

# Reporte practicas Teoría Computacional

## **ESCUELA SUPERIOR DE CÓMPUTO**

Colin Heredia Luis Antonio

Juárez Martínez Genaro

Grupo: 2CM5

8 de mayo de 2020

# Introducción práctica (Ajedrez NFD)

En esta práctica se utiliza un autómata finito no determinista NFD basado el tablero de ajedrez 4x4. Se colocarán dos piezas en el tablero en diferentes casillas iniciales y su objetivo es llegar a una casilla ganadora, para la pieza 1 su estado inicial sera la casilla 1 y su casilla final sera la 16. Para la pieza 2 su estado inicial sera 4 y su casilla ganadora 13. Para cada pieza se le dará una cadena en formato "BN" las cuales determinan el color de la casilla por la cual se quiere avanzar. Esta cadena sera evaluada por el autómata y dará todas las rutas posibles que puede tomar para llegar a su objetivo, así mismo, mostrará las rutas que no harán llegar a su objetivo. Estos caminos se animaran para ver el recorrido en el tablero.

## Código main

Practica4Chess.java

```
package practica4.chess;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;

/**
 *
 * @author colintony
 */
public class Practica4Chess extends Application {

    //para iniciar la ventana principal de JAVAFX
    @Override
    public void start(Stage stage) throws Exception {
        Parent root = FXMLLoader.load(getClass().getResource("Views/FXMLMain.fxml"));
        // cargamos nuestra vista

        Scene scene = new Scene(root); // cargamos la scene
        // se pone el stage y se muestra al usuario
        stage.setScene(scene);
        stage.show();
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        launch(args);
    }
}
```

## AutomataFND.java

```
package practica4.chess.Controllers;

import java.io.IOException;
import java.util.ArrayList;
import javafx.scene.control.TextArea;
import practica4.chess.Models.CaminosValidos;
import practica4.chess.Models.EstadoActual;
import practica4.chess.Models.Estados;
import practica4.chess.Models.TablaTR;

/**
 *
 * @author colin
 * Esta es la clase mas importante
 * ya que lleva el algoritmo del calculo de las rutas
 * usando la tabla y funcion de transicion.
 */
public class AutomataFND {
    private String cadena;
    private TablaTR tablaTransicion;
    private EstadoActual qA;
    private Estados conjuntosEst;
    private ArchivoRutas archivoRutas;
    private ArchivoRutas archivoRutasValidas;
    private ArrayList<String> caminos;
    private CaminosValidos caminoValido;
    private ArrayList<CaminosValidos> caminosValidos;

    // constructor
    // se debe inicializar con un estado actual, y su estado valido
    // tambien necesitamos el nombre del archivo para guardar las rutas
    // como parametro final es la cadena a evaluar.
    public AutomataFND(int estadoActual, int estadoValido, String nameArchivo, String
        cadena) throws IOException
    {
        this.caminos = new ArrayList<String>();
        this.caminosValidos = new ArrayList<CaminosValidos>();
        this.qA = new EstadoActual(estadoActual, estadoValido);
        this.caminos.add("q"+this.qA.getEstadoActual());
        this.tablaTransicion = new TablaTR();
        this.archivoRutas = new ArchivoRutas(nameArchivo);
        this.archivoRutasValidas = new ArchivoRutas(this.archivoRutas.
            getNameArchivo()+" Validas");
        this.conjuntosEst = new Estados();
        this.cadena = cadena;
    }

    // Algoritmo para evaluar la cadena
    // esta funcion recorre la cadena caracter por caracter
    // cada caracter lo manda a evaluarCaracter
    public void evaluarCadena() throws IOException
    {
        this.cadena = cadena;
        // mandar cada caracyer a evaluar el caracter
    }
}
```

```

    for(int i=0; i<this.cadena.toCharArray().length; i++)
    {
        char cadChar = this.cadena.toCharArray()[i];
        evaluarCaracter(cadChar,i);
    }
}
// En esta funcion recibimos el contador del for de evaluarCadena
// lo necesitamos para saber si es la primera evaluacion o no
// si es la primera evaluacion, no hay conjuntos de estados que puede tomar
// solo se evalua directamente el primer caracter con el estado inicial
private void evaluarCaracter(char caracter,int i) throws IOException
{
    int caminosCount= this.caminos.size(); // se obtiene el tama o de los
        caminos
    // si no lo hacemos asi se a adiran mas caminos y el size ira creciendo en
        el for
    // por eso usamos una variable con el tama o.

    // La primera evaluacion es directa ya no tenemos conjuntos a los que
    // podamos ir.
    if(i == 0)
    {
        this.conjuntosEst = tablaTransicion.funcionTransicion(qA, String.
            valueOf(caracter));
        escribirCaminos(i);
    }else
    {
        for(int j = 0; j<caminosCount; j++)
        {
            // pasamos entre el conjunto de los estados obtenidos
            // debemos obtenerlos de los caminos que ya tenemos es el la ultima
                posicion
            // donde se guardara el estado que vamos a tomar.

            // convertirmos a un entero para pasarlo al estado actual
            String SestadoAux = "";

            // determinamos si es un caracter de dos cifras el ultimo estado
                del camino
            if(Character.isDigit(this.caminos.get(j).charAt(this.caminos.get(j).
                length()-2)))
                SestadoAux = this.caminos.get(j).substring(this.caminos.get(j).
                    length()-2);
            else
                SestadoAux = this.caminos.get(j).substring(this.caminos.get(j).
                    length()-1);

            // se lo mandamos como nuevo estado actual
            this.qA.setEstadoActual(Integer.valueOf(SestadoAux));
            this.conjuntosEst = tablaTransicion.funcionTransicion(qA, String.
                valueOf(caracter));

            // escribimos los caminos
            escribirCaminos(j);
        }
    }
}

```

```

    }
}
// para escribir el camino tomamos el punto de referencia
// que en realidad es el index del camino en el que vamos
private void escribirCamino(int puntoReferencia)
{
    String cadAux;
    cadAux = this.camino.get(puntoReferencia); // tomamos el camino
    // en posicion 0 siempre sera igual tomar del camino tomado le
    // pegamos el estado de los conjunto de estados
    this.camino.set(puntoReferencia, cadAux+"->q"+this.conjuntosEstados.getEstadosQ().get(0));
    // le vamos agregando el nuevo estado a los caminos. y listo.
    if(this.conjuntosEstados.isMoreOne())
        for(int k = 1; k < this.conjuntosEstados.getEstadosQ().size(); k++)
            this.camino.add(cadAux+"->q"+this.conjuntosEstados.getEstadosQ().get(k));
}

// para guardar las rutas
public void guardarRutas(TextArea areaText) throws IOException
{
    areaText.clear(); // limpiamos el textArea
    this.archivoRutas.borrarContenido(); // borramos tambien el contenido del
    archivo
    this.caminoValidos.clear();
    for(int i = 0; i < this.camino.size(); i++){
        // determinamos los caracteres del estado final
        if(Character.isDigit(this.camino.get(i).charAt(this.camino.get(i).length()-2)))
            this.qA.setIsValida(this.camino.get(i).substring(this.camino.get(i).length()-2).equals(String.valueOf(this.qA.getEstadoValido())));
        else
            this.qA.setIsValida(this.camino.get(i).substring(this.camino.get(i).length()-1).equals(String.valueOf(this.qA.getEstadoValido())));

        if(this.qA.isIsValida())
        {
            // guardamos el indice del camino correcto
            this.caminoValido = new CaminoValido();
            this.caminoValido.setIndexCamino(i);
            this.caminoValido.setCaminoValido(this.camino.get(i));
            this.caminoValidos.add(this.caminoValido);
            this.archivoRutasValidas.escribirArchivo(this.camino.get(i)+"\n");
            this.archivoRutas.escribirArchivo(camino.get(i)+"*"+"\n");
            areaText.appendText(this.camino.get(i)+"*"+"\n");
        }else
        {
            this.archivoRutas.escribirArchivo(camino.get(i)+"\n");
            areaText.appendText(this.camino.get(i)+"\n");
        }
    }
}
}

```

```

        // getters para los caminos
        public ArrayList<CaminosValidos> getCaminosValidos() {
            return caminosValidos;
        }
    }
}

```

## Código ajedrez los modelos de clases usados

CaminosValidos.java

```

package practica4.chess.Models;

/**
 *
 * @author colin
 * Esta clase sera usada para guardar rutas validas y su index
 */
public class CaminosValidos {
    private String caminoValido;
    private int indexCaminos;

    // constructor
    public CaminosValidos() {
    }

    // getters and setters
    public String getCaminoValido() {
        return caminoValido;
    }

    public void setCaminoValido(String caminoValido) {
        this.caminoValido = caminoValido;
    }

    public int getIndexCaminos() {
        return indexCaminos;
    }

    public void setIndexCaminos(int indexCaminos) {
        this.indexCaminos = indexCaminos;
    }
}

```

## EstadoActual.java

```
package practica4.chess.Models;

/**
 *
 * @author colin
 * Esta clase va a registrar el estado actual de un automata y si es valida
 */
public class EstadoActual {
    private int estadoActual;
    private boolean isValida;
    private int estadoValido;

    // constructor
    public EstadoActual(int estadoActual, int estadoValido)
    {
        this.isValida = false;
        this.estadoActual = estadoActual;
        this.estadoValido = estadoValido;
    }

    public int getEstadoActual() {
        return estadoActual;
    }

    public void setEstadoActual(int estadoActual) {
        this.estadoActual = estadoActual;
    }

    public boolean isIsValida() {
        return isValida;
    }

    public void setIsValida(boolean isValida) {
        this.isValida = isValida;
    }

    public int getEstadoValido() {
        return estadoValido;
    }

    public void setEstadoValido(int estadoValido) {
        this.estadoValido = estadoValido;
    }
}
```

## Estados.java

```
package practica4.chess.Models;

import java.util.ArrayList;

/**
 *
 * @author colin
 * Este es un objeto Estado que determinara el
 * valor del estado o valor del conjunto de estados
 */
public class Estados {
    private ArrayList<Integer> estadosQ;
    private boolean isMoreOne;

    // constructor
    public Estados()
    {
        this.estadosQ = new ArrayList<>();
        this.isMoreOne = false;
    }
    // getters and setters
    public ArrayList<Integer> getEstadosQ() {
        return estadosQ;
    }

    public void setEstadosQ(ArrayList<Integer> estadosQ) {
        this.estadosQ = estadosQ;
    }

    public boolean isMoreOne()
    {
        if(estadosQ.size() > 1)
            this.isMoreOne = true;
        return this.isMoreOne;
    }
}
```



## Dentro de los modelos tenemos nuestra tabla y función de transición

TablaTR.java

```
package practica4.chess.Models;

import java.util.ArrayList;

/**
 *
 * @author colin
 * En esta clase se guardara la informacion de la tabla de transicion
 * tendra su funcion transicion donde recibira un estado actual
 * y devolvera el siguiente estado o el conjunto de estados posibles
 */
public class TablaTR {
    private ArrayList<ArrayList<Estados>> estadosTR;
    private Estados conjunto;

    // Este constructor lo que hace es construir la tabla de
    // transicion. Es una lista de listas. La tabla de transicion
    // se realizo con el tablero de 4x4.
    public TablaTR()
    {

        // constructor
        this.estadosTR = new ArrayList<>();
        this.conjunto = new Estados();

        // Columna de la N (Negro)
        // q1 -> N -> q2, q5
        this.conjunto.getEstadosQ().add(2); // recordar que en la tabla es q1 y
        // debemos restarle uno por el array
        this.conjunto.getEstadosQ().add(5);
        this.estadosTR.add(new ArrayList<Estados>());
        this.estadosTR.get(0).add(conjunto); // la columna N esta en el index 0

        // q2 -> N -> q5, q7
        this.conjunto = new Estados();
        this.estadosTR.add(new ArrayList<Estados>());
        this.conjunto.getEstadosQ().add(5);
        this.conjunto.getEstadosQ().add(7);
        this.estadosTR.get(0).add(conjunto);

        // q3 -> N -> q2, q4, q7
        this.conjunto = new Estados();
        this.estadosTR.add(new ArrayList<Estados>());
        this.conjunto.getEstadosQ().add(2);
        this.conjunto.getEstadosQ().add(4);
        this.conjunto.getEstadosQ().add(7);
        this.estadosTR.get(0).add(conjunto);

        // q4 -> N -> q7
        this.conjunto = new Estados();
        this.estadosTR.add(new ArrayList<Estados>());
        this.conjunto.getEstadosQ().add(7);
```

```

this.estadosTR.get(0).add(conjunto);

//  $q5 \rightarrow N \rightarrow q2, q10$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(2);
this.conjunto.getEstadosQ().add(10);
this.estadosTR.get(0).add(conjunto);

//  $q6 \rightarrow N \rightarrow q2, q5, q7, q10$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(2);
this.conjunto.getEstadosQ().add(5);
this.conjunto.getEstadosQ().add(7);
this.conjunto.getEstadosQ().add(10);
this.estadosTR.get(0).add(conjunto);

//  $q7 \rightarrow N \rightarrow q2, q4, q10, q12$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(2);
this.conjunto.getEstadosQ().add(4);
this.conjunto.getEstadosQ().add(10);
this.conjunto.getEstadosQ().add(12);
this.estadosTR.get(0).add(conjunto);

//  $q8 \rightarrow N \rightarrow q4, q7, q12$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(4);
this.conjunto.getEstadosQ().add(7);
this.conjunto.getEstadosQ().add(12);
this.estadosTR.get(0).add(conjunto);

//  $q9 \rightarrow N \rightarrow q5, q10, q13$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(5);
this.conjunto.getEstadosQ().add(10);
this.conjunto.getEstadosQ().add(13);
this.estadosTR.get(0).add(conjunto);

//  $q10 \rightarrow N \rightarrow q5, q7, q13, q15$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(5);
this.conjunto.getEstadosQ().add(7);
this.conjunto.getEstadosQ().add(13);
this.conjunto.getEstadosQ().add(15);
this.estadosTR.get(0).add(conjunto);

//  $q11 \rightarrow N \rightarrow q7, q10, q12, q15$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(7);

```

```

this.conjunto.getEstadosQ().add(10);
this.conjunto.getEstadosQ().add(12);
this.conjunto.getEstadosQ().add(15);
this.estadosTR.get(0).add(conjunto);
// q12→N→q7, q15
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(7);
this.conjunto.getEstadosQ().add(15);
this.estadosTR.get(0).add(conjunto);
// q13→N→q10
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(10);
this.estadosTR.get(0).add(conjunto);
// q14→N→q10, q13, q15
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(10);
this.conjunto.getEstadosQ().add(13);
this.conjunto.getEstadosQ().add(15);
this.estadosTR.get(0).add(conjunto);
// q15→N→q10, q12
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(10);
this.conjunto.getEstadosQ().add(12);
this.estadosTR.get(0).add(conjunto);
// q16→N→q12, q15
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(12);
this.conjunto.getEstadosQ().add(15);
this.estadosTR.get(0).add(conjunto);

// Columna de B ( blanco )
// q1 → B → q6
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(6);
this.estadosTR.get(1).add(conjunto);

// q2→B→q1, q3, q6
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(1);
this.conjunto.getEstadosQ().add(3);
this.conjunto.getEstadosQ().add(6);
this.estadosTR.get(1).add(conjunto);

// q3→B→q6, q8
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(6);

```

```

this.conjunto.getEstadosQ().add(8);
this.estadosTR.get(1).add(conjunto);

//  $q4 \rightarrow B \rightarrow q3, q8$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(3);
this.conjunto.getEstadosQ().add(8);
this.estadosTR.get(1).add(conjunto);

//  $q5 \rightarrow B \rightarrow q1, q6, q9$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(1);
this.conjunto.getEstadosQ().add(6);
this.conjunto.getEstadosQ().add(9);
this.estadosTR.get(1).add(conjunto);

//  $q6 \rightarrow B \rightarrow q1, q3, q9, q11$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(1);
this.conjunto.getEstadosQ().add(3);
this.conjunto.getEstadosQ().add(9);
this.conjunto.getEstadosQ().add(11);
this.estadosTR.get(1).add(conjunto);

//  $q7 \rightarrow B \rightarrow q3, q6, q8, q11$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(3);
this.conjunto.getEstadosQ().add(6);
this.conjunto.getEstadosQ().add(8);
this.conjunto.getEstadosQ().add(11);
this.estadosTR.get(1).add(conjunto);

//  $q8 \rightarrow B \rightarrow q3, q11$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(3);
this.conjunto.getEstadosQ().add(11);
this.estadosTR.get(1).add(conjunto);

//  $q9 \rightarrow B \rightarrow q6, q14$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(6);
this.conjunto.getEstadosQ().add(14);
this.estadosTR.get(1).add(conjunto);

//  $q10 \rightarrow B \rightarrow q6, q9, q11, q14$ 
this.conjunto = new Estados();
this.estadosTR.add(new ArrayList<Estados>());
this.conjunto.getEstadosQ().add(6);
this.conjunto.getEstadosQ().add(9);

```

```

    this.conjunto.getEstadosQ().add(11);
    this.conjunto.getEstadosQ().add(14);
    this.estadosTR.get(1).add(conjunto);
    //q11->B->q6, q8, q14, q16
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(6);
    this.conjunto.getEstadosQ().add(8);
    this.conjunto.getEstadosQ().add(14);
    this.conjunto.getEstadosQ().add(16);
    this.estadosTR.get(1).add(conjunto);
    //q12->B->q8, q11, q16
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(8);
    this.conjunto.getEstadosQ().add(11);
    this.conjunto.getEstadosQ().add(16);
    this.estadosTR.get(1).add(conjunto);
    //q13->B->q9, q14
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(9);
    this.conjunto.getEstadosQ().add(14);
    this.estadosTR.get(1).add(conjunto);
    //q14->B->q9, q11
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(9);
    this.conjunto.getEstadosQ().add(11);
    this.estadosTR.get(1).add(conjunto);
    //q15->B->q11, q14, q16
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(11);
    this.conjunto.getEstadosQ().add(14);
    this.conjunto.getEstadosQ().add(16);
    this.estadosTR.get(1).add(conjunto);
    //q16->B->q11
    this.conjunto = new Estados();
    this.estadosTR.add(new ArrayList<Estados>());
    this.conjunto.getEstadosQ().add(11);
    this.estadosTR.get(1).add(conjunto);
}

// nuestra funcion de transicion lo que hace es recibir un estado actual
// y una cadena que contiene B o N. La cual la evalua en la tabla de transicion
// haciendo la interseccion del estado actual con N o B de ahi regresara
// el conjunto de estados que puede tomar el automata.
public Estados funcionTransicion(EstadoActual qA, String nOrB)
{
    // Se recibe un estado actual y se busca en la interseccion
    // en la tabla de transicion.
    int colu = 0;

    // verificamos que sea N

```

```

        if(nOrB.equalsIgnoreCase("B"))
            colu = 1;

        this.conjunto = this.estadosTR.get(colu).get(qA.getEstadoActual()-1);
        return this.conjunto;
    }
}

```

## Para las animaciones de la pieza

Pieza.java

```

package practica4.chess.Models;

import static java.lang.Thread.sleep;
import java.util.ArrayList;
import java.util.logging.Level;
import java.util.logging.Logger;
import javafx.animation.Animation;
import javafx.animation.KeyFrame;
import javafx.animation.KeyValue;
import javafx.animation.SequentialTransition;
import javafx.animation.Timeline;
import javafx.animation.TranslateTransition;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.image.ImageView;
import javafx.scene.media.AudioClip;
import javafx.util.Duration;

/**
 *
 * @author colin
 * Este objeto sera quien guarde las animaciones de la pieza
 * individualmente. Guardara las posiciones
 */
public class Pieza {

    private ImageView imagenPieza;
    private int posEstActual;
    private int posX;
    private int posY;
    private int posAnteriorX;
    private int posAnteriorY;
    private SequentialTransition animSecuencia;

    // constructor
    public Pieza(ImageView imagenPieza, int posEstActual) {
        this.imagenPieza = imagenPieza;
        this.posEstActual = posEstActual;
        this.posX = 0;
        this.posY = 0;
        this.posAnteriorX = 0;
        this.posAnteriorY = 0;

        this.animSecuencia = new SequentialTransition();
    }
}

```

```

}

// movimiento de la pieza en x , actualizamos las posiciones y se manda
// a la animacion
public void moverX(boolean isReverse) throws InterruptedException
{
    if(isReverse){
        this.posEstActual--;
        this.posAnteriorX = posX;
        this.posX -= 115;
        this.addAnim(posAnteriorX , posX);
    }
    else{
        this.posEstActual++;
        this.posAnteriorX = this.posX;
        this.posX += 115;
        this.addAnim(this.posAnteriorX , this.posX);
    }
    this.posEstActual = Math.abs(posEstActual);
}

// movimiento en Y.
public void moverY(boolean isReverse) throws InterruptedException
{
    if(isReverse){
        this.posEstActual-=4;
        this.posAnteriorY = this.posY;
        this.posY -=115;
        this.addAnimY(this.posAnteriorY , this.posY);
    }else{
        this.posEstActual+=4;
        this.posAnteriorY = this.posY;
        this.posY+=115;
        this.addAnimY(this.posAnteriorY , this.posY);
    }
    this.posEstActual = Math.abs(posEstActual);
}

// diagonal hacia arriba
public void diagonalArriba(boolean isReverse) throws InterruptedException
{
    if(isReverse){
        this.posEstActual+=3;
        this.posAnteriorX = this.posX;
        this.posAnteriorY = this.posY;
        this.posX -= 115;
        this.posY += 115;
        this.addAnim(posAnteriorX , posX , posAnteriorY , posY);
    }else
    {
        this.posEstActual-=3;
        this.posAnteriorX = this.posX;
        this.posAnteriorY = this.posY;
        this.posX += 115;
        this.posY -= 115;
        this.addAnim(posAnteriorX , posX , posAnteriorY , posY);
    }
}

```

```

        this.posEstActual = Math.abs(posEstActual);
    }
    // diagonal hacia abajo
    public void diagonalAbajo(boolean isReverse) throws InterruptedException
    {
        if(isReverse)
        {
            this.posEstActual -= 5;
            this.posAnteriorX = this.posX;
            this.posAnteriorY = this.posY;
            this.posX -= 115;
            this.posY -= 115;
            this.addAnim(posAnteriorX, posX, posAnteriorY, posY);
        } else
        {
            this.posEstActual += 5;
            this.posAnteriorX = this.posX;
            this.posAnteriorY = this.posY;
            this.posX += 115;
            this.posY += 115;
            this.addAnim(posAnteriorX, posX, posAnteriorY, posY);
        }
        this.posEstActual = Math.abs(posEstActual);
    }

    // restamos la posicion actual con la nueva y dependiendo el valor
    // es el movimiento que haremos el movimiento de la pieza
    public void moverPieza(int posNueva) throws InterruptedException
    {
        int movimiento = this.posEstActual - posNueva;
        switch(movimiento)
        {
            case 5:
                // si la diferencia es 5 es un movimiento diagonal abajo hacia
                // atras
                this.diagonalAbajo(true);
                break;
            case 4:
                // si la diferencia es 4 es solo subir en el tablero
                this.moverY(true);
                break;
            case 3:
                // si la diferencia es 3 es diagonal arriba
                this.diagonalArriba(false);
                break;
            case 1:
                // si la diferencia es uno entonces nos regresamos una posicion
                this.moverX(true);
                break;
            case -1:
                // menos uno es movernos en X hacia delante
                this.moverX(false);
                break;
            case -3:
                // diagonal arriba hacia atras

```



```

        this.diagonalArriba(true);
        break;
    case -4:
        // es bajar en el tablero
        this.moverY(false);
        break;
    case -5:
        // diagonal abajo
        this.diagonalAbajo(false);
        break;
    default:
        // se queda en el mismo lugar
        break;
}
//System.out.println(this.posEstActual); para ver el movimiento correcto
}

// a ade un movimiento en X como animacion
private void addAnim(int posInicial, int posFinal) throws InterruptedException
{
    TranslateTransition movimiento = new TranslateTransition(Duration.millis
        (1700), imagenPieza);
    movimiento.setFromX(posInicial);
    movimiento.setToX(posFinal);

    this.animSecuencia.getChildren().add(movimiento);
}
// a ade un movimiento en Y como animacion
private void addAnimY(int posInicial, int posFinal)
{
    TranslateTransition movimiento = new TranslateTransition(Duration.millis
        (1700), imagenPieza);
    movimiento.setFromY(posInicial);
    movimiento.setToY(posFinal);
    this.animSecuencia.getChildren().add(movimiento);
}
// un movimiento diagonal como animacion
private void addAnim(int posInicialX, int posFinalX, int posInicialY, int
    posFinalY) throws InterruptedException
{
    TranslateTransition movimiento = new TranslateTransition(Duration.millis
        (1700), imagenPieza);
    movimiento.setFromX(posInicialX);
    movimiento.setToX(posFinalX);
    movimiento.setFromY(posInicialY);
    movimiento.setToY(posFinalY);

    this.animSecuencia.getChildren().add(movimiento);
}

// este codigo se dedica a dividir la cadena enviada
// la cadena nos llega en un formato q1->qx->qy->qz
// se quitan las q y los > para quedarnos en una cadena
// de tipo x-y-z de ahí se sacan solo los valores
// x,y,z. Los cuales serán las posiciones a donde se quiere mover

```

```

// con ellas se las mandamos a mover pieza quien se encarga de
// ir guardando las animaciones para despues reproducirlas
public void cargarAnimacion(String rutaElegida) throws InterruptedException
{
    this.animSecuencia = new SequentialTransition();
    // se escribe el codigo para animar la ruta elegida
    System.out.println(rutaElegida);
    rutaElegida = rutaElegida.substring(5);
    rutaElegida = rutaElegida.replace(">", "");
    rutaElegida = rutaElegida.replace("q", "");
    System.out.println(rutaElegida);
    String valor = "";

    for(int i=0; i<rutaElegida.length(); i++)
    {
        if(rutaElegida.charAt(i) == '-'){
            moverPieza(Integer.valueOf(valor));
            valor = "";
        }else
            valor += rutaElegida.charAt(i);
    }
    // para el ultimo digito encontrado
    moverPieza(Integer.valueOf(valor));
}
// inicia la animacion
public void iniciarAnimacion()
{
    this.animSecuencia.play();
}

// resetea los valores a los iniciales.
public void reiniciar()
{
    this.animSecuencia = new SequentialTransition();
    TranslateTransition movimiento = new TranslateTransition(Duration.millis
        (900), imagenPieza);
    movimiento.setFromX(this.posX);
    movimiento.setToX(0);
    movimiento.setFromY(this.posY);
    movimiento.setToY(0);

    this.animSecuencia.getChildren().add(movimiento);
    this.animSecuencia.play();
}
// devuelve las animaciones guardadas para el uso de
// las animaciones cuando se ejecuten juntas.
public SequentialTransition getAnimSecuencia() {
    return animSecuencia;
}
}

```

## Código ajedrez para los archivos

ArchivoRutas.java

```
package practica4.chess.Controllers;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

/**
 *
 * @author colin
 * Clase que se dedicara a escribir los archivos de caminos posibles
 */
public class ArchivoRutas {
    private String ruta;
    private String nombreArchivo;
    private String pathCompleto;
    // variables para lectura y escritura
    private File file;
    private FileWriter fw;
    private BufferedWriter bw;

    // nombre del archivo sin el txt
    // como constructor se crea un archivo nuevo
    // dentro de una carpeta ya establecida
    public ArchivoRutas(String nombreArchivo) throws IOException
    {
        this.ruta = "./src/practica4/chess/Controllers/txtRutas/";
        this.nombreArchivo = nombreArchivo;
        this.pathCompleto = ruta+nombreArchivo+".txt";

        this.file = new File(this.pathCompleto);

        // si el archivo no existe crearlo
        if(!file.exists())
            file.createNewFile();
    }
    // escribe el texto en el archivo
    public void escribirArchivo(String contenido) throws IOException
    {
        this.fw = new FileWriter(this.file, true);
        this.bw = new BufferedWriter(fw);
        this.bw.write(contenido);
        this.bw.close();
    }
    // deja el archivo en blanco
    public void borrarContenido() throws IOException
    {
        this.fw = new FileWriter(this.file);
        this.bw = new BufferedWriter(fw);
        this.bw.write("");
        this.bw.close();
    }
}
```

```

    }

    // getter para derivar del nombre un nuevo archivo con rutas validas
    public String getNameArchivo()
    {
        return this.nombreArchivo;
    }
}

```

## Código del autómata no determinista

AutomataFND.java

```

package practica4.chess.Controllers;

import java.io.IOException;
import java.util.ArrayList;
import javafx.scene.control.TextArea;
import practica4.chess.Models.CaminosValidos;
import practica4.chess.Models.EstadoActual;
import practica4.chess.Models.Estados;
import practica4.chess.Models.TablaTR;

/**
 *
 * @author colin
 * Esta es la clase mas importante
 * ya que lleva el algoritmo del calculo de las rutas
 * usando la tabla y funcion de transicion.
 */
public class AutomataFND {
    private String cadena;
    private TablaTR tablaTransicion;
    private EstadoActual qA;
    private Estados conjuntosEst;
    private ArchivoRutas archivoRutas;
    private ArchivoRutas archivoRutasValidas;
    private ArrayList<String> caminos;
    private CaminosValidos caminoValido;
    private ArrayList<CaminosValidos> caminosValidos;

    // constructor
    // se debe inicializar con un estado actual, y su estado valido
    // tambien necesitamos el nombre del archivo para guardar las rutas
    // como parametro final es la cadena a evaluar.
    public AutomataFND(int estadoActual, int estadoValido, String nameArchivo, String
        cadena) throws IOException
    {
        this.caminos = new ArrayList<String>();
        this.caminosValidos = new ArrayList<CaminosValidos>();
        this.qA = new EstadoActual(estadoActual, estadoValido);
        this.caminos.add("q"+this.qA.getEstadoActual());
        this.tablaTransicion = new TablaTR();
        this.archivoRutas = new ArchivoRutas(nameArchivo);
        this.archivoRutasValidas = new ArchivoRutas(this.archivoRutas.
            getNameArchivo()+"Validas");
    }
}

```

```

    this.conjuntosEst = new Estados();
    this.cadena = cadena;
}

// Algoritmo para evaluar la cadena
// esta funcion recorre la cadena caracter por caracter
// cada caracter lo manda a evaluarCaracter
public void evaluarCadena() throws IOException
{
    this.cadena = cadena;
    // mandar cada caracyer a evaluar el caracter
    for(int i=0; i<this.cadena.toCharArray().length; i++)
    {
        char cadChar = this.cadena.toCharArray()[i];
        evaluarCaracter(cadChar,i);
    }
}

// En esta funcion recibimos el contador del for de evaluarCadena
// lo necesitamos para saber si es la primera evaluacion o no
// si es la primera evaluacion, no hay conjuntos de estados que puede tomar
// solo se evalua directamente el primer caracter con el estado inicial
private void evaluarCaracter(char caracter,int i) throws IOException
{
    int caminosCount= this.caminos.size(); // se obtiene el tama o de los
        caminos
    // si no lo hacemos asi se a adiran mas caminos y el size ira creciendo en
        el for
    // por eso usamos una variable con el tama o.

    // La primera evaluacion es directa ya no tenemos conjuntos a los que
    // podamos ir.
    if(i == 0)
    {
        this.conjuntosEst = tablaTransicion.funcionTransicion(qA, String.
            valueOf(caracter));
        escribirCaminos(i);
    } else
    {
        for(int j = 0; j<caminosCount; j++)
        {
            // pasamos entre el conjunto de los estados obtenidos
            // debemos obtenerlos de los caminos que ya tenemos es el la ultima
                posicion
            // donde se guardara el estado que vamos a tomar.

            // convertirmos a un entero para pasarlo al estado actual
            String SestadoAux = "";

            // determinamos si es un caracter de dos cifras el ultimo estado
                del camino
            if(Character.isDigit(this.caminos.get(j).charAt(this.caminos.get(j)
                .length()-2)))
                SestadoAux = this.caminos.get(j).substring(this.caminos.get(j).
                    length()-2);
            else

```

```

        SestadoAux = this.caminos.get(j).substring(this.caminos.get(j).
            length()-1);

        // se lo mandamos como nuevo estado actual
        this.qA.setEstadoActual(Integer.valueOf(SestadoAux));
        this.conjuntosEst = tablaTransicion.funcionTransicion(qA, String.
            valueOf(caracter));

        // escribimos los caminos
        escribirCaminos(j);
    }
}

// para escribir el camino tomamos el punto de referencia
// que en realidad es el index del camino en el que vamos
private void escribirCaminos(int puntoReferencia)
{
    String cadAux;
    cadAux = this.caminos.get(puntoReferencia); // tomamos el camino
    // en posicion 0 siempre sera igual tomar del camino tomado le
    // pegamos el estado de los conjunto de estados
    this.caminos.set(puntoReferencia, cadAux+"->q"+this.conjuntosEst.
        getEstadosQ().get(0));
    // le vamos agregando el nuevo estado a los caminos. y listo.
    if(this.conjuntosEst.isMoreOne())
        for(int k = 1; k < this.conjuntosEst.getEstadosQ().size(); k++){
            this.caminos.add(cadAux+"->q"+this.conjuntosEst.getEstadosQ().get(k)
                );
        }
}

// para guardar las rutas
public void guardarRutas(TextArea areaText) throws IOException
{
    areaText.clear(); // limpiamos el textArea
    this.archivoRutas.borrarContenido(); // borramos tambien el contenido del
        archivo
    this.caminosValidos.clear();
    for(int i = 0; i < this.caminos.size(); i++){
        // determinamos los caracteres del estado final
        if(Character.isDigit(this.caminos.get(i).charAt(this.caminos.get(i).
            length()-2)))
            this.qA.setIsValida(this.caminos.get(i).substring(this.caminos.get(
                i).length()-2).equals(String.valueOf(this.qA.getEstadoValido()))
            );
        else
            this.qA.setIsValida(this.caminos.get(i).substring(this.caminos.get(
                i).length()-1).equals(String.valueOf(this.qA.getEstadoValido()))
            );

        if(this.qA.isIsValida())
        {
            // guardamos el indice del camino correcto
            this.caminoValido = new CaminosValidos();
            this.caminoValido.setIndexCaminos(i);
            this.caminoValido.setCaminoValido(this.caminos.get(i));
        }
    }
}

```

```

        this.caminoValidos.add(this.caminoValido);
        this.archivoRutasValidas.escribirArchivo(this.caminoValido+"\n");
        this.archivoRutas.escribirArchivo(caminoValido+"\n");
        areaText.appendText(this.caminoValido+"\n");
    }else
    {
        this.archivoRutas.escribirArchivo(caminoValido+"\n");
        areaText.appendText(this.caminoValido+"\n");
    }
}
// getters para los caminos
public ArrayList<CaminosValidos> getCaminoValidos() {
    return caminoValidos;
}
}

```

## Control entre la vista y el autómata

FXMLMainController.java

```

package practica4.chess.Controllers;

import java.io.IOException;
import java.net.URL;
import java.util.ArrayList;
import java.util.Random;
import java.util.ResourceBundle;
import javafx.animation.SequentialTransition;
import javafx.beans.property.SimpleStringProperty;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.Alert;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.control.Spinner;
import javafx.scene.control.SpinnerValueFactory;
import javafx.scene.control.TableColumn;
import javafx.scene.control.TableView;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.scene.image.ImageView;
import javafx.stage.Stage;
import javafx.stage.StageStyle;
import practica4.chess.Models.CaminosValidos;
import practica4.chess.Models.Pieza;

/*
    Esta clase es la comunicacion entre la vista y el automata
    Toda la vista llama al controlador AutomataFND Finito no determinista

```

```

*/
public class FXMLMainController {

    // botones y elementos de la vista
    @FXML
    private Button btnAnim1A;
    @FXML
    private ImageView pieza1;
    @FXML
    private ImageView pieza2;
    @FXML
    private Button btnSalir;
    @FXML
    private Button btnReiniciar;
    @FXML
    private TextField txtCadenaA1;
    @FXML
    private TextArea areaTxtA1;
    @FXML
    private TextField txtCadenaA2;
    @FXML
    private Button btnGenRutasA2;
    @FXML
    private TextArea areaTxtA2;
    @FXML
    private Button btnAnim2A;
    @FXML
    private Button btnGenRutasA1;
    @FXML
    private Button bntVerRutaA1;
    @FXML
    private Button btnVerRutaA2;
    @FXML
    private Button btnAmbasAnim;
    @FXML
    private Spinner<Integer> spinerNumber;
    @FXML
    private Button btnGenerarCadenas;

    @FXML
    private TableView<CaminoValidos> tableRutasVA1;
    @FXML
    private TableColumn<CaminoValidos, Integer> columnIndexA1;
    @FXML
    private TableColumn<CaminoValidos, String> columnRVA1;
    @FXML
    private TableView<CaminoValidos> tableRutasVA2;
    @FXML
    private TableColumn<CaminoValidos, Integer> columnIndexA2;
    @FXML
    private TableColumn<CaminoValidos, String> columnRVA2;
    //
    // variables fuera de la vista
    private Pieza pieza1AObj;

```



```

private Pieza pieza2AObj;

// automatas
private AutomataFND automata1;
private AutomataFND automata2;

// Esta funcion inicializa el automata, manda la cadena dentro del
// txtCadena y llama a la funcion importante que es la de evaluar la cadena
// se manda el areaTex a guardar rutas para llenar con datos el textArea
// se llaman a los estados validos y si estan vacios significa
// que no es una cadena valida.
@FXML
void GenerarRutasA2(ActionEvent event) throws IOException {

    // determinamos si no esta vacio el texto del TextField para automata 1
    if (!txtCadenaA2.getText().isEmpty())
    {
        // si no esta vacio
        // convertimos el texto a mayusculas
        // se inicializa empezando en 4 y terminando en 13 como estado valido
        // se pueden cambiar los estados iniciales y finales
        this.automata2 = new AutomataFND(4,13,"automata2", txtCadenaA2.getText()
            ().toString().trim().toUpperCase());
        this.automata2.evaluarCadena();
        this.automata2.guardarRutas(this.areaTxtA2); // se guardan las rutas en
            el archivo
        // habilitamos su boton para iniciar la animacion
        this.btnAnim2A.setDisable(false);
        // lenamos la tabla
        this.tableRutasVA2.getItems().setAll(this.automata2.getCaminosValidos());
        if (!this.tableRutasVA2.getItems().isEmpty())
        {
            // si no hay rutas validas
            this.tableRutasVA2.getSelectionModel().select(0);
            this.tableRutasVA2.getSelectionModel().focus(0);
            this.btnAnim2A.setDisable(false);
            this.btnVerRutaA2.setDisable(false);
        }else
            alerta("NO_HAY_RUTAS", "La_cadena_ingresada_no_tiene_una_ruta_
                valida");
    }
    else{
        // si esta vacio el texto
        btnAnim2A.setDisable(true);
        alerta("!ERROR", "Debes_escribir_una_cadena_para_el_automata_2");
    }
}

// Esta funcion inicializa el automata, manda la cadena dentro del
// txtCadena y llama a la funcion importante que es la de evaluar la cadena
// se manda el areaTex a guardar rutas para llenar con datos el textArea
// se llaman a los estados validos y si estan vacios significa
// que no es una cadena valida.
@FXML
void generarRutasA1(ActionEvent event) throws IOException {

```

```

// determinamos si no esta vacio el texto del TextField para automata 1
if (!txtCadenaA1.getText().isEmpty())
{
    // si no esta vacio
    // convertimos el texto a mayusculas
    // se inicializa empezando en 1 y terminando en 16 como estado valido
    this.automata1 = new AutomataFND(1,16,"automata1", txtCadenaA1.getText()
        ().toString().trim().toUpperCase());
    this.automata1.evaluarCadena();
    this.automata1.guardarRutas(this.areaTxtA1); // se guardan las rutas en
        el archivo

    // llenamos la tabla de rutas validas
    this.tableRutasVA1.getItems().setAll(this.automata1.getCaminosValidos()
        );
    if (!this.tableRutasVA1.getItems().isEmpty())
    {
        // si hay rutas validas
        this.tableRutasVA1.getSelectionModel().select(0);
        this.tableRutasVA1.getSelectionModel().focus(0);
        // habilitamos su boton para iniciar la animacion
        this.btnAnim1A.setDisable(false);
        this.bntVerRutaA1.setDisable(false);
    } else
        alerta("NO_HAY_RUTAS", "La cadena ingresada no tiene una ruta
            valida");
}
else{
    // si esta vacio el texto
    btnAnim1A.setDisable(true);
    alerta("!ERROR", "Debes escribir una cadena para el automata 1");
}
}
// Iniciar una animacion independiente.
// Se toma la informacion de la seleccion de la tabla y se la manda a cargar
// nuestra animacion despues de cargarla se ejecuta la animacion
@FXML
void iniciarAnimA1(ActionEvent event) throws IOException, InterruptedException
{
    CaminosValidos camino = this.tableRutasVA1.getSelectionModel().
        getSelectedItem();
    // se elige una animacion
    this.pieza1AObj.cargarAnimacion(camino.getCaminoValido());
    this.pieza1AObj.iniciarAnimacion();
}
// Iniciar una animacion independiente.
// Se toma la informacion de la seleccion de la tabla y se la manda a cargar
// nuestra animacion despues de cargarla se ejecuta la animacion
@FXML
void iniciarAnimA2(ActionEvent event) throws InterruptedException {
    CaminosValidos camino2 = this.tableRutasVA2.getSelectionModel().
        getSelectedItem();
    this.pieza2AObj.cargarAnimacion(camino2.getCaminoValido());
    this.pieza2AObj.iniciarAnimacion();
}

```

```

}
// Iniciar una animaciones al mismo tiempo.
// Se toma la informacion de la seleccion de la tabla y se la manda a cargar
// esta informacion es de cada una de las tablas
// se identifica el camino mas grande y ese sera quien se llene al final
@FXML
void animA1A2(ActionEvent event) throws InterruptedException {

    if(this.btnAnim1A.isDisable() || this.btnAnim2A.isDisable())
        alerta("Error", "Una_de_las_cadenas_no_es_correcta");
    else
    {
        SequentialTransition sq = new SequentialTransition();
        CaminosValidos caminos = this.tableRutasVA1.getSelectionModel().
            getSelectedItem();
        CaminosValidos caminos2 = this.tableRutasVA2.getSelectionModel().
            getSelectedItem();

        this.pieza1AObj.cargarAnimacion(caminos.getCaminoValido());
        this.pieza2AObj.cargarAnimacion(caminos2.getCaminoValido());

        int sizeA1= this.pieza1AObj.getAnimSecuencia().getChildren().size();
        int sizeA2 =this.pieza2AObj.getAnimSecuencia().getChildren().size();
        if(sizeA1>=sizeA2) // cual tiene mas animaciones
            for(int i=0; i<sizeA1; i++)
            {
                sq.getChildren().add(this.pieza1AObj.getAnimSecuencia().
                    getChildren().get(i));
                if(i<sizeA2)
                    sq.getChildren().add(this.pieza2AObj.getAnimSecuencia().
                        getChildren().get(i));
            }
        if(sizeA1<sizeA2) // cual tiene mas animaicones
            for(int i = 0; i<sizeA2; i++)
            {
                sq.getChildren().add(this.pieza2AObj.getAnimSecuencia().
                    getChildren().get(i));
                if(i<sizeA1)
                    sq.getChildren().add(this.pieza1AObj.getAnimSecuencia().
                        getChildren().get(i));
            }

        sq.play();
    }

}

// reinicia los valores para hacerlo de nuevo
@FXML
void reiniciar(ActionEvent event) throws InterruptedException , IOException {
    this.pieza1AObj.reiniciar();
    this.pieza2AObj.reiniciar();
    this.pieza1AObj = new Pieza(pieza1, 1);
    this.pieza2AObj = new Pieza(pieza2, 4);
}
// cierra la ventana

```

```

@FXML
void salir(ActionEvent event) {
    Stage stageActual = (Stage) this.btnSalir.getScene().getWindow();
    stageActual.close();
}
// mostramos una alerta con la informacion de la ruta
@FXML
void verRutaA1(ActionEvent event) {
    Alert dialogAlert = new Alert(Alert.AlertType.INFORMATION); // creamos el
        dialogo de alerta
    dialogAlert.setTitle("RUTA_ELEGIDA");
    dialogAlert.setContentText(this.tableRutasVA1.getSelectionModel().
        getSelectedItem().getCaminoValido());
    dialogAlert.setHeaderText(null);
    dialogAlert.initStyle(StageStyle.UTILITY);
    dialogAlert.showAndWait();
}
// mostramos una alerta con la informacion de la ruta
@FXML
void verRutaA2(ActionEvent event) {
    Alert dialogAlert = new Alert(Alert.AlertType.INFORMATION); // creamos el
        dialogo de alerta
    dialogAlert.setTitle("RUTA_ELEGIDA");
    dialogAlert.setContentText(this.tableRutasVA2.getSelectionModel().
        getSelectedItem().getCaminoValido());
    dialogAlert.setHeaderText(null);
    dialogAlert.initStyle(StageStyle.UTILITY);
    dialogAlert.showAndWait();
}
// para generar cadenas aleatorias
// leemos la cantidad de caracteres y
// se hace un random para generar 2 cadenas aleatorias
@FXML
void generarCadenas(ActionEvent event) {
    int azar = 0;
    Random r = new Random();
    int maxCad = Integer.valueOf(this.spinerNumber.getEditor().getText());
    String nuevaCad = "";
    if(maxCad>0){
        for(int j=0; j<2; j++)
        {
            for(int i = 0; i<maxCad; i++)
            {
                azar = r.nextInt(2);
                System.out.println(azar);
                if(azar == 1)
                    nuevaCad += "N";
                else
                    nuevaCad += "B";
            }
        }
        if(j == 0)
            this.txtCadenaA1.setText(nuevaCad);
        else
            this.txtCadenaA2.setText(nuevaCad);
    }
}

```

```

        nuevaCad = "";
    }
}
else
    alerta("ERROR", "Debes_elegir_una_cadena_mayor_a_0");
}

// inicializamos las vistas y datos
@FXML
void initialize() {
    // inicializando las piezas y diciendoles cuales son para animarlas con
    // la clase Pieza
    this.pieza1AObj = new Pieza(this.pieza1,1);
    this.pieza2AObj = new Pieza(this.pieza2,4);
    // inicializamos las columnas de las tablas
    this.columnRVA1.setCellValueFactory(new PropertyValueFactory<>("
        caminoValido"));
    this.columnIndexA1.setCellValueFactory(new PropertyValueFactory<>("
        indexCaminos"));
    this.columnRVA2.setCellValueFactory(new PropertyValueFactory<>("
        caminoValido"));
    this.columnIndexA2.setCellValueFactory(new PropertyValueFactory<>("
        indexCaminos"));
    this.tableRutasVA1.setPlaceholder(new Label("No_hay_caminos_validos"));
    this.tableRutasVA2.setPlaceholder(new Label("No_hay_caminos_validos"));
    SpinnerValueFactory<Integer> valuesFactory = new SpinnerValueFactory.
        IntegerSpinnerValueFactory(0, 15, 0);
    this.spinerNumber.setValueFactory(valuesFactory);
}

// funcion para mostrar alerta de dialogo
public void alerta(String titulo,String contenido)
{
    Alert dialogAlert = new Alert(Alert.AlertType.ERROR); // creamos el dialogo
    de alreta
    dialogAlert.setTitle(titulo);
    dialogAlert.setContentText(contenido);
    dialogAlert.setHeaderText(null);
    dialogAlert.initStyle(StageStyle.UTILITY);
    dialogAlert.showAndWait();
}
}
}

```

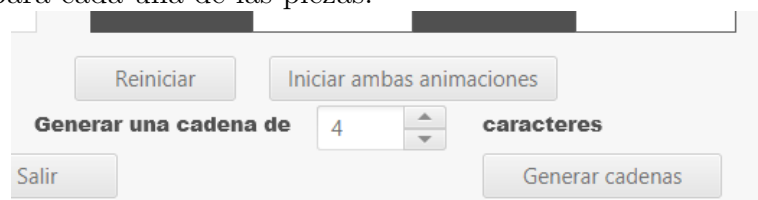
# Funcionamiento del programa

## Cadena generada automáticamente

La interfaz del programa es la siguiente:



Se puede generar la cadena aleatoriamente eligiendo el numero de caracteres de la cadena. Se genera una cadena para cada una de las piezas.



Cadenas generadas A1 =NBBB y A2 =NNBB:

Podemos ver que en la cadena NBBB si es una ruta valida. Por ello se generan todos los caminos posibles.

automata1: Bloc de notas

[Archivo](#) [Edición](#) [Formato](#) [Ve](#)

```
q1->q2->q1->q6->q1
q1->q5->q1->q6->q1
q1->q2->q3->q6->q1
q1->q2->q6->q1->q6
q1->q5->q6->q1->q6
q1->q5->q9->q6->q1
q1->q2->q3->q8->q3
q1->q2->q6->q3->q6
q1->q2->q6->q9->q6
q1->q2->q6->q11->q6
q1->q5->q6->q3->q6
q1->q5->q6->q9->q6
q1->q5->q6->q11->q6
q1->q5->q9->q14->q9
q1->q2->q1->q6->q3
q1->q2->q1->q6->q9
q1->q2->q1->q6->q11
q1->q5->q1->q6->q3
q1->q5->q1->q6->q9
q1->q5->q1->q6->q11
q1->q2->q3->q6->q3
q1->q2->q3->q6->q9
q1->q2->q3->q6->q11
q1->q5->q9->q6->q3
q1->q5->q9->q6->q9
q1->q5->q9->q6->q11
q1->q2->q3->q8->q11
q1->q2->q6->q3->q8
q1->q2->q6->q9->q14
q1->q2->q6->q11->q8
q1->q2->q6->q11->q14
q1->q2->q6->q11->q16*
q1->q5->q6->q3->q8
q1->q5->q6->q9->q14
q1->q5->q6->q11->q8
q1->q5->q6->q11->q14
q1->q5->q6->q11->q16*
a1->a5->a9->a14->a11
```

automata1Validas: Bloc de nota

[Archivo](#) [Edición](#) [Formato](#) [Ver](#)

```
q1->q6->q11->q16
q1->q2->q6->q11->q16
q1->q5->q6->q11->q16
```

Para la cadena NNBB no es una cadena valida por ello el programa nos muestra el error. Aún así se generan nuestras cadenas no validas.



automata2: Bloc de notas

Archivo Edición Formato Ver

```

q4->q7->q2->q1->q6
q4->q7->q4->q3->q6
q4->q7->q10->q6->q1
q4->q7->q12->q8->q3
q4->q7->q2->q3->q6
q4->q7->q2->q6->q1
q4->q7->q4->q8->q3
q4->q7->q10->q9->q6
q4->q7->q10->q11->q6
q4->q7->q10->q14->q9
q4->q7->q12->q11->q6
q4->q7->q12->q16->q11
q4->q7->q4->q3->q8
q4->q7->q10->q6->q3
q4->q7->q10->q6->q9
q4->q7->q10->q6->q11
q4->q7->q12->q8->q8
q4->q7->q2->q6->q3
q4->q7->q2->q6->q9
q4->q7->q2->q6->q11
q4->q7->q2->q6->q16
q4->q7->q4->q8->q11
q4->q7->q10->q9->q14
q4->q7->q10->q11->q8
q4->q7->q10->q11->q14
q4->q7->q10->q11->q16
q4->q7->q10->q14->q11
q4->q7->q12->q11->q8
q4->q7->q12->q11->q14
q4->q7->q12->q11->q16

```



Para que se haga una cadena valida le borramos la ultima B y ponemos una N. Se convierte en una cadena NNBN la cual es valida y se generan sus caminos.

automata2: Bloc de notas

Archivo Edición Formato Ver

q4->q7->q2->q1->q2  
 q4->q7->q4->q3->q2  
 q4->q7->q10->q6->q2  
 q4->q7->q12->q8->q4  
 q4->q7->q2->q3->q2  
 q4->q7->q2->q6->q2  
 q4->q7->q4->q8->q4  
 q4->q7->q10->q9->q5  
 q4->q7->q10->q11->q7  
 q4->q7->q10->q14->q10  
 q4->q7->q12->q11->q7  
 q4->q7->q12->q16->q12  
 q4->q7->q2->q1->q5  
 q4->q7->q4->q3->q4  
 q4->q7->q4->q3->q7  
 q4->q7->q10->q6->q5

q4->q7->q10->q6->q7



q4->q7->q10->q6->q10  
 q4->q7->q12->q8->q7  
 q4->q7->q12->q8->q12  
 q4->q7->q2->q3->q4  
 q4->q7->q2->q3->q7  
 q4->q7->q2->q6->q5  
 q4->q7->q2->q6->q7  
 q4->q7->q2->q6->q10  
 q4->q7->q4->q8->q7  
 q4->q7->q4->q8->q12  
 q4->q7->q10->q9->q10  
 q4->q7->q10->q9->q13\*  
 q4->q7->q10->q11->q10  
 q4->q7->q10->q11->q12  
 q4->q7->q10->q11->q15  
 q4->q7->q10->q14->q13  
 q4->q7->q10->q14->q15  
 q4->q7->q12->q11->q10  
 q4->q7->q12->q11->q12  
 q4->q7->q12->q11->q15  
 q4->q7->q12->q16->q15

automata2Validas: Bloc de notas

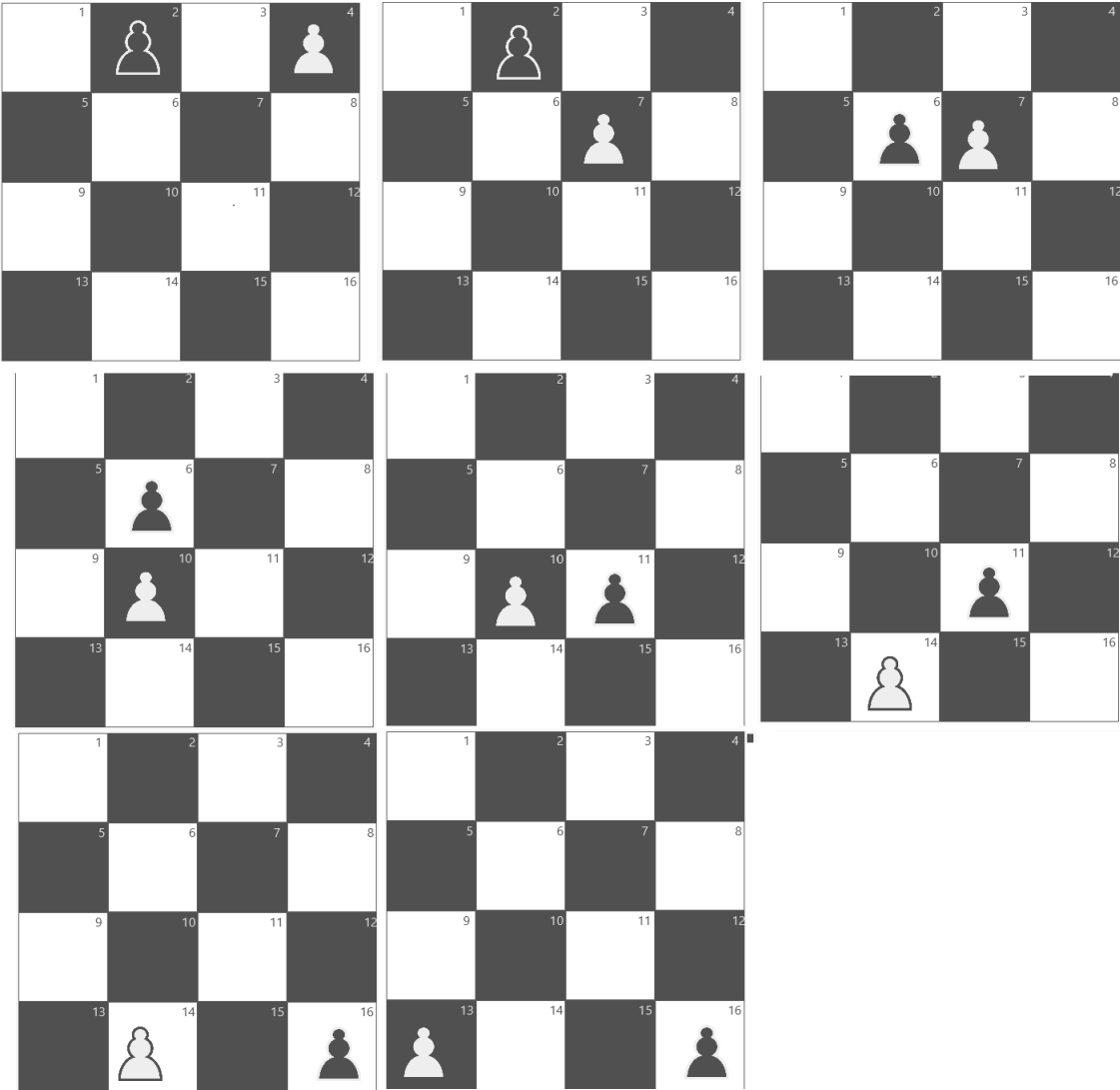
Archivo Edición Formato Ver Ayuda

q4->q7->q10->q9->q13  
 q4->q7->q10->q14->q13

Eligiendo las rutas que tomaran.

RUTA ELEGIDA	RUTA ELEGIDA
<div>  q1-&gt;q2-&gt;q6-&gt;q11-&gt;q16 </div> <div>Aceptar</div>	<div>  q4-&gt;q7-&gt;q10-&gt;q14-&gt;q13 </div> <div>Aceptar</div>

# Animando



# Introducción práctica (Buscador de Palabras)