****

**פרויקט גמר**

**למילוי חלקי של הדרישות לקבלת תואר**

**הנדסאי הנדסת תוכנה**

**בהתמחות: מחשבים**

נושא הפרויקט: בחירת שחקן מצטיין

שם הסטודנט: איתי שורץ

העבודה בוצעה בהנחיית: אודי מלכה, מוטי פניקשיוולי



ה צ ה ר ת ס ט ו ד נ ט

אני הסטודנט איתי יונתן שורץ מס' ת.ז 207034778 החתום מטה, מצהיר בזאת שכל עבודת הגמר/ הפרוייקט המוגש/ת בחוברת זו היינו/ה פרי עבודתי בלבד.

על בסיס הנחייתו של המנחה ותוך הסתמכות על מקורות הידע והמידע האחרים המצויים בביליוגרפיה המובאת בחוברת זאת.

אני מודע לאחריות שהנני מקבל על עצמי ע"י חתימתי על הצהרה זו שכל הנאמר בה הינו אמת ורק אמת.

חתימת מגיש החוברת : \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

אישור המנחה :

הנני מאשר הגשת החוברת להערכה \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

תוכן עניינים

מבוא............................................................................... עמ' 4

סביבת פיתוח.................................................................... עמ' 4

דרישות מערכת מהמשתמש................................................ עמ' 4

UML.............................................................................. עמ' 4

מדריך למשתמש............................................................... עמ' 5-6

בעיות שהסנכרון פותר........................................................ עמ' 6

פירוט מחלקות.................................................................. עמ' 7-8

קטעי קוד......................................................................... עמ' 8-15

מבוא

בחירת שחקן mvp בליגה.

לכל לקוח יש 2 אופציות:

1) בחר שחקן, בוחרים שחקן מצטיין מבין האופציות שיש. לכל לקוח יש רק אופצית בחירה אחת, לאחר שהוא בוחר ינעל לו הכפתור של "בחר שחקן"

2) הצג נתונים, לאחר בחירה של כמה שחקנים, כפתור "הצג נתונים" יציג את השחקן שהכי הרבה לקוחות בחרו בו לשחקן המצטיין.

מטרת הפרויקט הייתה מימוש מערכת קליינט-שרת ומניעת קיפאון.

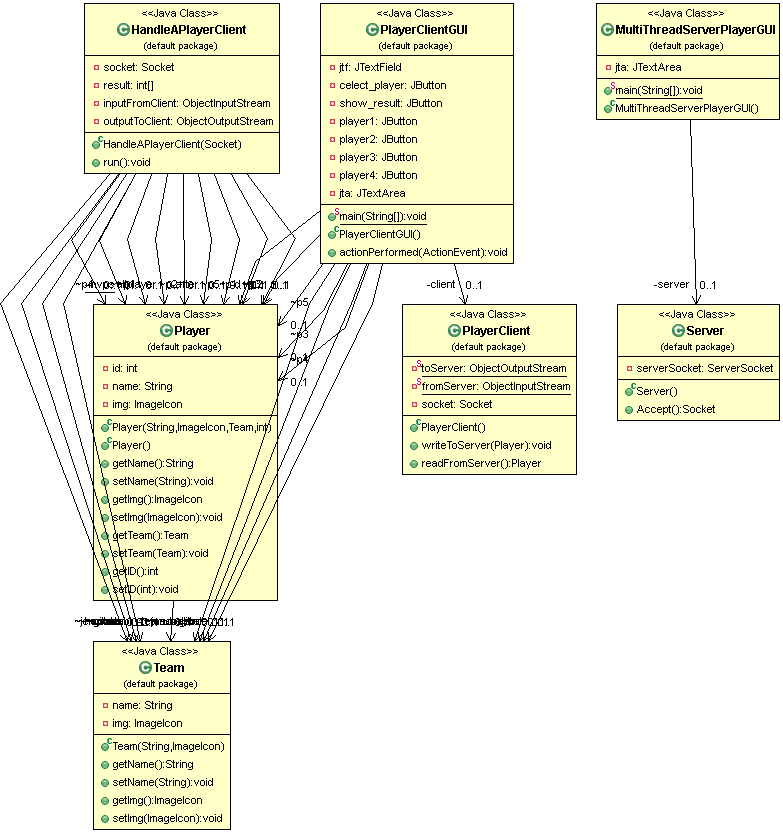
סביבת פיתוח

* Windows 7/10
* Eclipse Java

דרישות מערכת מהמשתמש

* Windows 7/10
* Java

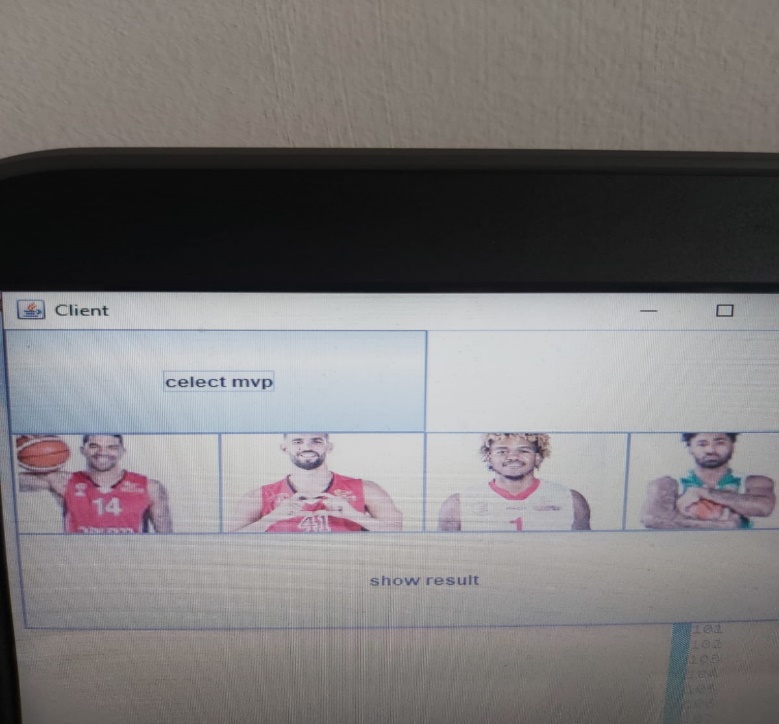
UML



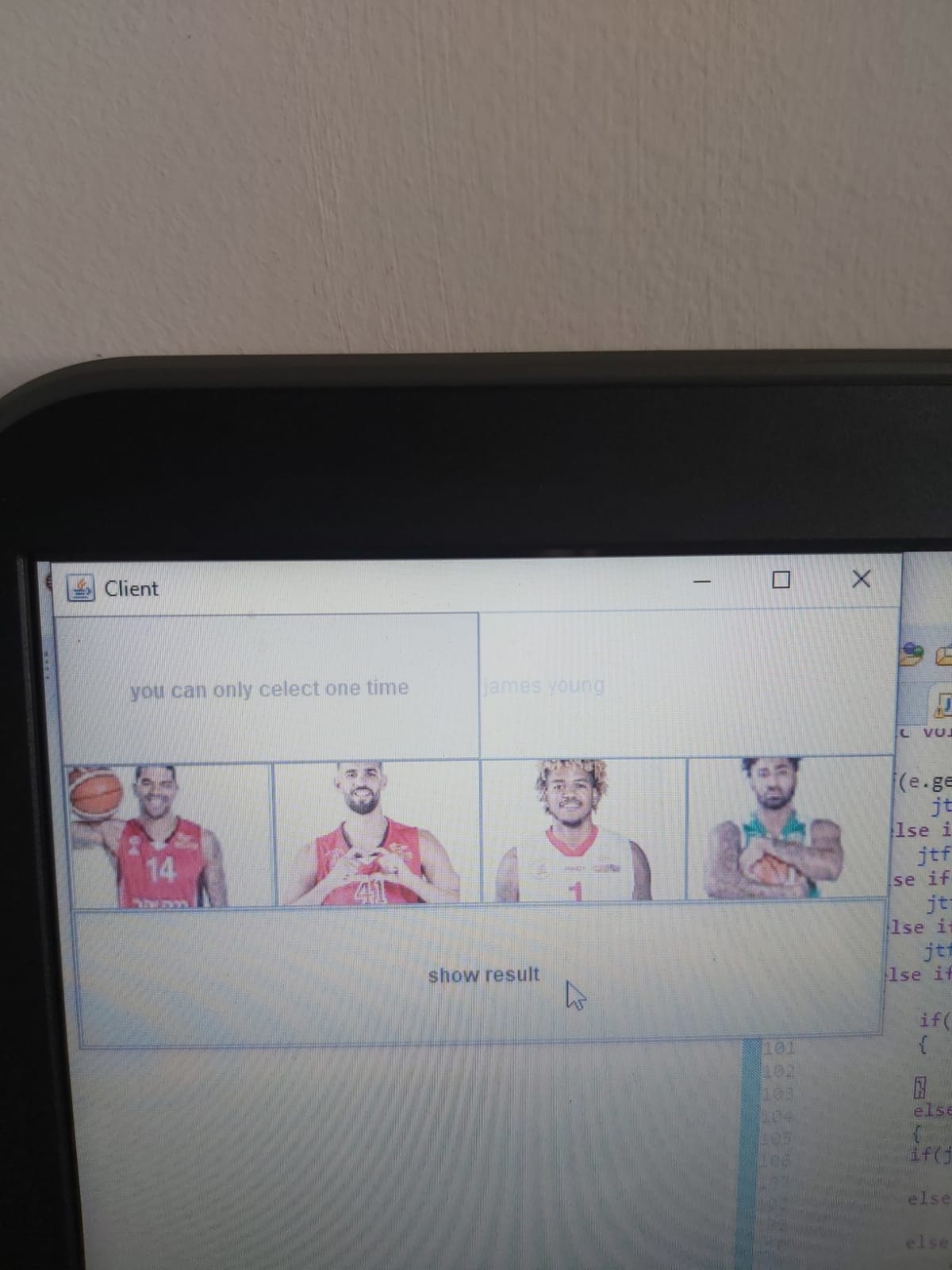
מדריך למשתמש:

מסך אחד של לקוח ובו:

אופציה לבחור את השחקן המצטיין, כפתור "בחר שחקן" וכפתור "הצג תוצאה".

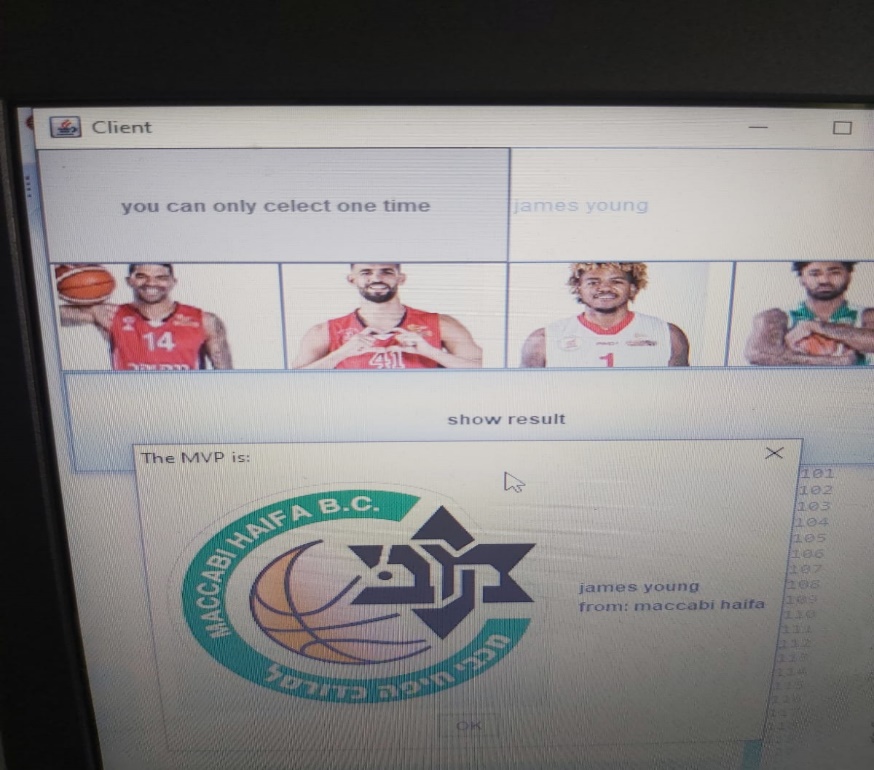


כשלוחצים בחר שחקן, מעבירים לשרת אובייקט של שחקן ומכניסים אותו לוקטור שחקנים.



ככה עושים עם כל שחקן שנבחר.

כשלוחצים על "הצג תוצאה" נעבור על וקטור השחקנים בשרת ונראה איזה שחקן מופיע הכי הרבה פעמים. אחרי שנגלה נציג את השחקן במסך של הלקוח+ הקבוצה שלו.



בעיות שהסנכרון פותר:

1)לקוח אחד לוחץ על "הצג תוצאה" ולקוח אחר לוחץ על "בחר שחקן" ואז " הצג נתונים"

צריך לדאוג לסנכרון שזה יציג ללקוח הראשון את התוצאות בלי הכנסה של השחקן שהלקוח השני בחר.

אחרי שהוא יציג את התוצאות ללקוח הראשון, הוא יתפנה לשחקן השני, יכניס את השחקן שהוא בחר ואז יחשב לו את הנתונים.

לכן אני משתמש פה בוקטור( שהוא synchronized) ולא במערך(שהוא לא synchronized).

2) מבצעים בדיקת תקינות, עבור כל איבר שנכנס לוקטור השחקנים, בודקים שהid שלו חיובי. בודקים זאת ע"י הוצאה של האיבר מהוקטור ורק אם הוא חיובי מחזירים אותו בחזרה. אם לא נעשה את זה בסנכרון האיבר שנוציא בפעולה הזאת לא בהכרח יכנס למקום שאליו הוא נכנס בהתחלה בוקטור, עצם הסנכרון אנחנו דואגים שאם יכנסו עוד נתונים בזמן שהוצאנו ובדקנו את הוקטור הם לא יכנסו ישירות לוקטור אלא יחכו שהתהליך יסתיים, שהאיבר שהוצאנו יכנס במחזרה למקומו ורק אז יכנסו לוקטור.

פירוט מחלקות:

מחלקת PlayerClient

מחלקה זו אחראית על שליחת מידע מהקליינט ושליחת מידע לקליינט.

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה אשר אחראית על שליחת שחקן מצטיין לסרבר | **public** **void** writeToServer(Player d) |
| פונקציה אשר אחראית על קבלת שחקן מצטיין מהסרבר | **public** Player readFromServer() |

מחלקת PlayerClientGui

מחלקה גרפית של המשתמש עבור התוכנה.

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה שמפעילים אותו לאחר לחיצה על אחד הכפתורים של בחירת שחקן או הצגת שחקן מצטיין והיא אחראית לשלוח את השחקן הנבחר לסרבר או להציג את השחקן המצטיין כעת | **public** **void** actionPerformed(ActionEvent e) |

מחלקת HandleAPlayerClient

מחלקה אשר מטפלת במידע שהקליינט שולח לסרבר.

|  |  |
| --- | --- |
| פירוט | פונקציה |
| פונקציה אשר מתקבלת ממשק התהליכון. פונקציה זו תמיד מאזינה לבקשות הקליינט.  בנוסף הפונקציה אחראית להכניס כל בחירה של משתמש בשחקן מצטיין לוקטור של שחקנים וכאשר אחד המשתמשים מבקש להציג את השחקן המצטיין הפונקציה דואגת לגלות מי זה. | **public** **void** run() |

מחלקת Server

שרת עבור התוכנה

|  |  |
| --- | --- |
| פירוט | פונקציה |
| ממתין לחיבור שיבוצע לסוקט ומקבל אותו. | **public** Socket Accept() |

מחלקת MultiThreadServerPlayerGUI

מחלקה גרפית של השרת היוצרת תהליך לשליחת הודעות ממשתמשים.

מחלקת: Player

מחלקה המייצגת שחקן ומכילה את כל הפרטים הנדרשים על שחקן.

מחלקת Team

מחלקה המייצגת קבוצה ומכילה את כל הפרטים הנדרשים על קבוצה.

קטעי קוד

מחלקת PlayerClient

**class** PlayerClient {

// IO streams

**private** **static** ObjectOutputStream *toServer*;

**private** **static** ObjectInputStream *fromServer*;

**private** Socket socket;

**public** PlayerClient(){

**try** {

// Create a socket to connect to the server

socket = **new** Socket("localhost", 8000);

// Create an output stream to send data

// to the server

*toServer* = **new** ObjectOutputStream(socket.getOutputStream());

// Create an input stream to receive data

// from the server

*fromServer* = **new** ObjectInputStream(socket.getInputStream());

}

**catch** (IOException ex) { }

}

**public** **void** writeToServer(Player d){

**try** {

*toServer*.writeObject(d);

*toServer*.flush();

} **catch** (IOException e) {e.printStackTrace(); }

}

**public** Player readFromServer(){

**try** {

**try** {

**return** (Player) *fromServer*.readObject();

} **catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

} **catch** (IOException e) {e.printStackTrace(); }

**return** **null**;

}

}

מחלקת PlayerClientGui

**public** **class** PlayerClientGUI **extends** JFrame **implements** ActionListener{

**private** JTextField jtf = **new** JTextField();

**private** JButton celect\_player;

**private** JButton show\_result ;

**private** JButton player1;

**private** JButton player2;

**private** JButton player3;

**private** JButton player4;

**private** JTextArea jta = **new** JTextArea();

**private** PlayerClient client = **new** PlayerClient();

Team gilboa= **new** Team("gilboa", **new** ImageIcon("img/gilboa.jpg"));

Team hapoal= **new** Team("hapoal tel aviv", **new** ImageIcon("img/hapoal.png"));

Team jerusalem= **new** Team("jerusalem", **new** ImageIcon("img/jerusalem.png"));

Team maccabi= **new** Team("maccabi haifa", **new** ImageIcon("img/maccabi.png"));

Player p1= **new** Player("james feldin", **new** ImageIcon("img/feldin.jpg"), jerusalem,1);

Player p2= **new** Player("tomer ginat", **new** ImageIcon("img/ginat.jpg"), hapoal,2);

Player p3= **new** Player("jastin tilman", **new** ImageIcon("img/tilman.jpg"), gilboa,3);

Player p4= **new** Player("james young", **new** ImageIcon("img/young.jpg"), maccabi,4);

Player p5= **new** Player("", **null**, **null**,0);//for "show result"

**public** **static** **void** main(String[] args) {

**new** PlayerClientGUI();

}

**public** PlayerClientGUI() {

celect\_player= **new** JButton("celect mvp");

show\_result = **new** JButton("show result");

player1 = **new** JButton(**new** ImageIcon("img/feldin.jpg"));

player2 = **new** JButton(**new** ImageIcon("img/ginat.jpg"));

player3 = **new** JButton(**new** ImageIcon("img/tilman.jpg"));

player4 = **new** JButton(**new** ImageIcon("img/young.jpg"));

player1.setBackground(Color.***WHITE***);

player2.setBackground(Color.***WHITE***);

player3.setBackground(Color.***WHITE***);

player4.setBackground(Color.***WHITE***);

player1.addActionListener( **this**);

player2.addActionListener( **this**);

player3.addActionListener( **this**);

player4.addActionListener( **this**);

celect\_player.addActionListener(**this**);

show\_result.addActionListener( **this**);

JPanel p = **new** JPanel(**new** GridLayout(3, 4));

JPanel p1 = **new** JPanel();

p1.setLayout(**new** GridLayout(1, 2));

p1.add(celect\_player,BorderLayout.***WEST***);

p1.add(jtf, BorderLayout.***CENTER***);

JPanel p2 = **new** JPanel();

p2.setLayout(**new** GridLayout(1, 4));

p2.add(player1);

p2.add(player2);

p2.add(player3);

p2.add(player4);

JPanel p3 = **new** JPanel();

p3.setLayout(**new** GridLayout(1, 1));

p3.add(show\_result);

p.add(p1);

p.add(p2);

p.add(p3);

add(p);

jtf.setEnabled(**false**);

setTitle("Client");

setSize(500, 300); setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setVisible(**true**);

}

**public** **void** actionPerformed(ActionEvent e) {

**if**(e.getSource()==player1)

jtf.setText(p1.getName());

**else** **if**(e.getSource()==player2)

jtf.setText(p2.getName());

**else** **if**(e.getSource()==player3)

jtf.setText(p3.getName());

**else** **if**(e.getSource()==player4)

jtf.setText(p4.getName());

**else** **if**(e.getSource()== celect\_player)

{

**if**(jtf.getText().trim().equals(""))

{

}

**else**

{

**if**(jtf.getText().trim().equals(p1.getName()))

client.writeToServer(p1);

**else** **if**(jtf.getText().trim().equals(p2.getName()))

client.writeToServer(p2);

**else** **if**(jtf.getText().trim().equals(p3.getName()))

client.writeToServer(p3);

**else**

client.writeToServer(p4);

celect\_player.setEnabled(**false**);

celect\_player.setText("you can only celect one time");

}

}

**else**

{

client.writeToServer(p5);

Player after = client.readFromServer();

**if**(after.getID()==0)

JOptionPane.*showMessageDialog*(**null**, "there is draw!"+"\n"+"kepp voting!!!: " , "The MVP is:", JOptionPane.***ERROR\_MESSAGE***);

**else**

JOptionPane.*showMessageDialog*(**null**, after.getName()+"\n"+"from: "+after.getTeam().getName() , "The MVP is:", JOptionPane.***INFORMATION\_MESSAGE*** ,after.getTeam().getImg());

}

}

}

מחלקת HandleAPlayerClient

**class** HandleAPlayerClient **implements** Runnable {

**private** Socket socket; // A connected socket

**private** **int** result[];

**private** ObjectInputStream inputFromClient;

**private** ObjectOutputStream outputToClient;

**private** Player d;

**private** Player after;

**private** **static** Vector<Player> *mvps*=**new** Vector<Player>();

Team gilboa= **new** Team("gilboa", **new** ImageIcon("img/gilboa.jpg"));

Team hapoal= **new** Team("hapoal tel aviv", **new** ImageIcon("img/hapoal.png"));

Team jerusalem= **new** Team("jerusalem", **new** ImageIcon("img/jerusalem.png"));

Team maccabi= **new** Team("maccabi haifa", **new** ImageIcon("img/maccabi.png"));

Player p1= **new** Player("james feldin", **new** ImageIcon("img/feldin.jpg"), jerusalem,1);

Player p2= **new** Player("tomer ginat", **new** ImageIcon("img/ginat.jpg"), hapoal,2);

Player p3= **new** Player("jastin tilman", **new** ImageIcon("img/tilman.jpg"), gilboa,3);

Player p4= **new** Player("james young", **new** ImageIcon("img/young.jpg"), maccabi,4);

Player p5= **new** Player("", **null**, **null**,0);//for draw

Player[] allplayer=**new** Player[] {p1, p2,p3,p4};

**public** HandleAPlayerClient(Socket socket) {

**this**.socket = socket;

**this**.result=**new** **int**[4];

**for**(**int** i=0; i<4; i++)

**this**.result[i]=0;

}

**public** **void** run() {

**try** {

// Create data input and output streams

outputToClient = **new** ObjectOutputStream(socket.getOutputStream());

inputFromClient = **new** ObjectInputStream(socket.getInputStream());

// Continuously serve the client

**while** (**true**) {

// Receive sentence from the client

**try** {

d = (Player) inputFromClient.readObject();

**if**(d.getID()==0)//show result

{

**this**.result[0]=0;**this**.result[1]=0;**this**.result[2]=0;**this**.result[3]=0;

**for**(Player a:*mvps*)

{

**if**(a.getID()==1)

**this**.result[0]++;

**else** **if**(a.getID()==2)

**this**.result[1]++;

**else** **if**(a.getID()==3)

**this**.result[2]++;

**else**

**this**.result[3]++;

}

**int** max\_index=0;

**for**(**int** j=0; j<4; j++)

{

**if**(result[j]>result[max\_index])

max\_index=j;

}

**boolean** isdraw= **false**;

**for**(**int** j=0; j<4; j++)

{

**if**( j!= max\_index &&result[j]==result[max\_index])

isdraw=**true**;

}

// Send revised sentence back to the client

**if**(isdraw)

outputToClient.writeObject(p5);

**else**

outputToClient.writeObject(allplayer[max\_index]);

}

**else**//add mvp

{

*mvps*.add(d);

Thread t=**new** Thread()

{

**public** **void** run() {

Player p;

//אם לא נעשה את זה בסכנרון האיבר שנוציא בפעולה הזאת לא בהרח יכנס למקום שאליו הוא נכנס בהתחלה בוקטור, עצם הסנכרון אנחנו דואגים שאם יכנסו עוד נתונים בזמן שהוצאנו ובדקנו את הוקטור הם לא יכנסוו ישירות לוקטור אלא יחכו שהתהליך יסתיים, שהוקטור שהוצאנו יכנס במחזרה למקומו ורק אז יכנסו לוקטור

**synchronized**(*mvps*) {

p= *mvps*.elementAt(*mvps*.size()-1);

*mvps*.removeElementAt(*mvps*.size()-1);

**try** {

//אפשר להגדיל את שמן ההשעיה כדי להמחיש שזה עובד

Thread.*sleep*(1000);

} **catch** (InterruptedException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

**if**(p.getID()>0)

*mvps*.add(p);

}// synchronized

}// run

};// thread

t.start();

} // else

//System.out.println(mvps.toString());

}

**catch** (ClassNotFoundException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

} **catch** (IOException e) {

System.***err***.println(e);

}

}

}

מחלקת Server

**class** Server {

**private** ServerSocket serverSocket;

**public** Server() {

**try** {

serverSocket = **new** ServerSocket(8000);

} **catch** (IOException e) {

e.printStackTrace();

}

}

**public** Socket Accept() {

**try** {

**return** serverSocket.accept();

} **catch** (IOException e) {

e.printStackTrace();

}

**return** **null**;

}

}

מחלקת MultiThreadServerPlayerGUI

**public** **class** MultiThreadServerPlayerGUI **extends** JFrame {

**private** JTextArea jta = **new** JTextArea();

**private** Server server = **new** Server();

**public** **static** **void** main(String[] args) {

**new** MultiThreadServerPlayerGUI();

}

**public** MultiThreadServerPlayerGUI() {

// Place revised sentence on the frame

setLayout(**new** BorderLayout());

add(**new** JScrollPane(jta), BorderLayout.***CENTER***);

setTitle("MultiThreadServer");

setSize(500, 300);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setVisible(**true**); // It is necessary to show the frame here!

jta.append("MultiThreadServer started at " + **new** Date() + '\n');

**int** clientNo = 1;

**while** (**true**) {

// Listen for a new connection request

Socket socket = server.Accept();

// Display the client number

jta.append("Starting thread for client " + clientNo + " at " + **new** Date() + '\n');

// Find the client's host name, and IP address

InetAddress inetAddress = socket.getInetAddress();

jta.append("Client " + clientNo + "'s host name is " + inetAddress.getHostName() + "\n");

jta.append("Client " + clientNo + "'s IP Address is " + inetAddress.getHostAddress() + "\n");

// Create a new task for the connection

Thread task = **new** Thread(**new** HandleAPlayerClient(socket));

task.start();

clientNo++;

}

}

}

מחלקת Team

**public** **class** Team **implements** Serializable{

**private** String name;

**private** ImageIcon img;

**public** Team(String name, ImageIcon img) {

**this**.name = name;

**this**.img = img;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** ImageIcon getImg() {

**return** img;

}

**public** **void** setImg(ImageIcon img) {

**this**.img = img;

{

מחלקת: Player

**public** **class** Player **implements** Serializable{

**private** **int** id;

**private** String name;

**private** ImageIcon img;

**private** Team team;

**public** Player(String name, ImageIcon img, Team team, **int** id) {

**this**.id=id;

**this**.name = name;

**this**.img = img;

**this**.team = team;

}

**public** Player() {

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** ImageIcon getImg() {

**return** img;

}

**public** **void** setImg(ImageIcon img) {

**this**.img = img;

}

**public** Team getTeam() {

**return** team;

}

**public** **void** setTeam(Team team) {

**this**.team = team;

}

**public** **int** getID() {

**return** id;

}

**public** **void** setID(**int** id) {

**this**.id = id;

}

}