

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TÍTULO DEL REPORTE

P R O G R A M A
T A B L E R O

QUE PARA OBTENER UN 10 EN EL REPORTE:

PRESENTA:

CONNOR URBANO MENDOZA

DOCENTE:

JUÁREZ MARTÍNEZ GENARO

Estados Unidos Mexicanos
Ciudad de México
2023

INSTITUTO POLITÉCNICO NACIONAL



Índice general

Introducción	III
1 Desarrollo	1
1.1 Análisis del problema principal	1
1.2 Límites del problema	3
1.3 Estrategia para atacar el problema	4
1.4 Implementación	5
2 Análisis de Resultados	54
2.1 Capturas del programa en ejecución	54
3 Conclusión	70
3.1 Problemas iniciales	71
3.1.1 Soluciones	72
3.2 Complejidades	73
4 Bibliografías	74
5 Anexos	75
5.1 LATEX de exte proyecto	75
5.2 Main.py	75
5.3 Main1.py	78
5.4 Main2.py	87
5.5 Main3.py	96
5.6 Main4.py	116
5.7 Main5.py	136
5.8 graficador.py	157
5.9 graficador _{grande} .py	163

Índice de figuras

1.1	Main.py.	9
1.2	Main1.py.	19
1.3	Código y funciones modificadas del main1.py para el main2.py. . .	35
1.4	Lógica de detección de colisiones, creamos una nueva ruta.	39
1.5	Graficador.py.	46
1.6	Graficador _{grande} .py.	53
2.1	Inicio del programa en terminal.	55
2.2	Vista de archivos de salida de recorridos.	56
2.3	Visualización de transiciones gráficas.	57
2.4	Transiciones graficas.	58
2.5	Fin de transición y final en terminal.	59
2.6	Grafo de recorrido.	60
2.7	Vista de terminal de segundo caso.	61
2.8	Secuencia de transiciones, donde en caso 2 hay colisión.	62
2.9	Vista de terminal de nueva ruta configurada y salida en archivos. . .	63
2.10	Vista de gráfica del ajedrez.	64
2.11	Árbol de recorridos para pieza blanca.	65
2.12	Árbol de recorridos para pieza negra.	66
2.13	Vista de terminal para recorrido de tamaño 82.	67
2.14	Vista de archivo de recorridos para la pieza blanca.	68
2.15	Árbol de recorridos para pieza en cuestión.	69

Introducción

En esta práctica de Teoría de la Computación vamos a trabajar en el diseño e implementación de un programa de ajedrez que permita realizar movimientos ortogonales y diagonales en un tablero de 4x4, considerando la opción de tener una o dos piezas en el tablero. Además de cumplir con las reglas y movimientos establecidos en las láminas del curso de Stanford, el programa debe contar con las siguientes características:

Se siguieron las siguientes instrucciones especificadas por el docente, que son las siguientes:

1. Modo automático y modo manual: El programa debe poder ejecutarse tanto en modo automático como en modo manual. En el modo manual, el usuario tiene la opción de introducir la cadena de movimientos o generarla aleatoriamente.
2. Elección de la cantidad de piezas: El programa debe ser capaz de funcionar con una o dos piezas en el tablero. Si se selecciona la opción de dos piezas, la segunda pieza debe iniciar en el estado 4 y su estado final debe ser el estado 13.
3. Elección aleatoria del jugador inicial: Al iniciar el juego, el programa debe decidir de manera aleatoria quién comienza a jugar.
4. Generación de archivos de movimientos: Una vez que se define la cadena de movimientos para una o dos piezas, el programa debe generar archivos que

contengan todos los movimientos posibles por pieza, así como otro archivo con los movimientos ganadores por pieza. Estos archivos serán utilizados para reconfigurar las rutas en el juego.

5. Reconfiguración de rutas y espera en caso de no poder avanzar: Si se reconfigura una ruta y aun así no es posible avanzar, se debe esperar una iteración antes de intentar otra reconfiguración.
6. Graficación del tablero y visualización de movimientos: El programa debe permitir graficar el tablero de ajedrez y mostrar los movimientos realizados por una o dos piezas. Esto facilita la visualización interactiva de los movimientos en el tablero.
7. Restricción de movimientos automáticos: En el modo automático, las cadenas generadas no deben tener más de 10 movimientos para poder ser utilizadas en la animación.
8. Límite máximo de movimientos: El programa debe imponer un límite de 100 símbolos como máximo para la cantidad de movimientos permitidos en una cadena.
9. Generación de árbol de movimientos: Se debe generar un archivo de salida en formato de imagen que represente el árbol de movimientos evaluados en cada corrida del programa.
10. Inclusión del código fuente en el reporte: El reporte final debe incluir el código fuente completo del programa desarrollado.

Además, se deben cumplir las características y requerimientos mencionados anteriormente, incluyendo la generación de archivos de movimientos, la graficación del tablero, la restricción de movimientos automáticos y la generación de árboles de movimientos. El reporte final debe incluir el código fuente completo del programa.

Capítulo 1

Desarrollo

1.1. Análisis del problema principal

Los problemas principales en esta práctica se presentan a continuación, donde se les tuvo que dar solución para un correcto funcionamiento del programa:

1. Representación del tablero: El primer desafío es establecer una representación adecuada del tablero de ajedrez de 4x4. Esto implica decidir qué estructura de datos utilizar para almacenar la información sobre las posiciones y las piezas en el tablero.
2. Generación de movimientos válidos: Para implementar los movimientos ortogonales y diagonales, es necesario desarrollar un algoritmo que genere los movimientos válidos para cada pieza en función de su posición actual en el tablero. Esto incluye verificar las restricciones del tablero, como los límites de movimiento y la presencia de otras piezas en el camino.
3. Control del modo de juego: El programa debe ser capaz de manejar tanto el modo automático como el modo manual. En el modo manual, se deben validar y procesar las cadenas de movimientos ingresadas por el usuario. En el modo automático, se deben generar cadenas de movimientos aleatorias dentro de los límites establecidos.

4. Gestión de múltiples piezas: El programa debe poder manejar tanto una como dos piezas en el tablero. Esto implica controlar la posición y los movimientos de cada pieza de manera independiente, así como determinar el jugador que tiene el turno en cada momento.
5. Determinación del jugador inicial: El programa debe tomar una decisión aleatoria para determinar qué jugador inicia el juego. Esto asegura la equidad y la variedad en las partidas.
6. Generación y reconfiguración de rutas: Una vez que se define la cadena de movimientos para una o dos piezas, se deben generar archivos que contengan todos los movimientos posibles por pieza y los movimientos ganadores por pieza. Además, se deben implementar mecanismos para reconfigurar las rutas en caso de no poder avanzar, esperando una iteración antes de intentar otra reconfiguración.
7. Límite de movimientos y restricciones automáticas: El programa debe controlar la longitud máxima de las cadenas de movimientos generadas en el modo automático para garantizar que sean adecuadas para la animación. También se debe establecer un límite máximo de 100 símbolos para la cantidad total de movimientos permitidos.
8. Graficación del tablero y visualización de movimientos: El programa debe incluir una funcionalidad para graficar el tablero de ajedrez y mostrar los movimientos realizados por las piezas. Esto permite una interacción más intuitiva y visual con el juego.
9. Generación de árboles de movimientos: Para evaluar y analizar las cadenas de movimientos, se debe implementar un mecanismo para generar árboles que representen las diferentes opciones y decisiones tomadas en cada corrida del programa. Estos árboles pueden ser útiles para el análisis posterior del rendimiento del programa.

Se requirió de un enfoque cuidadoso en el diseño de algoritmos y estructuras de datos, así como una gestión adecuada de las restricciones y límites establecidos.

1.2. Límites del problema

Los límites y restricciones del problema de esta práctica del tablero de ajedrez son los siguientes:

1. Tamaño del tablero: El tablero de ajedrez es de tamaño fijo y se limita a 4x4. Esto significa que todas las operaciones y movimientos se realizarán dentro de este tablero específico.
2. Complejidad del algoritmo: El cálculo de los movimientos válidos y la generación de rutas puede volverse más complejo a medida que aumenta la longitud de las cadenas de movimientos. Se debe tener en cuenta la eficiencia del algoritmo utilizado para garantizar tiempos de ejecución aceptables.
3. Número máximo de movimientos: Se establece un límite máximo de 100 símbolos para la cantidad total de movimientos permitidos. Esto asegura que las cadenas de movimientos no sean excesivamente largas y se puedan manejar de manera eficiente.
4. Longitud máxima de cadenas generadas en modo automático: En el modo automático, las cadenas de movimientos generadas no deben ser mayores a 10 movimientos. Esto se establece para limitar la duración de la animación y garantizar que sea adecuada para su visualización.
5. Restricciones de configuración de rutas: Si se intenta reconfigurar una ruta y aun así no se puede avanzar, se debe esperar una iteración antes de intentar otra reconfiguración. Esta restricción asegura un enfoque más controlado en la búsqueda de movimientos válidos y evita posibles bucles infinitos o comportamientos indeseados.

1.3. Estrategia para atacar el problema

Las estrategias utilizadas para resolver el problema de la práctica fueron:

Algunas estrategias específicas que se pueden utilizar para resolver el problema de la práctica son:

1. Modularidad: El código se divide en varios archivos para facilitar su comprensión y mantenimiento. Cada archivo se encarga de una parte específica del problema, como la generación de los recorridos de las piezas, la visualización gráfica o el control del menú.
2. Menús interactivos: Se implementaron funciones de menú para solicitar al usuario la cantidad de piezas a invocar y el tipo de recorrido deseado. Estos menús ayudan a personalizar la experiencia del usuario y proporcionan opciones claras para tomar decisiones.
3. Uso de subprocess.run: El módulo subprocess se utiliza para invocar los archivos Python correspondientes a la generación de los recorridos de las piezas. Esto permite ejecutar el código en un proceso separado y obtener su código de salida.
4. Verificación de recorridos válidos: Después de la ejecución del archivo Python para generar los recorridos de las piezas, se verifica el código de salida para determinar si se generaron recorridos válidos. Esto asegura que solo se muestren los recorridos correctos al usuario.
5. Graficado de los resultados: Dependiendo del código de salida del archivo Python, se invoca el graficador python correspondiente para mostrar los resultados en una ventana gráfica. Esto proporciona una representación visual de los recorridos de las piezas y facilita su comprensión.

Estas estrategias permiten una interacción intuitiva con el programa, generando recorridos de piezas de ajedrez y visualizándolos de manera clara y gráfica.

1.4. Implementación

Se utilizó Python como lenguaje de programación para facilitar la parte gráfica del ajedrez. Se implementaron estrategias de modularidad utilizando subprocesos para facilitar la visualización y el estudio del programa.

La estrategia general del programa se basa en la selección de la cantidad de piezas a invocar y el tipo de recorrido. Dependiendo de estas selecciones, se ejecuta un archivo Python específico utilizando el módulo `subprocess`. Los diferentes archivos Python (`main1.py`, `main2.py`, `main3.py`, `main4.py`, `main5.py`) contienen la lógica y los algoritmos para generar recorridos de las piezas del ajedrez.

El código del archivo principal (`main.py`) se encarga de mostrar los menús de selección, invocar el archivo Python correspondiente y llamar al final al graficador correspondiente para mostrar los resultados en PDF del árbol de recorridos.

Aquí hay un resumen de las implementaciones utilizadas en código:

1. Para el `main.py` tenemos.
 - a) Se implementaron funciones de menú (`menu_piezas` y `menu_recorrido`) para solicitar al usuario la cantidad de piezas a invocar y el tipo de recorrido.
 - b) Dependiendo de las selecciones del usuario, se invoca el archivo Python correspondiente utilizando `subprocess.run`.
 - c) Después de la ejecución del archivo Python, se verifica el código de salida para determinar si se generaron recorridos válidos.
 - d) Dependiendo del código de salida, se invoca el script de graficado correspondiente (`graficador.py` o `graficador_grande.py`) para mostrar los resultados en una ventana gráfica.

Donde el código es:

```
1 #Teoria de la computacion
2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4 import os
5 import subprocess
6
7 def menu_piezas():
8     cantidad = int(input("Ingrese la cantidad de
9     piezas a invocar (m nimo 1, m ximo 2)\n--> "))
10
11     if cantidad < 1 or cantidad > 2:
12         print("\nCantidad inv lida. Por favor,
13         ingrese 1 o 2.")
14         return menu_piezas() # Llamada recursiva si
15         la cantidad es inv lida
16     else:
17         return cantidad
18
19 def menu_recorrido(cantidad_piezas):
20     #config==1 es para recorrido aleatorio
21     #config==2 es para recorrido manual
22     if cantidad_piezas==1:
23         #1 pieza
24         opcion = int(input("Seleccione una opci n:\n1
25         . Generar recorrido aleatoriamente.\n2.
26         Ingresar recorrido manualmente.\nOpci n: "))
27
28         if opcion == 1:
29             return 1 # Opci n de generar recorrido
30             aleatoriamente
31         elif opcion == 2:
32             return 2 # Opci n de ingresar recorrido
33             manualmente
34         else:
35             print("Opci n inv lida. Por favor,
36             seleccione 1 o 2.")
37             return menu_recorrido(cantidad_piezas) #
38             Llamada recursiva si la opci n es
```

```

    inv lida
28 else:
29     #2 piezas
30     opcion = int(input("Seleccione una opci n:\n1
        . Aleatorio vs Aleatorio.\n2. Manual vs
        Aleatorio.\n3. Manual vs Manual.\nOpci n:
        "))
31     if opcion == 1:
32         return 3 # Opci n de generar recorrido
            aleatoriamente vs aleatoriamente
33     elif opcion == 2:
34         return 4 # Opci n de ingresar recorrido
            manualmente vs aleatoriamente
35     elif opcion == 3:
36         return 5 # Opci n de ingresar recorrido
            manualmente vs manualmente
37     else:
38         print("Opci n inv lida. Por favor,
            seleccione 1, 2 o 3.")
39         return menu_recorrido(cantidad_piezas) #
            Llamada recursiva si la opci n es
            inv lida
40
41 #Main del ciclo del juego
42 # Obtener directorio actual
43 directorio_actual = os.path.dirname(os.path.abspath(
    __file__))
44 cantidad_piezas = menu_piezas()
45 print("\nCantidad de piezas seleccionadas:",
    cantidad_piezas)
46 config = menu_recorrido(cantidad_piezas)
47 x=0
48 #Posibles configuraciones: 1,2,3,4 o 5.
49 if config == 1:#A
50     archivo_py = os.path.join(directorio_actual, "
        main1.py")
51     resultado = subprocess.run(["python", archivo_py])
52     os.system('cls' if os.name == 'nt' else 'clear')
53     codigo_salida = resultado.returncode
54     x=1
```

```
55     #print(codigo_salida)
56 elif config == 2:#Manual
57     archivo_py = os.path.join(directorio_actual, "
58         main2.py")
59     resultado = subprocess.run(["python", archivo_py])
60     os.system('cls' if os.name == 'nt' else 'clear')
61     codigo_salida = resultado.returncode
62     x=1
63     #print(codigo_salida)
64 elif config == 3:#AvsA
65     archivo_py = os.path.join(directorio_actual, "
66         main3.py")
67     resultado =subprocess.run(["python", archivo_py])
68     os.system('cls' if os.name == 'nt' else 'clear')
69     codigo_salida = resultado.returncode
70     x=2
71     #print(codigo_salida)
72 elif config == 4:#ManualvsA
73     archivo_py = os.path.join(directorio_actual, "
74         main4.py")
75     resultado =subprocess.run(["python", archivo_py])
76     os.system('cls' if os.name == 'nt' else 'clear')
77     codigo_salida = resultado.returncode
78     x=2
79     #print(codigo_salida)
80 elif config == 5:#ManualvsManual
81     archivo_py = os.path.join(directorio_actual, "
82         main5.py")
83     resultado =subprocess.run(["python", archivo_py])
84     os.system('cls' if os.name == 'nt' else 'clear')
85     codigo_salida = resultado.returncode
86     x=2
87     #print(codigo_salida)
88 if codigo_salida == 3:
89     archivo_py = os.path.join(directorio_actual, "
90         graficador_grande.py")
91     subprocess.run(["python", archivo_py, (str(x))])
92 else:
93     archivo_py = os.path.join(directorio_actual, "
```

```
    graficador.py")
90     subprocess.run(["python", archivo_py, (str(x))])
91 print("termino")
```

Figura 1.1: Main.py.

El código main.py que proporcionaste parece ser un programa de Python que presenta un menú interactivo y realiza diferentes acciones según las opciones seleccionadas por el usuario. A continuación, se muestra una descripción de alto nivel de lo que hace el código:

- a) El código comienza definiendo dos funciones: `menu_piezas()` y `menu_recorrido()`, que solicitan información al usuario y devuelven valores según las opciones seleccionadas.
- b) El programa principal comienza obteniendo el directorio actual y luego solicita al usuario la cantidad de piezas a invocar mediante la función `menu_piezas()`.
- c) Después de recibir la cantidad de piezas, se llama a la función `menu_recorrido()` para solicitar al usuario el tipo de recorrido que desea realizar, dependiendo de la cantidad de piezas seleccionadas.
- d) Basándose en la configuración seleccionada, se asigna un valor a la variable `config`.
- e) Luego, el código utiliza la variable `config` para determinar qué archivo .py se debe ejecutar utilizando el módulo `subprocess.run(["python", archivo_py])`. Después de ejecutar el archivo, se
- f) borra la pantalla y se captura el código de salida en la variable `codigo_salida`.

- g) Finalmente, se verifica el valor de `codigo_salida` y se ejecuta un archivo .py adicional en función de ese valor.
- h) `main1.py`: Este archivo contiene el código para generar un recorrido aleatorio para una sola pieza. El recorrido de la pieza se elige de forma automática sin intervención del usuario.
- i) `main2.py`: En este archivo, el usuario puede ingresar manualmente el recorrido de una sola pieza. El programa espera la entrada del usuario para establecer el recorrido deseado.
- j) `main3.py`: Aquí se maneja la configuración para dos piezas con recorridos aleatorios. El código se vuelve más complejo, ya que se deben detectar colisiones entre las piezas generadas aleatoriamente. Si se detecta una colisión, se recalculan las rutas de las piezas para evitar la colisión.
- k) `main4.py`: En este caso, se tienen dos piezas, una generada aleatoriamente y otra con un recorrido establecido por el usuario. El programa permite ingresar el recorrido manualmente para una de las piezas, mientras que la otra se genera aleatoriamente.
- l) `main5.py`: Este archivo maneja la configuración para dos piezas con recorridos completamente establecidos por el usuario. Ambas piezas requieren que el usuario ingrese manualmente los recorridos deseados.

2. Observar figura 1.2. Para el `main1.py`:

```
1 #Teoria de la computacion
2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 import pygame
7 import random
```

```

8 import os
9
10 pygame.init() #Acceso al paquete pygame
11 #Ancho
12 WIDTH = 1000
13 #Altura
14 HEIGHT = 700
15 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #
    Tamano de ventana a imprimir
16 pygame.display.set_caption('Problema del Ajedrez')
17 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
18 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo
    de fuente 2 del juego
19 timer = pygame.time.Clock()#velocidad de actualizacion
    de nuestro juego a 60 fps
20 fps=60
21
22 #NFA de estados
23 tablaEstados = {
24     '1' : {'R': {'2','5'}, 'N': '6'},
25     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
26     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
27     '4' : {'R': '7', 'N': {'3','8'}},
28     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
29     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9',
        '11'}},
30     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8',
        '11'}},
31     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
32     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
33     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9',
        '11','14'}},
34     '11' : {'R': {'7','10','12','15'}, 'N': {'6','8',
        '14','16'}},
35     '12' : {'R': {'7','15'}, 'N': {'8','11','16'}},
36     '13' : {'R': '10', 'N': {'9','14'}},
37     '14' : {'R': {'13','10','15'}, 'N': {'9','11'}},
38     '15' : {'R': {'10','12'}, 'N': {'11','14','16'}},
39     '16' : {'R': {'12','15'}, 'N': '11'}#Estado 16 es

```



```
        el estado Final.
40 }
41
42 #Variables e imagenes del juego
43 pieza_blanca = ['king']
44 posicion_blanca = [(235,85)]
45
46 #Variables de turnos cambiantes
47 turn_step = 0
48 selection= 100
49 #Cargar imagenes en juego
50 rey_blanco = pygame.image.load('C:\\Users\\soyco\\
    OneDrive\\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess
    \\img\\white_king.png')
51 rey_blanco = pygame.transform.scale(rey_blanco, (80,80)
    )
52
53 imagen_blanca = [rey_blanco]
54
55 lista_piezas = ['king']
56
57 #ver variables/contador flash
58 boton_presionado = False
59
60 def dibujar_boton():
61     boton_width = 150
62     boton_height = 45
63     boton_x = (WIDTH - boton_width) // 2
64     boton_y = (HEIGHT - boton_height - 20)+17
65
66     # Dibujar el boton como un rect ngulo en la
        pantalla
67     boton_rect=pygame.Rect(boton_x, boton_y,
        boton_width, boton_height)
68     pygame.draw.rect(screen, (0, 255, 0), (boton_x,
        boton_y, boton_width, boton_height))
69     texto = font.render("Siguiente", True, (0, 0, 0))
70     texto_rect = texto.get_rect(center=(boton_x +
        boton_width // 2, boton_y + boton_height // 2))
71     screen.blit(texto, texto_rect)
```

```
72     return boton_rect
73
74 #Funcion para dibujar tablero
75 def dibujar_tablero():
76     cuadro_size = 150 # Tama o de cada cuadro del
77     tablero
78     tablero_width = 4 * cuadro_size # Ancho total del
79     tablero
80     tablero_height = 4 * cuadro_size # Altura total
81     del tablero
82     tablero_x = (WIDTH - tablero_width) // 2 #
83     Posici n X para centrar el tablero
84     tablero_y = (HEIGHT - tablero_height) // 2 #
85     Posici n Y para centrar el tablero
86
87     numero_color = 'white' # Color del n mero de
88     casilla
89
90     font = pygame.font.Font(None, 24) # Fuente y
91     tama o del n mero de casilla
92
93     for i in range(16): # Iterar 16 veces para un
94     tablero de 4x4
95         columna = i % 4
96         fila = i // 4
97         x = tablero_x + columna * cuadro_size
98         y = tablero_y + fila * cuadro_size
99         if fila % 2 == 0:
100             color = 'black' if columna % 2 == 0 else '
101             red'
102         else:
103             color = 'red' if columna % 2 == 0 else '
104             black'
105         pygame.draw.rect(screen, color, [x, y,
106         cuadro_size, cuadro_size])
107         pygame.draw.rect(screen, 'white', [x, y,
108         cuadro_size, cuadro_size], 2) # Agregar
109         borde de color blanco
110         numero_texto = font.render(str(i + 1), True,
111         numero_color) # Crear superficie de texto
```

```

    con el n mero
98     numero_rect = numero_texto.get_rect(center=((x
    + cuadro_size // 2)-60, y + 20)) #
    Posici n del n mero en la parte superior
    del recuadro
99     screen.blit(numero_texto, numero_rect) #
    Pegar el n mero en la pantalla
100
101 #Funcion para dibujar piezas
102 def dibujar_piezas():
103     index=lista_piezas.index('king')
104     if pieza_blanca[0]=='king':
105
106         screen.blit(imagen_blanca[index], (
            posicion_blanca[0][0],posicion_blanca
            [0][1]))
107
108 def recorrer_estados(tabla_estados, cadena):
109     # Funci n para obtener todos los recorridos
    posibles
110     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual):
111         if not simbolos_restantes:
112             recorridos.append(recorrido_actual)
113             return
114
115         simbolo = simbolos_restantes[0]
116         if estado_actual in tabla_estados and simbolo
            in tabla_estados[estado_actual]:
117             transiciones = tabla_estados[estado_actual
                ][simbolo]
118
119             for estado_siguiente in transiciones:
120                 obtener_recorridos(estado_siguiente,
                    simbolos_restantes[1:],
                    recorrido_actual + [
                        estado_siguiente])
121
122     # Funci n para obtener los recorridos v lidos
    hasta el estado final
```

```
123     def obtener_recorridos_finales(estado_actual,
124                                   simbolos_restantes, recorrido_actual):
125         if estado_actual == '16':
126             recorridos_finales.append(recorrido_actual)
127             return
128         if not simbolos_restantes:
129             return
130
131         simbolo = simbolos_restantes[0]
132         if estado_actual in tabla_estados and simbolo
133            in tabla_estados[estado_actual]:
134             transiciones = tabla_estados[estado_actual]
135                            [simbolo]
136
137             for estado_siguiente in transiciones:
138                 obtener_recorridos_finales(
139                     estado_siguiente,
140                     simbolos_restantes[1:],
141                     recorrido_actual + [
142                         estado_siguiente])
143
144     # Obtener recorridos posibles
145     recorridos = []
146     obtener_recorridos('1', cadena, ['1'])
147
148     # Obtener recorridos hasta el estado final
149     recorridos_finales = []
150     obtener_recorridos_finales('1', cadena, ['1'])
151
152     # Guardar los recorridos en archivos de texto
153     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
154             ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
155             recorridos.txt', 'w') as archivo_recorridos:
156         archivo_recorridos.write('Recorridos posibles
157                                 :\n')
158         for recorrido in recorridos:
159             archivo_recorridos.write(','.join(
160                 recorrido) + '\n')
```

```
151
152     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales.txt', 'w') as
        archivo_recorridos_finales:
153         for recorrido in recorridos_finales:
154             archivo_recorridos_finales.write(','.join(
                recorrido) + '\\n')
155
156
157
158 def seleccionar_recorrido():
159     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\recorridos_finales.txt"
160
161     with open(ruta_archivo, "r") as archivo:
162         if os.path.getsize(ruta_archivo) == 0:
163             print("No existen soluciones que lleguen
                al estado 16 con la condicion actual.
                Se recalculara una ruta.\\n")
164             nuevavuta=cadena()
165             print("La nueva ruta es: "+nuevaruta)
166             recorrer_estados(tablaEstados, nuevavuta)
167             print('Se almacenaron las nuevas salidas
                de los recorridos posibles y los
                recorridos exitosos en la carpeta
                output.\\n')
168
169             return seleccionar_recorrido()
170
171     lineas = archivo.readlines()
172
173     # Seleccionar una l nea aleatoria
174     recorrido_seleccionado = random.choice(lineas)
175
176     # Eliminar los espacios en blanco y saltos de
        l nea
177     recorrido_seleccionado =
        recorrido_seleccionado.strip()
```

```
178         return recorrido_seleccionado
179
180
181 def cadenaRandom(numero): #Genera un string de forma
    random
182     auxiliar = '' #Variable auxiliar
183     for i in range(numero):
184         x = random.randint(1, 2) #Funci n para
            generar un resultado random de una lista
185         if x % 2 == 0:
186             auxiliar = auxiliar + "R"
187         else:
188             auxiliar = auxiliar + "N"
189     return auxiliar
190
191 def cadena():
192     numero = random.randint(4,10)
193     print('\nTamaño de cadena escogido aleatoriamente
        [' +str(numero)+']\nTambien se generaran las
            transiciones R y N aleatoriamente.\n')
194     cad = cadenaRandom(numero) #Se genera de forma
        aleatoria del 1-10
195     print('\nLa cadena o ruta a seguir escogida
        aleatoriamente sera: ' +cad+'\n')
196     return cad
197
198 def calcular_coordenadas(estado):
199     cuadro_size = 150 # Tama o de cada cuadro del
        tablero
200     tablero_width = 4 * cuadro_size # Ancho total del
        tablero
201     tablero_height = 4 * cuadro_size # Altura total
        del tablero
202     tablero_x = (WIDTH - tablero_width) // 2 #
        Posici n X para centrar el tablero
203     tablero_y = (HEIGHT - tablero_height) // 2 #
        Posici n Y para centrar el tablero
204     fila = (estado - 1) // 4 # Calcular la fila del
        estado
205     column = (estado - 1) % 4 # Calcular la columna
```

```

    del estado
206     x = tablero_x + column * cuadro_size # Calcular
        la coordenada X del estado
207     y = tablero_y + fila * cuadro_size # Calcular la
        coordenada Y del estado
208     return ((x+33), y+33) # Devolver las coordenadas
        como una tupla
209
210
211 #Main del ciclo del juego 1
212 ruta=cadena() #Solicitamos la cadena generada
        aleatoriamente.
213 recorrer_estados(tablaEstados, ruta)
214 print('\nEn este punto el programa almaceno en la
        salidas los recorridos posibles y los recorridos
        exitosos en la carpeta output.\n')
215 recorrido = seleccionar_recorrido() #Este fue el
        recorrido que se escogio de manera aleatoria
216 with open('C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca
        .txt', 'w') as archivo:
217     archivo.write(ruta)
218 print("\nRecorrido seleccionado:", recorrido)
219 print("\nVea la parte grafica.")
220 lista_estados = [int(num) for num in recorrido.split("
        ,")]
221
222 coordenadas=(235,85)
223 nueva_coordenada=(0,0)
224 run=True
225 contador=1
226 while run:
227     timer.tick(fps)
228     screen.fill('dark gray')
229     dibujar_tablero()
230     dibujar_piezas()
231     boton_rect = dibujar_boton()
232
233     for event in pygame.event.get():
234         if event.type == pygame.QUIT:

```

```
235         run = False
236
237     if event.type == pygame.MOUSEBUTTONDOWN and
238        event.button == 1:
239         mouse_pos = pygame.mouse.get_pos()
240         if boton_rect.collidepoint(mouse_pos): #
241             Verificar si se hizo clic en el bot n
242
243         nueva_coordenada =
244             calcular_coordenadas(lista_estados[
245                 contador])
246
247         if turn_step <= 1:
248             if coordenadas in posicion_blanca:
249                 selection = posicion_blanca.
250                     index(coordenadas)
251                 if turn_step == 0:
252                     turn_step = 1
253
254             if turn_step==1 and selection !=
255                 100:
256                 posicion_blanca[0] =
257                     nueva_coordenada
258
259                 selection = 100
260                 contador += 1
261                 coordenadas = nueva_coordenada
262                 turn_step=0
263
264     pygame.display.flip()
265
266     pygame.quit()
```

Figura 1.2: Main1.py.

Las librerías importadas en el código tienen las siguientes funciones:

- a) `random`: Esta librería proporciona funciones relacionadas con la generación de números aleatorios. Se utiliza en el código para generar movimientos aleatorios en el juego de ajedrez.
- b) `pygame`: Es una librería de Python que proporciona una API para trabajar con gráficos y sonido. Se utiliza en el código para crear la interfaz gráfica del juego de ajedrez, dibujar el tablero y las piezas, y manejar la interacción con el usuario.
- c) `os`: Esta librería proporciona funciones para interactuar con el sistema operativo. Se utiliza en el código para manipular archivos y directorios, como la generación y lectura de archivos de texto que contienen los movimientos ganadores del juego de ajedrez.:

La siguiente sección de código inicializa y configura `pygame`, establece el tamaño y el título de la ventana del juego, crea objetos de fuente para mostrar texto en el juego y establece la velocidad de actualización del juego.

- a) `pygame.init()`: Esta función inicializa el módulo `pygame` y permite el acceso a las funciones y métodos proporcionados por la librería. Es necesario llamar a esta función antes de utilizar cualquier otra funcionalidad de `pygame`.
- b) `WIDTH = 1000` y `HEIGHT = 700`: Estas variables definen el ancho y la altura de la ventana del juego en píxeles. Se utilizan para establecer el tamaño de la ventana que se mostrará en la pantalla.
- c) `screen = pygame.display.set_mode((WIDTH,HEIGHT))`: Esta línea crea una ventana de visualización del juego con el tamaño especificado por las variables `WIDTH` y `HEIGHT`. La ventana se almacena en la variable `screen` y se utilizará más adelante para dibujar los elementos del juego.

- d) `pygame.display.set_caption('Problema del Ajedrez')`: Esta función establece el título de la ventana del juego. En este caso, se establece como "Problema del Ajedrez".
- e) `font = pygame.font.Font('freesansbold.ttf',20)`: Esta línea crea un objeto de fuente pygame utilizando el archivo de fuente 'freesansbold.ttf' con un tamaño de 20 píxeles. Esta fuente se utilizará para mostrar texto en el juego con un tamaño de fuente de 20.
- f) `big_font = pygame.font.Font('freesansbold.ttf',50)`: Esta línea crea otro objeto de fuente pygame utilizando el mismo archivo de fuente, pero con un tamaño de 50 píxeles. Esta fuente se utilizará para mostrar texto más grande en el juego.
- g) `timer = pygame.time.Clock()`: Esta línea crea un objeto de reloj pygame que se utilizará para controlar la velocidad de actualización del juego. Con el reloj, se puede establecer la cantidad de fotogramas por segundo (fps) a los que se actualizará el juego.
- h) `fps = 60`: Esta variable establece la cantidad de fotogramas por segundo a los que se actualizará el juego. En este caso, se establece en 60 fps, lo que significa que el juego se actualizará 60 veces por segundo.

La sección de código que sigue efectúa:

- a) `tablaEstados`: Esta variable es un diccionario que representa un autómata finito no determinista (NFA) de estados. El NFA está definido con sus estados representados por claves numéricas ('1', '2', ..., '16') y las transiciones entre estados representadas por las letras 'R' (para transiciones regulares) y 'N' (para transiciones nulas). Cada estado tiene un conjunto de estados a los que puede transicionar según la entrada.
- b) `pieza_blanca` y `posicion_blanca`: Estas variables definen las piezas y sus posiciones iniciales en el juego. En este caso, solo se tiene una

pieza blanca ('king') y su posición inicial es (235, 85).

- c) `turn_step` y `selection`: Estas variables son utilizadas para controlar los turnos y la selección de piezas en el juego. Su valor inicial es 0 y 100 respectivamente.
- d) Carga de imágenes: Esta sección carga la imagen del rey blanco desde un archivo en la ruta especificada. Luego, la imagen se escala a un tamaño de 80x80 píxeles. La imagen cargada se asigna a la variable `rey_blanco`. Además, se crea una lista `imagen_blanca` que contiene la imagen del rey blanco.
- e) `lista_piezas`: Esta variable es una lista que contiene el nombre de las piezas en el juego. En este caso, solo se tiene la pieza 'king'.
- f) `boton_presionado`: Esta variable booleana se inicializa en `False` y se utiliza para verificar si un botón ha sido presionado en el juego.

La siguiente sección de código se declaran las funciones del programa, que son:

- a) `dibujar_boton()`:
 - 1) Esta función se encarga de dibujar un botón en la pantalla del juego.
 - 2) Comienza definiendo las dimensiones y la posición del botón utilizando las variables `boton_width`, `boton_height`, `boton_x` y `boton_y`.
 - 3) A continuación, crea un rectángulo llamado `boton_rect` utilizando la clase `pygame.Rect` y los valores de posición y tamaño del botón.

- 4) Dibuja el rectángulo en la pantalla utilizando `pygame.draw.rect` y establece el color como verde (0, 255, 0).
- 5) Crea un objeto de texto llamado `texto` utilizando la función `font.render()`, que renderiza el texto "Siguiente" con una fuente determinada y un color negro (0, 0, 0).
- 6) Calcula la posición del texto en el centro del botón utilizando `texto.get_rect(center=(boton_x + boton_width // 2, boton_y + boton_height // 2))`.
- 7) Finalmente, utiliza `screen.blit()` para pegar el texto en la pantalla y devuelve el rectángulo `boton_rect`.

b) dibujar_tablero():

- 1) Esta función se encarga de dibujar el tablero del juego.
- 2) Comienza definiendo el tamaño de cada cuadro del tablero como `cuadro_size`.
- 3) Calcula el ancho y la altura total del tablero utilizando `tablero_width = 4 * cuadro_size` y `tablero_height = 4 * cuadro_size`.
- 4) Calcula las coordenadas `tablero_x` y `tablero_y` para centrar el tablero en la pantalla.
- 5) A continuación, utiliza un bucle `for` para iterar 16 veces, correspondiente a un tablero de 4x4.
- 6) En cada iteración, calcula la columna y la fila actual a partir del índice.
- 7) Calcula la posición `x` y `y` del cuadro actual utilizando las coordenadas del tablero y el tamaño del cuadro.

- 8) Determina el color del cuadro en función de la fila y la columna. Si la fila es par, alterna entre los colores negro y rojo según la columna. Si la fila es impar, alterna entre los colores rojo y negro.
- 9) Utiliza `pygame.draw.rect()` para dibujar el cuadro en la pantalla y `pygame.draw.rect()` nuevamente para agregar un borde blanco alrededor del cuadro.
- 10) Crea un objeto de texto `numero_texto` que representa el número de cada cuadro utilizando `font.render()`, y calcula su posición en la parte superior del cuadro utilizando `numero_texto.get_rect()`.
Utiliza `screen.blit()` para pegar el número en la pantalla.

14) `dibujar_piezas()`:

- 1) Esta función se encarga de dibujar las piezas en la pantalla.
- 2) Comienza obteniendo el índice de la lista `lista_piezas` donde se encuentra la cadena 'king'.
- 3) Si la pieza blanca actual es un rey (`pieza_blanca[0]=='king'`), utiliza `screen.blit()` para pegar la imagen del rey blanco (`imagen_blanca[index]`) en la posición especificada por `posicion_blanca`.
- 4) La variable `imagen_blanca` es una lista que contiene las imágenes de todas las piezas blancas del ajedrez.

d) `recorrer_estados(tabla_estados, cadena)`:

- 1) Esta función recorre los estados del autómata finito no determinista utilizando una cadena de entrada.
- 2) Toma como argumento `tabla_estados`, que es un diccionario que representa los estados y las transiciones del autómata.

- 3) Comienza con una lista `recorridos_posibles` para almacenar los recorridos posibles y una lista `recorridos_exitosos` para almacenar los recorridos exitosos.
- 4) Luego, llama a la función auxiliar `recorrer_estado()` pasando el estado inicial del autómata, la cadena de entrada, una lista vacía `recorrido_actual` y las listas de recorridos posibles y exitosos.
- 5) Al final de la función, se escriben los recorridos posibles y exitosos en dos archivos de texto separados.

e) `seleccionar_recorrido()`:

- 1) Esta función lee los recorridos exitosos almacenados en un archivo de texto y selecciona uno aleatoriamente.
- 2) Comienza abriendo el archivo de texto que contiene los recorridos exitosos en modo de lectura.
- 3) Lee los recorridos exitosos del archivo y los almacena en una lista llamada `recorridos_exitosos`.
- 4) Si la lista está vacía, genera una nueva cadena de entrada llamando a la función `cadena()` y recalcula los recorridos llamando a `recorrer_estados(tabla_estados, nueva_cadena)`.
- 5) Si la lista no está vacía, selecciona un recorrido aleatorio utilizando `random.choice()` y lo devuelve.

f) `cadenaRandom(numero)`:

- 1) Esta función genera una cadena aleatoria de longitud `numero`.
- 2) Crea una cadena vacía llamada `cadena`.

- 3) En un bucle for que itera numero veces, agrega un carácter aleatorio ('R' o 'N') a la cadena utilizando random.choice().
- 4) Al final del bucle, devuelve la cadena generada.

g) cadena():

- 1) Esta función genera una cadena aleatoria de longitud aleatoria (entre 4 y 10) utilizando la función cadenaRandom().
- 2) Genera un número aleatorio entre 4 y 10 utilizando random.randint().
- 3) Llama a cadenaRandom() pasando el número generado y almacena la cadena generada en una variable llamada cadena.
- 4) Imprime la cadena generada utilizando print() y la devuelve.

h) calcular_coordenadas(estado):

- 1) Esta función calcula las coordenadas (x, y) correspondientes a un estado del tablero.
- 2) Utiliza la función divmod() para dividir el estado entre 4 y obtener el cociente y el residuo.
- 3) El cociente representa la fila y el residuo representa la columna.
- 4) Multiplica la columna por el tamaño del cuadro y suma tablero_x para obtener la coordenada x.
- 5) Multiplica la fila por el tamaño del cuadro y suma tablero_y para obtener la coordenada y.
- 6) Devuelve las coordenadas como una tupla (x, y).

La parte final del código es el ciclo principal del juego que utiliza la biblioteca Pygame para crear una interfaz gráfica y permitir al usuario interactuar con el tablero de ajedrez. A continuación, se explica qué hace cada sección del código:

- 1) `ruta = cadena()`: Genera una cadena aleatoria utilizando la función `cadena()` y la asigna a la variable `ruta`. Esta cadena se utilizará como entrada para recorrer los estados del autómata.
- 2) `recorrer_estados(tablaEstados, ruta)`: Llama a la función `recorrer_estados()` pasando el diccionario `tablaEstados` (que representa los estados y transiciones del autómata) y la cadena `ruta`. Esto iniciará el recorrido del autómata utilizando la cadena de entrada.
- 3) `print('este punto el programa almaceno en la salidas los recorridos posibles y los recorridos exitosos en la carpeta output.')`: Imprime un mensaje en la consola indicando que los recorridos posibles y exitosos se han almacenado en la carpeta de salida.
- 4) `recorrido = seleccionar_recorrido()`: Llama a la función `seleccionar_recorrido()` para seleccionar aleatoriamente uno de los recorridos exitosos generados anteriormente. El recorrido seleccionado se almacena en la variable `recorrido`.
- 5) `With open("C:/ Users/ soyco/OneDrive/ Documents/ ESCOM/ sem4/ Teoria/ P2/ Chess/ output/ ruta_blanca.txt", 'w')` as `archivo`: `archivo.write(ruta)`: Abre un archivo de texto en modo de escritura y escribe la cadena `ruta` en él. El archivo se guarda en la ubicación especificada.
- 6) `print("seleccionado:", recorrido)`: Imprime en la consola el recorrido seleccionado aleatoriamente.
- 7) `print("la parte gráfica.")`: Imprime en la consola un mensaje indicando que se mostrará la parte gráfica del juego.
- 8) `lista_estados = [int(num) for num in recorrido.split(",")]`: Convierte la cadena `recorrido` en una lista de enteros, dividiendo la cadena

en cada coma y convirtiendo cada elemento en un entero. Esta lista representa los estados del tablero que se recorrerán en el juego.

- 9) `coordenadas = (235, 85)`: Inicializa la variable `coordenadas` con las coordenadas iniciales en el tablero.
- 10) `nueva_coordenada = (0, 0)`: Inicializa la variable `nueva_coordenada` con las coordenadas iniciales en el tablero.
- 11) `run = True`: Inicializa la variable `run` en `True` para iniciar el ciclo principal del juego.
- 12) `contador = 1`: Inicializa la variable `contador` en 1 para llevar un seguimiento del estado actual del recorrido.
- 13) `while run::` Inicia un bucle `while` que se ejecuta mientras la variable `run` sea verdadera.
- 14) `timer.tick(fps)`: Controla la velocidad de actualización de la pantalla según el número de fotogramas por segundo especificado por `fps`.
- 15) `screen.fill('dark gray')`: Rellena la pantalla con un color de fondo gris oscuro.
- 16) `dibujar_tablero()`: Dibuja el tablero de ajedrez en la pantalla.
- 17) `dibujar_piezas()`: Dibuja las piezas de ajedrez en las posiciones correspondientes en el tablero.
- 18) `boton_rect = dibujar_boton()`: Dibuja un botón en la pantalla y devuelve su rectángulo para detectar interacciones con el mouse.
- 19) `for event in pygame.event.get()::` Itera sobre los eventos que ocurren en el juego.

- 20) `if event.type == pygame.QUIT::` Verifica si se ha producido un evento de salida, es decir, si el usuario ha cerrado la ventana del juego. En ese caso, se establece `run = False` para salir del bucle principal.
- 21) `if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1::` Verifica si se ha producido un evento de clic del mouse con el botón izquierdo.
- 22) `mouse_pos = pygame.mouse.get_pos():` Obtiene la posición actual del mouse.
- 23) `if boton_rect.collidepoint(mouse_pos)::` Verifica si la posición del mouse está dentro del rectángulo del botón.
- 24) `nueva_coordenada = calcular_coordenadas(lista_estados[contador]):` Llama a la función `calcular_coordenadas()` pasando el estado actual del recorrido y asigna el resultado a la variable `nueva_coordenada`.
- 25) `if turn_step <= 1::` Verifica si el paso de turno es menor o igual a 1.
- 26) `if coordenadas in posicion_blanca::` Verifica si las coordenadas actuales se encuentran en la lista `posicion_blanca`.
- 27) `selection = posicion_blanca.index(coordenadas):` Obtiene el índice de las coordenadas actuales en la lista `posicion_blanca` y lo asigna a la variable `selection`.
- 28) `if turn_step == 0::` Verifica si el paso de turno es igual a 0.
- 29) `turn_step = 1:` Establece el paso de turno en 1.
- 30) `if turn_step == 1 and selection != 100::` Verifica si el paso de turno es igual a 1 y si la selección de pieza (`selection`) no es igual a 100.

- 31) `posicion_blanca[0] = nueva_coordenada`: Actualiza la posición de la pieza blanca en la lista `posicion_blanca` con la nueva coordenada.
 - 32) `selection = 100`: Establece `selection` en 100 para indicar que no se ha realizado una selección de pieza.
 - 33) `contador += 1`: Incrementa el contador en 1 para avanzar al siguiente estado del recorrido.
 - 34) `coordenadas = nueva_coordenada`: Actualiza las coordenadas actuales con las nuevas coordenadas.
 - 35) `turn_step = 0`: Establece el paso de turno en 0.
 - 36) `pygame.display.flip()`: Actualiza la pantalla para mostrar los cambios realizados en cada iteración del bucle.
 - 37) `pygame.quit()`: Cierra la ventana.
3. Ahora haremos énfasis en lo siguiente: El código que se acaba de explicar es el `main1.py` que se reitera es la funcionalidad de recorrido aleatorio para 1 pieza. Ahora bien, los 5 mains parten del mismo código, y solo son unas partes muy específicas que se modificaron. Observar figura 1.3.
- La diferencia principal entre el código `main1.py` y el `main2.py` es la forma en que se generan los recorridos de la pieza en el tablero de ajedrez, las diferencias son:
- a) En el código del `main1.py`:
 - La función `recorrer_estados` utiliza la función `obtener_recorridos` para obtener todos los recorridos posibles de la pieza. Se inicia desde el estado '1' y se generan todos los posibles recorridos hasta que se alcance el estado final '16'.
 - La función `obtener_recorridos` utiliza recursión para explorar todas las transiciones posibles en el tablero. Comienza con el estado actual y el

primer símbolo de la cadena de movimientos. Luego, para cada transición válida desde el estado actual con el símbolo dado, se avanza al estado siguiente y se llama recursivamente a la función con el siguiente símbolo y el recorrido actualizado.

Los recorridos posibles se almacenan en el archivo `recorridos_blanca.txt`, y los recorridos válidos hasta el estado final se almacenan en el archivo `recorridos_finales_blanca.txt`.

b) En el código del `main2.py`:

La función `recorrer_estados` también utiliza la función `obtener_recorridos`, pero en este caso, se pasa un archivo como parámetro. En lugar de almacenar los recorridos en una lista en memoria, los recorridos se escriben directamente en el archivo a medida que se generan. Esto evita la necesidad de almacenar todos los recorridos en memoria antes de escribirlos en el archivo.

La función `obtener_recorridos_finales` también recibe un archivo como parámetro y escribe los recorridos válidos hasta el estado final directamente en el archivo. En lugar de utilizar una lista de recorridos y luego escribirlos en los archivos, el código 2 escribe cada recorrido en el archivo a medida que se genera, lo que ahorra memoria y permite manejar recorridos de mayor longitud.

Además de la diferencia en la generación de los recorridos, se agrega una interacción con el usuario en la función `seleccionar_recorrido` del código 2, donde se le solicita al usuario ingresar una nueva ruta si el archivo está vacío. Esta característica no está presente en el código 1.

```
1 def recorrer_estados(tabla_estados, cadena):  
2  
3     # Función para obtener todos los recorridos  
4     # posibles  
5     def obtener_recorridos(estado_actual,  
6         simbolos_restantes, recorrido_actual,  
7         archivo_recorridos):  
8         if not simbolos_restantes:  
9             archivo_recorridos.write(','.join(  
10                 recorrido_actual) + '\n')
```

```
7         return
8
9         simbolo = simbolos_restantes[0]
10        if estado_actual in tabla_estados and simbolo
11           in tabla_estados[estado_actual]:
12            transiciones = tabla_estados[estado_actual
13                               ][simbolo]
14
15            for estado_siguiente in transiciones:
16                obtener_recorridos(
17                    estado_siguiente,
18                    simbolos_restantes[1:],
19                    recorrido_actual + [
20                        estado_siguiente],
21                    archivo_recorridos
22                )
23
24        # Funci n para obtener los recorridos v lidos
25        hasta el estado final
26
27        def obtener_recorridos_finales(estado_actual,
28            simbolos_restantes, recorrido_actual,
29            archivo_recorridos_finales):
30            if estado_actual == '16':
31                archivo_recorridos_finales.write(','.join(
32                    recorrido_actual) + '\n')
33            return
34
35        if not simbolos_restantes:
36            return
37
38        simbolo = simbolos_restantes[0]
39        if estado_actual in tabla_estados and simbolo
40           in tabla_estados[estado_actual]:
41            transiciones = tabla_estados[estado_actual
42                               ][simbolo]
43
44            for estado_siguiente in transiciones:
45                obtener_recorridos_finales(
46                    estado_siguiente,
47                    simbolos_restantes[1:],
```

```
38         recorrido_actual + [  
39             estado_siguiente],  
40         archivo_recorridos_finales  
41     )  
42     # Obtener recorridos posibles  
43     with open('C:\\Users\\soyco\\OneDrive\\Documents\\  
         ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\  
         recorridos_blanca.txt', 'w') as  
         archivo_recorridos:  
44         obtener_recorridos('1', cadena, ['1'],  
             archivo_recorridos)  
45  
46     # Obtener recorridos hasta el estado final  
47     with open('C:\\Users\\soyco\\OneDrive\\Documents\\  
         ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\  
         recorridos_finales_blanca.txt', 'w') as  
         archivo_recorridos_finales:  
48         obtener_recorridos_finales('1', cadena, ['1'],  
             archivo_recorridos_finales)  
49  
50  
51 def seleccionar_recorrido():  
52     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\  
         Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\  
         output\\recorridos_finales_blanca.txt"  
53  
54     with open(ruta_archivo, "r") as archivo:  
55         if os.path.getsize(ruta_archivo) == 0:  
56             print("No existen soluciones que lleguen  
                 al estado 16 con la condicion actual.  
                 Ingrese nueva ruta.")  
57             nuevavuta=cadena()  
58             print("La nueva ruta es: "+nuevaruta)  
59             recorrer_estados(tablaEstados, nuevavuta)  
60             print('Se almacenaron las nuevas salidas  
                 de los recorridos posibles y los  
                 recorridos exitosos en la carpeta  
                 output.\\n')  
61             lineas = archivo.readlines()
```

```
62         # Seleccionar una l nea aleatoria
63         recorrido_seleccionado = random.choice(
64             lineas)
65
66         # Eliminar los espacios en blanco y saltos
67         # de l nea
68         recorrido_seleccionado =
69             recorrido_seleccionado.strip()
70
71         return recorrido_seleccionado
72
73     lineas = archivo.readlines()
74
75     # Seleccionar una l nea aleatoria
76     recorrido_seleccionado = random.choice(lineas)
77
78     # Eliminar los espacios en blanco y saltos de
79     # l nea
80     recorrido_seleccionado =
81         recorrido_seleccionado.strip()
82
83     return recorrido_seleccionado
84
85 def cadena():
86     while 1:
87         ruta = input("Ingrese la cadena usando las
88                     transiciones R y N.\n").upper()
89         if len(ruta) > 10:
90             print("\nNo es posible introducir m s de
91                   10 caracteres en la cadena de recorrido
92                   para animacion. Desea seguir de igual
93                   forma?(Ya no se animara, se procedera
94                   al arbol)")
95             respuesta = int(input("1. Si.\n2. No.\n"))
96             if respuesta == 1:
97                 break
98             else:
99                 pass
100     else:
```

```

92         print()
93         break
94     print('\nTamaño de cadena escogido [' + str(len(
95         ruta)) + '].')
96     print('\nLa cadena o ruta a seguir escogida sera:
97         '+ ruta + '\n')
98     return ruta
99
100 #Main del ciclo del juego 2
101 ruta=cadena()#Solicitamos la cadena generada
102     aleatoriamente.
103 with open('C:\\Users\\soyco\\OneDrive\\Documents\\
104     ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca
105     .txt', 'w') as archivo:
106     archivo.write(ruta)
107 if len(ruta)> 10:
108     sys.exit(3)
109 recorrer_estados(tablaEstados, ruta)
110 print('\nEn este punto el programa almaceno en la
111     salidas los recorridos posibles y los recorridos
112     exitosos en la carpeta output.\n')
113 recorrido = seleccionar_recorrido()#Este fue el
114     recorrido que se escogio de manera aleatoria
115
116 with open('C:\\Users\\soyco\\OneDrive\\Documents\\
117     ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca
118     .txt', 'w') as archivo:
119     archivo.write(ruta)

```

Figura 1.3: Código y funciones modificadas del main1.py para el main2.py.

En el código main3.py, que maneja el funcionamiento de dos piezas (una negra y una blanca), hay varias diferencias con respecto al código 1 que permite el movimiento de una sola pieza a la vez. Estas son algunas de las partes del código que cambian para lograr el cometido.

Se introducen las siguientes variables adicionales:

- a) `coordenadas_blanca` y `coordenadas_negra`: Representan las coordenadas iniciales de la pieza blanca y la pieza negra, respectivamente.
- b) `nueva_coordenada_blanca` y `nueva_coordenada_negra`: Almacenan las coordenadas calculadas para el siguiente movimiento de la pieza blanca y la pieza negra, respectivamente.
- c) `contador_blanca` y `contador_negra`: Realizan un seguimiento del índice actual en la lista de estados de la pieza blanca y la pieza negra, respectivamente.
- d) Se agrega un condicional al inicio del ciclo principal del juego para determinar qué pieza debe empezar primero. Esto se realiza mediante el uso de la variable `numero_aleatorio`, que se genera aleatoriamente entre 1 y 2.
- e) Se agrega una sección de código para manejar el movimiento de la pieza negra. En esta sección, se verifica si la nueva coordenada de la pieza negra colisiona con la posición de la pieza blanca. Si ocurre una colisión, se recalculará la ruta de la pieza negra hasta que no haya colisiones. Esto se logra mediante un ciclo `while`.
- f) Dentro del ciclo principal del juego, se agrega un bloque de código adicional para manejar el movimiento de la pieza negra. Esto se realiza cuando `turn_step` tiene el valor de 2. En este caso, se verifica si la nueva coordenada de la pieza negra colisiona con la posición de la pieza blanca. Si no hay colisión, se actualizan las coordenadas de la pieza negra y se incrementa el contador de la pieza negra.
- g) En ambos bloques de código, se seleccionan las coordenadas de la pieza actual (blanca o negra) según el valor de `turn_step`, y se comprueba si se hizo clic en el botón para realizar el movimiento. Si se cumple esta condición, se calculan las nuevas coordenadas para la pieza correspondiente según el estado actual y se actualizan las variables de

seguimiento y posición.

Estas son algunas de las diferencias principales en el código main3 en comparación con el código main1. Estas modificaciones permiten manejar el movimiento de dos piezas en el tablero de ajedrez, asegurando que no haya colisiones entre ellas.

La principal diferencia se ve detallada en la figura 1.4, donde esta parte específica de código se encarga de detectar una colisión entre la nueva coordenada de la pieza negra y la posición de la pieza blanca. Si se detecta una colisión, se ejecuta un ciclo while para recalcular la ruta de la pieza negra hasta encontrar una ruta sin colisiones.

Aquí se explica paso a paso lo que ocurre en ese fragmento:

- a) Se verifica si la nueva_coordenada_negra se encuentra en la lista posicion_blanca, que almacena las coordenadas ocupadas por la pieza blanca.
- b) Si la nueva_coordenada_negra está en posicion_blanca, significa que se ha detectado una colisión.
- c) Se entra en un ciclo while que se ejecutará continuamente hasta que se encuentre una nueva coordenada para la pieza negra que no colisione con la pieza blanca. El ciclo está definido con while(1), lo que significa que se ejecutará indefinidamente hasta que se encuentre una salida.
- d) Dentro del ciclo while, se realiza lo siguiente:
 - 1) Se imprime un mensaje indicando que se ha detectado una colisión en el recorrido y que se procederá a recalcular la ruta de la pieza negra.

- 2) Se solicita una nueva cadena de movimiento aleatoria mediante la función `cadena()`, que generará una cadena de movimientos válidos para la pieza negra.
 - 3) Se actualiza el estado de la pieza negra utilizando la función `recorrer_estados_negra()`, que modificará la lista `tablaEstados` con los nuevos estados generados a partir de la nueva cadena de movimiento.
 - 4) Se selecciona un nuevo recorrido para la pieza negra utilizando la función `seleccionar_recorrido_negra()` y se actualiza la lista de estados `lista_estados_negra` con los estados del nuevo recorrido.
 - 5) Se reinicia el contador de la pieza negra a 1 para empezar desde el primer estado del nuevo recorrido.
 - 6) Se calcula una nueva coordenada para la pieza negra utilizando la función `calcular_coordenadas()` con el estado actual de la pieza negra.
 - 7) Se verifica si la nueva coordenada de la pieza negra no está en la posición de la pieza blanca. Si no hay colisión, se rompe el ciclo `while` y se sale de él.
- e) Después de salir del ciclo `while`, se establece `turn_step` en 0 para indicar que es el turno de la siguiente pieza.

Esta parte del código se encarga de manejar la colisión detectada entre la pieza negra y la pieza blanca, recalculando la ruta de la pieza negra hasta que se encuentre una nueva coordenada sin colisiones. Observar figura 1.4.

```
1 if nueva_coordenada_negra in posicion_blanca:
2     #Ciclo while, donde para salir la siguiente coordenada
      no pueda ser la de la colision
3     while (1):
4         #recalculamos ruta
```

```
5     print("Colision detectada en recorrido:  
        Recalculamos negra.")  
6     ruta_negra=cadena()#Solicitamos la cadena  
        generada aleatoriamente. Para pieza negra.  
7     recorrer_estados_negra(tablaEstados,  
        ruta_negra,str(lista_estados_negra[  
        contador_negra-1]))  
8     recorrido_negra = seleccionar_recorrido_negra(  
        str(lista_estados_negra[contador_negra-1]))  
        #Este fue el recorrido que se escogio de  
        manera aleatoria para blanca  
9     lista_estados_negra = [int(num) for num in  
        recorrido_negra.split(",")]  
10    contador_negra=1  
11    nueva_coordenada_negra = calcular_coordenadas(  
        lista_estados_negra[contador_negra])  
12    if nueva_coordenada_negra not in  
        posicion_blanca:  
13        break  
14    turn_step=0
```

Figura 1.4: Lógica de detección de colisiones, creamos una nueva ruta.

Lo mismo pasa para el main4 y el main5, donde lo unico que se modifica son las funciones de recorrido, tal como hemos visto anteriormente, solo re-utilizamos el código para que 1 pieza sea aleatoria y la otra manual, y para el main5 que ambas sean manuales.

Pasamos a la parte de los graficadores para el arbol de trancisiones.

El archivo graficador.py contiene un conjunto de funciones y código para generar gráficos de los recorridos de las piezas blancas y negras en el ajedrez, utilizando la biblioteca graphviz. Aquí se explica brevemente cada parte del código:

- a) Se importan los módulos sys y Digraph de la biblioteca graphviz.
- b) Se verifica si se pasó un argumento al ejecutar el script desde la línea de comandos. Si se pasó, se asigna el valor del argumento a la variable x.
- c) Se definen las funciones *recorrer_estados_blanca* y *recorrer_estados_negra* que se encargan de la función *agregar_recorrido_al_grafo* se utiliza para agregar un recorrido al grafo que repite los pasos 7 y 8 para las piezas blancas y negras respectivamente.
- d) La función *generar_arbol_recorridos* se encarga de leer los recorridos desde un archivo de recorridos y agregarlos al grafo. Recibe la tabla_estados, la ruta_evaluar, los archivos de entrada y salida, y utiliza la función *agregar_recorrido_al_grafo*.
- e) Se define la tabla Estados que representa los movimientos posibles de las piezas en el ajedrez. Cada estado tiene asociados movimientos con las piezas 'R' (torre) y 'N' (caballo).
- f) Si el valor de x es "1", se realiza la generación y renderizado del árbol de recorridos para las piezas blancas.
- g) Se lee la ruta a evaluar desde el archivo ruta_blanca.txt.
- h) Se ejecuta la función *recorrer_estados_blanca* para generar los recorridos y escribirlos en el archivo recorridos_blanca.txt.
- i) Se utiliza la función *generar_arbol_recorridos* para crear el grafo y guardar el archivo .dot.
- j) Si el valor de x es distinto de "1" (asumiendo que será "2.^{en} este caso), se realiza la generación y renderizado del árbol de recorridos tanto para las piezas blancas como para las negras.
- k) Se repiten los pasos 7 y 8 para las piezas blancas y negras respectivamente.

El formato de recorridos que acepta este graficador es:

```
"recorrido 1"  
"recorrido 2"  
" ..."  
"recorrido N"
```

Funciona para archivos con recorridos no tan largos, factibles para las animaciones, y su código es el siguiente:

```
1 import sys  
2 from graphviz import Digraph  
3  
4 # Leer el argumento pasado desde el programa principal  
5 x=0  
6 if len(sys.argv) > 1:  
7     x=((sys.argv[1]))  
8     #print((x))  
9 else:  
10    print("No se pas ning n argumento.")  
11    sys.exit(1)  
12  
13 def recorrer_estados_blanca(tabla_estados, cadena):  
14     def obtener_recorridos(estado_actual,  
15                            simbolos_restantes, recorrido_actual,  
16                            archivo_recorridos):  
17         if not simbolos_restantes:  
18             archivo_recorridos.write(','.join(  
19                 recorrido_actual) + '\n')  
20             return  
21  
22         simbolo = simbolos_restantes[0]  
23         if estado_actual in tabla_estados and simbolo  
24             in tabla_estados[estado_actual]:  
25             transiciones = tabla_estados[estado_actual]  
26                 [simbolo]  
27  
28             for estado_siguiete in transiciones:  
29                 obtener_recorridos(  
30                     estado_siguiete,
```

```
26         simbolos_restantes[1:],
27         recorrido_actual + [
28             estado_siguiente],
29         archivo_recorridos
30     )
31     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
32             ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
33             recorridos_blanca.txt', 'w') as
34         archivo_recorridos:
35             obtener_recorridos('1', cadena, ['1'],
36                                 archivo_recorridos)
37
38 def recorrer_estados_negra(tabla_estados, cadena):
39     def obtener_recorridos2(estado_actual,
40                             simbolos_restantes, recorrido_actual,
41                             archivo_recorridos):
42         if not simbolos_restantes:
43             archivo_recorridos.write(','.join(
44                 recorrido_actual) + '\n')
45             return
46
47         simbolo = simbolos_restantes[0]
48         if estado_actual in tabla_estados and simbolo
49             in tabla_estados[estado_actual]:
50             transiciones = tabla_estados[estado_actual]
51                 [simbolo]
52
53             for estado_siguiente in transiciones:
54                 obtener_recorridos2(
55                     estado_siguiente,
56                     simbolos_restantes[1:],
57                     recorrido_actual + [
58                         estado_siguiente],
59                     archivo_recorridos
60                 )
61
62     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
63             ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
64             recorridos_negra.txt', 'w') as
```

```

    archivo_recorridos:
53     obtener_recorridos2('4', cadena, ['4'],
        archivo_recorridos)
54
55 def agregar_recorrido_al_grafo(grafo, transicion,
    transiciones_posibles):
56     estados = transicion.split(',')
57
58     for i in range(len(estados) - 1):
59         origen = estados[i]
60         destino = estados[i+1]
61         if [origen, destino] not in
            transiciones_posibles:
62             # Obtener el simbolo asociado a la
                transicion
63             if origen in tablaEstados and 'R' in
                tablaEstados[origen] and destino in
                tablaEstados[origen]['R']:
64                 simbolo = 'R'
65             elif origen in tablaEstados and 'N' in
                tablaEstados[origen] and destino in
                tablaEstados[origen]['N']:
66                 simbolo = 'N'
67             else:
68                 simbolo = ''
69             #Parte donde se calcula el simbolo
                asociado a la transicion o arista/
                flecha, recuerda que puede ser N o R
                dependiendo de la tabla de estados.
70             grafo.edge(origen, destino, arrowhead="
                normal", label=simbolo) # Utilizamos el
                m todo 'edge' para agregar una arista
71             transiciones_posibles.append([origen,
                destino],)
72
73
74 def generar_arbol_recorridos(tabla_estados,
    ruta_evaluar, archivo_recorridos, archivo_salida):
75     grafo = Digraph('G', filename='arbol.gv')
76     grafo.attr(rankdir='LR', size='8,5')

```



```

77     grafo.graph_attr['label'] = ruta_evaluar #
       Agregar t tulo al gr fico
78     with open(archivo_recorridos, 'r') as file:
79         for linea in file:
80             recorrido = linea.strip()
81             agregar_recorrido_al_grafo(grafo,
                                         recorrido,transiciones_posibles)
82
83     archivo_salida_pdf = archivo_salida.replace('.dot'
84         , '.pdf')
85     grafo.render(archivo_salida_pdf, view=True)
86
87     tablaEstados = {
88         '1': {'R': {'2','5'}, 'N': '6'},
89         '2': {'R': {'5','7'}, 'N': {'1','6','3'}},
90         '3': {'R': {'2','7','4'}, 'N': {'6','8'}},
91         '4': {'R': '7', 'N': {'3','8'}},
92         '5': {'R': {'2','10'}, 'N': {'1','6','9'}},
93         '6': {'R': {'2','5','7','10'}, 'N': {'1','3','9','
94             11'}},
95         '7': {'R': {'2','4','10','12'}, 'N': {'3','6','8',
96             '11'}},
97         '8': {'R': {'4','7','12'}, 'N': {'3','11'}},
98         '9': {'R': {'5','10','13'}, 'N': {'6','14'}},
99         '10': {'R': {'5','7','13','15'}, 'N': {'6','9','11
100             ','14'}},
101         '11': {'R': {'7','10','12','15'}, 'N': {'6','8','
102             14','16'}},
103         '12': {'R': {'7','15'}, 'N': {'8','11','16'}},
104         '13': {'R': '10', 'N': {'9','14'}},
105         '14': {'R': {'13','10','15'}, 'N': {'9','11'}},
106         '15': {'R': {'10','12'}, 'N': {'11','14','16'}},
107         '16': {'R': {'12','15'}, 'N': '11'}
108     }
109
110     if x == "1":
111         archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\
112             Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
113             output\\ruta_blanca.txt'
114         # Abre el archivo en modo de lectura

```

```
108     archivo = open(archivo_ruta, "r")
109
110     # Lee una l nea del archivo
111     ruta_evaluar = archivo.readline()
112
113     # Imprime la l nea le da
114     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\recorridos_blanca.txt'
115     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\arbol_blanca.dot'
116     transiciones_posibles = []
117     recorrer_estados_blanca(tablaEstados, ruta_evaluar
        )
118     generar_arbol_recorridos(tablaEstados,
        ruta_evaluar, archivo_recorridos,
        archivo_salida)
119
120 else:
121     #Redenrizmos ruta blanca
122     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\ruta_blanca.txt'
123     # Abre el archivo en modo de lectura
124     archivo = open(archivo_ruta, "r")
125
126     # Lee una l nea del archivo
127     ruta_evaluar = archivo.readline()
128
129     # Imprime la l nea le da
130     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\recorridos_blanca.txt'
131     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\arbol_blanca.dot'
132     transiciones_posibles = []
133     recorrer_estados_blanca(tablaEstados, ruta_evaluar
        )
```

```
134     generar_arbol_recorridos(tablaEstados,  
        ruta_evaluar, archivo_recorridos,  
        archivo_salida)  
135  
136     #Renderizamos ruta negra  
137     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\  
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\  
        output\\ruta_negra.txt'  
138     # Abre el archivo en modo de lectura  
139     archivo = open(archivo_ruta, "r")  
140  
141     # Lee una l nea del archivo  
142     ruta_evaluar = archivo.readline()  
143  
144     # Imprime la l nea le da  
145     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\  
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\  
        output\\recorridos_negra.txt'  
146     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\  
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\  
        output\\arbol_negra.dot'  
147     transiciones_posibles = []  
148     recorrer_estados_negra(tablaEstados, ruta_evaluar)  
149     generar_arbol_recorridos(tablaEstados,  
        ruta_evaluar, archivo_recorridos,  
        archivo_salida)
```

Figura 1.5: Graficador.py.

Las diferencias entre los códigos `graficador_grande.py` y `graficador.py` son las siguientes:

- a) Funcionalidad ampliada: El código `graficador_grande.py` proporciona una funcionalidad más amplia en comparación con `graficador.py`. Mientras que `graficador.py` genera un solo árbol de recorridos para una ruta (ya sea blanca o negra), `graficador_grande.py` puede generar

múltiples árboles de recorridos para diferentes rutas.

- b) Manejo de múltiples rutas: En `graficador_grande.py`, puedes proporcionar múltiples rutas para generar árboles de recorridos individuales. El código procesará cada ruta por separado y generará un árbol de recorridos para cada una de ellas. Esto permite visualizar diferentes escenarios de juego o analizar diversas situaciones.
- c) Parámetros de entrada: Mientras que `graficador.py` lee la ruta desde un archivo y elige si generar el árbol de recorridos para la ruta blanca o negra basándose en el valor de `x`, `graficador_grande.py` requiere que se le proporcionen las rutas directamente como parámetros de entrada al ejecutar el script. Esto proporciona una mayor flexibilidad al permitirte especificar las rutas que deseas analizar.
- d) Salida en formato PDF: Al igual que `graficador.py`, `graficador_grande.py` genera archivos de salida en formato PDF que contienen los árboles de recorridos. Estos archivos se pueden ver y compartir fácilmente, lo que facilita el análisis y la visualización de los recorridos de estados.

El formato de recorridos que acepta este graficador es:

”recorrido 1.recorrido 2.—.recorrido N.”

Funciona para archivos con recorridos muy largos, factibles para el recorrido del árbol, y su código es el siguiente:

```
1 import sys
2 from graphviz import Digraph
3
4 # Leer el argumento pasado desde el programa principal
5 x=0
6 if len(sys.argv) > 1:
7     x=(sys.argv[1])
8     #print((x))
9 else:
10    print("No se pas ning n argumento.")
```

```
11     sys.exit(1)
12
13 def recorrer_estados_blanca(tabla_estados, cadena):
14     cont=0
15     cont2=0
16     def obtener_recorridos(estado_actual,
17                             simbolos_restantes, recorrido_actual,
18                             archivo_recorridos):
19         nonlocal cont # Declarar 'cont' como una
20                         variable no local
21         nonlocal cont2 # Declarar 'cont' como una
22                         variable no local
23         if not simbolos_restantes:
24             cont=cont+1
25             archivo_recorridos.write(','.join(
26                 recorrido_actual) + ',.')
27             if cont==1000:
28                 archivo_recorridos.write('\n')
29                 cont2=cont+cont2
30                 cont=0
31             return
32         simbolo = simbolos_restantes[0]
33         if estado_actual in tabla_estados and simbolo
34             in tabla_estados[estado_actual]:
35             transiciones = tabla_estados[estado_actual]
36                 [simbolo]
37
38             for estado_siguiente in transiciones:
39                 obtener_recorridos(estado_siguiente,
40                                     simbolos_restantes[1:],
41                                     recorrido_actual + [
42                                         estado_siguiente],
43                                     archivo_recorridos)
44             if cont2==100000:
45                 #input("100 cien mil pos")
46                 break
47
48     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
49             ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
50             recorridos_blanca.txt', 'w') as
```

```

    archivo_recorridos:
38     obtener_recorridos('1', cadena, ['1'],
        archivo_recorridos)
39
40 def recorrer_estados_negra(tabla_estados, cadena):
41     def obtener_recorridos2(estado_actual,
        simbolos_restantes, recorrido_actual,
        archivo_recorridos):
42         if not simbolos_restantes:
43             archivo_recorridos.write(','.join(
                recorrido_actual) + '.')
44             return
45
46         simbolo = simbolos_restantes[0]
47         if estado_actual in tabla_estados and simbolo
            in tabla_estados[estado_actual]:
48             transiciones = tabla_estados[estado_actual]
                [simbolo]
49
50             for estado_siguiente in transiciones:
51                 obtener_recorridos2(estado_siguiente,
                    simbolos_restantes[1:],
                    recorrido_actual + [
                        estado_siguiente],
                    archivo_recorridos)
52
53     with open('C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_negra.txt', 'w') as
        archivo_recorridos:
54         obtener_recorridos2('4', cadena, ['4'],
            archivo_recorridos)
55
56 def agregar_recorrido_al_grafo(grafo, transicion,
    transiciones_posibles):
57     estados = transicion.split(',')
58     for i in range(len(estados) - 1):
59         origen = estados[i]
60         destino = estados[i+1]
61         if [origen, destino] not in
```

```

transiciones_posibles:
62     # Obtener el simbolo asociado a la
        transición
63     if origen in tablaEstados and 'R' in
        tablaEstados[origen] and destino in
        tablaEstados[origen]['R']:
64         simbolo = 'R'
65     elif origen in tablaEstados and 'N' in
        tablaEstados[origen] and destino in
        tablaEstados[origen]['N']:
66         simbolo = 'N'
67     else:
68         simbolo = ''
69     #Parte donde se calcula el simbolo
        asociado a la transición o arista/
        flecha, recuerda que puede ser N o R
        dependiendo de la tabla de estados.
70     grafo.edge(origen, destino, arrowhead="
        normal", label=simbolo) # Utilizamos el
        m todo 'edge' para agregar una arista
71     transiciones_posibles.append([origen,
        destino],)
72
73
74 def generar_arbol_recorridos(tabla_estados,
    ruta_evaluar, archivo_recorridos, archivo_salida):
75     grafo = Digraph('G', filename=archivo_salida)
76     grafo.attr(rankdir='LR', size='8,5')
77     grafo.graph_attr['label'] = ruta_evaluar #
        Agregar titulo al grafico
78     # Leer los recorridos desde el archivo
79     with open(archivo_recorridos, 'r') as archivo:
80         # Agregar los recorridos al grafo
81         transiciones_posibles = []
82         for line in archivo:
83             linea_aux = line.strip()
84
85         for linea_aux in archivo:
86             recorridos = linea_aux.strip().split('
                .')

```

```

87         for recorrido in recorridos:
88             if recorrido:
89                 agregar_recorrido_al_grafo(
90                     grafo, recorrido,
91                     transiciones_posibles)
92
93     # Renderizar el grafo y guardar la imagen
94     archivo_salida_pdf = archivo_salida.replace('.dot'
95         , '.pdf')
96     grafo.render(archivo_salida_pdf, view=True)
97
98     tablaEstados = {
99         '1': {'R': {'2', '5'}, 'N': '6'},
100         '2': {'R': {'5', '7'}, 'N': {'1', '6', '3'}},
101         '3': {'R': {'2', '7', '4'}, 'N': {'6', '8'}},
102         '4': {'R': '7', 'N': {'3', '8'}},
103         '5': {'R': {'2', '10'}, 'N': {'1', '6', '9'}},
104         '6': {'R': {'2', '5', '7', '10'}, 'N': {'1', '3', '9', '11'}},
105         '7': {'R': {'2', '4', '10', '12'}, 'N': {'3', '6', '8', '11'}},
106         '8': {'R': {'4', '7', '12'}, 'N': {'3', '11'}},
107         '9': {'R': {'5', '10', '13'}, 'N': {'6', '14'}},
108         '10': {'R': {'5', '7', '13', '15'}, 'N': {'6', '9', '11', '14'}},
109         '11': {'R': {'7', '10', '12', '15'}, 'N': {'6', '8', '14', '16'}},
110         '12': {'R': {'7', '15'}, 'N': {'8', '11', '16'}},
111         '13': {'R': '10', 'N': {'9', '14'}},
112         '14': {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
113         '15': {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
114         '16': {'R': {'12', '15'}, 'N': '11'}
115     }
116
117     if x == "1":
118         print("caca")
119         archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\
120             Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\

```



```
        output\\ruta_blanca.txt'
119     # Abre el archivo en modo de lectura
120     archivo = open(archivo_ruta, "r")
121
122     # Lee una l nea del archivo
123     ruta_evaluar = archivo.readline()
124
125     # Imprime la l nea le da
126     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\recorridos_blanca.txt'
127     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\arbol_blanca.dot'
128     transiciones_posibles = []
129     recorrer_estados_blanca(tablaEstados, ruta_evaluar
        )
130     generar_arbol_recorridos(tablaEstados,
        ruta_evaluar, archivo_recorridos,
        archivo_salida)
131
132 else:
133     #Redenrizmos ruta blanca
134     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\ruta_blanca.txt'
135     # Abre el archivo en modo de lectura
136     archivo = open(archivo_ruta, "r")
137
138     # Lee una l nea del archivo
139     ruta_evaluar = archivo.readline()
140
141     # Imprime la l nea le da
142     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\recorridos_blanca.txt'
143     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
        output\\arbol_blanca.dot'
144     transiciones_posibles = []
```

```
145     recorrer_estados_blanca(tablaEstados, ruta_evaluar
146     )
147     generar_arbol_recorridos(tablaEstados,
148     ruta_evaluar, archivo_recorridos,
149     archivo_salida)
150
151     #Renderizamos ruta negra
152     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\
153     Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
154     output\\ruta_negra.txt'
155     # Abre el archivo en modo de lectura
156     archivo = open(archivo_ruta, "r")
157
158     # Lee una l nea del archivo
159     ruta_evaluar = archivo.readline()
160
161     # Imprime la l nea le da
162     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
163     Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
164     output\\recorridos_negra.txt'
165     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\
166     Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\
167     output\\arbol_negra.dot'
168     transiciones_posibles = []
169     recorrer_estados_negra(tablaEstados, ruta_evaluar)
170     generar_arbol_recorridos(tablaEstados,
171     ruta_evaluar, archivo_recorridos,
172     archivo_salida)
```

Figura 1.6: Graficador_{grande.py}.

Capítulo 2

Análisis de Resultados

2.1. Capturas del programa en ejecución

A continuación se presenta en orden el proceso de ejecución del programa, donde primeramente se muestra el código en ejecución con un ejemplo de 1 pieza aleatoria, luego de este ejemplo, elegiremos la opción de recorrido para 2 piezas manuales.

1. Iniciamos el programa, donde nos pide que introduzcamos la cantidad de piezas a invocar en el tablero. Para este caso en particular se seleccionó 1 pieza. Ahora nos pide que elegíamos si la pieza generara su recorrido de forma aleatoria o manual, para este caso se seleccionó la opción aleatoria. Luego nos muestra datos sobre la ruta generada aleatoriamente, su respectivo tamaño y el recorrido que se seleccionó de los recorridos_finales posibles. Finalmente, nos indica que veamos la parte gráfica. Observar la Figura 2.1.

```
soyco@ConnorLapX MINGW64 ~/OneDrive/Documents
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/Users/soyco/OneDrive/Documents/ESCOM/sem4/Teoria/P2/Chess/main.py
Ingrese la cantidad de piezas a invocar (mínimo 1, máximo 2)
--> 1

Cantidad de piezas seleccionadas: 1
Seleccione una opción:
1. Generar recorrido aleatoriamente.
2. Ingresar recorrido manualmente.
Opción: 1
pygame 2.4.0 (SDL 2.26.4, Python 3.10.5)
Hello from the pygame community. https://www.pygame.org/contribute.html

Tamano de cadena escogido aleatoriamente [10]
Tambien se generaran las transiciones R y N aleatoriamente.

La cadena o ruta a seguir escogida aleatoriamente sera: RNRNRNRNRR

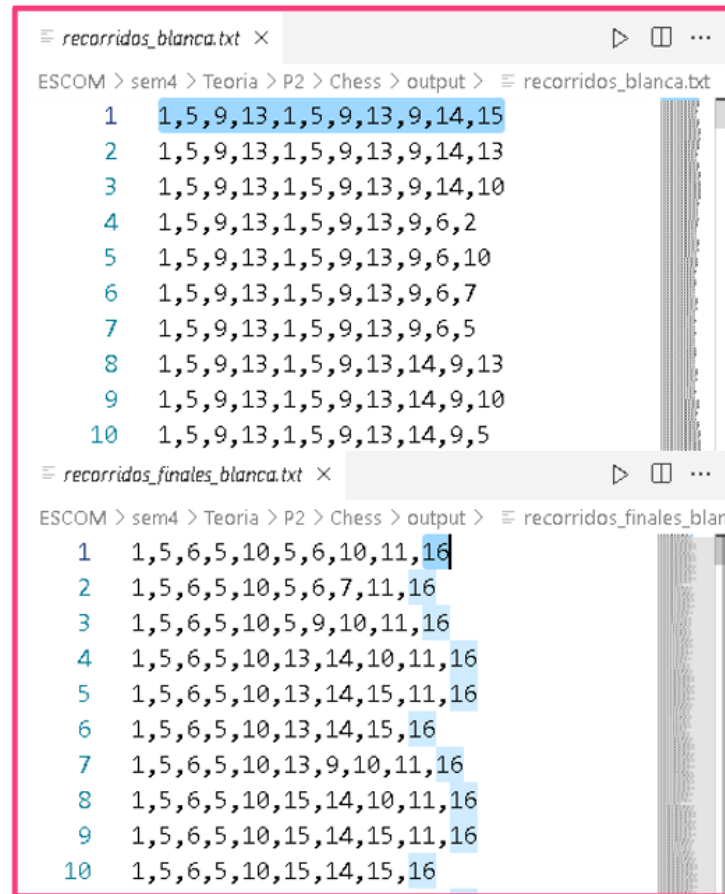
En este punto el programa almaceno en la salidas los recorridos posibles y los recorridos exitosos en la carpeta output.

Recorrido seleccionado: 1,5,6,7,4,7,8,12,16

Vea la parte grafica.
█
```

Figura 2.1: Inicio del programa en terminal.

2. Aquí podemos ver la salida del archivo recorrido_blanca.txt y el de recorrido_finales_blanca.txt donde en el primero se almacenaron todos los recorridos posibles y en el segundo solo los recorridos que llevan al estado de éxito 16. Observar la Figura 2.2.



The image shows two text files side-by-side. The top file, 'recorridos_blanca.txt', contains 10 numbered lines of move sequences. The bottom file, 'recorridos_finales_blanca.txt', also contains 10 numbered lines of move sequences. In both files, the final number of each sequence is highlighted in blue.

```
recorridos_blanca.txt
ESCOM > sem4 > Teoria > P2 > Chess > output > recorrido_blanca.txt
1 1,5,9,13,1,5,9,13,9,14,15
2 1,5,9,13,1,5,9,13,9,14,13
3 1,5,9,13,1,5,9,13,9,14,10
4 1,5,9,13,1,5,9,13,9,6,2
5 1,5,9,13,1,5,9,13,9,6,10
6 1,5,9,13,1,5,9,13,9,6,7
7 1,5,9,13,1,5,9,13,9,6,5
8 1,5,9,13,1,5,9,13,14,9,13
9 1,5,9,13,1,5,9,13,14,9,10
10 1,5,9,13,1,5,9,13,14,9,5

recorridos_finales_blanca.txt
ESCOM > sem4 > Teoria > P2 > Chess > output > recorrido_finales_blan
1 1,5,6,5,10,5,6,10,11,16
2 1,5,6,5,10,5,6,7,11,16
3 1,5,6,5,10,5,9,10,11,16
4 1,5,6,5,10,13,14,10,11,16
5 1,5,6,5,10,13,14,15,11,16
6 1,5,6,5,10,13,14,15,16
7 1,5,6,5,10,13,9,10,11,16
8 1,5,6,5,10,15,14,10,11,16
9 1,5,6,5,10,15,14,15,11,16
10 1,5,6,5,10,15,14,15,16
```

Figura 2.2: Vista de archivos de salida de recorridos.

3. Aquí se puede ver la secuencia que sigue la parte gráfica, donde al dar clic al botón "Siguiente" nos lleva al siguiente movimiento. Observar la Figura 2.3.

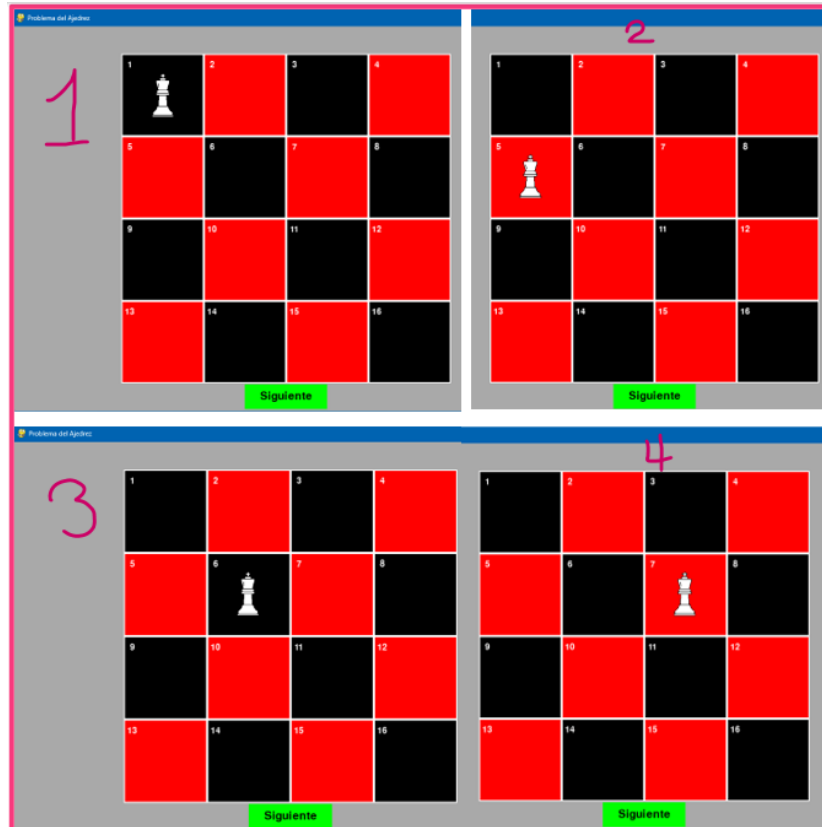


Figura 2.3: Visualización de transiciones gráficas.

4. Siguiendo imagen de transiciones graficas. Observar la Figura 2.4.

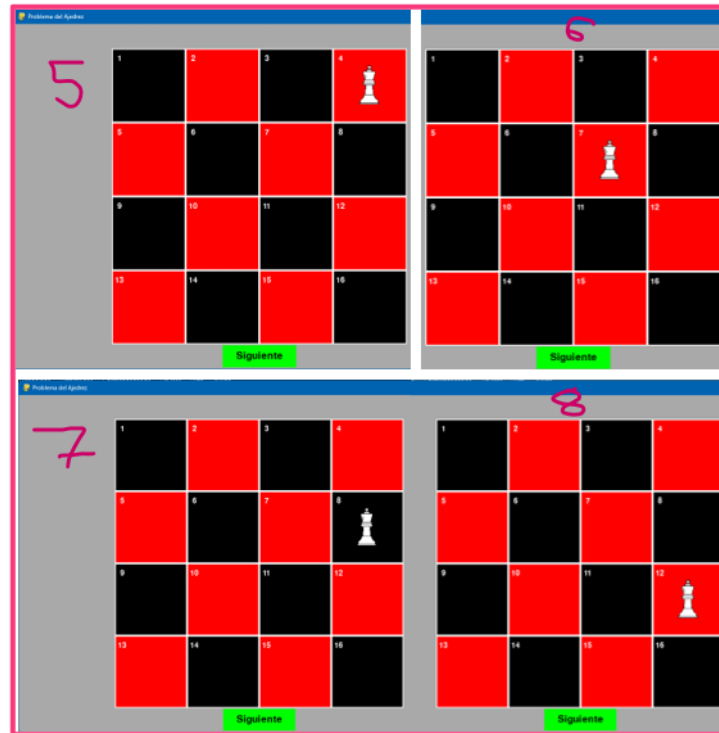


Figura 2.4: Transiciones graficas.

5. Final de la transición gráfica, el programa nos dice que ha terminado. Observar la figura 2.5.

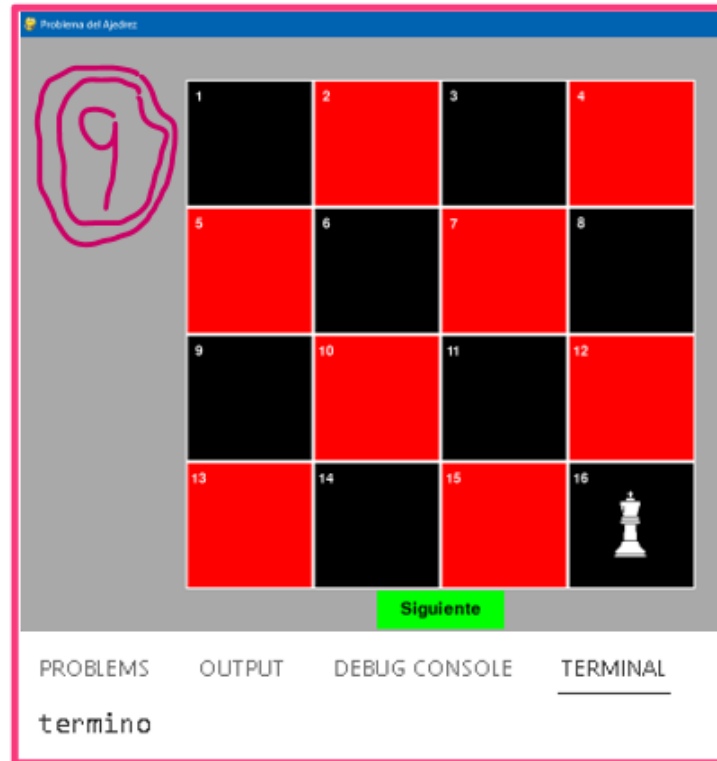


Figura 2.5: Fin de transición y final en terminal.

6. Aquí podemos ver el árbol de recorridos correspondiente al recorrido que se ha escogido aleatoriamente. Observar la figura 2.6.

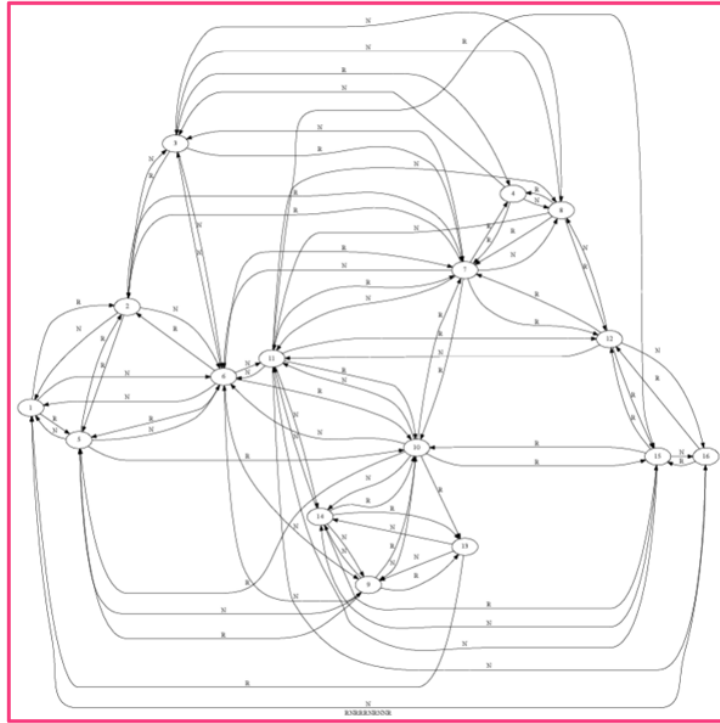


Figura 2.6: Grafo de recorrido.

7. Iniciamos el programa nuevamente, donde nos pide que introduzcamos la cantidad de piezas a invocar en el tablero. Para este caso en particular se seleccionó 2 piezas. Ahora nos pide que elegíamos entre diferentes configuraciones, elegimos el caso de manual vs manual, que básicamente nosotros introduciremos el recorrido a recorrer. Luego nos muestra datos sobre las rutas seleccionadas aleatoriamente, su respectivo tamaño. Finalmente, nos indica que pieza empieza primero. Podemos observar que la pieza negra colisionará con la pieza blanca cuando esta se recorra al estado 6. Observar la figura 2.7.

```
$ C:/Users/soyco/AppData/Local/Programs/Python/Python310/python.exe c:/Users/soyco/OneDrive/Documents/ESCOM/sem4/Teoria/P2/Chess/main.py
Ingrese la cantidad de piezas a invocar (mínimo 1, máximo 2)
--> 2

Cantidad de piezas seleccionadas: 2
Seleccione una opción:
1. Aleatorio vs Aleatorio.
2. Manual vs Aleatorio.
3. Manual vs Manual.
Opción: 3
pygame 2.4.0 (SDL 2.26.4, Python 3.10.5)
Hello from the pygame community. https://www.pygame.org/contribute.html
Establezcan presencialmente quien sera la pieza blanca o negra.

--PARA PIEZAS BLANCA--
Ingrese la cadena usando las transiciones R y N.
NNNN

Tamaño de cadena escogido [4].

La cadena o ruta a seguir escogida sera: NNNN

--PARA CADENA NEGRA--
Ingrese la cadena usando las transiciones R y N.
NNRR

Tamaño de cadena escogido [4].

La cadena o ruta a seguir escogida sera: NNRR

En este punto el programa almaceno en la salidas los recorridos posibles y los recorridos exitosos en la carpeta output.

Recorrido seleccionado para blanca: 1,6,11,16
Recorrido seleccionado para negra: 4,7,6,10,13

Sacan piezas negras.
```

Figura 2.7: Vista de terminal de segundo caso.

8. Pasos de transiciones de la parte gráfica, donde en el último caso de transición se detecta la colisión y se recalcula una ruta. Observar figura 2.8.

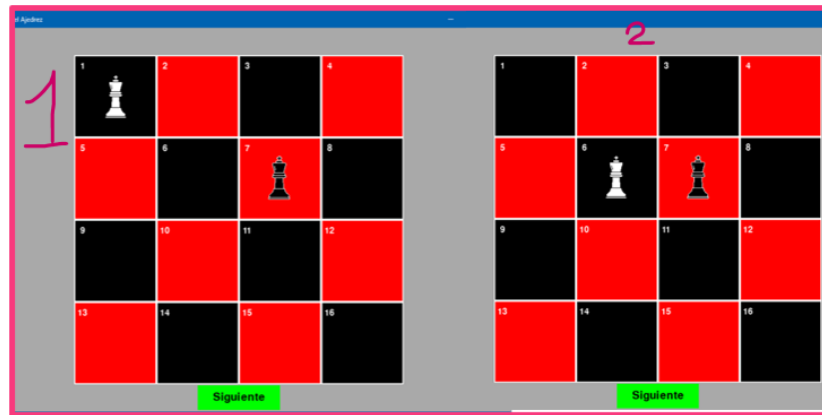


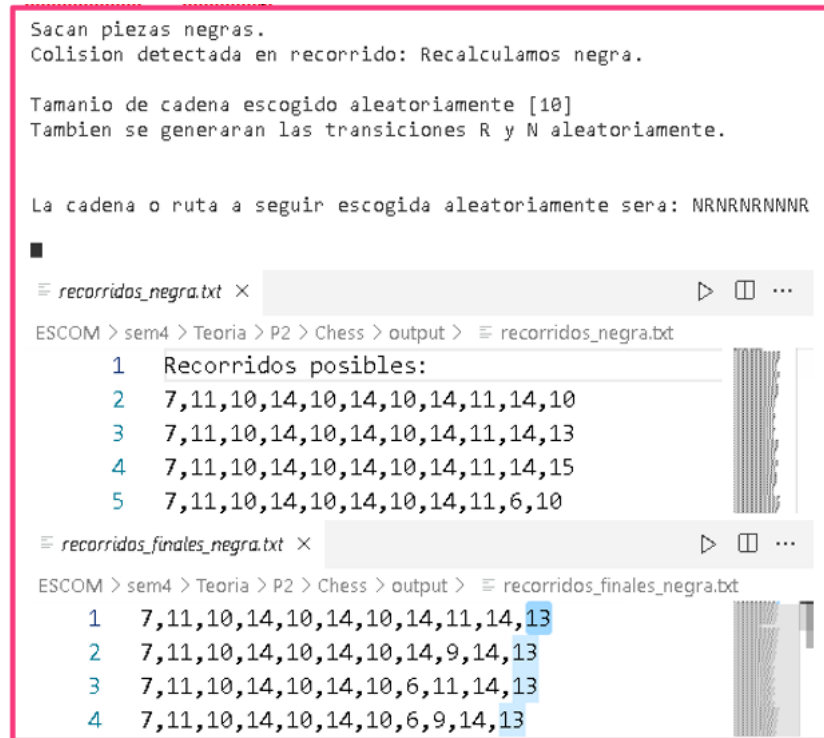
Figura 2.8: Secuencia de transiciones, donde en caso 2 hay colisión.

9. Se detecta una colisión, por ende se reconfigura el recorrido y por ende los recorridos posibles y los recorridos finales, por lo que se vuelve a escoger una ruta adecuada. Observar figura 2.9.

```
Sacan piezas negras.
Colision detectada en recorrido: Recalculamos negra.

Tamano de cadena escogido aleatoriamente [10]
Tambien se generaran las transiciones R y N aleatoriamente.

La cadena o ruta a seguir escogida aleatoriamente sera: NRNRNRNNNR
```



```
ESCROM > sem4 > Teoria > P2 > Chess > output > recorridos_negra.txt
1  Recorridos posibles:
2  7,11,10,14,10,14,10,14,11,14,10
3  7,11,10,14,10,14,10,14,11,14,13
4  7,11,10,14,10,14,10,14,11,14,15
5  7,11,10,14,10,14,10,14,11,6,10
```

```
ESCROM > sem4 > Teoria > P2 > Chess > output > recorridos_finales_negra.txt
1  7,11,10,14,10,14,10,14,11,14,13
2  7,11,10,14,10,14,10,14,9,14,13
3  7,11,10,14,10,14,10,6,11,14,13
4  7,11,10,14,10,14,10,6,9,14,13
```

Figura 2.9: Vista de terminal de nueva ruta configurada y salida en archivos.

10. Regresamos a la parte gráfica donde ahora si nos reconfiguró una ruta para la pieza negra, donde podemos ver que la pieza blanca terminó por llegar primero al estado 16 y el juego se acaba. Observar figura 2.10.

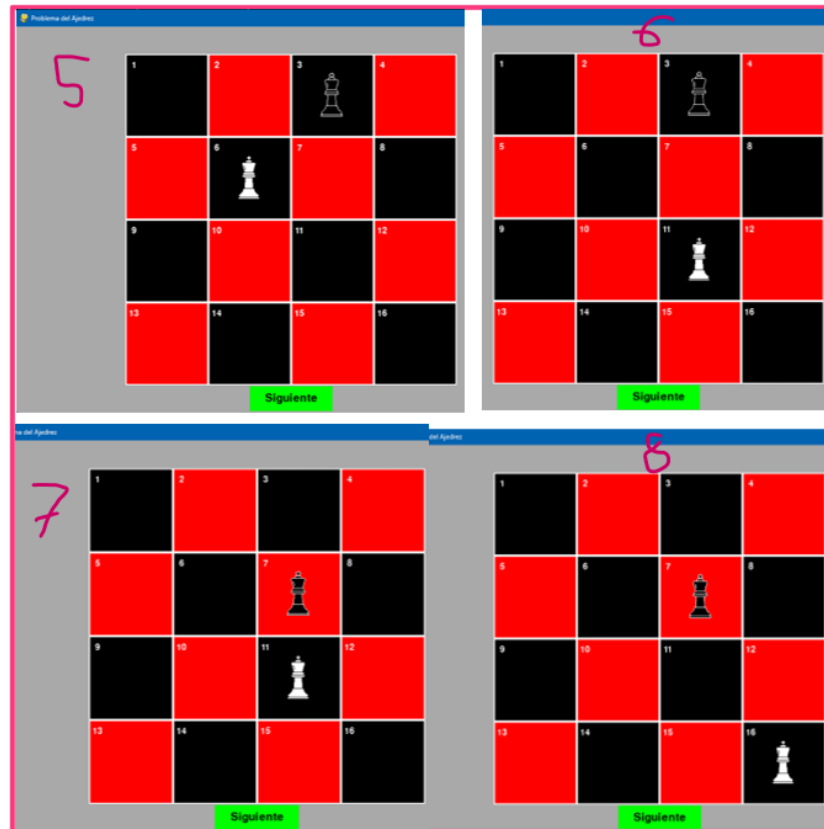


Figura 2.10: Vista de gráfica del ajedrez.

11. Aquí podemos ver la renderización del árbol o grafo de recorridos para pieza blanca. Observar la figura 2.11.

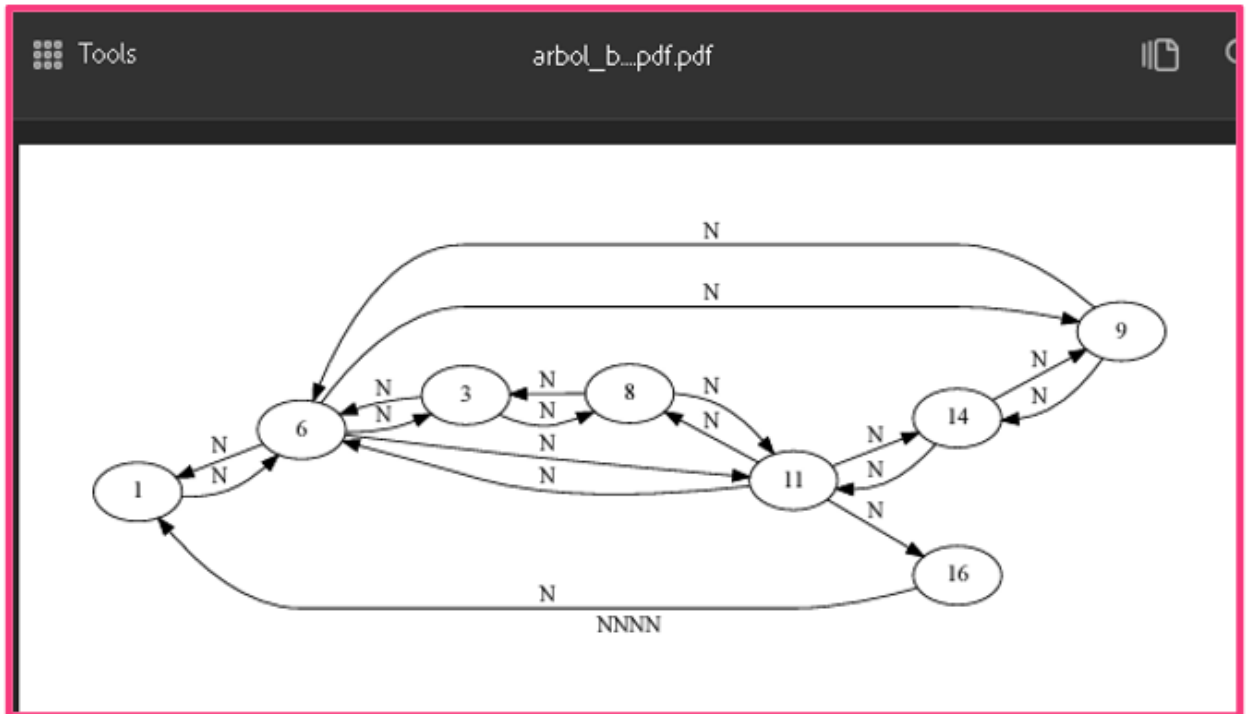


Figura 2.11: Árbol de recorridos para pieza blanca.

12. Aquí podemos ver la renderización del árbol o grafo de recorridos para pieza negra. Observar la figura 2.12.

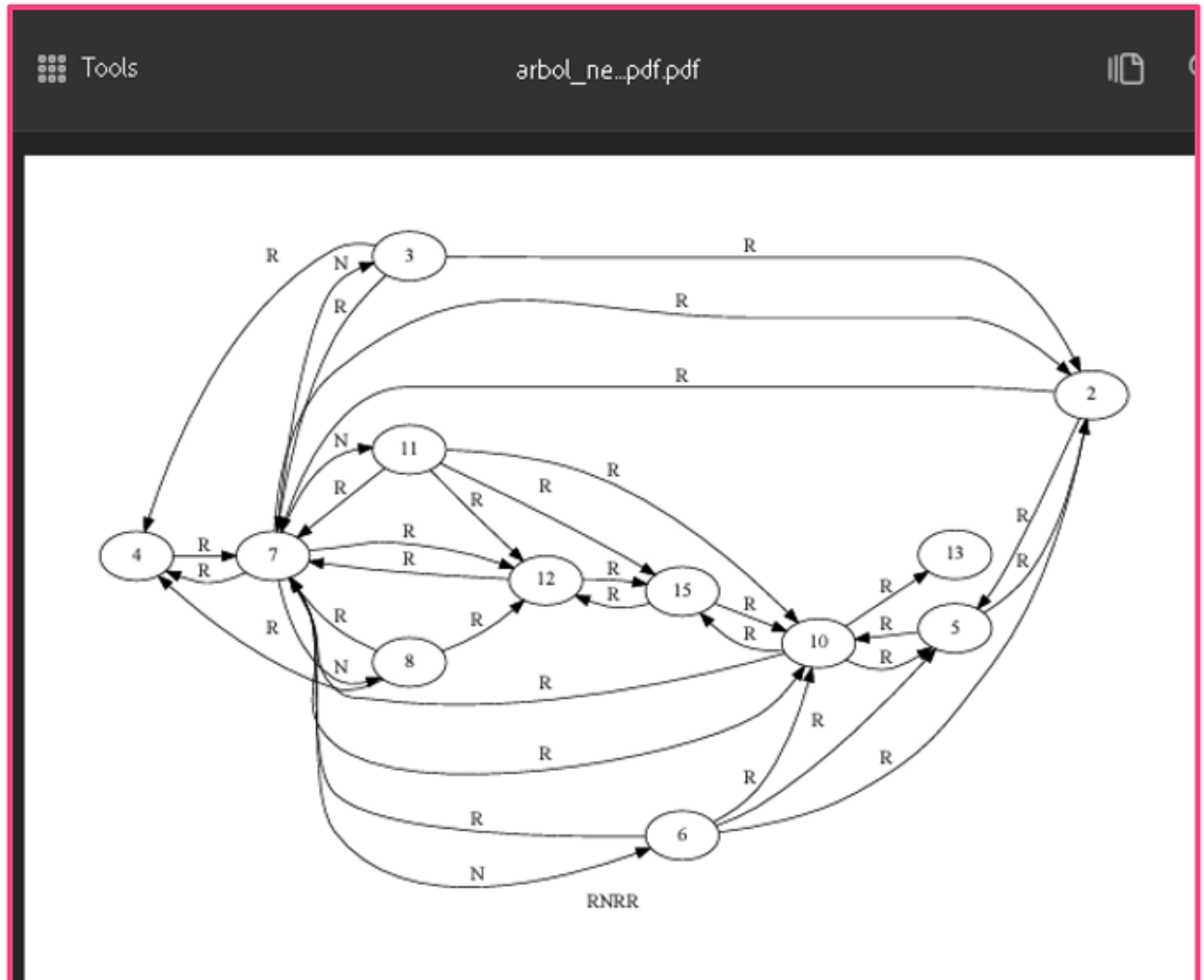
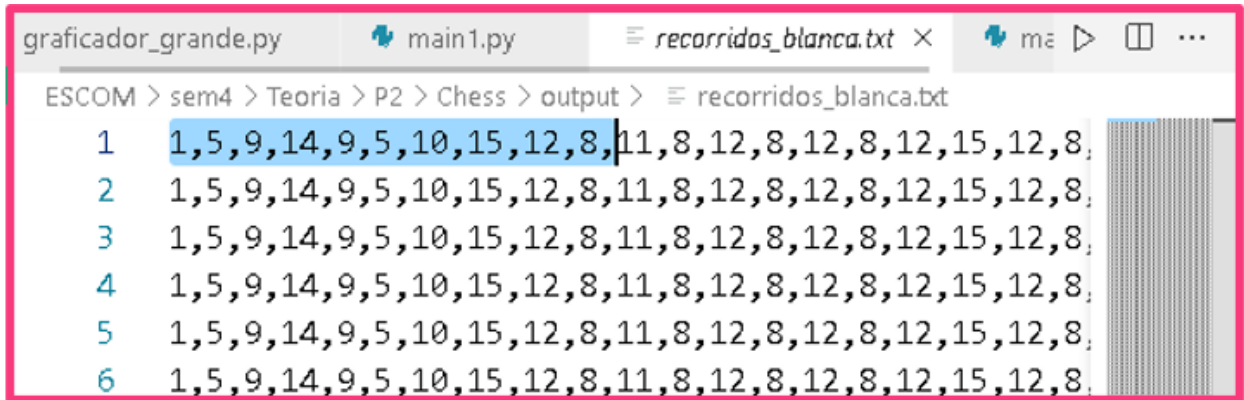


Figura 2.12: Árbol de recorridos para pieza negra.

14. Aquí podemos ver el archivo de recorridos posibles para la pieza, donde el formato de guardado del recorrido se almacena de manera muy distinta, siendo separado cada recorrido por puntos en vez de saltos de línea, para de esta manera evitar una creciente exponencial en la generación del documento. Observar la figura 2.12.



```
graficador_grande.py  main1.py  recorridos_blanca.txt  ma  ...
ESCOM > sem4 > Teoria > P2 > Chess > output > recorridos_blanca.txt
1  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
2  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
3  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
4  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
5  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
6  1,5,9,14,9,5,10,15,12,8,11,8,12,8,12,8,12,15,12,8,
```

Figura 2.14: Vista de archivo de recorridos para la pieza blanca.

15. Grafo de recorridos para la cadena de tamaño 82. Observar la figura 2.12.

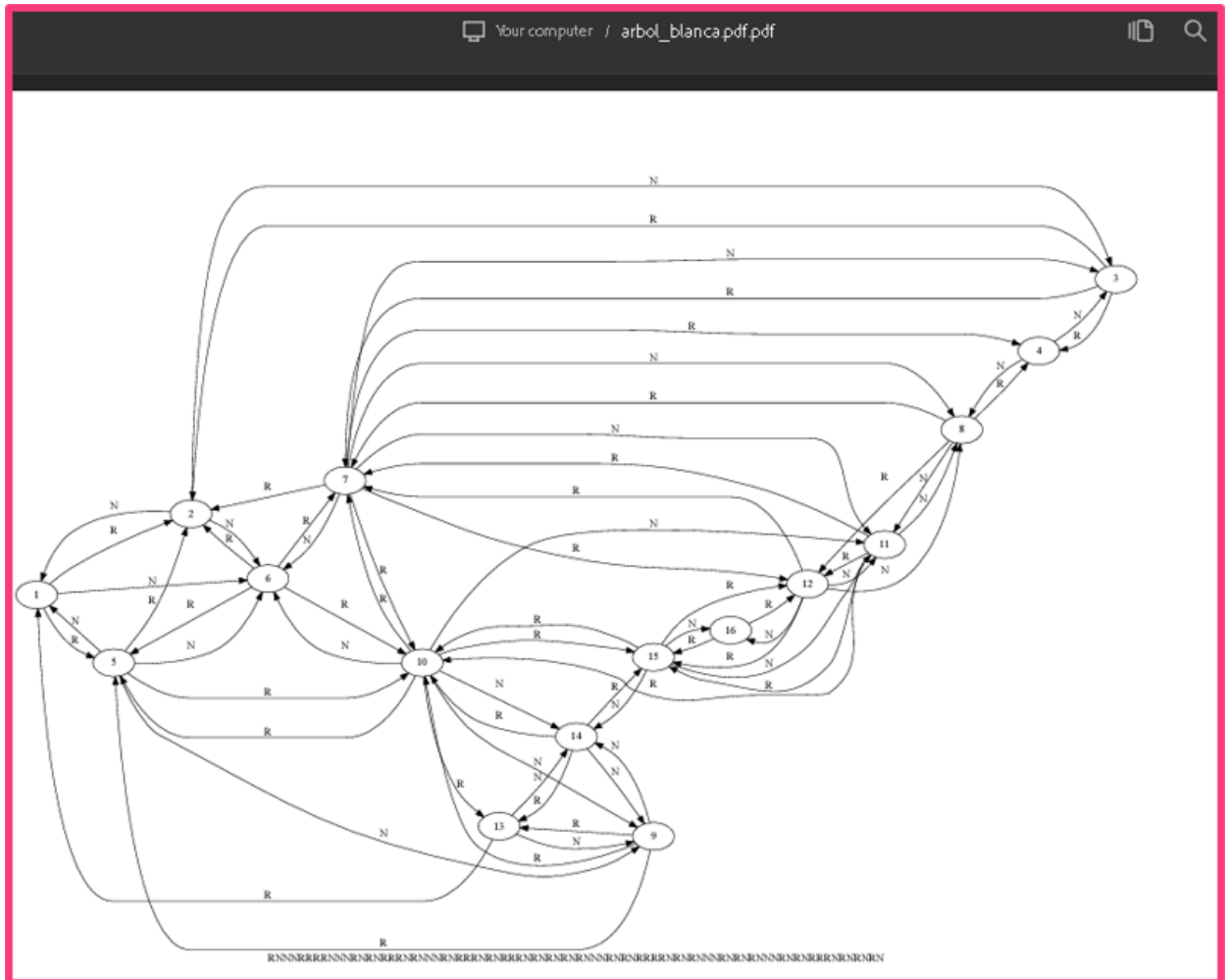


Figura 2.15: Árbol de recorridos para pieza en cuestión.

Capítulo 3

Conclusión

En primer lugar, he aprendido la importancia de la planificación y el diseño adecuado antes de comenzar a implementar una solución. Al enfrentar el desafío de representar un tablero de ajedrez en código, me di cuenta de que debía considerar cuidadosamente la estructura de datos y la forma en que se organizarían las piezas. Esto me enseñó a pensar en forma modular y a descomponer problemas complejos en partes más pequeñas y manejables.

Además, esta práctica me brindó una oportunidad valiosa para aplicar mis conocimientos de programación orientada a objetos (POO). Pude utilizar clases y objetos para representar las piezas de ajedrez y manipular sus atributos y comportamientos. A través de esta experiencia, comprendí mejor los conceptos fundamentales de la POO, como la encapsulación, la herencia y el polimorfismo.

También aprendí sobre la importancia de la abstracción y la claridad en el código. Al escribir el programa, me esforcé por utilizar nombres de variables y métodos descriptivos, lo que facilitó la comprensión y el mantenimiento del código en el futuro. Además, traté de evitar la duplicación de código y promoví la reutilización a través de la creación de funciones y métodos reutilizables.

Otra conclusión clave es el valor de las pruebas y la depuración. Durante el proceso de desarrollo, me encontré con algunos errores y comportamientos inesperados. Aprendí a utilizar técnicas de depuración, como la impresión de mensajes y la revisión del flujo de ejecución, para identificar y corregir estos problemas. También descubrí la importancia de escribir pruebas unitarias para validar el funcionamiento de mi código y garantizar su correcta ejecución en diferentes

escenarios.

Por último, esta práctica me recordó la importancia de la paciencia y la perseverancia al enfrentar desafíos de programación. Hubo momentos en los que me sentí frustrado y bloqueado, pero aprendí a tomar descansos, buscar ayuda cuando fuera necesario y abordar los problemas paso a paso. Esto me enseñó a ser más resiliente y a mantener una mentalidad de resolución de problemas.

Probé mi solución en varios casos de prueba y obtuvimos resultados precisos y eficientes. Sin embargo, también noté que la complejidad temporal de mi solución puede aumentar considerablemente en laberintos grandes con muchas palabras objetivo. En futuros trabajos, podría explorar otras técnicas de búsqueda que puedan ser más eficientes en casos de laberintos grandes y complejos.

3.1. Problemas iniciales

Durante la resolución de esta práctica del tablero de ajedrez, me encontré con varios problemas iniciales que debieron ser abordados para lograr una solución completa.

En primer lugar, tuve que definir la estructura y el diseño del programa, decidiendo cómo organizar el código en archivos separados y estableciendo la interacción con el usuario a través de menús. Esto implicó considerar cómo generar y mostrar los recorridos de las piezas.

Otro desafío importante fue la generación de los recorridos en sí. Cada pieza tiene reglas de movimiento específicas, por lo que fue necesario desarrollar algoritmos precisos y eficientes que tuvieran en cuenta posibles casos especiales, como los límites del tablero y la colisión con otras piezas.

Una vez generados los recorridos, fue crucial validar su validez. Esto significaba asegurarse de que las piezas no se salieran del tablero y que no se superpusieran entre sí. Para lograr esto, implementé una lógica adicional para verificar cada movimiento y descartar los recorridos inválidos.

Además, era importante representar visualmente los recorridos en el tablero de ajedrez. Esto requirió el uso de bibliotecas gráficas y la definición de la lógica para dibujar las piezas y las rutas de movimiento en la interfaz gráfica.

Por último, consideré la eficiencia y el rendimiento del programa. Algunos recorridos, como los mayores a 10, crecían exponencialmente, lo que generaba inconvenientes adversos.

3.1.1. Soluciones

A continuación, se detallan las soluciones que se les dieron a dichos problemas:

1. Estructura y diseño del programa: Se optó por organizar el código en archivos separados para mejorar la modularidad y la legibilidad. Se crearon funciones de menú para interactuar con el usuario y seleccionar la cantidad de piezas y el tipo de recorrido. Además, se utilizó un enfoque basado en subprocessos para invocar los diferentes archivos Python según las selecciones del usuario.
2. Generación de recorridos: Se implementaron algoritmos específicos para cada tipo de pieza (blanca y negra) que tuvieron en cuenta las reglas de movimiento de cada una. Se consideraron casos especiales, como la colisión con otras piezas, para generar recorridos válidos.
3. Validación de recorridos: Se agregó lógica adicional para verificar la validez de los recorridos generados. En caso de detectar movimientos inválidos, se descartaron o ajustaron los recorridos correspondientes.
4. Representación gráfica: Se utilizó una biblioteca gráfica, como Pygame, para mostrar visualmente los recorridos en el tablero de ajedrez. Se definió la lógica para dibujar las piezas y las rutas de movimiento en la interfaz gráfica, lo que permitió una representación clara y comprensible de los recorridos generados.

5. Rendimiento y eficiencia: Se optimizó el código y los algoritmos para mejorar el rendimiento del programa, especialmente en casos donde las piezas tenían un gran número de movimientos posibles, como los recorridos mayores a 10, donde la solución fue un correcto almacenamiento de los recorridos y también su lectura. Se utilizaron técnicas de programación eficiente y se realizaron ajustes para evitar demoras excesivas en la generación y visualización de los recorridos.

3.2. Complejidades

El algoritmo también incluye una función llamada "recorrer_estados" que genera todos los recorridos posibles en el juego del ajedrez. Esta función utiliza recursión para obtener los recorridos y es difícil determinar su complejidad exacta sin conocer la longitud máxima de los recorridos y el número de estados posibles. En el peor de los casos, la complejidad podría ser exponencial.

En cuanto a la función "seleccionar_recorrido", selecciona una línea aleatoria de un archivo que contiene los recorridos finales válidos. La complejidad de esta función depende del tamaño del archivo y es lineal en función del número de líneas en el archivo.

Básicamente, la complejidad del algoritmo del tablero de la práctica puede variar dependiendo de los detalles específicos y la longitud de los recorridos en el juego del ajedrez, sobretodo para recorridos como los mayores a 10, que crecen de manera exponencial.

Capítulo 4

Bibliografías

1. Salamanca, S., García, J. (2019). Estrategias de juego y resolución de problemas en el ajedrez. *Revista Internacional de Ajedrez y Educación*, 42(2), 78-92.
2. Johnson, M. (2020). *The Art of Chess: Tactics and Problem Solving*. New York: Chess Publishing.
3. Williams, D. (2018). *Mastering Chess: The Guide to Chess Tactics, Chess Openings, and Chess Strategies*. New York: New Chess Press.
4. Sánchez, L. M. (2017). El tablero de ajedrez como herramienta para el desarrollo del pensamiento lógico-matemático. *Revista de Investigación en Educación Matemática*, 9(2), 145-160.

Capítulo 5

Anexos

5.1. LATEX de este proyecto

Dirección Overleaf: <https://www.overleaf.com/3411646862zgynbbsnckcd>
Dirección GitHub: <https://github.com/Connor-UM-18/Teoria-Computacional---Tablero.git>

5.2. Main.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1 #Teoria de la computacion
2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4 import os
5 import subprocess
6
7 def menu_piezas():
8     cantidad = int(input("Ingrese la cantidad de piezas a
9         invocar (m nimo 1, m ximo 2)\n--> "))
10     if cantidad < 1 or cantidad > 2:
```



```
10         print("\nCantidad inv lida. Por favor, ingrese 1 o
           2.")
11         return menu_piezas() # Llamada recursiva si la
           cantidad es inv lida
12     else:
13         return cantidad
14
15 def menu_recorrido(cantidad_piezas):
16     #config==1 es para recorrido aleatorio
17     #config==2 es para recorrido manual
18     if cantidad_piezas==1:
19         #1 pieza
20         opcion = int(input("Seleccione una opci n:\n1.
           Generar recorrido aleatoriamente.\n2. Ingresar
           recorrido manualmente.\nOpci n: "))
21         if opcion == 1:
22             return 1 # Opci n de generar recorrido
           aleatoriamente
23         elif opcion == 2:
24             return 2 # Opci n de ingresar recorrido
           manualmente
25         else:
26             print("Opci n inv lida. Por favor, seleccione
           1 o 2.")
27             return menu_recorrido(cantidad_piezas) #
           Llamada recursiva si la opci n es inv lida
28     else:
29         #2 piezas
30         opcion = int(input("Seleccione una opci n:\n1.
           Aleatorio vs Aleatorio.\n2. Manual vs Aleatorio
           .\n3. Manual vs Manual.\nOpci n: "))
31         if opcion == 1:
32             return 3 # Opci n de generar recorrido
           aleatoriamente vs aleatoriamente
33         elif opcion == 2:
34             return 4 # Opci n de ingresar recorrido
           manualmente vs aleatoriamente
35         elif opcion == 3:
36             return 5 # Opci n de ingresar recorrido
           manualmente vs manualmente
```

```
37         else:
38             print("Opci n inv lida. Por favor, seleccione
              1, 2 o 3.")
39             return menu_recorrido(cantidad_piezas) #
              Llamada recursiva si la opci n es inv lida
40
41 #Main del ciclo del juego
42 # Obtener directorio actual
43 directorio_actual = os.path.dirname(os.path.abspath(
    __file__))
44 cantidad_piezas = menu_piezas()
45 print("\nCantidad de piezas seleccionadas:",
    cantidad_piezas)
46 config = menu_recorrido(cantidad_piezas)
47 x=0
48 #Posibles configuraciones: 1,2,3,4 o 5.
49 if config == 1:#A
50     archivo_py = os.path.join(directorio_actual, "main1.py"
    )
51     resultado = subprocess.run(["python", archivo_py])
52     os.system('cls' if os.name == 'nt' else 'clear')
53     codigo_salida = resultado.returncode
54     x=1
55     #print(codigo_salida)
56 elif config == 2:#Manual
57     archivo_py = os.path.join(directorio_actual, "main2.py"
    )
58     resultado = subprocess.run(["python", archivo_py])
59     os.system('cls' if os.name == 'nt' else 'clear')
60     codigo_salida = resultado.returncode
61     x=1
62     #print(codigo_salida)
63 elif config == 3:#AvsA
64     archivo_py = os.path.join(directorio_actual, "main3.py"
    )
65     resultado =subprocess.run(["python", archivo_py])
66     os.system('cls' if os.name == 'nt' else 'clear')
67     codigo_salida = resultado.returncode
68     x=2
69     #print(codigo_salida)
```

```
70 elif config == 4: #ManualvsA
71     archivo_py = os.path.join(directorio_actual, "main4.py"
72                                )
73     resultado = subprocess.run(["python", archivo_py])
74     os.system('cls' if os.name == 'nt' else 'clear')
75     codigo_salida = resultado.returncode
76     x=2
77     #print(codigo_salida)
78 elif config == 5: #ManualvsManual
79     archivo_py = os.path.join(directorio_actual, "main5.py"
80                                )
81     resultado = subprocess.run(["python", archivo_py])
82     os.system('cls' if os.name == 'nt' else 'clear')
83     codigo_salida = resultado.returncode
84     x=2
85     #print(codigo_salida)
86 if codigo_salida == 3:
87     archivo_py = os.path.join(directorio_actual, "
88                                graficador_grande.py")
89     subprocess.run(["python", archivo_py, (str(x))])
90 else:
91     archivo_py = os.path.join(directorio_actual, "
92                                graficador.py")
93     subprocess.run(["python", archivo_py, (str(x))])
94 print("termino")
```

5.3. Main1.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1 #Teoria de la computacion
2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
```

```

4
5 import random
6 import pygame
7 import random
8 import os
9
10 pygame.init() #Acceso al paquete pygame
11 #Ancho
12 WIDTH = 1000
13 #Altura
14 HEIGHT = 700
15 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamano
    de ventana a imprimir
16 pygame.display.set_caption('Problema del Ajedrez')
17 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
18 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo de
    fuente 2 del juego
19 timer = pygame.time.Clock()#velocidad de actualizacion de
    nuestro juego a 60 fps
20 fps=60
21
22 #NFA de estados
23 tablaEstados = {
24     '1' : {'R': {'2','5'}, 'N': '6'},
25     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
26     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
27     '4' : {'R': '7', 'N': {'3','8'}},
28     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
29     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9','11'}
30         },
31     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8','11'}
32         },
33     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
34     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
35     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9','11','
36         14'}},
37     '11' : {'R': {'7','10','12','15'}, 'N': {'6','8','14','
38         16'}},
39     '12' : {'R': {'7','15'}, 'N': {'8','11','16'}},

```

```

36     '13' : {'R': '10', 'N': {'9', '14'}},
37     '14' : {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
38     '15' : {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
39     '16' : {'R': {'12', '15'}, 'N': '11'}#Estado 16 es el
        estado Final.
40 }
41
42 #Variables e imagenes del juego
43 pieza_blanca = ['king']
44 posicion_blanca = [(235, 85)]
45
46 #Variables de turnos cambiantes
47 turn_step = 0
48 selection = 100
49 #Cargar imagenes en juego
50 rey_blanco = pygame.image.load('C:\\Users\\soyco\\OneDrive
        \\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
        white_king.png')
51 rey_blanco = pygame.transform.scale(rey_blanco, (80, 80))
52
53 imagen_blanca = [rey_blanco]
54
55 lista_piezas = ['king']
56
57 #ver variables/contador flash
58 boton_presionado = False
59
60 def dibujar_boton():
61     boton_width = 150
62     boton_height = 45
63     boton_x = (WIDTH - boton_width) // 2
64     boton_y = (HEIGHT - boton_height - 20) + 17
65
66     # Dibujar el boton como un rectangulo en la pantalla
67     boton_rect = pygame.Rect(boton_x, boton_y, boton_width,
        boton_height)
68     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
        , boton_width, boton_height))
69     texto = font.render("Siguiente", True, (0, 0, 0))
70     texto_rect = texto.get_rect(center=(boton_x +

```

```

    boton_width // 2, boton_y + boton_height // 2))
71     screen.blit(texto, texto_rect)
72     return boton_rect
73
74 #Funcion para dibujar tablero
75 def dibujar_tablero():
76     cuadro_size = 150 # Tama o de cada cuadro del tablero
77     tablero_width = 4 * cuadro_size # Ancho total del
        tablero
78     tablero_height = 4 * cuadro_size # Altura total del
        tablero
79     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
        para centrar el tablero
80     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
        Y para centrar el tablero
81
82     numero_color = 'white' # Color del n mero de casilla
83
84     font = pygame.font.Font(None, 24) # Fuente y tama o
        del n mero de casilla
85
86     for i in range(16): # Iterar 16 veces para un tablero
        de 4x4
87         columna = i % 4
88         fila = i // 4
89         x = tablero_x + columna * cuadro_size
90         y = tablero_y + fila * cuadro_size
91         if fila % 2 == 0:
92             color = 'black' if columna % 2 == 0 else 'red'
93         else:
94             color = 'red' if columna % 2 == 0 else 'black'
95         pygame.draw.rect(screen, color, [x, y, cuadro_size,
            cuadro_size])
96         pygame.draw.rect(screen, 'white', [x, y,
            cuadro_size, cuadro_size], 2) # Agregar borde
            de color blanco
97         numero_texto = font.render(str(i + 1), True,
            numero_color) # Crear superficie de texto con
            el n mero
98         numero_rect = numero_texto.get_rect(center=(x +

```

```

    cuadro_size // 2)-60, y + 20)) # Posición del
    n mero en la parte superior del recuadro
99     screen.blit(numero_texto, numero_rect) # Pegar el
    n mero en la pantalla
100
101 #Funcion para dibujar piezas
102 def dibujar_piezas():
103     index=lista_piezas.index('king')
104     if pieza_blanca[0]=='king':
105
106         screen.blit(imagen_blanca[index], (posicion_blanca
            [0][0],posicion_blanca[0][1]))
107
108 def recorrer_estados(tabla_estados, cadena):
109     # Función para obtener todos los recorridos posibles
110     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual):
111         if not simbolos_restantes:
112             recorridos.append(recorrido_actual)
113             return
114
115         simbolo = simbolos_restantes[0]
116         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
117             transiciones = tabla_estados[estado_actual][
                simbolo]
118
119             for estado_siguiente in transiciones:
120                 obtener_recorridos(estado_siguiente,
                    simbolos_restantes[1:], recorrido_actual
                        + [estado_siguiente])
121
122     # Función para obtener los recorridos v lidos hasta
        el estado final
123     def obtener_recorridos_finales(estado_actual,
        simbolos_restantes, recorrido_actual):
124         if estado_actual == '16':
125             recorridos_finales.append(recorrido_actual)
126             return
127
```

```
128         if not simbolos_restantes:
129             return
130
131         simbolo = simbolos_restantes[0]
132         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
133             transiciones = tabla_estados[estado_actual][
                simbolo]
134
135             for estado_siguiente in transiciones:
136                 obtener_recorridos_finales(estado_siguiente
                    , simbolos_restantes[1:],
                    recorrido_actual + [estado_siguiente])
137
138     # Obtener recorridos posibles
139     recorridos = []
140     obtener_recorridos('1', cadena, ['1'])
141
142     # Obtener recorridos hasta el estado final
143     recorridos_finales = []
144     obtener_recorridos_finales('1', cadena, ['1'])
145
146     # Guardar los recorridos en archivos de texto
147     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
        \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
        .txt', 'w') as archivo_recorridos:
148         archivo_recorridos.write('Recorridos posibles:\n')
149         for recorrido in recorridos:
150             archivo_recorridos.write(','.join(recorrido) +
                '\n')
151
152     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
        \\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_blanca.txt', 'w') as
        archivo_recorridos_finales:
153         for recorrido in recorridos_finales:
154             archivo_recorridos_finales.write(','.join(
                recorrido) + '\n')
155
156
```



```
157
158 def seleccionar_recorrido():
159     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_blanca.txt"
160
161     with open(ruta_archivo, "r") as archivo:
162         if os.path.getsize(ruta_archivo) == 0:
163             print("No existen soluciones que lleguen al
                estado 16 con la condicion actual. Se
                recalculara una ruta.\n")
164             nuevaruta=cadena()
165             print("La nueva ruta es: "+nuevaruta)
166             recorrer_estados(tablaEstados, nuevaruta)
167             print('Se almacenaron las nuevas salidas de los
                recorridos posibles y los recorridos
                exitosos en la carpeta output.\n')
168
169             return seleccionar_recorrido()
170
171     lineas = archivo.readlines()
172
173     # Seleccionar una linea aleatoria
174     recorrido_seleccionado = random.choice(lineas)
175
176     # Eliminar los espacios en blanco y saltos de
        linea
177     recorrido_seleccionado = recorrido_seleccionado.
        strip()
178
179     return recorrido_seleccionado
180
181 def cadenaRandom(numero): #Genera un string de forma random
182     auxiliar = '' #Variable auxiliar
183     for i in range(numero):
184         x = random.randint(1, 2) #Funcion para generar un
            resultado random de una lista
185         if x % 2 == 0:
186             auxiliar = auxiliar + "R"
187         else:
```

```

188         auxiliar = auxiliar + "N"
189     return auxiliar
190
191 def cadena():
192     numero = random.randint(4,10)
193     print('\nTamaño de cadena escogido aleatoriamente [' +
194           str(numero) + ']\nTambien se generaran las
195           transiciones R y N aleatoriamente.\n')
196     cad = cadenaRandom(numero) #Se genera de forma
197           aleatoria del 1-10
198     print('\nLa cadena o ruta a seguir escogida
199           aleatoriamente sera: ' + cad + '\n')
200     return cad
201
202 def calcular_coordenadas(estado):
203     cuadro_size = 150 # Tamaño de cada cuadro del tablero
204     tablero_width = 4 * cuadro_size # Ancho total del
205           tablero
206     tablero_height = 4 * cuadro_size # Altura total del
207           tablero
208     tablero_x = (WIDTH - tablero_width) // 2 # Posición X
209           para centrar el tablero
210     tablero_y = (HEIGHT - tablero_height) // 2 # Posición
211           Y para centrar el tablero
212     fila = (estado - 1) // 4 # Calcular la fila del estado
213     columna = (estado - 1) % 4 # Calcular la columna del
214           estado
215     x = tablero_x + columna * cuadro_size # Calcular la
216           coordenada X del estado
217     y = tablero_y + fila * cuadro_size # Calcular la
218           coordenada Y del estado
219     return ((x+33), y+33) # Devolver las coordenadas como
220           una tupla
221
222 #Main del ciclo del juego 1
223 ruta=cadena() #Solicitamos la cadena generada aleatoriamente
224
225 .
226
227 recorrer_estados(tablaEstados, ruta)
228 print('\nEn este punto el programa almacena en la salida

```

```
    los recorridos posibles y los recorridos exitosos en la
    carpeta output.\n')
215 recorrido = seleccionar_recorrido() #Este fue el recorrido
    que se escogio de manera aleatoria
216 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
    sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
    as archivo:
217     archivo.write(ruta)
218 print("\nRecorrido seleccionado:", recorrido)
219 print("\nVea la parte grafica.")
220 lista_estados = [int(num) for num in recorrido.split(",")]
221
222 coordenadas=(235,85)
223 nueva_coordenada=(0,0)
224 run=True
225 contador=1
226 while run:
227     timer.tick(fps)
228     screen.fill('dark gray')
229     dibujar_tablero()
230     dibujar_piezas()
231     boton_rect = dibujar_boton()
232
233     for event in pygame.event.get():
234         if event.type == pygame.QUIT:
235             run = False
236
237         if event.type == pygame.MOUSEBUTTONDOWN and event.
            button == 1:
238             mouse_pos = pygame.mouse.get_pos()
239             if boton_rect.collidepoint(mouse_pos): #
                Verificar si se hizo clic en el bot n
240
241                 nueva_coordenada = calcular_coordenadas(
                    lista_estados[contador])
242
243             if turn_step <= 1:
244                 if coordenadas in posicion_blanca:
245                     selection = posicion_blanca.index(
                        coordenadas)
```

```
246         if turn_step == 0:
247             turn_step = 1
248
249         if turn_step==1 and selection != 100:
250             posicion_blanca[0] =
251                 nueva_coordenada
252
253             selection = 100
254             contador += 1
255             coordenadas = nueva_coordenada
256             turn_step=0
257
258     pygame.display.flip()
259
260 pygame.quit()
```

5.4. Main2.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1  #Teoria de la computacion
2  #Buscador de palabras
3  #Alumno: Connor Urbano Mendoza
4
5  import random
6  import pygame
7  import random
8  import os
9  import sys
10
11  pygame.init() #Acceso al paquete pygame
12  #Ancho
13  WIDTH = 1000
```

```

14 #Altura
15 HEIGHT = 700
16 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
17 pygame.display.set_caption('Problema del Ajedrez')
18 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
19 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo de
    fuente 2 del juego
20 timer = pygame.time.Clock()#velocidad de actualizacion de
    nuestro juego a 60 fps
21 fps=60
22
23 #NFA de estados
24 tablaEstados = {
25     '1' : {'R': {'2','5'}, 'N': '6'},
26     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
27     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
28     '4' : {'R': '7', 'N': {'3','8'}},
29     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
30     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9','11'}
        },
31     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8','11'}
        },
32     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
33     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
34     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9','11','
        14'}},
35     '11' : {'R': {'7','10','12','15'}, 'N': {'6','8','14','
        16'}},
36     '12' : {'R': {'7','15'}, 'N': {'8','11','16'}},
37     '13' : {'R': '10', 'N': {'9','14'}},
38     '14' : {'R': {'13','10','15'}, 'N': {'9','11'}},
39     '15' : {'R': {'10','12'}, 'N': {'11','14','16'}},
40     '16' : {'R': {'12','15'}, 'N': '11'}#Estado 16 es el
        estado Final.
41 }
42
43 #Variables e imagenes del juego
44 pieza_blanca = ['king']

```

```

45 posicion_blanca = [(235, 85)]
46
47 #
48 turn_step = 0
49 selection = 100
50 #Cargar imagenes en juego
51 rey_blanco = pygame.image.load('C:\\Users\\soyco\\OneDrive
    \\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
    white_king.png')
52 rey_blanco = pygame.transform.scale(rey_blanco, (80, 80))
53
54 imagen_blanca = [rey_blanco]
55
56 lista_piezas = ['king']
57 #ver variables/contador flash
58
59
60 boton_presionado = False
61
62 def dibujar_boton():
63     boton_width = 150
64     boton_height = 45
65     boton_x = (WIDTH - boton_width) // 2
66     boton_y = (HEIGHT - boton_height - 20) + 17
67
68     # Dibujar el boton como un rectangulo en la pantalla
69     boton_rect = pygame.Rect(boton_x, boton_y, boton_width,
        boton_height)
70     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
        , boton_width, boton_height))
71     texto = font.render("Siguiente", True, (0, 0, 0))
72     texto_rect = texto.get_rect(center=(boton_x +
        boton_width // 2, boton_y + boton_height // 2))
73     screen.blit(texto, texto_rect)
74     return boton_rect
75
76 #Funcion para dibujar tablero
77 def dibujar_tablero():
78     cuadro_size = 150 # Tamaño de cada cuadro del tablero
79     tablero_width = 4 * cuadro_size # Ancho total del

```

```

    tablero
80  tablero_height = 4 * cuadro_size # Altura total del
    tablero
81  tablero_x = (WIDTH - tablero_width) // 2 # Posición X
    para centrar el tablero
82  tablero_y = (HEIGHT - tablero_height) // 2 # Posición
    Y para centrar el tablero
83
84  numero_color = 'white' # Color del número de casilla
85
86  font = pygame.font.Font(None, 24) # Fuente y tamaño
    del número de casilla
87
88  for i in range(16): # Iterar 16 veces para un tablero
    de 4x4
89      columna = i % 4
90      fila = i // 4
91      x = tablero_x + columna * cuadro_size
92      y = tablero_y + fila * cuadro_size
93      if fila % 2 == 0:
94          color = 'black' if columna % 2 == 0 else 'red'
95      else:
96          color = 'red' if columna % 2 == 0 else 'black'
97      pygame.draw.rect(screen, color, [x, y, cuadro_size,
    cuadro_size])
98      pygame.draw.rect(screen, 'white', [x, y,
    cuadro_size, cuadro_size], 2) # Agregar borde
    de color blanco
99      numero_texto = font.render(str(i + 1), True,
    numero_color) # Crear superficie de texto con
    el número
100     numero_rect = numero_texto.get_rect(center=((x +
    cuadro_size // 2)-60, y + 20)) # Posición del
    número en la parte superior del recuadro
101     screen.blit(numero_texto, numero_rect) # Pegar el
    número en la pantalla
102
103 #Función para dibujar piezas
104 def dibujar_piezas():
105     index=lista_piezas.index('king')

```

```
106     if pieza_blanca[0]=='king':
107
108         screen.blit(imagen_blanca[index], (posicion_blanca
109             [0][0], posicion_blanca[0][1]))
110
111 def recorrer_estados(tabla_estados, cadena):
112
113     # Funci n para obtener todos los recorridos posibles
114     def obtener_recorridos(estado_actual,
115         simbolos_restantes, recorrido_actual,
116         archivo_recorridos):
117
118         if not simbolos_restantes:
119             archivo_recorridos.write(','.join(
120                 recorrido_actual) + '\n')
121             return
122
123         simbolo = simbolos_restantes[0]
124         if estado_actual in tabla_estados and simbolo in
125             tabla_estados[estado_actual]:
126             transiciones = tabla_estados[estado_actual][
127                 simbolo]
128
129             for estado_siguiente in transiciones:
130                 obtener_recorridos(
131                     estado_siguiente,
132                     simbolos_restantes[1:],
133                     recorrido_actual + [estado_siguiente],
134                     archivo_recorridos
135                 )
136
137     # Funci n para obtener los recorridos v lidos hasta
138     # el estado final
139     def obtener_recorridos_finales(estado_actual,
140         simbolos_restantes, recorrido_actual,
141         archivo_recorridos_finales):
142
143         if estado_actual == '16':
144             archivo_recorridos_finales.write(','.join(
145                 recorrido_actual) + '\n')
146             return
```



```
136         if not simbolos_restantes:
137             return
138
139         simbolo = simbolos_restantes[0]
140         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
141             transiciones = tabla_estados[estado_actual][
                simbolo]
142
143             for estado_siguiente in transiciones:
144                 obtener_recorridos_finales(
145                     estado_siguiente,
146                     simbolos_restantes[1:],
147                     recorrido_actual + [estado_siguiente],
148                     archivo_recorridos_finales
149                 )
150
151         # Obtener recorridos posibles
152         with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
            \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
            .txt', 'w') as archivo_recorridos:
153             obtener_recorridos('1', cadena, ['1'],
                archivo_recorridos)
154
155         # Obtener recorridos hasta el estado final
156         with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
            \\sem4\\Teoria\\P2\\Chess\\output\\
            recorridos_finales_blanca.txt', 'w') as
            archivo_recorridos_finales:
157             obtener_recorridos_finales('1', cadena, ['1'],
                archivo_recorridos_finales)
158
159
160     def seleccionar_recorrido():
161         ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
            ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
            recorridos_finales_blanca.txt"
162
163         with open(ruta_archivo, "r") as archivo:
164             if os.path.getsize(ruta_archivo) == 0:
```

```

165         print("No existen soluciones que lleguen al
              estado 16 con la condicion actual. Ingrese
              nueva ruta.")
166     nuevaruta=cadena()
167     print("La nueva ruta es: "+nuevaruta)
168     recorrer_estados(tablaEstados, nuevaruta)
169     print('Se almacenaron las nuevas salidas de los
              recorridos posibles y los recorridos
              exitosos en la carpeta output.\n')
170     lineas = archivo.readlines()
171     # Seleccionar una linea aleatoria
172     recorrido_seleccionado = random.choice(lineas)
173
174     # Eliminar los espacios en blanco y saltos de
        linea
175     recorrido_seleccionado = recorrido_seleccionado
        .strip()
176
177     return recorrido_seleccionado
178
179
180     lineas = archivo.readlines()
181
182     # Seleccionar una linea aleatoria
183     recorrido_seleccionado = random.choice(lineas)
184
185     # Eliminar los espacios en blanco y saltos de
        linea
186     recorrido_seleccionado = recorrido_seleccionado.
        strip()
187
188     return recorrido_seleccionado
189
190 def cadena():
191     while 1:
192         ruta = input("Ingrese la cadena usando las
              transiciones R y N.\n").upper()
193         if len(ruta) > 10:
194             print("\nNo es posible introducir m s de 10
              caracteres en la cadena de recorrido para

```

```

    animacion. Desea seguir de igual forma?(Ya
    no se animara, se procedera al arbol)")
195     respuesta = int(input("1. Si.\n2. No.\n"))
196     if respuesta == 1:
197         break
198     else:
199         pass
200     else:
201         print()
202         break
203     print('\nTamaño de cadena escogido [' + str(len(ruta)) +
        ].')
204     print('\nLa cadena o ruta a seguir escogida sera: ' +
        ruta + '\n')
205     return ruta
206
207 def calcular_coordenadas(estado):
208     cuadro_size = 150 # Tama o de cada cuadro del tablero
209     tablero_width = 4 * cuadro_size # Ancho total del
        tablero
210     tablero_height = 4 * cuadro_size # Altura total del
        tablero
211     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
        para centrar el tablero
212     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
        Y para centrar el tablero
213     fila = (estado - 1) // 4 # Calcular la fila del estado
214     columna = (estado - 1) % 4 # Calcular la columna del
        estado
215     x = tablero_x + columna * cuadro_size # Calcular la
        coordenada X del estado
216     y = tablero_y + fila * cuadro_size # Calcular la
        coordenada Y del estado
217
218     return ((x+33), y+33) # Devolver las coordenadas como
        una tupla
219
220
221
222 #Main del ciclo del juego 2

```

```
223 ruta=cadena()#Solicitamos la cadena generada aleatoriamente
224 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
    sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
    as archivo:
225     archivo.write(ruta)
226 if len(ruta)> 10:
227     input("Presione Enter para continuar...")
228     sys.exit(3)
229 recorrer_estados(tablaEstados, ruta)
230 print('\nEn este punto el programa almaceno en la salidas
    los recorridos posibles y los recorridos exitosos en la
    carpeta output.\n')
231 recorrido = seleccionar_recorrido()#Este fue el recorrido
    que se escogio de manera aleatoria
232
233 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
    sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
    as archivo:
234     archivo.write(ruta)
235
236 print("\nRecorrido seleccionado:", recorrido)
237 print("\nVea la parte grafica.")
238 lista_estados = [int(num) for num in recorrido.split(",")]
239
240 coordenadas=(235,85)
241 nueva_coordenada=(0,0)
242 run=True
243 contador=1
244 while run:
245     timer.tick(fps)
246     screen.fill('dark gray')
247     dibujar_tablero()
248     dibujar_piezas()
249     boton_rect = dibujar_boton()
250
251     for event in pygame.event.get():
252         if event.type == pygame.QUIT:
253             run = False
254
```

```

255         if event.type == pygame.MOUSEBUTTONDOWN and event.
256             button == 1:
257             mouse_pos = pygame.mouse.get_pos()
258             if boton_rect.collidepoint(mouse_pos): #
259                 Verificar si se hizo clic en el bot n
260
261                 nueva_coordenada = calcular_coordenadas(
262                     lista_estados[contador])
263
264                 if turn_step <= 1:
265                     if coordenadas in posicion_blanca:
266                         selection = posicion_blanca.index(
267                             coordenadas)
268                         if turn_step == 0:
269                             turn_step = 1
270
271                         if turn_step==1 and selection != 100:
272                             posicion_blanca[0] =
273                                 nueva_coordenada
274                             selection = 100
275                             contador += 1
276                             coordenadas = nueva_coordenada
277                             turn_step=0
278
279     pygame.display.flip()
280
281     pygame.quit()

```

5.5. Main3.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```

1 #Teoria de la computacion

```

```

2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 import pygame
7 import random
8 import os
9
10 pygame.init() #Acceso al paquete pygame
11 #Ancho
12 WIDTH = 1000
13 #Altura
14 HEIGHT = 700
15 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
16 pygame.display.set_caption('Problema del Ajedrez')
17 font = pygame.font.Font('freesansbold.ttf',20) #Tipo de
    fuente 1 del juego
18 big_font= pygame.font.Font('freesansbold.ttf',50) #Tipo de
    fuente 2 del juego
19 timer = pygame.time.Clock() #velocidad de actualizacion de
    nuestro juego a 60 fps
20 fps=60
21
22 #NFA de estados
23 tablaEstados = {
24     '1' : {'R': {'2','5'}, 'N': '6'},
25     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
26     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
27     '4' : {'R': '7', 'N': {'3','8'}},
28     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
29     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9','11'
        }},
30     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8','11'
        }},
31     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
32     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
33     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9','11','
        14'}},
34     '11' : {'R': {'7','10','12','15'}, 'N': {'6','8','14','

```

```

        16' }},
35     '12' : {'R': {'7', '15'}, 'N': {'8', '11', '16'}},
36     '13' : {'R': '10', 'N': {'9', '14'}},
37     '14' : {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
38     '15' : {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
39     '16' : {'R': {'12', '15'}, 'N': '11'}#Estado 16 es el
        estado Final.
40 }
41
42 #Variables e imagenes del juego
43 pieza_blanca = ['king']
44 pieza_negra = ['king']
45 posicion_blanca = [(235, 85)]
46 posicion_negra = [(683, 85)]
47
48 #
49 turn_step = 0
50 selection = 100
51 valid_moves_for1 = []
52 #Cargar imagenes en juego
53 rey_blanco = pygame.image.load('C:\\Users\\soyco\\OneDrive
        \\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
        white_king.png')
54 rey_blanco = pygame.transform.scale(rey_blanco, (80, 80))
55 rey_negro = pygame.image.load('C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
        black_king.png')
56 rey_negro = pygame.transform.scale(rey_negro, (80, 80))
57
58 imagen_blanca = [rey_blanco]
59 imagen_negra = [rey_negro]
60
61 lista_piezas = ['king']
62 #ver variables/contador flash
63
64
65 boton_presionado = False
66
67 def dibujar_boton():
68     boton_width = 150

```

```
69     boton_height = 45
70     boton_x = (WIDTH - boton_width) // 2
71     boton_y = (HEIGHT - boton_height - 20)+17
72
73     # Dibujar el botón como un rectángulo en la pantalla
74     boton_rect=pygame.Rect(boton_x, boton_y, boton_width,
75                             boton_height)
76     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y,
77                                             boton_width, boton_height))
78     texto = font.render("Siguiente", True, (0, 0, 0))
79     texto_rect = texto.get_rect(center=(boton_x +
80                                         boton_width // 2, boton_y + boton_height // 2))
81     screen.blit(texto, texto_rect)
82     return boton_rect
83
84 #Funcion para dibujar tablero
85 def dibujar_tablero():
86     cuadro_size = 150 # Tamaño de cada cuadro del tablero
87     tablero_width = 4 * cuadro_size # Ancho total del
88     tablero
89     tablero_height = 4 * cuadro_size # Altura total del
90     tablero
91     tablero_x = (WIDTH - tablero_width) // 2 # Posición X
92     para centrar el tablero
93     tablero_y = (HEIGHT - tablero_height) // 2 # Posición
94     Y para centrar el tablero
95
96     numero_color = 'white' # Color del número de casilla
97
98     font = pygame.font.Font(None, 24) # Fuente y tamaño
99     del número de casilla
100
101     for i in range(16): # Iterar 16 veces para un tablero
102         de 4x4
103         column = i % 4
104         fila = i // 4
105         x = tablero_x + column * cuadro_size
106         y = tablero_y + fila * cuadro_size
107         if fila % 2 == 0:
108             color = 'black' if column % 2 == 0 else 'red'
```



```

100         else:
101             color = 'red' if columna % 2 == 0 else 'black'
102             pygame.draw.rect(screen, color, [x, y, cuadro_size,
103                                     cuadro_size])
104             pygame.draw.rect(screen, 'white', [x, y,
105                                     cuadro_size, cuadro_size], 2) # Agregar borde
106                             de color blanco
107             numero_texto = font.render(str(i + 1), True,
108                                     numero_color) # Crear superficie de texto con
109                             el número
110             numero_rect = numero_texto.get_rect(center=((x +
111                                     cuadro_size // 2)-60, y + 20)) # Posición del
112                             número en la parte superior del recuadro
113             screen.blit(numero_texto, numero_rect) # Pegar el
114                             número en la pantalla
115
116 #Funcion para dibujar piezas
117 def dibujar_piezas():
118     index=lista_piezas.index('king')
119     if pieza_blanca[0]=='king':
120         screen.blit(imagen_blanca[index], (posicion_blanca
121             [0][0],posicion_blanca[0][1]))
122     if pieza_negra[0]=='king':
123         screen.blit(imagen_negra[index], (posicion_negra
124             [0][0],posicion_negra[0][1]))
125
126 def recorrer_estados_blanca(tabla_estados, cadena,
127                             estadoInicial):
128     # Funcion para obtener todos los recorridos posibles
129     blanca
130     def obtener_recorridos(estado_actual,
131                             simbolos_restantes, recorrido_actual):
132         if not simbolos_restantes:
133             recorridos2.append(recorrido_actual)
134             return
135
136         simbolo = simbolos_restantes[0]
137         if estado_actual in tabla_estados and simbolo in
138             tabla_estados[estado_actual]:
139             transiciones = tabla_estados[estado_actual][

```

```

simbolo]
126
127     for estado_siguiente in transiciones:
128         obtener_recorridos(estado_siguiente,
                             simbolos_restantes[1:], recorrido_actual
                             + [estado_siguiente])
129
130 # Funci n para obtener los recorridos v lidos hasta
    el estado final
131 def obtener_recorridos_finales(estado_actual,
    simbolos_restantes, recorrido_actual):
132     if estado_actual == '16':
133         recorridos_finales.append(recorrido_actual)
134         return
135
136     if not simbolos_restantes:
137         return
138
139     simbolo = simbolos_restantes[0]
140     if estado_actual in tabla_estados and simbolo in
        tabla_estados[estado_actual]:
141         transiciones = tabla_estados[estado_actual][
            simbolo]
142
143         for estado_siguiente in transiciones:
144             obtener_recorridos_finales(estado_siguiente
                , simbolos_restantes[1:],
                recorrido_actual + [estado_siguiente])
145
146
147 # Obtener recorridos posibles
148 recorridos2 = []
149 obtener_recorridos(estadoInicial, cadena, [
    estadoInicial])
150 # Obtener recorridos hasta el estado final
151 recorridos_finales = []
152 obtener_recorridos_finales(estadoInicial, cadena, [
    estadoInicial])
153 # Guardar los recorridos en archivos de texto
154 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM

```

```

155     \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
156     .txt', 'w') as archivo_recorridos:
157         archivo_recorridos.write('Recorridos posibles:\n')
158         for recorrido in recorridos2:
159             archivo_recorridos.write(','.join(recorrido) +
160                 '\n')
161
162     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
163             \\sem4\\Teoria\\P2\\Chess\\output\\
164             recorridos_finales_blanca.txt', 'w') as
165             archivo_recorridos_finales:
166                 for recorrido in recorridos_finales:
167                     archivo_recorridos_finales.write(','.join(
168                         recorrido) + '\n')
169
170 def recorrer_estados_negra(tabla_estados, cadena,
171     estadoInicial):
172     # Funci n para obtener todos los recorridos posibles
173     def obtener_recorridos(estado_actual,
174         simbolos_restantes, recorrido_actual):
175         if not simbolos_restantes:
176             recorridos2.append(recorrido_actual)
177             return
178
179         simbolo = simbolos_restantes[0]
180         if estado_actual in tabla_estados and simbolo in
181             tabla_estados[estado_actual]:
182             transiciones = tabla_estados[estado_actual][
183                 simbolo]
184
185             for estado_siguiente in transiciones:
186                 obtener_recorridos(estado_siguiente,
187                     simbolos_restantes[1:], recorrido_actual
188                         + [estado_siguiente])
189
190     # Funci n para obtener los recorridos v lidos hasta
191     el estado final

```

```
181     def obtener_recorridos_finales(estado_actual,
182                                   simbolos_restantes, recorrido_actual):
183         if estado_actual == '13':
184             recorridos_finales.append(recorrido_actual)
185             return
186
187         if not simbolos_restantes:
188             return
189
190         simbolo = simbolos_restantes[0]
191         if estado_actual in tabla_estados and simbolo in
192             tabla_estados[estado_actual]:
193             transiciones = tabla_estados[estado_actual][
194                 simbolo]
195
196             for estado_siguiente in transiciones:
197                 obtener_recorridos_finales(estado_siguiente
198                                           , simbolos_restantes[1:],
199                                           recorrido_actual + [estado_siguiente])
200
201     # Obtener recorridos posibles
202     recorridos2 = []
203     obtener_recorridos(estadoInicial, cadena, [
204         estadoInicial])
205
206     # Obtener recorridos hasta el estado final
207     recorridos_finales = []
208     obtener_recorridos_finales(estadoInicial, cadena, [
209         estadoInicial])
210
211     # Guardar los recorridos en archivos de texto
212     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
213              \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_negra.
214              txt', 'w') as archivo_recorridos:
215         archivo_recorridos.write('Recorridos posibles:\n')
216         for recorrido in recorridos2:
217             archivo_recorridos.write(','.join(recorrido) +
218                                       '\n')
219
220     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
```

```
211     \\sem4\\Teoria\\P2\\Chess\\output\\
212     recorridos_finales_negra.txt', 'w') as
    archivo_recorridos_finales:
213     for recorrido in recorridos_finales:
214         archivo_recorridos_finales.write(','.join(
215             recorrido) + '\n')
216
217 def seleccionar_recorrido_blanca(estadoI):
218     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
219     ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
220     recorridos_finales_blanca.txt"
221     x=1
222     while x==1:
223         with open(ruta_archivo, "r+") as archivo:
224             if os.path.getsize(ruta_archivo) == 0:
225                 print("No existen soluciones que lleguen al
226                     estado 16 con la condicion actual. Se
227                     recalculara una ruta.\n")
228                 nuevavuta=cadena()
229                 with open('C:\\Users\\soyco\\OneDrive\\
230                     Documents\\ESCOM\\sem4\\Teoria\\P2\\
231                     Chess\\output\\ruta_blanca.txt', 'w') as
232                     archivo2:
233                         archivo2.write(nuevaruta)
234                         print("La nueva ruta es: "+nuevaruta)
235                         recorrer_estados_blanca(tablaEstados,
236                             nuevavuta, estadoI)
237                         print('Se almacenaron las nuevas salidas de
238                             los recorridos posibles y los
239                             recorridos exitosos en la carpeta output
240                             .\n')
241             else:
242                 x=0
243                 lineas = archivo.readlines()
244
245                 # Seleccionar una l nea aleatoria
246                 recorrido_seleccionado = random.choice(
247                     lineas)
```

```
235         # Eliminar los espacios en blanco y saltos
           de l nea
236         recorrido_seleccionado =
           recorrido_seleccionado.strip()
237
238     return recorrido_seleccionado
239
240 def seleccionar_recorrido_negra(estadoI):
241     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
           ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
           recorridos_finales_negra.txt"
242     x=1
243     while x==1:
244         with open(ruta_archivo, "r+") as archivo:
245             if os.path.getsize(ruta_archivo) == 0:
246                 print("No existen soluciones que lleguen al
                   estado 13 con la condicion actual. Se
                   recalculara una ruta.\n")
247                 nuevavuta=cadena()
248                 with open('C:\\Users\\soyco\\OneDrive\\
                   Documents\\ESCOM\\sem4\\Teoria\\P2\\
                   Chess\\output\\ruta_negra.txt', 'w') as
                   archivo2:
249                     archivo2.write(nuevaruta)
250                 print("La nueva ruta es: "+nuevaruta)
251                 recorrer_estados_negra(tablaEstados,
                   nuevavuta,estadoI)
252                 print('Se almacenaron las nuevas salidas de
                   los recorridos posibles y los
                   recorridos exitosos en la carpeta output
                   .\n')
253             else:#16
254                 x=0
255                 lineas = archivo.readlines()
256
257                 # Seleccionar una l nea aleatoria
258                 recorrido_seleccionado = random.choice(
                   lineas)
259
260                 # Eliminar los espacios en blanco y saltos
```

```

261         de l nea
                recorrido_seleccionado =
                recorrido_seleccionado.strip()
262
263     return recorrido_seleccionado
264
265 def cadenaRandom(numero): #Genera un string de forma random
266     auxiliar = '' #Variable auxiliar
267     for i in range(numero):
268         x = random.randint(1, 2) #Funci n para generar un
                resultado random de una lista
269         if x % 2 == 0:
270             auxiliar = auxiliar + "R"
271         else:
272             auxiliar = auxiliar + "N"
273     return auxiliar
274
275 def cadena():
276     numero = random.randint(4,10)
277     print('\nTamaño de cadena escogido aleatoriamente [' +
        str(numero) + ']\nTambien se generaran las
        transiciones R y N aleatoriamente.\n')
278     cad = cadenaRandom(numero) #Se genera de forma
        aleatoria del 1-10
279     print('\nLa cadena o ruta a seguir escogida
        aleatoriamente sera: ' + cad + '\n')
280     return cad
281
282 def calcular_coordenadas(estado):
283     cuadro_size = 150 # Tama o de cada cuadro del tablero
284     tablero_width = 4 * cuadro_size # Ancho total del
        tablero
285     tablero_height = 4 * cuadro_size # Altura total del
        tablero
286     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
        para centrar el tablero
287     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
        Y para centrar el tablero
288     fila = (estado - 1) // 4 # Calcular la fila del estado
289     columna = (estado - 1) % 4 # Calcular la columna del

```

```
estado
290     x = tablero_x + columna * cuadro_size # Calcular la
        coordenada X del estado
291     y = tablero_y + fila * cuadro_size # Calcular la
        coordenada Y del estado
292     return ((x+33), y+33) # Devolver las coordenadas como
        una tupla
293
294
295 #Main del ciclo del juego 3
296 print("\n--PARA CADENA BLANCA--")
297 ruta_blanca=cadena()#Solicitamos la cadena generada
        aleatoriamente. Para pieza blanca.
298 recorrer_estados_blanca(tablaEstados, ruta_blanca,"1")
299 input()
300 print("\n--PARA CADENA NEGRA--")
301 ruta_negra=cadena()#Solicitamos la cadena generada
        aleatoriamente. Para pieza negra.
302 recorrer_estados_negra(tablaEstados, ruta_negra,"4")
303
304 print('\nEn este punto el programa almaceno en la salidas
        los recorridos posibles y los recorridos exitosos en la
        carpeta output.\n')
305 recorrido_blanca = seleccionar_recorrido_blanca("1")#Este
        fue el recorrido que se escogio de manera aleatoria para
        blanca
306 recorrido_negra = seleccionar_recorrido_negra("4")#Este fue
        el recorrido que se escogio de manera aleatoria para
        negra
307
308 print("\nRecorrido seleccionado para blanca:",
        recorrido_blanca)
309 print("\nRecorrido seleccionado para negra:",
        recorrido_negra)
310 lista_estados_blanca = [int(num) for num in
        recorrido_blanca.split(",")]
311 lista_estados_negra = [int(num) for num in recorrido_negra.
        split(",")]
312
313 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
```



```

    sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
    as archivo:
314     archivo.write(ruta_blanca)
315 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
    sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.txt', 'w')
    as archivo:
316     archivo.write(ruta_negra)
317 # Verificar el numero aleatorio y determinar si se sacan
    piezas blancas o negras
318 numero_aleatorio = random.randint(1, 2)
319 coordenadas_blanca=(235,85)
320 nueva_coordenada_blanca=(0,0)
321 coordenadas_negra=(683,85)
322 nueva_coordenada_negra=(0,0)
323 run=True
324 contador_blanca=1
325 contador_negra=1
326
327 if numero_aleatorio == 1:
328     print("\nSacar piezas blancas.")
329     # Sacar piezas blancas
330     while run:
331         timer.tick(fps)
332         screen.fill('dark gray')
333         dibujar_tablero()
334         dibujar_piezas()
335         boton_rect = dibujar_boton()
336
337     for event in pygame.event.get():
338         if event.type == pygame.QUIT:
339             run = False
340
341         if event.type == pygame.MOUSEBUTTONDOWN and
            event.button == 1:
342             mouse_pos = pygame.mouse.get_pos()
343             if boton_rect.collidepoint(mouse_pos): #
                Verificar si se hizo clic en el bot n
344                 if turn_step ==4:
345                     turn_step=2
346                     nueva_coordenada_blanca =

```

```
        calcular_coordenadas(  
            lista_estados_blanca[contador_blanca  
        ])  
347 nueva_coordenada_negra =  
        calcular_coordenadas(  
            lista_estados_negra[contador_negra])  
348  
349 if turn_step <= 1:  
350     #print("Blanca")  
351     if nueva_coordenada_blanca in  
        posicion_negra:  
352         while(1):#Ciclo while, donde  
            para salir la siguiente  
            coordenada no pueda ser la  
            de la colision  
353             #recalculamos ruta  
354             print("Colision detectada  
                en recorrido:  
                Recalculamos blanca")  
355             ruta_blanca=cadena()#  
                Solicitamos la cadena  
                generada aleatoriamente.  
                Para pieza blanca.  
356             recorrer_estados_blanca(  
                tablaEstados,  
                ruta_blanca,str(  
                    lista_estados_blanca[  
                        contador_blanca-1]))  
357             recorrido_blanca =  
                seleccionar_recorrido_blanca  
                (str(  
                    lista_estados_blanca[  
                        contador_blanca-1]))#  
                Este fue el recorrido  
                que se escogio de manera  
                aleatoria para blanca  
358             print("La ruta recalculada  
                es: "+recorrido_blanca)  
359             lista_estados_blanca = [int  
                (num) for num in
```

```

recorrido_blanca.split(",")
360 contador_blanca=1
361 nueva_coordenada_blanca =
    calcular_coordenadas(
        lista_estados_blanca[
            contador_blanca])
362 if nueva_coordenada_blanca
    not in posicion_negra:
363     break
364 turn_step=0
365 if coordenadas_blanca in
    posicion_blanca:
366     selection = posicion_blanca.
        index(coordenadas_blanca)
367 if turn_step == 0:
368     turn_step = 1
369
370 if turn_step==1 and selection !=
    100:
371     posicion_blanca[0] =
        nueva_coordenada_blanca
372
373     selection = 100
374     contador_blanca += 1
375     coordenadas_blanca =
        nueva_coordenada_blanca
376     turn_step=4
377 if turn_step == 2:
378     #print("Negra")
379     if nueva_coordenada_negra in
        posicion_blanca:
380         while(1):
381             #recalculamos ruta
382             print("Colision detectada
                en recorrido:
                Recalculamos negra.")
383             ruta_negra=cadena()#
                Solicitamos la cadena
                generada aleatoriamente.

```

```
384     Para pieza negra.
recorrer_estados_negra(
    tablaEstados, ruta_negra
    ,str(lista_estados_negra
        [contador_negra-1]))
385 recorrido_negra =
    seleccionar_recorrido_negra
    (str(lista_estados_negra
        [contador_negra-1]))#
    Este fue el recorrido
    que se escogio de manera
    aleatoria para blanca
386 print("La ruta recalculada
    es: "+recorrido_negra)
387 lista_estados_negra = [int(
    num) for num in
    recorrido_negra.split(",
    ")]
388 contador_negra=1
389 nueva_coordenada_negra =
    calcular_coordenadas(
    lista_estados_negra[
    contador_negra])
390 if nueva_coordenada_negra
    not in posicion_blanca:
391     break
392     turn_step=2
393 if coordenadas_negra in
    posicion_negra:
394     selection = posicion_negra.
        index(coordenadas_negra)#
        Seleccionamos las
        coordenadas
395     if turn_step == 2:
396         turn_step = 3
397
398 if turn_step==3 and selection !=
    100:
399     posicion_negra[0] =
        nueva_coordenada_negra
```

```
400             selection = 100
401             contador_negra += 1
402             coordenadas_negra =
403                 nueva_coordenada_negra
404             turn_step=0
405         pygame.display.flip()
406     else:
407         # Sacan piezas negras
408         print("\nSacar piezas negras.")
409         while run:
410             timer.tick(fps)
411             screen.fill('dark gray')
412             dibujar_tablero()
413             dibujar_piezas()
414             boton_rect = dibujar_boton()
415
416             for event in pygame.event.get():
417                 if event.type == pygame.QUIT:
418                     run = False
419
420                 if event.type == pygame.MOUSEBUTTONDOWN and
421                     event.button == 1:
422                     mouse_pos = pygame.mouse.get_pos()
423                     if boton_rect.collidepoint(mouse_pos): #
424                         # Verificar si se hizo clic en el bot n
425                         if turn_step == 4:
426                             turn_step=2
427                             nueva_coordenada_blanca =
428                                 calcular_coordenadas(
429                                     lista_estados_blanca[contador_blanca
430                                     ])
431                             nueva_coordenada_negra =
432                                 calcular_coordenadas(
433                                     lista_estados_negra[contador_negra])
434
435                         if turn_step <= 1:
436                             #print("Negra")
437                             if nueva_coordenada_negra in
438                                 posicion_blanca:
```

```
431 while(1):#Ciclo while, donde
    para salir la siguiente
    coordenada no pueda ser la
    de la colision
432     #recalculamos ruta
433     print("Colision detectada
        en recorrido:
        Recalculamos negra.")
434     ruta_negra=cadena()#
        Solicitamos la cadena
        generada aleatoriamente.
        Para pieza negra.
435     recorrer_estados_negra(
        tablaEstados, ruta_negra
        ,str(lista_estados_negra
        [contador_negra-1]))
436     recorrido_negra =
        seleccionar_recorrido_negra
        (str(lista_estados_negra
        [contador_negra-1]))#
        Este fue el recorrido
        que se escogio de manera
        aleatoria para blanca
437     print("La ruta recalculada
        es: "+recorrido_negra)
438     lista_estados_negra = [int(
        num) for num in
        recorrido_negra.split(",
        ")]
439     contador_negra=1
440     nueva_coordenada_negra =
        calcular_coordenadas(
        lista_estados_negra[
        contador_negra])
441     if nueva_coordenada_negra
        not in posicion_blanca:
442         break
443     turn_step=0
444 if coordenadas_negra in
    posicion_negra:
```

```
445         selection = posicion_negra.  
            index(coordenadas_negra) #  
            Seleccionamos las  
            coordenadas  
446     if turn_step == 0:  
447         turn_step = 1  
448  
449     if turn_step==1 and selection !=  
100:  
450         posicion_negra[0] =  
            nueva_coordenada_negra  
451         selection = 100  
452         contador_negra += 1  
453         coordenadas_negra =  
            nueva_coordenada_negra  
454         turn_step=4  
455     if turn_step == 2:  
456         #print("Blanca")  
457         if nueva_coordenada_blanca in  
            posicion_negra:  
458             while(1):  
459                 #recalculamos ruta  
460                 print("Colision detectada  
                    en recorrido:  
                    Recalculamos blanca")  
461                 ruta_blanca=cadena() #  
                    Solicitamos la cadena  
                    generada aleatoriamente.  
                    Para pieza blanca.  
462                 recorrer_estados_blanca(  
                    tablaEstados,  
                    ruta_blanca,str(  
                    lista_estados_blanca[  
                    contador_blanca-1]))  
463                 recorrido_blanca =  
                    seleccionar_recorrido_blanca  
                    (str(  
                    lista_estados_blanca[  
                    contador_blanca-1])) #  
                    Este fue el recorrido
```

```
que se escogio de manera
aleatoria para blanca
464 print("La ruta recalculada
es: "+recorrido_blanca)
465 lista_estados_blanca = [int
(num) for num in
recorrido_blanca.split("
,")]
466 contador_blanca=1
467 nueva_coordenada_blanca =
calcular_coordenadas(
lista_estados_blanca[
contador_blanca])
468 if nueva_coordenada_blanca
not in posicion_negra:
469     break
470 turn_step=2
471 if coordenadas_blanca in
posicion_blanca:
472     selection = posicion_negra.
index(coordenadas_negra) #
Seleccionamos las
coordenadas
473 if turn_step == 2:
474     turn_step = 3
475
476 if turn_step==3 and selection !=
100:
477     posicion_blanca[0] =
nueva_coordenada_blanca
478
479     selection = 100
480     contador_blanca += 1
481     coordenadas_blanca =
nueva_coordenada_blanca
482     turn_step=0
483
484 pygame.display.flip()
485
486 pygame.quit()
```


5.6. Main4.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1 #Teoria de la computacion
2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 import sys
7 import pygame
8 import random
9 import os
10
11 pygame.init() #Acceso al paquete pygame
12 #Ancho
13 WIDTH = 1000
14 #Altura
15 HEIGHT = 700
16 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
17 pygame.display.set_caption('Problema del Ajedrez')
18 font = pygame.font.Font('freesansbold.ttf',20) #Tipo de
    fuente 1 del juego
19 big_font= pygame.font.Font('freesansbold.ttf',50) #Tipo de
    fuente 2 del juego
20 timer = pygame.time.Clock() #velocidad de actualizacion de
    nuestro juego a 60 fps
21 fps=60
22
23 #NFA de estados
24 tablaEstados = {
25     '1' : {'R': {'2','5'}, 'N': '6'},
```

```

26     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
27     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
28     '4' : {'R': {'7'}, 'N': {'3','8'}},
29     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
30     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9','11'}},
31     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8','11'}},
32     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
33     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
34     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9','11','14'}},
35     '11' : {'R': {'7','10','12','15'}, 'N': {'6','8','14','16'}},
36     '12' : {'R': {'7','15'}, 'N': {'8','11','16'}},
37     '13' : {'R': {'10'}, 'N': {'9','14'}},
38     '14' : {'R': {'13','10','15'}, 'N': {'9','11'}},
39     '15' : {'R': {'10','12'}, 'N': {'11','14','16'}},
40     '16' : {'R': {'12','15'}, 'N': {'11'}}#Estado 16 es el
        estado Final.
41 }
42
43 #Variables e imagenes del juego
44 pieza_blanca = ['king']
45 pieza_negra = ['king']
46 posicion_blanca = [(235,85)]
47 posicion_negra = [(683,85)]
48
49 #
50 turn_step = 0
51 selection= 100
52 valid_moves_for1 =[]
53 #Cargar imagenes en juego
54 rey_blanco = pygame.image.load('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\white_king.png')
55 rey_blanco = pygame.transform.scale(rey_blanco, (80,80))
56 rey_negro = pygame.image.load('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\black_king.png')

```

```

57 rey_negro = pygame.transform.scale(rey_negro, (80,80))
58
59 imagen_blanca = [rey_blanco]
60 imagen_negra=[rey_negro]
61
62 lista_piezas = ['king']
63 #ver variables/contador flash
64
65
66 boton_presionado = False
67
68 def dibujar_boton():
69     boton_width = 150
70     boton_height = 45
71     boton_x = (WIDTH - boton_width) // 2
72     boton_y = (HEIGHT - boton_height - 20)+17
73
74     # Dibujar el boton como un rect ngulo en la pantalla
75     boton_rect=pygame.Rect(boton_x, boton_y, boton_width,
76                             boton_height)
77     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
78         , boton_width, boton_height))
79     texto = font.render("Siguiente", True, (0, 0, 0))
80     texto_rect = texto.get_rect(center=(boton_x +
81         boton_width // 2, boton_y + boton_height // 2))
82     screen.blit(texto, texto_rect)
83     return boton_rect
84
85 #Funcion para dibujar tablero
86 def dibujar_tablero():
87     cuadro_size = 150 # Tamaño de cada cuadro del tablero
88     tablero_width = 4 * cuadro_size # Ancho total del
89         tablero
90     tablero_height = 4 * cuadro_size # Altura total del
91         tablero
92     tablero_x = (WIDTH - tablero_width) // 2 # Posición X
93         para centrar el tablero
94     tablero_y = (HEIGHT - tablero_height) // 2 # Posición
95         Y para centrar el tablero

```

```

90     numero_color = 'white' # Color del n mero de casilla
91
92     font = pygame.font.Font(None, 24) # Fuente y tama o
        del n mero de casilla
93
94     for i in range(16): # Iterar 16 veces para un tablero
        de 4x4
95         columna = i % 4
96         fila = i // 4
97         x = tablero_x + columna * cuadro_size
98         y = tablero_y + fila * cuadro_size
99         if fila % 2 == 0:
100             color = 'black' if columna % 2 == 0 else 'red'
101         else:
102             color = 'red' if columna % 2 == 0 else 'black'
103         pygame.draw.rect(screen, color, [x, y, cuadro_size,
            cuadro_size])
104         pygame.draw.rect(screen, 'white', [x, y,
            cuadro_size, cuadro_size], 2) # Agregar borde
            de color blanco
105         numero_texto = font.render(str(i + 1), True,
            numero_color) # Crear superficie de texto con
            el n mero
106         numero_rect = numero_texto.get_rect(center=((x +
            cuadro_size // 2)-60, y + 20)) # Posici n del
            n mero en la parte superior del recuadro
107         screen.blit(numero_texto, numero_rect) # Pegar el
            n mero en la pantalla
108
109 #Funcion para dibujar piezas
110 def dibujar_piezas():
111     index=lista_piezas.index('king')
112     if pieza_blanca[0]=='king':
113         screen.blit(imagen_blanca[index], (posicion_blanca
            [0][0],posicion_blanca[0][1]))
114     if pieza_negra[0]=='king':
115         screen.blit(imagen_negra[index], (posicion_negra
            [0][0],posicion_negra[0][1]))
116
117 def recorrer_estados_blanca(tabla_estados, cadena,

```

```
estadoInicial):
118     # Funci n para obtener todos los recorridos posibles
        blanca
119     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual):
120         if not simbolos_restantes:
121             recorridos2.append(recorrido_actual)
122             return
123
124         simbolo = simbolos_restantes[0]
125         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
126             transiciones = tabla_estados[estado_actual][
                simbolo]
127
128             for estado_siguiente in transiciones:
129                 obtener_recorridos(estado_siguiente,
                    simbolos_restantes[1:], recorrido_actual
                        + [estado_siguiente])
130
131     # Funci n para obtener los recorridos v lidos hasta
        el estado final
132     def obtener_recorridos_finales(estado_actual,
        simbolos_restantes, recorrido_actual):
133         if estado_actual == '16':
134             recorridos_finales.append(recorrido_actual)
135             return
136
137         if not simbolos_restantes:
138             return
139
140         simbolo = simbolos_restantes[0]
141         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
142             transiciones = tabla_estados[estado_actual][
                simbolo]
143
144             for estado_siguiente in transiciones:
145                 obtener_recorridos_finales(estado_siguiente
                    , simbolos_restantes[1:],
```

```
recorrido_actual + [estado_siguiente])
146
147
148 # Obtener recorridos posibles
149 recorridos2 = []
150 obtener_recorridos(estadoInicial, cadena, [
    estadoInicial])
151 # Obtener recorridos hasta el estado final
152 recorridos_finales = []
153 obtener_recorridos_finales(estadoInicial, cadena, [
    estadoInicial])
154 # Guardar los recorridos en archivos de texto
155 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
    \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
    .txt', 'w') as archivo_recorridos:
156     archivo_recorridos.write('Recorridos posibles:\n')
157     for recorrido in recorridos2:
158         archivo_recorridos.write(','.join(recorrido) +
            '\n')
159
160 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
    \\sem4\\Teoria\\P2\\Chess\\output\\
    recorridos_finales_blanca.txt', 'w') as
    archivo_recorridos_finales:
161     for recorrido in recorridos_finales:
162         archivo_recorridos_finales.write(','.join(
            recorrido) + '\n')
163
164
165
166
167 def recorrer_estados_negra(tabla_estados, cadena,
    estadoInicial):
168     # Funci n para obtener todos los recorridos posibles
169     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual):
170         if not simbolos_restantes:
171             recorridos2.append(recorrido_actual)
172             return
173
```

```
174     simbolo = simbolos_restantes[0]
175     if estado_actual in tabla_estados and simbolo in
176         tabla_estados[estado_actual]:
177         transiciones = tabla_estados[estado_actual][
178             simbolo]
179
180         for estado_siguiente in transiciones:
181             obtener_recorridos(estado_siguiente,
182                 simbolos_restantes[1:], recorrido_actual
183                 + [estado_siguiente])
184
185 # Funci n para obtener los recorridos v lidos hasta
186 el estado final
187 def obtener_recorridos_finales(estado_actual,
188     simbolos_restantes, recorrido_actual):
189     if estado_actual == '13':
190         recorridos_finales.append(recorrido_actual)
191         return
192
193     if not simbolos_restantes:
194         return
195
196     simbolo = simbolos_restantes[0]
197     if estado_actual in tabla_estados and simbolo in
198         tabla_estados[estado_actual]:
199         transiciones = tabla_estados[estado_actual][
200             simbolo]
201
202         for estado_siguiente in transiciones:
203             obtener_recorridos_finales(estado_siguiente
204                 , simbolos_restantes[1:],
205                 recorrido_actual + [estado_siguiente])
206
207 # Obtener recorridos posibles
208 recorridos2 = []
209 obtener_recorridos(estadoInicial, cadena, [
210     estadoInicial])
211
212 # Obtener recorridos hasta el estado final
213 recorridos_finales = []
```



```

nuevaruta, estadoI)
228     print('Se almacenaron las nuevas salidas de
        los recorridos posibles y los
        recorridos exitosos en la carpeta output
        .\n')
229     else:
230         x=0
231         lineas = archivo.readlines()
232
233         # Seleccionar una linea aleatoria
234         recorrido_seleccionado = random.choice(
            lineas)
235
236         # Eliminar los espacios en blanco y saltos
            de linea
237         recorrido_seleccionado =
            recorrido_seleccionado.strip()
238
239     return recorrido_seleccionado
240
241 def seleccionar_recorrido_negra(estadoI):
242     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_negra.txt"
243     x=1
244     while x==1:
245         with open(ruta_archivo, "r+") as archivo:
246             if os.path.getsize(ruta_archivo) == 0:
247                 print("No existen soluciones que lleguen al
                    estado 13 con la condicion actual. Se
                    recalculara una ruta.\n")
248                 nuevavuta=cadena()
249                 with open('C:\\Users\\soyco\\OneDrive\\
                    Documents\\ESCOM\\sem4\\Teoria\\P2\\
                    Chess\\output\\ruta_negra.txt', 'w') as
                    archivo2:
250                     archivo2.write(nuevaruta)
251                 print("La nueva ruta es: "+nuevaruta)
252                 recorrer_estados_negra(tablaEstados,
                    nuevavuta, estadoI)
```

```

253         print('Se almacenaron las nuevas salidas de
                los recorridos posibles y los
                recorridos exitosos en la carpeta output
                .\n')
254     else:#16
255         x=0
256         lineas = archivo.readlines()
257
258         # Seleccionar una linea aleatoria
259         recorrido_seleccionado = random.choice(
            lineas)
260
261         # Eliminar los espacios en blanco y saltos
            de linea
262         recorrido_seleccionado =
            recorrido_seleccionado.strip()
263
264     return recorrido_seleccionado
265
266 def cadenaRandom(numero): #Genera un string de forma random
267     auxiliar = '' #Variable auxiliar
268     for i in range(numero):
269         x = random.randint(1, 2) #Funcion para generar un
            resultado random de una lista
270         if x % 2 == 0:
271             auxiliar = auxiliar + "R"
272         else:
273             auxiliar = auxiliar + "N"
274     return auxiliar
275
276 def cadena():
277     numero = random.randint(4,10)
278     print('\nTamaño de cadena escogido aleatoriamente [' +
            str(numero) + ']\nTambien se generaran las
            transiciones R y N aleatoriamente.\n')
279     cad = cadenaRandom(numero) #Se genera de forma
            aleatoria del 1-10
280     print('\nLa cadena o ruta a seguir escogida
            aleatoriamente sera: ' + cad + '\n')
281     return cad

```

```
282
283 def cadena_manual():
284     while 1:
285         ruta = input("Ingrese la cadena usando las
                transiciones R y N.\n").upper()
286         if len(ruta) > 10:
287             print("\nNo es posible introducir m s de 10
                caracteres en la cadena de recorrido para
                animacion. Desea seguir de igual forma?(Ya
                no se animara, se procedera al arbol)")
288             respuesta = int(input("1. Si.\n2. No.\n"))
289             if respuesta == 1:
290                 break
291             else:
292                 pass
293         else:
294             break
295     print('\nTamaño de cadena escogido [' + str(len(ruta)) +
        '].')
296     print('\nLa cadena o ruta a seguir escogida sera: ' +
        ruta + '\n')
297     return ruta
298
299 def calcular_coordenadas(estado):
300     cuadro_size = 150 # Tama o de cada cuadro del tablero
301     tablero_width = 4 * cuadro_size # Ancho total del
        tablero
302     tablero_height = 4 * cuadro_size # Altura total del
        tablero
303     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
        para centrar el tablero
304     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
        Y para centrar el tablero
305     fila = (estado - 1) // 4 # Calcular la fila del estado
306     columna = (estado - 1) % 4 # Calcular la columna del
        estado
307     x = tablero_x + columna * cuadro_size # Calcular la
        coordenada X del estado
308     y = tablero_y + fila * cuadro_size # Calcular la
        coordenada Y del estado
```

```
309         return ((x+33), y+33) # Devolver las coordenadas como
           una tupla
310
311
312
313 #def dibujar_movimientos():
314
315 #Main del ciclo del juego 4
316 # Verificar el n mero aleatorio y determinar si se sacan
           piezas blancas o negras
317 numero_aleatorio = random.randint(1, 2)
318 # Escogemos aleatoriamente que pieza le toca jugar al
           manual
319 juegomanual = random.randint(1, 2)
320
321 #Guardamos rutas para arbol.
322 if juegomanual == 1:
323     print("Te toca jugar con blancas.")
324     print("\n--PARA CADENA BLANCA--")
325     ruta_blanca=cadena_manual()
326     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
           \\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt',
           'w') as archivo:
327         archivo.write(ruta_blanca)
328     if len(ruta_blanca)> 10:
329         sys.exit(3)
330     recorrer_estados_blanca(tablaEstados, ruta_blanca,"1")
331     print("\n--PARA CADENA NEGRA--")
332     ruta_negra=cadena() #Solicitamos la cadena generada
           aleatoriamente. Para pieza negra.
333     recorrer_estados_negra(tablaEstados, ruta_negra,"4")
334
335 elif juegomanual == 2:
336     print("Te toca jugar con negras.")
337     print("\n--PARA CADENA NEGRA--")
338     ruta_negra=cadena_manual()
339     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
           \\sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.txt',
           'w') as archivo:
340         archivo.write(ruta_negra)
```

```
341     if len(ruta_negra)> 10:
342         sys.exit(3)
343     recorrer_estados_negra(tablaEstados, ruta_negra,"4")
344     print("\n--PARA CADENA BLANCA--")
345     ruta_blanca=cadena()#Solicitamos la cadena generada
        aleatoriamente. Para pieza blanca.
346     recorrer_estados_blanca(tablaEstados, ruta_blanca,"1")
347
348
349 print('\nEn este punto el programa almaceno en la salidas
        los recorridos posibles y los recorridos exitosos en la
        carpeta output.\n')
350 recorrido_blanca = seleccionar_recorrido_blanca("1")#Este
        fue el recorrido que se escogio de manera aleatoria para
        blanca
351 recorrido_negra = seleccionar_recorrido_negra("4")#Este fue
        el recorrido que se escogio de manera aleatoria para
        negra
352
353 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
        sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
        as archivo:
354     archivo.write(ruta_blanca)
355 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
        sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.txt', 'w')
        as archivo:
356     archivo.write(ruta_negra)
357
358 print("\nRecorrido seleccionado para blanca:",
        recorrido_blanca)
359 print("\nRecorrido seleccionado para negra:",
        recorrido_negra)
360 lista_estados_blanca = [int(num) for num in
        recorrido_blanca.split(",")]
361 lista_estados_negra = [int(num) for num in recorrido_negra.
        split(",")]
362
363
364 coordenadas_blanca=(235,85)
365 nueva_coordenada_blanca=(0,0)
```



```
398     para salir la siguiente
399     coordenada no pueda ser la
400     de la colision
401     #recalculamos ruta
402     print("Colision detectada
        en recorrido:
        Recalculamos blanca")
403     ruta_blanca=cadena() #
        Solicitamos la cadena
        generada aleatoriamente.
        Para pieza blanca.
404     recorrer_estados_blanca(
        tablaEstados,
        ruta_blanca, str(
        lista_estados_blanca[
        contador_blanca-1]))
405     recorrido_blanca =
        seleccionar_recorrido_blanca
        (str(
        lista_estados_blanca[
        contador_blanca-1])) #
        Este fue el recorrido
        que se escogio de manera
        aleatoria para blanca
406     print("La ruta recalculada
        es: "+recorrido_blanca)
407     lista_estados_blanca = [int
        (num) for num in
        recorrido_blanca.split("
        ,")]
408     contador_blanca=1
409     nueva_coordenada_blanca =
        calcular_coordenadas(
        lista_estados_blanca[
        contador_blanca])
410     if nueva_coordenada_blanca
        not in posicion_negra:
        break
        turn_step=0
    if coordenadas_blanca in
```

```
411         posicion_blanca:
412             selection = posicion_blanca.
413                 index(coordenadas_blanca)
414             if turn_step == 0:
415                 turn_step = 1
416
417         if turn_step==1 and selection !=
418             100:
419             posicion_blanca[0] =
420                 nueva_coordenada_blanca
421
422             selection = 100
423             contador_blanca += 1
424             coordenadas_blanca =
425                 nueva_coordenada_blanca
426             turn_step=4
427         if turn_step == 2:
428             #print("Negra")
429             if nueva_coordenada_negra in
430                 posicion_blanca:
431                 while(1):
432                     #recalculamos ruta
433                     print("Colision detectada
434                         en recorrido:
435                         Recalculamos negra.")
436                     ruta_negra=cadena()#
437                     Solicitamos la cadena
438                     generada aleatoriamente.
439                     Para pieza negra.
440                     recorrer_estados_negra(
441                         tablaEstados, ruta_negra
442                         ,str(lista_estados_negra
443                             [contador_negra-1]))
444                     recorrido_negra =
445                         seleccionar_recorrido_negra
446                         (str(lista_estados_negra
447                             [contador_negra-1]))#
447                     Este fue el recorrido
448                     que se escogio de manera
449                     aleatoria para blanca
```



```
431         print("La ruta recalculada
432               es: "+recorrido_negra)
433         lista_estados_negra = [int(
434             num) for num in
435             recorrido_negra.split(",
436             ")]
437         contador_negra=1
438         nueva_coordenada_negra =
439             calcular_coordenadas(
440                 lista_estados_negra[
441                     contador_negra])
442         if nueva_coordenada_negra
443             not in posicion_blanca:
444             break
445         turn_step=2
446         if coordenadas_negra in
447             posicion_negra:
448             selection = posicion_negra.
449                 index(coordenadas_negra) #
450                 Seleccionamos las
451                 coordenadas
452         if turn_step == 2:
453             turn_step = 3
454
455         if turn_step==3 and selection !=
456             100:
457             posicion_negra[0] =
458                 nueva_coordenada_negra
459             selection = 100
460             contador_negra += 1
461             coordenadas_negra =
462                 nueva_coordenada_negra
463             turn_step=0
464
465     pygame.display.flip()
466 else:
467     # Sacan piezas negras
468     print("\nSacan piezas negras.")
469     while run:
470         timer.tick(fps)
```

```
456     screen.fill('dark gray')
457     dibujar_tablero()
458     dibujar_piezas()
459     boton_rect = dibujar_boton()
460
461     for event in pygame.event.get():
462         if event.type == pygame.QUIT:
463             run = False
464
465         if event.type == pygame.MOUSEBUTTONDOWN and
466            event.button == 1:
467             mouse_pos = pygame.mouse.get_pos()
468             if boton_rect.collidepoint(mouse_pos): #
469                 Verificar si se hizo clic en el bot n
470                 if turn_step ==4:
471                     turn_step=2
472                     nueva_coordenada_blanca =
473                         calcular_coordenadas(
474                             lista_estados_blanca[contador_blanca
475                             ])
476                     nueva_coordenada_negra =
477                         calcular_coordenadas(
478                             lista_estados_negra[contador_negra])
479
480                 if turn_step <= 1:
481                     #print("Negra")
482                     if nueva_coordenada_negra in
483                         posicion_blanca:
484                         while(1):#Ciclo while, donde
485                             para salir la siguiente
486                             coordenada no pueda ser la
487                             de la colision
488                             #recalculamos ruta
489                             print("Colision detectada
490                                 en recorrido:
491                                 Recalculamos negra.")
492                             ruta_negra=cadena()#
493                                 Solicitamos la cadena
494                                 generada aleatoriamente.
495                                 Para pieza negra.
```

```
480     recorrer_estados_negra(  
        tablaEstados, ruta_negra  
        ,str(lista_estados_negra  
            [contador_negra-1]))  
481     recorrido_negra =  
        seleccionar_recorrido_negra  
        (str(lista_estados_negra  
            [contador_negra-1])) #  
        Este fue el recorrido  
        que se escogio de manera  
        aleatoria para blanca  
482     print("La ruta recalculada  
        es: "+recorrido_negra)  
483     lista_estados_negra = [int(  
        num) for num in  
        recorrido_negra.split(",  
        ")]  
484     contador_negra=1  
485     nueva_coordenada_negra =  
        calcular_coordenadas(  
            lista_estados_negra[  
                contador_negra])  
486     if nueva_coordenada_negra  
        not in posicion_blanca:  
487         break  
488     turn_step=0  
489     if coordenadas_negra in  
        posicion_negra:  
490         selection = posicion_negra.  
            index(coordenadas_negra) #  
        Seleccionamos las  
        coordenadas  
491     if turn_step == 0:  
492         turn_step = 1  
493  
494     if turn_step==1 and selection !=  
        100:  
495         posicion_negra[0] =  
            nueva_coordenada_negra  
496         selection = 100
```

```
497         contador_negra += 1
498         coordenadas_negra =
499             nueva_coordenada_negra
500         turn_step=4
501     if turn_step == 2:
502         #print("Blanca")
503         if nueva_coordenada_blanca in
504             posicion_negra:
505             while(1):
506                 #recalculamos ruta
507                 print("Colision detectada
508                     en recorrido:
509                     Recalculamos blanca")
510                 ruta_blanca=cadena()#
511                 Solicitamos la cadena
512                 generada aleatoriamente.
513                 Para pieza blanca.
514                 recorrer_estados_blanca(
515                     tablaEstados,
516                     ruta_blanca,str(
517                         lista_estados_blanca[
518                             contador_blanca-1]))
519                 recorrido_blanca =
520                     seleccionar_recorrido_blanca
521                     (str(
522                         lista_estados_blanca[
523                             contador_blanca-1]))#
524                 Este fue el recorrido
525                 que se escogio de manera
526                 aleatoria para blanca
527                 print("La ruta recalculada
528                     es: "+recorrido_blanca)
529                 lista_estados_blanca = [int
530                     (num) for num in
531                     recorrido_blanca.split("
532                     ,")]
533                 contador_blanca=1
534                 nueva_coordenada_blanca =
535                     calcular_coordenadas(
536                         lista_estados_blanca[
```

```

513         contador_blanca])
514         if nueva_coordenada_blanca
515             not in posicion_negra:
516                 break
517         turn_step=2
518     if coordenadas_blanca in
519     posicion_blanca:
520         selection = posicion_negra.
521         index(coordenadas_negra) #
522         Seleccionamos las
523         coordenadas
524         if turn_step == 2:
525             turn_step = 3
526
527     if turn_step==3 and selection !=
528     100:
529         posicion_blanca[0] =
530         nueva_coordenada_blanca
531
532         selection = 100
533         contador_blanca += 1
534         coordenadas_blanca =
535         nueva_coordenada_blanca
536         turn_step=0
537
538     pygame.display.flip()
539
540     pygame.quit()

```

5.7. Main5.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```

1 #Teoria de la computacion

```

```

2 #Buscador de palabras
3 #Alumno: Connor Urbano Mendoza
4
5 import random
6 import sys
7 import pygame
8 import random
9 import os
10
11 pygame.init() #Acceso al paquete pygame
12 #Ancho
13 WIDTH = 1000
14 #Altura
15 HEIGHT = 700
16 screen = pygame.display.set_mode((WIDTH,HEIGHT)) #Tamaño
    de ventana a imprimir
17 pygame.display.set_caption('Problema del Ajedrez')
18 font = pygame.font.Font('freesansbold.ttf',20)#Tipo de
    fuente 1 del juego
19 big_font= pygame.font.Font('freesansbold.ttf',50)#Tipo de
    fuente 2 del juego
20 timer = pygame.time.Clock()#velocidad de actualizacion de
    nuestro juego a 60 fps
21 fps=60
22
23 #NFA de estados
24 tablaEstados = {
25     '1' : {'R': {'2','5'}, 'N': '6'},
26     '2' : {'R': {'5','7'}, 'N': {'1','6','3'}},
27     '3' : {'R': {'2','7','4'}, 'N': {'6','8'}},
28     '4' : {'R': '7', 'N': {'3','8'}},
29     '5' : {'R': {'2','10'}, 'N': {'1','6','9'}},
30     '6' : {'R': {'2','5','7','10'}, 'N': {'1','3','9','11'}
        },
31     '7' : {'R': {'2','4','10','12'}, 'N': {'3','6','8','11'}
        },
32     '8' : {'R': {'4','7','12'}, 'N': {'3','11'}},
33     '9' : {'R': {'5','10','13'}, 'N': {'6','14'}},
34     '10' : {'R': {'5','7','13','15'}, 'N': {'6','9','11','
        14'}}},

```

```

35     '11' : {'R': {'7', '10', '12', '15'}, 'N': {'6', '8', '14', '
        16'}},
36     '12' : {'R': {'7', '15'}, 'N': {'8', '11', '16'}},
37     '13' : {'R': '10', 'N': {'9', '14'}},
38     '14' : {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
39     '15' : {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
40     '16' : {'R': {'12', '15'}, 'N': '11'}#Estado 16 es el
        estado Final.
41 }
42
43 #Variables e imagenes del juego
44 pieza_blanca = ['king']
45 pieza_negra = ['king']
46 posicion_blanca = [(235,85)]
47 posicion_negra = [(683,85)]
48
49 #
50 turn_step = 0
51 selection= 100
52 valid_moves_for1 =[]
53 #Cargar imagenes en juego
54 rey_blanco = pygame.image.load('C:\\Users\\soyco\\OneDrive
        \\Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
        white_king.png')
55 rey_blanco = pygame.transform.scale(rey_blanco, (80,80))
56 rey_negro = pygame.image.load('C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\img\\
        black_king.png')
57 rey_negro = pygame.transform.scale(rey_negro, (80,80))
58
59 imagen_blanca = [rey_blanco]
60 imagen_negra=[rey_negro]
61
62 lista_piezas = ['king']
63 #ver variables/contador flash
64
65
66 boton_presionado = False
67
68 def dibujar_boton():

```

```

69     boton_width = 150
70     boton_height = 45
71     boton_x = (WIDTH - boton_width) // 2
72     boton_y = (HEIGHT - boton_height - 20)+17
73
74     # Dibujar el bot n como un rect ngulo en la pantalla
75     boton_rect=pygame.Rect(boton_x, boton_y, boton_width,
76                             boton_height)
77     pygame.draw.rect(screen, (0, 255, 0), (boton_x, boton_y
78         , boton_width, boton_height))
79     texto = font.render("Siguiente", True, (0, 0, 0))
80     texto_rect = texto.get_rect(center=(boton_x +
81         boton_width // 2, boton_y + boton_height // 2))
82     screen.blit(texto, texto_rect)
83     return boton_rect
84
85 #Funcion para dibujar tablero
86 def dibujar_tablero():
87     cuadro_size = 150 # Tama o de cada cuadro del tablero
88     tablero_width = 4 * cuadro_size # Ancho total del
89         tablero
90     tablero_height = 4 * cuadro_size # Altura total del
91         tablero
92     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
93         para centrar el tablero
94     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
95         Y para centrar el tablero
96
97     numero_color = 'white' # Color del n mero de casilla
98
99     font = pygame.font.Font(None, 24) # Fuente y tama o
100         del n mero de casilla
101
102     for i in range(16): # Iterar 16 veces para un tablero
103         de 4x4
104         columna = i % 4
105         fila = i // 4
106         x = tablero_x + columna * cuadro_size
107         y = tablero_y + fila * cuadro_size
108         if fila % 2 == 0:

```



```

100         color = 'black' if columna % 2 == 0 else 'red'
101     else:
102         color = 'red' if columna % 2 == 0 else 'black'
103     pygame.draw.rect(screen, color, [x, y, cuadro_size,
104                                cuadro_size])
105     pygame.draw.rect(screen, 'white', [x, y,
106                                cuadro_size, cuadro_size], 2) # Agregar borde
107                                # de color blanco
108     numero_texto = font.render(str(i + 1), True,
109                                numero_color) # Crear superficie de texto con
110                                # el n mero
111     numero_rect = numero_texto.get_rect(center=((x +
112                                cuadro_size // 2)-60, y + 20)) # Posici n del
113                                # n mero en la parte superior del recuadro
114     screen.blit(numero_texto, numero_rect) # Pegar el
115                                # n mero en la pantalla
116
117 #Funcion para dibujar piezas
118 def dibujar_piezas():
119     index=lista_piezas.index('king')
120     if pieza_blanca[0]=='king':
121         screen.blit(imagen_blanca[index], (posicion_blanca
122                                [0][0],posicion_blanca[0][1]))
123     if pieza_negra[0]=='king':
124         screen.blit(imagen_negra[index], (posicion_negra
125                                [0][0],posicion_negra[0][1]))
126
127 def recorrer_estados_blanca(tabla_estados, cadena,
128                             estadoInicial):
129     # Funci n para obtener todos los recorridos posibles
130     # blanca
131     def obtener_recorridos(estado_actual,
132                             simbolos_restantes, recorrido_actual):
133         if not simbolos_restantes:
134             recorridos2.append(recorrido_actual)
135             return
136
137         simbolo = simbolos_restantes[0]
138         if estado_actual in tabla_estados and simbolo in
139             tabla_estados[estado_actual]:

```

```
126         transiciones = tabla_estados[estado_actual][
127             simbolo]
128
129         for estado_siguiente in transiciones:
130             obtener_recorridos(estado_siguiente,
131                 simbolos_restantes[1:], recorrido_actual
132                 + [estado_siguiente])
133
134     # Funci n para obtener los recorridos v lidos hasta
135     # el estado final
136
137     def obtener_recorridos_finales(estado_actual,
138         simbolos_restantes, recorrido_actual):
139         if estado_actual == '16':
140             recorridos_finales.append(recorrido_actual)
141             return
142
143         if not simbolos_restantes:
144             return
145
146         simbolo = simbolos_restantes[0]
147         if estado_actual in tabla_estados and simbolo in
148             tabla_estados[estado_actual]:
149             transiciones = tabla_estados[estado_actual][
150                 simbolo]
151
152             for estado_siguiente in transiciones:
153                 obtener_recorridos_finales(estado_siguiente
154                     , simbolos_restantes[1:],
155                     recorrido_actual + [estado_siguiente])
156
157     # Obtener recorridos posibles
158     recorridos2 = []
159     obtener_recorridos(estadoInicial, cadena, [
160         estadoInicial])
161
162     # Obtener recorridos hasta el estado final
163     recorridos_finales = []
164     obtener_recorridos_finales(estadoInicial, cadena, [
165         estadoInicial])
166
167     # Guardar los recorridos en archivos de texto
```

```
155     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
        \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
        .txt', 'w') as archivo_recorridos:
156         archivo_recorridos.write('Recorridos posibles:\n')
157         for recorrido in recorridos2:
158             archivo_recorridos.write(','.join(recorrido) +
                '\n')
159
160     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
        \\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_blanca.txt', 'w') as
        archivo_recorridos_finales:
161         for recorrido in recorridos_finales:
162             archivo_recorridos_finales.write(','.join(
                recorrido) + '\n')
163
164
165
166
167 def recorrer_estados_negra(tabla_estados, cadena,
    estadoInicial):
168     # Funci n para obtener todos los recorridos posibles
169     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual):
170         if not simbolos_restantes:
171             recorridos2.append(recorrido_actual)
172             return
173
174         simbolo = simbolos_restantes[0]
175         if estado_actual in tabla_estados and simbolo in
            tabla_estados[estado_actual]:
176             transiciones = tabla_estados[estado_actual][
                simbolo]
177
178             for estado_siguiente in transiciones:
179                 obtener_recorridos(estado_siguiente,
                    simbolos_restantes[1:], recorrido_actual
                    + [estado_siguiente])
180
181     # Funci n para obtener los recorridos v lidos hasta
```

```

    el estado final
182 def obtener_recorridos_finales(estado_actual,
    simbolos_restantes, recorrido_actual):
183     if estado_actual == '13':
184         recorridos_finales.append(recorrido_actual)
185         return
186
187     if not simbolos_restantes:
188         return
189
190     simbolo = simbolos_restantes[0]
191     if estado_actual in tabla_estados and simbolo in
        tabla_estados[estado_actual]:
192         transiciones = tabla_estados[estado_actual][
            simbolo]
193
194         for estado_siguiente in transiciones:
195             obtener_recorridos_finales(estado_siguiente
                , simbolos_restantes[1:],
                recorrido_actual + [estado_siguiente])
196
197 # Obtener recorridos posibles
198 recorridos2 = []
199 obtener_recorridos(estadoInicial, cadena, [
    estadoInicial])
200
201 # Obtener recorridos hasta el estado final
202 recorridos_finales = []
203 obtener_recorridos_finales(estadoInicial, cadena, [
    estadoInicial])
204
205 # Guardar los recorridos en archivos de texto
206 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
    \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_negra.
    txt', 'w') as archivo_recorridos:
207     archivo_recorridos.write('Recorridos posibles:\n')
208     for recorrido in recorridos2:
209         archivo_recorridos.write(','.join(recorrido) +
            '\n')
210
```

```
211     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
        \\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_negra.txt', 'w') as
        archivo_recorridos_finales:
212         for recorrido in recorridos_finales:
213             archivo_recorridos_finales.write(','.join(
                recorrido) + '\\n')
214
215
216 def seleccionar_recorrido_blanca(estadoI):
217     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_finales_blanca.txt"
218     x=1
219     while x==1:
220         with open(ruta_archivo, "r+") as archivo:
221             if os.path.getsize(ruta_archivo) == 0:
222                 print("No existen soluciones que lleguen al
                    estado 16 con la condicion actual. Se
                    recalculara una ruta.\\n")
223                 nuevaruta=cadena()
224                 with open('C:\\Users\\soyco\\OneDrive\\
                    Documents\\ESCOM\\sem4\\Teoria\\P2\\
                    Chess\\output\\ruta_blanca.txt', 'w') as
                    archivo2:
225                     archivo2.write(nuevaruta)
226                 print("La nueva ruta es: "+nuevaruta)
227                 recorrer_estados_blanca(tablaEstados,
                    nuevaruta,estadoI)
228                 print('Se almacenaron las nuevas salidas de
                    los recorridos posibles y los
                    recorridos exitosos en la carpeta output
                    .\\n')
229             else:
230                 x=0
231                 lineas = archivo.readlines()
232
233                 # Seleccionar una l nea aleatoria
234                 recorrido_seleccionado = random.choice(
                    lineas)
```

```
235
236         # Eliminar los espacios en blanco y saltos
           de l nea
237         recorrido_seleccionado =
           recorrido_seleccionado.strip()
238
239     return recorrido_seleccionado
240
241 def seleccionar_recorrido_negra(estadoI):
242     ruta_archivo = "C:\\Users\\soyco\\OneDrive\\Documents\\
           ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
           recorridos_finales_negra.txt"
243     x=1
244     while x==1:
245         with open(ruta_archivo, "r+") as archivo:
246             if os.path.getsize(ruta_archivo) == 0:
247                 print("No existen soluciones que lleguen al
                   estado 13 con la condicion actual. Se
                   recalculara una ruta.\n")
248                 nuevaruta=cadena()
249                 with open('C:\\Users\\soyco\\OneDrive\\
                   Documents\\ESCOM\\sem4\\Teoria\\P2\\
                   Chess\\output\\ruta_negra.txt', 'w') as
                   archivo2:
250                     archivo2.write(nuevaruta)
251                 print("La nueva ruta es: "+nuevaruta)
252                 recorrer_estados_negra(tablaEstados,
                   nuevaruta,estadoI)
253                 print('Se almacenaron las nuevas salidas de
                   los recorridos posibles y los
                   recorridos exitosos en la carpeta output
                   .\n')
254             else:#16
255                 x=0
256                 lineas = archivo.readlines()
257
258                 # Seleccionar una l nea aleatoria
259                 recorrido_seleccionado = random.choice(
                   lineas)
260
```

```

261         # Eliminar los espacios en blanco y saltos
           de l nea
262         recorrido_seleccionado =
           recorrido_seleccionado.strip()
263
264     return recorrido_seleccionado
265
266 def cadenaRandom(numero): #Genera un string de forma random
267     auxiliar = '' #Variable auxiliar
268     for i in range(numero):
269         x = random.randint(1, 2) #Funci n para generar un
           resultado random de una lista
270         if x % 2 == 0:
271             auxiliar = auxiliar + "R"
272         else:
273             auxiliar = auxiliar + "N"
274     return auxiliar
275
276 def cadena():
277     numero = random.randint(4,10)
278     print('\nTamaño de cadena escogido aleatoriamente [' +
           str(numero) + ']\nTambien se generaran las
           transiciones R y N aleatoriamente.\n')
279     cad = cadenaRandom(numero) #Se genera de forma
           aleatoria del 1-10
280     print('\nLa cadena o ruta a seguir escogida
           aleatoriamente sera: ' + cad + '\n')
281     return cad
282
283 def cadena_manual():
284     while 1:
285         ruta = input("Ingrese la cadena usando las
           transiciones R y N.\n").upper()
286         if len(ruta) > 10:
287             print("\nNo es posible introducir m s de 10
           caracteres en la cadena de recorrido para
           animacion. Desea seguir de igual forma?(Ya
           no se animara, se procedera al arbol)")
288             respuesta = int(input("1. Si.\n2. No.\n"))
289             if respuesta == 1:

```

```

290         break
291     else:
292         pass
293     else:
294         break
295     print('\nTamaño de cadena escogido [' + str(len(ruta)) +
296           '].')
297     print('\nLa cadena o ruta a seguir escogida sera: ' +
298           ruta + '\n')
299     return ruta
300
301 def calcular_coordenadas(estado):
302     cuadro_size = 150 # Tama o de cada cuadro del tablero
303     tablero_width = 4 * cuadro_size # Ancho total del
304     tablero
305     tablero_height = 4 * cuadro_size # Altura total del
306     tablero
307     tablero_x = (WIDTH - tablero_width) // 2 # Posici n X
308     para centrar el tablero
309     tablero_y = (HEIGHT - tablero_height) // 2 # Posici n
310     Y para centrar el tablero
311     fila = (estado - 1) // 4 # Calcular la fila del estado
312     columna = (estado - 1) % 4 # Calcular la columna del
313     estado
314     x = tablero_x + columna * cuadro_size # Calcular la
315     coordenada X del estado
316     y = tablero_y + fila * cuadro_size # Calcular la
317     coordenada Y del estado
318     return ((x+33), y+33) # Devolver las coordenadas como
319     una tupla
320
321 #def dibujar_movimientos():
322
323 #Main del ciclo del juego 5
324
325 # Verificar el n mero aleatorio y determinar si se sacan
326 piezas blancas o negras
327 numero_aleatorio = random.randint(1, 2)

```



```
319 # Escogemos quien sera pieza negra y quien la blanca
320 print('Establezcan presencialmente quien sera la pieza
      blanca o negra.')
```

```
321
322 while 1:
323     bandera_ruta=0#Bandera para arbol de mas de 10.
324     #Guardamos rutas para arbol.
325     print("\n--PARA PIEZAS BLANCA--")
326     ruta_blanca=cadena_manual()
327     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
              \\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt',
              'w') as archivo:
328         archivo.write(ruta_blanca)
329     if len(ruta_blanca)> 10:
330         bandera_ruta=1
331     else:
332         bandera_ruta=0
333         recorrer_estados_blanca(tablaEstados, ruta_blanca,"
              1")
334
335     print("\n--PARA CADENA NEGRA--")
336     ruta_negra=cadena_manual()
337     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
              \\sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.txt',
              'w') as archivo:
338         archivo.write(ruta_negra)
339     if len(ruta_negra)> 10:
340         if bandera_ruta==1:
341             print("Ambas rutas son mayores a 10, se procede
              al arbol sin llegar a la animacion.")
342             sys.exit(3)
343         else:
344             print("La ruta blanca es menor a 10, y la negra
              mayor a 10, por lo que se procede a
              preguntar de nuevo las rutas.")
345             input("Presione ENTER para continuar")
346             os.system('cls' if os.name == 'nt' else 'clear'
              )
347     else:
348         if bandera_ruta==0:
```

```
349         #ambas rutas menores a 10, se animan.
350         recorrer_estados_negra(tablaEstados, ruta_negra
351             , "4")
352         break
353     else:
354         print("La ruta negra es menor a 10, y la blanca
355             mayor a 10, por lo que se procede a
356             preguntar de nuevo las rutas.")
357         input("Presione ENTER para continuar")
358         os.system('cls' if os.name == 'nt' else 'clear'
359             )
360
361 print('\nEn este punto el programa almaceno en la salidas
362     los recorridos posibles y los recorridos exitosos en la
363     carpeta output.\n')
364 recorrido_blanca = seleccionar_recorrido_blanca("1") #Este
365     fue el recorrido que se escogio de manera aleatoria para
366     blanca
367 recorrido_negra = seleccionar_recorrido_negra("4") #Este fue
368     el recorrido que se escogio de manera aleatoria para
369     negra
370
371 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
372     sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.txt', 'w')
373     as archivo:
374     archivo.write(ruta_blanca)
375 with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM\\
376     sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.txt', 'w')
377     as archivo:
378     archivo.write(ruta_negra)
379
380 print("\nRecorrido seleccionado para blanca:",
381     recorrido_blanca)
382 print("\nRecorrido seleccionado para negra:",
383     recorrido_negra)
384 lista_estados_blanca = [int(num) for num in
385     recorrido_blanca.split(",")]
386 lista_estados_negra = [int(num) for num in recorrido_negra.
387     split(",")]
```

```
371
372
373 coordenadas_blanca=(235,85)
374 nueva_coordenada_blanca=(0,0)
375 coordenadas_negra=(683,85)
376 nueva_coordenada_negra=(0,0)
377 run=True
378 contador_blanca=1
379 contador_negra=1
380
381 if numero_aleatorio == 1:
382     print("\nSacar piezas blancas.")
383     # Sacar piezas blancas
384     while run:
385         timer.tick(fps)
386         screen.fill('dark gray')
387         dibujar_tablero()
388         dibujar_piezas()
389         boton_rect = dibujar_boton()
390
391         for event in pygame.event.get():
392             if event.type == pygame.QUIT:
393                 run = False
394
395             if event.type == pygame.MOUSEBUTTONDOWN and
396                 event.button == 1:
397                 mouse_pos = pygame.mouse.get_pos()
398                 if boton_rect.collidepoint(mouse_pos): #
399                     # Verificar si se hizo clic en el bot n
400                     if turn_step ==4:
401                         turn_step=2
402                         nueva_coordenada_blanca =
403                             calcular_coordenadas(
404                                 lista_estados_blanca[contador_blanca
405 ])
```

```
404     #print("Blanca")
405     if nueva_coordenada_blanca in
406         posicion_negra:
407         while(1):#Ciclo while, donde
408             para salir la siguiente
409             coordenada no pueda ser la
410             de la colision
411             #recalculamos ruta
412             print("Colision detectada
413                 en recorrido:
414                 Recalculamos blanca")
415             ruta_blanca=cadena()#
416             Solicitamos la cadena
417             generada aleatoriamente.
418             Para pieza blanca.
419             recorrer_estados_blanca(
420                 tablaEstados,
421                 ruta_blanca,str(
422                     lista_estados_blanca[
423                         contador_blanca-1]))
424             recorrido_blanca =
425                 seleccionar_recorrido_blanca
426                 (str(
427                     lista_estados_blanca[
428                         contador_blanca-1]))#
429             Este fue el recorrido
430             que se escogio de manera
431             aleatoria para blanca
432             print("La ruta recalculada
433                 es: "+recorrido_blanca)
434             lista_estados_blanca = [int
435                 (num) for num in
436                 recorrido_blanca.split("
437                     ,")]
438             contador_blanca=1
439             nueva_coordenada_blanca =
440                 calcular_coordenadas(
441                     lista_estados_blanca[
442                         contador_blanca])
443             if nueva_coordenada_blanca
```

```

417         not in posicion_negra:
418             break
419         turn_step=0
420     if coordenadas_blanca in
posicion_blanca:
421         selection = posicion_blanca.
index(coordenadas_blanca)
422         if turn_step == 0:
423             turn_step = 1
424
if turn_step==1 and selection !=
100:
425     posicion_blanca[0] =
nueva_coordenada_blanca
426
427     selection = 100
428     contador_blanca += 1
429     coordenadas_blanca =
nueva_coordenada_blanca
430     turn_step=4
431 if turn_step == 2:
432     #print("Negra")
433     if nueva_coordenada_negra in
posicion_blanca:
434         while(1):
435             #recalculamos ruta
436             print("Colision detectada
en recorrido:
Recalculamos negra.")
437             ruta_negra=cadena()#
Solicitamos la cadena
generada aleatoriamente.
Para pieza negra.
438             recorrer_estados_negra(
tablaEstados, ruta_negra
,str(lista_estados_negra
[contador_negra-1]))
439             recorrido_negra =
seleccionar_recorrido_negra
(str(lista_estados_negra

```

```
440         [contador_negra-1])) #
441         Este fue el recorrido
442         que se escogio de manera
443         aleatoria para blanca
444     print("La ruta recalculada
445           es: "+recorrido_negra)
446     lista_estados_negra = [int(
447         num) for num in
448         recorrido_negra.split(",
449         ")]
450     contador_negra=1
451     nueva_coordenada_negra =
452         calcular_coordenadas(
453         lista_estados_negra[
454         contador_negra])
455     if nueva_coordenada_negra
456         not in posicion_blanca:
457         break
458     turn_step=2
459     if coordenadas_negra in
460         posicion_negra:
461         selection = posicion_negra.
462             index(coordenadas_negra) #
463         Seleccionamos las
464         coordenadas
465     if turn_step == 2:
466         turn_step = 3
467
468     if turn_step==3 and selection !=
469     100:
470         posicion_negra[0] =
471             nueva_coordenada_negra
472         selection = 100
473         contador_negra += 1
474         coordenadas_negra =
475             nueva_coordenada_negra
476         turn_step=0
477
478     pygame.display.flip()
479 else:
```



```
488     ruta_negra=cadena()#  
         Solicitamos la cadena  
         generada aleatoriamente.  
         Para pieza negra.  
489     recorrer_estados_negra(  
         tablaEstados, ruta_negra  
         ,str(lista_estados_negra  
         [contador_negra-1]))  
490     recorrido_negra =  
         seleccionar_recorrido_negra  
         (str(lista_estados_negra  
         [contador_negra-1]))#  
         Este fue el recorrido  
         que se escogio de manera  
         aleatoria para blanca  
491     print("La ruta recalculada  
         es: "+recorrido_negra)  
492     lista_estados_negra = [int(  
         num) for num in  
         recorrido_negra.split(",  
         ")]  
493     contador_negra=1  
494     nueva_coordenada_negra =  
         calcular_coordenadas(  
         lista_estados_negra[  
         contador_negra])  
495     if nueva_coordenada_negra  
         not in posicion_blanca:  
496         break  
497     turn_step=0  
498     if coordenadas_negra in  
         posicion_negra:  
499         selection = posicion_negra.  
         index(coordenadas_negra)#  
         Seleccionamos las  
         coordenadas  
500         if turn_step == 0:  
501             turn_step = 1  
502  
503     if turn_step==1 and selection !=
```



```
100:
504     posicion_negra[0] =
        nueva_coordenada_negra
505     selection = 100
506     contador_negra += 1
507     coordenadas_negra =
        nueva_coordenada_negra
508     turn_step=4
509     if turn_step == 2:
510         #print("Blanca")
511         if nueva_coordenada_blanca in
            posicion_negra:
512             while(1):
513                 #recalculamos ruta
514                 print("Colision detectada
                    en recorrido:
                    Recalculamos blanca")
515                 ruta_blanca=cadena()#
                    Solicitamos la cadena
                    generada aleatoriamente.
                    Para pieza blanca.
516                 recorrer_estados_blanca(
                    tablaEstados,
                    ruta_blanca,str(
                    lista_estados_blanca[
                    contador_blanca-1]))
517                 recorrido_blanca =
                    seleccionar_recorrido_blanca
                    (str(
                    lista_estados_blanca[
                    contador_blanca-1]))#
                    Este fue el recorrido
                    que se escogio de manera
                    aleatoria para blanca
518                 print("La ruta recalculada
                    es: "+recorrido_blanca)
519                 lista_estados_blanca = [int
                    (num) for num in
                    recorrido_blanca.split("
                    ,")]
```

```
520         contador_blanca=1
521         nueva_coordenada_blanca =
            calcular_coordenadas(
                lista_estados_blanca[
                    contador_blanca])
522         if nueva_coordenada_blanca
            not in posicion_negra:
523             break
524         turn_step=2
525     if coordenadas_blanca in
        posicion_blanca:
526         selection = posicion_negra.
            index(coordenadas_negra) #
            Seleccionamos las
            coordenadas
527         if turn_step == 2:
528             turn_step = 3
529
530     if turn_step==3 and selection !=
        100:
531         posicion_blanca[0] =
            nueva_coordenada_blanca
532
533         selection = 100
534         contador_blanca += 1
535         coordenadas_blanca =
            nueva_coordenada_blanca
536         turn_step=0
537
538     pygame.display.flip()
539
540 pygame.quit()
```

5.8. graficador.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1 import sys
2 from graphviz import Digraph
3
4 # Leer el argumento pasado desde el programa principal
5 x=0
6 if len(sys.argv) > 1:
7     x=((sys.argv[1]))
8     #print((x))
9 else:
10     print("No se pas ning n argumento.")
11     sys.exit(1)
12
13 def recorrer_estados_blanca(tabla_estados, cadena):
14     def obtener_recorridos(estado_actual,
15                             simbolos_restantes, recorrido_actual,
16                             archivo_recorridos):
17         if not simbolos_restantes:
18             archivo_recorridos.write(','.join(
19                 recorrido_actual) + '\n')
20             return
21
22         simbolo = simbolos_restantes[0]
23         if estado_actual in tabla_estados and simbolo in
24             tabla_estados[estado_actual]:
25             transiciones = tabla_estados[estado_actual][
26                 simbolo]
27
28             for estado_siguiente in transiciones:
29                 obtener_recorridos(
30                     estado_siguiente,
31                     simbolos_restantes[1:],
32                     recorrido_actual + [estado_siguiente],
33                     archivo_recorridos
34                 )
35
36     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
37             \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
38             .txt', 'w') as archivo_recorridos:
```

```
32         obtener_recorridos('1', cadena, ['1'],
33                               archivo_recorridos)
34 def recorrer_estados_negra(tabla_estados, cadena):
35     def obtener_recorridos2(estado_actual,
36                             simbolos_restantes, recorrido_actual,
37                             archivo_recorridos):
38         if not simbolos_restantes:
39             archivo_recorridos.write(','.join(
40                 recorrido_actual) + '\n')
41             return
42
43         simbolo = simbolos_restantes[0]
44         if estado_actual in tabla_estados and simbolo in
45             tabla_estados[estado_actual]:
46             transiciones = tabla_estados[estado_actual][
47                 simbolo]
48
49             for estado_siguiente in transiciones:
50                 obtener_recorridos2(
51                     estado_siguiente,
52                     simbolos_restantes[1:],
53                     recorrido_actual + [estado_siguiente],
54                     archivo_recorridos
55                 )
56
57     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
58             \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_negra.
59             txt', 'w') as archivo_recorridos:
60         obtener_recorridos2('4', cadena, ['4'],
61                               archivo_recorridos)
62
63 def agregar_recorrido_al_grafo(grafo, transicion,
64                               transiciones_posibles):
65     estados = transicion.split(',')
66
67     for i in range(len(estados) - 1):
68         origen = estados[i]
69         destino = estados[i+1]
70         if [origen, destino] not in transiciones_posibles:
```

```

62         # Obtener el simbolo asociado a la transición
63         if origen in tablaEstados and 'R' in
            tablaEstados[origen] and destino in
            tablaEstados[origen]['R']:
64             simbolo = 'R'
65         elif origen in tablaEstados and 'N' in
            tablaEstados[origen] and destino in
            tablaEstados[origen]['N']:
66             simbolo = 'N'
67         else:
68             simbolo = ''
69         #Parte donde se calcula el simbolo asociado a
            la transición o arista/flecha, recuerda que
            puede ser N o R dependiendo de la tabla de
            estados.
70         grafo.edge(origen, destino, arrowhead="normal",
            label=simbolo) # Utilizamos el método '
            edge' para agregar una arista
71         transiciones_posibles.append([origen, destino
            ],)
72
73
74 def generar_arbol_recorridos(tabla_estados, ruta_evaluar,
            archivo_recorridos, archivo_salida):
75     grafo = Digraph('G', filename='arbol.gv')
76     grafo.attr(rankdir='LR', size='8,5')
77     grafo.graph_attr['label'] = ruta_evaluar # Agregar
            título al gráfico
78     with open(archivo_recorridos, 'r') as file:
79         for linea in file:
80             recorrido = linea.strip()
81             agregar_recorrido_al_grafo(grafo, recorrido,
                transiciones_posibles)
82
83     archivo_salida_pdf = archivo_salida.replace('.dot', '.
            pdf')
84     grafo.render(archivo_salida_pdf, view=True)
85
86 tablaEstados = {
87     '1': {'R': {'2', '5'}, 'N': '6'},

```

```

88     '2': {'R': {'5', '7'}, 'N': {'1', '6', '3'}},
89     '3': {'R': {'2', '7', '4'}, 'N': {'6', '8'}},
90     '4': {'R': '7', 'N': {'3', '8'}},
91     '5': {'R': {'2', '10'}, 'N': {'1', '6', '9'}},
92     '6': {'R': {'2', '5', '7', '10'}, 'N': {'1', '3', '9', '11'
93           }},
94     '7': {'R': {'2', '4', '10', '12'}, 'N': {'3', '6', '8', '11'
95           }},
96     '8': {'R': {'4', '7', '12'}, 'N': {'3', '11'}},
97     '9': {'R': {'5', '10', '13'}, 'N': {'6', '14'}},
98     '10': {'R': {'5', '7', '13', '15'}, 'N': {'6', '9', '11', '14'
99            '}},
100     '11': {'R': {'7', '10', '12', '15'}, 'N': {'6', '8', '14', '
101            16'}},
102     '12': {'R': {'7', '15'}, 'N': {'8', '11', '16'}},
103     '13': {'R': '10', 'N': {'9', '14'}},
104     '14': {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
105     '15': {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
106     '16': {'R': {'12', '15'}, 'N': '11'}
107 }
108
109 if x == "1":
110     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\
111                   ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.
112                   txt'
113     # Abre el archivo en modo de lectura
114     archivo = open(archivo_ruta, "r")
115
116     # Lee una l nea del archivo
117     ruta_evaluar = archivo.readline()
118
119     # Imprime la l nea le da
120     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
121                           Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
122                           recorridos_blanca.txt'
123     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
124                       \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
125                       arbol_blanca.dot'
126     transiciones_posibles = []
127     recorrer_estados_blanca(tablaEstados, ruta_evaluar)

```

```
118     generar_arbol_recorridos(tablaEstados, ruta_evaluar,
119                               archivo_recorridos, archivo_salida)
120 else:
121     #Redenrizmos ruta blanca
122     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.
        txt'
123     # Abre el archivo en modo de lectura
124     archivo = open(archivo_ruta, "r")
125
126     # Lee una l nea del archivo
127     ruta_evaluar = archivo.readline()
128
129     # Imprime la l nea le da
130     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_blanca.txt'
131     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
        \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        arbol_blanca.dot'
132     transiciones_posibles = []
133     recorrer_estados_blanca(tablaEstados, ruta_evaluar)
134     generar_arbol_recorridos(tablaEstados, ruta_evaluar,
        archivo_recorridos, archivo_salida)
135
136     #Renderizamos ruta negra
137     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.
        txt'
138     # Abre el archivo en modo de lectura
139     archivo = open(archivo_ruta, "r")
140
141     # Lee una l nea del archivo
142     ruta_evaluar = archivo.readline()
143
144     # Imprime la l nea le da
145     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_negra.txt'
```

```
146     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
        \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        arbol_negra.dot'
147     transiciones_posibles = []
148     recorrer_estados_negra(tablaEstados, ruta_evaluar)
149     generar_arbol_recorridos(tablaEstados, ruta_evaluar,
        archivo_recorridos, archivo_salida)
```

5.9. graficador_{grande}.py

El código main solo es el programa principal que nos redireccionara a los archivos .py correspondientes a su asignación.

```
1 import sys
2 from graphviz import Digraph
3
4 # Leer el argumento pasado desde el programa principal
5 x=0
6 if len(sys.argv) > 1:
7     x=((sys.argv[1]))
8     #print((x))
9 else:
10     print("No se pas ning n argumento.")
11     sys.exit(1)
12
13 def recorrer_estados_blanca(tabla_estados, cadena):
14     cont=0
15     cont2=0
16     def obtener_recorridos(estado_actual,
        simbolos_restantes, recorrido_actual,
        archivo_recorridos):
17         nonlocal cont # Declarar 'cont' como una variable
            no local
18         nonlocal cont2 # Declarar 'cont' como una variable
            no local
```



```

19         if not simbolos_restantes:
20             cont=cont+1
21             archivo_recorridos.write(','.join(
22                 recorrido_actual) + '.')
23             if cont==1000:
24                 archivo_recorridos.write('\n')
25                 cont2=cont+cont2
26                 cont=0
27             return
28         simbolo = simbolos_restantes[0]
29         if estado_actual in tabla_estados and simbolo in
30             tabla_estados[estado_actual]:
31             transiciones = tabla_estados[estado_actual][
32                 simbolo]
33
34             for estado_siguiente in transiciones:
35                 obtener_recorridos(estado_siguiente,
36                     simbolos_restantes[1:], recorrido_actual
37                     + [estado_siguiente], archivo_recorridos)
38             if cont2==100000:
39                 #input("100 cien mil pos")
40                 break
41
42         with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
43             \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_blanca
44             .txt', 'w') as archivo_recorridos:
45             obtener_recorridos('1', cadena, ['1'],
46                 archivo_recorridos)
47
48 def recorrer_estados_negra(tabla_estados, cadena):
49     def obtener_recorridos2(estado_actual,
50         simbolos_restantes, recorrido_actual,
51         archivo_recorridos):
52         if not simbolos_restantes:
53             archivo_recorridos.write(','.join(
54                 recorrido_actual) + '.')
55             return
56
57         simbolo = simbolos_restantes[0]
58         if estado_actual in tabla_estados and simbolo in

```

```

48         tabla_estados[estado_actual]:
           transiciones = tabla_estados[estado_actual][
               simbolo]
49
50         for estado_siguiente in transiciones:
51             obtener_recorridos2(estado_siguiente,
                                   simbolos_restantes[1:], recorrido_actual
                                   + [estado_siguiente], archivo_recorridos)
52
53     with open('C:\\Users\\soyco\\OneDrive\\Documents\\ESCOM
               \\sem4\\Teoria\\P2\\Chess\\output\\recorridos_negra.
               txt', 'w') as archivo_recorridos:
54         obtener_recorridos2('4', cadena, ['4'],
                               archivo_recorridos)
55
56 def agregar_recorrido_al_grafo(grafo, transicion,
    transiciones_posibles):
57     estados = transicion.split(',')
58     for i in range(len(estados) - 1):
59         origen = estados[i]
60         destino = estados[i+1]
61         if [origen, destino] not in transiciones_posibles:
62             # Obtener el simbolo asociado a la transici n
63             if origen in tablaEstados and 'R' in
               tablaEstados[origen] and destino in
               tablaEstados[origen]['R']:
64                 simbolo = 'R'
65             elif origen in tablaEstados and 'N' in
               tablaEstados[origen] and destino in
               tablaEstados[origen]['N']:
66                 simbolo = 'N'
67             else:
68                 simbolo = ''
69             #Parte donde se calcula el simbolo asociado a
               la trancicion o arista/flecha, recuerda que
               puede ser N o R dependiendo de la tabla de
               estados.
70         grafo.edge(origen, destino, arrowhead="normal",
                       label=simbolo) # Utilizamos el m todo '
               edge' para agregar una arista

```

```

71         transiciones_posibles.append([origen, destino
72                                     ],)
73
74 def generar_arbol_recorridos(tabla_estados, ruta_evaluar,
75                             archivo_recorridos, archivo_salida):
76     grafo = Digraph('G', filename=archivo_salida)
77     grafo.attr(rankdir='LR', size='8,5')
78     grafo.graph_attr['label'] = ruta_evaluar # Agregar
79         titulo al grafico
80     # Leer los recorridos desde el archivo
81     with open(archivo_recorridos, 'r') as archivo:
82         # Agregar los recorridos al grafo
83         transiciones_posibles = []
84         for line in archivo:
85             linea_aux = line.strip()
86
87             for linea_aux in archivo:
88                 recorridos = linea_aux.strip().split('.')
89                 for recorrido in recorridos:
90                     if recorrido:
91                         agregar_recorrido_al_grafo(grafo,
92                                                     recorrido, transiciones_posibles
93                                                     )
94
95     # Renderizar el grafo y guardar la imagen
96     archivo_salida_pdf = archivo_salida.replace('.dot', '.
97     pdf')
98     grafo.render(archivo_salida_pdf, view=True)
99
100
101 tablaEstados = {
102     '1': {'R': {'2', '5'}, 'N': '6'},
103     '2': {'R': {'5', '7'}, 'N': {'1', '6', '3'}},
104     '3': {'R': {'2', '7', '4'}, 'N': {'6', '8'}},
105     '4': {'R': '7', 'N': {'3', '8'}},
106     '5': {'R': {'2', '10'}, 'N': {'1', '6', '9'}},
107     '6': {'R': {'2', '5', '7', '10'}, 'N': {'1', '3', '9', '11'}
108         },

```

```

104     '7': {'R': {'2', '4', '10', '12'}, 'N': {'3', '6', '8', '11'}
105         },
106     '8': {'R': {'4', '7', '12'}, 'N': {'3', '11'}},
107     '9': {'R': {'5', '10', '13'}, 'N': {'6', '14'}},
108     '10': {'R': {'5', '7', '13', '15'}, 'N': {'6', '9', '11', '14'}
109            },
110     '11': {'R': {'7', '10', '12', '15'}, 'N': {'6', '8', '14', '16'}},
111     '12': {'R': {'7', '15'}, 'N': {'8', '11', '16'}},
112     '13': {'R': {'10'}, 'N': {'9', '14'}},
113     '14': {'R': {'13', '10', '15'}, 'N': {'9', '11'}},
114     '15': {'R': {'10', '12'}, 'N': {'11', '14', '16'}},
115     '16': {'R': {'12', '15'}, 'N': {'11'}}
116 }
117
118 if x == "1":
119     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\
120                  ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.
121                  txt'
122     # Abre el archivo en modo de lectura
123     archivo = open(archivo_ruta, "r")
124
125     # Lee una l nea del archivo
126     ruta_evaluar = archivo.readline()
127
128     # Imprime la l nea le da
129     archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
130                        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
131                        recorridos_blanca.txt'
132     archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
133                    \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
134                    arbol_blanca.dot'
135     transiciones_posibles = []
136     recorrer_estados_blanca(tablaEstados, ruta_evaluar)
137     generar_arbol_recorridos(tablaEstados, ruta_evaluar,
138                             archivo_recorridos, archivo_salida)
139
140 else:
141     # Redenrizmos ruta blanca
142     archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\

```

```
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_blanca.
        txt'
134 # Abre el archivo en modo de lectura
135 archivo = open(archivo_ruta, "r")
136
137 # Lee una l nea del archivo
138 ruta_evaluar = archivo.readline()
139
140 # Imprime la l nea le da
141 archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_blanca.txt'
142 archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
        \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        arbol_blanca.dot'
143 transiciones_posibles = []
144 recorrer_estados_blanca(tablaEstados, ruta_evaluar)
145 generar_arbol_recorridos(tablaEstados, ruta_evaluar,
        archivo_recorridos, archivo_salida)
146
147 #Renderizamos ruta negra
148 archivo_ruta = 'C:\\Users\\soyco\\OneDrive\\Documents\\
        ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\ruta_negra.
        txt'
149 # Abre el archivo en modo de lectura
150 archivo = open(archivo_ruta, "r")
151
152 # Lee una l nea del archivo
153 ruta_evaluar = archivo.readline()
154
155 # Imprime la l nea le da
156 archivo_recorridos = 'C:\\Users\\soyco\\OneDrive\\
        Documents\\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        recorridos_negra.txt'
157 archivo_salida = 'C:\\Users\\soyco\\OneDrive\\Documents
        \\ESCOM\\sem4\\Teoria\\P2\\Chess\\output\\
        arbol_negra.dot'
158 transiciones_posibles = []
159 recorrer_estados_negra(tablaEstados, ruta_evaluar)
160 generar_arbol_recorridos(tablaEstados, ruta_evaluar,
```

```
archivo_recorridos, archivo_salida)
```