École Centrale Lyon

UE D-3
Rapport

Compte-rendu Application WEB

Élèves :
Enzo Gennari
Martin Gouron

Enseignants :
René Chalon Daniel
Muller

29 mars 2022

Table des matières

1 Introduction

Ce rapport présente notre travail réalisé pour l'UE Applications WEB. Nous avons choisi de faire un jeu gérant deux joueurs en serveur, utilisant l'API Socket et Canvas. Malheureusement, il nous a été impossible de finir le projet demandé dans le temps imparti. Notre travail se divise donc en deux parties.

Premièrement, nous avons créé un jeu de morpion qui, dans la même page, permet à deux joueur de s'affronter à tour de rôle. Nous avons ensuite utilisé Socket pour créer un serveur gérant un chat. Cette partie s'est révélée plus compliquée qu'attendu. Ainsi, nous n'avons pas pu adapter le morpion, pour intégrer le multijoueur.

2 Morpion

2.1 Considérations générales

Nous avons choisi de gérer le jeu par une matrice 3*3, donc chaque case vaut 0 si personne n'a joué, 1 si le joueur 1 y a joué et 2 pour le joueur 2. Une variable player enregistre quel joueur va jouer. Nous avons utilisé Canvas pour afficher les différents éléments graphiques du jeu. Le jeu est joué entièrement à la souris, donc les clics sont captés par une fonction *addEventListener*

2.2 Gestion des coups

Lors d'un clic de souris, on appelle si besoin la fonction *addPlayingPiece*, on prends les coordonnées, on vérifie qu'elles correspondent à une des cases et qu'elle est vide, et ensuite on affiche la pièce correspondante au joueur. On vérifie également si un des joueurs a gagné :

```
/* Ajout d'une pièce */
function addPlayingPiece (mouse) {
  var xCordinate;
  var yCordinate;

  for (var x = 0;x < 3;x++) {
    for (var y = 0;y < 3;y++) {
      xCordinate = x * sectionSize;
      yCordinate = y * sectionSize;

      if (
          mouse.x >= xCordinate && mouse.x <= xCordinate + sectionSize &&
          mouse.y >= yCordinate && mouse.y <= yCordinate + sectionSize
        ) {

        if (board[x][y] == 0) {
          clearPlayingArea(xCordinate, yCordinate);
          if (player === 1) {
            drawX(xCordinate, yCordinate);
            board[x][y] = 1;
          } else {
            drawO(xCordinate, yCordinate);
            board[x][y] = 2;
          }
          testWinner(player)
          player = 3 - player;
        }
```

FIGURE 1 – La fonction *addPlayingPiece*

2.3 Détection de vainqueur/fin de partie

La fonction *testWinner* vérifie si le joueur qui vient de jouer a gagné. On teste toute les combinaisons possibles (vertical, horizontal, diagonal). On vérifie également une égalité potentielle

```
/* Détermine si le winner gagne, teste l'égalité*/
function testWinner (winner) {
  var isWinning = false;
  for (var x = 0;x < 3;x++) {
    if (board[x][0] == winner && board[x][1] == winner && board[x][2] == winner) {
      isWinning = true;
    }
    if (board[0][x] == winner && board[1][x] == winner && board[2][x] == winner) {
      isWinning = true;
    }
  }
    if (board[0][0] == winner && board[1][1] == winner && board[2][2] == winner) {
      isWinning = true;
    }
    if (board[2][0] == winner && board[1][1] == winner && board[0][2] == winner) {
      isWinning = true;
    }

    if (isWinning) {
      if (winner == 1) {
        document.getElementById( elementId: "div_message").innerHTML = "Vainqueur : X !";
      } else {
        document.getElementById( elementId: "div_message").innerHTML = "Vainqueur : O !";
      }
```

FIGURE 2 – Début de la fonction *testWinner*

2.4 Affichage de la partie

On utilise pour cela un canvas, qui permet d'afficher les différentes formes voulues. On illustre ici avec le tracé du cercle :

```
/* Affiche un O */
function drawO (xCordinate, yCordinate) {
  var halfSectionSize = (0.5 * sectionSize);
  var centerX = xCordinate + halfSectionSize;
  var centerY = yCordinate + halfSectionSize;
  var radius = (sectionSize - 100) / 2;
  var startAngle = 0;
  var endAngle = 2 * Math.PI;

  context.lineWidth = 10;s
  context.strokeStyle = "#01bBC2";
  context.beginPath();
  context.arc(centerX, centerY, radius, startAngle, endAngle);
  context.stroke();
}
```

FIGURE 3 – La fonction *drawO*, qui permet de tracer un cercle

2.5 Affichage des scores

On utilise un fichier CSS pour créer des emplacement pour gérer les scores. Ceux-ci sont mis à jour avec une commande du type *document.getElementById("div_score1").innerHTML = score[1];*

```css
/* Score joueur 1 */
.score1{
  font-weight: bold;
  position : absolute;
  top: 20%;
  left:20%;
}
```

FIGURE 4 – Classe affichant le score du joueur 1

3 Chat

3.1 Considérations générales

On identifie chacun par un surnom (ie nom d'utilisateur). Grâce à l'utilisation d'un salon, il est nécessaire d'avoir choisi un surnom pour voir les autres messages envoyés et pour envoyer soit-même des messages

3.2 Gestion des surnoms

Pour pouvoir utiliser le surnom comme identifiant, il faut s'assurer de son unicité. La liste usernames stocke tout les noms des utilisateurs connectés :

```
io.sockets.on( type: 'connection', listener: function(socket) {

    //Permet d'afficher le nombre de personnes connectées sur la page d'accueil à l'utilisateur qui se connecte
    socket.emit('updateUsersCount', Object.keys(usernames).length);

    //Lorsque l'utilisateur rentre un pseudo on va vérifier que le pseudo n'est pas pris
    socket.on( type: 'adduser', listener: function(newusername) {

        if (usernames.hasOwnProperty(newusername)) {
            socket.emit('usernametaken');
            return;
        }

        socket.username = newusername;
        usernames[newusername] = newusername;
```

FIGURE 5 – Vérification des noms d'utilisateurs

Il est également important de libérer les noms d'utilisateurs en cas de déconnexion, ce que l'on fait :

```
// Gestion des déconnexions
socket.on( type: 'disconnect', listener: function() {
    delete usernames[socket.username];
    socket.broadcast.to(default_room).emit('updatechat', '', '<span style="color:grey;">' + socket.username + ' s\'est déconnec
    socket.leave(socket.room);
```

FIGURE 6 – Gestion des déconnexions

3.3 Gestion des messages

Quand on envoie le message, on ajoute le nom d'utilisateur afin de l'identifier. Il y a également un message pour chaque connexion/déconnexion

```
    // Gestion des messages
    socket.on( type: 'sendchat', listener: function(data) {
        io.sockets.in(socket.room).emit('updatechat', socket.username + ' : ', data);
    });
```

FIGURE 7 – Envoi des messages

```
//pour maj le chat
socket.on( type: 'updatechat', listener: function(username, data) {
    $('#conversation').append('<b>' + username + '</b>' + data + '<br>');
});
```

FIGURE 8 – Affichage des messages

3.4 Affichage

On paramètre les différents élements pour qu'ils s'affichent correctement. On a par exemple l'affichage du chat avec le paramètre "z-index : -1", qui permet de l'afficher derrière le panneau d'accueil, et "width : 98.5% ;" afin de laisser la place à la barre de scroll

```
#bg_scroller {
  color:black;
  position: fixed;
  left: 0;
  top: 0;
  padding:10px;
  bottom: 50px;
  width: 98.5%;
  z-index: -1;
  background-color:#FBFBFB;
  overflow-y: scroll;
  -webkit-box-shadow: 0px 0px 65px -24px rgba(0,0,0,0.75);
  -moz-box-shadow: 0px 0px 65px -24px rgba(0,0,0,0.75);
  box-shadow: 0px 0px 65px -24px rgba(0,0,0,0.75);
}
```

FIGURE 9 – Paramètres de l'affichage des messages

4 Axes d'amélioration

Au-delà de l'évidente fusion des deux programmes pour respecter les consignes, il y a plusieurs axes d'amélioration possibles pour notre travail

4.1 Morpion

Si l'aspect technique du jeu est satisfaisant, il est probablement possible d'améliorer l'aspect esthétique. Cependant, cela sera fait plus simplement avec l'utilisation d'images plutôt que par des dessins sur canvas. Il est également possible de créer des jeux plus avancés, comme un puissance 4, avec une logique similaire au morpion

4.2 Chat

On peut ajouter la gestion de salons, qui est déjà entamée (mais avec un seul salon). Cela pourrait par exemple permettre les messages privés ou tout simplement d'avoir plusieurs groupes de discussion. L'aspect esthétique est également améliorable