



Instituto Politécnico Nacional  
Escuela Superior de Cómputo



Aplicaciones para Comunicaciones en Red

## **Práctica 2: “Buscaminas”**

Alumnos:

Malagón Baeza Alan Adrian  
Martínez Chávez Jorge Alexis

Profesor:

Moreno Cervantes Axel Ernesto

Grupo: 6CM1

# 1. Introducción

Los *DatagramSocket* son el mecanismo de Java para la comunicación de red a través de UDP en lugar de TCP. Se puede usar Java DatagramSocket para enviar y recibir datagramas UDP.

UDP funciona un poco diferente a TCP. Cuando se envían datos a través de TCP, primero se crea una conexión. Una vez que se establece la conexión TCP, se garantiza que los datos lleguen al otro extremo, o le dirá que ocurrió un error.

Con UDP, sólo envía paquetes de datos (datagramas) a alguna dirección IP en la red. No tiene ninguna garantía de que los datos lleguen. Tampoco tiene garantía sobre el orden en que los paquetes UDP llegan al receptor. Esto significa que UDP tiene menos sobrecarga de protocolo (sin verificación de integridad de transmisión) que TCP.

UDP es apropiado para transferencias de datos donde no importa si un paquete se pierde en la transición.

## 1.1 Envío de datos a través de un DatagramSocket

Para enviar datos a través de Java, DatagramSocket primero debe crear un archivo DatagramPacket. Así es como se hace:

```
byte[] búfer = nuevo byte[65508];  
dirección InetAddress = InetAddress.getByName("jenkov.com");  
  
Paquete DatagramPacket = new DatagramPacket(  
    búfer, búfer.longitud, dirección, 9000);
```

El búfer de bytes (la matriz de bytes) son los datos que se enviarán en el datagrama UDP. La longitud del búfer anterior, 65508 bytes, es la cantidad máxima de datos que puede enviar en un solo paquete UDP.

La longitud dada al DatagramPacket constructor es la longitud de los datos en el búfer para enviar. Todos los datos en el búfer después de esa cantidad de datos se ignoran.

La InetAddress instancia contiene la dirección del nodo (por ejemplo, servidor) para enviar el paquete UDP. La InetAddress clase representa una dirección IP (Dirección de Internet). El getByName() método devuelve una InetAddress instancia con la dirección IP que coincide con el nombre de host dado.

El parámetro del puerto es el puerto UDP en el que el servidor que recibe los datos está escuchando. Los puertos UDP y TCP no son lo mismo. Una computadora puede tener diferentes procesos escuchando, por ejemplo, en el puerto 80 en UDP y en TCP al mismo tiempo. Para enviar datos se llama al método send()

## 1.2 Recepción de datos a través de un DatagramSocket

La recepción de datos a través de un DatagramSocket se realiza primero creando un DatagramPacket y luego recibiendo datos en él a través del DatagramSocket método de receive(). Aquí hay un ejemplo:

```
DatagramSocket datagramSocket = new DatagramSocket(80);

byte[] búfer = nuevo byte[10];
Paquete DatagramPacket = new DatagramPacket(buffer, buffer.length);

datagramSocket.receive(paquete);
```

Obsérvese cómo DatagramSocket se crea una instancia con el valor de parámetro 80 pasado a su constructor. Este parámetro es el puerto UDP en el DatagramSocket que se recibirán los paquetes UDP. Como se mencionó anteriormente, los puertos TCP y UDP no son iguales y, por lo tanto, no se superponen. Puede tener dos procesos diferentes escuchando en el puerto 80 TCP y UDP, sin ningún conflicto.

En segundo lugar, se crea un búfer de bytes y un DatagramPacket. La instancia DatagramPacket no tiene información sobre el nodo al que enviar datos, como lo hace al crear un nodo para enviar DatagramPacket datos. Esto se debe a que vamos a utilizar el DatagramPacket para recibir datos, no para enviarlos. Por lo tanto, no se necesita una dirección de destino.

Finalmente se llama al DatagramSocket método receive(). Este método bloquea hasta DatagramPacket recibe un paquete.

## **2. Desarrollo**

En esta práctica se desarrollará una aplicación donde el servidor mandará un buscaminas al cliente, permitiéndole seleccionar diferentes niveles. Después de seleccionar el nivel correspondiente el cliente podrá jugar el buscaminas, al finalizar el juego se guarda en un archivo de texto el puntaje y tiempo del jugador.

A continuación se explicará las clases que se implementan en Java para esta práctica las cuales son: Clase Server, MainApplication, MainController, Objeto, Buscaminas, BuscaminasControllers, Record, RecordController y ButtonInfo.

### **2.1 Clase Server**

En la clase ServerSoup se crea una instancia de la clase DatagramSocket que toma como parámetro el número de puerto al cual deseamos acceder, luego se instancia la clase InetAddress con el host deseado. Posteriormente se crea un objeto de la clase DatagramPacket con un tamaño de 65535 bytes, con el método receive() el servidor queda en espera del envío de paquetes por parte del cliente. Cuando se recibe el primer paquete se utiliza el método connect() con el número de puerto y dirección para filtrar la entrega de paquetes de cada cliente, es decir, es una forma de evitar el envío o la recepción de paquetes hacia o desde otras direcciones. A continuación se muestra el código en donde el servidor se inicializa y se espera la recepción de paquetes.

Ya después del establecimiento del servidor, se inicia la comunicación bidireccional por parte del cliente y el servidor. En este caso el servidor recibe el nombre del cliente, luego envía los niveles disponibles, recibe la selección de los niveles disponibles (Principiante, Intermedio, Experto) y posteriormente envía las dimensiones del tablero y la cantidad de minas a colocar.

Finalmente, cuando el cliente haya terminado el juego se recibe el tiempo y puntaje obtenido para guardarlo en un archivo de texto (Puntajes.txt)



```

        casillasW = 16;
        casillasH = 16;
        minas = 40;
        break;
    case 3:
        casillasW = 16;
        casillasH = 30;
        minas = 99;
        break;
    }
    o5 = new Objeto(casillasW+","+casillasH+","+minas);

    System.out.println("Configuración de juego enviada: \n" +
        "Casillas Ancho: " + casillasW + "\n" +
        "Casillas Alto: " + casillasH + "\n" +
        "Minas: " + casillasH);
    oos.writeObject(o5);
    oos.flush();
    b = baos.toByteArray();
    DatagramPacket p5 = new DatagramPacket(b, b.length,
p.getAddress(), p.getPort());
    s.send(p5);

    Objeto o3 = new Objeto("Inicio del buscaminas...");
    oos.writeObject(o3);
    oos.flush();
    b = baos.toByteArray();
    DatagramPacket p3 = new DatagramPacket(b, b.length,
p.getAddress(), p.getPort());
    s.send(p3);
    System.out.println("¡Inicio del juego!");

    s.receive(p);
    Objeto o8 = (Objeto) ois.readObject();
    String points = o8.getStr();
    System.out.println("Puntaje: " + points);

    s.receive(p);
    Objeto o4 = (Objeto) ois.readObject();
    time = o4.getStr();
    System.out.println("Tiempo: " + time + " seg");

    s.receive(p);
    Objeto o6 = (Objeto) ois.readObject();
    boolean gano = Boolean.parseBoolean(o6.getStr());
    System.out.println(" gano "+gano);

    BufferedWriter bw = null;
    FileWriter fw = null;
    File file = new File("Puntajes.txt");
    if (!file.exists()) {
        file.createNewFile();
    }
    fw = new FileWriter(file.getAbsolutePath(), true);
    bw = new BufferedWriter(fw);

```

```

        String data = "+ Jugador: " + nickName + "    Direccion: " + p.getAddress() + "    Puerto: " + p.getPort()
                        + "    Nivel: " + nivel + "    Puntos: " + points
                        + "    Tiempo: " + time + " seg\n";

        bw.write(data);
        System.out.println("Información agregada: " + file.getAbsolutePath());
        bw.close();
        fw.close();

        if(gano) {
            ObjectMapper objectMapper = new ObjectMapper();
            List<Record> records = objectMapper.readValue(new File("records.json"), new TypeReference<List<Record>>() {
            });

            Record record = new Record(nickName, p.getAddress().toString(), p.getPort(), nivel, Integer.parseInt(points), time);

            records.add(record);

            objectMapper.writerWithDefaultPrettyPrinter().writeValue(new File("records.json"), records);
        }

        s.disconnect();
    } //while
} catch (Exception e) {
    e.printStackTrace();
} //catch
}
}

```

## 2.2 Clase Buscaminas

La clase Buscaminas es el cliente que hace uso de la clase BuscaminasControler para crear la lógica del juego y desplegar el buscaminas como interfaz gráfica.

Para el caso del cliente al igual que el servidor se crea una instancia de la clase DatagramSocket, también un objeto de la clase InetAddress con el host deseado. Posteriormente para el envío de paquetes se crean instancias de las clases ByteArrayOutputStream y ObjectOutputStream para establecer el flujo de salida y escribir los objetos que se desean enviar, con el método toByteArray() se consigue que el objeto que se desea enviar se convierta en un arreglo de bytes. Después de este proceso se crea un DatagramPacket con el arreglo de bytes a ser enviado, el tamaño de arreglo, la dirección destino y el número de puerto correspondiente, con el método send() el cliente puede enviar el paquete al servidor. En seguida se muestra el código correspondiente donde se puede observar la inicialización del cliente y el envío del nombre del cliente.

Como se observa luego del establecimiento del cliente, se envía el nombre del cliente, después se reciben los niveles disponibles, se envía la selección de niveles correspondiente y finalmente se recibe la configuración del juego. ¡Así ya puede comenzar a jugar!

Finalmente, cuando el cliente inicia el juego se toma una marca de tiempo personal y se cuentan los puntos obtenidos, cada palabra casilla desbloqueada equivale a las minas que tiene alrededor. El puntaje y tiempo obtenidos se envían al servidor.

```
public class Buscaminas extends Application {

    static String nickName;
    static String nivel = "Principiante";

    static int niveln = 1;
    static int casillasW = 9;
    static int casillasH = 9;
    static int minas = 10;

    static String[] configuracion;
    static int points = 0;
    static int finalPoints = 0;

    @Override
    public void start(Stage stage) throws IOException {
        FXMLLoader fxmlLoader = new
FXMLLoader(Buscaminas.class.getResource("main.fxml"));

        Scene scene = new Scene(fxmlLoader.load());
        stage.setTitle("Inicio");
        stage.setScene(scene);
        stage.show();
    }
}
```



```

    }

    public static void main(String[] args) {
        launch();
    }

    public static void iniciarJuego() {
        final String host = "127.0.0.1";
        final int port = 8009;

        try{
            InetAddress dst = InetAddress.getByName(host);
            DatagramSocket cl = new DatagramSocket();
            System.out.println("Cliente iniciado "+host+" : "+port);

            Objeto obj = new Objeto(nickName);
            ByteArrayOutputStream baos= new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(obj);
            oos.flush();

            byte[] b = baos.toByteArray();
            DatagramPacket p = new DatagramPacket(b,b.length,dst,port);
            cl.send(p);

            System.out.println("Nickname del jugador enviado:
            "+obj.getStr());

            System.out.println("Esperando recibir nivel...");
            DatagramPacket p1 = new DatagramPacket(new byte[65535],65535);
            cl.receive(p1);

            ObjectInputStream ois = new ObjectInputStream(new
            ByteArrayInputStream(p1.getData()));

```

```

Objeto o1 = (Objeto)ois.readObject();
System.out.println(o1.getStr());

int nv1 = 1;
niveln = nv1;
if(nivel.equalsIgnoreCase("principiante")) {
    nv1 = 1;
}
else if(nivel.equalsIgnoreCase("intermedio")) {
    nv1 = 2;
}
else if (nivel.equalsIgnoreCase("experto")) {
    nv1 = 3;
}
Objeto obj1 = new Objeto(nv1+"");
oos.writeObject(obj1);
oos.flush();
b = baos.toByteArray();
DatagramPacket p2 = new DatagramPacket(b,b.length,dst,port);
cl.send(p2);
System.out.println("Nivel enviado: "+nivel);

cl.receive(p1);

                                                                    configuracion =
((Objeto)ois.readObject()).getStr().split(",");
System.out.println(configuracion);
casillasW = Integer.parseInt(configuracion[0]);
casillasH = Integer.parseInt(configuracion[1]);
minas = Integer.parseInt(configuracion[2]);

cl.receive(p1);
Objeto obj2 = (Objeto)ois.readObject();

```

```

        System.out.println(obj2.getStr());

        long startTime = System.currentTimeMillis();

        FXMLLoader fxmlLoader = new
FXMLLoader(Buscaminas.class.getResource("buscaminas.fxml"));

        Stage stage = new Stage();

        Scene scene = new Scene(fxmlLoader.load());

        stage.setTitle("Buscaminas");

        stage.setScene(scene);

        stage.showAndWait();

        long stopTime = System.currentTimeMillis();

        double elapsedTime = (stopTime - startTime)/1000.0;

        System.out.println("Marca de tiempo: " + elapsedTime+ " seg");

        Objeto obj4 = new Objeto(Buscaminas.points+"");
        oos.writeObject(obj4);
        oos.flush();

        b = baos.toByteArray();
        DatagramPacket p4 = new DatagramPacket(b,b.length,dst,port);
        cl.send(p4);
        System.out.println("Puntaje enviada al servidor");

        Objeto obj3 = new Objeto(elapsedTime+"");
        oos.writeObject(obj3);
        oos.flush();

        b = baos.toByteArray();
        DatagramPacket p3 = new DatagramPacket(b,b.length,dst,port);
        cl.send(p3);
        System.out.println("Marca de tiempo enviada al servidor");

        Objeto obj5 = new
Objeto(Boolean.toString(points==finalPoints));

        oos.writeObject(obj5);

```

```

        oos.flush();

        b = baos.toByteArray();

        DatagramPacket p5 = new DatagramPacket(b,b.length,dst,port);

        cl.send(p5);

        System.out.println("Si es ganador enviado");

        cl.close();

    }catch(Exception exception){

        exception.printStackTrace();

    }

}
}

```

## 2.3 Clase MainController

```

public class MainController implements Initializable {

    @FXML

    private TextField nickTxt;

    @FXML

    private ComboBox<String> comboBox;

    @FXML

    void jugar(MouseEvent event) {

        String nick = nickTxt.getText();

        if(nick.isEmpty() || nick.isBlank()){

            Alert a = new Alert(Alert.AlertType.INFORMATION);

            a.setHeaderText(null);

            a.setContentText("Ingresa un nick");

            a.show();

            return;

        } else if (nickTxt.getLength() > 12){

```

```

        Alert a = new Alert(Alert.AlertType.INFORMATION);
        a.setHeaderText(null);
        a.setContentText("El nick debe tener 12 caracteres o menos");
        a.show();

        return;
    }

    Buscaminas.nickname = nick;

    Stage stage = (Stage) nickTxt.getScene().getWindow();
    stage.close();

    Buscaminas.iniciarJuego();

}

@Override
public void initialize(URL url, ResourceBundle resourceBundle) {
    comboBox.getItems().addAll("Principiante", "Intermedio", "Experto");
    comboBox.setOnAction((ActionEvent ev) -> {
        Buscaminas.nivel =
comboBox.getSelectionModel().getSelectedItem().toString();

    });
}
}

```

## 2.4 Clase Objeto

La clase objeto nos permitió hacer el envío y recepción de cadenas de texto durante la comunicación entre el cliente y el servidor. Como se puede observar se tiene un constructor, un getter y un setter.

```

public class Objeto implements Serializable {
    private String str;

```

```

public Objeto(String str) {
    this.str = str;
}

public void setStr(String str) {
    this.str = str;
}

public String getStr() {
    return str;
}
}

```

## 2.5 Clase BuscaminasController

Esta clase es la encargada de realizar la lógica del juego buscaminas entre los métodos fundamentales con los que cuenta esta clase se encuentran colocarMinas() el cual coloca minas en posiciones aleatorias en el tablero cuidado que no haya más de 3 a su alrededor, crearBuscaminar() llena el tablero con los números contando la cantidad de minas alrededor de cada casilla, revelarCasilla() para mostrar el contenido de la casilla; si es una mina finaliza el juego, si es una casilla sin minas alrededor revela las demás casillas a su alrededor sin minas y se detiene cuando encuentra alguna casilla con alguna mina alrededor o cuando encuentra una mina, revelarMinas() para mostrar la ubicación de todas las minas si el jugador pierde y revelarCasillasAlrededor() junto revelarCasilla() de forma recursiva para desbloquear las adyacentes vacías anteriormente mencionado.

En esta clase existen otros métodos que contribuyen a la lógica y a la presentación de la aplicación como styleButton(), estaBloqueada() y checarMina().

```

public class BuscaminasController implements Initializable {
    @FXML
    private HBox content;
}

```

```

@FXML
private Label flagsLbl;

@FXML
private AnchorPane ap;

private int i1, j1;

static int casillasW = Buscaminas.casillasW;
static int casillasH = Buscaminas.casillasH;
static int minas = Buscaminas.minas;

private final int[][] board = new int[casillasW][casillasH];
        private final Button[][] boardButtons = new
Button[casillasW][casillasH];

private int points = 0;
private int finalPoints = 0;

private static String greenColor="#A3D14A";
private static String dirtColor="#E5C29F";
private static String mudColor="#D7B899";
private static String limeColor="#A9D751";

@FXML
void logros(MouseEvent event) throws IOException {
        FXMLLoader fxmllLoader = new
FXMLLoader(Buscaminas.class.getResource("records.fxml"));

        Stage stage = new Stage();

        Scene scene = new Scene(fxmllLoader.load());

        stage.setTitle("Logros");

        stage.setScene(scene);

        stage.show();

}

```

```

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

    crearBuscaminas();

    flagsLbl.setText(minas+"");

    VBox rows = new VBox();

    for (int i = 0; i < board.length; i++) {

        HBox columns = new HBox();

        for (int j = 0; j < board[i].length; j++) {

            boardButtons[i][j] = new Button();

            if(i%2 == 0){

                if(j%2==0){

                    boardButtons[i][j].setStyle("-fx-background-color:
" + greenColor + " ;"+
                    "-fx-background-radius: 0;");

                }else{

                    boardButtons[i][j].setStyle("-fx-background-color:
" + limeColor + " ;"+
                    "-fx-background-radius: 0;");

                }

            }else{

                if(j%2==0){

                    boardButtons[i][j].setStyle("-fx-background-color:
" + limeColor + " ;"+
                    "-fx-background-radius: 0;");

                }else{

                    boardButtons[i][j].setStyle("-fx-background-color:
" + greenColor + " ;"+
                    "-fx-background-radius: 0;");

                }

            }

            boardButtons[i][j].setFont(Font.font(14));

            boardButtons[i][j].setTextFill(Color.BLACK);

```



```

        boardButtons[i][j].setPrefSize(32, 32);

        boardButtons[i][j].setUserData(new ButtonInfo(i, j));

        boardButtons[i][j].setOnMouseClicked(buttonClick);

        boardButtons[i][j].setMaxWidth(32);

        boardButtons[i][j].setMinWidth(32);

        boardButtons[i][j].setMaxHeight(32);

        boardButtons[i][j].setMinHeight(32);

        columns.getChildren().add(boardButtons[i][j]);

    }

    rows.getChildren().add(columns);

}

content.getChildren().add(0, rows);

content.setMaxHeight(Control.USE_COMPUTED_SIZE);

Platform.runLater(() -> {

    Stage stage = (Stage) ap.getScene().getWindow();

    double topBarSize = 89;

    double w = boardButtons[0][0].getPrefWidth();

    double finalW = (w * casillasH)+w/2;

    stage.setWidth(finalW);

    double h = boardButtons[0][0].getPrefHeight();

    double finalH = (h * casillasW)+topBarSize;

    stage.setHeight(finalH);

});

for (int[] x : board)

{

    for (int y : x)

    {

        System.out.print(y + " ");

    }

    System.out.println();
}

```

```

    }

}

private void crearBuscaminas() {

    colocarMinas(minas);

    finalPoints += minas*9;

    for (int i=0;i<casillasW;i++){

        for (int j=0;j<casillasH;j++){

            if (board[i][j] == 0){

                int minas = checarMina(i+1,j)+

                    checarMina(i-1,j)+

                    checarMina(i,j+1)+

                    checarMina(i,j-1)+

                    checarMina(i-1,j+1)+

                    checarMina(i+1,j+1)+

                    checarMina(i-1,j-1)+

                    checarMina(i+1,j-1);

                board[i][j] = minas;

                finalPoints+=minas;

            }

        }

    }

    Buscaminas.finalPoints = finalPoints;

}

private List<ButtonInfo> botonesMinas = new ArrayList<>();

private void colocarMinas(int minas){

    if(minas==0) return;

    int im = (int) Math.floor(Math.random() * (casillasW));

    int hm = (int) Math.floor(Math.random() * (casillasH));

    if(board[im][hm] == -1)

        colocarMinas(minas);
}

```

```

else{

    int minasN = checarMina(im+1,hm)+

        checarMina(im-1,hm)+

        checarMina(im,hm+1)+

        checarMina(im,hm-1)+

        checarMina(im-1,hm+1)+

        checarMina(im+1,hm+1)+

        checarMina(im-1,hm-1)+

        checarMina(im+1,hm-1);

    if(minasN<4) {

        board[im][hm] = -1;

        ButtonInfo buttonInfo = new ButtonInfo(im,hm);

        botonesMinas.add(buttonInfo);

        colocarMinas(minas - 1);

    }else

        colocarMinas(minas);

}

}

private void revelarCasilla(int i, int j){

    int n = board[i][j];

    if(n == -1){

        styleButton(i,j,"mine",false);

        revelarMinas();

        Buscaminas.points = points;

        Alert alert = new Alert(Alert.AlertType.INFORMATION);

        alert.setHeaderText("¡Perdiste!");

        alert.showAndWait();

        Platform.exit();

        return;
    }
}

```

```

    }

    points+=n;

    styleButton(i,j,n+"",false);

    if(n == 0)

        revelarCasillasAlrededor(i,j);

}

private void revelarMinas() {

    Collections.shuffle(botonesMinas);

    botonesMinas.forEach(mina -> {

        int i = mina.getI();

        int j = mina.getJ();

        boardButtons[i][j].setGraphic(null);

        boardButtons[i][j].setText("●");

        Random random = new Random();

        boardButtons[i][j].setStyle("-fx-background-color: " +
String.format("#%06x", random.nextInt(0xffffffff + 1)) + ";" +
        "-fx-background-radius: 0;");

    });

}

private void revelarCasillasAlrededor(int i,int j){

    if (i > 0 && estaBloqueada(i - 1, j)) revelarCasilla(i - 1, j);

    if (i < casillasW-1 && estaBloqueada(i + 1, j)) revelarCasilla(i +
1, j);

    if (j > 0 && estaBloqueada(i, j - 1)) revelarCasilla(i, j - 1);

    if (j < casillasW-1 && estaBloqueada(i, j + 1)) revelarCasilla(i,
j + 1);

    if (i > 0 && j > 0 && estaBloqueada(i - 1, j - 1))
revelarCasilla(i - 1, j - 1);

    if (i < casillasW-1 && j < casillasH-1 && estaBloqueada(i + 1, j +
1)) revelarCasilla(i + 1, j + 1);

    if (i > 0 && j < casillasH-1 && estaBloqueada(i - 1, j + 1))
revelarCasilla(i - 1, j + 1);

```

```

        if (i < casillasW-1 && j > 0 && estaBloqueada(i + 1, j - 1))
revelarCasilla(i + 1, j - 1);

    }

    private boolean estaBloqueada(int i,int j){

        if(boardButtons[i][j].getGraphic() == null && board[i][j] != -1 &&
board[i][j] !=9)

            return true;

        return false;

    }

    private int checarMina(int i, int j){

        if(i < 0 || i > casillasW-1 || j < 0 || j > casillasH-1) return 0;

        if(board[i][j]==-1) return(1);

        return 0;

    }

    private void styleButton( int i, int j,String url,boolean isGreen) {

        String light;

        String dark;

        if(isGreen){

            light=limeColor;

            dark=greenColor;

        }else{

            light=dirtColor;

            dark=mudColor;

        }

        if(i%2 == 0){

            if(j%2==0){

                boardButtons[i][j].setStyle("-fx-background-color: " +
dark + ";" +

                "-fx-background-radius: 0;");

```

```

        }else{
            boardButtons[i][j].setStyle("-fx-background-color: " +
light + ";" +
            "-fx-background-radius: 0;");
        }
    }else{
        if(j%2==0){
            boardButtons[i][j].setStyle("-fx-background-color: " +
light + ";" +
            "-fx-background-radius: 0;");
        }else{
            boardButtons[i][j].setStyle("-fx-background-color: " +
dark + ";" +
            "-fx-background-radius: 0;");
        }
    }
    if(url.equals("")) return;
    Button button = boardButtons[i][j];

    Image image = new
Image(getClass().getResourceAsStream(url+".png"));

    ImageView imageView = new ImageView(image);
    imageView.setPreserveRatio(true);
    imageView.fitWidthProperty().bind(button.widthProperty());
    imageView.fitHeightProperty().bind(button.heightProperty());

    button.setGraphic(imageView);
}

private final EventHandler<MouseEvent> buttonClick = new
EventHandler<MouseEvent>() {
    @Override
    public void handle(MouseEvent mouseEvent) {

```

```

        Button button = (Button) mouseEvent.getTarget();

        button.setText("");

        ButtonInfo buttonInfo = (ButtonInfo) button.getUserData();

        i1 = buttonInfo.getI();

        j1 = buttonInfo.getJ();

        int casillaSeleccionada = board[i1][j1];

        if(mouseEvent.getButton() == MouseButton.PRIMARY){

            if(casillaSeleccionada != 9){

                revelarCasilla(i1,j1);

            }

        } else if(mouseEvent.getButton() == MouseButton.SECONDARY) {

            if(casillaSeleccionada != 9 &&
boardButtons[i1][j1].getGraphic() == null){

                if(casillaSeleccionada == -1) points+=9;

                styleButton(i1,j1,"flag_icon",true);

                board[i1][j1]=9;

flagsLbl.setText(Integer.parseInt(flagsLbl.getText())-1+"");

            }

            else if(casillaSeleccionada == 9){

                points-=9;

flagsLbl.setText(Integer.parseInt(flagsLbl.getText())+1+"");

                button.setGraphic(null);

                board[i1][j1]=0;

            }

        }

        if(points==finalPoints){

            Buscaminas.points=points;

```

```

        Alert alert = new Alert(Alert.AlertType.INFORMATION);

        alert.setHeaderText("¡Ganaste!");

        alert.showAndWait();

        Platform.exit();

    }

}

};

}

```

## 2.6 Clase Record

La clase Record nos permite guardar los records en formato JSONr. Como se puede observar se tiene un constructor, un getter y un setter.

```

public class Record {

    private String jugador;

    private String direccion;

    private int puerto;

    private int nivel;

    private int puntos;

    private String tiempo;

    public Record(String jugador, String direccion, int puerto, int nivel,
int puntos, String tiempo) {

        this.jugador = jugador;

        this.direccion = direccion;

        this.puerto = puerto;

        this.nivel = nivel;

        this.puntos = puntos;

        this.tiempo = tiempo;

    }
}

```



```
public String getJugador() {  
    return jugador;  
}  
  
public void setJugador(String jugador) {  
    this.jugador = jugador;  
}  
  
public String getDireccion() {  
    return direccion;  
}  
  
public void setDireccion(String direccion) {  
    this.direccion = direccion;  
}  
  
public int getPuerto() {  
    return puerto;  
}  
  
public void setPuerto(int puerto) {  
    this.puerto = puerto;  
}  
  
public int getNivel() {  
    return nivel;  
}  
  
public void setNivel(int nivel) {  
    this.nivel = nivel;  
}
```

```

public int getPuntos() {
    return puntos;
}

public void setPuntos(int puntos) {
    this.puntos = puntos;
}

public String getTiempo() {
    return tiempo;
}

public Record() {
}

public void setTiempo(String tiempo) {
    this.tiempo = tiempo;
}
}

```

## 2.7 Clase RecordController

```

public class RecordController implements Initializable {

    @FXML
    private Label playersLbl;

    @FXML
    private Label timeLbl;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        playersLbl.setText("");
    }
}

```

```

        timeLbl.setText("");

        ObjectMapper objectMapper = new ObjectMapper();

        List<Record> records = null;

        try {

            records = objectMapper.readValue(new File("records.json"), new
TypeReference<List<Record>>() {});

        } catch (IOException e) {

            throw new RuntimeException(e);

        }

        System.out.println(records);

        records.removeIf(r -> r.getNivel() != Buscaminas.niveln);

                                                                    Collections.sort(records,
Comparator.comparing(Record::getTiempo));

        int i = 0;

        for (Record r : records) {

            if(i==6) return;

            playersLbl.setText(playersLbl.getText()+r.getJugador()+"\n");

            timeLbl.setText(timeLbl.getText()+r.getTiempo()+"\n");

            i++;

        }

    }

}

```

## 2.8 Clase ButtonInfo

```

public class ButtonInfo {
    // Coordinates
    private int i,j;

    public ButtonInfo(int i, int j) {
        this.i = i;
        this.j = j;
    }

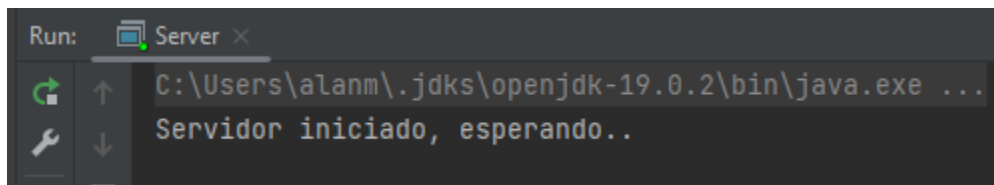
    public int getI() {
        return i;
    }
}

```

```
public int getJ() {  
    return j;  
}  
}
```

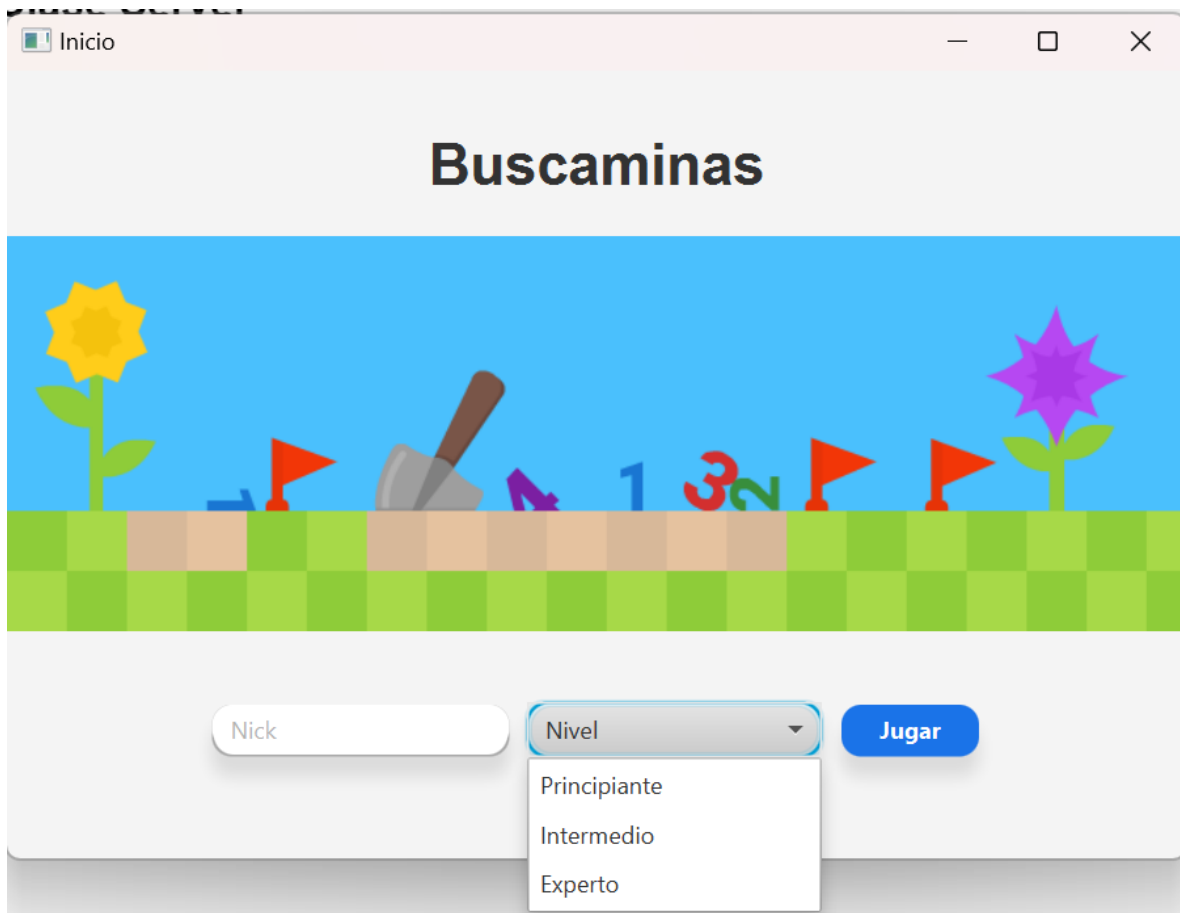
### 3. Pruebas

Primeramente se ejecuta la clase Server, la cual desplegará el siguiente mensaje



```
Run: Server x  
C:\Users\alanm\.jdk\openjdk-19.0.2\bin\java.exe ...  
Servidor iniciado, esperando..
```

Luego se ejecuta la clase Buscaminas (Cliente), se le indicará al usuario en qué puerto y dirección se inició el cliente. El jugador debe introducir su nombre y seleccionar un nivel para enviarlos al servidor.



```
Buscaminas x
C:\Users\alanm\.jdk\openjdk-19.0.2\bin\java.exe ...
Cliente iniciado 127.0.0.1 : 8009
Nickname del jugador enviado: Alan
Esperando recibir nivel...
Seleccione un nivel
Principiante
Intermedio
Experto
Nivel enviado: Principiante
[Ljava.lang.String;@596a18d2
Inicio del buscaminas...
-1 -1 2 2 -1 1 0 0 0
2 2 2 -1 2 1 0 0 0
0 0 1 1 1 0 0 0 0
1 1 0 0 0 0 0 0 0
-1 1 1 1 1 0 0 0 0
2 2 2 -1 1 0 0 0 0
1 -1 3 3 3 1 0 0 0
2 2 3 -1 -1 1 0 0 0
1 -1 2 2 2 1 0 0 0
```

El servidor envía las dimensiones de la matriz del juego y el número de minas según el nivel seleccionado.

```
Run: Server x
Inet address /127.0.0.1
Nickname cliente: Alan
Niveles disponibles enviados.
Configuración de juego enviada:
Casillas Ancho: 9
Casillas Alto: 9
Minas: 9
¡Inicio del juego!
```

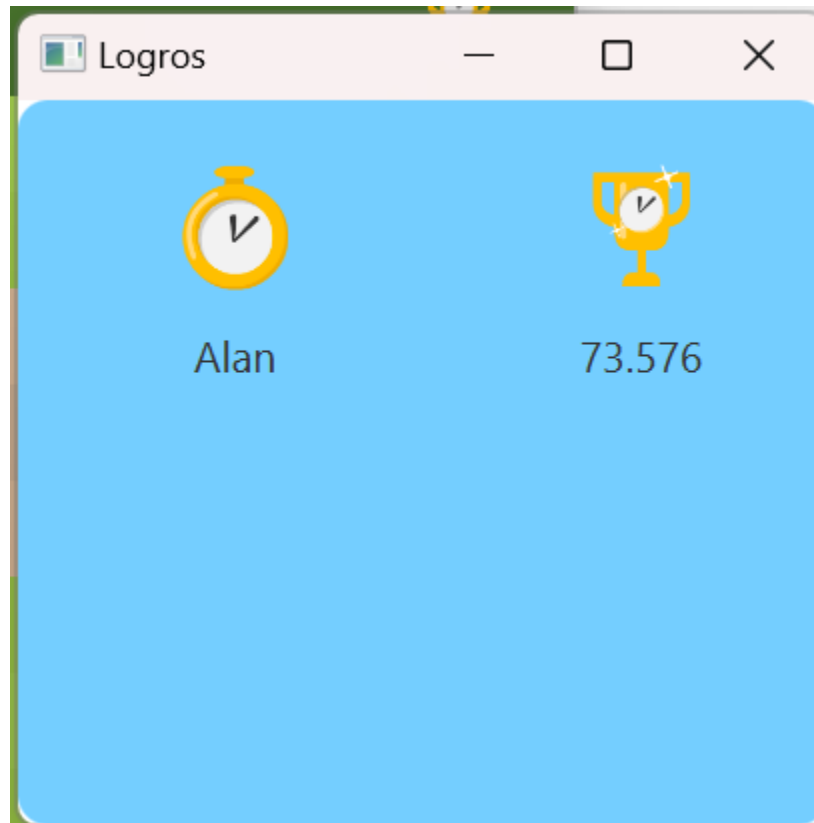
Al cliente se le muestra la aplicación, en la parte superior izquierda tiene la cantidad de bandera que deberá poner (número de minas) y a la derecha un botón de trofeo donde podrá ver los records actuales de ese nivel.



Para desbloquear casillas deberá dar clic izquierdo sobre los cuadrados verdes, y para colocar banderas y quitarlas deberá dar clic derecho



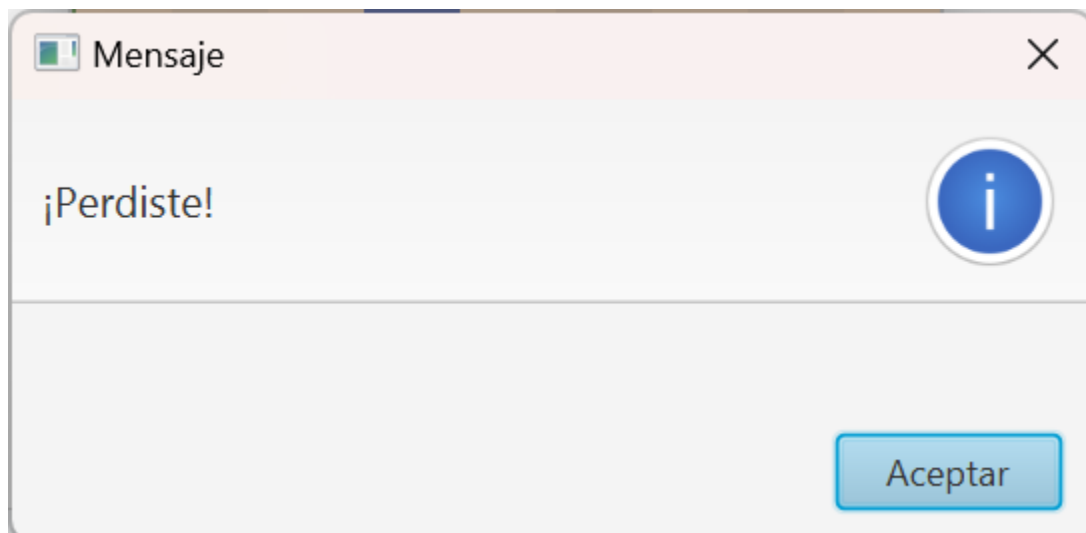
Ventana de logros actuales dependiendo el nivel en el que se encuentre.





Si da clic sobre una mina se desbloquearan todas las minas y mandará mensaje informando que perdió.





Al terminar el juego el jugador puede observar su marca de tiempo, es decir, el tiempo que estuvo jugando el buscaminas.

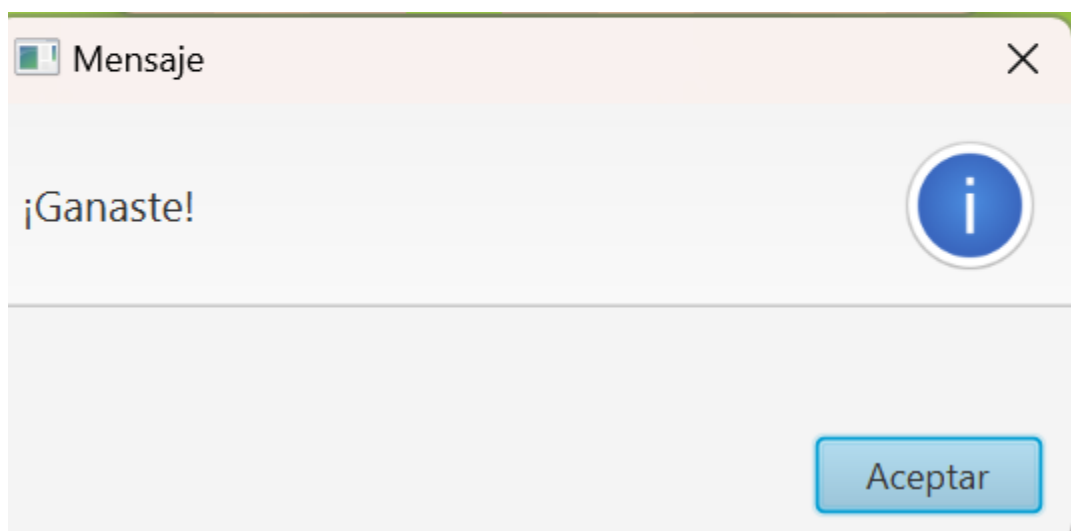
```
Marca de tiempo: 134.43 seg
Puntaje enviada al servidor
Marca de tiempo enviada al servidor
Si es ganador enviado
```

El servidor recibe el puntaje y la marca de tiempo del jugador, y escribe estos datos en el archivo de texto Puntajes.txt

```
Puntaje: 57
Tiempo: 134.43 seg
gano false
Información agregada: C:\Users\alanm\Desktop\Redes2\Buscaminas\Puntajes.txt
```

	Jugador	Direccion	Puerto	Nivel	Puntos	Tiempo
1	+ Jugador: Alan	Direccion: /127.0.0.1	Puerto: 55788	Nivel: 1	Puntos: 0	Tiempo: 15.655 seg
2	+ Jugador: Alan	Direccion: /127.0.0.1	Puerto: 52223	Nivel: 1	Puntos: 146	Tiempo: 53.864 seg
3						

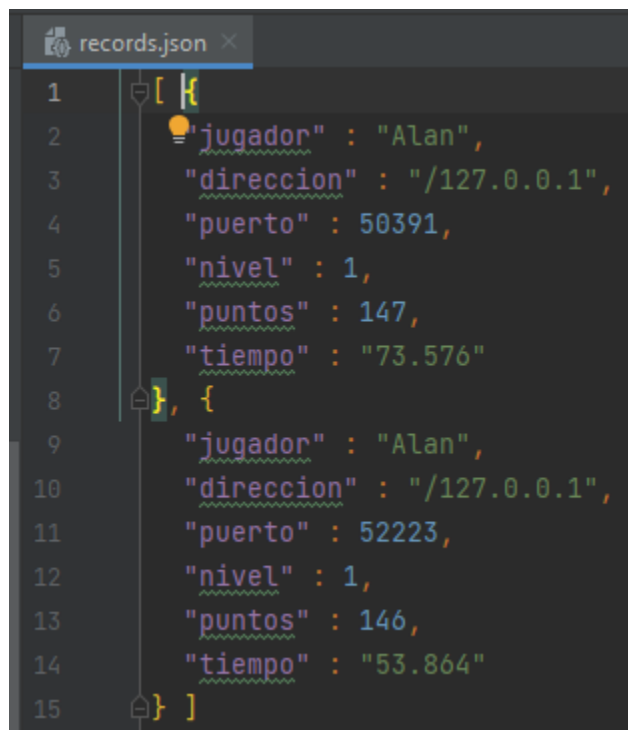
En caso de ganar se manda mensaje informando que ganó.



```
Puntaje: 146
Tiempo: 53.864 seg
gano true
Información agregada: C:\Users\alanm\Desktop\Redes2\Buscaminas\Puntajes.txt
```

```
Marca de tiempo: 53.864 seg
Puntaje enviada al servidor
Marca de tiempo enviada al servidor
Si es ganador enviado
```

Solo si se gana la partida se agregan los datos también al archivo records.json



```
1  [ {
2    "jugador" : "Alan",
3    "direccion" : "/127.0.0.1",
4    "puerto" : 50391,
5    "nivel" : 1,
6    "puntos" : 147,
7    "tiempo" : "73.576"
8  }, {
9    "jugador" : "Alan",
10   "direccion" : "/127.0.0.1",
11   "puerto" : 52223,
12   "nivel" : 1,
13   "puntos" : 146,
14   "tiempo" : "53.864"
15 } ]
```

## 4. Conclusiones

Con el desarrollo de esta práctica nos quedó más claro el uso de datagramas, UDP es bastante diferente a TCP. Además, es importante comprender que la falta de sobrecarga puede hacer que sea significativamente más rápido que TCP.

Lo que pudimos observar es que la creación de aplicaciones UDP es muy similar a la creación de un sistema TCP; la única diferencia es que no establecemos una conexión punto a punto entre un cliente y un servidor. La configuración también es muy sencilla. Ya que Java cuenta con soporte de red incorporado para UDP, que es parte del paquete `java.net`. Por lo tanto, para realizar operaciones de red sobre UDP, solo lo que necesitamos es importar las clases: `java.net.DatagramSocket` y `java.net.DatagramPacket`.

Finalmente comprobamos que en cualquier programa que se ocupen datagramas los pasos para enviar y recibir datos a través de UDP son los mismos: Para enviar un paquete a través de UDP, debemos saber 4 cosas, el mensaje a enviar, su longitud, la dirección IP de destino, el puerto en el que está escuchando el destino. Una vez que sabemos todas estas cosas, podemos crear el objeto socket para transportar los paquetes. Luego invocamos la llamada `send()/receive()` para enviar/recibir paquetes. Y por último se extraen los datos del paquete recibido.

## Bibliografía

- Tanenbaum, Andrew. "Redes de Computadoras". Cuarta Edición, Pearson Prentice Hall, 2003.
- Jenkov, J. (2014). Java Networking: UDP DatagramSocket. [jenkov.com](http://tutorials.jenkov.com/java-networking/udp-datagram-sockets.html). Retrieved 20 March 2022, from <http://tutorials.jenkov.com/java-networking/udp-datagram-sockets.html>.
- A typical UDP socket session. [ibm.com](https://www.ibm.com/docs/en/zos/2.2.0?topic=services-typical-udp-socket-session). (2022). Retrieved 21 March 2022, from <https://www.ibm.com/docs/en/zos/2.2.0?topic=services-typical-udp-socket-session>.