Integrantes

Saúl Atochero, Fernando Guio, David Hernández y Diana Tena

Posición

```
public class Posicion {

// Aquí se guarda la fila y la columna del último movimiento del jugador
        int fila;
        int columna;

 // Aquí se indica el símbolo que se pone; una X o O
        char letra;

// Notificado será para que el hilo se despierte para que comience su turno
        private boolean notificado = false;

// Este método mantendrá al hilo interrumpido hasta que el otro jugador realice su
movimiento o pase su turno
        public synchronized boolean espera() {
            try {

            // Espera hasta ser notificado o hasta que pasen 10 segundos para que
            el turno pase a sí mismo y despertarse
                this.wait(10000);

            // Retorna el estado de 'notificado'
                return notificado;
            } catch (InterruptedException e) {

            // Retorna false si el hilo fue interrumpido
                return false;
            } finally {

            // Restablece el estado para el próximo uso
                notificado = false;
            }
        }


 // Esto para despertar el hilo y continúe el programa. El turno ahora lo tiene el
jugador.
        public synchronized void despierto() {
            this.notificado = true;
```

```
            this.notify();
        }



// Carga la posición del último movimiento
        public synchronized void cargaPosicion(int pfila, int pcolumna) {
          this.fila = pfila;
          this.columna = pcolumna;
        }

// Carga el símbolo del jugador
        public synchronized void cargaLetra(char pletra) {
          this.letra = pletra;
        }

// Devuelve la fila, la columna y el símbolo del último movimiento
        public synchronized int fila() {
          return this.fila;
        }
        public synchronized int columna() {
          return this.columna;
        }
        public synchronized char letra() {
          return this.letra;
        }

// Devuelve lo opuesto del que te dan la funcion de arriba
        public synchronized char otraletra() {
          return (char)(this.letra == 'X' ? 'O' : 'X');
        }
      }
```

Tablero

```
// Tablero es el gráfico. Extiende de JFrame para el diálogo modal e implementa ActionListener para que sepa cuando clicamos, como si tuviera una oreja que estuviera “escuchando” lo que hacemos en el programa cuando se ejecuta
public class Tablero extends JFrame implements ActionListener {

// Para tablero 3 x 3
        final int n = 3;
// Creamos los distintos botones
        JButton[][] boton;
// Creamos una fuente para las letras
        Font f;
// Variable que indica si el jugador puede tocar o no el tablero
        boolean activo;
// Referencia para las posiciones
        Posicion p;

// Constructor para crear tablero
        public Tablero(Posicion pp) {
// Se le pone un título al tablero
          super("tres en raya");
// Creamos las posiciones
          this.p = pp;
// Indicamos el tamaño de la pantalla y si se redimensiona o no
          this.setSize(500, 500);
          this.setResizable(false);
// Además de ello, se indica que por el momento NO se puede tocar el tablero, se indica que es falso
          this.activo = false;
// Se indica la fuente de la letra, el estilo y el tamaño de la letra
          this.f = new Font("Monospaced", 0, 100);
// Se indica dónde se colocan los botones, en un array de 3x3
          this.boton = new JButton[3][3];
// Se crea un Layout a través de JFrame, que será un Grid de 3x3
          this.setLayout(new GridLayout(3, 3));

// Esto es para crear los botones en el array
          for(int i = 0; i < 3; ++i) {
            for(int j = 0; j < 3; ++j) {
              this.boton[i][j] = new JButton();
// Nos permite indicar el valor del botón para cuando vayamos a utilizarlo; lo identificamos
              this.boton[i][j].setActionCommand(i + "-" + j);
// Cuando se haga click en el botón, se invocará el actionPerformed del ActionListener del componente
              this.boton[i][j].addActionListener(this);
```

```
// Se indica la fuente
                this.boton[i][j].setFont(this.f);
// Se añade al array
                this.add(this.boton[i][j]);
              }
            }
        // Refresca la interfaz
            this.repaint();
        // Hace visible la ventana
            this.setVisible(true);
          }

// Esto es para colocar la X o O en una posición específica
          public void Poner(int i, int j, char letra)
            i= fila j = columna
        // Pinta la X o O
            this.boton[i][j].setText(String.valueOf(letra));
        // Desactiva ese botón
            this.boton[i][j].setEnabled(false);
        // Refresca la interfaz
            this.repaint();
          }

// Es lo mismo que lo anterior, pero se le pasa el botón en vez de las coordenadas
          public void Poner(JButton j, char letra) {
            j.setText(String.valueOf(letra));
            j.setEnabled(false);
            j.repaint();
          }

// Cuando pulsamos un botón
          public void actionPerformed(ActionEvent e) {
// En caso de que estemos activamos
            if (this.activo) {
        // Obtenemos las coordenadas del botón, lo cual indicamos anteriormente
              String[] aux = e.getActionCommand().split("-");

// Obtiene el primer caracter del aux y lo define como su fila y el siguiente caracter es
la columna
              int fila = Integer.parseInt(aux[0]);
              int columna = Integer.parseInt(aux[1]);

// Con el método “Poner” se indica que en ese botón seleccionado, a través de la
Posición, se coloca la X o el O del jugador en el botón
              this.Poner((JButton)e.getSource(), this.p.letra());
// La indica y guarda en la Posición
              this.p.cargaPosicion(fila, columna);
```

```
// Desactivamos el tablero para no poder usarlo
                this.activo = false;

// Notificamos al hilo de que ya se ha pulsado el botón para que se despierte y continúe el juego
                this.p.despierto();
              }
            }

// Activamos el tablero
            public void Activo() {
              this.setTitle("Es tu turno");
              this.activo = true;
            }

//Desactivamos el tablero
            public void Desactivo() {
              this.setTitle("Espera a que el otro juegue");
              this.activo = false;
            }

// En caso de ganar imprimer 3 en raya
            public void gano() {
              this.setTitle(" HAY TRES EN RAYA ");
            }

// Imprime el empate
            public void empate() {
                      this.setTitle(" EMPATE ");
                    }

// Comprueba si hay huecos en el tablero con un bucle
            public boolean hueco() {
              for(int i = 0; i < 3; ++i) {
                for(int j = 0; j < 3; ++j) {
                  if (this.boton[i][j].getText().equals("")) {
                  // En caso de que haya algún hueco vacío, se mantiene en true
                    return true;
                  }
                }
              }

          // En caso de que NO haya huecos, entonces retorna falso
              return false;//retorna false si esta lleno
            }
```

// Según ChatGPT esto sirve para lo siguiente:

En la primera verificación intenta comprobar si la posición está o no vacía; en caso de que esté vacía, entonces no puede haber una línea e inmediatamente la función retorna false

En la segunda verificación compara el texto en las posiciones (x0, y0) con las posiciones (x1, y1) para ver si tienen el mismo texto y continúa verificando.

En la tercera verificación compara lo que hay en (x1, y1) con lo que contiene (x2, y2).

// Si las 3 condiciones son verdaderas, hay línea. Si no, pues no.

```
    public boolean linea(int x0, int y0, int x1, int y1, int x2, int y2)
      return !this.boton[x0][y0].getText().equals("") &&
this.boton[x0][y0].getText().equals(this.boton[x1][y1].getText()) &&
this.boton[x1][y1].getText().equals(this.boton[x2][y2].getText());
    }
```

// Distintas combinaciones ganadoras para el 3 en raya

```
    public boolean enraya() {
      return this.linea(0, 0, 0, 1, 0, 2) || this.linea(1, 0, 1, 1, 1, 2) || this.linea(2, 0, 2, 1, 2,
2) || this.linea(0, 0, 1, 0, 2, 0) || this.linea(0, 1, 1, 1, 2, 1) || this.linea(0, 2, 1, 2, 2, 2) ||
this.linea(0, 0, 1, 1, 2, 2) || this.linea(0, 2, 1, 1, 2, 0);
    }
  }
```

Lógica

```
// Extiende de Thread que implementa de Runnable para el método run
public class Logica extends Thread {

// Identificador del jugador
    int jugador;
// Socket para la conexión TCP
    Socket estelado = null;
// Stream para enviar datos
    DataOutputStream salida = null;
// Stream para recibir datos
    DataInputStream entrada = null;
    Tablero t;
    Posicion p;

// Constructor para crear logica a partir de tablero y la posición
    public Logica(Tablero pt, Posicion pp) {
        this.t = pt;
        this.p = pp;
    }

// Conexión hacia el server
    public void Conecto() {
        boolean termino = false;

// Loop infinito hasta que conecte
        while (!termino) {
            termino = true;

            try {

// Dirección del servidor y puerto
                this.estelado = new Socket("localhost", 12345);

            } catch (IOException e) {
// En caso de que no se pueda conectar, termina el bucle e indica que no se puede
conectar
                termino = false;
                System.out.println("No me puedo conectar");
            }
        }
```

```
        // Cuando se sale del while porque ha conectado con el servidor, inicia los
streams de envío y recibo de datos entre los jugadores
        try {
            this.salida = new DataOutputStream(this.estelado.getOutputStream());
            this.entrada = new DataInputStream(this.estelado.getInputStream());
        }
            catch (IOException e) {
            // Manejar excepción adecuadamente
        }
    }

// Obtiene el turno del jugador desde el servidor
    public int turno() {
        int aux = -1;

// Intenta leer aquello que obtiene de la entrada de datos y posteriormente lo retorna
para el método
        try {
            aux = this.entrada.readInt();
        } catch (IOException e) {

        }

        return aux;
    }

// Inicia el juego y el hilo de ejecución invocada desde inicio. PJ es el jugador.
    public void inicio(int pj) {
        this.jugador = pj;
        this.start();
    }

// Método principal que maneja la logica del juego y que proviene de Thread que
proviene a su vez de Runnable
    public void run() {

// Determina si es el turno del jugador
        boolean miturno = this.jugador == 1;
```

```
// Mientras haya huecos y nadie haya ganado se hará este bucle
        while (this.t.hueco() && !this.t.enraya()) {

// En caso de que sea mi turno
            if (miturno) {

// Activa el tablero y esperamos a que pulse un botón o pase un tiempo para que
pierda el turno. El “espera” proviene de la posición explicada en el punto anterior.
                this.t.Activo();
                boolean accionRealizada = this.p.espera();

// Si me devuelve true en el método del espera, es porque ha pulsado un botón
                    if (accionRealizada) {
                            try {

// El 1 indica que se ha realizado un movimiento
                                    this.salida.writeInt(1);
                                    this.salida.writeInt(this.p.fila());
                                    this.salida.writeInt(this.p.columna());
                            } catch (IOException e) {
                                    System.out.println("Error al enviar movimiento: " +
                                e.getMessage());
                            }
                        }

// En caso de que no haya pulsado ningún botón se pasa el turno
                    else {
                            try {
                                System.out.println("Paso de turno");
            // El 0 indica ausencia de movimiento, enviamos 3 ceros por que el servidor
            //pide de 3 en 3
                                this.salida.writeInt(0);
                                this.salida.writeInt(0);
                                this.salida.writeInt(0);
                            }
                                catch (IOException e) {
                                // Error atrapado
                            }
                }
            }
```

```
// En caso de que NO sea mi turno, me desactivo con el método del tablero
        else {
            this.t.Desactivo();

            try {
// Lee el indicador de movimiento, sobre si ha recibido o no un movimiento, la fila y la columna
                int indicador = this.entrada.readInt();
                int fila = this.entrada.readInt();
                int columna = this.entrada.readInt();

// En caso de que el indicador sea 1, entonces se ha recibido un movimiento y se coloca la letra en la fila y columna correspondientes a través del método del tablero y la letra de la posición que se escoge en el método correspondiente
        if (indicador == 1) {
                    this.t.Poner(fila, columna, this.p.otraletra());
                }

// En el caso contrario, en el caso de que le llegue 0, porque solo envía un 0 o un 1, entonces no se ha recibido el movimiento
        else{
                    System.out.println("No se recibió movimiento.");
                }
            } catch (IOException e) {
                //System.out.println("Error al recibir movimiento: " + e.getMessage());
            }
        }

// Cambia de turno
        miturno = !miturno;
        try {
// Espera medio segundo para el siguiente turno
            Thread.sleep(500);
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

// En caso de que hagamos “en raya” y esto se compruebe a través del método del tablero, hacemos lo siguiente
  if (this.t.enraya()) {//Ejecuta todo el resto si hay enraya o empate
                this.t.gano();
                    if(this.t.PopUpSiNo()) {//Se ejecuta una ventana emergente para decir si quiere continuar jugando
                                dichoSI=true;//En caso de si, pone a true la variable
                                confirmarjuego(); //Este es un metodo que va a retornar la variable de DichoSi
```

```
                        cerrarConexiones();//Se encarga de cerrar la conexion con el
servidor
                }else {
                        dichoSI=false;
                        this.t.dispose();//En caso de No cierra la pantalla y no hace nada
mas
                        cerrarConexiones();
                }

                        } else { //En caso de que nadie haya ganado, y haya habido un
empate se ejecuta esto
                                this.t.empate();
                                if(this.t.PopUpSiNo()) {//Se ejecuta una ventana emergente
para decir si quiere continuar jugando
                        dichoSI=true;//En caso de si, pone a true la variable
                        confirmarjuego();
                        cerrarConexiones();
                }else {
                        dichoSI=false;
                        this.t.dispose();//En caso de No ,cierra la pantalla y no hace nada
más
                        cerrarConexiones();
                }
                        }
        try {
            Thread.sleep(1000); // Espera 1 segundos
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
        if (dichoSI) {//Si anteriormente han dicho Si, quita la pantalla y pone una nueva
            this.t.dispose();
            Inicio i = null;
            i.nuevoJuego();
                }


// Se cierran las conexiones, la de los clientes y el servidor
        cerrarConexiones();
    }

// Se cierran las conexiones del servidor a través de este método
    private void cerrarConexiones() {
        try {
            // Se cierra la entrada y salida de datos junto al servidor
            if (this.entrada != null) this.entrada.close();
            if (this.salida != null) this.salida.close();
            if (this.estelado != null) this.estelado.close();
        } catch (IOException e) {
```

```
            System.out.println("Error cerrando conexiones: " + e.getMessage());
        }
    }
}
```

GameSession

```
// Este es el hilo del juego de la sesión
class GameSession extends Thread {

// Creamos dos Sockets distintos para los jugadores
    private Socket player1;
    private Socket player2;
// Aquí se indica si es que el jugador competirá contra una IA
    private boolean isAgainstAI;

// Creamos un tablero para que siga el server la cuenta de las 3 en raya
    private int[][] tablero = new int[3][3];
    private Server s;

// En el método indicamos quién es el jugador 1, el jugador 2, si es que jugamos
contra la IA y el servidor que referencia al Server
    public GameSession(Socket player1, Socket player2, boolean isAgainstAI, Server s) {
        this.player1 = player1;
        this.player2 = player2;
        this.isAgainstAI = isAgainstAI;
// Importante: esto almacena la referencia a Server
        this.s = s;
    }

// Como extiende de Thread, usamos el método run
    public void run() {
        try {

// Los DataOutput e Input son para la inserción de datos y el envío de los mismos de
los distintos jugadores
            DataOutputStream out1 = new DataOutputStream(player1.getOutputStream());
            DataInputStream in1 = new DataInputStream(player1.getInputStream());

// Cuando los vamos a crear para el segundo jugador, preguntamos si es que vamos a
competir contra la IA y, en base a eso, creamos o no estos elementos
            DataOutputStream out2 = isAgainstAI ? null : new
DataOutputStream(player2.getOutputStream());
            DataInputStream in2 = isAgainstAI ? null : new
DataInputStream(player2.getInputStream());

            int turnoJugador1 = 1;
            int turnoJugador2 = 2;
```

```
// Es el turno del jugador 1 y, en caso de que no juguemos contra la IA, le toca al 2
        out1.writeInt(turnoJugador1);
        if (!isAgainstAI) {
                out2.writeInt(turnoJugador2);
        }

        // Asume que es el turno del jugador 1 inicialmente
        boolean esTurnoDelJugador1 = true;

// Mientras que sea cierto
        while (true) {

// En caso de ser el turno del primero
            if (esTurnoDelJugador1) {

        // Espera por el movimiento del jugador 1
              int indicador = in1.readInt();
                System.out.println(indicador);

        // Se indica la fila y la columna y se inserta en el tablero
                int fila = in1.readInt();
                int columna = in1.readInt();
                tablero[fila][columna] = 1;

        // En caso de que juguemos con otro, se hace lo mismo
                if (!isAgainstAI) {

        // Indica al jugador 2 que hay un movimiento
                    out2.writeInt(indicador);
                    out2.writeInt(fila);
                    out2.writeInt(columna);
                }

           } else {

        // Gestiona el movimiento del jugador 2 o de la IA
              if (!isAgainstAI) {
                int indicador = in2.readInt();
                System.out.println(indicador);
                    int fila = in2.readInt();
                    int columna = in2.readInt();
                    tablero[fila][columna] = 2;

        // Indica al jugador 1 que hay un movimiento
                    out1.writeInt(indicador);
                    out1.writeInt(fila);
                    out1.writeInt(columna);
              }
```

```
 else {
        // Lógica para el movimiento de la IA
                int[] movimiento = generarMovimientoAleatorioParaIA();
                int fila = movimiento[0];
                int columna = movimiento[1];
                tablero[fila][columna] = 2;
                out1.writeInt(1); // Indica al jugador 1 que hay un movimiento de la IA
                out1.writeInt(fila);
                out1.writeInt(columna);
              }
            }
        // Cambia el turno
            esTurnoDelJugador1 = !esTurnoDelJugador1;

        // Verifica el fin del juego
            if (tableroLleno() || hayTresEnRaya()) {
                cerrarConexiones();
                return;
            }
        }
      } catch (IOException e) {
          System.out.println("Error en GameSession: " + e.getMessage());
      } finally {
          cerrarConexiones();
      }
   }


   private int[] generarMovimientoAleatorioParaIA() {
      Random random = new Random();
      int fila, columna;
      do {
// Asumiendo un tablero de 3x3
          fila = random.nextInt(3);
          columna = random.nextInt(3);
      }

// Repite hasta encontrar una celda vacía
      while (!esCeldaVacia(fila, columna));

      // Realiza el movimiento en el tablero para la IA y asumimos que el 2 representa
los movimientos de la IA
      tablero[fila][columna] = 2;
      return new int[]{fila, columna};
   }
```

```
    private boolean esCeldaVacia(int fila, int columna) {
// 0 indica celda vacía
        return tablero[fila][columna] == 0;
    }

// Se cierran las conexiones dependiendo de si hemos jugado contra la IA o no
    private void cerrarConexiones() {
            if(isAgainstAI) {
                    this.s.sumar(1);
            }else {
                    this.s.sumar(2);
            }

// Se cierran los Sockets de los jugadores
        try {
            player1.close();
            if (!isAgainstAI) {
                player2.close();
            }
            System.out.println("Conexiones con los clientes cerradas.");
        } catch (IOException e) {
            System.out.println("Error cerrando conexiones: " + e.getMessage());
        }
    }

    private boolean tableroLleno() {
        for (int i = 0; i < 3; i++) {
            for (int j = 0; j < 3; j++) {
                if (tablero[i][j] == 0) {
// Hay al menos una celda vacía, el tablero no está lleno.
                    return false;
                }
            }
        }
// No se encontró ninguna celda vacía, el tablero está lleno.
        return true;
    }
```

```
    private boolean hayTresEnRaya() {
        // Verifica todas las posibles líneas de tres en el tablero.
        // Horizontal, vertical y diagonales.
        for (int i = 0; i < 3; i++) {
// Hay tres en raya horizontalmente
            if (tablero[i][0] != 0 && tablero[i][0] == tablero[i][1] && tablero[i][1] == tablero[i][2]) {
                return true;
            }
// Hay tres en raya verticalmente
            if (tablero[0][i] != 0 && tablero[0][i] == tablero[1][i] && tablero[1][i] == tablero[2][i]) {
                return true;
            }
        }
 // Hay tres en raya en la diagonal principal.
        if (tablero[0][0] != 0 && tablero[0][0] == tablero[1][1] && tablero[1][1] == tablero[2][2]) {
            return true;
        }
// Hay tres en raya en la diagonal inversa.
        if (tablero[0][2] != 0 && tablero[0][2] == tablero[1][1] && tablero[1][1] == tablero[2][0]) {
            return true;
        }
 // No se encontró ninguna línea de tres.
        return false;
    }
}
```

Server

```
public class Server {

// Este es el número máximo de permisiones permitidas
    private static int MAX_CONNECTIONS = 4;
// Este es el tiempo de espera para que el segundo jugador aparezca en milisegundos
    private static final long WAIT_TIME_FOR_SECOND_PLAYER = 10000;
// Por si nos desconectamos antes de iniciar la partida
boolean nosalir=true;

// Al inicio creamos una estancia del servidor para que empiece a escuchar las
conexiones
    public static void main(String[] args) {
        new Server().startServer();
    }

// Inicia el servidor
    public void startServer() {

// Se crea una lista de Sockets de los clientes que es un arrayList
        List<Socket> clients = new ArrayList<>();

// Se intenta crear un objeto ServerSocket al que le añadimos el puerto
        try (ServerSocket serverSocket = new ServerSocket(12345)) {
            System.out.println("Servidor esperando conexiones...");

// Timer nos permitirá ejecutar distintas tareas en diversos intervalos de tiempo
            Timer timer = new Timer();

// Mientras no salgamos de la conexión
            while (nosalir) {
            // Y mientras el número del array de clientes sea menor a las conexiones
máximas anteriormente estandarizadas
                    if(clients.size() < MAX_CONNECTIONS) {
// Aceptamos en el socket al nuevo cliente
                Socket client = serverSocket.accept();
                clients.add(client);

// Si el tamaño de los clientes, cuyo resto sea distinto de 0, sean impares, se hace lo
siguiente
                if (clients.size() % 2 != 0) {
// Se indica el jugador que está conectado
                    System.out.println("Jugador 1 conectado desde: " +
client.getInetAddress().getHostAddress());
```

```
// Se crea una tarea con tiempo
                TimerTask task = new TimerTask() {

// Ejecutaremos lo siguiente
                    public void run() {

// En caso de seguir siendo impar el número de clientes ya que no se ha conectado otro para emparejar, entonces se jugará contra la IA
                        if (clients.size() % 2 != 0) {
                            System.out.println("Iniciando juego contra la IA.");
            // Se crea un socket falso para representar la IA y se añade al array de clientes
                            Socket iaPlayer = new Socket();
                            clients.add(iaPlayer);


// Nota el cambio: Server.this se usa para referirse a la instancia de Server desde la clase anónima
            // Se crea una sesión de juego con el cliente, la IA, un booleano que nos dice si jugamos o no contra la IA y servidor
                            GameSession gameSession = new GameSession(client, iaPlayer, true, Server.this);
                            gameSession.start();
                        }
                    }
                };

// Esta línea permite programa la tarea para que se ejecute después del tiempo previsto
                timer.schedule(task, WAIT_TIME_FOR_SECOND_PLAYER);
            } else {

            // Sino, se conectará un segundo jugador
                System.out.println("Jugador 2 conectado desde: " + client.getInetAddress().getHostAddress());
            // Se cancelará si el segundo jugador se conecta
                timer.cancel();
// Se crea un futuro Timer para posteriores esperas
                timer = new Timer();
```

```
// Se crean dos sockets distintos de jugadores.
        El primer jugador tiene el valor del penúltimo que se conectó y el segundo
jugador tiene el valor del último cliente que se conectó.
                Socket player1 = clients.get(clients.size() - 2);
                Socket player2 = client;
                System.out.println("Iniciando una nueva sesión de juego.");
                // Aquí también usamos Server.this para referirse a la instancia actual de
Server; el false nos dice que no jugamos contra ninguna IA
                GameSession gameSession = new GameSession(player1, player2, false,
Server.this);
                gameSession.start();
            }
                }else {
                        System.out.println("Se ha alcanzado el máximo número de
conexiones.");
                }
        }

      } catch (IOException e) {
          System.out.println("Error en el servidor: " + e.getMessage());
          e.printStackTrace();
      }
  }

// Este método de sumar incrementa el valor de las máximas conexiones de forma
segura en un hilo aparte para que no haya interferencias o choques
   public synchronized void sumar(int cantidad) {
        MAX_CONNECTIONS=MAX_CONNECTIONS+cantidad;
   }
}
```

Inicio

```
public class Inicio{
//Definimos las variables p,t,juego, para su uso posterior
     Posicion p;
     Tablero t;
     Logica juego;
    public static void main(String[] args) {
//Al iniciar Inicio usa el metodo nuevoJuego, que se encarga de crear la partida
    nuevoJuego();



    }
     public static void nuevoJuego() {
//Creamos la posición
          Posicion p = new Posicion();
//Creamos el tablero
           Tablero t = new Tablero(p);
// Creamos una lógica metiendo la posición y tablero
           Logica juego = new Logica(t, p);
// Invoca la función de Logica de conectar con el servidor
           juego.Conecto();
// Obtiene el turno el jugador que se conecta
           int turno = juego.turno();
// Establecemos el símbolo que le toca a cada uno en base a quién se ha conectado
antes
           p.cargaLetra((char)(turno == 1 ? 'X' : 'O'));
           //System.out.println("eeeeeeeee"+turno);
           juego.inicio(turno);
      }

}
```

PROYECTO

Nos descargamos el Servidor Básico
Con WinSCP nos conectamos a la máquina virtual y metemos el servidor mencionado
Tenemos que ejecutarlo con sudo bash

EXPLICACIÓN PSP

INICIO

Crearemos una posición y un tablero
Crearemos la lógica metiendo la posición y un tablero
Invocamos la función de lógica para conectar con el servidor
Obtendremos el turno y establece el símbolo que le toca a cada uno e indica el turno

LÓGICA

Crearemos un constructor para crear la lógica del tablero y posición
Se hará una conexión hacia el server con un loop infinito hasta que se conecte, y le daremos la dirección del servidor y puerto. Cuando se sale del while inicia los streams de envío y recibo de datos.
Obtiene el turno del jugador desde el servidor
Inicia el juego y el hilo de ejecución invocado desde inicio
Metodo principal que maneja la logica del juego
Determina si es el turno del jugador
Mientras haya huecos y nadia haya ganado,

- **Si es mi turno**
  - Activa el tablero, espera a que se le pulse un botón o pase un tiempo a que pierda el turno
    - Si me devuelve true es porque ha pulsado un botón e indica que se ha realizado un movimiento, si no indica ausencia de movimient

- **Si no es mi turno**
  - Lee el indicador de movimiento
    - Si el indicador de movimiento es igual a 1, llamara al método Poner() para colocar una pieza en el tablero, si no nos saltará un mensaje que no se recibió un movimiento válido

Cambia de turno
Espera medio segundo para el siguiente turno e interrumpe el hilo actual si se produce una excepción
Si hay tres enraya muestra mensaje de victoria en el título, si no muestra mensaje de empate, a la vez de que muestra una mini ventana, indicando de si quieren volver a jugar.
Espera 2 segundos e interrumpe el hilo si se produce una excepción
Anteriormente si han marcado **SI**, cierran la pantalla y empieza nueva partida, en caso de **NO** , solo cierra la pantalla.
Cierra conexiones del servidor

TABLERO

Constructor para crear el tablero con un for para crear los botones, refresca la interfaz gráfica y hace visible la ventana
En el método Poner se coloca la X o O en una posición de i= Fila, j= Columna, se pintara la X o O una vez pintado se desactiva el botón y refresca la interfaz
Otro método Poner que hace lo mismo pero se le pase un botón en vez de coordenadas
Cuando pulso un botón

- **Si esta activado**
    - Obtiene las coordenadas del botón, obteniendo el primer carácter del string y lo define como fila y lo mismo con la columna. Colocara la letra X o O del jugador en el botón, pone la posición y desactiva el tablero .
      Notifica que ha pulsado al hilo para que despierte y siga el juego

Activa el tablero e imprime es tu turno → Activo()
Desactiva el tablero e imprime espere a que juegue el otro → Desactivo()
Imprime hay 3 enraya → Ganar()
Imprime empate → Empate()
Comprueba si hay hueco en el tablero si hay algo vacio true y retorna a false si está lleno
Compara los botones el 1 con el 2 y el 2 con el 3 → Linea()
Nos muestra las combinaciones ganadoras del 3 en raya → Enraya()

POSICIÓN

Hilo Espera() → Espera hasta ser notificado o hasta que pasen 10 segundos, retorna el estado de 'notificado' , retorna a false si el hilo fue interrumpido y restablece el estado para el próximo uso
Hilo Despierto() → Se despertara el hilo para que siga el programa
Hilo CargaPosicion() → Nos cargara la posición del último movimiento
Hilo CargaLetra() → Nos carga el simbolo del jugador
Hilo Fila() → Nos devuelve la fila del ultimo movimiento
Hilo Columna() → Nos devuelve la columna del ultimo movimiento
Hilo Letra() → Carga la letra del jugador
Hilo OtraLetra() → Devuelve lo opuesto de lo que te de la función

SERVER

Crearemos una instancia de Server y comienza a escuchar conexiones
Lista para almacenar los socket de los clientes, se creará el ServerSocket en el puerto 12345 y nos mostrará un mensaje que el servidor espera conexiones

- Se creará un temporizador y tendrá un bucle para aceptar conexiones hasta que no salir sea falso, verifica si se han alcanzado el maximo de conexiones permitidas. se aceptará la conexión del cliente y se agrega el socket del cliente a la lista
    - **Si el numero de clientes es impar (solo un jugador conectado)**
        - Se crea un TimerTask para esperar al segundo jugador
    - **Si sigue siendo impar, no se ha conectado el segundo jugador**
        - Mostrará un mensaje indicando que jugara con la IA, habra un Socket falso para representar a la IA, se agregara el socket falso de la IA a la lista y se creara la sesión de juego contra la IA y se iniciara el juego
        - Se programa el temporizador para esperar al segundo jugador
    - **Si no se cumple la condición anterior**
        - Mensaje indicando la conexión del jugador 2, se cancelará el temporizador y se crea un nuevo Timer para futuras esperas, se obtiene el socket del jugador 1 y del jugador 2, tendra un mensaje indicando una nueva sesion de juego y se creara la sesion de juego entre los dos jugadores y se inicia el juego

Tendremos un método sincronizado para modificar la cantidad maxima de conexiones permitidas y tendremos la suma de cantidad proporcionada al maximo de conexiones permitidas → **La cantidad maxima son 4**

GAMESESSION

Constructor de la clase GameSession, asigna el socket del jugador 1 y del jugador 2, indica si esta jugando contra la IA o no y almacena la referencia al servidor
Metodo principal de la clase GameSession, ejecutando cuando se incia la sesion del juego, obtiene los flujos de entrada y salida de datos de los jugadores, con los turnos de los jugadores, envios el turno inicial a los jugadores

- **Si es distinto a la IA**
    - Jugará el jugador 2

Indica si es el turno del jugador 1
Hará un bucle principal del juego

- **Si es turno del jugador 1**
    - Espera el movimiento del jugador 1, lee el indicador de movimiento, la fila, la columna y marca el movimiento en el tablero
- **Si no se esta jugando contra la IA**
    - Envia e**l** movimiento del jugador 2, envía la fila del movimiento y la columna
- **Gestiona el movimiento del jugador 2 o de la IA, si no se está jugando contra la IA**
    - Espera el movimiento del jugador 2, lee el indicador de movimiento , la fila, la columna y marca el movimiento en el tablero. Indica al jugador 1 que hay un movimiento, envia la fila del movimiento y la columna
- **Si se esta jugando contra la IA, ejecuta la lógica para el movimiento de la IA**
    - Genera un movimiento aleatorio, obteniendo la fila del movimiento y la columna, marca el movimiento de la IA en el tablero. Indica al jugador 1 que hay un movimiento de la IA y envía la fila y la columna

Cambia el turno entre los jugadores

- **Verifica si el juego ha terminado**
    - Si hay tres en raya o está el tablero lleno, cierra las conexiones de los jugadores y sale del bucle principal del juego

MovimientoAleatorio() → Método para generar un movimiento aleatorio para la IA
Crea un objeto Random para generar números aleatorios, genera una fila y una columna entre (0 y 2), repite hasta encontrar una celda vacía y marca el movimiento de la IA en el tablero, retornando la fila y la columna del movimiento generado

CeldaVacia() → Método para verificar si una celda del tablero está vacía
Retorna a true si la celda está vacía, y false en caso contrario

CerrarConexiones() → Método para cerrar las conexiones de los jugadores
Suma el número correcto de conexiones dependiendo si se está jugando contra la IA o no

- **Si se juega contra la IA**
    - Incrementa el número de conexiones del servidor en 1
- **Sí no**
    - Incrementa el número de conexiones del servidor en 2

Cierra el socket del jugador 1

- **Si es distinto a la IA**
    - Si no juega contra la IA, cierra el socket del jugador 2

TableroLLeno() → Metodo para verificar si el tablero esta lleno
Retorna a false si encuentra al menos una celda vacia en el tablero

Retrona a true si no encuentra ninguna celda vacia en el tablero

HayTresEnRaya() → Metodo para verificar si hay tres en raya en el tablero

Verifica todas las posibles lineas de tres en el tablero(Horizontal, Vertical, Diagonal)
Retorna a true si hay tres en raya horizontalmente, verticalmente, diagonalmente
Retorna a false si no se encontró ninguna linea de tres