Escuela Superior de Cómputo

Teoría de la Computación

Práctica 4:

Tablero

Autor:

Rodrigo Gerardo Trejo Arriaga

Octubre 2023

Práctica 4:
Tablero

Los Autómatas Finitos No Deterministas (AFND) son un concepto fundamental en la teoría de la computación y la teoría de lenguajes formales. Son una extensión de los Autómatas Finitos Deterministas (AFD) y se caracterizan por su capacidad para tener múltiples transiciones posibles desde un estado dado en respuesta a un símbolo de entrada o incluso la posibilidad de no tener una transición definida. Esto contrasta con los AFD, que siguen un proceso determinista y tienen una única transición para cada combinación de estado y símbolo.

Los componentes de un AFND son similares a los de un AFD:

- **Conjunto de Estados (Q):** Un conjunto finito de estados que representa los diferentes estados internos de la máquina. Por ejemplo, $Q = \{q_0, q_1, q_2\}$.
- **Alfabeto de Entrada (Σ):** Un conjunto finito de símbolos utilizados como entrada. Por ejemplo, $\Sigma = \{0, 1\}$.
- **Función de Transición (δ):** A diferencia de un AFD, la función de transición en un AFND puede devolver múltiples estados o incluso el estado vacío $\emptyset$. Formalmente, $\delta : Q \times \Sigma \to 2^Q$, donde 2^Q representa el conjunto de subconjuntos de Q.
- **Estado Inicial (q_0):** El estado en el que se encuentra la máquina al comienzo de la ejecución.
- **Conjunto de Estados Finales (F):** Un subconjunto de estados que se consideran estados de aceptación. Si la máquina termina en al menos uno de estos estados después de procesar la entrada, se acepta la cadena.

El proceso de reconocimiento en un AFND es similar al de un AFD, pero con la posibilidad de múltiples transiciones o la falta de transiciones en ciertos casos. Un AFND puede explorar múltiples ramas posibles durante el procesamiento de una cadena.

Una cadena se considera aceptada por un AFND si existe al menos una secuencia de transiciones que lleve a la máquina a un estado final después de procesar toda la entrada. No es necesario que todas las transiciones sean deterministas.

El lenguaje aceptado por un AFND es el conjunto de todas las cadenas que la máquina acepta, de acuerdo con la definición de aceptación anterior. Es importante destacar que un AFND puede aceptar lenguajes más generales que los AFD.

Es importante notar que cualquier lenguaje aceptado por un AFND también puede ser aceptado por un AFD. Existe un procedimiento para convertir un AFND en un AFD, lo que significa que los lenguajes regulares reconocidos por AFND y AFD son equivalentes. Sin embargo, los AFND son más expresivos y pueden representar de manera más concisa ciertos patrones en lenguajes regulares.

Al igual que los AFD, los Autómatas Finitos No Deterministas son adecuados para reconocer lenguajes regulares. No pueden reconocer lenguajes más complejos, como los lenguajes libres de contexto o los

lenguajes sensibles al contexto. Para lenguajes más complejos, se utilizan otros tipos de autómatas, como los autómatas de pila o las máquinas de Turing.

I. Instrucciones

A continuación, se presentan las instrucciones para el programa, que debe ser capaz de ejecutarse en modo automático y manual:

1. **Modos de Ejecución:** El programa debe permitir dos modos de ejecución: automático y manual.
2. **Entrada de Movimientos:** En el modo manual, el usuario podrá introducir la cadena de movimientos o generarla aleatoriamente.
3. **Número de Piezas:** El programa puede funcionar con una pieza o dos piezas.
4. **Segunda Pieza:** En el caso de dos piezas, la segunda pieza debe iniciar en el estado 4 y su estado final es el estado 13.
5. **Iniciativa Aleatoria:** Cuando inicie el juego, de manera aleatoria el programa debe decidir quién inicia.
6. **Generación de Archivos:** Una vez definida la cadena de movimientos para una o dos piezas, el programa debe generar dos tipos de archivos:

   a) Archivo con todos los movimientos posibles por pieza.

   b) Archivo con todos los movimientos ganadores por pieza.

   Estos archivos servirán para reconfigurar las rutas durante el juego.
7. **Reconfiguración de Rutas:** Si se reconfigura una ruta y aún así no se puede avanzar, el programa debe esperar una iteración para continuar.
8. **Gráficos:** El programa debe ser capaz de graficar el tablero y mostrar los movimientos de una o dos piezas. Además, debe generar una representación gráfica de toda la red generada por los movimientos.
9. **Número de Movimientos:** El número máximo de movimientos permitidos debe estar entre 4 y 50 símbolos.

II. Resultados

III. Código de Implementación

III.1. Programa principal

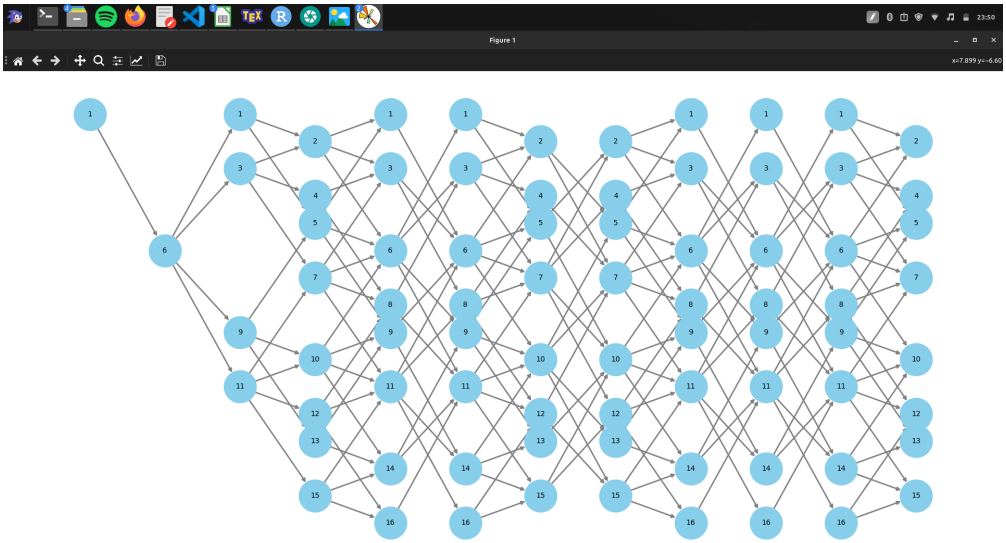

Figura 1: Grafo de la pieza rosa

```
import pygame
import sys
import time
import os
from rutas import *
import random
import time
import multiprocessing
import pandas as pd
import matplotlib.pyplot as plt
import networkx as nx



def eliminar_archs(nombre_arch):
"""Funci n que elimina un archivo si existe en el directorio

Args:
nombre_arch (str): Nombre del archivo que deseas eliminar
"""
archivo1 = nombre_arch
if os.path.exists(archivo1):
os.remove(archivo1)


class Tablero:
def __init__(self, dimension, tablero, indices, num_movs,
    cant_fichas=1):
```

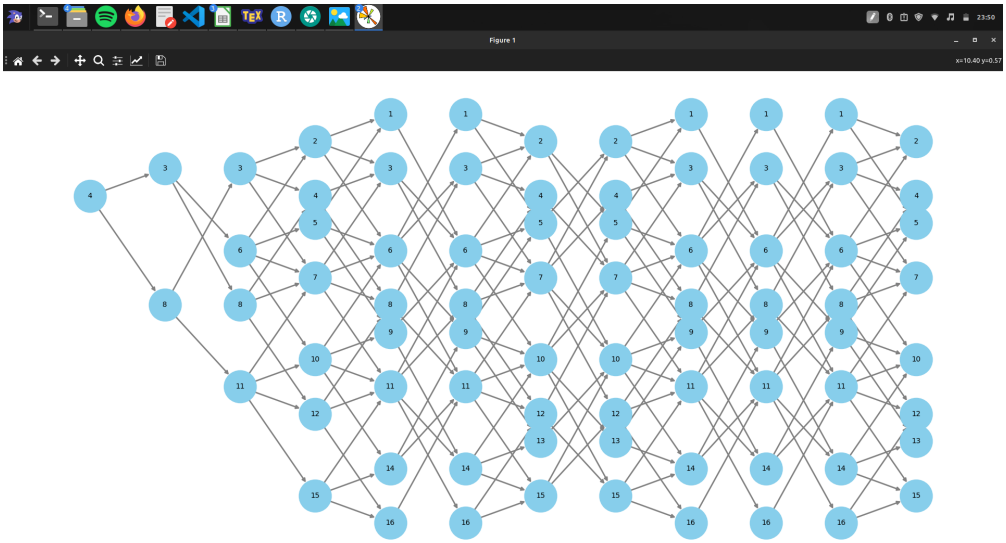

Figura 2: Grafo de la pieza amarilla

```
28      self.DIMENSIONES = dimension
29      self.TAMANO_CELDA = 100
30      self.ROJO = (124, 18, 66)
31      self.NEGRO = (0, 0, 0)
32      pygame.init()
33      self.tablero = tablero
34      self.indices = indices
35      self.cant_fichas = cant_fichas
36      self.num_movs = num_movs
37      self.VENTANA_TAMANO = (self.DIMENSIONES * self.TAMANO_CELDA, self
            .DIMENSIONES * self.TAMANO_CELDA)
38      self.ventana = pygame.display.set_mode(self.VENTANA_TAMANO)
39      pygame.display.set_caption("Simulaci n de Ajedrez")
40
41      self.movs_casillas_ficha1 = None
42      self.movs_casillas_ficha16 = None
43
44      self.pieza1_pos = (0, 0)
45      self.pieza16_pos = (0, 3)
46      self.tiempo_siguiente_movimiento = 6
47      self.indice_movimiento_p1 = 0
48      self.indice_movimiento_p16 = 0
49      self.archivo_pieza1 = open("f1_ganadoras.txt", "w+")
50      self.archivo_pieza16 = open("f16_ganadoras.txt", "w+")
51      self.aux_pos_p1 = (0,0)
52      self.aux_pos_p16 = (0,3)
53      self.ruta_pieza1 = None
54      self.ruta_pieza16 = None
55
```

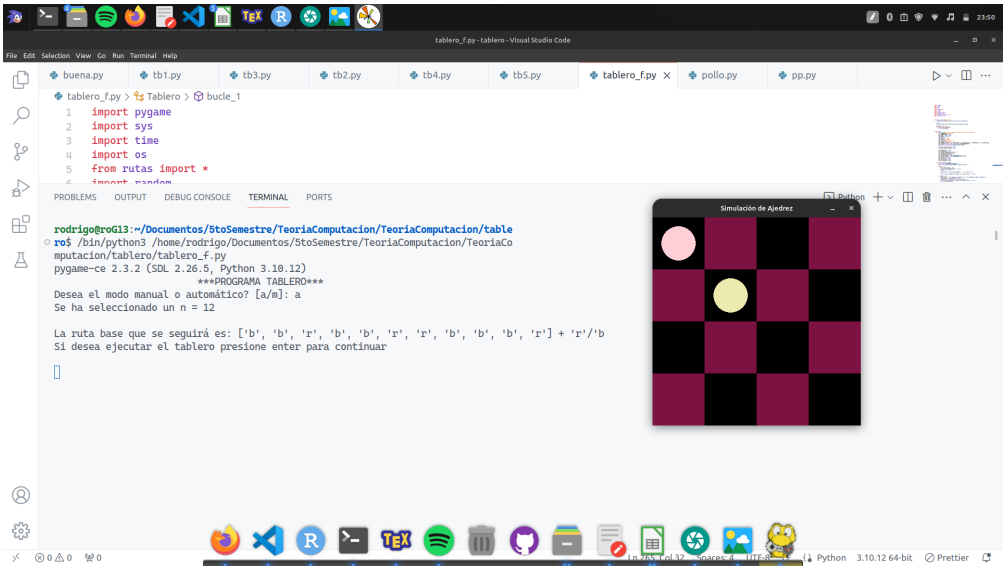

Figura 3: Programa en ejecución

```
56      def calcular_ganadoras(self):
57      print("\t\t\t ***PROGRAMA TABLERO***")
58      modo = input("Desea el modo manual o autom tico? [a/m]: ")
59
60      if modo == "a":
61      n = random.randint(4, 20)
62      print(f"Se ha seleccionado un n = {n}")
63      cadenas = ["r", "b"]
64
65      ruta_colores = [random.choice(cadenas) for _ in range(0, n-1)]
66
67      print(f"\nLa ruta base que se seguir  es: {ruta_colores} + 'r'/'
            b")
68
69      elif modo == "m":
70      ruta_colores = input("Ingrese las cadena de 'r' y 'b' separadas
            por comas  nicamente : ")
71      ruta_colores = ruta_colores.split(",")
72      ruta_colores = [i.lower() for i in ruta_colores]
73      print(f"\nLa ruta base que se seguir  es: {ruta_colores} + 'r'/'
            b")
74      n = len(ruta_colores) +1
75
76
77      resultados1 = {}
78      resultados16 = {}
79
80      q1 = multiprocessing.Queue()
81      q2 = multiprocessing.Queue()
```

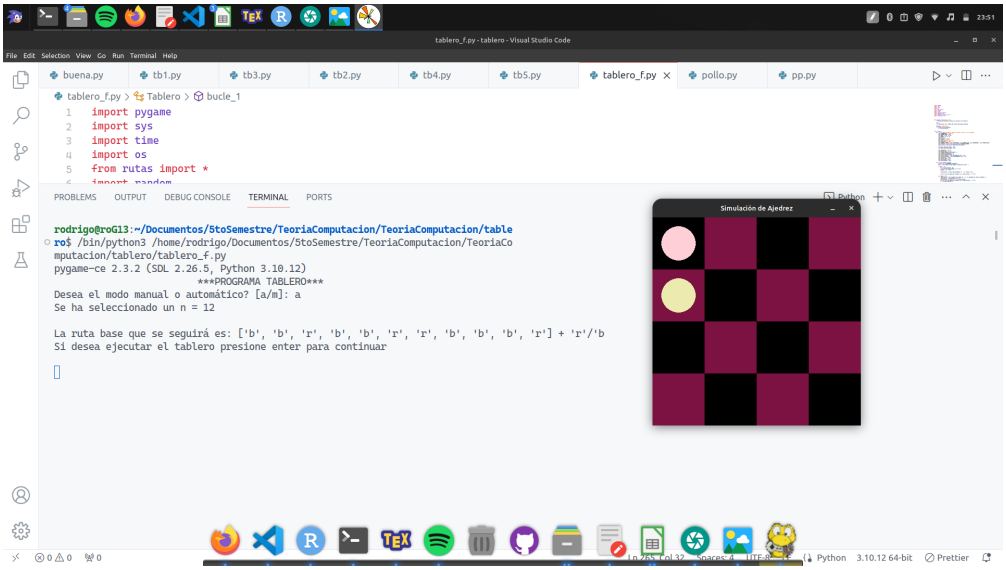

Figura 4: Programa en ejecución

```
82
83      generar_rutas_ficha1 = multiprocessing.Process(target=self.
            generar_rutas_s, args=(resultados1, tuple(self.tablero), dict(
            self.indices), 1, 16, ruta_colores+["b"], "f1", q1))
84      generar_rutas_ficha16 = multiprocessing.Process(target=self.
            generar_rutas_s, args=(resultados16, tuple(self.tablero), dict
            (self.indices), 4, 13, ruta_colores+["r"], "f16", q2))
85
86      generar_rutas_ficha1.start()
87      generar_rutas_ficha16.start()
88
89      self.movs_casillas_ficha1 = q1.get()
90      self.movs_casillas_ficha16 = q2.get()
91      generar_rutas_ficha1.join()
92      generar_rutas_ficha16.join()
93      self.ruta_pieza1 = self.archivo_pieza1.readline()
94      self.ruta_pieza1 = self.ruta_pieza1[0: len(self.ruta_pieza1)-1].
            split(",")
95      self.ruta_pieza16 = self.archivo_pieza16.readline()
96      self.ruta_pieza16 = self.ruta_pieza16 [0: len(self.ruta_pieza16)
            -1].split(",")
97      self.num_movs = n
98
99
100     def generar_rutas_s(self, res, tablero: tuple, indices: dict,
            estado_inicial:int, estado_final: int, ruta_colores:list,
        nombre_arch:str, q=None):
101
102
103     file_lock = threading.Lock()
```

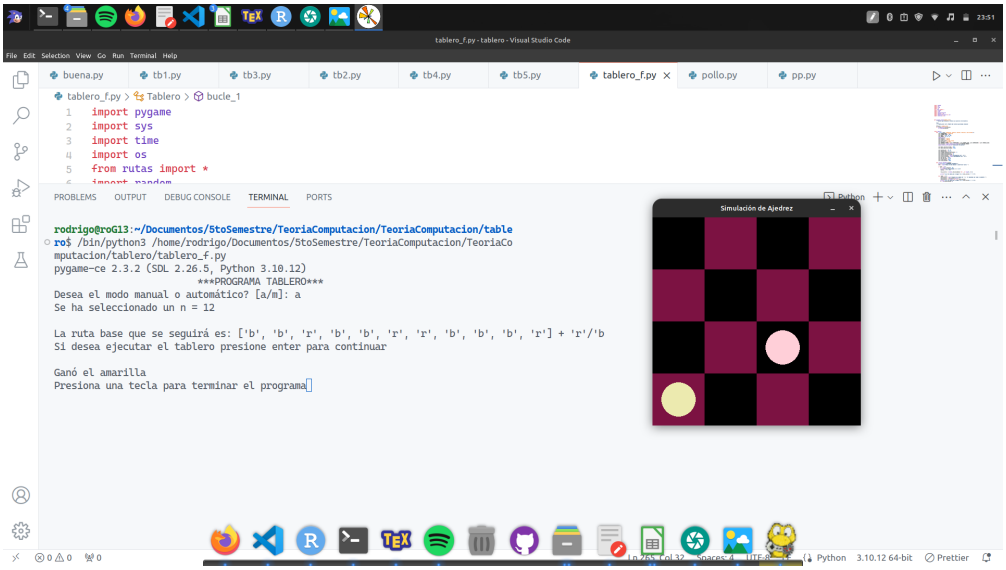

Figura 5: Ganó el amarillo

```
    variable_lock = threading.Lock()

    movs_casillas = {}

    for i in range(1, len(ruta_colores)+1):
    movs_casillas[i] = {}

    with open(f"{nombre_arch}_todas.txt", "w") as todas, open(f"{
       nombre_arch}_ganadoras.txt", "w") as ganadoras:
    resultados = threaded_generar_rutas(file_lock, variable_lock,
       ruta_colores, tablero, indices, estado_inicial, estado_final,
       todas, ganadoras, movs_casillas, res)
    for i in resultados.keys():
    todas.write("\n".join(resultados[i][0]))
    todas.write("\n")
    ganadoras.write("\n".join(resultados[i][1]))
    ganadoras.write("\n")

    if q:
    q.put(movs_casillas)
    return movs_casillas



    def dibujar_tablero(self):
    for fila in range(self.DIMENSIONES):
    for columna in range(self.DIMENSIONES):
    color = self.NEGRO if (fila + columna) % 2 == 0 else self.ROJO
```

```
129     pygame.draw.rect(self.ventana, color, (columna * self.
           TAMANO_CELDA, fila * self.TAMANO_CELDA, self.TAMANO_CELDA,
           self.TAMANO_CELDA))
130
131
132     def dibujar_piezas(self):
133     if self.cant_fichas >= 1:
134     pygame.draw.circle(self.ventana, (255, 207, 216), (self.
           pieza1_pos[1] * self.TAMANO_CELDA + self.TAMANO_CELDA // 2,
           self.pieza1_pos[0] * self.TAMANO_CELDA + self.TAMANO_CELDA //
           2), self.TAMANO_CELDA // 3)
135     if self.cant_fichas >= 2:
136     pygame.draw.circle(self.ventana, (236, 234, 175), (self.
           pieza16_pos[1] * self.TAMANO_CELDA + self.TAMANO_CELDA // 2,
           self.pieza16_pos[0] * self.TAMANO_CELDA + self.TAMANO_CELDA //
            2), self.TAMANO_CELDA // 3)
137
138
139     def revisar_existencia_movs(self, num_mov, casilla_actual, evitar
           , casilla):
140     if casilla == 1:
141     for elemento in self.movs_casillas_ficha1[int(num_mov)][int(
           casilla_actual)]:
142     if str(elemento) != str(evitar):
143     return True
144     else:
145     for elemento in self.movs_casillas_ficha16[int(num_mov)][int(
           casilla_actual)]:
146     if str(elemento) != str(evitar):
147     return True
148     return False
149
150
151     def dame_mov(self, num_mov, casilla_actual, evitar, casilla):
152
153     bandera_otro_mov = self.revisar_existencia_movs(num_mov,
           casilla_actual, evitar, casilla)
154
155     if casilla == 1:
156     if bandera_otro_mov:
157     nueva_ruta = self.read_and_filter(f"f{casilla}_ganadoras.txt",
           num_mov, evitar, self.indices[self.pieza1_pos])
158     return nueva_ruta
159     else:
160     if bandera_otro_mov:
161     nueva_ruta = self.read_and_filter(f"f{casilla}_ganadoras.txt",
           num_mov, int(evitar), self.indices[self.pieza16_pos])
162     return nueva_ruta
```

```
163
164
165
166     def read_and_filter(self, file_path, n, x, y, chunk_size=10000):
167     """
168     Lee un archivo en chunks y avanza hasta encontrar una fila donde
           en la columna "n" haya un elemento
169     distinto de x, pero en la columna "n-1" haya un elemento y.
           Retorna la fila encontrada
170     como un string separado por comas.
171
172     Args:
173     - file_path (str): Ruta al archivo.
174     - n (int):  ndice  de la columna n.
175     - x: Valor que no debe estar en la columna n.
176     - y: Valor que debe estar en la columna n-1.
177     - chunk_size (int): Tama o del chunk para leer el archivo.
178
179     Returns:
180     - String que representa la fila encontrada o None si no se
           encuentra.
181     """
182
183     desired_dtype = int
184
185     chunk_iter = pd.read_csv(file_path, sep=',', header=None,
           chunksize=chunk_size)
186
187     for chunk in chunk_iter:
188     chunk.astype(desired_dtype)
189     mask = (chunk[int(n)] != int(x)) & (chunk[int(n)-1] == int(y))
190     filtered_chunk = chunk[mask]
191
192     # Si encontramos al menos una fila que cumpla los criterios
193     if not filtered_chunk.empty:
194     # Convertir la primera fila que cumple con los criterios a string
195     found_row = filtered_chunk.iloc[0]
196     nueva = ','.join(map(str, found_row))
197     nueva = nueva.split(",")
198     return nueva
199
200     # Si se recorrieron todos los chunks y no se encontr  ninguna
           fila que cumpla los criterios
201     return None
202
203
204     def bucle_1(self):
205     mover_pieza1 = True  # Bandera para decidir qu  pieza se mueve
```

```
206     tiempo_proximo_movimiento = time.time() + 2  # Inicializamos el
           tiempo para el primer movimiento
207
208     while True:
209     for evento in pygame.event.get():
210     if evento.type == pygame.QUIT:
211     pygame.quit()
212     sys.exit()
213
214     tiempo_actual = time.time()
215
216     if tiempo_actual > tiempo_proximo_movimiento:
217     if mover_pieza1 and self.indice_movimiento_p1 <= self.num_movs:
218     aux = self.indices[int(self.ruta_pieza1[self.indice_movimiento_p1
           ])]
219     if aux != self.pieza16_pos:
220     self.pieza1_pos = aux
221     self.indice_movimiento_p1 += 1
222     else:
223     ruta = self.dame_mov(self.indice_movimiento_p1, int(self.indices[
           self.pieza1_pos]), int(self.indices[aux]), 1)
224     if ruta is not None:
225     self.ruta_pieza1 = ruta
226     self.pieza1_pos = self.indices[int(self.ruta_pieza1[self.
           indice_movimiento_p1])]
227     self.dibujar_piezas()
228     self.indice_movimiento_p1 += 1
229
230     if self.indice_movimiento_p1 > self.num_movs:
231     print("Gan  la rosada")
232     self.ventana.fill(self.ROJO)
233     self.dibujar_tablero()
234     self.dibujar_piezas()
235     pygame.display.flip()
236     input("Presione una tecla para continuar")
237     input()
238     break
239
240     elif not mover_pieza1 and self.indice_movimiento_p16 <= self.
           num_movs:
241     aux = self.indices[int(self.ruta_pieza16[self.
           indice_movimiento_p16])]
242     if aux != self.pieza1_pos:
243     self.pieza16_pos = aux
244     self.indice_movimiento_p16 += 1
245     else:
246     ruta = self.dame_mov(str(self.indice_movimiento_p16), int(self.
           indices[self.pieza16_pos]), int(self.indices[aux]), 16)
```

```
247
248     if ruta is not None:
249     self.ruta_pieza16 = ruta
250     aux = self.indices[int(self.ruta_pieza16[self.
           indice_movimiento_p16])]
251     self.pieza16_pos = self.indices[int(self.ruta_pieza16[self.
           indice_movimiento_p16])]
252     self.ventana.fill(self.ROJO)
253     self.dibujar_tablero()
254     self.dibujar_piezas()
255     pygame.display.flip()
256     self.indice_movimiento_p16 += 1
257
258     if self.indice_movimiento_p16 > self.num_movs:
259     print("Gan  el amarilla")
260     self.ventana.fill(self.ROJO)
261     self.dibujar_tablero()
262     self.dibujar_piezas()
263     pygame.display.flip()
264     input("Presiona una tecla para terminar el programa")
265     input()
266     break
267
268     # Cambiamos la bandera para el pr ximo movimiento y ajustamos el
           tiempo
269     mover_pieza1 = not mover_pieza1
270     tiempo_proximo_movimiento = tiempo_actual + 2
271
272
273
274     self.ventana.fill(self.ROJO)
275     self.dibujar_tablero()
276     self.dibujar_piezas()
277     pygame.display.flip()
278
279
280     def dibujar_red_neuronal(self, data):
281     G = nx.DiGraph()
282     pos = {}
283     layers = list(data.keys())
284
285     for i, layer in enumerate(layers):
286     states = list(data[layer].keys())
287     for state in states:
288     G.add_node(f"{layer}-{state}", label=str(state))
289     pos[f"{layer}-{state}"] = (i, -state)  # (x,y) coordenadas
290
291     next_layer = layer + 1
```

```
292     if next_layer in data:  # Verificar si existe la siguiente capa
293     for connection in data[layer][state]:
294     if connection in data[next_layer]:  # Verificar si el estado de
           conexi n existe en la siguiente capa
295     G.add_edge(f"{layer}-{state}", f"{next_layer}-{connection}")
296
297     labels = {node: G.nodes[node]['label'] for node in G.nodes()}
298
299     # Dibuja el grafo
300     plt.figure(figsize=(10, 6))
301     nx.draw(G, pos, labels=labels, node_size=2000, node_color="
           skyblue", font_size=10, width=2, edge_color="gray")
302     plt.title("Grafo tipo Red Neuronal")
303     plt.show()
304
305
306
307     if __name__ == "__main__":
308
309     eliminar_archs("f1_ganadoras.txt")
310     eliminar_archs("f1_todas.txt")
311     eliminar_archs("f16_ganadoras.txt")
312     eliminar_archs("f16_todas.txt")
313
314     tablero = (
315     (("b", False, 1), ("r", False, 2), ("b", False, 3), ("r", False,
           4)),
316     (("r", False, 5), ("b", False, 6), ("r", False, 7), ("b", False,
           8)),
317     (("b", False, 9), ("r", False, 10), ("b", False, 11), ("r", False
           , 12)),
318     (("r", False, 13), ("b", False, 14), ("r", False, 15), ("b",
           False, 16))
319     )
320
321     indices = {1: (0,0), 2: (0,1), 3: (0,2), 4: (0,3),
322         5: (1,0), 6: (1,1), 7: (1,2), 8: (1,3),
323         9: (2,0), 10: (2,1), 11: (2,2), 12: (2,3),
324         13: (3,0), 14: (3,1), 15: (3,2), 16: (3,3),
325         (0,0):1, (0,1):2, (0,2):3, (0,3):4,
326         (1,0):5, (1,1):6, (1,2):7, (1,3):8,
327         (2,0):9,(2,1):10,(2,2):11,(2,3):12,
328         (3,0):13, (3,1):14,(3,2):15,(3,3):16}
329
330
331     juego = Tablero(4, tablero, indices, 0, cant_fichas=2)
332     juego.calcular_ganadoras()
333     juego.dibujar_red_neuronal(juego.movs_casillas_ficha1)
```

```
334     juego.dibujar_red_neuronal(juego.movs_casillas_ficha16)
335     print("Si desea ejecutar el tablero presione enter para continuar
            ")
336     juego.bucle_1()
```

III.2. Generador de rutas

```
1     import threading
2
3     def crear_matriz_ventanita(tablero, fila, col):
4     ventanita = []
5     for i in range(fila - 1, fila + 2):
6     fila_aux = []
7     for j in range(col - 1, col + 2):
8     if 0 <= i < len(tablero) and 0 <= j < len(tablero[0]):
9     fila_aux.append(tablero[i][j])
10    else:
11    fila_aux.append([None, None, None])
12    ventanita.append(fila_aux)
13    ventanita[1][1] = [None, None, None]
14    return ventanita
15
16    def seleccionar_hijos(estado, tablero, indices, movimientos,
            contador):
17    fila, col = indices[estado]
18    ventanita = crear_matriz_ventanita(tablero, fila, col)
19    hijos = []
20    for fila in ventanita:
21    for casilla in fila:
22    if casilla[0] == movimientos[contador]:
23    hijos.append(casilla[2])
24    return hijos
25
26    def generar_rutas(file_lock, variable_lock,movs_casillas,
            movimientos, tablero, indices, estado, ruta, estado_final,
            todas, ganadoras, lista_estados, lista_ganadoras):
27    cte = 1000000
28
29    if len(ruta) <= len(movimientos):
30    hijos = seleccionar_hijos(estado, tablero, indices, movimientos,
            len(ruta)-1)
31
32    for hijo in hijos:
33    with variable_lock:
34    crear_bifu(movs_casillas, len(ruta), ruta[-1])
35    movs_casillas[len(ruta)][ruta[-1]].add(int(hijo))
```

```
36      generar_rutas(file_lock, variable_lock,movs_casillas, movimientos
            , tablero, indices, hijo, ruta + [hijo], estado_final, todas,
            ganadoras, lista_estados, lista_ganadoras)
37
38      else:
39      if len(lista_estados) < cte:
40      lista_estados.append(f"{','.join(map(str, ruta))}")
41      elif len(lista_estados) >= cte:
42      lista_estados.append(f"{','.join(map(str, ruta))}")
43      with file_lock:
44      todas.write("\n".join(lista_estados))
45      todas.write("\n")
46      lista_estados.clear()
47
48      if str(ruta[-1]) == str(estado_final) and len(lista_ganadoras) <
            cte:
49      lista_ganadoras.append(f"{','.join(map(str, ruta))}")
50      elif ruta[-1] == estado_final and len(lista_ganadoras) >= cte:
51      lista_ganadoras.append(f"{','.join(map(str, ruta))}")
52      with file_lock:
53      ganadoras.write("\n".join(lista_ganadoras))
54      ganadoras.write("\n")
55      lista_ganadoras.clear()
56
57      def crear_bifu(dict_movs, num_mov, llave):
58      if llave not in dict_movs[num_mov]:
59      dict_movs[num_mov][llave] = set()
60
61      def threaded_generar_rutas(file_lock, variable_lock, movimientos,
            tablero, indices, estado_inicial, estado_final, todas,
            ganadoras, movs_casillas, resultados={}):
62
63      ruta = [estado_inicial]
64      hilos = {}
65
66      hijos_iniciales = seleccionar_hijos(estado_inicial, tablero,
            indices, movimientos, 0)
67
68      if hijos_iniciales == 1:
69      ruta = [estado_inicial, hijos_iniciales[0]]
70      crear_bifu(movs_casillas, len(ruta), ruta[-1])
71      movs_casillas[len(ruta)][ruta[-1]].add(hijos_iniciales[0])
72      hijos_iniciales = seleccionar_hijos(estado_inicial, tablero,
            indices, movimientos, 1)
73
74      for hijo in hijos_iniciales:
75      resultados[hijo] = [list(), list()]
76      with variable_lock:
```

```
77      crear_bifu(movs_casillas, len(ruta), ruta[-1])
78      movs_casillas[len(ruta)][ruta[-1]].add(int(hijo))
79      hilo = threading.Thread(target=generar_rutas, args=(file_lock,
           variable_lock,movs_casillas, movimientos, tablero, indices,
           hijo, ruta+[hijo], estado_final, todas, ganadoras, resultados[
           hijo][0], resultados[hijo][1]))
80      hilos[hijo] = hilo
81      hilo.start()
82
83      for _, hilo in hilos.items():
84      hilo.join()
85
86      return resultados
87
88      if __name__ == "__main__":
89      tablero = (
90      (("b", False, 1), ("r", False, 2), ("b", False, 3), ("r", False,
           4)),
91      (("r", False, 5), ("b", False, 6), ("r", False, 7), ("b", False,
           8)),
92      (("b", False, 9), ("r", False, 10), ("b", False, 11), ("r", False
           , 12)),
93      (("r", False, 13), ("b", False, 14), ("r", False, 15), ("b",
           False, 16))
94      )
95
96      indices = {1: (0,0), 2: (0,1), 3: (0,2), 4: (0,3),
97          5: (1,0), 6: (1,1), 7: (1,2), 8: (1,3),
98          9: (2,0), 10: (2,1), 11: (2,2), 12: (2,3),
99          13: (3,0), 14: (3,1), 15: (3,2), 16: (3,3)}
100
101     with open("todas.txt", "w") as todas, open("ganadoras.txt", "w")
           as ganadoras:
102     file_lock = threading.Lock()
103     variable_lock = threading.Lock()
104     ruta_colores = ['r', 'b', 'b', 'r', 'b', 'b', 'r', 'b', 'b', "r"]
105
106     movs_casillas = {}
107
108     for i in range(1, len(ruta_colores)+1):
109     movs_casillas[i] = {}
110
111     resultados = threaded_generar_rutas(file_lock, variable_lock,
           ruta_colores, tablero, indices, 1, 9, todas, ganadoras,
           movs_casillas)
112
113     for i in resultados.keys():
114     todas.write("\n".join(resultados[i][0]))
```

```
115     todas.write("\n")
116     ganadoras.write("\n".join(resultados[i][1]))
117     ganadoras.write("\n")
118 
119     print(movs_casillas)
120     print(movs_casillas)
```