Programa Tablero

Serrano Gayosso José Eduardo
Grupo: 4CV2

13 de octubre de 2024

1 Introducción

Los *Autómatas Finitos No Deterministas* (AFND) son una extensión de los autómatas finitos que permiten múltiples transiciones para un mismo estado y símbolo de entrada. A diferencia de los autómatas deterministas, en los cuales cada estado y símbolo tienen una única transición definida, los AFND pueden tener varias transiciones posibles o incluso ninguna para un mismo par de estado y símbolo. Esta característica proporciona a los AFND la capacidad de modelar sistemas donde el comportamiento no está completamente determinado, permitiendo explorar todas las posibles rutas que un sistema puede seguir.

Un AFND se define por un conjunto finito de estados, un alfabeto de entrada, un estado inicial, un conjunto de estados de aceptación, y una función de transición que asigna a cada estado y símbolo un conjunto de posibles estados siguientes. La potencia de los AFND reside en su habilidad para manejar múltiples rutas simultáneamente, lo que facilita el modelado de sistemas complejos como problemas de enrutamiento, búsqueda en profundidad y problemas de decisión.

En el contexto de este programa, los AFND se utilizan para modelar el movimiento de dos jugadores sobre un tablero de ajedrez de 5x5. Cada jugador tiene un autómata que determina las posibles transiciones entre las casillas del tablero, basadas en una secuencia de movimientos definidos por un conjunto de símbolos. El movimiento de cada jugador es no determinista, lo que significa que pueden existir múltiples rutas posibles para llegar a su objetivo final, lo que añade complejidad y desafío a la dinámica del juego.

2 Descripción del Programa

El programa está compuesto por varias clases y funciones que gestionan la lógica del juego, la representación gráfica del tablero, la ejecución de los movimientos y la visualización de las rutas. A continuación se describen los elementos principales:

2.1 Métodos del Programa

2.1.1 `AFND.__init__(estado_inicial, estado_final)`

El constructor `__init__` de la clase `AFND` inicializa el autómata finito no determinista (AFND) para cada jugador. Este autómata controla los movimientos de los jugadores en el tablero, permitiendo múltiples transiciones desde un mismo estado según el símbolo del alfabeto. A continuación, se describe detalladamente el funcionamiento de este método:

1. **Definición del estado inicial y final:**
   - El estado inicial (`estado_inicial`) es la casilla en la que comienza el jugador. Para el jugador 1, este estado es la casilla 1; para el jugador 2, es la casilla 5.
   - El estado final (`estado_final`) es el objetivo del jugador. El jugador 1 debe llegar a la casilla 25, mientras que el jugador 2 debe llegar a la casilla 21.
2. **Definición del alfabeto:**
   - El alfabeto del autómata está compuesto por dos símbolos: `'r'` y `'b'`, que representan las transiciones a casillas de color rojo (`'r'`) y negro (`'b'`), respectivamente.
3. **Definición de las transiciones:**
   - Se crea un diccionario `self.transiciones` que define las posibles transiciones desde cada estado del tablero (cada casilla) en función del símbolo leído.
   - Para cada casilla, el diccionario especifica a qué otras casillas puede moverse el jugador dependiendo de si la transición es sobre una casilla roja o negra. Este conjunto de transiciones permite que el autómata sea no determinista, ya que un mismo estado puede tener múltiples transiciones válidas para un símbolo determinado.

4. **Ejemplo de transiciones:**

    - Por ejemplo, el estado 1 (casilla 1) tiene las siguientes transiciones: con el símbolo `'r'`, el jugador puede moverse a las casillas 2 o 6, y con el símbolo `'b'`, puede moverse a la casilla 7.
    - Este patrón de transiciones se repite para todas las casillas del tablero, creando un autómata que gestiona los movimientos posibles en función del color de las casillas.

2.1.2 `AFND.procesar(cadena, jugador)`

El método `procesar` toma una cadena de movimientos y evalúa todas las rutas posibles que un jugador puede seguir en el tablero utilizando el autómata finito no determinista. Este método realiza un recorrido en profundidad de las posibles transiciones, gestionando múltiples rutas simultáneamente. Además, guarda tanto los estados visitados como los símbolos de transición. A continuación se describen sus pasos:

1. **Inicialización de la pila:**

    - Se inicializa una pila que contiene tuplas con cuatro elementos: el estado actual del jugador, la parte restante de la cadena por procesar, la ruta seguida hasta ese punto, y la lista de transiciones (símbolos 'r' o 'b') utilizadas.
    - La pila comienza con el estado inicial del jugador, la cadena completa, una lista que contiene solo el estado inicial, y una lista vacía para las transiciones.

2. **Procesamiento de la cadena:**

    - El método sigue un enfoque de búsqueda en profundidad (*Depth-First Search, DFS*). Mientras la pila no esté vacía, se toma el último elemento insertado.
    - Si la cadena restante es vacía, significa que se ha completado una ruta. En este caso, la ruta junto con los símbolos de transición se escribe en un archivo:
        - Si el jugador está en su estado final, la ruta y los símbolos se escriben en el archivo `Ganadoras{jugador}.txt`.
        - Si no está en el estado final, la ruta y los símbolos se clasifican como perdedores y se escriben en `Perdedoras{jugador}.txt`.

- Todas las rutas, junto con sus transiciones, se registran también en el archivo `Rutas{jugador}.txt`.

3. **Evaluación de transiciones:**

   - Si la cadena aún contiene símbolos, se obtiene el primer símbolo de la cadena (`simbolo`).
   - Se consultan las posibles transiciones desde el estado actual utilizando el símbolo correspondiente. Si no hay transiciones disponibles, se muestra un mensaje de error y el programa termina.
   - Para cada estado posible de transición, se añade una nueva tupla a la pila, que contiene el nuevo estado, la parte restante de la cadena, la ruta actualizada y la lista de símbolos de transición actualizada con el símbolo actual.

4. **Resultado:**

   - El método continúa hasta que todas las rutas posibles se han evaluado. Devuelve un valor booleano que indica si se encontró alguna ruta ganadora.

2.1.3 `inicializar_tablero()`

El método `inicializar_tablero` genera la configuración inicial del tablero de ajedrez 5x5, con colores alternados para las casillas y las posiciones iniciales de los jugadores. A continuación se describe cómo funciona:

1. **Creación de la lista del tablero:**

   - Se crea una lista vacía llamada `tablero`. Esta lista contendrá 25 elementos, cada uno representando una casilla del tablero.
   - Un ciclo `for` doble recorre las filas y columnas del tablero (de 5x5), calculando el estado de cada casilla.

2. **Colores de las casillas:**

   - Las casillas alternan entre negro (`0`) y rojo (`1`), con un patrón en forma de tablero de ajedrez.
   - Si la suma de la fila y la columna es par, la casilla es negra (`0`). Si la suma es impar, la casilla es roja (`1`).

3. **Posiciones iniciales de los jugadores:**

- La casilla 1 (estado 1) se asigna al jugador 1 y se marca con el color azul (`2`).
- La casilla 5 (estado 5) se asigna al jugador 2 y se marca con el color café (`3`).

4. **Devolución del tablero:**

   - El método devuelve la lista del tablero, que se utilizará posteriormente para representar gráficamente el estado del juego.

2.1.4 `mostrar_tablero(tablero, ax)`

El método `mostrar_tablero` dibuja gráficamente el tablero en una ventana utilizando la librería `matplotlib`. A continuación se describe cómo se implementa:

1. **Conversión de la lista a matriz:**

   - La lista `tablero`, que contiene 25 elementos (uno por cada casilla), se convierte en una matriz de 5x5 utilizando `numpy`. Esto permite que el tablero sea visualizado como una cuadrícula.

2. **Creación del mapa de colores:**

   - Se define un mapa de colores utilizando `ListedColormap` de `matplotlib`. Los colores negro, rojo, azul y café representan las casillas negras, rojas, la posición del jugador 1, y la posición del jugador 2, respectivamente.

3. **Dibujo del tablero:**

   - La función `imshow` se utiliza para mostrar la matriz del tablero con los colores adecuados. Las casillas se muestran en una cuadrícula de 5x5.
   - Se añaden líneas de cuadrícula (`grid`) para resaltar las separaciones entre las casillas.

4. **Números de las casillas:**

   - Se agregan los números de las casillas (estados) utilizando la función `text`. Estos números permiten al usuario identificar fácilmente el estado en el que se encuentra cada jugador.

5. **Actualización gráfica:**

- Después de dibujar el tablero, se llama a `plt.draw()` para actualizar la ventana gráfica y mostrar el estado actual del tablero.
- Se añade una pausa de 1 segundo (`plt.pause(1)`) para que el usuario pueda ver cada actualización de forma clara antes de que se realicen nuevos movimientos.

2.1.5 `actualizar_tablero(tablero, estado_actual, estado_previo, jugador, ax)`

El método `actualizar_tablero` gestiona el movimiento de un jugador en el tablero, actualizando las casillas correspondientes a la posición anterior y la nueva posición del jugador. A continuación se describe su funcionamiento:

1. **Restauración del estado previo:**
   - Si el jugador se ha movido previamente, la casilla anterior (`estado_previo`) se restaura a su color original (negro o rojo).
   - Para determinar el color correcto, se calcula la fila y la columna de la casilla anterior, y se restaura el color basado en si la suma de la fila y la columna es par o impar.
2. **Actualización del estado actual:**
   - La casilla en la que el jugador se encuentra ahora (`estado_actual`) se actualiza con el color del jugador:
     - Si el jugador es el jugador 1, se usa el color azul (`2`).
     - Si es el jugador 2, se usa el color café (`3`).
3. **Dibujo del tablero actualizado:**
   - El método `mostrar_tablero()` se llama para redibujar el tablero con los cambios en las posiciones de los jugadores. Esto asegura que el tablero se actualice visualmente con cada movimiento.

2.1.6 `juego(ruta1, ruta2, primero)`

El método `juego` es el núcleo del programa que gestiona la ejecución del juego en el tablero para ambos jugadores. El método coordina la lectura de las rutas ganadoras generadas para cada jugador, simula los turnos de cada uno, gestiona posibles conflictos de movimientos entre jugadores, y actualiza el tablero visualmente para mostrar el progreso. A continuación, se detalla el funcionamiento del método:

1. **Lectura de rutas ganadoras:**

   - El método comienza abriendo los archivos de rutas ganadoras `Ganadoras1.txt` y `Ganadoras2.txt`, que contienen las rutas generadas previamente para el jugador 1 y el jugador 2, respectivamente.
   - Para cada jugador, se leen todas las líneas del archivo correspondiente y se almacenan en las listas `lineas1` y `lineas2`. En estos archivos, las líneas impares contienen los estados por los que pasa el jugador, mientras que las líneas pares contienen los símbolos de transición correspondientes ('r' o 'b').
   - El método filtra únicamente las líneas que contienen los estados, ignorando las líneas que contienen los símbolos de transición.
   - Si uno de los jugadores no tiene rutas ganadoras, el otro jugador es declarado ganador automáticamente y el juego termina.
   - Si ninguno de los jugadores tiene rutas ganadoras, ambos son declarados perdedores y el juego termina.

2. **Inicialización de variables de estado:**

   - Se selecciona la primera ruta de cada jugador como la *jugada actual*. Estas rutas están formadas por secuencias de estados que representan los movimientos permitidos para cada jugador. La ruta del jugador 1 se almacena en `jugadaActual1` y la del jugador 2 en `jugadaActual2`.
   - Los estados iniciales de los jugadores (`estadoActual1` y `estadoActual2`) se inicializan al primer estado de sus respectivas rutas.
   - Las variables `contadorEstado1` y `contadorEstado2` se utilizan para controlar el progreso de los jugadores a lo largo de sus rutas. Comienzan en 0 y se incrementan conforme los jugadores avanzan.

3. **Inicialización del tablero:**

   - El método `inicializar_tablero()` se llama para generar la configuración inicial del tablero. Este tablero se representa como una lista que contiene los colores de las casillas y las posiciones iniciales de los jugadores.
   - Las variables `estadoPrevio1` y `estadoPrevio2` se inicializan como `None`, ya que al inicio no hay estados previos.

- Se crea una ventana gráfica mediante `matplotlib`, y el método `mostrar_tablero()` se utiliza para dibujar el tablero inicial con los jugadores en sus posiciones de salida.

4. **Simulación del juego:**

   - La variable `turno` se inicializa con el valor de `primero`, que determina aleatoriamente qué jugador comienza. El juego alterna los turnos entre los jugadores.
   - **Turno del jugador 1:**
     - Durante el turno del jugador 1, el método verifica si `contadorEstado1` puede incrementarse (es decir, si el jugador aún tiene movimientos en su ruta). Si es así, se calcula `siguienteEstado1`, que es el próximo estado en la ruta.
     - Si el próximo estado no está ocupado por el jugador 2, el jugador 1 avanza a `siguienteEstado1`. Se actualiza el estado anterior (`estadoPrevio1`) y el actual (`estadoActual1`), y se llama a `actualizar_tablero()` para mover al jugador 1 en el tablero.
     - Si el jugador 1 alcanza su último estado, el método imprime un mensaje de victoria y el bucle termina.
     - Si el próximo estado está ocupado por el jugador 2, el jugador 1 intenta reconfigurar su ruta. El método busca en las rutas almacenadas si existe una alternativa que permita al jugador evitar al jugador 2. Si se encuentra una nueva ruta válida (cuyo estado actual sea el mismo que el de la ruta actual y su estado siguiente sea diferente al estado actual del jugador 2), el jugador 1 la sigue; de lo contrario, cede su turno al jugador 2.
   - **Turno del jugador 2:**
     - El proceso para el jugador 2 es análogo. Se verifica si el jugador puede avanzar en su ruta. Si el próximo estado está disponible, se actualiza la posición del jugador 2 en el tablero mediante `actualizar_tablero()`.
     - Si el jugador 2 alcanza su último estado, se imprime un mensaje de victoria y el juego termina.
     - Si el próximo estado está ocupado por el jugador 1, el jugador 2 intenta reconfigurar su ruta de la misma manera que el jugador 1. Si no puede encontrar una nueva ruta válida, cede su turno al jugador 1.

5. **Manejo de reconfiguraciones:**

    - Si durante el turno de un jugador, este no puede avanzar porque su próximo estado está ocupado por el otro jugador, el método intentará encontrar una ruta alternativa en las líneas restantes de las rutas ganadoras almacenadas.
    - Si se encuentra una nueva ruta cuyo próximo estado no está ocupado, el jugador tomará esta nueva ruta y continuará el juego. Si no se encuentra una alternativa, el jugador pierde su turno.

6. **Cierre de la ventana gráfica:**

    - Después de que uno de los jugadores ha ganado, el método `plt.show()` se llama para mantener la ventana del tablero abierta, permitiendo que el usuario observe el estado final del tablero.

Este método encapsula la lógica del juego, gestionando los movimientos de los jugadores, las reconfiguraciones de ruta en caso de conflictos, y la visualización en tiempo real de los avances en el tablero. El uso de rutas predefinidas para ambos jugadores, con la posibilidad de reconfigurar sus caminos en situaciones conflictivas, hace que el juego sea dinámico y no determinista. Además, como las rutas generadas contienen tanto los estados como los símbolos de transición utilizados, es posible visualizar el comportamiento completo de los jugadores durante la partida.

2.1.7 `vaciarArchivos()`

Este método se asegura de que los archivos de rutas, ganadoras y perdedoras, estén vacíos antes de comenzar una nueva ejecución del juego.

2.1.8 `generarCadena(tamañoCadena)`

Este método genera una cadena de movimientos aleatoria para los jugadores.

1. Crea una cadena de longitud variable, utilizando los símbolos 'r' y 'b'.
2. El último símbolo siempre es 'b' para asegurar que el último movimiento sea hacia una casilla negra.

2.1.9 `graficar_rutas(archivo_rutas, jugador)`

El método `graficar_rutas` genera una representación visual de las rutas de los jugadores utilizando la librería `networkx` para crear un grafo dirigido. Esta gráfica permite observar las posibles rutas ganadoras de un jugador en el tablero de ajedrez y cómo se conectan entre diferentes estados (casillas), junto con las transiciones que indican los símbolos ('r' o 'b') que se utilizaron para pasar de un estado a otro.

1. **Lectura del archivo de rutas:**
    - El método recibe como entrada el nombre del archivo que contiene las rutas ganadoras para un jugador (`archivo_rutas`) y el número del jugador (`jugador`).
    - Utilizando una operación de apertura de archivos estándar en Python (`open`), se leen todas las líneas del archivo. En este archivo, las líneas impares contienen los estados por los que pasa el jugador, mientras que las líneas pares contienen los símbolos de transición (`'r'` o `'b'`) que indican el tipo de movimiento entre estados.
    - Las rutas y sus respectivas transiciones se almacenan en una lista llamada `lineas`.
2. **Creación del grafo dirigido:**
    - Se crea una instancia de un grafo dirigido (`DiGraph`) utilizando la librería `networkx`.
    - Este grafo dirigido permitirá visualizar tanto los estados por los que pasa un jugador desde su estado inicial hasta su estado final, como los símbolos de transición que indican el tipo de movimiento entre esos estados.
3. **Añadir nodos y arcos al grafo:**
    - Se itera sobre las líneas de `lineas`, procesando las rutas y sus transiciones en pares: las líneas impares contienen los estados y las líneas pares contienen los símbolos de transición asociados a esos estados.
    - Para cada par de líneas, la línea de estados se convierte en una lista de enteros, donde cada entero representa un estado del tablero, mientras que la línea de transiciones se convierte en una lista de símbolos (`'r'` o `'b'`).

- Para cada estado en la ruta (excepto el último), se añade un arco al grafo dirigido conectando el estado actual con el siguiente estado en la secuencia. El símbolo de transición correspondiente se añade como etiqueta del arco, utilizando el atributo `label`.
- Así, se crean todos los arcos entre los estados por los que pasa el jugador en cada ruta, mostrando tanto la conexión entre estados como el símbolo de transición que se utilizó para pasar de un estado a otro.

4. **Generación de posiciones para los nodos:**

   - Una vez añadido el conjunto de nodos y arcos al grafo, se genera un layout o disposición gráfica de los nodos. Para ello, se utiliza la función `spring_layout` de `networkx`, que distribuye los nodos del grafo en una disposición visualmente equilibrada y comprensible.
   - Esta disposición utiliza un modelo de resorte para posicionar los nodos de forma que las conexiones se vean claras, minimizando el solapamiento entre nodos y arcos.
   - La variable `pos` almacena las coordenadas de cada nodo (estado) en este layout.

5. **Dibujo del grafo:**

   - Utilizando la función `draw` de `networkx`, se dibuja el grafo con los siguientes elementos para mejorar la visualización:
     - `with_labels=True`: Muestra etiquetas en cada nodo, correspondientes al número del estado en el tablero.
     - `node_size=500`: Define el tamaño de los nodos, haciendo que sean lo suficientemente grandes para que las etiquetas sean legibles.
     - `node_color='lightblue'`: Los nodos se colorean de azul claro para destacarse visualmente sobre el fondo.
     - `font_size=10` y `font_weight='bold'`: Las etiquetas de los nodos se muestran en un tamaño de fuente legible y en negrita.
     - `arrowstyle='->'` y `arrowsize=15`: Los arcos entre los nodos se dibujan con flechas que indican la dirección de la transición entre los estados, y el tamaño de la flecha se ajusta para que sea claramente visible.
   - Además, se dibujan las etiquetas de los arcos (las transiciones `'r'` o `'b'`) utilizando la función `draw_networkx_edge_labels`. Estas

etiquetas muestran el símbolo de transición utilizado para pasar de un estado al siguiente, y se dibujan en color rojo (`font_color='red'`).

6. **Título de la gráfica:**

   - Se añade un título a la gráfica utilizando `plt.title`, que indica el jugador al que pertenecen las rutas que se están visualizando. El título se adapta dinámicamente según el número del jugador recibido como argumento.
   - El título puede ser, por ejemplo, `"Caminos del Jugador 1"` o `"Caminos del Jugador 2"`, lo que facilita la identificación de las rutas graficadas.

7. **Mostrar la gráfica:**

   - Finalmente, la gráfica se muestra en una ventana utilizando `plt.show()`, lo que permite al usuario visualizar la red de caminos ganadores seguidos por el jugador, junto con los símbolos de transición que indican cómo se movió de un estado a otro.

2.1.10 `mostrar_caminos()`

Este método grafica las rutas de ambos jugadores utilizando `graficar_rutas()`.

2.2 Inicialización del Juego

El programa comienza vaciando los archivos de rutas. Luego, el usuario elige entre el modo automático y manual:

- En el modo automático, las cadenas de movimientos se generan aleatoriamente, y las rutas se procesan con el autómata.
- En el modo manual, el usuario ingresa las cadenas de movimientos para ambos jugadores.

2.3 Juego

El jugador que comienza se decide aleatoriamente. Durante el juego, los jugadores se mueven por el tablero siguiendo sus rutas. Si ambos jugadores llegan al mismo estado, el jugador afectado intenta reconfigurar su ruta buscando una nueva en sus rutas ganadoras almacenadas.

El juego finaliza cuando un jugador alcanza su estado final o cuando algún jugador no cuenta con rutas que lo lleven a un estado final

2.4 Código Fuente

```
import random
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.colors as mcolors
import networkx as nx
import time

class AFND:
    def __init__(self, estado_inicial, estado_final):
        self.estado_inicial = estado_inicial
        self.estado_final = estado_final
        self.alfabeto = {'r', 'b'}
        self.transiciones = {1: {'r': [2, 6], 'b': [7]},
                             2: {'r': [6, 8], 'b': [1, 3,
    7]},
                             3: {'r': [2, 4, 8], 'b': [7,
    9]},
                             4: {'r': [8, 10], 'b': [3, 5,
    9]},
                             5: {'r': [4, 10], 'b': [9]},
                             6: {'r': [2, 12], 'b': [1, 7,
    11]},
                             7: {'r': [2, 6, 8, 12], 'b': [1,
     3, 11, 13]},
                             8: {'r': [2, 4, 12, 14], 'b':
    [3, 7, 9, 13]},
                             9: {'r': [4, 8, 10, 14], 'b':
    [3, 5, 13, 15]},
                             10: {'r': [4, 14], 'b': [5, 9,
    15]},
                             11: {'r': [6, 12, 16], 'b': [7,
    17]},
                             12: {'r': [6, 8, 16, 18], 'b':
    [7, 11, 13, 17]},
                             13: {'r': [8, 12, 14, 18], 'b':
    [7, 9, 17, 19]},
                             14: {'r': [8, 10, 18, 20], 'b':
    [9, 13, 15, 19]},
                             15: {'r': [10, 14, 20], 'b': [9,
     19]},
                             16: {'r': [12, 22], 'b': [11,
    17, 21]},
                             17: {'r': [12, 16, 18, 22], 'b':
     [11, 13, 21, 23]},
                             18: {'r': [12, 14, 22, 24], 'b':
     [13, 17, 19, 23]},
                             19: {'r': [14, 18, 20, 24], 'b':
```

```
 [13, 15, 23, 25]},
                            20: {'r': [14, 24], 'b': [15,
19, 25]},
                            21: {'r': [16, 22], 'b': [17]},
                            22: {'r': [16, 18], 'b': [17,
21, 23]},
                            23: {'r': [18, 22, 24], 'b':
[17, 19]},
                            24: {'r': [18, 20], 'b': [19,
23, 25]},
                            25: {'r': [20, 24], 'b': [19]}}

 def procesar(self, cadena, jugador):
     pila = [(self.estado_inicial, cadena, [self.
estado_inicial], [])]

     resultado = False

     while pila:
         estado_actual, cadena_restante, ruta,
transiciones = pila.pop()

         if len(cadena_restante) == 0:
             with open(f"Rutas{jugador}.txt", "a") as
archivo:
                 archivo.write(" ".join(map(str, ruta)) +
"\n")
                 archivo.write(" ".join(transiciones) + "\
n")
             if estado_actual == self.estado_final:
                 resultado = True
                 with open(f"Ganadoras{jugador}.txt", "a")
 as archivo:
                     archivo.write(" ".join(map(str, ruta)
) + "\n")
                     archivo.write(" ".join(transiciones)
+ "\n")
             else:
                 with open(f"Perdedoras{jugador}.txt", "a"
) as archivo:
                     archivo.write(" ".join(map(str, ruta)
) + "\n")
                     archivo.write(" ".join(transiciones)
+ "\n")
             continue

         simbolo = cadena_restante[0]
         estados_siguientes = self.transiciones.get(
estado_actual, {}).get(simbolo, [])
```

```
64              if estados_siguientes == []:
65                  print(f"Error: No hay transiciones
    disponibles desde el estado {estado_actual} con el
    s mbolo '{simbolo}'")
66                  exit()
67              for estado_sig in estados_siguientes:
68                  pila.append((estado_sig, cadena_restante[1:],
     ruta + [estado_sig], transiciones + [simbolo]))
69
70          return resultado
71
72  def inicializar_tablero():
73      tablero = []
74      for row in range(5):
75          for col in range(5):
76              estado = row * 5 + col + 1
77              if estado == 1:
78                  tablero.append(2)  # Azul para el jugador 1
79              elif estado == 5:
80                  tablero.append(3)  # Caf  para el jugador 2
81              elif (row + col) % 2 == 0:
82                  tablero.append(0)  # Negro
83              else:
84                  tablero.append(1)  # Rojo
85      return tablero
86
87  def actualizar_tablero(tablero, estado_actual, estado_previo,
     jugador, ax):
88      # Restaurar la casilla anterior al color original
89      if estado_previo is not None:
90          row_prev = (estado_previo - 1) // 5
91          col_prev = (estado_previo - 1) % 5
92          if (row_prev + col_prev) % 2 == 0:
93              tablero[estado_previo - 1] = 0  # Negro
94          else:
95              tablero[estado_previo - 1] = 1  # Rojo
96
97      # Actualizar la casilla actual con el color del jugador
98      if jugador == 1:
99          tablero[estado_actual - 1] = 2  # Azul para el
    jugador 1
100     elif jugador == 2:
101         tablero[estado_actual - 1] = 3  # Caf  para el
    jugador 2
102
103     mostrar_tablero(tablero, ax)
104
105 def mostrar_tablero(tablero, ax):
106     tablero_2d = np.array(tablero).reshape((5, 5))
```

```
107
108     ax.clear()
109     cmap = mcolors.ListedColormap(['black', 'red', 'blue', '
    brown'])
110     ax.imshow(tablero_2d, cmap=cmap, extent=[0, 5, 0, 5])
111
112     ax.set_xticks(np.arange(6))
113     ax.set_yticks(np.arange(6))
114     ax.grid(True, which='both')
115
116     for i in range(5):
117         for j in range(5):
118             estado = i * 5 + j + 1
119             ax.text(j + 0.5, 4 - i + 0.5, str(estado), ha='
    center', va='center', color='white', fontsize=12)
120
121     ax.set_xticks([])
122     ax.set_yticks([])
123
124     plt.draw()
125     plt.pause(1)
126
127 def juego(ruta1, ruta2, primero):
128     with open(ruta1, "r") as Ganadoras1:
129         lineas1 = Ganadoras1.readlines()
130     with open(ruta2, "r") as Ganadoras2:
131         lineas2 = Ganadoras2.readlines()
132
133     # Filtrar las l neas que contienen los estados (solo las
     l neas impares, ya que las pares contienen los s mbolos
    )
134     jugadas1 = [lineas1[i] for i in range(0, len(lineas1), 2)
    ]  # Solo las l neas impares (0, 2, 4, ...)
135     jugadas2 = [lineas2[i] for i in range(0, len(lineas2), 2)
    ]  # Solo las l neas impares
136
137     if len(jugadas1) >= 1:
138         if len(jugadas2) >= 1:
139             jugadaActual1 = [int(estado) for estado in
    jugadas1[0].strip().split()]
140             jugadaActual2 = [int(estado) for estado in
    jugadas2[0].strip().split()]
141         else:
142             print("El jugador 2 no tiene jugadas ganadoras.
    Jugador 1 gana por default.")
143             exit()
144     else:
145         if len(jugadas2) >= 1:
146             print("El jugador 1 no tiene jugadas ganadoras.
```

```
    Jugador 2 gana por default.")
147             exit()
148         else:
149             print("Ning n jugador tiene jugadas ganadoras.
    Ambos pierden.")
150             exit()
151 
152     contadorEstado1 = contadorEstado2 = 0
153     contadorJugada1 = contadorJugada2 = 0
154     topeContador = len(jugadaActual1) - 1  # Todas las
    jugadas tienen la misma longitud
155 
156     # Inicializar el tablero con los colores predeterminados
    de los jugadores 1 y 2
157     tablero = inicializar_tablero()
158     estadoActual1 = jugadaActual1[0]
159     estadoActual2 = jugadaActual2[0]
160     estadoPrevio1 = estadoPrevio2 = None
161 
162     # Crear la figura para graficar
163     fig, ax = plt.subplots()
164     mostrar_tablero(tablero, ax)
165 
166     # Esperar 1 segundo antes de empezar
167     time.sleep(1)
168 
169     turno = primero  # 1 o 2
170 
171     while True:
172         if turno == 1:
173             # Movimiento del jugador 1
174             if contadorEstado1 + 1 <= topeContador:
175                 siguienteEstado1 = jugadaActual1[
    contadorEstado1 + 1]
176                 if siguienteEstado1 != estadoActual2:
177                     estadoPrevio1 = estadoActual1
178                     estadoActual1 = siguienteEstado1
179                     contadorEstado1 += 1
180                     actualizar_tablero(tablero, estadoActual1
    , estadoPrevio1, 1, ax)
181                     print(f"Jug1: {estadoActual1}")
182                     if contadorEstado1 == topeContador:
183                         print("Jugador 1 Gana.")
184                         break
185                     turno = 2  # Cambiar el turno al jugador
    2
186                 else:
187                     # Intentar reconfigurar la ruta
188                     ruta_encontrada = False
```

```
189                     for i in range(contadorJugada1 + 1, len(
    jugadas1)):
190                         posible_jugada = [int(estado) for
    estado in jugadas1[i].strip().split()]
191                         if posible_jugada[contadorEstado1] ==
     estadoActual1 and posible_jugada[contadorEstado1 + 1] !=
    estadoActual2:
192                             jugadaActual1 = posible_jugada
193                             contadorJugada1 = i
194                             siguienteEstado1 = jugadaActual1[
    contadorEstado1 + 1]
195                             estadoPrevio1 = estadoActual1
196                             estadoActual1 = siguienteEstado1
197                             contadorEstado1 += 1
198                             actualizar_tablero(tablero,
    estadoActual1, estadoPrevio1, 1, ax)
199                             print(f"Jugador 1 reconfigur  su
     ruta a: {jugadaActual1}")
200                             print(f"Jug1: {estadoActual1}")
201                             if contadorEstado1 ==
    topeContador:
202                                 print("Jugador 1 Gana.")
203                                 plt.show()
204                                 return
205                             ruta_encontrada = True
206                             turno = 2
207                             break
208                     if not ruta_encontrada:
209                         print(f"Jugador 1 no puede moverse en
     el turno {contadorEstado1 + 1}. Cede el turno.")
210                         turno = 2  # Ceder el turno al
    jugador 2
211             else:
212                 print("Jugador 1 no tiene m s movimientos.")
213                 turno = 2
214 
215         else:
216             # Movimiento del jugador 2
217             if contadorEstado2 + 1 <= topeContador:
218                 siguienteEstado2 = jugadaActual2[
    contadorEstado2 + 1]
219                 if siguienteEstado2 != estadoActual1:
220                     estadoPrevio2 = estadoActual2
221                     estadoActual2 = siguienteEstado2
222                     contadorEstado2 += 1
223                     actualizar_tablero(tablero, estadoActual2
    , estadoPrevio2, 2, ax)
224                     print(f"Jug2: {estadoActual2}")
225                     if contadorEstado2 == topeContador:
```

```
226                         print("Jugador 2 Gana.")
227                         break
228                     turno = 1  # Cambiar el turno al jugador
    1
229                 else:
230                     # Intentar reconfigurar la ruta
231                     ruta_encontrada = False
232                     for i in range(contadorJugada2 + 1, len(
    jugadas2)):
233                         posible_jugada = [int(estado) for
    estado in jugadas2[i].strip().split()]
234                         if posible_jugada[contadorEstado2] ==
     estadoActual2 and posible_jugada[contadorEstado2 + 1] !=
    estadoActual1:
235                             jugadaActual2 = posible_jugada
236                             contadorJugada2 = i
237                             siguienteEstado2 = jugadaActual2[
    contadorEstado2 + 1]
238                             estadoPrevio2 = estadoActual2
239                             estadoActual2 = siguienteEstado2
240                             contadorEstado2 += 1
241                             actualizar_tablero(tablero,
    estadoActual2, estadoPrevio2, 2, ax)
242                             print(f"Jugador 2 reconfigur  su
     ruta a: {jugadaActual2}")
243                             print(f"Jug2: {estadoActual2}")
244                             if contadorEstado2 ==
    topeContador:
245                                 print("Jugador 2 Gana.")
246                                 plt.show()
247                                 return
248                             ruta_encontrada = True
249                             turno = 1
250                             break
251                     if not ruta_encontrada:
252                         print(f"Jugador 2 no puede moverse en
     el turno {contadorEstado2 + 1}. Cede el turno.")
253                         turno = 1  # Ceder el turno al
    jugador 1
254             else:
255                 print("Jugador 2 no tiene m s movimientos.")
256                 turno = 1
257
258
259     # Mantener la ventana abierta al finalizar el juego
260     plt.show()
261
262 def vaciarArchivos():
263     with open("Rutas1.txt", "w") as archivo:
```

```
264         archivo.write("")
265     with open("Ganadoras1.txt", "w") as archivo:
266         archivo.write("")
267     with open("Perdedoras1.txt", "w") as archivo:
268         archivo.write("")
269     with open("Rutas2.txt", "w") as archivo:
270         archivo.write("")
271     with open("Ganadoras2.txt", "w") as archivo:
272         archivo.write("")
273     with open("Perdedoras2.txt", "w") as archivo:
274         archivo.write("")
275 
276 def generarCadena(tama oCadena):
277     cadena = ""
278     for i in range(tama oCadena - 1):
279         cadena += random.choice(['r', 'b'])
280     cadena += 'b'
281     return cadena
282 
283 def graficar_rutas(archivo_rutas, jugador):
284     with open(archivo_rutas, "r") as archivo:
285         lineas = archivo.readlines()
286 
287     # Crear un grafo dirigido
288     Grafo = nx.DiGraph()
289 
290     # Procesar las rutas y sus transiciones (estados en una
    l nea , s mbolos en la siguiente)
291     for i in range(0, len(lineas), 2):
292         estados = list(map(int, lineas[i].strip().split()))
293         transiciones = lineas[i+1].strip().split()
294 
295         # Agregar los nodos y arcos al grafo con las
    etiquetas correspondientes
296         for j in range(len(estados) - 1):
297             Grafo.add_edge(estados[j], estados[j + 1], label=
    transiciones[j])
298 
299     # Obtener posiciones arbitrarias de los nodos usando un
    layout de resorte
300     pos = nx.spring_layout(Grafo)
301 
302     # Dibujar el grafo
303     plt.figure(figsize=(8, 8))
304     nx.draw(Grafo, pos, with_labels=True, node_size=500,
    node_color='lightblue', font_size=10, font_weight='bold',
    arrowstyle='->', arrowsize=15)
305 
306     # Dibujar las etiquetas de los arcos (las transiciones 'r
```

```
    ' o 'b')
307     edge_labels = nx.get_edge_attributes(Grafo, 'label')
308     nx.draw_networkx_edge_labels(Grafo, pos, edge_labels=
    edge_labels, font_color='red')
309 
310     # A adir t tulo
311     plt.title(f"Caminos del Jugador {jugador}")
312     plt.show()
313 
314 def mostrar_caminos():
315     graficar_rutas("Ganadoras1.txt", 1)
316     graficar_rutas("Ganadoras2.txt", 2)
317     # Mostrar ambas gr ficas juntas
318     plt.show()
319 
320 def main():
321     vaciarArchivos()
322     print("Bienvenido al juego del Tablero")
323     print("1. Modo Autom tico\n2. Modo Manual")
324     while True:
325         try:
326             opcion = int(input("Seleccione una opci n (1 o
    2): "))
327             if opcion in [1, 2]:
328                 break
329             else:
330                 print("Por favor, ingrese 1 o 2.")
331         except ValueError:
332             print("Entrada no v lida. Por favor, ingrese un
    n mero entero.")
333 
334     if opcion == 1:
335         rey1 = AFND(1, 25)
336         rey2 = AFND(5, 21)
337         tama oCadena = random.randint(5, 100)
338         cadena1 = generarCadena(tama oCadena)
339         cadena2 = generarCadena(tama oCadena)
340 
341         print(f"Cadena para el jugador 1: {cadena1}")
342         print(f"Cadena para el jugador 2: {cadena2}")
343 
344         rey1.procesar(cadena1, 1)
345         rey2.procesar(cadena2, 2)
346         primero = random.choice([1, 2])
347 
348         print(f"Empieza el jugador {primero}")
349         juego("Ganadoras1.txt", "Ganadoras2.txt", primero)
350         mostrar_caminos()
351     else:
```

```
352         rey1 = AFND(1, 25)
353         rey2 = AFND(5, 21)
354         while True:
355             cadena1 = input("Ingrese la cadena con la que
    jugar  el jugador 1:\n")
356             if len(cadena1) < 5 or len(cadena1) > 100:
357                 print("Por favor, ingrese una cadena de entre
     5 y 100 caracteres de longitud.")
358             else:
359                 break
360         while True:
361             cadena2 = input("Ingrese la cadena con la que
    jugar  el jugador 2:\n")
362             if len(cadena2) == len(cadena1):
363                 break
364             else:
365                 print(f"Por favor, ingrese una cadena de la
    misma longitud. La longitud de la primera cadena es: {len(
    cadena1)} ")
366
367         print(f"Cadena para el jugador 1: {cadena1}")
368         print(f"Cadena para el jugador 2: {cadena2}")
369
370         if cadena1[-1] == 'r':
371             if cadena2[-1] == 'r':
372                 print("Ning n jugador tiene jugadas
    ganadoras. Ambos pierden.")
373                 exit()
374             else:
375                 print("El jugador 1 no tiene jugadas
    ganadoras. Jugador 2 gana por default.")
376                 exit()
377         else:
378             if cadena2[-1] == 'r':
379                 print("El jugador 2 no tiene jugadas
    ganadoras. Jugador 1 gana por default.")
380                 exit()
381
382         rey1.procesar(cadena1, 1)
383         rey2.procesar(cadena2, 2)
384         primero = random.choice([1, 2])
385
386         print(f"Empieza el jugador {primero}")
387         juego("Ganadoras1.txt", "Ganadoras2.txt", primero)
388         mostrar_caminos()
389
390 if __name__ == "__main__":
```

```
391     main()
```

Listing 1: Código Fuente del Programa

3 Ejecución

A continuación, se presentan los resultados de la ejecución del programa.

3.1 Ejecución del Código

```
Bienvenido al juego del Tablero
1. Modo Automático
2. Modo Manual
Seleccione una opción (1 o 2): 2
Ingrese la cadena con la que jugará el jugador 1:
rrrbbb
Ingrese la cadena con la que jugará el jugador 2:
brbrbb
Cadena para el jugador 1: rrrbbb
Cadena para el jugador 2: brbrbb
Empieza el jugador 2
Jug2: 9
Jug1: 6
Jug2: 14
Jug1: 12
Jug2: 19
Jug1: 18
Jugador 2 no puede moverse en el turno 4. Cede el turno.
Jug1: 23
Jug2: 18
Jug1: 19
Jug2: 17
Jug1: 25
Jugador 1 Gana.
```

3.2 Archivos Generados

```
1 6 12 18 23 19 25
r r r b b b
1 6 12 18 23 19 23
r r r b b b
1 6 12 18 23 19 15
r r r b b b
1 6 12 18 23 19 13
r r r b b b
1 6 12 18 23 17 23
r r r b b b
1 6 12 18 23 17 21
r r r b b b
1 6 12 18 23 17 13
r r r b b b
1 6 12 18 23 17 11
r r r b b b
1 6 12 18 19 25 19
r r r b b b
```

Figure 1: Archivo de rutas del jugador 1

```
5 9 14 19 24 25 19
b r b r b b
5 9 14 19 24 23 19
b r b r b b
5 9 14 19 24 23 17
b r b r b b
5 9 14 19 24 19 25
b r b r b b
5 9 14 19 24 19 23
b r b r b b
5 9 14 19 24 19 15
b r b r b b
5 9 14 19 24 19 13
b r b r b b
5 9 14 19 20 25 19
b r b r b b
5 9 14 19 20 19 25
b r b r b b
5 9 14 19 20 19 23
b r b r b b
```

Figure 2: Archivo de rutas del jugador 2

```
1 6 12 18 23 19 25
r r r b b b
1 6 12 18 13 19 25
r r r b b b
1 6 12 8 13 19 25
r r r b b b
1 6 2 8 13 19 25
r r r b b b
1 2 8 14 15 19 25
r r r b b b
1 2 8 14 13 19 25
r r r b b b
1 2 8 12 13 19 25
r r r b b b
1 2 6 12 13 19 25
r r r b b b
```

Figure 3: Archivo de rutas ganadoras del jugador 1

```
5 9 14 19 18 17 21
b r b r b b
5 9 14 13 18 17 21
b r b r b b
5 9 14 13 12 17 21
b r b r b b
5 9 8 13 18 17 21
b r b r b b
5 9 8 13 12 17 21
b r b r b b
5 9 8 7 12 17 21
b r b r b b
```

Figure 4: Archivo de rutas ganadoras del jugador 2

```
1 6 12 18 23 19 23
r r r b b b
1 6 12 18 23 19 15
r r r b b b
1 6 12 18 23 19 13
r r r b b b
1 6 12 18 23 17 23
r r r b b b
1 6 12 18 23 17 21
r r r b b b
1 6 12 18 23 17 13
r r r b b b
1 6 12 18 23 17 11
r r r b b b
1 6 12 18 19 25 19
r r r b b b
1 6 12 18 19 23 19
r r r b b b
```

Figure 5: Archivo de rutas perdedoras del jugador 1

```
5 9 14 19 24 25 19
b r b r b b
5 9 14 19 24 23 19
b r b r b b
5 9 14 19 24 23 17
b r b r b b
5 9 14 19 24 19 25
b r b r b b
5 9 14 19 24 19 23
b r b r b b
5 9 14 19 24 19 15
b r b r b b
5 9 14 19 24 19 13
b r b r b b
5 9 14 19 20 25 19
b r b r b b
5 9 14 19 20 19 25
b r b r b b
```

Figure 6: Archivo de rutas perdedoras del jugador 2

3.3 Animaciones del Tablero

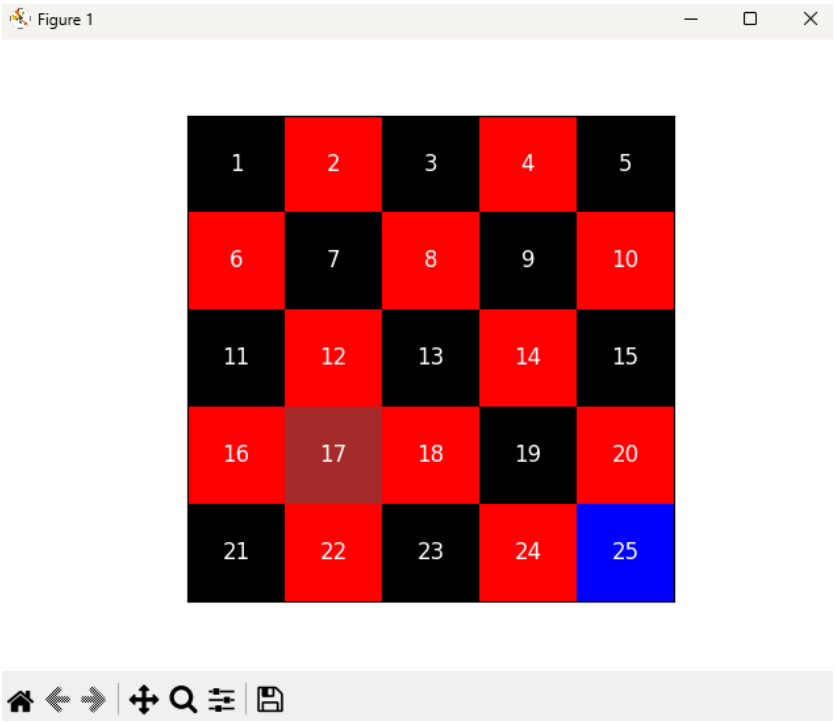

Figure 7: Captura de pantalla del tablero

3.4 Gráficas

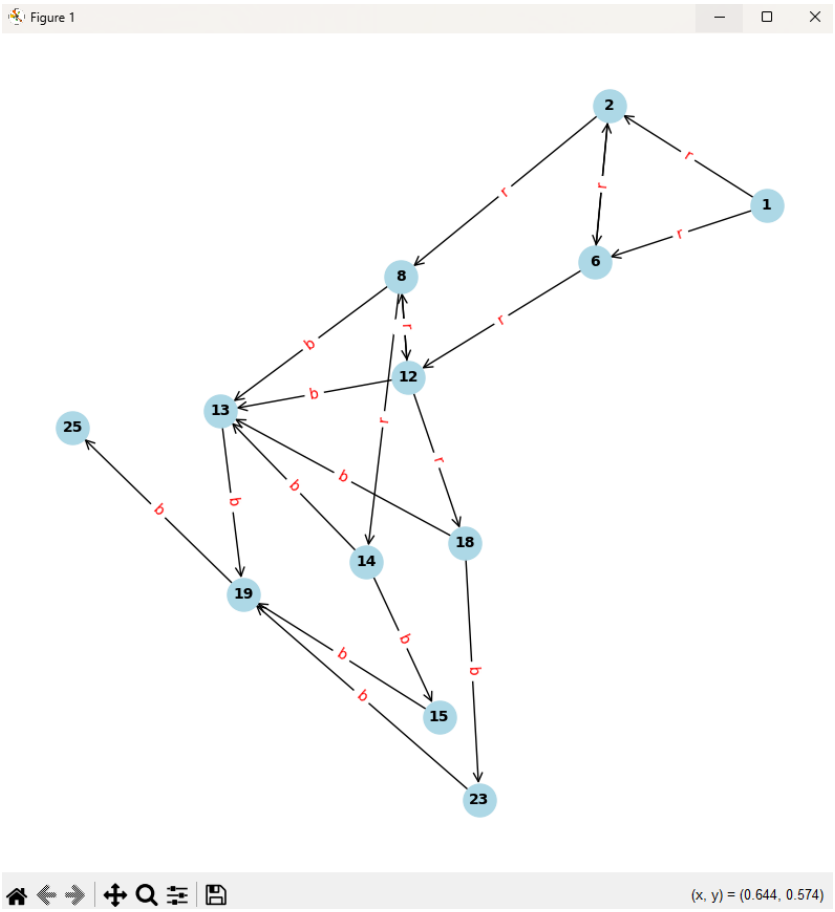

Figure 8: Red de movimientos del jugador 1

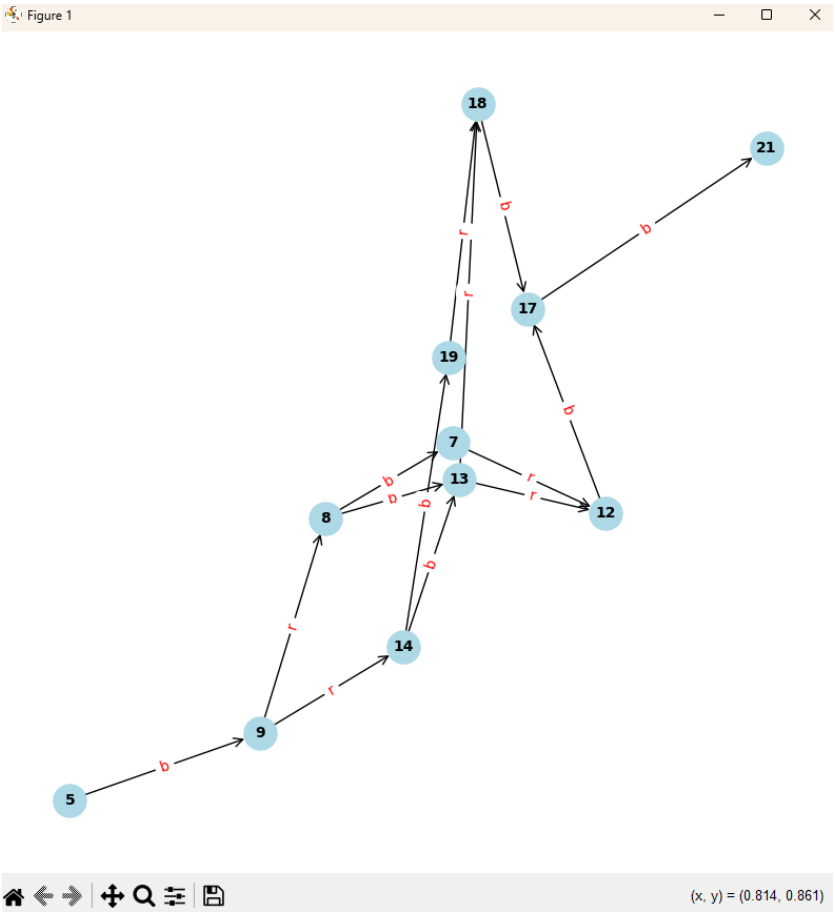

Figure 9: Red de movimientos del jugador 2

4 Conclusión

El programa "Tablero" proporciona una simulación completa y visualmente atractiva del movimiento de dos jugadores en un tablero de ajedrez de 5x5, utilizando autómatas finitos no deterministas (AFND). A través de esta implementación, se ha demostrado cómo los conceptos de teoría de autómatas pueden aplicarse para modelar y resolver problemas de simulación de trayectorias en un espacio finito con múltiples opciones de movimiento.

Los jugadores actúan como piezas de rey, moviéndose de manera no determinista por el tablero, mientras el programa calcula sus rutas y las almacena en archivos de texto. Las rutas ganadoras y perdedoras son analizadas y reconfiguradas en caso de conflictos, lo que añade un componente estratégico a la simulación. La visualización gráfica del tablero y las redes de movimientos de los jugadores facilitan la comprensión del estado y evolución del juego.

Además, el manejo eficiente de las estructuras de datos, como las pilas y los archivos de rutas, garantiza que el programa pueda manejar las complejidades de múltiples caminos y situaciones inesperadas. El uso de `matplotlib` y `networkx` para la representación gráfica hace que el programa sea interactivo y accesible para los usuarios.

5 Referencias

1. Hopcroft, J. E., Motwani, R., Ullman, J. D. (2006). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

2. Oracle. (s.f.). *Python 3.8 Documentation*. [Online] Disponible en: `https://docs.python.org/3.8/`.

3. Ullman, J. (2010). *Automata, Computation, and Complexity Lecture Notes*, Stanford University. [Online] Disponible en: `http://infolab.stanford.edu/~ullman/ialc/spr10/spr10.html#LECTURE%20NOTES`.