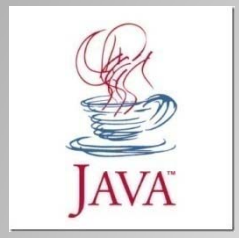
Caso di studio di
Reti di Calcolatori e Internet

Prof. Nicola Di Mauro

a.a. 2008/2009

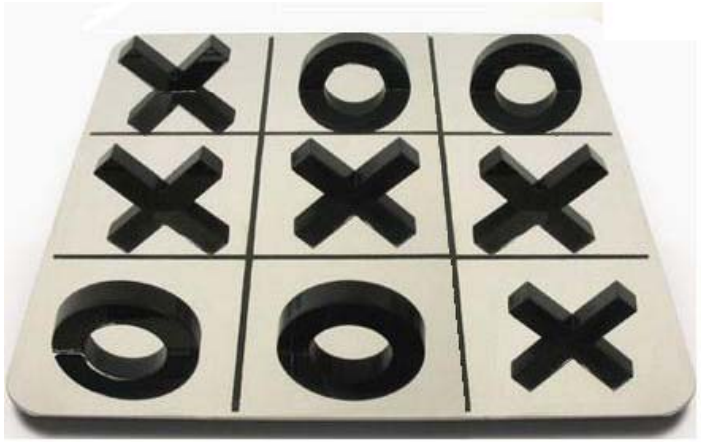
Titolo: NetTris

Autore: Francesco Serafino



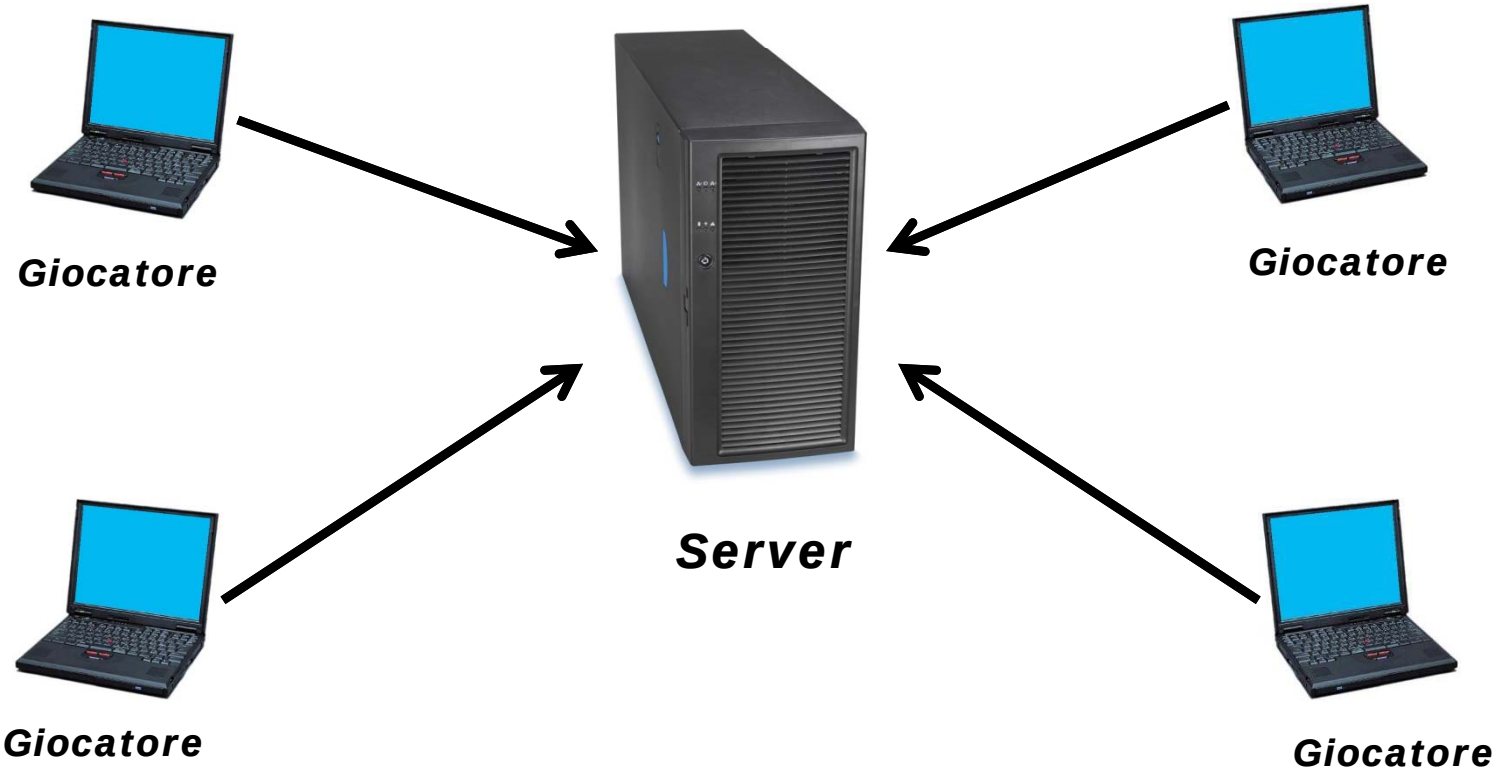
NetTris

Applicativo Java che consente di giocare a Tris in rete



Cos’è NetTris?

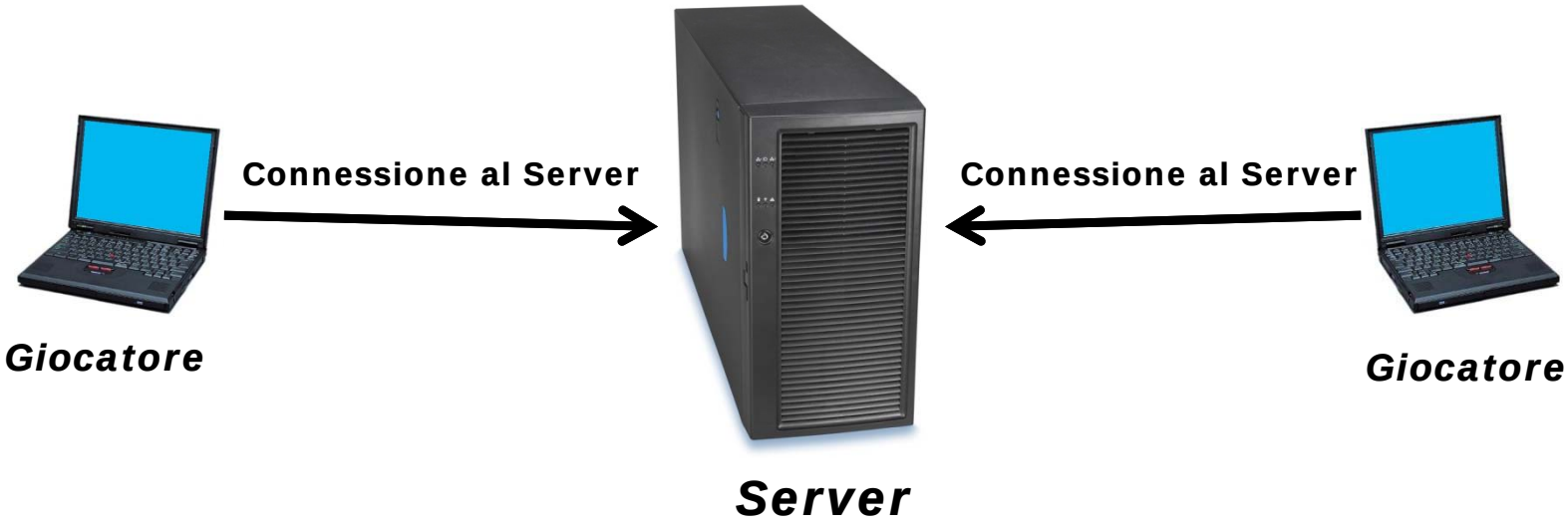
NetTris è un programma che consente a due o più utenti connessi ad internet di effettuare partite a Tris in rete tramite un Server che smista le richieste.



Come si stabilisce la connessione?

Per poter avviare la partita avvengono due connessioni identificabili da due fasi che si susseguono:

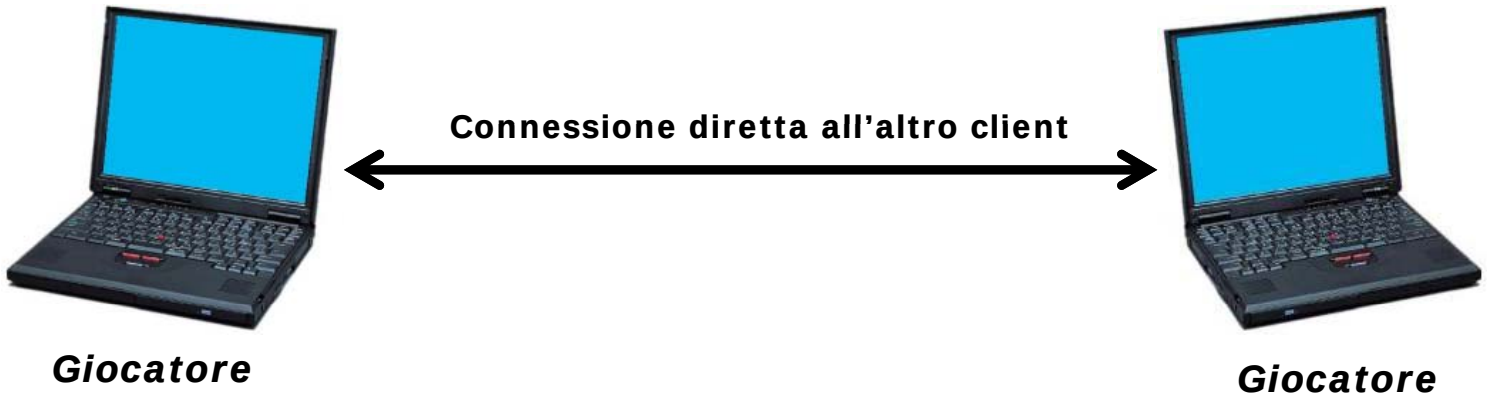
Prima fase: I client si collegano al Server



Come si stabilisce la connessione?

Non appena il server trova due client connessi e pronti, fornisce loro i dati per farli comunicare tra loro e la connessione col server si chiude e inizia la seconda fase:

Seconda Fase: I due client iniziano a comunicare tra loro indipendentemente dal Server e avviano la partita.



Come smista le richieste il Server?

Il server smista le richieste secondo la seguente logica:

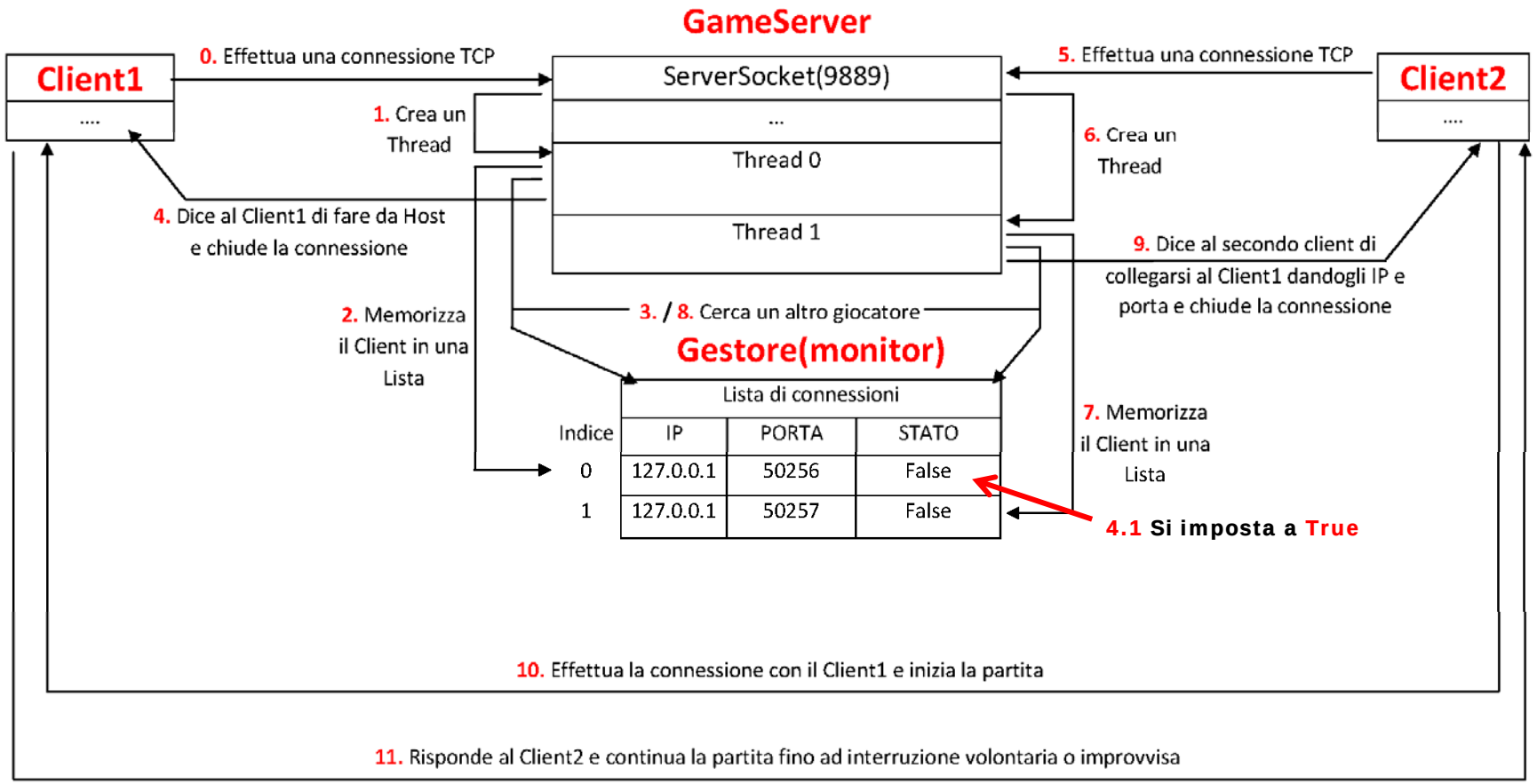
• Ogni qual volta un nuovo Client desidera effettuare una partita si **collega al Server** del gioco e attende che il Server trovi un altro giocatore.

• Se ci sono almeno **due** Client connessi al Server, quest'ultimo per ogni coppia di Client invia, al primo Client connesso (cronologicamente) il comando di fare da **Host** e al secondo Client l'**IP** e la **porta** del primo Client.

• Non appena i Client hanno tutte le informazioni per avviare la partita, entrambi **chiudono** la connessione col server e iniziano a comunicare **tra loro**. Il primo Client connesso al Server fa da **host** e l'altro esegue una **connessione TCP** al primo.

Per poter gestire più client, il server rappresenta un Client connesso come un **Thread**. Pertanto il server è implementato in **Multithreading**

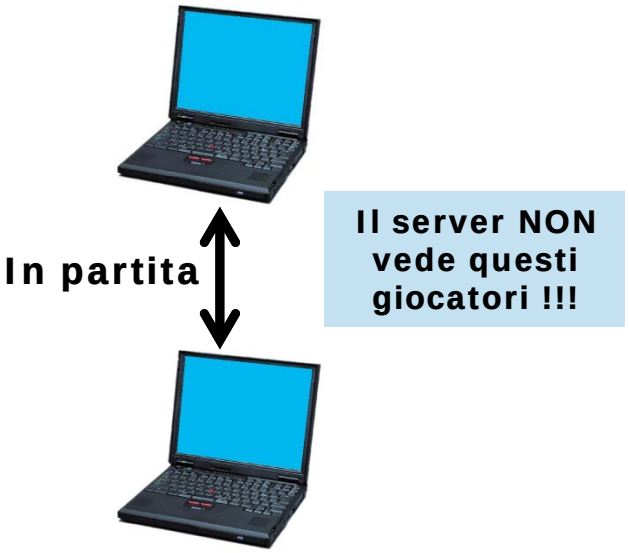
Schema di funzionamento del Server

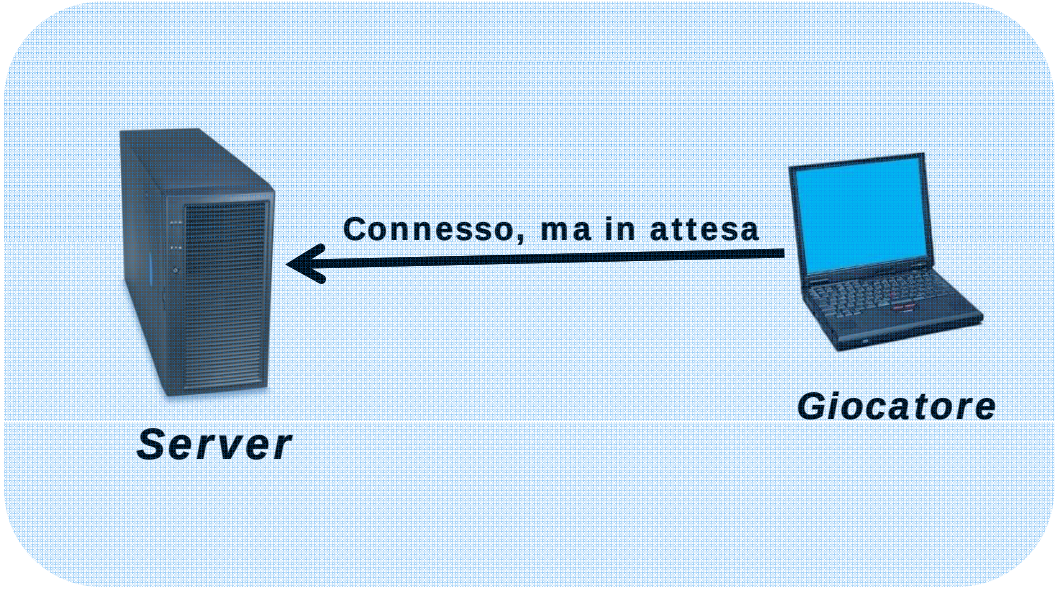


N.B. La sequenza dei numeri indica l'ordine cronologico in cui dovrebbero avvenire i flussi, ma avendo affidato ad ogni Client un Thread è chiaro che nella realtà l'ordine cronologico effettivo potrebbe non essere quello qui rappresentato. Ciò nonostante i processi giungono comunque a termine poiché le operazioni critiche sono affidate ad un monitor (chiamato Gestore) e al limite potrà accadere che il Client che farà da host non sarà il Client1, ma il Client2 (o comunque un qualunque Client connesso al Server).

E se i Client sono più di due?

Se ci sono più Client connessi al Server questi verranno smistati in ordine di arrivo **cronologico**, ma sempre a due a due. Ciò significa che se il numero dei Client connessi è dispari, ci sarà un giocatore che **attenderà** fino a quando un nuovo giocatore farà richiesta di giocare o fino a quando un giocatore che è già in partita termina la connessione in cui è impegnato e decide di effettuare un'altra partita.



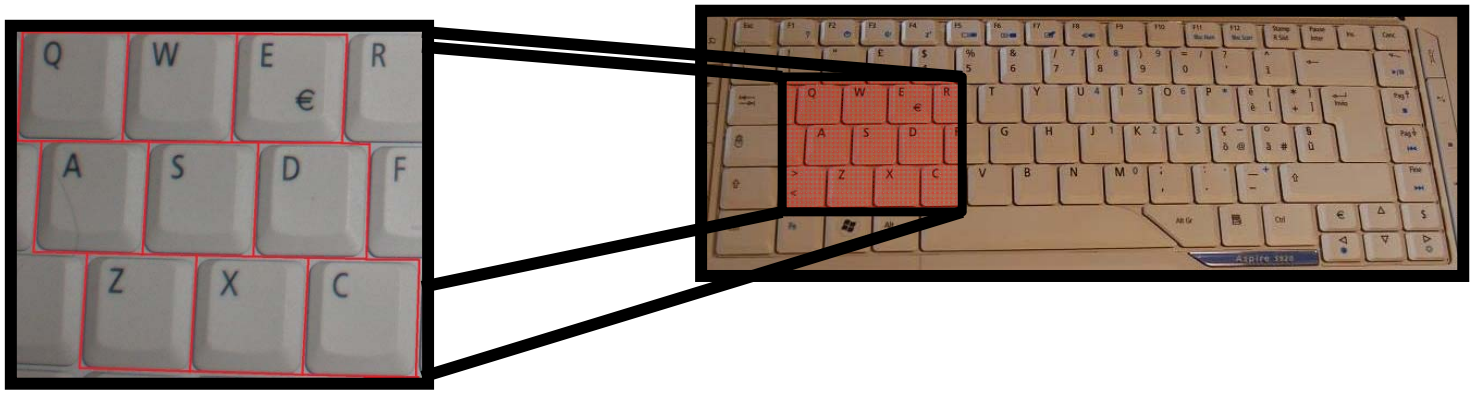


Come avviene la partita?

Entrambi i giocatori, dopo essere stati messi in comunicazione tra loro dal Server, giocano tra loro utilizzando le lettere della tastiera:

```
+---+---+---+
| Q | W | E |
+---+---+---+
| A | S | D |
+---+---+---+
| Z | X | C |
+---+---+---+
```

Ogni lettera corrisponde ad una casella della tabella del gioco del tris. Sono state scelte queste lettere poiché sono le prime tre lettere delle tre righe di tasti della tastiera standard.

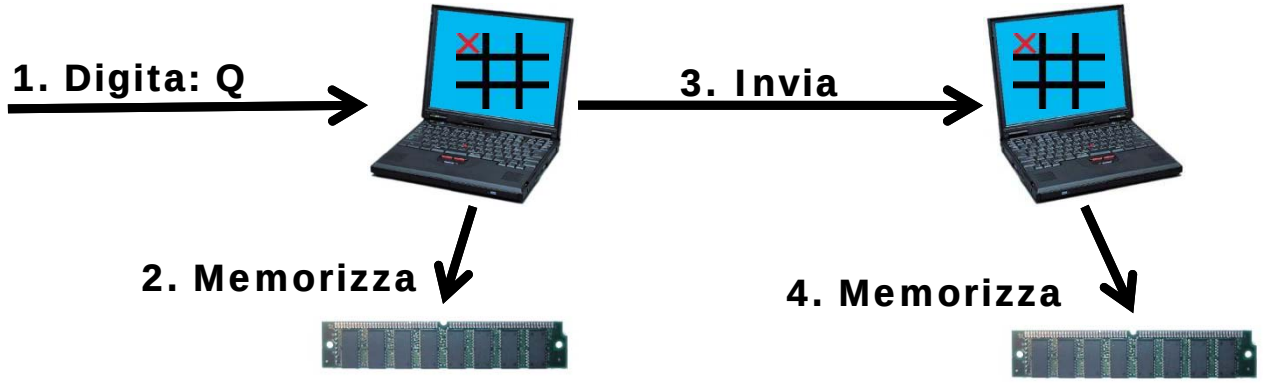


Come avviene la partita?

Quando è il proprio turno, il giocatore digita la mossa, la memorizza e la invia all'altro giocatore nel seguente modo:

• Ognuno dei due client comunica con l'altro tramite dei **flussi di stringhe** che alternano un **invio** di testo ad una **ricezione** di testo. Questo per far si che ogni volta che un giocatore inserisce la propria mossa questa viene inviata all'altro client e **memorizzata** da entrambe le parti.

• Alla chiusura della connessione tra i due Client (alla fine della partita o per improvvisa disconnessione della rete) viene posta a entrambi la possibilità di ricollegarsi al server per una **nuova partita** o, in caso di risposta negativa, l'**uscita** dal programma.



Quando finisce la partita?

Quando finiscono le caselle o quando un giocatore mette in fila tre dei suoi simboli orizzontalmente, verticalmente o sulle diagonali la partita finisce stampando un messaggio a video a entrambi i giocatori mostrando chi ha vinto.

Per fare ciò ogni volta che viene inserita o ricevuta una mossa si effettua un **controllo** sulle righe, sulle colonne e sulle diagonali e se si trova un vincitore la partita termina.

Ad esempio al vincitore verrà stampato:
`Partita finita! Hai vinto!`

Al perdente verrà stampato:
`Partita finita! Ha vinto l'avversario!`

In caso di partita patta verrà stampato:
`Partita Patta`

Il Codice

```
                import
        java.io.PrintStream;
     public class ASmileyFace {
  public static void main(String[]
a) { PrintStream out = System.out ;
out.print(     "This"     + " is " +
"both a "        +         "reform"+
"atted J"    +  "av"  +    "a prog"+
"ram, an"        +         "d a sm"+
"i".concat(   "ley " +   "face. Th"+
"is "    + "shows how to " +    "f"+
  "orm"     + "at Jav" +     "a "+
    "usi"                  +"n"+
      "g whitespace only.")
             } ;) }


  PROGRAMMING IN THE 21ST CENTURY
  SURE DID GAIN SOME AFFECTATIONS
  DON'T YOU THINK SO ??????? I DO
```

Il Codice: l'IP del Server

Come il Client ottiene l'indirizzo IP del Server:

```
53  do
54  {
55      System.out.println("Inserisci l'IP del Server del Gioco. Se non inserisci ni
56      input = inFromUser.readLine();
57      if (input.equals(""))
58      {
59          input = "localhost";
60      }
61      else
62      {
63          try
64          {
65              InetAddress prova = InetAddress.getByName(input);
66          }
67          catch (UnknownHostException k)
68          {
69              System.out.println("Hai inserito un Indirizzo non valido!");
70              input = "error";
71          }
72      }
73  }
74  while (input.equals("error"));
```

Nota: La prima istruzione di stampa riporta il seguente testo:
"Inserisci l'IP del Server del Gioco. Se non inserisci niente sarà preso di default il localhost"

Il Codice: Connessione al Server

Come avviene la connessione al Server dal lato Client:

```
79  /*Si collega al server dei giochi (GameServer)*/
80  Socket client = new Socket (input, 9889);
81  DataInputStream inFromServer = new DataInputStream(client.getInputStream());
82  DataOutputStream outToServer = new DataOutputStream(client.getOutputStream());
83
84  int porta = client.getLocalPort();
85  String portaString = porta + "\n";
86
87  outToServer.writeBytes(portaString);
88  outToServer.flush();
89
90  System.out.println(inFromServer.readLine());
91  System.out.println(inFromServer.readLine());
92
93  String ipHost = inFromServer.readLine();
94  String portaHostString = inFromServer.readLine();
95  int portaHost = Integer.parseInt(portaHostString);
96
97  inFromServer.close();
98  outToServer.close();
99  client.close();
```

Il Codice: Lato Server

Cosa accade quando arriva una richiesta di connessione al Server:

```
14  /*Classe principale che crea il server*/
15  public class GameServer
16  {
17      public static void main (String args[]) throws Exception
18      {
19          Gestore gestore = new Gestore();
20          ServerSocket myServer = new ServerSocket(9889);
21          System.out.println("Il server e' attivo sulla porta 9889. Attendo connessioni...");
22
23          while (true)    //Il server e' sempre in ascolto di nuove connessioni
24          {
25              Socket connessione = myServer.accept();
26              if (connessione != null) //Per ogni connessione trovata crea un Thread
27              {
28                  Giocatore giocatore = new Giocatore(connessione, gestore);
29                  giocatore.start();
30              }
31          }
32      }
33  }
```

Il Codice: I thread – pag 1/3

Campi d'esemplare e costruttore:

```
35  /*Classe che estende la classe Thread per gestire tutti i Client connessi*/
36  class Giocatore extends Thread
37  {
38      private Socket connessione;
39      private Gestore gestore = new Gestore();
40
41      private BufferedReader inFromClient;        //Buffer per ricevere dati dal client
42      private DataOutputStream outToClient;       //Stream per inviare dati al client
43
44      private String ip = "";             //contiene l'ip del client
45      private String porta = "";          //contiene la porta del client
46
47      private int posizione = -1;         //posizione del client nell'ArrayList
48      private Utente utente = new Utente();
49
50      /*Costruttore che memorizza al suo interno il riferimento alla connessione e al gestore e inizializz
51      public Giocatore(Socket s, Gestore g) throws IOException
52      {
53          gestore = g;
54          connessione = s;
55          inFromClient = new BufferedReader(new InputStreamReader(connessione.getInputStream()));
56          outToClient = new DataOutputStream(connessione.getOutputStream());
57      }
58
```

Il Codice: I thread – pag 2/3

Metodo run() – Invio di messaggi di avvenuta connessione al client:

```
60      public void run()
61      {
62          try
63          {
64              InetAddress clientIp = connessione.getInetAddress();
65              String ip = clientIp.getHostAddress(); //memorizzo l'ip del Client
66
67              porta = inFromClient.readLine(); //ricevo dal Client la sua porta
68
69              System.out.println("Il client con IP: " + ip + " e con porta: " + porta + " e' connesso
70
71              outToClient.writeBytes("Sei connesso al Server del Gioco!\n");
72              outToClient.flush();
73
74              outToClient.writeBytes("Ricerca di un altro giocatore!\n");
75              outToClient.flush();
76
77
78              posizione = gestore.entra(ip, porta); //invia al gestore ip e porta e gli viene restitui
79              System.out.println("Il client ha posizione " + posizione + " nel server.");
80
81              utente = gestore.avviaPartita(posizione); //chiede di avviare la partita e gli viene res
82
```

Il Codice: I thread – pag 3/3

Metodo run() – Invio delle istruzioni al Client per avviare la partita:

```
83              /*Invia al client le istruzioni per la nuova connessione in base allo stato*/
84              if(utente.getStat())
85              {
86                  outToClient.writeBytes("host\n");
87                  outToClient.flush();
88                  outToClient.writeBytes(utente.getPort() + "\n");
89                  outToClient.flush();
90              }
91              else
92              {
93                  outToClient.writeBytes(utente.getIp() + "\n");
94                  outToClient.flush();
95                  outToClient.writeBytes(utente.getPort() + "\n");
96                  outToClient.flush();
97              }
98              /*Chiude gli Stream e la Socket con il client*/
99              outToClient.close();
100             inFromClient.close();
101             connessione.close();
102         }
103         catch(IOException e)
104         {
105             System.out.println("Eccezione!");
106         }
107     }
108 }
109 /*Fine del Thread*/
```

Il Codice: Il Monitor Gestore – pag 1/3

Costruttore e metodo entra():

```
113 class Gestore
114 {
115     private ArrayList<Utente> utenti = new ArrayList<Utente>(); //Utilizza un ArrayList di oggetti per
116     private ArrayList<Integer> posti = new ArrayList<Integer>();//Utilizzato per memorizzare le posizi
117 
118     public Gestore()
119     {
120         //Costruttore Vuoto
121     }
122 
123     /*Metodo sincronizzato a cui accedono tutti i Thread per salvare le informazioni dei client ad ess
124     public synchronized int entra(String ip, String porta)
125     {
126         int posizione = -1;
127         if (posti.size() > 0) //Verifica se ci sono posti liberi prima di allocarne nuovi
128         {
129             utenti.remove(posti.get(0));
130             utenti.add(posti.get(0), new Utente(ip, porta, false));
131             posizione = posti.get(0);
132             posti.remove(0);
133             return posizione;   //Restituisce la posizione nell'ArrayList
134         }
135         else
136         {
137             utenti.add(new Utente(ip, porta, false));
138             posizione = utenti.size() - 1;
139             return posizione;   //Restituisce la posizione nell'ArrayList
140         }
141     }
```

Il Codice: Il Monitor Gestore – pag 2/3

Metodo avviaPartita() – Ricerca del giocatore:

```
144     public synchronized Utente avviaPartita(int posizione)
145     {
146         /*Inizio algoritmo di ricerca altri client connessi*/
147         Utente temp = new Utente();
148         int i = 0;
149         boolean trovato = false;
150         int miaPosizione = posizione;
151         int altraPosizione = -1;
152 
153         while ((i < utenti.size()) && (!trovato))
154         {
155             temp = utenti.get(i);
156             if (temp.getStat())
157             {
158                 trovato = true;
159                 altraPosizione = i;
160             }
161             i++;
162         }
163         /*Fine algoritmo di ricerca altri client connessi*/
```

Il Codice: Il Monitor Gestore – pag 3/3

Metodo avviaPartita() – Invio istruzioni al Thread e pulizia ArrayList:

```
165         /*Invia le istruzioni al client nel seguente modo:
166          *Se ha trovato un altro client che sta facendo da host vi si connette, altrimenti inizia a fa
167         if (altraPosizione == -1)
168         {
169             temp = utenti.get(miaPosizione);
170             temp = new Utente(temp.getIp(), temp.getPort(), true);
171             utenti.remove(miaPosizione);
172             utenti.add(miaPosizione, temp);
173             return temp;
174         }
175         else
176         {
177             temp = utenti.get(altraPosizione);
178             temp = new Utente(temp.getIp(), temp.getPort(), false);
179             utenti.remove(altraPosizione);
180             utenti.add(altraPosizione, temp);
181 
182             /*Ha trovato 2 client che vogliono giocare tra loro e provvede alla rimozione dall'ArrayLi
183             if (altraPosizione < miaPosizione)
184             {
185                 posti.add(altraPosizione); //aggiunge all'ArrayList posti i posti appena liberati senz
186                 posti.add(miaPosizione); //aggiunge all'ArrayList posti i posti appena liberati senza
187             }
188             else
189             {
190                 posti.add(miaPosizione);  //aggiunge all'ArrayList posti i posti appena liberati senza
191                 posti.add(altraPosizione);   //aggiunge all'ArrayList posti i posti appena liberati se
192             }
193             System.out.println("I client con posizioni " + miaPosizione + " e " + altraPosizione + " s
194             System.out.println("I client con posizioni " + miaPosizione + " e " + altraPosizione + " s
195             return temp;
196         }
197     }
198 }
```

Il Codice: L'oggetto Utente

Come è rappresentato l'oggetto di tipo Utente:

```
201 class Utente
202 {
203     private String indirizzo;
204     private String porta;
205     private boolean stato;
206 
207     public Utente()
208     {
209         //Costruttore vuoto
210     }
211 
212     /*Costruttore che definisce un oggetto di tipo Utente formato da IP, porta e stato*/
213     public Utente(String ip, String port, boolean stat)
214     {
215         indirizzo = ip;
216         porta = port;
217         stato = stat;
218     }
219 
220     /*Metodi che restituiscono rispettivamente: IP, porta e stato al chiamante*/
221     public String getIp()
222     {
223         return indirizzo;
224     }
225 
226     public String getPort()
227     {
228         return porta;
229     }
230 
231     public boolean getStat()
232     {
233         return stato;
234     }
235 }
```

Il Codice: Ritorno al Client

Dopo che il server ha cercato un giocatore avvia la partita secondo la modalità indicata dal Server (Host/Join):

```
101 Partita partita = new Partita();
102 /*Non appena il server risponde al client, questo controlla se dovra' fare hosting o dovra' connettersi
103 if (ipHost.equals("host"))
104 {
105     System.out.println("Nessun giocatore disponibile. In attesa che un altro giocatore si colleghi...");
106     partita.host(portaHost);
107 }
108 else
109 {
110     System.out.println("Trovato un altro giocatore. Inizio partita...");
111     partita.join(ipHost, portaHost);
112 }
```

Il Codice: Client che fa da HOST

Quando il client riceve istruzione di fare HOST, esegue:

```
164 public void host(int porta) throws Exception
165 {
166     try
167     {
168         ServerSocket myServer = new ServerSocket(porta);     //ServerSocket per la connessione
169         String mossaUscita = "";          //mossa da inviare all'altro client
170         String mossaEntrata = "";         //mossa ricevuta dall'altro client
171         String r = "";                    //Stringa contentente la riga
172         String c = "";                    //Stringa contentente la colonna
173         int riga;                         //intero contentente la riga
174         int colonna;                      //intero contentente la colonna
175 
176         while (true)     //Ciclo infinito (in realta' se la connessione viene accettata, si avvia una sola volta)
177         {
178             try
179             {
180                 /*Inizializzazione della connessione e dei vari Stream di comunicazione*/
181                 Socket connessione = myServer.accept();
182                 BufferedReader inFromPeer = new BufferedReader(new InputStreamReader(connessione.getInputStream()));
183                 DataOutputStream outToPeer = new DataOutputStream(connessione.getOutputStream());
184                 BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
185 
186                 /*Avvisa l'utente che sara' il secondo ad avere la mossa*/
187                 System.out.println("Trovato un altro giocatore. Inizio partita...");
188                 System.out.println("Stai giocando come Pallino {O} ed e' il turno dell'avversario...");
189                 setToEmpty(); //azzera la matrice
190                 printTable(); //stampa la griglia (vuota)
191                 ... altre istruzioni ...
```

e poi...

```
261     ... altre istruzioni ...
262     /*Chiude la connessione e gli Stream*/
263     inFromPeer.close();
264     outToPeer.close();
265     connessione.close();
266     break; //termina il ciclo a fine partita
267 }
```

Il Codice: Client che fa il JOIN

Quando il client riceve istruzione di fare il JOIN, esegue:

```
284 public void join(String ip, int porta) throws Exception
285 {
286     String mossaUscita = "";          //mossa da inviare all'altro client
287     String mossaEntrata = "";         //mossa ricevuta dall'altro client
288     String r = "";                    //Stringa contentente la riga
289     String c = "";                    //Stringa contentente la colonna
290     int riga;                         //intero contentente la riga
291     int colonna;                      //intero contentente la colonna
292
293     Thread.sleep(3000);     //Attende 3 secondi per dare tempo all'host di essere pronto (non necessario,
294
295     try
296     {
297         /*Inizializzazione della connessione e dei vari Stream di comunicazione*/
298         Socket connessione = new Socket (ip, porta);
299         DataInputStream inFromPeer = new DataInputStream(connessione.getInputStream());
300         DataOutputStream outToPeer = new DataOutputStream(connessione.getOutputStream());
301         BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
302
303         /*Avvisa l'utente che sara' il primo ad avere la mossa*/
304         System.out.println("Stai giocando come Crocetta {X} ed e' il tuo turno.");
305         setToEmpty();//azzera la matrice
306         printTable();//stampa la griglia (vuota)
307         ... altre istruzioni ...
```

e poi...

```
377     ... altre istruzioni ...
378     /*Chiude la connessione e gli Stream*/
379     inFromPeer.close();
380     outToPeer.close();
381     connessione.close();
382 }
```

Screenshot – Lato Client – pag 1/2

Screenshot – Lato Client – pag 2/2

```
Stato tabella:
+---+---+---+
|   |   |   |
+---+---+---+
|   | O |   |
+---+---+---+
| X |   |   |
+---+---+---+

E' il turno dell'avversario! Attendi...

Stato tabella:
+---+---+---+
|   |   | X |
+---+---+---+
|   | O |   |
+---+---+---+
| X |   |   |
+---+---+---+

E' il tuo turno! Fai la tua mossa...
q

Stato tabella:
+---+---+---+
| O |   | X |
+---+---+---+
|   | O |   |
+---+---+---+
| X |   |   |
+---+---+---+

E' il turno dell'avversario! Attendi...

Stato tabella:
+---+---+---+
| O |   | X |
+---+---+---+
|   | O |   |
+---+---+---+
| X |   | X |
+---+---+---+

E' il tuo turno! Fai la tua mossa...
d
```

```
Stato tabella:
+---+---+---+
| O |   | X |
+---+---+---+
|   | O | O |
+---+---+---+
| X |   | X |
+---+---+---+

E' il turno dell'avversario! Attendi...

Stato tabella:
+---+---+---+
| O |   | X |
+---+---+---+
|   | O | O |
+---+---+---+
| X | X | X |
+---+---+---+

Partita finita! Ha vinto l'avversario!


Scegli un'operazione da effettuare:

N. Nuova Partita
I. Istruzioni
Q. Esci

q
```

Screenshot – Lato Server