

## 4.- Automata finito no determinista (Juego de tablero)

---

Felipe Sánchez Martínez  
Escuela Superior de Cómputo, IPN

Septiembre de 2023



<b>1. Planteamiento del problema.</b>	<b>1</b>
<b>2. Marco Teórico.</b>	<b>3</b>
2.0.1. Autómatas Finitos No Deterministas (AFND) . . . . .	3
<b>3. Desarrollo del problema.</b>	<b>5</b>
3.0.1. Approach. . . . .	5
3.0.2. Complejidad . . . . .	8
3.0.3. Código fuente. . . . .	8
3.0.4. Casos de prueba. . . . .	21
<b>4. Conclusión</b>	<b>51</b>
<b>5. Referencias.</b>	<b>53</b>



# CAPÍTULO 1

---

## Planteamiento del problema.

---

Un autómata finito no determinista tiene la capacidad de estar en varios estados al mismo tiempo, y aunque puede ser representado como un autómata determinista, esta característica nos ayuda a expresar de manera más sencilla un problema dado.

Por ejemplo, el desarrollo de esta práctica tiene como finalidad crear un AFND (Autómata Finito No Determinista) para jugar de forma autónoma en la solución de un tablero de ajedrez.

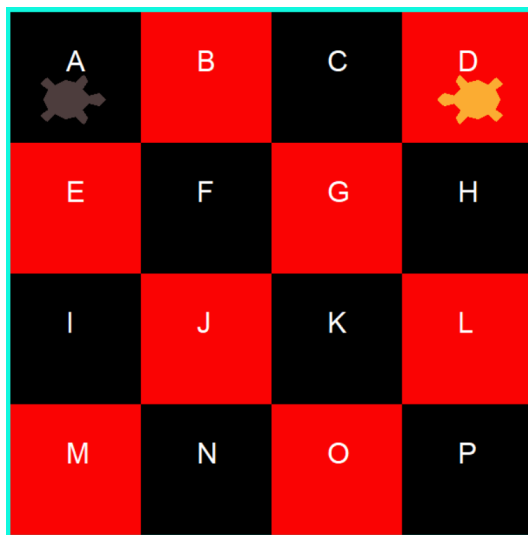


Figura 1.1: Tablero de juego.

Donde cada casilla representa un estado dentro del autómata y puede avanzar en múltiples estados a la vez, ya sea en forma diagonal o lineal. Esto nos proporciona una conectividad de 8 en el centro del tablero 4x4 y varía en los bordes, lo que nos da una red de múltiples estados en los que puede terminar el autómata. Aunque es finito, al final de la secuencia contiene un estado exitoso.

---

Tomaremos como estado inicial de cada autómata una esquina superior, ya sea derecha o izquierda, y el estado final será una de las esquinas inferiores, ya sea izquierda o derecha.

### 2.0.1. Autómatas Finitos No Deterministas (AFND)

Un autómata finito no determinista (AFND) tiene la capacidad de estar en varios estados al mismo tiempo. Esta capacidad se expresa a menudo como la capacidad de *adivinar* algo sobre su entrada. Por ejemplo, cuando se utiliza el autómata para buscar ciertas secuencias de caracteres (por ejemplo, palabras clave) en una cadena de texto larga, es útil *adivinar* que estamos al comienzo de una de esas cadenas y usar una secuencia de estados para verificar que la cadena aparece, carácter por carácter. Veremos un ejemplo de este tipo de aplicación en la Sección 2.4. Antes de examinar las aplicaciones, necesitamos definir autómatas finitos no deterministas y mostrar que cada uno de ellos acepta un lenguaje que también es aceptado por algún AF determinista. Es decir, los AFND aceptan exactamente los lenguajes regulares, al igual que los AF deterministas. Sin embargo, existen razones para pensar en los AFND. A menudo son más concisos y más fáciles de diseñar que los AF deterministas. Además, aunque siempre podemos convertir un AFND en un AF determinista, este último puede tener exponencialmente más estados que el AFND; afortunadamente, los casos de este tipo son raros.

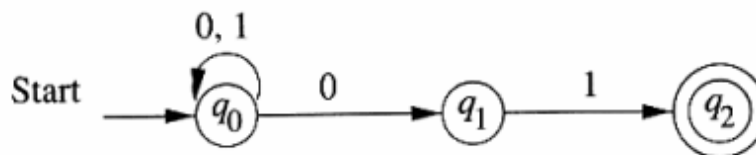


Figura 2.1: NFA.





---

Desarrollo del problema.

---

**3.0.1. Approach.**

Se decidió implementar el algoritmo en dos partes: la creación de la red de todos los estados posibles y el juego del ajedrez.

En la segunda parte del juego, es simple: al inicio, se elige aleatoriamente a un jugador y se declara como el jugador actual, mientras que el otro es el competidor. Se intercambian roles hasta que haya un ganador.

Dado que se trata de un juego por turnos, cuando el jugador actual no pueda moverse con su ruta actual, elegirá otra entre las rutas posibles; en caso de no encontrar ninguna, deberá ceder el turno.

```
1  while not winner:
2      time.sleep(10)
3
4      # wineer break
5      if competitor_index == len(competitor_chosed) - 1:
6          winner = competitor
7          break
8
9      # Move the turtle
10     # "AF" 'BF' 'AF' 'B'
11     # print(curr_chosed, curr_index)
12     # print(curr_chosed[curr_index + 1])
13     if curr_chosed[curr_index + 1] == competitor_path[-1]:
14         # print("-----CHANGE PATH -----")
15         # print('curr path:', curr_chosed)
16
17         # Change Path
18         aux_path = curr_chosed
19         # print( curr_path, competitor_path[-1])
20         curr_chosed = find_path(curr_file, curr_path, competitor_path[-1])
21         # print('new path: ', curr_chosed)
```

```

22         if not curr_chosed:
23             # pass next turn
24             curr_chosed = aux_path
25
26             # change turn
27             curr_file, competitor_file = competitor_file, curr_file
28             curr_player, competitor = competitor, curr_player
29             curr_chosed, competitor_chosed = competitor_chosed, curr_chosed
30             curr_path, competitor_path = competitor_path, curr_path
31             curr_index, competitor_index = competitor_index, curr_index
32             continue
33
34     curr_index += 1
35     curr_path += curr_chosed[curr_index]
36     Move_to_position(curr_player, curr_path[-1], positions)
37
38
39     # change turn
40     curr_file, competitor_file = competitor_file, curr_file
41     curr_player, competitor = competitor, curr_player
42     curr_chosed, competitor_chosed = competitor_chosed, curr_chosed
43     curr_path, competitor_path = competitor_path, curr_path
44     curr_index, competitor_index = competitor_index, curr_index

```

Dentro de la función utilizada para encontrar un camino correcto, se busca con la ayuda de una expresión regular dentro del archivo de rutas posibles; si no existe, se devuelve una lista vacía.

```

1 def find_path(paths, actual_path, without_letter=""):
2     # Leer paths
3     with open(paths, "r") as file:
4         content = file.read()
5
6     pattern = r'\b' + re.escape(actual_path) + r'[' + re.escape(without_letter) +
7         r']\w*'
8
9     good_paths = re.findall(pattern, content)
10
11     if not good_paths:
12         # print("No way man")
13         return None
14     # print(good_paths)
15     return good_paths[0]

```

Finalmente, para poder crear todas las rutas posibles dentro del autómata, utilizamos una representación simple del problema como una serie de grafos.

Los representamos dentro de un tablero 4x4 en una matriz, donde cada letra es un estado en la red. Una vez visto como una matriz y un problema de grafos, podemos hacer una búsqueda de rutas, ya sea BFS o DFS. Se optó por una búsqueda DFS debido al crecimiento exponencial en términos de memoria del problema, ya que una DFS no ocupará mucha memoria RAM de los estados posibles, simplemente encontrará un estado final y lo agregará a las rutas posibles.

```
1 def dfs(r, c, index, path, configurations):
2
3     # Base case
4     if min(r, c) < 0 or max(r, c) > 3 or index > len(configurations) or (r, c)
5         not in color[configurations[index - 1]]:
6             return
7
8     # Recursive case
9     path += board[r][c]
10
11     if len(path) >= len(configurations) + 1:
12         paths.write(path + "\n")
13         return
14
15     # front and back
16     dfs(r - 1, c, index + 1, path, configurations)
17     dfs(r + 1, c, index + 1, path, configurations)
18     dfs(r, c - 1, index + 1, path, configurations)
19     dfs(r, c + 1, index + 1, path, configurations)
20
21     # Diagonals
22     dfs(r - 1, c - 1, index + 1, path, configurations)
23     dfs(r - 1, c + 1, index + 1, path, configurations)
24     dfs(r + 1, c - 1, index + 1, path, configurations)
25     dfs(r + 1, c + 1, index + 1, path, configurations)
26
27     # First player
28
29     dfs(0, 1, 1, "A", player1_configurations)
30     dfs(1, 0, 1, "A", player1_configurations)
31     dfs(1, 1, 1, "A", player1_configurations)
32
33     # Player 2
34     dfs(0, 2, 1, 'D', player2_configurations)
35     dfs(1, 2, 1, 'D', player2_configurations)
36     dfs(1, 3, 1, 'D', player2_configurations)
```

La DFS se implementa generalmente a partir de un estado inicial. Se consideran los casos especiales o situaciones límite y se explora en todas las posiciones posibles.

---

### 3.0.2. Complejidad

#### Complejidad Temporal

La complejidad del algoritmo se divide en dos componentes principales: la generación de los estados posibles y sus posteriores lecturas.

Si nos enfocamos en la búsqueda de rutas posibles de cada autómatas, tenemos una complejidad temporal de:

$$T : O(8^{n \cdot m \cdot k}) = O(8^{4 \cdot 4 \cdot k})$$

donde 'n' y 'm' son las dimensiones del tablero, el número 8 corresponde a la conectividad ocho en la casilla del centro, y el valor de 'k' corresponde a la longitud de la cadena o configuración que se usará.

Por otro lado, al considerar la función del autómatas para leer las rutas y clasificarlas, las tenemos que leer linealmente todas, teniendo una complejidad de:

$$T : O(b)$$

donde 'b' es el número total de rutas.

En conjunto, la complejidad temporal total del algoritmo es:

$$T : O(b) + O(8^{4 \cdot 4 \cdot k}) = O(8^{4 \cdot 4 \cdot k})$$

#### Complejidad Espacial

Dentro del algoritmo, la generación del output no se considera en la complejidad espacial, lo que resulta en una complejidad espacial de:

$$S : O(1)$$

Pero estaremos generando espacio en memoria debido a la llamada recursiva, que podemos ver como:

$$S : O(n \cdot m \cdot k) + O(1) = O(4 \cdot 4 \cdot k)$$

donde 'k' es la longitud de la cadena y 'm' y 'n' son las dimensiones del tablero.

### 3.0.3. Código fuente.

```
1 import random
2 import time
3
4 from math import sqrt
5 # graph libs
6 import pandas as pd
7 import matplotlib.pyplot as plt
8
9 import argparse
10
11 import turtle
12 from PIL import Image
13
14
15 import re
```

```

16 '''
17     start = 0, 0
18     0      1      2      3
19 0      A      [B]    C      [D]
20 1      [E]      F      [G]    H
21 2      I      [J]    K      [L]
22 3      [M]      N      [O]    P
23
24 '''
25
26
27 def main(args):
28     paths = open("paths.txt", "w")
29
30     board = [['A', 'B', 'C', 'D'], ['E', 'F', 'G', 'H'], ['I', 'J', 'K', 'L'],
31              ['M', 'N', 'O', 'P']]
32
33     black_positions = set([(0, 0), (0, 2), (1, 1), (1, 3), (2, 0), (2, 2), (3, 1),
34                           (3, 3)])
35     red_positions = set([(0, 1), (0, 3), (1, 0), (1, 2), (2, 1), (2, 3), (3, 0),
36                          (3, 2)])
37
38     red_squares = ['B', 'D', 'E', 'G', 'J', 'L', 'M', 'O']
39     black_squares = ['A', 'C', 'F', 'H', 'I', 'K', 'N', 'P']
40
41     player1_configurations = input("Enter the string of the player1: ")
42     player2_configurations = input("Enter the string of the player2: ")
43
44     size = random.randint(5, 50)
45
46     if not player1_configurations and not player2_configurations:
47         for _ in range(size):
48             player1_configurations += random.choice(["r", "b"])
49             player2_configurations += random.choice(["r", "b"])
50
51     # Checking if the size of both strings are equal in size
52     if len(player1_configurations) != len(player2_configurations):
53         size = max(len(player1_configurations), len(player2_configurations))
54         if size == len(player1_configurations):
55             for _ in range(size - len(player2_configurations)):
56                 player2_configurations += random.choice(["r", "b"])
57         else:
58             for _ in range(size - len(player1_configurations)):
59                 player1_configurations += random.choice(["r", "b"])
60
61     if player1_configurations[-1] != "b":

```

```

60     player1_configurations = player1_configurations[:-1] + "b"
61
62     if player2_configurations[-1] != "r":
63         player2_configurations = player2_configurations[:-1] + "r"
64
65
66
67
68
69     color = {"r": red_positions, "b": black_positions}
70
71     squares = {"r": red_squares, "b": black_squares}
72
73
74     def dfs(r, c, index, path, configurations):
75
76         # Base case
77         if min(r, c) < 0 or max(r, c) > 3 or index > len(configurations) or (r, c)
78             not in color[configurations[index - 1]]:
79             return
80
81         # Recursive case
82         path += board[r][c]
83
84         if len(path) >= len(configurations) + 1:
85             paths.write(path + "\n")
86             return
87
88         # front and back
89         dfs(r - 1, c, index + 1, path, configurations)
90         dfs(r + 1, c, index + 1, path, configurations)
91         dfs(r, c - 1, index + 1, path, configurations)
92         dfs(r, c + 1, index + 1, path, configurations)
93
94         # Diagonals
95         dfs(r - 1, c - 1, index + 1, path, configurations)
96         dfs(r - 1, c + 1, index + 1, path, configurations)
97         dfs(r + 1, c - 1, index + 1, path, configurations)
98         dfs(r + 1, c + 1, index + 1, path, configurations)
99
100     # First player
101
102     dfs(0, 1, 1, "A", player1_configurations)
103     dfs(1, 0, 1, "A", player1_configurations)
104     dfs(1, 1, 1, "A", player1_configurations)
105

```

```

106
107
108
109 # Player 2
110 dfs(0, 2, 1, 'D', player2_configurations)
111 dfs(1, 2, 1, 'D', player2_configurations)
112 dfs(1, 3, 1, 'D', player2_configurations)
113
114 # closing files
115 paths.close()
116
117 # Classify paths
118 classify_paths()
119
120
121 # draw paths
122 columns = len(player1_configurations) + 1
123 p1_paths_file = "player1_paths.txt"
124 p2_paths_file = "player2_paths.txt"
125
126 draw_paths(columns, player1_configurations, squares, p1_paths_file,
127            "A", 'Player1Paths')
128 draw_paths(columns, player2_configurations, squares, p2_paths_file,
129            "D", 'Player2Paths')
130
131 print("String configuration of the player1: {}".format(player1_configurations))
132 print("Size of the string of p1: {}".format(len(player1_configurations)))
133 print("String configuration of the player2: {}".format(player2_configurations))
134 print("Size of the string of p2: {}".format(len(player2_configurations)))
135
136 # Animation game
137 p1_paths_file = "player1_correct_paths.txt"
138 p2_paths_file = "player2_correct_paths.txt"
139 make_animation(p1_paths_file, p2_paths_file, board)
140
141 # find_path("player1_correct_paths.txt", "DCF", 'F')
142
143
144 def make_player(color, x, y):
145     simon = turtle.Turtle()
146     simon.shape('turtle')
147     simon.penup()
148     simon.showturtle()
149     simon.setpos(x,y)
150     simon.color(color)

```

```

151     simon.shapesize(3,3,3)
152     return simon
153
154
155 def find_path(paths, actual_path, without_letter=""):
156     # Leer paths
157     with open(paths, "r") as file:
158         content = file.read()
159
160     pattern = r'\b' + re.escape(actual_path) + r'^' + re.escape(without_letter) +
161         r']\w*'
162
163     good_paths = re.findall(pattern, content)
164
165     if not good_paths:
166         # print("No way man")
167         return None
168     # print(good_paths)
169     return good_paths[0]
170
171 def move_turtle(simon, letter, positions):
172     simon.setpos(positions[letter][0], positions[letter][1])
173
174 def congratulate(simon, winner):
175     simon.penup()
176     simon.setpos(-200, -300)
177     simon.color('#FFFFFF')
178     simon.write("Congratulations {} you won!".format(winner), font=('Arial', 25,
179         'normal'))
180     simon.penup()
181     simon.setpos(0, 0)
182
183 def define_positions_in_board(board):
184     positions = {}
185     pos_y = 250
186     for row in board:
187         pos_x = -240
188         for letter in row:
189             positions[letter] = (pos_x, pos_y)
190             pos_x += 160
191             pos_y -= 150
192
193     return positions
194
195 def Move_to_position(simon, letter, positions):
196     simon.setpos(positions[letter][0], positions[letter][1])

```



---

```

196 def make_animation(player1_file, player2_file, board):
197
198     turtle.TurtleScreen._RUNNING=True
199     simon = initTurtle()
200     window = initWindow()
201     make_board(simon, board)
202
203
204
205     player1 = make_player('#4D3D3D', -240, 250)
206     player2 = make_player('#FBAC32', 240,250)
207     player2.left(180)
208
209     turtle.tracer(1,50)
210
211     # Positions:
212     positions = define_positions_in_board(board)
213
214
215     winner = False
216
217
218     # Whose first?
219     path_player1 = "A"
220     index_player1 = 0
221     index_player2 = 0
222     path_player2 = "D"
223
224     path_chosed_player1 = find_path(player1_file, 'A', 'X')
225     path_chosed_player2 = find_path(player2_file, 'D', 'X')
226
227     if not path_chosed_player1 or not path_chosed_player2:
228         print("Theres a problem with the paths of:")
229         if not path_chosed_player1:
230             print("Player1")
231         else:
232             print("Player2")
233
234         return
235
236     curr_player = random.choice([player1, player2])
237     competitor = player1 if curr_player == player2 else player2
238
239     curr_chosed = path_chosed_player1 if curr_player == player1 else
240         path_chosed_player2
241     competitor_chosed = path_chosed_player1 if curr_player == player2 else
242         path_chosed_player2

```

---

```

241 curr_path = path_player1 if curr_player == player1 else path_player2
242 competitor_path = path_player1 if curr_player == player2 else path_player2
243
244
245 curr_index = index_player1 if curr_player == player1 else index_player2
246 competitor_index = index_player1 if curr_player == player2 else index_player2
247
248 curr_file = player1_file if curr_player == player1 else player2_file
249 competitor_file = player1_file if curr_player == player2 else player2_file
250
251 while not winner:
252
253
254     # wineer break
255     if competitor_index == len(competitor_chosed) - 1:
256         winner = competitor
257         break
258
259     # Move the turtle
260     # "AF" 'BF' 'AF' 'B'
261     # print(curr_chosed, curr_index)
262     # print(curr_chosed[curr_index + 1])
263     if curr_chosed[curr_index + 1] == competitor_path[-1]:
264         # print("-----CHANGE PATH -----")
265         # print('curr path:', curr_chosed)
266
267         # Change Path
268         aux_path = curr_chosed
269         # print( curr_path, competitor_path[-1])
270         curr_chosed = find_path(curr_file, curr_path, competitor_path[-1])
271         # print('new path: ', curr_chosed)
272         if not curr_chosed:
273             # pass next turn
274             curr_chosed = aux_path
275
276         # change turn
277         curr_file, competitor_file = competitor_file, curr_file
278         curr_player, competitor = competitor, curr_player
279         curr_chosed, competitor_chosed = competitor_chosed, curr_chosed
280         curr_path, competitor_path = competitor_path, curr_path
281         curr_index, competitor_index = competitor_index, curr_index
282         continue
283
284     curr_index += 1
285     curr_path += curr_chosed[curr_index]
286     Move_to_position(curr_player, curr_path[-1], positions)
287

```

```

288
289     # change turn
290     curr_file, competitor_file = competitor_file, curr_file
291     curr_player, competitor = competitor, curr_player
292     curr_chosed, competitor_chosed = competitor_chosed, curr_chosed
293     curr_path, competitor_path = competitor_path, curr_path
294     curr_index, competitor_index = competitor_index, curr_index
295
296
297
298     if winner == player1:
299         congratulate(simon, "player1")
300     else:
301         congratulate(simon, "player2")
302
303
304     window.exitonclick()
305
306
307 def write_board(simon, x, y, curr_color, letter):
308     simon.penup()
309     simon.setpos(x + 65, y + 75)
310     simon.color('#FFFFFF')
311     simon.write(letter, font=('Arial', 25, 'normal'))
312     simon.penup()
313     simon.setpos(x, y)
314     simon.color(curr_color)
315
316
317 def make_board(simon, board):
318     turtle.tracer(0, 0)
319
320     simon.penup()
321     simon.setpos(-300, 200)
322     simon.pendown()
323     red = '#FA0303'
324     black = '#000000'
325
326     curr_color = red
327     aux_y = 200
328     for row in board:
329         simon.setpos(-300, aux_y)
330         curr_color = red if curr_color == black else black
331         for letter in row:
332             simon.fillcolor(curr_color)
333             simon.begin_fill()
334             # square

```

```

335         for _ in range(4):
336             simon.forward(150)
337             simon.left(90)
338             simon.end_fill()
339
340             write_board(simon, simon.xcor(), simon.ycor(), curr_color, letter)
341
342             simon.forward(150)
343             curr_color = red if curr_color == black else black
344
345             simon.penup()
346             aux_y -= 150
347
348
349 def classify_paths():
350     p1_solutions = open("player1_correct_paths.txt", "w")
351     p2_solutions = open("player2_correct_paths.txt", "w")
352
353     p1_paths = open("player1_paths.txt", "w")
354     p2_paths = open("player2_paths.txt", "w")
355
356     # read paths
357     paths = open("paths.txt", "r")
358
359     # the end should be 9
360     for line in paths:
361         if line[0] == 'A':
362             if line[-2] == 'P':
363                 p1_solutions.write(line)
364             else:
365                 p1_paths.write(line)
366         else:
367             if line[-2] == 'M':
368                 p2_solutions.write(line)
369             else:
370                 p2_paths.write(line)
371
372     # close files
373     p1_solutions.close()
374     p2_solutions.close()
375     p1_paths.close()
376     p2_paths.close()
377
378
379 def initWindow():
380     window = turtle.Screen()
381

```

```

382     window.setup(width=900, height=800)
383     window.bgcolor('#10F7DE')
384     window.title('Parity Automaton')
385     return window
386
387 def initTurtle():
388     simon = turtle.Turtle()
389     simon.speed('fastest')
390     simon.hideturtle()
391     simon.setpos(0,0)
392     simon.pensize(1)
393     simon.pendown()
394     return simon
395
396 def drawStates(x, y, simon, state, size):
397     simon.penup()
398     simon.setpos(x, y)
399     simon.pendown()
400     simon.dot(size, '#18D694')
401
402     simon.penup()
403     simon.setpos(x, y)
404     letter_size = int(16*size/100)
405     simon.write(state, font=('Arial', letter_size, 'normal'))
406
407 def drawArrows(x, y, simon, size):
408     simon.penup()
409     simon.setpos(x, y)
410     simon.pendown()
411     simon.dot(size, '#000000')
412
413 def drawConnectionLines(x, y, simon, state_size):
414     simon.pendown()
415     simon.setpos(x - state_size / 2, y)
416     drawArrows(x - state_size/2, y, simon, 15*state_size/100)
417     simon.penup()
418     simon.setpos(x + state_size / 2, y)
419     simon.penup()
420
421 def drawDigit(x,y, simon, digit):
422     simon.penup()
423
424
425     simon.setpos(x, -300)
426     simon.dot(50, '#EBFA0B')
427     simon.write(digit, font=('Arial', 16, 'normal'))
428

```

---

```

429     simon.penup()
430     simon.setpos(x, y)
431
432 def clearDigits(x,y, simon, digits, state_size, edge_size):
433     simon.penup()
434     simon.setpos(x - 30, -330)
435
436     simon.fillcolor('#10F7DE')
437
438     simon.begin_fill()
439     for _ in range(2):
440         simon.forward(1300)
441         simon.left(90)
442         simon.forward(60)
443         simon.left(90)
444
445     simon.end_fill()
446
447     simon.setpos(x, y)
448
449
450
451 def draw_paths(columns, configurations, squares, paths_file, start, file_name):
452     turtle.TurtleScreen._RUNNING=True
453     turtle.tracer(0,0)
454     simon = initTurtle()
455     window = initWindow()
456
457     # read paths
458     paths = open(paths_file, "r")
459
460     edge_size = 600 / (columns - 1)
461     state_size = 300 / (columns + 2)
462
463     arrow_size = 15*state_size/100
464
465     x_pos = -(450 - 450/(columns + 10))
466     y_pos = 350
467     # draw states
468     # start
469     drawStates(x_pos, y_pos, simon, start, state_size)
470
471     aux_x = x_pos
472     for i in range(0, len(configurations)):
473         aux_x += edge_size + state_size + 10
474         aux_y = y_pos
475         for color in squares[configurations[i]]:

```

```

476         drawStates(aux_x, aux_y, simon, color, state_size)
477         aux_y -= state_size + 30
478
479     # drawing connections
480     levels = {}
481     aux_y = y_pos
482     dobles = 0
483     boxes = "ABCDEFGHIIJKLMNOP"
484     for letter in boxes:
485         if dobles >= 2:
486             aux_y -= state_size + 30
487             dobles = 0
488             dobles += 1
489             levels[letter] = aux_y
490
491
492     for line in paths:
493         aux_x = x_pos
494         aux_y = y_pos
495         simon.penup()
496         simon.setpos(aux_x + state_size/2, aux_y)
497
498         clearDigits(aux_x, aux_y, simon, line[0:-1], state_size, edge_size)
499         simon.setpos(aux_x + state_size/2, aux_y)
500
501         for digit in line[1:-1]:
502             aux_x += edge_size + state_size + 10
503             aux_y = levels[digit]
504
505             drawConnectionLines(aux_x, aux_y, simon, state_size)
506
507             # time.sleep(2)
508
509     canvas = window.getcanvas()
510
511     canvas.postscript(file=file_name + '.eps')
512     Image.open(file_name + '.eps').save(file_name + '.png', 'png')
513
514     window.exitonclick()
515
516 def parse_args():
517     # setup arg parser
518     parser = argparse.ArgumentParser()
519
520     # no input
521     size = random.randint(4, 7)
522     # size = 7

```

```

523 configurations_player1 = ""
524 configurations_player2 = ""
525
526 for _ in range(size):
527     configurations_player1 += random.choice(["r", "b"])
528     configurations_player2 += random.choice(["r", "b"])
529
530
531 # add arguments
532 parser.add_argument("player1_configurations",
533                     type=str, help="string of the board configuration",
534                     default= configurations_player1, nargs='?')
535
536 parser.add_argument("player2_configurations",
537                     type=str, help="string of the board configuration",
538                     default= configurations_player2, nargs='?')
539
540 # parse args
541 args = parser.parse_args()
542
543 # Checking if the size of both strings are equal
544 if len(args.player1_configurations) != len(args.player2_configurations):
545     size = max(len(args.player1_configurations),
546               len(args.player2_configurations))
547     if size == len(args.player1_configurations):
548         for _ in range(size - len(args.player2_configurations)):
549             args.player2_configurations += random.choice(["r", "b"])
550     else:
551         for _ in range(size - len(args.player1_configurations)):
552             args.player1_configurations += random.choice(["r", "b"])
553
554 if args.player1_configurations[-1] != "b":
555     args.player1_configurations = args.player1_configurations[:-1] + "b"
556
557 if args.player2_configurations[-1] != "r":
558     args.player2_configurations = args.player2_configurations[:-1] + "r"
559
560 # return args
561 return args
562
563 # run script
564 if __name__ == "__main__":
565     # add space in logs
566     print("\n\n")
567     print("*" * 60)
568     start = time.time()
569
570 # parse args

```



```

569     args = parse_args()
570
571     # run main function
572     main(args)
573
574     end = time.time()
575     print("Total time taken: {}s (Wall time)".format(end - start))
576     # add space in logs
577     print("*" * 60)
578     print("\n\n")

```

### 3.0.4. Casos de prueba.

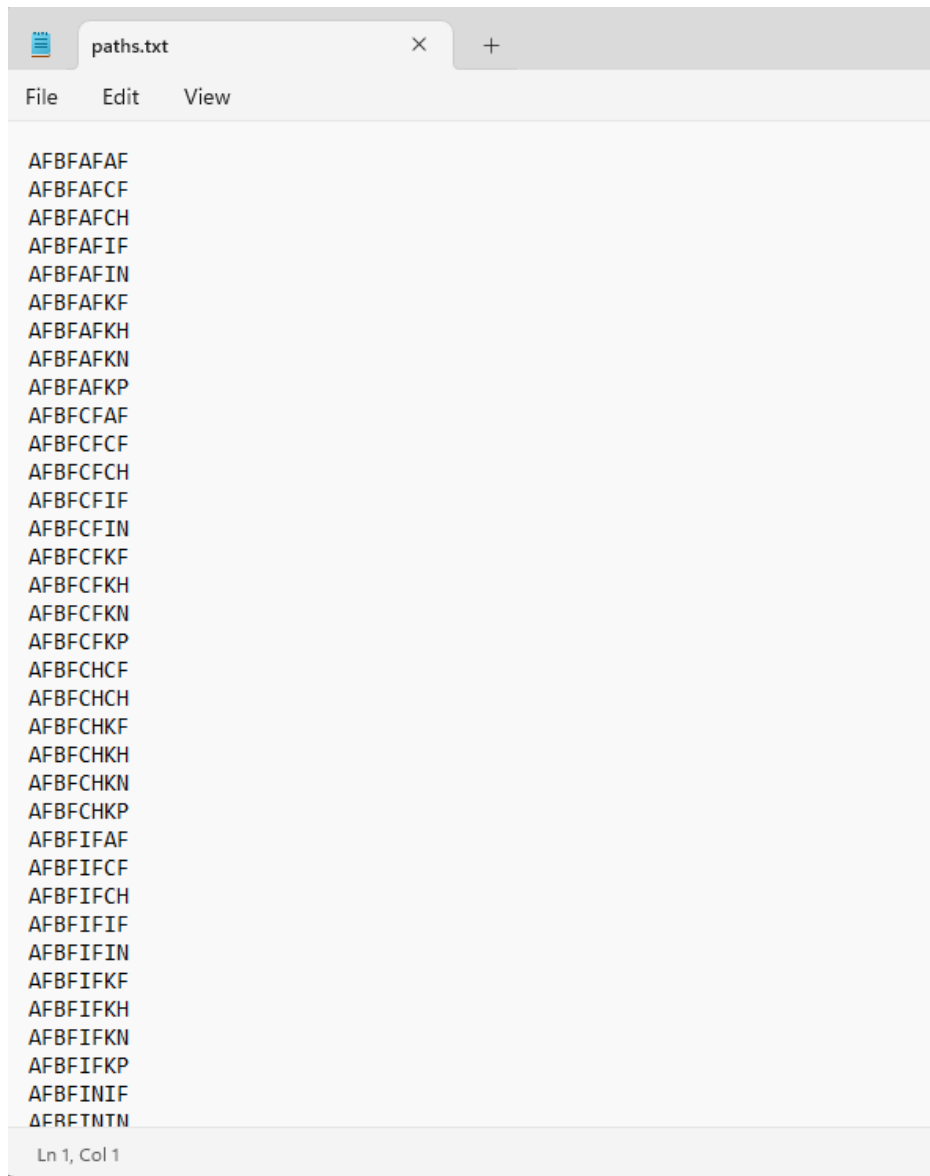
Sin nignun Input.

- Output en archivo de todos los caminos posibles:

```

AFBFAFAF
AFBFAFCF
AFBFAFCH
AFBFAFIF
AFBFAFIN
AFBFAFKF
AFBFAFKH
AFBFAFKN
AFBFAFKP
...
DGLOPLGB
DGLOPLGD
DGLOPLGJ
DGLOPLGL
DGLOPLOJ
DGLOPLOL
DGLOPOJE
DGLOPOJG

```



```
paths.txt
File Edit View

AFBFAFAF
AFBFAFCF
AFBFAFCH
AFBFAFIF
AFBFAFIN
AFBFAFKF
AFBFAFKH
AFBFAFKN
AFBFAFKP
AFBFCFAF
AFBFCFCF
AFBFCFCH
AFBFCFIF
AFBFCFIN
AFBFCFKF
AFBFCFKH
AFBFCFKN
AFBFCFKP
AFBFCHCF
AFBFCHCH
AFBFCHKF
AFBFCHKH
AFBFCHKN
AFBFCHKP
AFBFIFAF
AFBFIFCF
AFBFIFCH
AFBFIFIF
AFBFIFIN
AFBFIFKF
AFBFIFKH
AFBFIFKN
AFBFIFKP
AFBFINIF
ΔFRETTNTN

Ln 1, Col 1
```

Figura 3.1: Caminos posibles.

---

- Output en archivo de paths del jugador 1:

AFBFAFKP  
AFBFCFKP  
AFBFCHKP  
AFBFIFKP  
AFBFINKP  
AFBFKFKP  
AFBFKHKP  
AFBFKNKP  
AFBFKPKP  
AFJFAFKP  
AFJFCFKP  
AFJFCHKP  
AFJFIFKP  
...  
AFGFKPKP  
AFGHCFKP  
AFGHCHKP  
AFGHKFKP  
AFGHKHKP  
AFGHKNKP  
AFGHKPKP



```
AFBFAFKP
AFBFCFKP
AFBFCHKP
AFBFIFKP
AFBFINKP
AFBFKFKP
AFBFKHKP
AFBFKNKP
AFBFKPKP
AFJFAFKP
AFJFCFKP
AFJFCHKP
AFJFIFKP
AFJFINKP
AFJFKFKP
AFJFKHKP
AFJFKNKP
AFJFKPKP
AFJNIFKP
AFJNINKP
AFJNKFKP
AFJNKHKP
AFJNKNKP
AFJNPKPKP
AFJFAFKP
AFJFCFKP
AFJFCHKP
AFJFIFKP
AFJFINKP
AFJFKFKP
AFJFKHKP
AFJFKNKP
AFJFKPKP
AFGFFAFKP
AFGFCEFKP
```

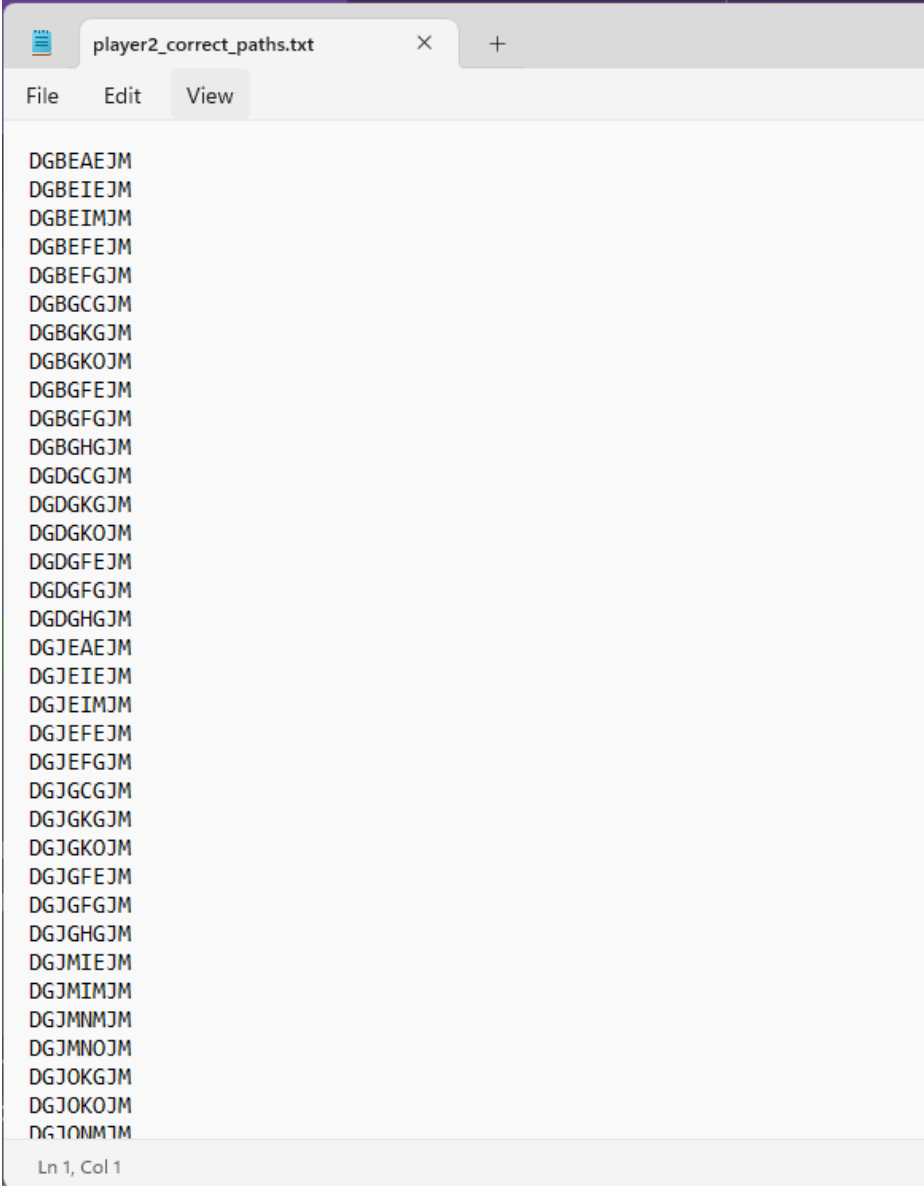
Ln 1, Col 1

Figura 3.2: Caminos posibles.

---

- Output en archivo de paths del jugador 2:

DGBEAEJM  
DGBEIEJM  
DGBEIMJM  
DGBEFEJM  
DGBEFGJM  
DGBGCGJM  
DGBGKGJM  
DGBGKOJM  
DGBGFEJM  
DGBGFGJM  
DGBGHGJM  
DGDGCGJM  
DGDGKGJM  
...  
DGLGFEJM  
DGLGFGJM  
DGLGHGJM  
DGLOKGJM  
DGLOKOJM  
DGLONMJM  
DGLONOJM  
DGLOPOJM



```
player2_correct_paths.txt
File Edit View
DGBEAEJM
DGBEIEJM
DGBEIMJM
DGBEFEJM
DGBEFGJM
DGBGCGJM
DGBGKGJM
DGBGKOJM
DGBGFEJM
DGBGFGJM
DGBGHGJM
DGDGCGJM
DGDGKGJM
DGDGKOJM
DGDGFEJM
DGDGFGJM
DGDGHGJM
DGJAEJM
DGJIEJM
DGJIMJM
DGJEFEJM
DGJEFGJM
DGJGCGJM
DGJGKGJM
DGJGKOJM
DGJGFEJM
DGJGFGJM
DGJGHGJM
DGJMIEJM
DGJMIMJM
DGJMNMJM
DGJMNOJM
DGJOKGJM
DGJOKOJM
DGJONMIM
Ln 1, Col 1
```

Figura 3.3: Caminos posibles.

---

- Mensaje en la terminal:

```
C:\Users\CristoRey\Escom\SEM_5>py TwoPlayersAutomatom.py
*****
Enter the string of the player1:
Enter the string of the player2:
String configuration of the player1: brbbbbbb
Size of the string of p1: 7
String configuration of the player2: rrrbrrr
Size of the string of p2: 7
Total time taken: 157.83887362480164s (Wall time)
*****
```



```
Command Prompt
C:\Users\CristoRey\OneDrive\Música\Documentos\Yo\Escom\SEM_5>py test.py

*****
Enter the string of the player1:
Enter the string of the player2:
String configuration of the player1: brbbbbbb
Size of the string of p1: 7
String configuration of the player2: rrrbrrr
Size of the string of p2: 7
Total time taken: 20.1107439994812s (Wall time)
*****

C:\Users\CristoRey\OneDrive\Música\Documentos\Yo\Escom\SEM_5>|
```

Figura 3.4: Terminal.

- Gráfica de caminos del primer jugador:

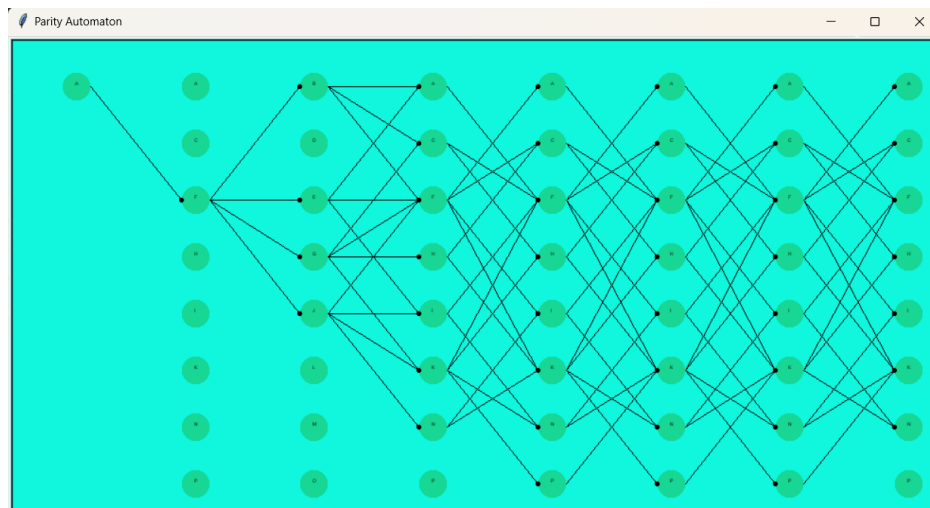


Figura 3.5: Caminos posibles del primer jugador.



- 
- Gráfica de caminos del segundo jugador:

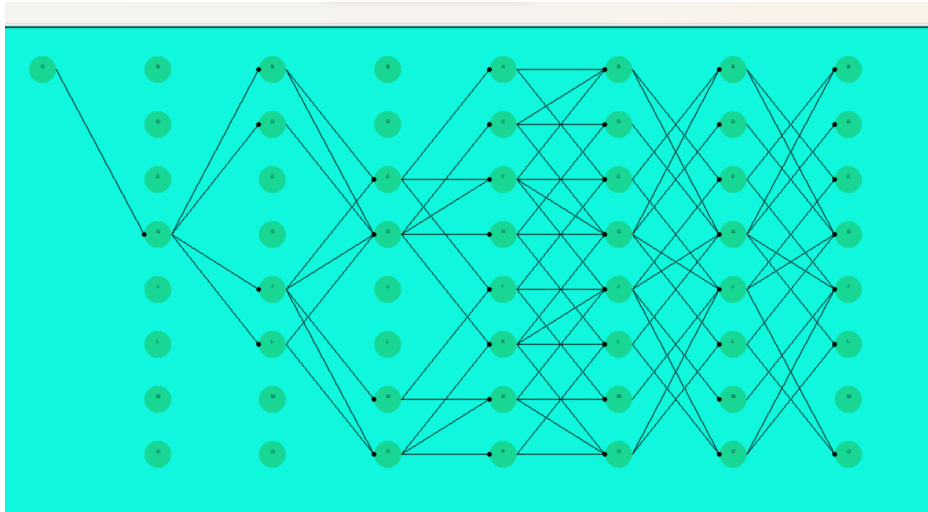
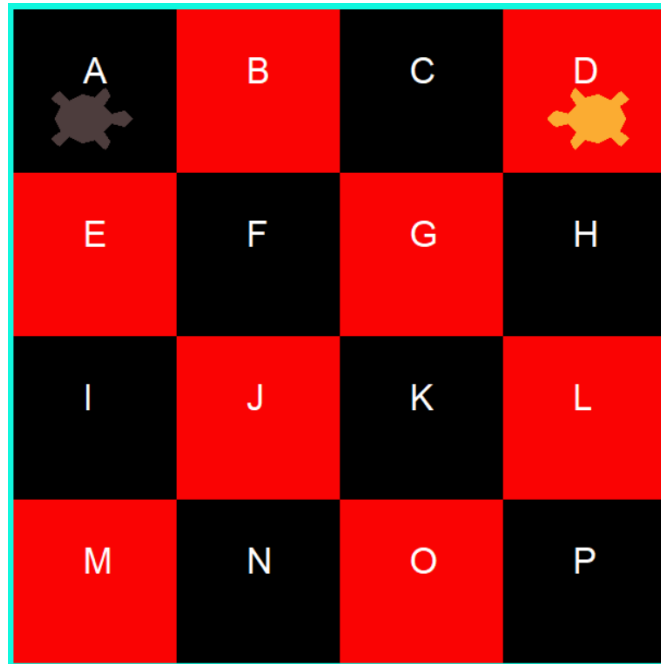
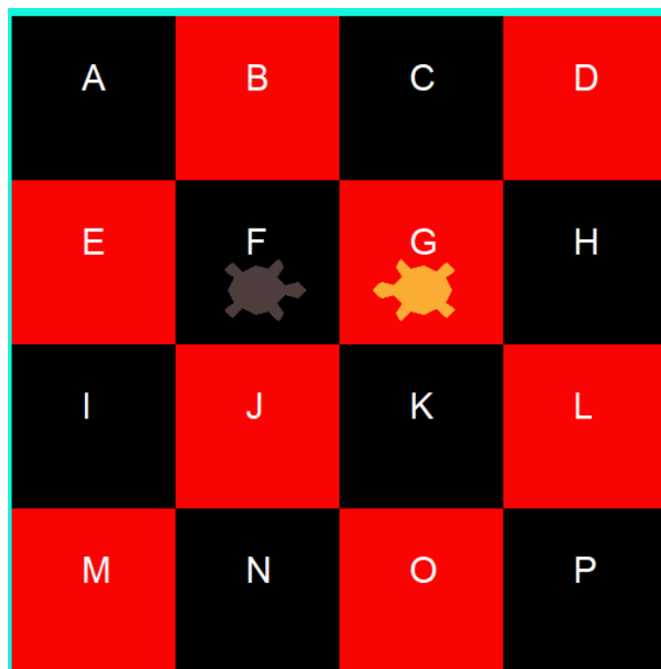
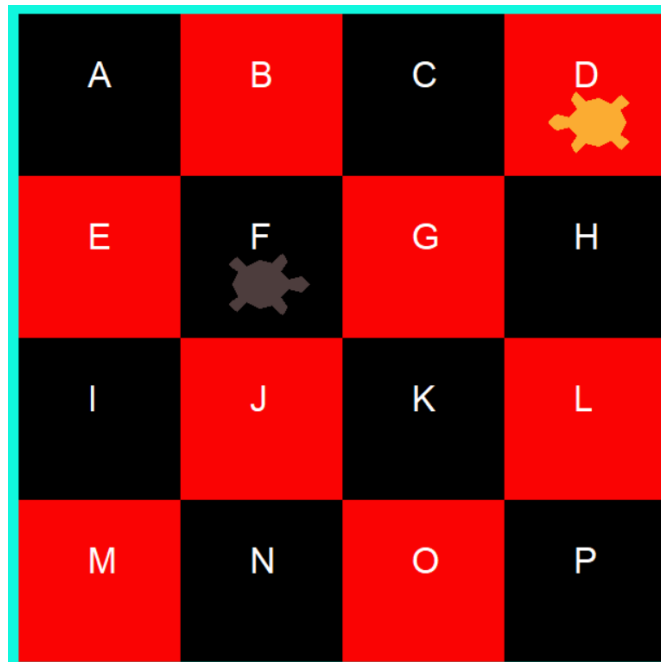


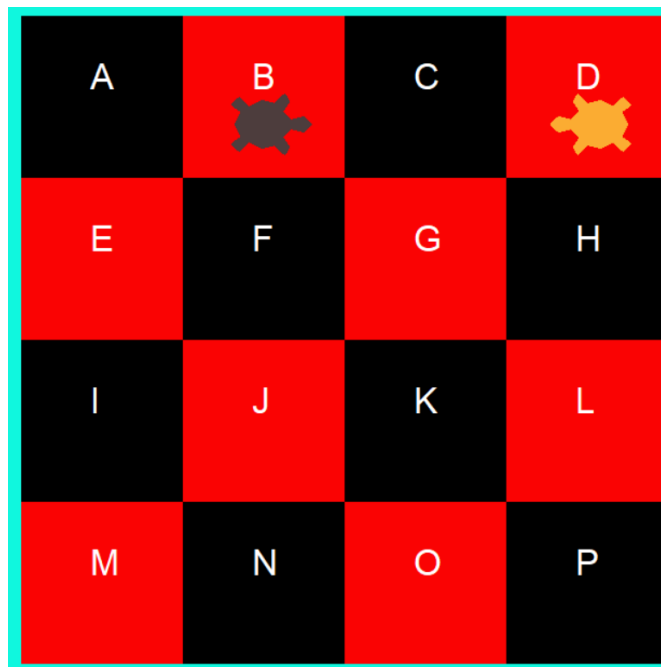
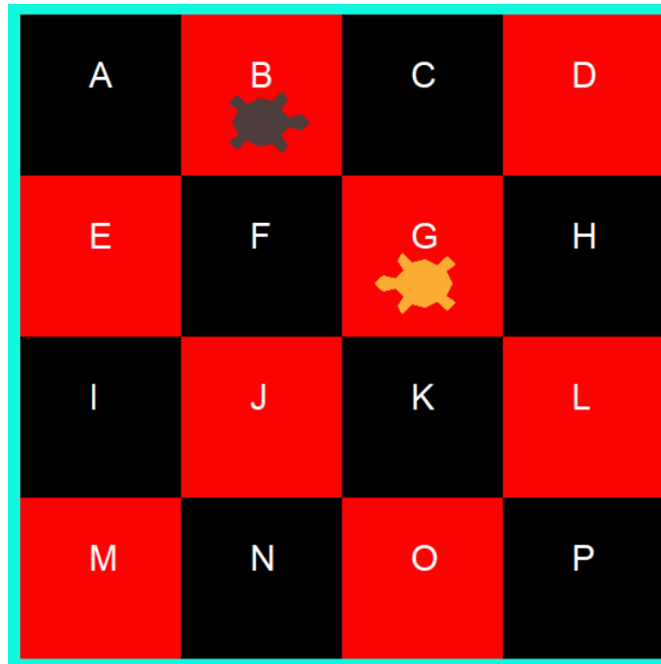
Figura 3.6: Caminos posibles del segundo jugador.

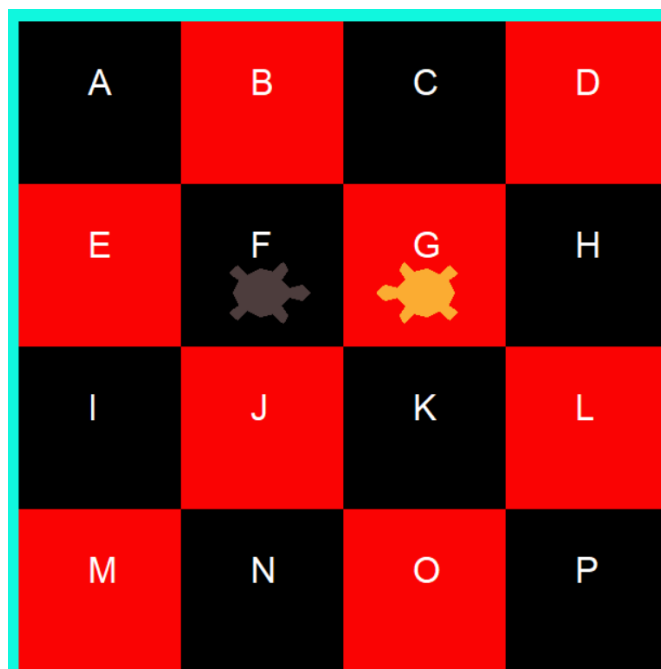
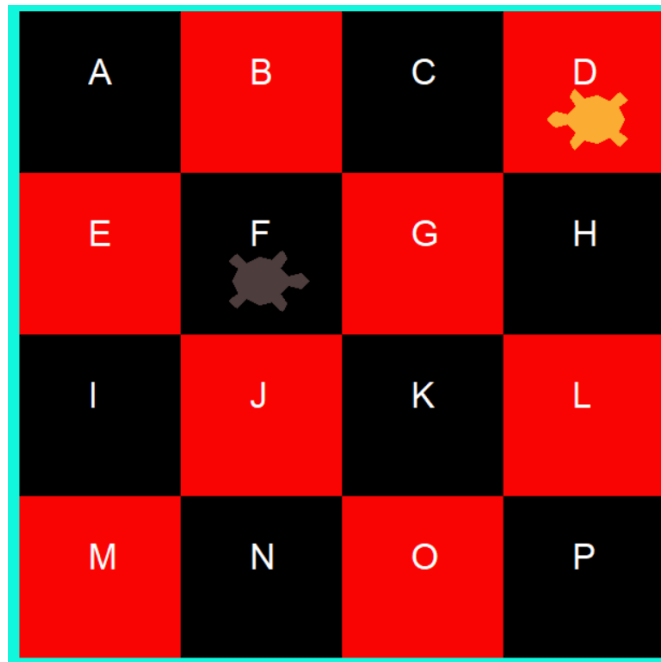
---

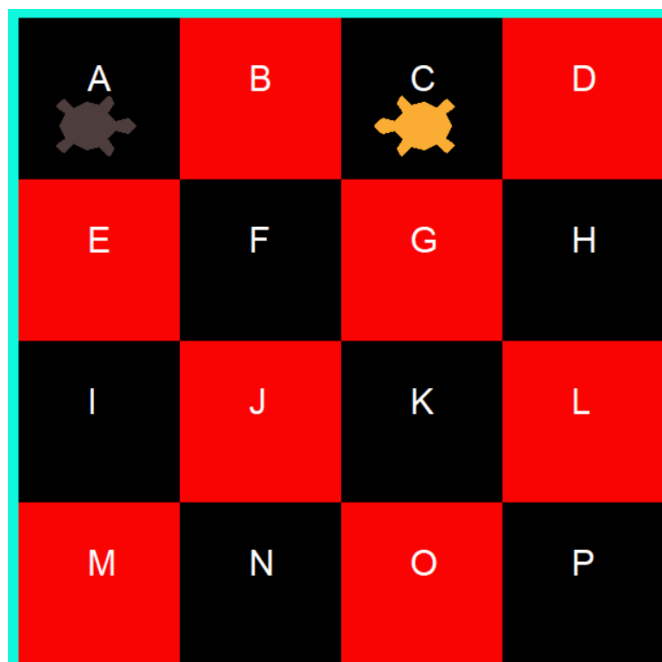
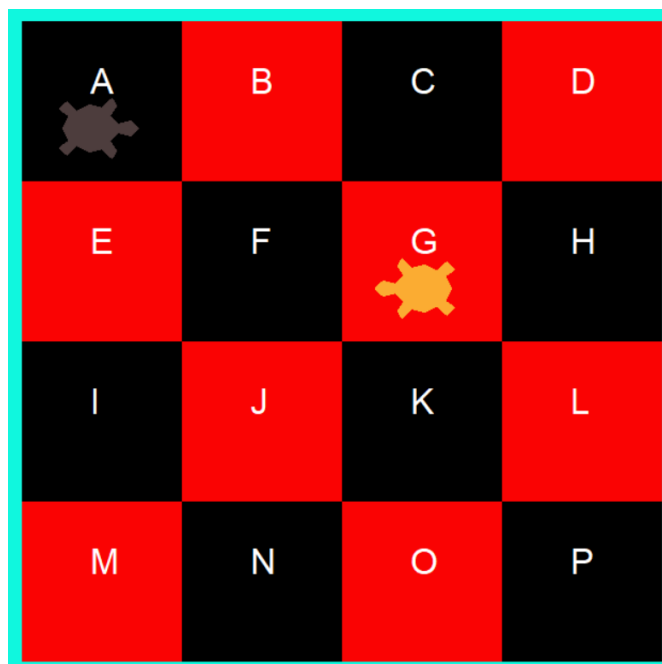
- Animación:

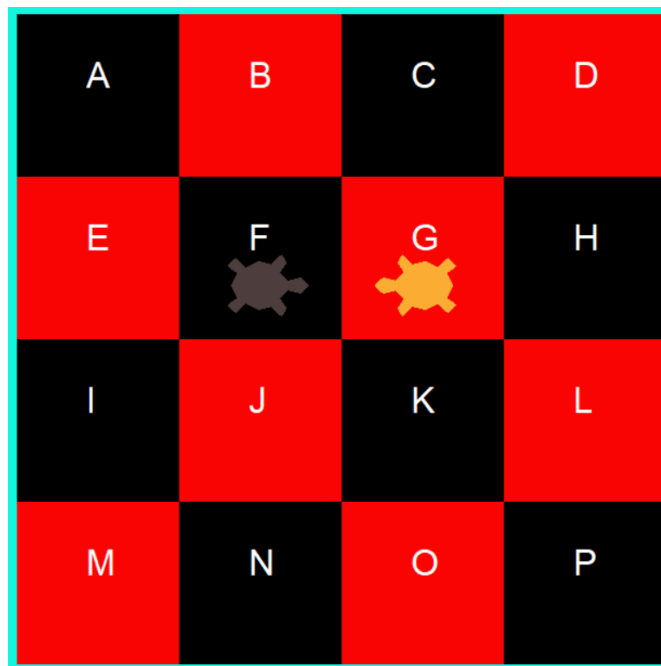
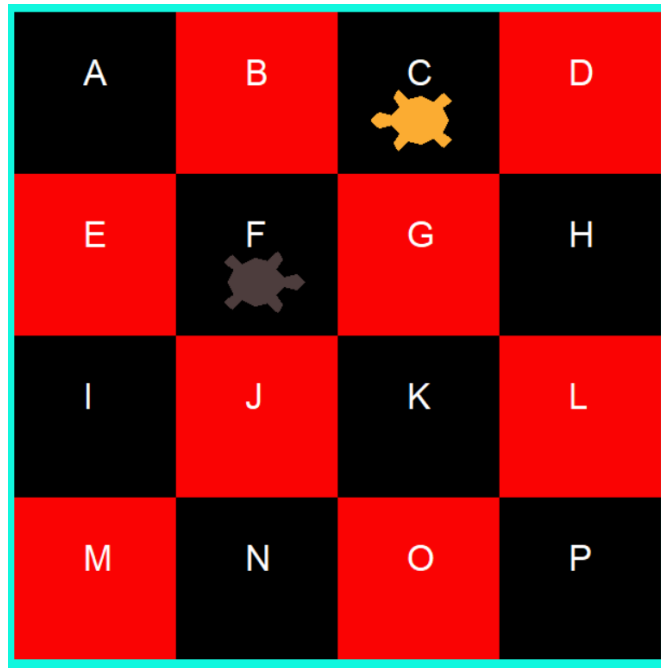


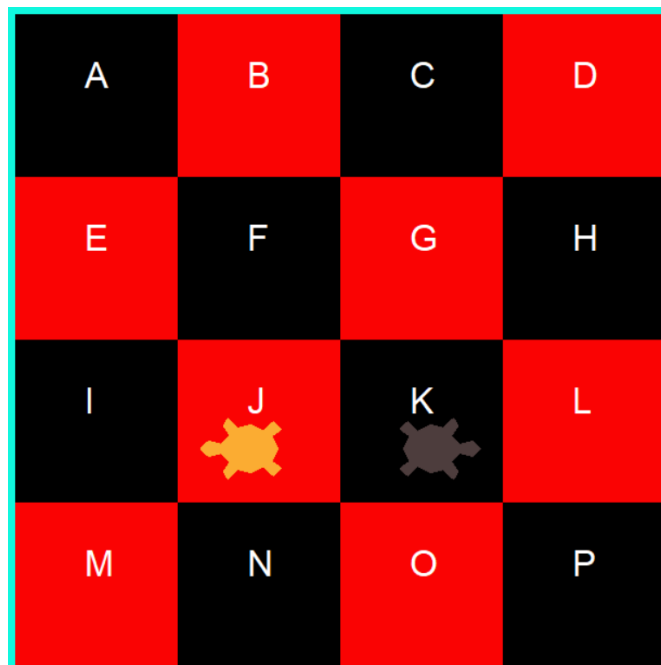
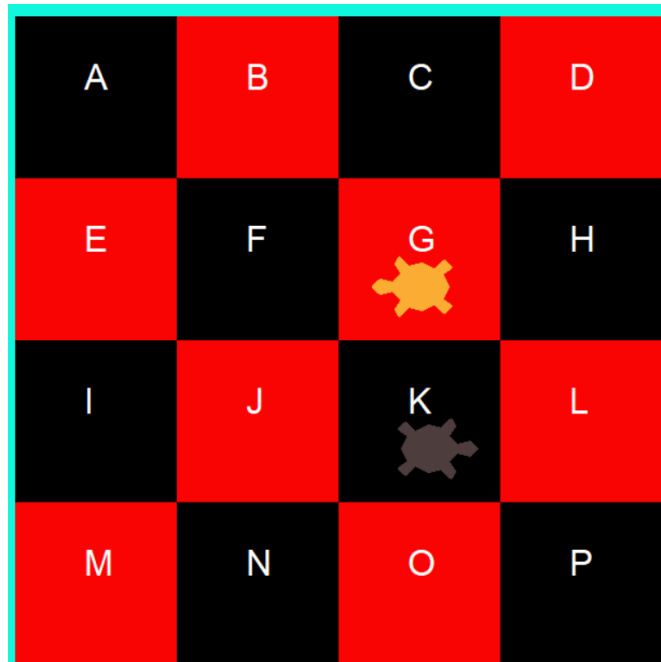




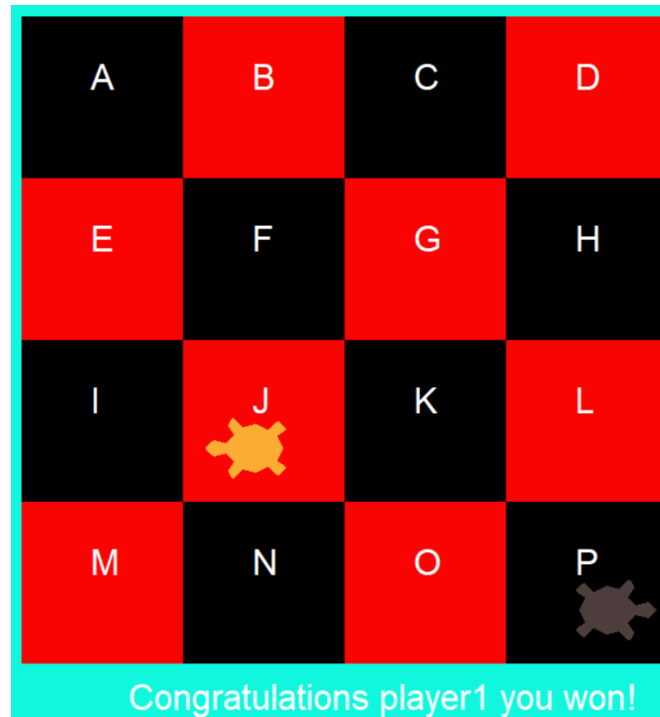












#### Declarando una secuencia en ambos automatatas

- Output en archivo de todos los caminos posibles:

ABFAEA  
ABFAEI  
ABFAEF  
ABFABF  
ABFABA  
ABFABC  
ABFCGC  
ABFCGK  
ABFCGF  
ABFCGH  
ABFCBF  
ABFCBA  
ABFCBC  
ABFCDH  
...  
DGHGKO  
DGHGKJ  
DGHGKL  
DGHGFB  
DGHGFJ  
DGHGFE

---

DGHGFG  
DGHGHD  
DGHGHL  
DGHGHG

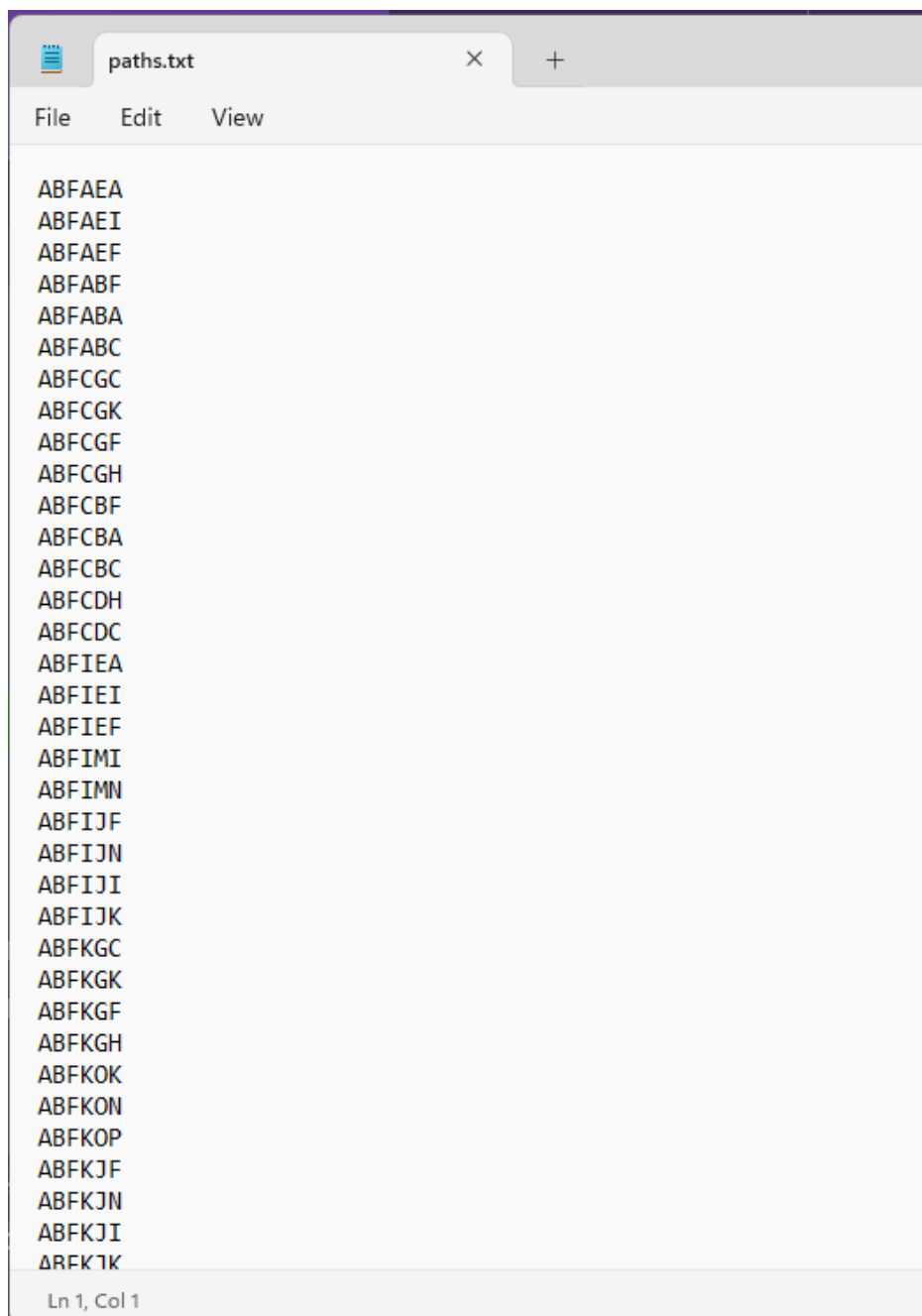


Figura 3.7: Caminos posibles.

---

- Output en archivo de paths del jugador 1:

ABFKOP  
ABFKLP  
ABCHLP  
AEINOP  
AEFKOP  
AEFKLP

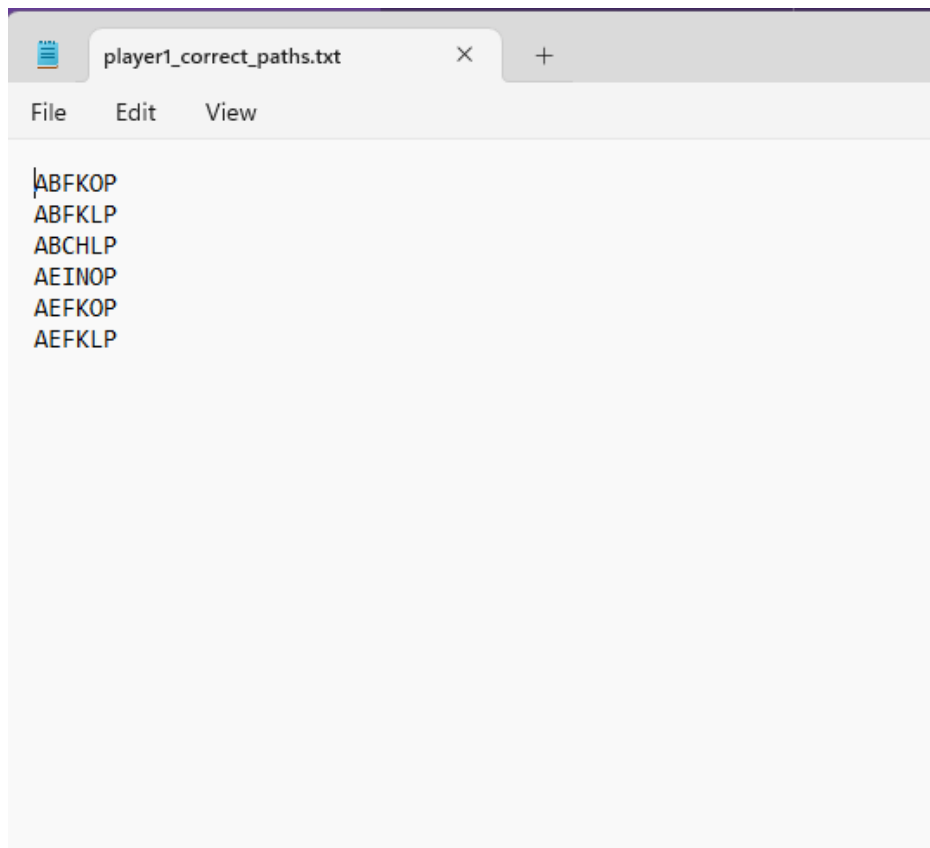


Figura 3.8: Caminos posibles.

---

- Output en archivo de paths del jugador 2:

DGKONM  
DGKJNM  
DGKJIM  
DGFJNM  
DGFJIM  
DGFEIM

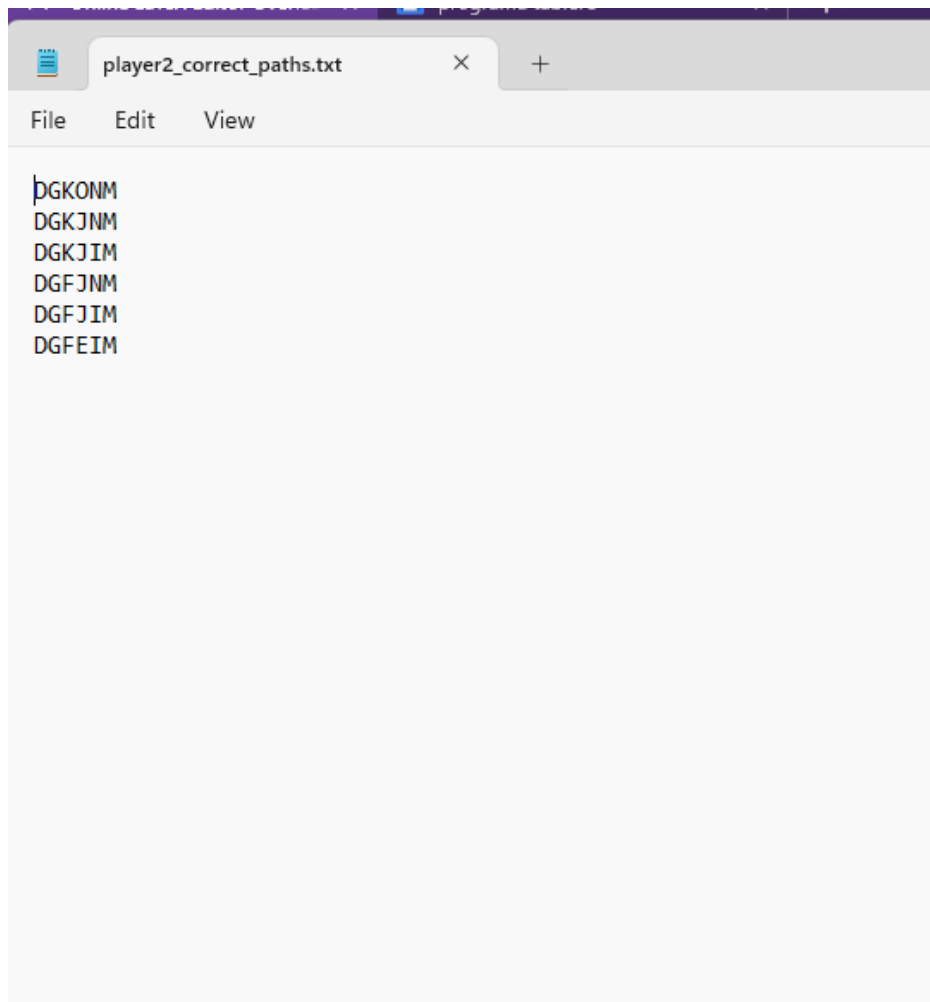


Figura 3.9: Caminos posibles.

---

- Mensaje en la terminal:

```
C:\Users\CristoRey\Escm\SEM_5>py TwoPlayersAutomatom.py
*****
Enter the string of the player1: rbbrb
Enter the string of the player2: rbrbr
String configuration of the player1: rbbrb
Size of the string of p1: 5
String configuration of the player2: rbrbr
Size of the string of p2: 5
Total time taken: 13.737199783325195s (Wall time)
*****
```



```
Command Prompt
C:\Users\CristoRey\OneDrive\Música\Documentos\Yo\Escm\SEM_5>py test.py

*****
Enter the string of the player1: rbbrb
Enter the string of the player2: rbrbr
String configuration of the player1: rbbrb
Size of the string of p1: 5
String configuration of the player2: rbrbr
Size of the string of p2: 5
Total time taken: 23.504863262176514s (Wall time)
*****

C:\Users\CristoRey\OneDrive\Música\Documentos\Yo\Escm\SEM_5>
```

Figura 3.10: Caminos posibles.

- Gráfica de caminos del primer jugador:

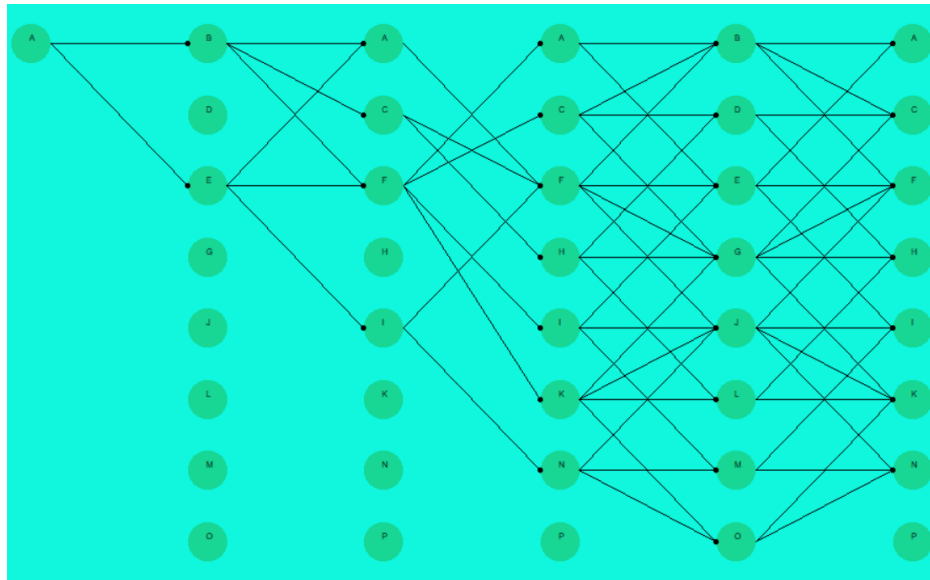


Figura 3.11: Caminos posibles del primer jugador.

- **Gráfica de caminos del segundo jugador:**

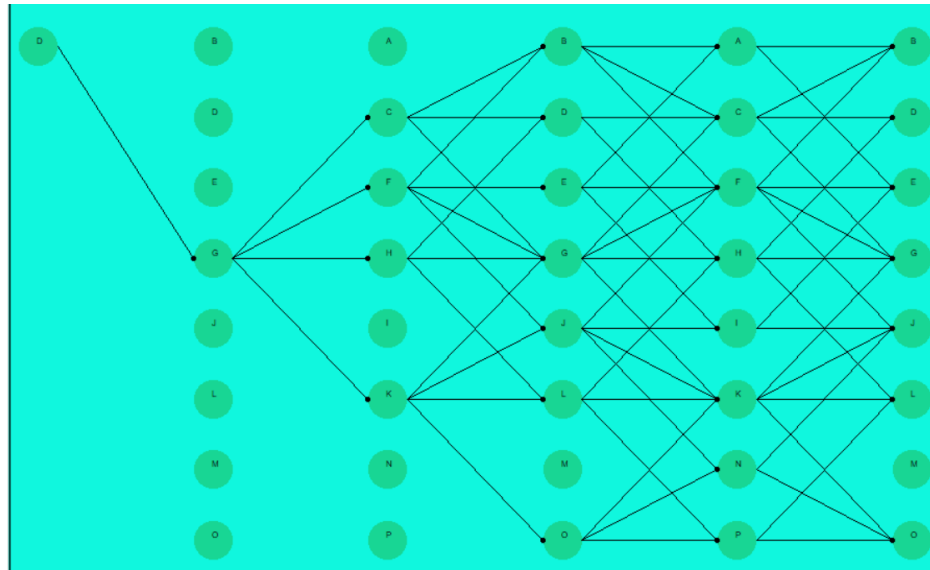
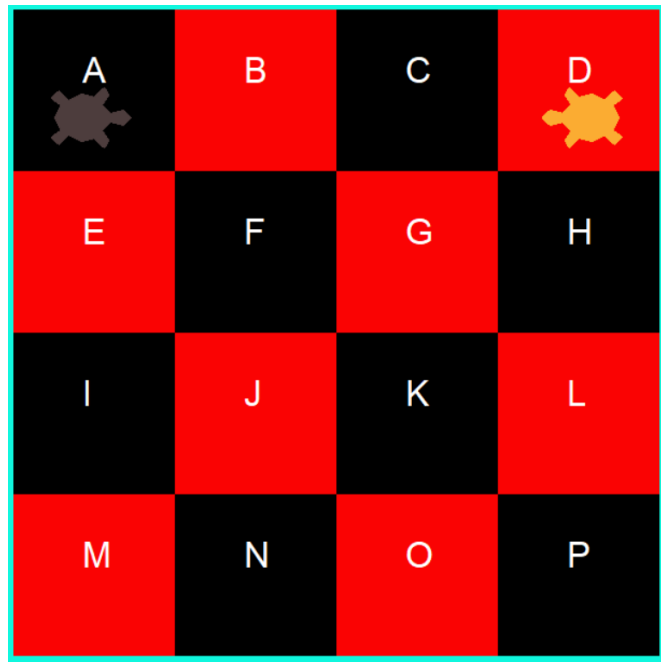


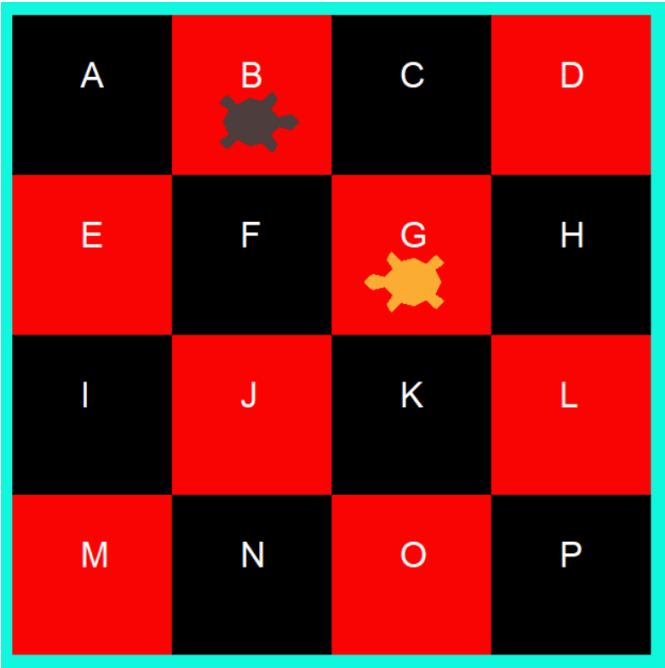
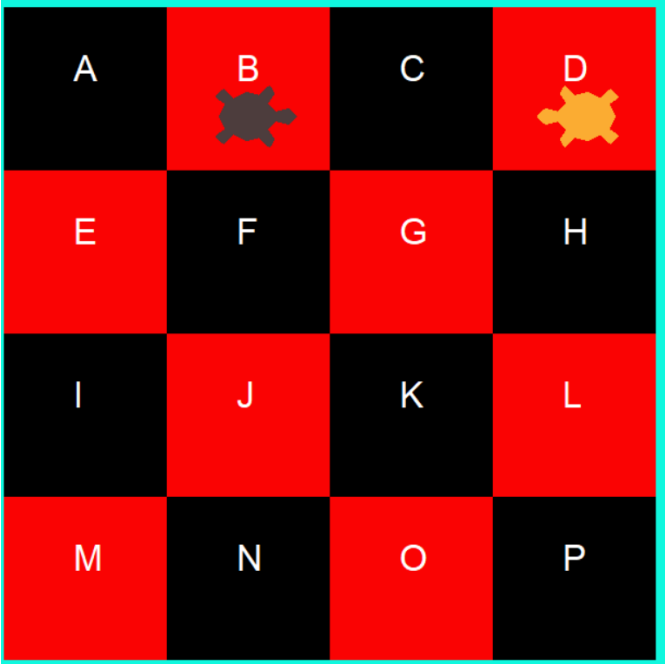
Figura 3.12: Caminos posibles del segundo jugador.

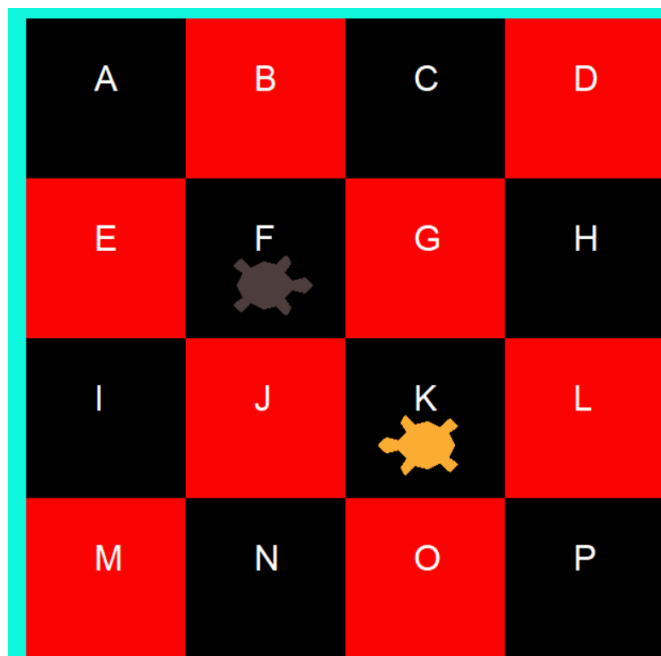
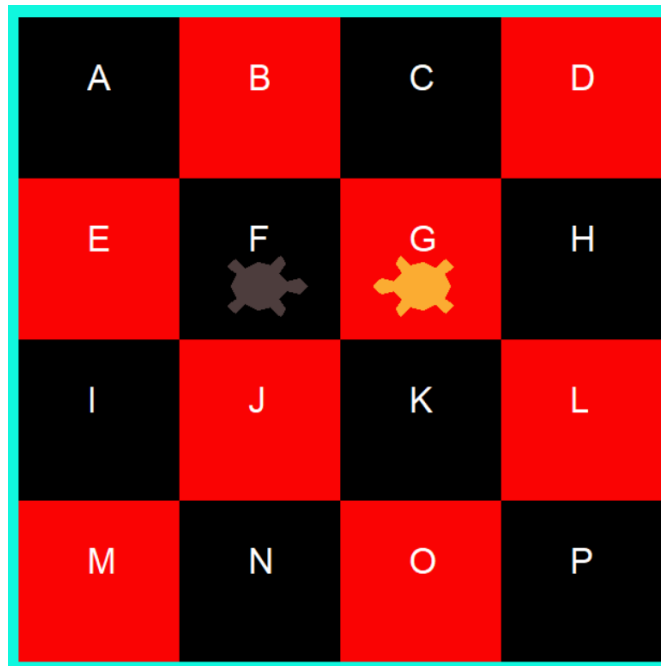


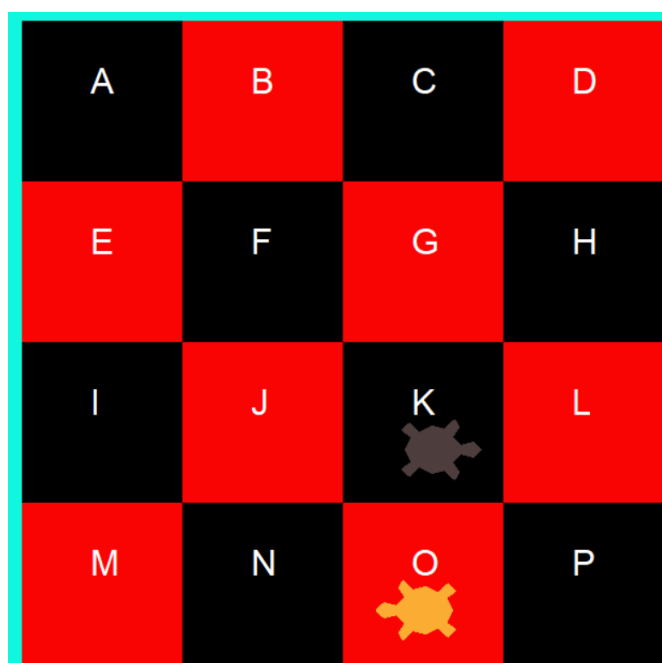
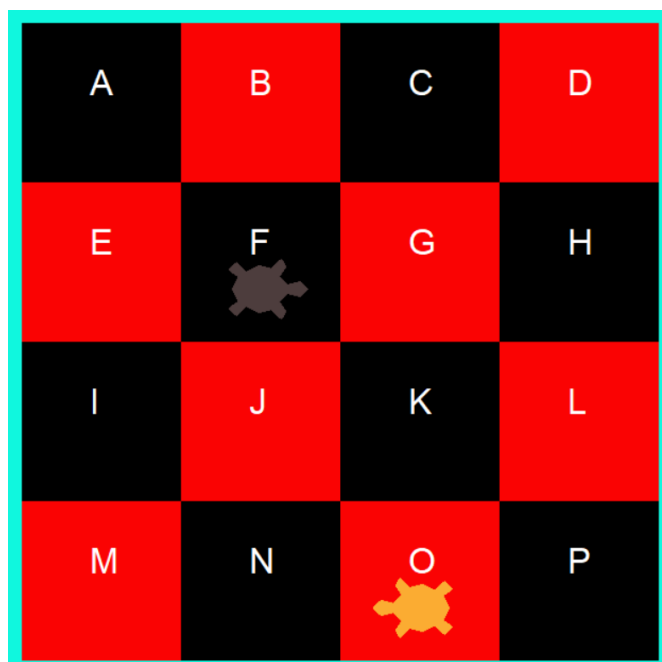
---

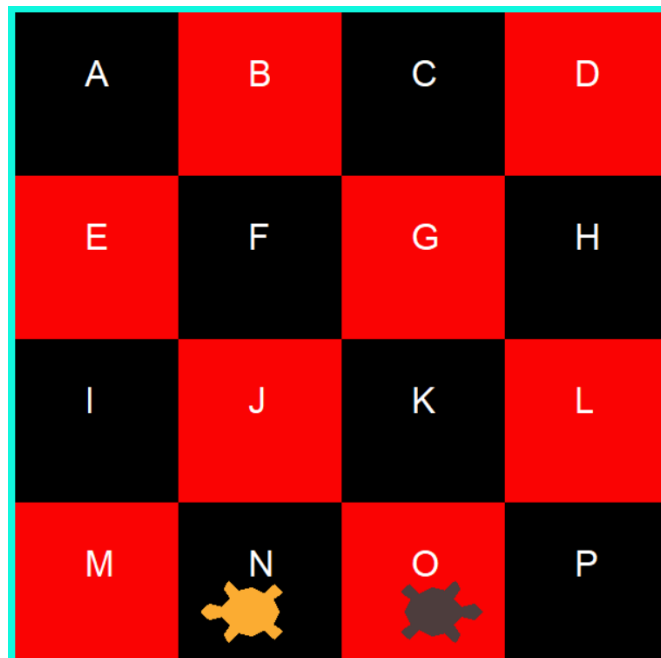
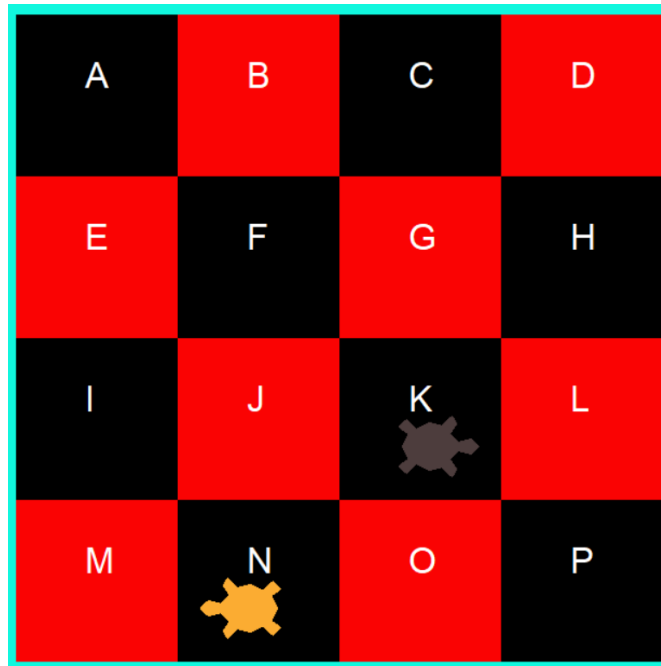
- Animación:

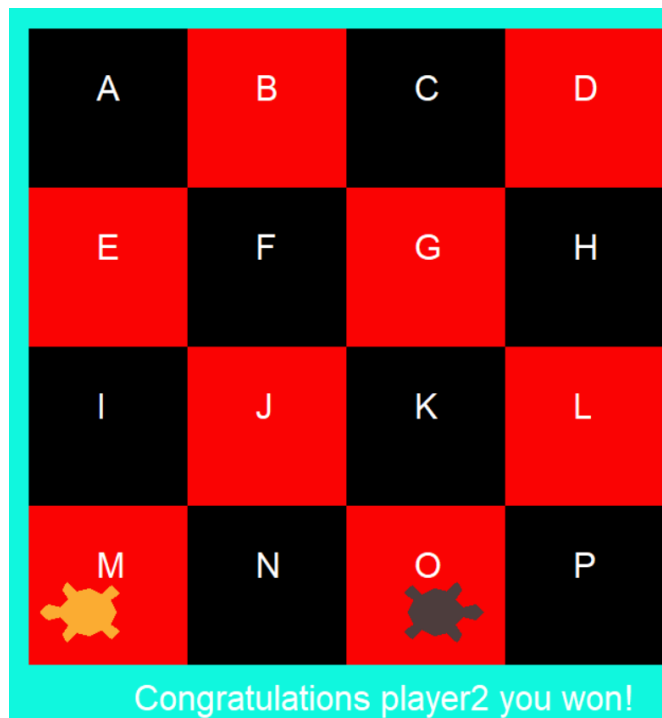












---

### Conclusión

---

Crear un autómata no determinista puede resultar muy fácil para representar un problema de la vida real; sin embargo, el hecho de que pueda estar en múltiples estados puede causar un problema dentro de la programación, llevando el problema a un problema de cómputo en paralelo.

Además, viendo el problema como un grafo nos resulta apto para poder aplicar los algoritmos de búsqueda convencionales, como fue el caso. En cambio, si tuviéramos que hacer el problema de forma en un autómata determinista, sinceramente, tardaría demasiado en poder tener una respuesta del autómata.





---

### Referencias.

---

- Hopcroft, J. E., Motwani, R., and Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson.
- Juancar M. (s.f). Definición Formal de un Autómata Finito Determinista (AFD) [Archivo de video]. Jauncar Molinero. <https://www.youtube.com/watch?v=P0AxQvJcN2Q>
- Sipser, M. (2006). Introduction to the Theory of Computation. Cengage Learning.