This project involves the use of client-server paradigm where in the players sit on the client and the game controller is a distant server. This can be designed in various ways. We discuss the major options possible at the server side and the client side and highlight the pros and cons of the approaches:

Server side programming options:
- We decided to use servlet programming since the group members wanted to explore Java Servlets.

Client side programming options
1) HTML
   There isn’t much control on the client side if we use html since html is a non-dynamic language and we have to periodically refresh the page, which increases the communication overhead. Also time is a major constraint in the boogle game since each player is only given 3 minutes for the game. In such a case it would not be good to implement a refresh mechanism which itself might take 2-3 seconds on the best connection.

2) Java programs
   Our game needs to be web enabled i.e. it should run on any web browser. But Java programs cannot run on web browsers hence this particular option has to be ruled out.

3) Applets:
   Applets can be run on any Java enabled browser. Also once the browser window is closed the applet is destroyed. Each game instance could be an applet at the client and the user can end the game whenever he likes by closing the browser window. This was the option chosen by us because we wanted our application web enabled.

Now we discuss the design of the Concurrent Boogle Game according to the steps outlined in class. The commented code is presented at the last of this document.
Basic Design Steps:

Organization of Classes
**Step 1) Identifying the Abstract Data Types (ADTs)**

We start our design by outlining the various classes required for the Boogle Game.

**Classes:**

BoogleGameServer  
BoogleGame  
Players  
Project(BoogleGameApplet)  
Message  
RegisterRequestMessage  
GamePiecesMessage  
ResultAndScoresMessage  
StartGameMessage  
WordMessage  
WordResultMessage

**Description of the ADTs:**

**BoogleGameServer**

This is the main servlet that is first contacted by the user. Whenever a player contacts this servlet, the servlet checks if there is a game that is in a state of waiting for the players to join. If such a game is present then the new user will be added to this game. In case there is no waiting game a new game instance is created. It sends an html page in which the applet is embedded.

**BoogleGame**

This is the main boogle game. When this object is created it is put in waiting for players to join in. This game can be started in two ways:
1. When four players have joined in, the game will be automatically started.
2. When one of the players decides to start the game irrespective of the number of players in the game.

Once a game has started no new players can be added (no one can join a running game).

BoogleGame class would have to return the Port on which it is listening so that the Applet will start communicating on this port.

The BoogleGame communicates these messages to the applet.

1) StartGame command
2) Word supplied is valid or not
3) Scores

**Players**
This class is used for communicating with the applet and serve as link between the BoogleGame and the Applet.

**Message**
This is an abstract class which do the serialization and deserialization of the object streams. The objects that are serialized serve as the Messages between the applet and the Player. These classes are derived from the Message Class
1. RegisterRequestMessage
2. GamePiecesMessage
3. ResultAndScoresMessage
4. StartGameMessage
5. WordMessage
6. WordResultMessage

**BoogleGameApplet:**
This implements the user interface on the player side. We are using the applet on the player side. Once an applet is loaded it would open a new connection with the servlet and keeps communicating with the servlet throughout the game.

The following things need to be communicated:
1) Game start request (if the new player is the fourth in the game group then the game starts automatically and hence no need to start the game)
2) Word for validation and also for score calculation
3) Score (updated after each word entry)
4) Time remaining in the game (There are two ways if doing this we would start the timer at the start of the game and let the applet time out .The second way is let the timer be at the server and the startgame and endgame commands are both given by the server. If using the first method the client gets exactly 3 minutes to work on the gamepieces but the client can cheat by resetting it’s clock midway. The second option can be unfair if the network communication overhead is not the same for all the clients and differs over the time the game is played.)

The score calculation and the word validation are done at the server side to ensure security and fair grading.
Step 2) List the threads used in the program

Server Side

BoogleGameServer - the servlet engine generates these threads
BoogleGame – Individual game threads (one thread per game), These threads are generated by the BoogleGameServer.
Player – these are the threads created by the BoogleGame. These correspond to the applets that are communicating.
EndGameTask, StartGameTask – These are timer threads used in the BoogleGame thread and they have to interact with the BoogleGame thread.

Client Side

BoogleGameApplet – individual player (one thread per player or per applet). This is present on the client side.
UpdateTimerTask – this thread is created on the applet for updating the remaining time.
StartListening – We needed to create this thread because of some peculiar behavior of the Browsers. The browsers refused to paint the Applet until the start method was complete so we created this thread inside the start method.
Step 3) Synchronization

**BoogleGameServer:**

In this the access to the BoogleGame is synchronized on the BoogleGameServer.class this is accessed while adding new player to the game.

**BoogleGame:**

In this the access to vector of Players is synchronized. This can be accessed by the BogleGameServer thread for adding players, BoogleGame thread for sending new joinee messages and the Player thread for deleting itself.

There is a producer Consumer problem between the BoogleGame and the players where the Players are the Producers and the BoogleGame is the Consumer. The players produce the Scores and words and the BoogleGame has to send the Scores and words to all the players.

Synchronization is also required for access to the WordsAlreadyUsedVec, This vector is accessed by the Players and the BoogleGame itself while sending words.

**Players:**

We are synchronizing access to the DataOutputStream. The Players and the BoogleGame access this.
Waiting for an applet to join a game

Incoming request to join a game

Try to add the player to a game and get port no

Success?
  Yes
  No

Create a new BoogleGame and add the player to it and get port no

Create an HTML page having the following information for the applet:
  PortNo of the game, Unique tag, and the name supplied by the CGI script

Send page to applet
Wait for registration

Identify the player
Supply the socket to the object

Create the player thread start the thread

Check if registered players = 4

Check if game start req = true

Prepare the barrier

Interrupt the player threads

BoogleGame Thread Flow Diagram
Start Thread

GameStarted = false

Waiting for read

Is startGame request

Call requestStartGame in BoogleGame

Wait in a barrier to start game

Send game pieces to Applet

End Game = true

Read word entered sent by applet

Enter word in Game

Send Result of word to Applet

End Thread

Player thread Flow Diagram
**BoogleGameApplet Design**

We have used Applet on the client side. The index.html page sends a query to the servlet that in turn sends the applet back to the browser and is loaded at the player (client) end. The Applet is communicating with the servlet over the socket throughout the lifetime of the applet.

An applet has four methods in its lifetime:
- **init()** – Initializes the applet. This method is called before the applet starts loading. It is called only once in a lifetime of an applet.
- **start()** - Called after the applet is initialized and every time the page is reloaded. The page is only visible after this method has finished execution.
- **stop()** – called every time the applet is minimized
- **destroy()** – called only once when the browser window is closed

We have used Java swings in order to prepare our GUI. Implementing Action Listener and Key Listener made the task of sending the word to the servlet for authentication possible. Action Listener made the capture of click event possible and Key Listener made the keyboard hit known in the program.

Each applet is a new window for the user and hence we need not worry about any synchronization in BoogleGame Applet class.

The word authentication and score computation is computed at the server side for security and integrity of the game. The control of the time of the game is at the server side. The timer for display was at the client end and would ensure that no player gets more than 3 minutes for the game. The GUI counts down for 3 minutes before all the events on the user side are disabled.
Figure: The hierarchy of Components used in the Applet design

The above figure explains how the hierarchy of components used in the BoogleGameApplet. The Container of the game object was initialized to contentpane and thus the Applet could be displayed on the user browser window.

Kindly note the BoogleGameApplet class is Project class in the Demo code.
Step 4) Any obvious optimizations

1. We had indicated in our preliminary design that we would be coding a separate class for checking whether the word is there in the dictionary, but we abandoned that effort and used the hashtable class provided in the class library.
Game Plan:

The user opens the location http://sand.cise.ufl.edu:40309/test and hits enter. The login screen as shown in figure 1 appears.

Figure1. Login screen

The player now enters his name and hits the **Submit** button. By clicking the button, the user sends a register request to the server and in turn an applet is sent back by the server. The Applet screen at the player end is shown in figure 2.
The game has not still started. The User screen shows the names of players joined in the same game along with their default scores, which is initially zero. Each player has the option of starting a game immediately by hitting the **Start Game** button or waiting for maximum of 3 other players (the game is automatically started as soon as 4 players join a game).

The screen when the game is in progress is shown in figure 3. As soon as the game starts the game alphabets appear in the place of Xs and the timer starts.
counting down from 180 to 0.

![Figure 3: Screen while the game is in progress](image)

The player plays the game by entering the word in the textbox in front of **Enter your word here**: and hitting the keyboard ENTER or clicking the **Enter** button. Note that the **Enter** button is disabled when the text box for entering the word is blank, since blank words are not allowed. The player immediately gets to know about the latest word submitted by him in the Applet status bar. He is also able to see his score as well as his competitor's scores throughout the game and thus
able to play according to it. The correct words that have already been used in the game are added to the list showing **Words already used.** The game demands that each word be used only once in each game.

Exactly after 3 minutes the game ends and all the player control buttons are disabled. A Dialog screen pops up congratulating the player if he has won (see figure 4) else intimating the name of the player who has won (see figure 5) in case he has lost.

![Figure 4: Screen shot at player “govind” after the game has ended. The player “govind” has won this game](image)

Page 16 of 73
Figure 5: Screen shot at player “ninad” after the game has ended. The player “ninad” has lost this game

After the player hits the OK button. If the same player is interested in playing another game he just reloads the page and he again sees the page shown in figure 2 else he just closes the window.

A new player initiates a game by opening the location mentioned above and all the discussed steps are repeated.
Technicalities:

The BoogleGameServer servlet decides to start a new game when there are 4 players in the game. This Object keeps the information whether there is any game waiting for some more players to join in.

When the game is started a BoogleGame object is created that maintains the game and the subsequent interaction with the clients through the Players object. Players object is in contact with the BoogleGameApplet throughout the game. The BoogleGame is started even if there is a single client connected since there is an option in the applet to start the game if it wants. Else the game waits until there are 4 clients connected to the game.
Possible Corrections in the Code that was presented in the Demo:

1) We would like to apologize for using the word Boogle Game instead the Boggle Game throughout the project.
2) We allowed only 3 or more alphabet words in our game instances. Hence the players were not able to use 2 alphabet words.
3) The mechanism of displaying Winner in the game could have been better. We calculated the maximum score amongst all the players and displayed the corresponding name of the winner. Due to this, in the case of a tie game wherein more than 1 player has same scores, our game displayed the name of the player who was added to the game at the last was declared a winner.

Code used in the Demo
for (int i = 0 ; i < scr.length ; i++)
{
    if(scr[i]>=maxScore){
        maxScore = scr[i];
        maxName = names[i];
        maxTag = tags[i];
    }
}
if(maxTag == tag)
{
    JOptionPane.showMessageDialog(this.getContentPane(),"Your score is "+maxScore+". Congratulations !! You are the winner");
    showStatus("Winner !!!");
}
else
{
    JOptionPane.showMessageDialog(this.getContentPane(),"The winner is "+maxName);
    showStatus("Thanks for playing with us");
}

Improved code:
By using the code given below, all the winners would be shown that they have won the game. The changes are marked in bold.

Int myscore=0;
for (int i = 0 ; i < scr.length ; i++)
{
    if(scr[i]>=maxScore)
    {
        maxScore = scr[i];
    }
maxName = names[i];
maxTag = tags[i];
if (tags[i] == tag)
    {myScore = scr[i];}

if (myScore == maxScore)
{
    JOptionPane.showMessageDialog(this.getContentPane(), "Your score is "+maxScore+". Congratulations !! You are the winner");
    showStatus("Winner !!!");
}
else
{
    JOptionPane.showMessageDialog(this.getContentPane(), "The winner is "+maxName);
    showStatus("Thanks for playing with us");
}
Distribution of work:

The design, integration and testing was done by both but for implementation (programming)

Ninad: Worked on the Server side programming involving servlets.

Govind: Worked on the Client side programming involving applets.
public class Barrier{
    private final int N;
    static int n;
    static int ri;

    Barrier(int N){ //N is the number of threads to synchronize
        this.N = N ;
        n = 0;
        ri = 0;
    }

    public synchronized void barrier() throws InterruptedException {
        int r = ri + 1;
        n++;
        if(n!=N){
            while(ri!=r) wait();
        }
        else if(n == N) {
            ri = r;
            n=0;
            notifyAll();
        }
    }

Explanation:

Once a player enters a game he is put into the barrier where he waits until the other players join the game.
import java.util.*;

/* This class is implemented as an Immutable class since we do not see the need to modify this class once this has been constructed */

public class Dice{

    // These are given characters from which the game instance is created
    char [][]letters = {
        {'w','t','a','t','o','o'},
        {'t','s','t','y','i','d'},
        {'r','y','e','v','l','d'},
        {'a','b','b','j','o','o'},
        {'m','u','i','o','t','c'},
        {'e','y','t','l','r','t'},
        {'s','u','e','e','n','i'},
        {'p','s','a','f','f','k'},
        {'w','r','e','t','v','h'},
        {'s','s','o','i','e','t'},
        {'r','l','r','e','x','d'},
        {'n','l','n','h','r','z'},
        {'e','g','n','w','e','h'},
        {'p','s','h','a','o','c'},
        {'e','g','n','e','a','a'},
        {'n','m','h','u','i','q'}
    };
    Die [] die;
    java.security.SecureRandom rnd;

    /* this constructor initializes all the individual dies and stores it */
    Dice(){

        /* we decided to use java.security.SecureRandom since it guarantees secure random number */
        rnd = new java.security.SecureRandom();

        die = new Die[16];
        for(int i = 0 ; i < die.length;i++){
            die[i] = new Die(letters[i]);
        }
    }
/* This method generates the 16 alphabets used towards the game instance. Every time this method is called new characters from the die object are generated and sent back to the calling method */

public String shakePieces(){

    StringBuffer buf = new StringBuffer();
    Vector randP = new Vector();
    for(int i = 0 ; i < 16 ; i++){
        randP.add(new Character(die[i].getRandomChar()));
    }
    int cnt = 15;
    for(int i = 0 ; i < 16 ; i++){
        int r = rnd.nextInt();
        if(r<0)
            r = r*(-1);
        if( cnt == 0)
            r = 0;
        else
            r = r%cnt;
        buf.append(((Character)randP.remove(r)).charValue());
        cnt--;
    }
    return buf.toString();
}
}
Die.java

import java.util.*;

/* This is an Immutable class since we would not be changing the object once we have created the class */

public class Die{

    char [] letters;
    java.security.SecureRandom rnd;

    /* In this constructor we just initialize the 6 faces of the die*/
    Die(char []arr){
        rnd = new java.security.SecureRandom();
        letters = arr;
    }

    /* This method simulates the throw of a die */
    char getRandomChar(){
        int r = rnd.nextInt();
        if(r<0)
            r = r*(-1);
        r = r%6;
        return letters[r];
    }
}

Message.java

import java.io.*;
//This class is used for generating the message

abstract class Message implements Serializable{

    char com;

    public byte [] encodeMessage() throws IOException{

        ByteArrayOutputStream baos = new 
        ByteArrayOutputStream();
        ObjectOutputStream oos = new
        ObjectOutputStream(baos);
        oos.writeObject(this);
        return baos.toByteArray();
    }

    static Object createObject(byte[] data) throws 
    IOException,ClassNotFoundException{

        ByteArrayInputStream bais = new
        ByteArrayInputStream(data);
        ObjectInputStream ois = new
        ObjectInputStream(bais);
        Object myObject = ois.readObject();
        return myObject;
    }

    char getTypeOfMessage(){

        return com;
    }
}

Explanation:
This is an immutable abstract class, the rest of the
Message classes are derived from this class
GamePiecesMessage.java

import java.io.*;

public class GamePiecesMessage extends Message implements Serializable{
    char [] pieces;
    GamePiecesMessage(char [] arr){
        com = 'g';
        pieces = arr;
    }
    public char [] getPieces(){
        return pieces;
    }
    /*This was generated to test the standalone code
    public static void main(String [] arg){
        String str = new String("abcdefg");
        GamePiecesMessage st = new GamePiecesMessage(str.toCharArray());
        try{
            System.out.println(st.encodeMessage());
            byte [] arr = st.encodeMessage();
            GamePiecesMessage st2 = (GamePiecesMessage) Message.createObject(arr);
            System.out.println("message type is "+st.getTypeOfMessage());
            System.out.println("message pieces is "+st.getPieces()[0]);
            }catch(Exception e){
                e.printStackTrace();
            }
    */
}

Explanation:
The player to send the game pieces to the applet uses this message.
ResultAndScoresMessage.java

```java
import java.io.*;

public class ResultAndScoresMessage extends Message implements Serializable{
    int numOfPlayers;
    String [] names;
    int [] scores;
    int [] tag;

    ResultAndScoresMessage(char type,int no,String []names,int []scores,int []tag){
        com = type;
        numOfPlayers = no;
        this.names = names;
        this.scores = scores;
        this.tag = tag;
    }

    public String [] getNames(){
        return names;
    }

    public int []getTag(){
        return tag;
    }

    public int []getScores(){
        return scores;
    }

    public void updateScores(int []scores){
        this.scores = scores;
    }

    public void updateType(char type){
        com = type;
    }
}
```

**Explanation:**
This message is sent by the BoogleGame to all the applets. This message also embeds messages like player join game, result message, end Game message.
StartGameMessage.java

    import java.io.*;

    public class StartGameMessage extends Message implements Serializable{

        StartGameMessage(){
            com = 'S';
        }
    }

Explanation:

This is the message used by the applet while requesting start of game.
import java.io.*;

public class WordMessage extends Message implements Serializable{
    String word;

    WordMessage(String word){
        com = 'w';
        this.word = word;
    }

    public void updateWord(String w){
        this.word = word;
    }

    public String getWord(){
        return word;
    }
}

Explanation:
This message is used by the Applet to send the entered words to the player and also by the BoogleGame to send the words already entered to the applets.
import java.io.*;

public class WordResultMessage extends Message implements Serializable{
    int res;

    WordResultMessage(int res){
        com = 'q';
        this.res = res;
    }

    public int getResult(){
        return res;
    }
}

Explaination:
The player to let the Applet know about the result of the word it entered uses this message.
RegisterRequestMessage.java

    import java.io.*;

    public class RegisterRequestMessage extends Message implements Serializable{

        String name;
        int tag;

        RegisterRequestMessage(String name, int tag){

            com = 'r';
            this.name = name;
            this.tag = tag;
        }

        public String getName(){
            return name;
        }

        public int getTag(){
            return tag;
        }

    }

Explanation:
This message is used by the applet for registering with the game.
// Importing the standard library files.
// Using AWT, AWT Events, Swings and Swing Events for GUI.
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.event.*;
import javax.swing.*;

// Importing java.net for Socket programming
import java.net.*;

// Importing java.io for File read and write
import java.io.*;

// Importing java. Util.Timer for timer implementation
// The timer is useful to refresh the time remaining field
// each second during run // // time
import java.util.Timer;
import java.util.TimerTask;

// encapsulating the Project class in Japplet and also
// implements the Action //Listener and Key Listener interface
public class Project extends JApplet implements
ActionListener,KeyListener
{
    // Buttons in the GUI
    // This button is used if the player needs to initiate the
game instead of waiting for the joining of the four players
    JButton bstartgame = new JButton("Start Game");
    // The word entered in the Text Entry text box is accepted
    // by the game by pressing Enter
    JButton benter=new JButton("Enter");
    // This button is used to clear the Text Entry text box
    JButton bclrent=new JButton("Clear entry");

    // Lists in the GUI
    // This list shows the players in the game along with their
    // scores in name --- score // format
    List ls1 = new List(4);
    // This list displays the correct words entered by the
    // players in a particular game // instance
    List ls2 = new List(4);

    // TextFields in the GUI
The player types in his words in this text box
JTextField tentry=new JTextField(35);

//Labels for displaying the game alphabets in the 4x4 grid
JLabel pieces[];
JLabel l12;
JLabel l2,l22;

//Top Level Panel in the GUI
Jframe mainfrm;

//Frame only for the dialogs
//Initializing the strings
String timerem="0";
String latscore="0";

//Variables required for connecting to the server
int port ;
int tag ;
Socket socket;
String name;
boolean gameStarted;
boolean gameEnd;
DataOutputStream out;
DataInputStream in;

//Timer variables for updating the time
int timeremi;
Timer timer;

// The main panel of the Applet
Container contentpane;

// Initialization of the applet. Creates User Interface.
public void init()
{
    //Getting parameters from the HTML page
    port = Integer.parseInt(getParameter("port"));
    tag = Integer.parseInt(getParameter("uniquetag"));
    name = getParameter("user");
    System.out.println("user = "+name+ " port = " + port + 
" tag " + tag);
    System.out.println("host "+getCodeBase().getHost());

    // The hierarchy of various components is shown in the
following diagram
contentpane=getContentPane();
   // There is a main panel in which other five panels are
added. This is mainly for the placement of the user
interface.
   Box mainbox=new Box(BoxLayout.Y_AXIS);

   // this panel shows the header of the Boogle Game
   JPanel phead=new JPanel();

   // First sub panel contains the header.
   JLabel lhead=new JLabel("Hi "+name+", Welcome to
Boggle Game",JLabel.CENTER);
lhead.setFont(new Font("Verdana",Font.BOLD,20));
phead.add(lhead);
   mainbox.add(phead);

   // Second sub panel
   // This panel covers the start button, Time remaining
label and the Score button
   JPanel p2=new JPanel();
   FlowLayout f2=new FlowLayout();
   JLabel l1=new JLabel("Time Remaining :");
l12=new JLabel(timerem);
l2=new JLabel("Score :");
l22=new JLabel(latscore);
p2.setLayout(f2);
p2.add(bstartgame);
p2.add(new JLabel("            "));p2.add(l1);
p2.add(l12);
p2.add(new JLabel("            "));p2.add(l2);
p2.add(l22);
   mainbox.add(p2);

   // Third sub panel
   // There are in turn 3 panels in the third sub panel
   // First panel of the third sub panel in turn has two
   // sub panels one for the text and other generates the grid
   // for display of the alphabets.
   // This panel in general shows the grid of alphabets,
Group members and scores and also the words already used.
   JPanel p3=new JPanel();
   GridLayout g3=new GridLayout(1,3);
p3.setLayout(g3);
JPanel p31 = new JPanel();
GridLayout g31 = new GridLayout(2, 1);
p31.setLayout(g31);
p31.add(new JLabel("Find words in", JLabel.LEFT));

JPanel p312 = new JPanel();
GridLayout g312 = new GridLayout(4, 4, 2, 2);
p312.setLayout(g312);
pieces = new JLabel[16];
for (int i = 0; i < 16; i++)
{
    pieces[i] = new JLabel("X", JLabel.CENTER);
    p312.add(pieces[i]);
}
p31.add(p312);
p3.add(p31);

// Second sub sub panel of the third sub panel
JPanel p32 = new JPanel();
GridLayout g32 = new GridLayout(2, 1);
p32.setLayout(g32);
p32.add(new JLabel("Group members ---
scores", JLabel.CENTER));
p32.add(ls1);
p3.add(p32);

// Third sub sub panel of the third sub panel
JPanel p33 = new JPanel();
GridLayout g33 = new GridLayout(2, 1);
p33.setLayout(g33);
p33.add(new JLabel("Words already
used", JLabel.CENTER));
p33.add(ls2);
p3.add(p33);
mainbox.add(p3);

// Fourth sub panel
// This panel covers the text entry by the player
JPanel p4 = new JPanel();
FlowLayout fl4 = new FlowLayout(FlowLayout.LEFT);
JLabel l3 = new JLabel("Enter your word here:");
p4.add(l3);
p4.add(tentry);
mainbox.add(p4);

// Fifth sub panel
// This panel contains the ENTER, CLEAR and EXIT button
JPanel p5 = new JPanel();
p5.add(benter);
p5.add(new JLabel(" "));  
p5.add(bclrent);
mainbox.add(p5);

// Adding the main panel to the present object
contentpane.add(mainbox);

// All the components are added to the container
contentpane and present ContentPane is set to contentpane.
setContentPane(contentpane);

// Adding the Action Listeners
bstartgame.addActionListener(this);
benter.addActionListener(this);
bclrent.addActionListener(this);
bexit.addActionListener(this);

// Initially the Text Box, ENTER, and CLEAR button are
disabled
benter.setEnabled(false);
bclrent.setEnabled(false);
tentry.setEnabled(false);

// Adding Key Listeners
tentry.addKeyListener(this);

gameStarted = false;

try {
    socket = new Socket(getCodeBase().getHost(), port);
    out = new DataOutputStream(socket.getOutputStream());
    RegisterRequestMessage reg = new RegisterRequestMessage(name, tag);
    System.out.println("type of message " + reg.getTypeOfMessage());
    out.write(reg.encodeMessage(), 0, reg.encodeMessage().length);
    System.out.println("wrote " + reg.encodeMessage().length);
    System.out.println("sent a register message to the")
In this method a new thread is started so that the Applet is displayed on the browser. It is found that the Applet is displayed only after the start method finishes and there has to be a while(true) loop for the operations of the socket.

```java
public void start()
{
  Runnable runW = new Runnable()
  {
    public void run(){startListening();}
  };
  Thread t = new Thread(runW);
  t.start();
}
```

```java
public void startListening()
{
  byte []data ;
  setVisible(true);  
data = new byte[1000];
  GamePiecesMessage gp = null;
  int dataCount;
  int bytesUsed = 0;
  while(true){
    try{
      if(in == null)
      {
        in = new DataInputStream(socket.getInputStream());
      }
      System.out.println("now before read ");
      data = new byte[1000];
      dataCount = in.read(data,0,999);
      if(dataCount < 1){
        System.out.println("got an error in read ");
        return;
      }
    }
  
```
while(dataCount > bytesUsed){
    System.out.println("in the while loop " +
    dataCount + " " + bytesUsed);
    Message msg =
(Message)Message.createObject(data);
    bytesUsed = msg.encodeMessage().length;
    System.out.println(name + "the message type
is "+ msg.getTypeOfMessage());
    // Getting the game pieces
    if(msg.getTypeOfMessage() == 'g') {
        gp = (GamePiecesMessage)
        Message.createObject(data);
        benter.setEnabled(true);
        bclrent.setEnabled(true);
        bexit.setEnabled(true);
        tentry.setEnabled(true);
        tentry.requestFocus();
        bstartgame.setEnabled(false);
        break;
    } else if (msg.getTypeOfMessage() == 't') {
        ResultAndScoresMessage res =
        (ResultAndScoresMessage) Message.createObject(data);
        String[] names = res.getNames();
        int[] scr = res.getScores();
        int[] tags = res.getTag();
        ls1.removeAll();
        // Removing the earlier instances and putting in the new
        player scores
        for(int i = 0 ; i < scr.length ; i++) {
            System.out.println(names[i] + " - " +
            scr[i]);
            if(tags[i] == tag){
                l22.setText(scr[i]+" ");
            }
            // Updating the names and the scores of each players
            ls1.add(names[i]+ " ---- " +scr[i]);
        }
    }
    byte[] dataBack = new byte[dataCount -
    bytesUsed];
    for(int i = bytesUsed ; i < dataCount ; i++){
        dataBack[i - bytesUsed] = data[i];
    }
    dataCount = dataCount - bytesUsed;
    bytesUsed = 0;
    data = dataBack;
/* check whether we have game pieces */
System.out.println(" out of the loop");
if(gp!=null)
    break;
}catch(IOException e){
e.printStackTrace();
break;
}catch(ClassNotFoundException e ){
e.printStackTrace();
break;
}
System.out.println(String.copyValueOf(gp.getPieces()))
;
String temp;
for(int i=0;i<gp.getPieces().length;i++)
{
    temp=String.valueOf(gp.getPieces()[i]);
    // The original game pieces have QU and do not have Q. Hence when we are //encountering a QU we interpret it as Q at the server end and again QU at the //client end
    if (temp.equals("q")==true)
        temp="qu";
    // Characters converted to upper case for display
    temp=temp.toUpperCase();
    pieces[i].setText(temp);
}
// at this point we have started the game ... so start the timer here */
timeremi=180;
// Timer task is started after game starts. The task of this timer is defined in Class //UpdateTask
<3>  timer = new Timer();
timer.schedule(new UpdateTask(),0,(1000));
tentry.requestFocus();
while(true)
{
    try{
        data = new byte[1000];
dataCount = in.read(data);
System.out.println("size of data " +
dataCount);
        if(dataCount < 1){
            System.out.println("got an error in read ");
            return;
        }
    }
}
while(dataCount > bytesUsed)
{
    Message msg = Message.createObject(data);
    bytesUsed = msg.encodeMessage().length;
    System.out.println("the message type is " +
        msg.getTypeOfMessage());
    if((msg.getTypeOfMessage() == 't') || (msg.getTypeOfMessage() == 'e'))
    {
        // The server sends this type of message when the game ends
        ResultAndScoresMessage res =
            (ResultAndScoresMessage) Message.createObject(data);
        String [] names = res.getNames();
        Int[] scr = res.getScores();
        Int[] tags = res.getTag();
        ls1.removeAll();
        for(int i = 0 ; i < scr.length ; i++)
        {
            System.out.println(names[i] + " " + scr[i]);
            if(tags[i] == tag)
            {
                l22.setText(scr[i] + " ");
            }
            ls1.add(names[i] + " " + scr[i]);
        }
    }
    if(msg.getTypeOfMessage() == 'e')
    {
        showStatus("The 3 minutes are over
            and the Game has ended");
        benter.setEnabled(false);
        bclrent.setEnabled(false);
        tentry.setEnabled(false);
        int maxScore = 0;
        String maxName = null;
        int maxTag = 0;
        for(int i = 0 ; i < scr.length ; i++)
        {
            if(scr[i] >= maxScore)
            {
                maxScore = scr[i];
                maxName = names[i];
                maxTag = tags[i];
            }
            System.out.println(names[i] + " - " + scr[i]);
        }
    }
if(maxTag == tag){
    // showMessageDialog displays the dialog box
    JOptionPane.showMessageDialog(this.getContentPane(),"Your score is "+maxScore+. Congratulations !! You are the winner");
    showStatus("Winner !!!");
} else {
    JOptionPane.showMessageDialog(this.getContentPane(),"The winner is "+maxName);
    showStatus("Thanks for playing with us");
} // By doing this the endless loop ends and the game ends.
return;
} else if(mesg.getTypeOfMessage()=='w'){
    // This message shows that a new legal word is entered by some user and hence //the list corresponding to it must be updated
    WordMessage wm = (WordMessage) Message.createObject(data);
    ls2.add(wm.getWord());
} else if(mesg.getTypeOfMessage()=='q'){
    // This message shows that the result of the word entered by the present player // result is taken back in terms of the integer. // The description of the errors is shown in the switch statements.
    WordResultMessage wm = (WordResultMessage) Message.createObject(data);
    switch(wm.getResult()){  
    case 100:  
        showStatus("Error: Word length should be more than 3 !!");  
        break;
    case 101:
        showStatus("Error: Word already used");
        break;
    case 102:
        showStatus("Error: not a legal word");
        break;
    default:
        showStatus("Invalid message received");
        break;
    case 103:
        showStatus("Error: Word not found");
        break;
    }
}
break;
case 103:
    showStatus("Error: word not in dictionary");
    break;
default :
    showStatus("Your word has been accepted");
    break;
}  
else {
}
byte [] dataBack = new byte[dataCount - bytesUsed];
for(int i = bytesUsed ; i < dataCount ; i++){
    dataBack[i - bytesUsed] = data[i];
}
dataCount = dataCount - bytesUsed;
bytesUsed = 0;
data = dataBack;
}
}catch(IOException e){
e.printStackTrace();
} catch(ClassNotFoundException e) { 
    e.printStackTrace();
}
}

public void actionPerformed(ActionEvent ae) {
    String s1 = tentry.getText().trim();
    // Checking if the Start Game Button is pressed
    if (ae.getSource()==bstartgame)
    { 
        try
        {
            StartGameMessage wm = new StartGameMessage();

            out.write(wm.encodeMessage(),0,wm.encodeMessage().length);
            System.out.println("Sending out a StartGame method");
        }
catch(IOException e) {
    e.printStackTrace();
}
benter.setEnabled(true);
bclrent.setEnabled(true);
bexit.setEnabled(true);
bstartgame.setEnabled(false);
tentry.setEnabled(true);
tentry.setText("");
tentry.requestFocus();
}

// Checking if the Enter Button is pressed
if (ae.getSource()==benter) {
    String word = ((tentry.getText()).trim()).toLowerCase();
tentry.setText("" );
benter.setEnabled(false);
tentry.requestFocus();
try {
    WordMessage wm = new WordMessage(word);
    out.write(wm.encodeMessage(),0,wm.encodeMessage().length);
} catch(IOException e) {
    e.printStackTrace();
    showStatus("The Boggle Game Server is busy. Please try later");
}
}

if (ae.getSource()==bclrent) {
    tentry.setText("" );
tentry.requestFocus();
}

// When any key is pressed
public void keyPressed(KeyEvent ke) {
    benter.setEnabled(true);
}
When Enter key is pressed -- >

```java
if (ke.getKeyCode()==10)
{
    String word = ((tentry.getText()).trim()).toLowerCase();
    tentry.setText("");
    benter.setEnabled(false);
    tentry.requestFocus();
    try
    {
        WordMessage wm = new WordMessage(word);
        out.write(wm.encodeMessage(),0,wm.encodeMessage().length);
    }
    catch (IOException e)
    {
        e.printStackTrace();
        showStatus("The Boogle Game Server is busy. Please try later");
    }
}
```

// This class defines the task for the TimerTask

```java
class UpdateTask extends TimerTask
{
    public void run()
    {
        timernmi=""+timeremni+"";
        l12.setText(timernmi);
        l12.repaint();
        tentry.requestFocus();
        if (timeremni == 0)
        {
            //showStatus("The timer has expired");
            timer.cancel();
        }
        timeremni--;
    }
}
```
Explanation

1> This thread was started since some browsers refused to display the Applet until the start method was complete. So we moved the processing for the start method to another method startProcess and in the start method created a thread which calls this method in the run.

2> The results for the word entered are read here so even though the user thinks that it is synchronous it is actually not.

3> The timer is used by the Applet for displaying the time remaining. This does not interact with the main applet just decrements the time remaining.

4> The text entry method used by the applet, these listeners sends the word to the GameServer.
public class BoogleGameServer extends HttpServlet{
    HashTable words;
    Dice dice;
    String spellFile = "//cise/homes/nghodke/tomcat/webapps/test/ispell_words.txt";
    BoogleGame bg = null;
    int uniqueTag = 0;

    protected void doGet(HttpServletRequest req,HttpServletResponse res) throws ServletException,
    IOException {
        int ret;
        int tag;
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        String name = req.getParameter("name");
        if(name == null)
            name = new String("John");
        if(name.equals("")
            name = new String("John");

            synchronized(BoogleGameServer.class){
                tag = uniqueTag++;
                <1>  if(bg == null){
                    /* create a new game */
                    <3>    bg = new BoogleGame(words,dice);
                    bg.start();
                            ret = bg.addPlayer(name,tag);
                }else{
                    ret = bg.addPlayer(name,tag);
                    if( ret == -1){
                        /* the boogle game is already playing so
                        create a new game */
                        <3>    bg = new BoogleGame(words,dice);
                        bg.start();
                                ret = bg.addPlayer(name,tag);
                    }
                }
            }
        }
    }
out.println("<HTML><HEAD><TITLE> Boogle Game
</TITLE>");

out.println("</HEAD><BODY>");

out.println("<!-- HTML CONVERTER -->");
out.println("<SCRIPT LANGUAGE="JavaScript"><!--
var _info = navigator.userAgent;
var _ns = false; var _ns6 = false;
out.println(" var _ie = (_info.indexOf("MSIE") > 0 &&
_win) > 0 &&
_windows3.1") < 0);";
out.println("//--></SCRIPT>";
out.println("<COMMENT><SCRIPT
LANGUAGE="JavaScript1.1"><!--
var _ns = (navigator.appName.indexOf("Netscape") >= 0 &&
_win) > 0 &&
_windows16") < 0 &&

out.println("java.lang.System.getProperty("os.version
").indexOf("3.5") < 0) ||");
out.println(" _info.indexOf("Sun") > 0));";
out.println(" var _ns6 = (_ns == true) &&
Mozilla/5") >= 0));";
out.println("//--></SCRIPT></COMMENT>";
out.println("";
out.println("<SCRIPT LANGUAGE="JavaScript"><!--
");
out.println(" if (_ie == true)
document.writeln("<OBJECT
";
out.println(" classid="clsid:E19F9331-3110-11d4-
991C-005004D3B3DB" $ObjectAttributes$";

out.println("codebase="$CabFileLocation$";<NOEMBED><
XMP>'");
out.println(" else if (_ns == true & _ns6 ==
false) document.writeln("<EMBED">
$type="$MimeType$"
$EmbedAttributes$";
out.println("$EmbedParams$ scriptable=false ");
```java
/* initialize the hashtable and dice here */
public void init(ServletConfig config) throws ServletException{
    super.init(config);
    words = new Hashtable(35000);// this value is the number of lines in file
    dice = new Dice();
    File fl = null;
    try{
        fl = new File(spellFile);
        FileReader fd = new FileReader(fl);
        BufferedReader reader = new BufferedReader(fd);
        String word;
        do{
            word = reader.readLine();
            if(word != null){
                words.put(word,new Boolean(true));
            }
        }while(word != null);
    }catch(IOException e){// just ignore this exception
        e.printStackTrace();
        System.out.println(fl.getAbsolutePath());
    }
}
```
public String getServletInfo() {
    return "Implementation of a Boogle Game";
}
}
Explanations:

1> We are synchronizing this on the class so that only one thread is executing in the critical section the access to the BoogleGame object is synchronized.

2> We are using only a single BoogleGame object since that is the only one we need once the game has started there is no need for us to remember it and hence that variable can be reused

3> Create the games and create a thread for that game

4> This is the dictionary we are using and is shared by all the threads, This is immutable hence not synchronized

5> The Dice object is also immutable and shared between all the games the games only get random pieces which do not change the internal state of the dice, the only thing that can change is the random number inside but that is ok it will be more random

6> The Dictionary is initialized at the init of the Servlet
import java.net.*;
import java.util.*;
import java.io.*;

/* Right now we are implementing this class as a thread it
needs to run initially listening for connections and at the
end of 3 mins to end the game when it should send an
endGame command to all the players in between it sends the
updated scores and the updated word list to the applets */
public class BoogleGame extends Thread{
    int port;
    int clientsConnected;
    Vector players;
    Dice dice;
    Hashtable words;
    Hashtable wordsAlreadyUsed;
    Vector wordsAlreadyUsedVec;
    Timer timer;
    Timer gameStartTimer = null;
    public boolean gameStarted;
    public boolean endGame;
    ServerSocket server_socket;
    String pieces;
    char wordArr[][];
    boolean sendData;
    int [] scores;
    Object game;
    ResultAndScoresMessage result;
    DataInputStream in;
    DataOutputStream out;
    Barrier bar;

    /*
    this is a constructor for the boogleGame
    this accepts a hashtable for searching the word
    */
    BoogleGame(Hashtable name,Dice dice){
        this.dice = dice;
        this.words = name;
        <1>    wordsAlreadyUsed = new Hashtable(60);// assuming the
                players would be able to enter 15 words each during the
                allotted 3 mins
        <1>    wordsAlreadyUsedVec = new Vector();
        <2>    players = new Vector();
                gameStarted = false ;
try{ 
    server_socket = new ServerSocket(0);
    <3> server_socket.setSoTimeout(500);
} catch (IOException e) {
    e.printStackTrace();
}
port = server_socket.getLocalPort();
pieces = dice.shakePieces();
wordArr = new char[4][4];
int count = 0;
char [] arr = pieces.toCharArray();
for (int i = 0 ; i < 4; i++) {
    for (int j = 0 ; j < 4 ; j++) {
        wordArr[i][j] = arr[count];
        count++;
    }
}
<4> sendData = false;
<5> clientsConnected = 0;
}

<7> String getPieces () {
    return pieces;
}

/* this function adds a user to the game. This returns portno if the user could be added and -1 otherwise. This might return -1 once the game has started or the number of users has reached 4 */
public int addPlayer (String name, int tag) {
    /* check whether the game has already started */
    /* if yes then return an error that signifies that the player has not been added to the game */
    <8> if (gameStarted) 
        return -1;
    Players pl = new Players (name, this, pieces, tag);
    <6> synchronized (players) {
        players.add (pl);
        if (players.size () >= 4) { // should not reach greater than ....
            /* we just set the value of gameStarted as true and let the run method figure out that the required number of players have joined in and let it start the game */
            <8> gameStarted = true;
public void delPlayer(int tag){
    synchronized(players){
        Players pl = getPlayer(tag);
        if(pl!= null)
            players.remove(pl);
        clientsConnected--;
    }
}

public void run(){
    System.out.println("The boogleGame has started");
    byte [] data = new byte[256];
    game = this;
    String [] name;
    int [] tags;
    Socket s = null;
    while(true){
        try{
            s = server_socket.accept();
            in = new DataInputStream(s.getInputStream());
            int dataCount = in.read(data);
            /* we have read some spurious data */
            if(dataCount < 1)
                continue;
            Message mr = (Message)Message.createObject(data);
            RegisterRequestMessage m = (RegisterRequestMessage)mr;
            if(m.getTypeOfMessage() == 'r'){
                /* we got a registration request 
                find out if we have the name in 
                the vector  get the player and 
                supply it this socket and start 
                the thread */
                synchronized(players){
                    Players pl = getPlayer(m.getTag());
                    pl.addSocket(s);
                    pl.startThread();
                    clientsConnected++;
                }
            }
        }
    }
}
scores = new int[players.size()];
name = new String[players.size()];
tags = new int[players.size()];

for(int i = 0 ; i < players.size() ;i++){
    name[i] = ((Players)players.elementAt(i)).getName();
    tags[i] = ((Players)players.elementAt(i)).getTag();
    scores[i] = 0;
}
result = new ResultAndScoresMessage('t',players.size(),name,scores,tags);

for(int i = 0 ; i < players.size(); i++){

    ((Players)players.elementAt(i)).sendResultAndScores(result);
}
/* now we have the number of required clients */
if(clientsConnected == 4){
    break;
}
/* some player has give a start game command so just check and break if all the players added have registered */

synchronized(players){
    if((gameStarted == true) && (players.size() == clientsConnected)){
        break;
    }else if((gameStarted == true) && (players.size() >= clientsConnected)){
        if(gameStartTimer == null){
            /* start a timer so we will eventually start this game */
            gameStartTimer = new Timer();
            gameStartTimer.schedule(new StartGameTask(),(30000));
        }
    }
}
and also the number of clients connected is equal to the number of players registered so we can start */

```java
synchronized(players) {
    if (players.size() == clientsConnected)
        break;
    else if ((gameStarted == true) && (players.size() >= clientsConnected)) {
        if (gameStartTimer == null)
            /* start a timer so we will eventually start this game */
            gameStartTimer = new Timer();

            gameStartTimer.schedule(new StartGameTask(), 30000);
        }
    }
}
}
```

```java
} catch (IOException e) {
    e.printStackTrace();
    if (gameStarted == true)
        synchronized(players) {
            if ((gameStarted == true) && (players.size() == clientsConnected)) {
                break;
            } else if ((gameStarted == true) && (players.size() >= clientsConnected)) {
                if (gameStartTimer == null)
                    /* start a timer so we will eventually start this game */
                    gameStartTimer = new Timer();

                    gameStartTimer.schedule(new StartGameTask(), 30000);
            }
        }
    }
}
```
}catch(ClassNotFoundException e){
   e.printStackTrace();
}

/* at this point we have a start game request we send all the clients an interrupt and ask them to stay at a barrier after signaling the barrier the threads will proceed */
startGame();

scores = new int[clientsConnected];
name = new String[clientsConnected];
tag = new int[clientsConnected];

for(int i = 0 ; i < players.size() ;i++){
   name[i] =
   ((Players)players.elementAt(i)).getName();
   tags[i] =
   ((Players)players.elementAt(i)).getTag();
}
result = new ResultAndScoresMessage('t',clientsConnected,name,scores,tags);

Timer timer = new Timer();
timer.schedule(new EndGameTask(),(182000));

int wordsSent = 0;
while(endGame!= true){
   try{
      /* its ok if we miss a few send by some players but in the end all the scores will be delivered to the players */
      synchronized(this){
         while(!sendData)
            wait();
         sendData = false;
         for(int i = 0 ; i < players.size(); i++){
            scores[i] =
            ((Players)players.elementAt(i)).getScore();
         }
      }
      result.updateScores(scores);
      for(int i = 0 ; i < clientsConnected; i++){
((Players)players.elementAt(i)).sendResultAndScores(result);

WordMessage wordMesg  = null;
boolean fnd = true;
while(fnd){
    synchronized(wordsAlreadyUsedVec){
        /* this means that there are words to send*/
        if(wordsAlreadyUsedVec.size()>wordsSent){
            wordMesg = new WordMessage((String)wordsAlreadyUsedVec.elementAt(wordsSent));
            fnd = true;
            wordsSent++;
        }else{
            fnd = false;
        }
    }
    if(fnd){
        for(int i = 0 ; i < clientsConnected; i++){
            ((Players)players.elementAt(i)).sendWord(wordMesg);
        }
    }
    if(endGame)
        break;
}catch(InterruptedException e){
    e.printStackTrace();
    if(endGame)
        break;
}finally{
    if(endGame)
        break;
}
sendEndGame();
result.updateType('e');
for(int i = 0 ; i < players.size() ; i++){
    scores[i] = ((Players)players.elementAt(i)).getScore();
}
result.updateScores(scores);
for(int i = 0 ; i < clientsConnected; i++){

    ((Players)players.elementAt(i)).sendResultAndScores(result);
}
System.out.println("Exiting game");

/* this function retrieves a player depending upon the name supplied and assumes the function calling this has already locked on players */
Players getPlayer(int tag){
    for(int i = 0 ; i < players.size(); i++){
        if(tag == ((Players)players.elementAt(i)).getTag()){
            return (Players)players.elementAt(i);
        }
    }
    return null;
}

synchronized void reqSend(){
    sendData = true;
    notifyAll();
}

/* this method interrupts the thread and attempts to start the game we cannot really start the game since 4 players might have joined in but not all of them might have connected */
public synchronized void requestStartGame(){
    if(gameStarted)
        return;
<8> gameStarted = true;
    ((Thread)game).interrupt();
}

/* this function interrupts all the player threads and sets the startGame flag so the players will send the startgame command to all the applets */
public void startGame(){
    gameStarted = true;
    /* signal barrier here so that all threads start at the same time */
<6> synchronized(Players){
    bar = new Barrier(players.size());
}
/* sleeping to give the last client time to load */

try{
    Thread.sleep(1000);
}catch(InterruptedException e){
    e.printStackTrace();
}

/* just interrupt all the threads */
for(int i = 0 ; i < players.size() ;i++){
    ((Players)players.elementAt(i)).supplyBarrier();
}
}

/* this cannot be synchronized since all the clients would be calling at the same time */
<12>void barrier() throws InterruptedException {
    bar.barrier();
}

/* this function ends a game and will send a endGame command to all the user threads */
void sendEndGame(){
    if(!endGame){
        WordMessage wm = null;
    }else{
        System.out.println("Really got an end game");
    }
    for(int i = 0 ; i < players.size() ;i++){
        ((Players)players.elementAt(i)).endGame();
    }
}

/* this is a timer class which counts down 3 mins and then ends this game*/
<14>class EndGameTask extends TimerTask{
    public void run(){
        endGame = true ;
        System.out.println("Ending the game");
        ((Thread)game).interrupt();
        //timer.cancel();
    }
}
/ this is a timer class which counts down 30 secs and then forcibly starts the game*/

```java
<8> class StartGameTask extends TimerTask{
    public void run()
    {
        gameStarted = true;
        synchronized(players){
            clientsConnected = players.size();
        }
        System.out.println("Forcing the start
game");
        ((Thread)game).interrupt();
    }
}

/* This function accepts a word from the client and checks if that word is correct or not and also returns the points that can be collected this function need to be synchronized since we will be updating the words already used list */
public synchronized int enterWord(String word){
    <13> if(word.length()<3){
          return 100;
    }
    Boolean got =
    (Boolean)wordsAlreadyUsed.get(word);
    /* we have found the word already present in the used list */
    if(got != null){
        return 101;
    }
    /* check whether the word is a consecutive sequence*/
    if(!checkLegalWord(word,-1,-1,null,(word.length() -1))){
        return 102;
    }
    /* at this point we don’t find the word already in the list check whether the word supplied is valid according to the dictionary */
    Boolean val = (Boolean)words.get(word);
    /* this is not a valid word */
    if(val == null){
        return 103;
    }
    wordsAlreadyUsed.put(word,new Boolean(true));
    /* we are adding this to a vector also to ease sending ... duplication*/
```
```java
synchronized(wordsAlreadyUsedVec) {
    wordsAlreadyUsedVec.add(word);
}

if (word.length() < 5)
    return word.length();
else if (word.length() < 9)
    return word.length() * 2;
else
    return word.length() * 3;

/* this function checks whether the word given has all */
/* the chars in sequence as in the gamepiece */
boolean checkLegalWord(String word, int pastX, int pastY, boolean[][] test, int iter) {
    boolean ret;
    if ((word.charAt(iter) == 'u') && (word.charAt(iter - 1) == 'q'))
        --iter;
    int lowerX, lowerY, upperX, upperY;
    if (test == null) {
        test = new boolean[4][4];
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                test[i][j] = false;
            }
        }
    }
    if (pastX == -1) {
        lowerX = 0;
        lowerY = 0;
        upperX = 3;
        upperY = 3;
    } else {
        lowerX = ((pastX == 0) ? 0 : (pastX - 1));
        lowerY = ((pastY == 0) ? 0 : (pastY - 1));
        upperX = ((pastX == 3) ? 3 : (pastX + 1));
        upperY = ((pastY == 3) ? 3 : (pastY + 1));
    }
    for (int i = lowerX; i <= upperX; i++) {
        for (int j = lowerY; j <= upperY; j++) {
            if ((j == pastY) && (i == pastX)) {
                } else {
                    if ((wordArr[i][j] == word.charAt(iter)) && (test[i][j] == false)) {
```
test[i][j] = true;
if(iter == 0)
    return true;
/* to take care of qu */
ret = checkLegalWord (word, i, j, test, (iter-1));
if(ret)
    return true;
test[i][j] = false;
}
}
return false;

/* this was written to test the boogle API on the command line */
public static void main(String []argv){
    Hashtable words = new Hashtable(35000); // this value is the number of lines in file
    try{
        FileReader fd = new FileReader("ispell_words.txt");
        BufferedReader reader = new BufferedReader(fd);
        String word;
        do{
            word = reader.readLine();
            if(word != null){
                words.put(word,new Boolean(true));
            }
        }while(word != null);
    }catch(IOException e){// just ignore this exception
        e.printStackTrace();
    }
    BoogleGame bg = new BoogleGame(words,new Dice());
    char []pie = bg.getPieces().toCharArray();
    int count = 0;
    for(int i = 0 ; i < 4 ; i++){
        for(int j = 0 ; j < 4 ; j++){
            System.out.print(" " + pie[count]+" ");
            count++;
        }
        System.out.println(" ");
    }
    System.out.println(bg.getPieces());
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
String data;
try{
do{
    System.out.print("?\n");
data = reader.readLine();
    int ret = bg.enterWord(data);
    System.out.println("the ret is "+ ret);
}while(data != null);
}catch(IOException e){
e.printStackTrace();
}
}
Explanations:

1> To maintain the words already used we have used a hashtable so searching it is very easy also in parallel we have maintained a vector of those words since accessing them sequentially is easy we needed it to send the words to the applet.

2> This is a vector as mentioned above.

3> The players added by the servlet are maintained in this vector. We instantiate the object when the Servlet adds the player but we instantiate the thread when the applet actually connects to the socket on which the BoogleGame is listening.

4> This is used in context of sending the words to the applet, This the flag to for the server to send the words.

5> This is the counter which maintains which Applets have actually connected.

6> We are synchronizing on the players on these occasions
   a. When the Servlet adds a player
   b. When the player removes itself from the game
   c. To get the number of players joined in
We need to synchronize these because the vector is constantly changing but we can do away with synchronization once the game has started because we won't be changing the contents of this vector.

7> No need to synchronize this since this is a read only operation

8> This is not synchronized since it is a boolean and according the java memory model is atomic and the only thread can change its value which is a true

9> At this point we have started the game so no need to synchronize the vector of players from here.

10> This is a producer consumer problem we have elected to use a Boolean, by using this we might miss a few requests for result update and word update while we are processing one but that is ok because we would be
sending the final results which would have the second result also and also since it is idempotent. This is the case where we will miss the send requests but in the end we are sending the actual scores of all the clients and not the updates hence we won't miss anything.

    Update(1,1)
    Update(2,4)
    Send results({1,1},{2,4});

11> This is the Words already used vector this is accessed by the enter word to add words and also by the thread to send the updates to the clients

12> This is the barrier at which all the players wait when the startgame command is given this is used to all the players start the game at the same time

13> This is one of the mistakes we made we assumed that the minimum word length required was 3 which should had been 2

14> This is the end game timer which would interrupt the game thread and set the endgame = true
import java.net.*;
import java.io.*;
import java.util.*;
/* this is implemented as a thread ... all the threads
would be blocked on reading the sockets ... one of the
socket can request a start command... the way we will start
or stop the game is to call the methods in the object and
interrupt the thread so it comes out of the wait state ...
we also setup a flag so it now what has happened */
public class Players implements Runnable{
    boolean gameStarted;
    boolean endGame;
    Socket socket = null;
    String name;
    BoogleGame game;
    String pieces;
    int score;
    DataInputStream in;
    DataOutputStream out;
    Thread t;
    int tag;

    Players(String name,BoogleGame game,String pieces,int tag){
        this.name = name;
        this.game = game;
        this.pieces = pieces;
        this.tag = tag;
    }

    public String getName(){
        return name;
    }

    public int getTag(){
        return tag;
    }

    /* the socket on which the client would be listening is
entered over here also intialize the input and the
output dataStreams */
    public void addSocket(Socket s){

socket = s;
try{
  socket.setSoTimeout(500);
  if(in == null)
    in = new DataInputStream(socket.getInputStream());
  if(out == null)
    out = new DataOutputStream(socket.getOutputStream());
} catch(SocketTimeoutException e){
  e.printStackTrace();
} catch(IOException e){
  e.printStackTrace();
}
}

public void startThread(){
  t = new Thread(this);
  t.start();
}

/* this basically just listens on the port and calls
the methods in
the BoogleGame Object */
public void run(){
  byte [] data = new byte[256];
  while(true){
    try{
      int dataCount = in.read(data,0,255);
      if(dataCount < 1){
        game.delPlayer(tag);
        return;
      }
      StartGameMessage msg = (StartGameMessage) Message.createObject(data);
      game.requestStartGame();
    } catch(SocketTimeoutException e){
      if(gameStarted == true){
        break;
      }
    } catch(SocketException e){
      if(gameStarted == true){
        break;
      }
      e.printStackTrace();
    } catch(Exception e){
      if(gameStarted == true){
        break;
      }
      e.printStackTrace();
    }
  }
}
break;
}
e.printStackTrace();
}catch(ClassNotFoundException e){
e.printStackTrace();
}catch(Exception e){
    if(gameStarted == true){
        break;
    }
e.printStackTrace();
}

/* we can assert over here that gameStarted==true */
prepareToStartGame();
/* at this point they have come out of the barrier */
sendPiecesToApplet();

/* at this point we have started the game ... so we wait in a loop waiting for the applet to send a word */
while(true){
    try{
        int dataCount = in.read(data,0,255);
        if(dataCount < 1){
            break;
        }
        Message mesg =
(Message)Message.createObject(data);
        if(mesg.getTypeOfMessage() != 'w')
            continue;
        WordMessage wm =
(WordMessage)Message.createObject(data);
        String wor = wm.getWord();
        int ret = game.enterWord(wor);
        sendWordResult(new
WordResultMessage(ret));
        if( (ret >0) && ( ret <90)){
            score = score + ret;
        }
game.reqSend();
    }
} catch(SocketException e){
    if(endGame == true){
        break;
    }
}
    // e.printStackTrace();
} catch(IOException e){
    if(endGame == true){
        break;
    }
    // e.printStackTrace();
} catch(ClassNotFoundException e){
    e.printStackTrace();
} finally{
    if(endGame == true){
        break;
    }
}
/* if we reach this point the game has ended */

/* this object sends scores and results back to the applet */
<synchronized void sendWordResult(WordResultMessage res){
    try{
        if(out == null )
            out = new 
            DataOutputStream(socket.getOutputStream());

            out.write(res.encodeMessage(),0,res.encodeMessage().length);
    }catch(IOException e){
        e.printStackTrace();
    }
}
/* this method sends scores and results back to the applet */
<synchronized void sendResultAndScores(ResultAndScoresMessage res){
    if(socket == null){
        System.out.println("player has not yet registered ");
        return;
    }
    System.out.println("res" + res);
    try{
        if(out == null )
            out = new 
            DataOutputStream(socket.getOutputStream());

out.write(res.encodeMessage(), 0, res.encodeMessage().length);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

/* this method sends wordsback to the applet */
<synchronized void sendWord(WordMessage res) {
    try{
        out.write(res.encodeMessage(), 0, res.encodeMessage().length);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

/* this functions initializes the barrier at which the thread will stop so that all the players start off at the same moment */
void supplyBarrier(){
    gameStarted = true;
    <3> t.interrupt();
    System.out.println("got a barrier for "+ name);
}

<int getScore(){
    return score;
}

/* this function waits in the barrier at the end it starts off */
void prepareToStartGame(){
    try{
        game.barrier();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

<synchronized void sendPiecesToApplet(){
    try{
        GamePiecesMessage gp = new GamePiecesMessage(pieces.toCharArray());
        GamePiecesMessage gp = new GamePiecesMessage(pieces.toCharArray());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
out.write(gp.encodeMessage(), 0, gp.encodeMessage().length);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

synchronized void startGame() {
    gameStarted = true;
}

/* this function will setup and interrupted exception
   for this thread and also set the endGame variable so
   that the thread exits */
synchronized void endGame() {
    endGame = true;
}
**Explanations**

1. Set SocketTimeout so we do not block on a read and keep polling for the endgame and StartGame flags.
2. If we get an IO Exception, which would be because of the client closing the browser, then delete this player and curtail the thread. At this point we are not holding any resources, so we can just return and end the thread.
3. When the game sets the StartGame flag it also interrupts the thread.
4. The score is not synchronized since only one thread is updating the score and we have defined score as an int. So we are ok according to the Java memory model.
5. These methods synchronize access to the outputDataStream since the player thread as well as the game thread can access it. We had experienced some corruption of data in OS when this stream was accessed simultaneously so we decided to synchronize the access to this.