CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Tema d'anno

in

Metodi di Controlli nei Sistemi di Elaborazione e di Telecomunicazione

PongRTC

Studenti:

Mariani Stefano

Porcelli Gianluca

Anno accademico 2017-2018

INDICE

1. Introduzione

Il lavoro descritto in questa relazione consiste nell'applicazione delle politiche imposte da una connessione peer to peer ad un ambiente di gioco precedentemente realizzato per un singolo giocatore.

La creazione di un *data channel* è esattamente ciò che hanno bisogno i due giocatori per condividere la propria esperienza di gioco non rinunciando alla possibilità di scambiare messaggi (chat) con lo sfidante.

La connessione fra i due peer che desiderano comunicare è guidata attraverso uno scambio di messaggi (trasparenti all'utente) contenenti i parametri di comunicazione dei due giocatori; La connessione effettiva verrà instaurata soltanto se al momento del click sul tasto "inizia a comunicare" (da parte del peer che ha avviato la comunicazione) non si sono verificati errori di connessione. Soltanto nel caso di avvenuta connessione sarà possibile condividere l'ambiente di gioco e la relativa chat, in caso contrario, entrambi i peer saranno isolati e non in comunicazione.

2.WebRTC

WebRTC significa *Real-Time Communication* via web, ovvero la possibilità di effettuare chat, chiamate e videochiamate tramite internet; è un progetto open source nato nel 2011 con l'obiettivo di rivoluzionare la comunicazione, sfruttando il potere del web.

La comunicazione peer to peer è il punto forte di questa tecnologia che permette la comunicazione (*chat* o *video-chat*) fra due utenti o il trasferimento dati (foto, video...) gratuitamente e semplicemente.

Cosa lo differenzia dagli altri sistemi?

Non è necessaria alcuna installazione di file aggiuntivi a quelli già presenti nei *browser web* (coloro i quali già utilizzano questa nuova filosofia di comunicazione). Tutto ciò che serve è capirne il funzionamento e provarne le potenzialità.

La connessione fra i due contendenti avviene mediante lo scambio di 3 messaggi in particolare:

-SDP offer
-SDP answer
-Set remote description

SDP Offer: Inviata da chi vuole iniziare la comunicazione; L'offerta SDP include informazioni su qualsiasi flusso di dati già operativo, sul *codec*, sulle opzioni supportate dal browser e su eventuali candidati (alla comunicazione) già raccolti dall' ICE server.

.**SDP Answer**: La risposta ad un'offerta di comunicazione contiene la stessa tipologia di informazioni inviate dall'altro peer remoto (costruite a partire dal messaggio ricevuto) ma che descrivono il *peer* che ha ricevuto l'invito di comunicazione. Per continuare correttamente la negoziazione dei parametri, questa risposta, dovrà necessariamente essere recapitata al peer che ha iniziato la comunicazione

Set remote description: Completa la procedura di connessione abortendo connessioni già instaurate qualora ce ne fossero. Questa

procedura è avviata dal chiamante che dopo aver ottenuto le informazioni richieste al peer remoto può connettersi ad esso in maniera “diretta” (bypassando il NAT *traversal*) grazie all’aiuto di un ICE server.

Questo scambio di messaggio è trasparente all’utente nel nostro applicativo ma dovrà essere gestito manualmente dai due peer sfruttando le procedure nascoste dietro il click di 3 bottoni (uno per ogni messaggio da inviare).

3. Linguaggi e librerie utilizzate

Principalmente, il linguaggio utilizzato, sia per instaurare la connessione tra i due *peer*, sia per la logica del gioco è **JavaScript**. Infatti, grazie alla programmazione guidata dagli eventi, una volta instaurata la connessione, essi potranno scambiare informazioni riguardanti le proprie condizioni di gioco che saranno utilizzate per la sincronizzazione dei *peer*. Questo è solo un esempio delle potenzialità di questo linguaggio. Nello specifico, per semplificare la manipolazione e la gestione degli eventi, abbiamo utilizzato la libreria **jQuery**. Infatti, una volta create le animazioni delle due palette e della pallina, circoscritte all'interno di un campo da gioco, opportunamente dimensionato, è stato semplice poterle richiamare attraverso jQuery, sia da parte del *sender* che del *receiver.*

Come già detto, per l'instaurazione della connessione è necessario l'inoltro di una *SDP Offer* e di una *SDP Answer*. Per la gestione automatica delle richieste ci siamo serviti di una tecnologia basata su *WebSocket*, in particolare della libreria **Socket.io**. Grazie a questa libreria ed ai comandi da essa forniti, sarà possibile condividere fra tutti i peer connessi alla pagina (ma non ancora in comunicazione P2P) una *text-box* che servirà alla gestione delle richieste di connessione. Per il suo funzionamento, *Socket.io* sfrutta **Node.js**, che è un *runtime* JavaScript, in grado di mantenere un servizio in *background* su di una macchina server, e quindi di ricevere delle richieste ed offrire delle risorse.

Dunque, nel nostro caso, è utile che questo processo sia in esecuzione su di un server, almeno nella prima fase di *signaling*.

Per l'integrazione del servizio di *WebSocket* ci posizioniamo nella cartella dove è salvato il file di *front-end* index.html, che contiene sia il codice **HTML** della pagina, gli eventuali script *e file di stile (*in **CSS3**). Se siamo in ambiente Linux, provvediamo ad installare Node.js attraverso il comando:

$ npm install node

Successivamente bisogna generare le cartella node_module*s* attraverso il comando:

$ npm install - -save express

Infine per il servizio Socket.io diamo il comando:

$ npm install - -save socket.io

A questo punto, abbiamo creato un file chiamato index.js; esso rappresenta il nostro server in ascolto delle richieste client (la porta scelta è stata la 3000).
Per l' esecuzione in *background* del processo diamo il comando:

$ node index.js

Una volta instaurata la connessione tra due peer, il processo in *background* può anche essere terminato in quanto i due giocatori risultano connessi direttamente, ciò non influirà sul gioco in esecuzione.

4. PongRTC

Per l'implementazione del gioco siamo partiti da una versione single-player, progettata usando i linguaggi HTML5, CSS3 e la libreria JQuery.

Prima di tutto abbiamo incluso la libreria jQuery nel file index.html. Successivamente abbiamo creato con HTML degli oggetti utili al gioco, come le palette, la pallina ed il *playground.* Poi abbiamo applicato lo stile a questi oggetti, facendo attenzione che la posizione assunta dagli oggetti in movimento sia una posizione assoluta, mentre il playground (statico) avrà una posizione relativa. Infatti, la logica del gioco è quella di variare opportunamente (attraverso jQuery) le proprietà CSS *left* e *top* affinché gli oggetti possano cambiare posizione attraverso degli input dell'utente. La funzione che contiene lo *script* utile al movimento delle palette è chiamata *movePaddles()*; invece quella utile al movimento della pallina, ed alla localizzazione di quest'ultima, al fine di assegnare un punteggio valido per ognuno dei due giocatori, è chiamata *moveBall()*. Queste due funzioni sono contenute all'interno di una funzione padre *gameloop(); essa* verrà richiamata a connessione avvenuta e poi successivamente ogni 40 millisecondi [usando questo comando: *setInterval(gameloop,40)]*. Nel caso si volesse sospendere questo ciclo di richiamo della funzione, attraverso l'input di un utente, si può usare il comando *clearInterval(),* il quale, permetterà di mettere in pausa il gioco stesso.

Una volta instaurata la connessione tra due *peer* ed avviata una sessione di gioco, i due client sfruttano il *DataChannel* per scambiarsi informazioni inerenti le posizioni della pallina e delle palette. Il canale di comunicazione esistente fra i due giocatori sarà utile a due scopi: scambiare messaggi con l'altro peer e mantenere il sincronismo delle schermate di gioco (inviando al peer remoto, tutti i tasti premuti da ogni giocatore).

Durante un test della nostra applicazione, abbiamo osservato che in seguito alla perdita del focus della pagina il loop di gioco viene sospeso e quindi non ci sarà più sincronismo fra i peer. A risoluzione di questa problematica è stato integrato un tasto che permette la risincronizzazione del gioco. In particolare, ad ogni richiesta di sincronizzazione da parte del *peer A*, il *peer B* riceve le informazioni di

gioco e le applica alla propria pagina locale. Il sincronismo delle palette non è soggetto a malfunzionamenti, perché esso non è incluso in un loop ma viene gestito ad ogni variazione (indetta da uno dei due peer) sfruttando le potenzialità del data channel; il canale dati ci permette di mettere al corrente il peer remoto di tale variazione nel rettangolo di gioco gestendo un semplice evento.

5. Conclusioni

In questo lavoro progettuale abbiamo voluto testare le potenzialità della tecnologia WebRTC in un ambiente abbastanza inedito. Lo sviluppo di un gioco non è mai semplice, ma ciò che lo rende ancora più complicato è l'idea di far giocare due utenti in remoto.

Per la risoluzione di questa problematica è stato necessario inizialmente permettere una connessione diretta fra i due giocatori; successivamente, a connessione stabilita, grazie alle potenzialità del data channel non è stato poi difficile gestire la coesistenza di due giocatori sulla stessa pagina Web. Il punto forte di un qualsiasi applicativo pensato sfruttando WebRTC è la possibilità di condividere esperienze di gioco (o altro) non preoccupandomi della tipologia di dispositivo con il quale sono connesso né tanto meno le componenti aggiuntive che ho già installato (mi basterà avviare un qualsiasi browser web).

APPENDICE

Funzione *moveBall():*

```
function moveBall() {
 // faccio riferimento a delle variabili utili
 var playgroundHeight = parseInt($("#playground").height());
 var playgroundWidth = parseInt($("#playground").width());
 var ball = pingpong.ball;

 // identifico i bordi del campo
 // identifico il confine inferiore
 if (ball.y + ball.speed*ball.directionY > playgroundHeight)
 {
 ball.directionY = -1;
 }
 // identifico il confine superiore
 if (ball.y + ball.speed*ball.directionY < 0)
 {
 ball.directionY = 1;
 }
 // identifico il confine destro
 if (ball.x + ball.speed*ball.directionX > playgroundWidth)
 {

 // il giocatore B ha perso
 pingpong.scoreA++;
$("#scoreA").html(pingpong.scoreA);
 // resetta la posizione della pallina
 ball.x = 250;
 ball.y = 100;
 $("#ball").css({
 "left": ball.x,
 "top" : ball.y
 });
 ball.directionX = -1;
 }
 // identifica confine sinistro
 if (ball.x + ball.speed*ball.directionX < 0)
 {
                // il giocatore A ha perso
                pingpong.scoreB++;
$("#scoreB").html(pingpong.scoreB);
 // resetta la posizione della pallina
 ball.x = 150;
 ball.y = 100;
 $("#ball").css({
 "left": ball.x,
 "top" : ball.y
 });
 ball.directionX = 1;
 }
 ball.x += ball.speed * ball.directionX;
 ball.y += ball.speed * ball.directionY;

//identifica posizione della paletta A e decidi direzione della pallina

 var paddleAX = parseInt($("#paddleA").css("left"))+parseInt($("#paddleA").css("width"));
var paddleAYBottom = parseInt($("#paddleA").css("top"))+parseInt($("#paddleA").css("height"));
var paddleAYTop = parseInt($("#paddleA").css("top"));
if (ball.x + ball.speed*ball.directionX < paddleAX)
{
 if (ball.y + ball.speed*ball.directionY <= paddleAYBottom &&
 ball.y + ball.speed*ball.directionY >= paddleAYTop)
 {
```

```
 ball.directionX = 1;
 }
}
// identifica posizione della paletta B e decide la direzione della pallina
var paddleBX = parseInt($("#paddleB").css("left"));
var paddleBYBottom = parseInt($("#paddleB").css("top"))+parseInt($("#paddleB").css("height"));
var paddleBYTop = parseInt($("#paddleB").css("top"));
if (ball.x + ball.speed*ball.directionX >= paddleBX)
{
 if (ball.y + ball.speed*ball.directionY <= paddleBYBottom && ball.y + ball.speed*ball.directionY >= paddleBYTop)
 {
 ball.directionX = -1;
 }
}
  // muovi la pallina con una certa direzione e velocità
 $("#ball").css({
 "left" : ball.x,
 "top" : ball.y
 });
}
```

Funzione *movePaddles():*

```
function movePaddles() {
 if (pingpong.pressedKeys[KEY.O]) { // o
 // muovi su la paletta B
 var top = parseInt($("#paddleB").css("top"));
 $("#paddleB").css("top",top-5);
 }
 if (pingpong.pressedKeys[KEY.L]) { // l
 // muovi giù la paletta B
 var top = parseInt($("#paddleB").css("top"));
 $("#paddleB").css("top",top+5);
 }

 if (pingpong.pressedKeys[KEY.W]) { // w
 // muovi su la paletta A
 var top = parseInt($("#paddleA").css("top"));
 $("#paddleA").css("top",top-5);
 }
 if (pingpong.pressedKeys[KEY.S]) { // s
 // muovi giù la paletta A
 var top = parseInt($("#paddleA").css("top"));
 $("#paddleA").css("top",top+5);
 }
}
```

Funzione gameloop():

```
function gameloop() {

 moveBall();

 movePaddles();

}
```

Funzione di reset del gioco:

1) Invio dei parametri di gioco:

```
function update(){

dataChannel.send("Mi risincronizzo all'altro peer");
```

```
var ball=pingpong.ball;

dataChannel.send(ball.x+1000);

dataChannel.send(ball.y+2000);

dataChannel.send(ball.directionX+3100);//sommo alla direzione X 3100 in quanto essa potrà assumere il valore -1

dataChannel.send(ball.directionY+4100);//sommo alla direzione Y 3100 in quanto essa potrà assumere il valore -1

pingpong.scoreA='0'; //effettuo un reset del punteggio del peer che ha richiesto il restart della sessione di gioco

$("#scoreA").html(pingpong.scoreA);

pingpong.scoreB='0';

$("#scoreB").html(pingpong.scoreB);

}
```

2) Ricezione dei parametri ed applicazione delle modifiche necessarie:

```
//in assenza della possibilità di gestire più di un canale di comunicazione fra i peer, ma conoscendo il range di valori che può assumere la posizione e la direzione della pallina (trovandosi in un area circoscritta), abbiamo deciso di inviare sul canale un numero compreso fra 1000 e 2000 per la posizione X della pallina (sommando 1000 al valore di ball.x); la posizione di Y assumerà valori compresi fra 2000 e 2800; le direzioni di X e Y saranno rispettivamente comprese fra 3000-3500 e 4000-5000 (la direzione varrà -1 o 1). Il numero ricevuto sul data channel verrà poi decrementato per essere utilizzato dal gioco.

if(1000<=event.data && event.data<=2000){

console.log("posX=");

posX = event.data-1000;

console.log(posX);



}

if(2000<=event.data && event.data<=2800){

console.log("posY=");

posY = event.data-2000;

console.log(posY);

}



if(3000<=event.data && event.data<=3500){

console.log("dirX=");

dirX = event.data-3100;

console.log(dirX);

}



if(4000<=event.data && event.data<=5000){
```

```
console.log("dirY=");

dirY = event.data-4100;

console.log(dirY);



//dopo aver ricevuto l'ultimo parametro di gioco mi occupo del reset del punteggio e dei parametri appena acquisiti del peer remoto

var ball=pingpong.ball;



ball.x=posX;

ball.y=posY;



$("#ball").css({

 "left": ball.x,

 "top" : ball.y

 });

 ball.directionX = dirX;

ball.directionY = dirY;

//reset dello score

pingpong.scoreA='0';

$("#scoreA").html(pingpong.scoreA);

pingpong.scoreB='0';

$("#scoreB").html(pingpong.scoreB);

}
```

BIBLIOGRAFIA

1) Data channel (nessun meccanismo di signaling):
https://github.com/ldecicco/webrtc-demos/blob/master/datachannel.html

2) NPM (utile all'installazione rapida di node.js):

https://www.npmjs.com/

3) Node:

https://nodejs.org/it/

4) Socket.io:

https://socket.io/docs/

5) WebRTC codici di esempio:

https://codelabs.developers.google.com/codelabs/webrtc-web/#0

5) RTCPeerConnection:

https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection

6) Pascal Retting, Professional HTML5 Mobile Game Development

7) Markzan, HTML5 Games Development by Example