Instituto Politécnico Nacional

Escuela Superior de Cómputo

Programa 3: Tablero

Autor: Colín Ramiro Joel

Materia: Teoría de la Computación

Grupo: 4CM2

Profesor: Juarez Martínez Gemaro

Fecha de entrega: 12 de Octubre 2021

Introduction

Instrucciones

Elaborar un programa para realizar movimientos ortogonales y diagonales en un tablero de ajedrez de 4x4 con una pieza. Los movimientos y las reglas están explicadas en las láminas del curso de Stanford. Adicionalmente, el programa debe de contar con las siguientes características:

1. Debe de correr en modo automático y forma manual.
2. El usuario podrá introducir la cadena de movimientos o generarla aleatoriamente, con un máximo de 100 caracteres para el caso aleatorio. Si se escoge el modo manual las cadenas generadas no deben ser mayores a 20 movimientos.
3. El autómata que se va a programar es el NFA.
4. El estado inicial es el estado 1 y el final el estado 16.
5. Una vez definida la cadena de movimientos para una pieza, se deben generar los archivos de todos los movimientos posibles y generar otro archivo con todos los movimientos ganadores.
6. Dibujar el tablero y mostrar los movimientos de dos jugadas seleccionadas aleatoriamente del archivo de movimientos ganadoras. Para el caso de la animación pueden intentar poner la pieza del rey con bitmap o dibujar un circulo dentro del cuadrado, para posteriormente despintar y pintar el círculo, de manera que parezca que se mueve. Sugerencia: para dibujar el tablero utilizar la función de la librería gráfica que refiera un cuadrado y aplicar un fill para ponerle el color.
7. En el reporte debe de estar también el código de la implementación.

Desarrollo

Este programa se realizó en un solo archivo a diferencia del programa anterior. El programa solicitaba que tuviese la opción de correr manual y automáticamente, asi que hicimos un tipo de menú para que el usuario escojiese la opción. Támbien con el NFA el programa deberia de tomar la cadena ya sea propuesta por el usuario o generada automáticamente por el programa y validar dicha cadena, la cual si resulta ganadora, se debia de escribir en un archivo txt llamado "Ganadoras.txt" y todas las demás en otro archivo txt llamado "Todas.txt"

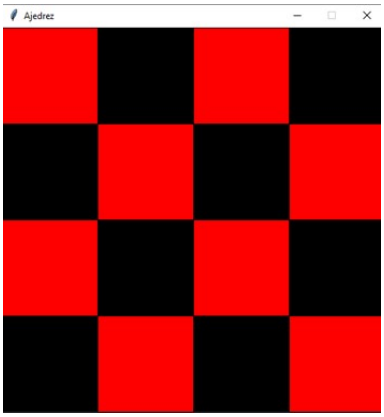
Capturas del Funcionamiento

Primordialmente se creó una especie de menú para que el usuario introdujese la opción requerida ya sea que el programa funcione manual o automaticamente.

```
import random
import chess
from graphics import *
class NFA(object):
    def __init__(self, Q, sigma, delta, q0, F):
        lista = []
        for i in range(Q):
            lista.append("q"+str(i+1))
        self.Q = lista
        self.sigma = sigma
        self.delta = delta
        self.completar_diccionario()
        self.q0 = q0
        self.F = F
        self.estado_actual = [q0]
        self.camino = []
    def get_estado_actual(self):
        return self.estado_actual
    def set_estado_actual(self, estado_actual):
        self.estado_actual = estado_actual
    def get_F(self):
        return self.F
    def completar_diccionario(self):
        sigmaTemp = self.sigma
        sigmaTemp.append(chr(1013))
        for i in self.delta.keys():
            if "rest" in self.delta[i].keys():
                for j in self.sigma:
                    if j != chr(1013) and not (j in self.delta[i].keys()):
                        self.delta[i][j] = None
                        continue
                    if not (j in self.delta[i].keys()):
                        self.delta[i][j] = self.delta[i]["rest"]
                        self.delta[i].pop("rest")
                    else:
                        for j in sigmaTemp:
                            if not j in self.delta[i].keys():
                                self.delta[i][j] = None
```

```
2
Recualculando Ruta...
Ingresa tu cadena: rbrbbbr
El camino que tomara el blanco es: ['q6', 'q2', 'q1', 'q2', 'q7', 'q12', 'q16']
Recualculando Ruta...
Ingresa tu cadena:
```

Y finalmente se crea el tablero en una ventana emergente via Python.



Código

La elaboración de este programa si resultó un código un poco largo, se adjuntan las capturas.

```python
import random
import chess
from graphics import *
class NFA(object):
    def __init__(self, Q, sigma, delta, q0, F):
        lista = []
        for i in range(Q):
            lista.append("q"+str(i+1))
        self.Q = lista
        self.sigma = sigma
        self.delta = delta
        self.completar_diccionario()
        self.q0 = q0
        self.F = F
        self.estado_actual = [q0]
        self.camino = []
    def get_estado_actual(self):
        return self.estado_actual
    def set_estado_actual(self, estado_actual):
        self.estado_actual = estado_actual
    def get_F(self):
        return self.F
    def completar_diccionario(self):
        sigmaTemp = self.sigma
        sigmaTemp.append(chr(1013))
        for i in self.delta.keys():
            if "rest" in self.delta[i].keys():
                for j in self.sigma:
                    if j == chr(1013) and not (j in self.delta[i].keys()):
                        self.delta[i][j] = None
                        continue
                    if not (j in self.delta[i].keys()):
                        self.delta[i][j] = self.delta[i]["rest"]
                        self.delta[i].pop("rest")
                    else:
                        for j in sigmaTemp:
                            if not j in self.delta[i].keys():
                                self.delta[i][j] = None
```

```python
def validar(self, cadena):
    indice = 0
    for i in cadena:
        estado_actual = self.get_estado_actual()
        print("Estado actual: " + str(estado_actual))
        estado_siguiente = []

        for estados in estado_actual:
            if estados == None:
                continue
            conexionesTemp = self.delta[estados][i]
            if conexionesTemp != None:
                if isinstance(conexionesTemp, list):
                    for sig_estados in conexionesTemp:
                        estado_siguiente.append(sig_estados)
                else:
                    estado_siguiente.append(conexionesTemp)
            if conexionesTemp == None:
                estado_siguiente.append(None)
            episilon = conexionesTemp = self.delta[estados][chr(1013)]
            if episilon == None:
                continue
            if isinstance(episilon ,list):
                for estados in episilon:
                    estado_siguiente.append(estados)
            else:
                estado_siguiente.append(episilon)

        self.set_estado_actual(estado_siguiente)
        indice +=1
    for i in self.get_estado_actual():
        if i in self.get_F():
            return True
    return False
```

```python
def pruebaRecursiva(self, cadena, estadoActual):
    if cadena == "":
        if estadoActual in self.get_F():
            return True
        else:
            return False
    siguientesEstados = self.delta[estadoActual][cadena[0]]
    if isinstance(siguientesEstados, list):
        for estado in siguientesEstados:
            if self.pruebaRecursiva(cadena[1:],estado):
                self.camino.append(estado)
                return True
    else:
        if self.pruebaRecursiva(cadena[1:], siguientesEstados):
            self.camino.append(siguientesEstados)
            return True
        else:
            return False
```

```python
print('''Elija una opcion:\n
    1.-Automático
    2.-Manual
    ''')
opc = input()
if (opc == '1'):
    elegir = False
elif (opc == '2'):
    elegir = True
```

```python
def generarCadena(ingresar = False, automata = None, estado = None):
    if not ingresar:
        cad = ""
        tam = random.randint(4,100)
        for i in range(tam):
            if random.randint(0,1):
                cad+="r"
            else:
                cad+="b"
        cad+="b"
        return cad
    while 1:
        cad = input("Ingresa la cadena: ")
        if automata.pruebaRecursiva(cad, estado):
            automata.camino.clear()
            automata.camino.reverse()
            return cad
        else:
            print("Cadena no valida")
            automata.camino.clear()
```

```python
q = 16
sigma = ["r", "b"]
delta = {
    "q1" : {"r": ["q6"], "b": ["q2","q5"]},
    "q2" : {"r": ["q1","q3","q6"], "b": ["q5","q7"]},
    "q3" : {"r": ["q6","q8"], "b": ["q2","q4","q7"]},
    "q4" : {"r": ["q3","q8"], "b": ["q7"]},
    "q5" : {"r": ["q1","q6","q9"], "b": ["q2","q10"]},
    "q6" : {"r": ["q1","q3","q9","q11"], "b": ["q2","q5","q7","q10"]},
    "q7" : {"r": ["q3","q8","q11"], "b": ["q2","q4","q10","q12"]},
    "q8" : {"r": ["q3","q11"], "b": ["q4","q7","q12"]},
    "q9" : {"r": ["q6","q14"], "b": ["q5","q10","q13"]},
    "q10" : {"r" : ["q6","q9","q11","q14"], "b" : ["q5","q7","q13","q15"]},
    "q11" : {"r" : ["q6","q8","q14","q16"], "b" : ["q7","q10","q12","q15"]},
    "q12" : {"r" : ["q8","q11","q16"], "b" : ["q7","q15"]},
    "q13" : {"r" : ["q9","q14"], "b" : ["q10"]},
    "q14" : {"r" : ["q9","q11"], "b" : ["q10","q13","q15"]},
    "q15" : {"r" : ["q11","q14","q16"], "b" : ["q10","q12"]},
    "q16" : {"r" : ["q11"], "b" : ["q12","q15"]},
}
q0 = "q1"
F = ["q16"]
automata = NFA(q, sigma, delta, q0, F)
```

```python
def recalcularRuta(automata, estadoA):
    print("Recualculando Ruta...")
    while 1:
        automata.camino.clear()
        if elegir:
            nuevaRuta = generarCadena(True, automata, estadoA)
            break
        else:
            nuevaRuta = generarCadena()
        if automata.pruebaRecursiva(nuevaRuta, estadoA):
            automata.camino.reverse()
            recalcularRuta(automata, automata.q0)
print("El camino que tomara la pieza es: "+ str(automata.camino))
casillas = {
    "q1": "a8",
    "q2": "b8",
    "q3": "c8",
    "q4": "d8",

    "q5": "a7",
    "q6": "b7",
    "q7": "c7",
    "q8": "d7",

    "q9": "a6",
    "q10": "b6",
    "q11": "c6",
    "q12": "d6",
```

```python
def dibujar():
    win  = GraphWin("Ajedrez", 500, 500)
    win.setBackground(color_rgb(0,0,0))
    rect = Rectangle(Point(0,0),Point(125, 125))
    rect.setFill(color_rgb(255, 0, 0))

    rect.draw(win)
    rect = Rectangle(Point(125,125),Point(250, 250))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(250,250),Point(375, 375))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(375,375),Point(500, 500))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(0,250),Point(125, 375))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(250,0),Point(375, 125))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(375,125),Point(500, 250))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
    rect = Rectangle(Point(125,375),Point(250, 500))
    rect.setFill(color_rgb(255, 0, 0))
    rect.draw(win)
```


i=0 #inicio en el indice del automata
estadoActual = automata.q0
while True:
siguienteJugada = automata.camino[i]
print("Se encuentra en la casilla: " + estadoActual+ "\n" + "La casilla a la que brinca es: " + siguienteJugada)
i += 1
movimiento = chess.Move.from_uci(casillas[estadoActual]+casillas[siguienteJugada])
estadoActual = siguienteJugada
if siguienteJugada in automata.F:
print("Cadenas Aceptadas")
break
dibujar()