

```
function onOpen(event) { alert("Connexion etablie."); }
function onClose(event) { alert("Connexion perdue."); }
function onError(event) { alert("Erreur"); }
function onMessage(event) { alert(event.data); }
```

Client – WebSocket et JavaScript

Aix*Marseille université

Envoie des messages vers le serveur :

```
for (var i = 0; i < 3; i++) {
  for (var j = 0; j < 3; j++) {
    $('#grid').append(
        '<div id="c'+i+""+j+'" class="case"></div>');
  }
  $('#grid').append("<br/>");
}

$('.case').click(function() {
  var id = $(this).attr('id');
  var r = parseInt(id.charAt(1));
  var c = parseInt(id.charAt(2));
  ws.send("P#"+r+"#"+c);
});
```

Serveur – Jetty - Simple serveur HTTP

Aix*Marseille université

Création d'un serveur HTTP avec Jetty :

```
public static void main(String[] args) throws Exception {
    Server httpServer = new Server(8080);
    ResourceHandler resourceHandler = new ResourceHandler();
    resourceHandler.setDirectoriesListed(true);
    resourceHandler.setWelcomeFiles(new String[]
                                        { "index.html" });
    resourceHandler.setResourceBase("/home/toto/client");
    httpServer.setHandler(resourceHandler);
    httpServer.start();
    httpServer.join();
}
```

Serveur – Jetty - Serveur WebSocket

Aix*Marseille université

Création d'un serveur WebSocket avec Jetty :

```
public static void main(String[] args) throws Exception {
   Server wsServer = new Server(8081);
   wsServer.setHandler(new MyServer());
   wsServer.start();
   wsServer.join();
}
```

La classe qui reçoit les notifications de connexion :

```
public class MyServer extends WebSocketHandler {

   public WebSocket doWebSocketConnect(
                              HttpServletRequest request,
                              String protocol) {
      Client client = new MyClient(this);
      return client;
   }
}
```

Serveur – Jetty - Serveur WebSocket

Aix*Marseille université

Gestion d'une connexion entre un client et le serveur :

```
public class MyClient implements WebSocket.OnTextMessage {
  private Connection connection;

  public void onOpen(Connection connection) {
    this.connection = connection;
    //TODO
  }

  public void onMessage(String data) { //TODO }

  public void onClose(int code, String msg) { // TODO }

  public void send(String data) {
    try { connection.sendMessage(data); }
    catch (IOException e) { connection.close(); }
  }
}
```

Jeu de morpion – Protocole

Aix*Marseille université

Les messages du serveur :

- W : Attendre le début de la partie.
- P : Vous devez jouer un coup.
- O : Votre adversaire est en train de jouer.
- V : Vous avez gagné.
- L : Vous avez perdu.
- D#r#c#j : Le joueur j à jouer la case (r, c).

Les messages du client :

- P#r#c : Je joue la case (r, c).

Jeu de morpion – Client

Aix*Marseille université

La page HTML du client :

```
<html>
  <head>
    <script type="text/javascript" src="jquery.js">
    </script>
    <script type="text/javascript" src="client.js">
    </script>
     ...
  </head>
  <body onload="init()">
    <div id="gameMessage" class="message"></div><br/>
    <div id="grid"></div>
    <div id="connectionMessage" class="message"></div>
  </body>
</html>
```

Jeu de morpion – Client

Aix*Marseille université

La fonction *init()* :

```
function init() {
  if ('WebSocket' in window)
    ws = new WebSocket('ws://localhost:8081');
  else if ('MozWebSocket' in window)
    ws = new MozWebSocket('ws://localhost:8081');

  ws.onopen = onOpen;
  ws.onmessage = onMessage;
  ws.onclose = onClose;

  // Code pour creer la grille.

  // Code pour ecouter les clics de souris.
}
```

Jeu de morpion – Client

Aix*Marseille université

Code pour créer la grille :

```
var i,j;
for (i = 0; i < 3; i++) {
  for (j = 0; j < 3; j++)
    $('#grid').append(
      '<div id="c'+i+""+j+'" class="case"></div>');
  $('#grid').append("<br/>");
}
```

Code pour écouter les clics de souris :

```
$('.case').click(function() {
  var id = $(this).attr('id');
  var r = parseInt(id.charAt(1));
  var c = parseInt(id.charAt(2));
   ws.send("P#"+r+"#"+c);
});
```

Jeu de morpion – Client

Traitement des messages du serveur :

```
function onMessage(event) {
  switch (event.data.charAt(0)) {
    case 'W' : setGameMessage("Attendre."); break;
    case 'P' : setGameMessage("Jouer."); break;
    case 'O' : setGameMessage("Votre adv. joue."); break;
    case 'V' : setGameMessage("Victoire."); break;
    case 'L' : setGameMessage("Perdu."); break;
    case 'D' : var r = parseInt(event.data.charAt(2));
               var c = parseInt(event.data.charAt(4));
               var p = parseInt(event.data.charAt(6));
               if (p==1) $("#c"+r+""+c).addClass("red");
               else $("#c"+r+""+c).addClass("blue");
               break;
  }
}

function setGameMessage(m) { $('#gameMessage').html(m); }
```

Jeu de morpion – Client

Traitement des connexions et déconnexions :

```
function onOpen() {
  $('#connectionMessage').html("Connexion etablie.");
}

function onClose() {
  $('#connectionMessage').html("Connexion perdue.");
}
```

Jeu de morpion – Serveur

```
public class Main {
  public static void main(String[] args) throws Exception {
    Server wsServer = new Server(8081);
    wsServer.setHandler(new MorpionServer());
    wsServer.start();

    Server htmlServer = new Server(8080);
    ResourceHandler rHandler = new ResourceHandler();
    rHandler.setDirectoriesListed(true);
    rHandler.setWelcomeFiles(new String[] {"index.html"});
    rHandler.setResourceBase("client");
    htmlServer.setHandler(rHandler);
    htmlServer.start();

    wsServer.join();
    htmlServer.join();
  }
}
```

Jeu de morpion – Serveur

```
public class MorpionServer extends WebSocketHandler {

  private Game game = new Game();

  public WebSocket doWebSocketConnect(
      HttpServletRequest request,
      String protocol) {
    Client client = new Client(this);
    return client;
  }

  public void addPlayer(Client client) {
    game.addPlayer(client);
    if (game.isComplete()) {
      game.start();
      game = new Game();
    }
  }
}
```

Jeu de morpion – Serveur

```
public class Client implements WebSocket.OnTextMessage {
  private Connection connection;
  private MorpionServer server;
  private Game game;
  private int position;

  public Client(MorpionServer server) {
    this.server = server;
  }

  public void onOpen(Connection connection) {
    this.connection = connection;
    server.addPlayer(this);
  }

  public void onMessage(String data) {
    game.onMessage(position, data);
  }
```

Jeu de morpion – Serveur

(Suite de la classe *Client*)

```
    public void onClose(int closeCode, String message) {
        game.finish();
    }

    public void setGame(Game game, int position) {
        this.game = game;
        this.position = position;
    }

    public void send(String data) {
        try { connection.sendMessage(data); }
        catch (IOException e) { connection.close(); }
    }

    public void close() { connection.close(); }
}
```

Jeu de morpion – Serveur

```
public class Game {

  private Client[] players;
  private int curPlayer;
  private int grid[][];

  public Game() {
    players = new Client[2];
    grid = new int[3][3];
  }

  public void addPlayer(Client client) {
    if (players[0] == null) {
      players[0] = client; client.setGame(this, 1);
      client.send("W");
    } else { players[1] = client;
             client.setGame(this, 2);
    }
  }
```

Jeu de morpion – Serveur

```
public boolean isComplete() {
  return (players[1] != null);
}

public void start() {
  curPlayer = 1;
  players[curPlayer - 1].send("P");
  players[2 - curPlayer].send("O");
}

public void finish() {
  for (int i = 0; i < 2; i++)
    if (players[i] != null) {
      players[i].close(); players[i] = null;
    }
}

private isWinner(int position) { ... }
```

Jeu de morpion – Serveur

```
    public void onMessage(int position, String d) {
        if (position != curPlayer) return;
        if (!d.matches("^P#[0-9]#[0-9]$")) return;
        int c = d.charAt(2) - '0'; int r = d.charAt(4) - '0';

        if (grid[c][r] != 0) return; grid[c][r] = position;

        players[0].send("D#" + c + "#" + r + "#" + curPlayer);
        players[1].send("D#" + c + "#" + r + "#" + curPlayer);

        if (isWinner(position)) {
            players[position - 1].send("V");
            players[2 - position].send("L"); finish();
        } else {
            curPlayer = 3 - curPlayer;
            players[curPlayer - 1].send("P");
            players[2 - curPlayer].send("O");
        }
    }
}
```