

# **Relazione Progetto Reti di Calcolatori**

Prof. Antonio Della Selva  
Sessione Estiva 2015/2016

Leonardo Manoni [MAT. 261049]  
Matteo Incicco [MAT. 261716]

# INTRODUZIONE

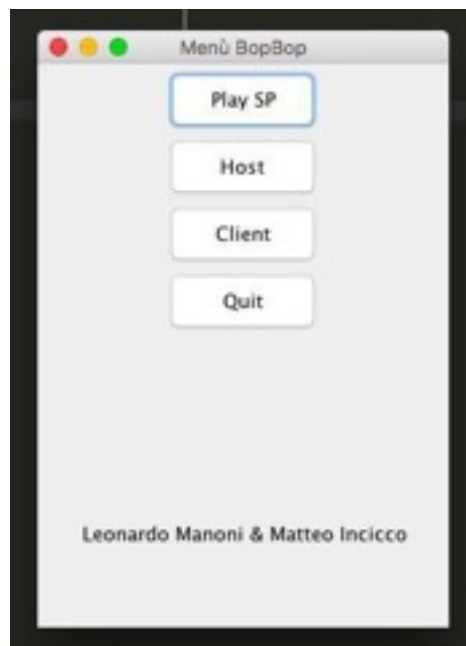
## 1. SCELTA DEL PROGETTO

Realizzazione di un Gioco, chiamato **BOPBOP**.  
Linguaggio di Programmazione utilizzato: **JAVA**.

*L'idea di scegliere questo gioco nasce dall'esigenza di sviluppare e approfondire le interfacce grafiche di java, per mettersi in gioco e per affrontare in prima persona problemi di comunicazione riguardanti la rete e approfondendo alcuni argomenti affrontati in classe.*

## PRESENTAZIONE GIOCO

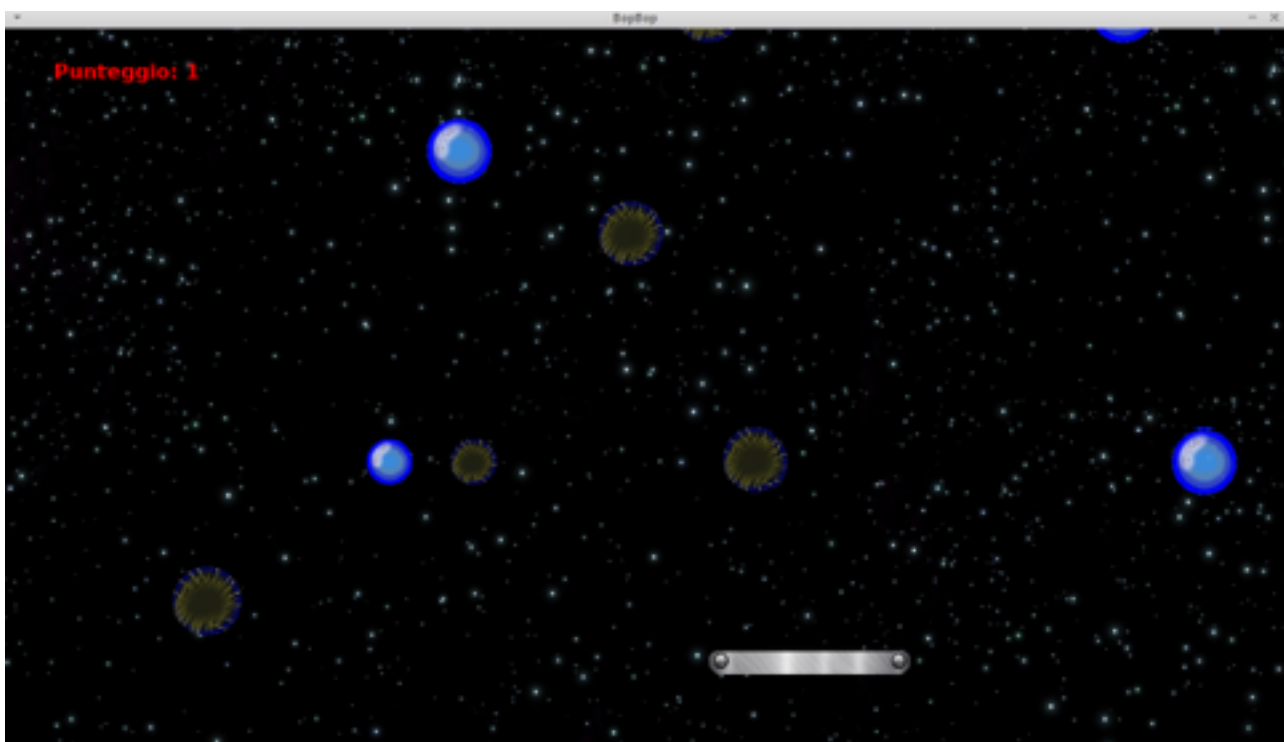
Il gioco consiste nel prendere, attraverso una barra metallica, le bolle con lo scopo di incrementare il proprio punteggio e/o detrarre il punteggio dell'avversario. Le bolle cadono dall'alto con velocità e dimensione variabili. La barra (anche chiamata in fase di sviluppo cesto), può solamente muoversi sull'asse orizzontale attraverso l'uso dei tasti direzionali o attraverso l'utilizzo del mouse. Il gioco parte con un **menù iniziale** che ci permette di scegliere la modalità con cui giocare.



Nella modalità **SinglePlayer** (locale), abbiamo due tipi di bolle, la “buona” e la “cattiva”.

La bolla “buona”, la Bolla Blu, incrementa il punteggio del Player di una unità, mentre la bolla “cattiva”, la Bolla Nera, decrementa il punteggio del Player di una unità.

L'abilità richiesta al giocatore sarà quella di prendere più Bolle Blu possibili schivando le Bolle Nere.



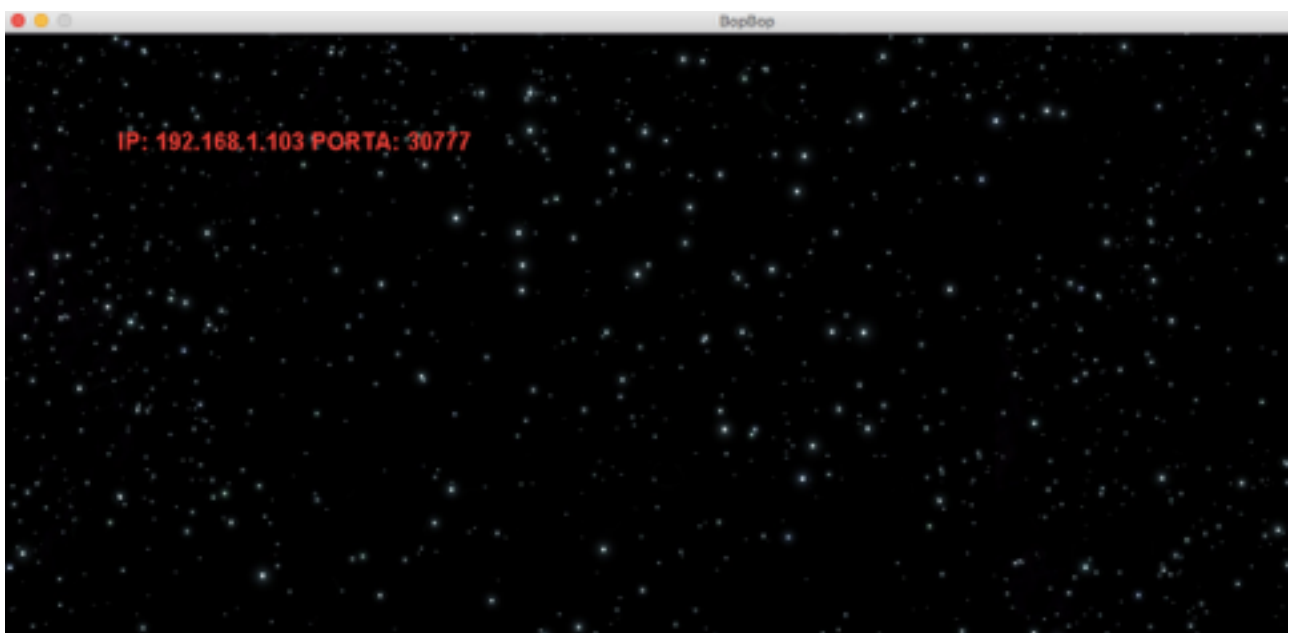
Oltre alla modalità SinglePlayer, c'è la possibilità che due utenti si sfidino a vicenda in rete.

#### Modalità **Host** o **Client**.

I due giocatori oltre a vedere in tempo reale il punteggio dell'avversario avranno la possibilità di penalizzare l'altro utente catturando la Bolla Gialla (Bolla Jolly) che ha la caratteristica di togliere un punto sul punteggio dell'avversario.

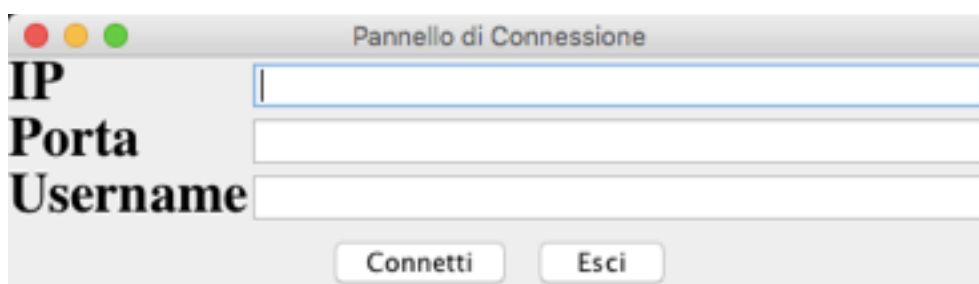
#### **Funzionamento:**

Nel momento in cui l'utente clicca il bottone Host, si mette in attesa di un Client e viene visualizzato un frame con su scritto l'indirizzo ip e la porta su cui quest'ultimo deve accedere per cominciare la partita.



Il Client, dopo aver effettuato l'accesso come tale, inserisce: IP, PORTA e USERNAME.

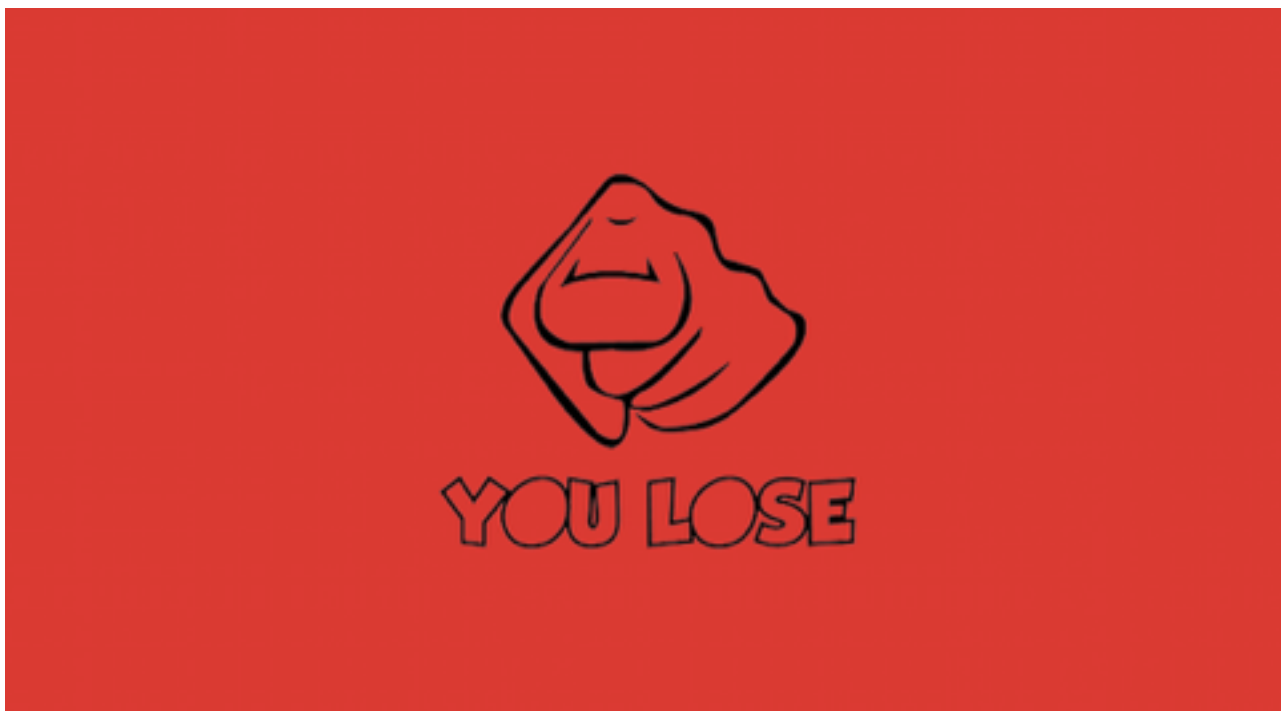
Nel momento in cui preme il tasto avvia, le due sessioni di gioco si avviano in sincronia.

A screenshot of a window titled "Pannello di Connessione". It has a light gray background. On the left side, there are three labels: "IP", "Porta", and "Username" in a bold, black, sans-serif font. To the right of each label is a white rectangular input field with a thin blue border. At the bottom of the window, there are two buttons: "Connetti" and "Esci", both with a light gray background and a thin border.

Il gioco **termina** quando uno dei due sfidanti raggiunge il punteggio di 50 punti.



Schermata di vittoria



Schermata di perdita

# PROTOCOLLO UTILIZZATO

Trattandosi di un gioco, abbiamo scelto di utilizzare il protocollo UDP, preferendo la velocità dello scambio dei dati alla sicurezza garantita dal protocollo TCP.

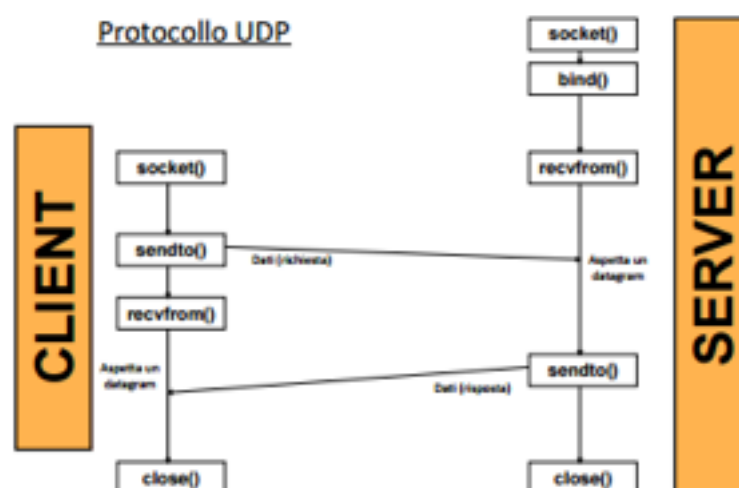
Il **protocollo UDP** (User Datagram Protocol) utilizza i datagrammi, cioè pacchetti di dati, di dimensione fissa, che vengono inviati velocemente senza attendere e senza assicurarsi che il destinatario li abbia ricevuti, ma continuando ad inviare pacchetti.

In pratica una comunicazione UDP non dà alcuna garanzia di ricezione dei dati, ma ha il vantaggio di far comunicare tra loro due computer molto più rapidamente.

Il server crea un socket, invoca `bind()` e attende, tramite la funzione `recvfrom()`, di ricevere dati. Il client crea il proprio socket, e invia dati tramite la funzione `sendto()`. L'interazione continua attraverso chiamate da ambo le parti a `recv` e `send`.

Alla fine, sia il server che il client chiudono i rispettivi socket.

La nostra socket è statica ed è la 30777.



# IMPLEMENTAZIONE E RISOLUZIONI PROBLEMI

## Problema della connessione tra Client e Host

Uno dei principali problemi riscontrati è stato quello di far avviare le due sessioni di gioco nello stesso istante, in modo da non favorire nessun giocatore.

Come già spiegato, il Giocatore Host una volta entrato, resta in attesa di un Client. Solo quando quest'ultimo si collega con l'Host, entrambi cominciano a giocare. Per connettersi, il Client dovrà scrivere le coordinate indicate dall'host nel pannello di connessione.

### 1. Problema della sincronizzazione delle istanze di gioco.

In progetti simili il nodo centrale del lavoro riguarda senza dubbio la corretta sincronizzazione delle due istanze di gioco: quella dell'Host e quella del Client.

Esse devono avviarsi nello stesso istante.

Nel caso in cui non si avviassero simultaneamente, un giocatore potrebbe essere avvantaggiato rispetto all'altro, in quanto potrebbe iniziare ad accumulare punteggio prima dell'avversario.

Per far ciò l'Host avvia subito la sua istanza di gioco, ma rimarrà bloccato nella sezione in cui si visualizzano solamente le coordinate.

```
private void disegna_coordinate()
{
    BufferStrategy buffer = this.getBufferStrategy();
    if(buffer == null)
    {
        createBufferStrategy(2);
        return;
    }

    Graphics g = buffer.getDrawGraphics();

    g.setFont(new Font("Arial", Font.BOLD, fontsize));
    g.setColor(Color.red);

    g.drawImage(sfondo, 0, 0, larghezza, altezza, this);
    g.drawString("IP: " + myhost.get_ip() + " PORTA: " + myhost.get_porta(), 100, 100);

    g.dispose();
    buffer.show();
}
```

Solo quando riceverà il primo messaggio da parte del Client, il Server andrà a “segnalare” alla sua istanza di gioco che la connessione è andata a buon fine. Per far ciò chiama un metodo della classe Gioco (“parti()”), che andrà a modificare un flag utilizzato come controllo.

```
if(messaggio != null && cont==0)
{
    // invia un messaggio al client per la sincronizzazione iniziale
    try
    {
        invia_punteggio(0);
    } catch (IOException ex)
    {
        Logger.getLogger(ServerUDP.class.getName()).log(Level.SEVERE, null, ex);
    }

    // modifico il flag in gioco, necessario per farlo partire
    gioco.parti();
    cont++;
}
```



```
// se è diverso da null vuol dire che sto esguendo come host
if(myhost != null)
{
    while(giocoAttivo)
    {
        // controllo sul flag, se falso vuol dire che il client non si è connesso e,
        // di conseguenza, visualizzerà soltanto le sue coordinate.
        if (partito == false)
        {
            disegna_coordinate();
        }
        // altrimenti se il flag è vero vuol dire che il client si è sincronizzato e
        // che può partire il gioco vero e proprio
        else
        {
            if (this.punteggio >= 50)
            {
                // ho vinto quindi stampo la vittoria
                disegna_vinto();
            }
            else if (this.punteggio_avversario >= 50)
            {
                // ho perso quindi stampo la perdita
                disegna_perso();
            }
            else
            {
                try
                {
                    aggiorna_host();
                } catch (IOException ex)
                {
                    System.out.println("Errore nell'aggiorna_host" + ex);
                }
                disegna_host();
            }
        }
    }
}
```



## 2. Problema della validazione delle coordinate

È necessario introdurre un controllo anche per la corretta sincronizzazione del client. Così restando, senza apportare ulteriori modifiche, se quest'ultimo inserisce delle coordinate sbagliate, esso farà partire subito la sua istanza di gioco senza concorrere con l'Host, in quanto, quest'ultimo, non ha ricevuto nessun messaggio.

Per ovviare questo problema abbiamo introdotto una "validazione" delle coordinate.

L'host appena ricevuto il messaggio da parte del client gli reinvia un messaggio di conferma e farà partire il gioco.

Il client rimarrà in attesa del messaggio di ritorno che, appena ricevuto, fa partire la sua istanza.

```
// invia un messaggio al client per la sincronizzazione iniziale
try
{
    invia_punteggio(0);
} catch (IOException ex)
{
    Logger.getLogger(ServerUDP.class.getName()).log(Level.SEVERE, null, ex);
}
```

Il server reinvia un messaggio subito dopo averlo ricevuto dal client.

Appena ricevuto il  
messaggio farà partire  
l'istanza di gioco

```
try
{
    c.receive(recv);
} catch (IOException ex)
{
    System.out.println("Errore nella ricezione");
}

if(cont == 0)
{
    System.out.println("CONNESSIONE STABILITA");
    gioco = new Gioco(this, null);
    cont++;
}
```

L'uso di un contatore serve per far partire una sola volta l'istanza di gioco.

## Problema dei punteggi

Lo scambio dei dati tra i due giocatori avviene ogni qual volta si ha un aggiornamento dei punteggi. Sostanzialmente si possono evidenziare due casi:

### 1° caso

Il giocatore (barra) collide con una bolla blu o una bolla nera. Il suo punteggio, rispettivamente, aumenta o diminuisce di un punto. A questo punto è necessario inviare il punteggio aggiornato all'avversario. Per far ciò, la classe Gioco invoca una funzione chiamata `invia_punteggio()`. Questa funzione utilizza il comando "send" per inviare all'avversario (host o client) il punteggio. Una volta ricevuto, la classe `ServerUDP` o `ClientUDP` richiama una funzione presente nella classe `Gioco`: "aggiorna\_punteggio". Quest'ultima aggiorna di conseguenza il valore presente nella variabile "punteggio\_avversario" che verrà stampata a video sul frame dello sfidante.

```
public void invia_punteggio(int punteggio) throws IOException
{

    String punteggi = String.valueOf(punteggio);
    try
    {
        addressCLIENT = InetAddress.getByName(client);
    } catch (UnknownHostException ex)
    {
        System.out.println("Errore nella creazione del IP CLIENT " + ex);
    }
    byte [] msg = {0};
    per_client = new DatagramPacket(msg, msg.length, addressCLIENT, portCLIENT);

    per_client.setData(punteggi.getBytes());
    per_client.setLength(punteggi.length());
    c.send(per_client);
}
```

```
146
147     public void invia_punteggio(int punteggio) throws IOException
148     {
149
150         String punteggi = String.valueOf(punteggio);
151         hi.setData(punteggi.getBytes());
152         hi.setLength(punteggi.length());
153         s.send(hi);
154     }
155 }
156
```

## 2° caso

Il giocatore (barra) collide con la bolla gialla. Il punteggio dell'avversario decrementa di una unità. Per evitare la creazione di un ulteriore canale, mandiamo un messaggio fittizio che verrà riconosciuto dalla classe Gioco e detrae un punto al proprio punteggio.

```
if(b instanceof BollaGialla)
{
    // invio 8888 così sà che deve detrarre punti
    this.myclient.invia_punteggio(8888);
}
```

```
public void aggiorna_punteggio(String nuovo_punteggio) throws IOException
{
    this.temp = Integer.parseInt(nuovo_punteggio);

    // se il messaggio ricevuto è uguale a 8888 allora vuol dire che devo detrarre punti al
    // mio punteggio, altrimenti aggiorno il punteggio avversario
    if (this.temp == 8888)
    {
        this.punteggio = this.punteggio - 1;

        // invio il punteggio aggiornato
        if(this.myhost != null)
        {
            this.myhost.invia_punteggio(this.punteggio);
        }
        else
        {
            this.myclient.invia_punteggio(this.punteggio);
        }
    }
    else
    {
        this.punteggio_avversario = Integer.parseInt(nuovo_punteggio);
    }
}
```

## CONCLUSIONI

È stato bello collaborare per la realizzazione di questo progetto. Siamo riusciti ad affrontare con successo le varie tematiche (e di conseguenza i vari problemi) proposte, districandoci tra i vari particolari della parte grafica, della programmazione ad oggetti e della connessione tra computer.

Di conseguenza, siamo felici di aver affrontato nella pratica il protocollo UDP del livello transport.