Facultad de Ciencias
UNAM

Inteligencia artificial

Carlos Eduardo González Arceo
Cecilia Reyes Peña
Karem Ramos Calpulalpan
Tania Michelle Rubí Rojas

22 de febrero de 2024

Equipo: I.A. Nexus
Práctica 2

Ricardo Bernabé Nicolás
ricardo_bernabe@ciencias.unam.mx

Carlos Eduardo González Arceo
carlos.ed0205@ciencias.unam.mx

Alvaro Hernández Hernández
draco_90@ciencias.unam.mx

José Carlos Buenrostro Rueda
jcbrueda@ciencias.unam.mx

Índice de Scripts

scripts/main.py 3
scripts/main.py 3
scripts/main.py 4
scripts/main.py 5
scripts/main.py 6
1. Código completo del agente que resuelve un laberinto 7
2. Salida del programa 9

El primer cambio a mencionar es la importación del paquete collections para poder imprimir la matriz auxiliar: visitado que se usa para almacenar la ruta seguida por el agente

```
import collections

laberinto = [
    ["E", 0, 1, 0],
    [1, 0, 1, 0],
    [0, 0, 0, 0],
    [1, 1, 0, "S"]
]

# Creamos una matriz auxiliar de tamaño igual al laberinto, para almacenar los puntos visitados
visitado = [[False] * len(laberinto[0]) for _ in range(len(laberinto))]
```

como es de esperar en el momento en que el agente tiene que realizar un movimiento se van a agregar dos características:

- marcar la casilla como visitada
- indicar si el movimiento fue válido o no

además de que se verifica que la casilla sea una casilla a la que el agente se puede mover y que no sea una casilla visitada.

```
class Agente:
    # Constructor del agente en una posición de la matriz
    def __init__(self, posicion):
        self.posicion = posicion  # La posición inicial del agente

    # Función que permite mover al agente dentro de la matriz
    def mover(self, direccion, laberinto):
        x, y = self.posicion
        if direccion == "arriba" and x > 0 and laberinto[x-1][y] != 1 and not visitado[x-1][y]:
            visitado[x-1][y] = True
            self.posicion = [x-1, y]
            return 1
        elif direccion == "abajo" and x < len(laberinto) - 1 and laberinto[x+1][y] != 1 and not visitado[x+1][y]:
            self.posicion = [x+1, y]
            visitado[x+1][y] = True
            return 1
        elif direccion == "izquierda" and y > 0 and laberinto[x][y-1] != 1 and not visitado[x][y-1]:
            self.posicion = [x, y-1]
            visitado[x][y-1] = True
            return 1
        elif direccion == "derecha" and y < len(laberinto[0]) - 1 and laberinto[x][y+1] != 1 and not visitado[x][y+1]:
            self.posicion = [x, y+1]
            visitado[x][y+1] = True
            return 1
        else:
            return 0  # Indica que el movimiento no fue válido
```

además de esas integraciones se hicieron dos funciones auxiliares:

encontrar_salida: Como se poniendo a prueba el agente con varios laberintos de diferentes tamaños se tuvo la necesidad de encontrar la casilla de entrada. Además para no tener que pasar el punto de inicio al momento de hacer la instancia del agente con: (0.0)

parse_laberinto: Para evitar encontrar otros caminos no válidos se obtiene el valor a perse de todo el laberinto

```
# Función auxiliar, recibe un laberinto y regresa la posición de entrada
def encontrar_entrada(laberinto):
    for fila in range(len(laberinto)):
        for columna in range(len(laberinto[0])):
            if laberinto[fila][columna] == "E":
                return [fila, columna]
    return None

# Función auxiliar, recibe un laberinto y lo regresa estandarizado
def parse_laberinto(laberinto):
    for fila in range(len(laberinto)):
        for columna in range(len(laberinto[0])):
            if laberinto[fila][columna] == '0':
                laberinto[fila][columna] = 0
            if laberinto[fila][columna] == '1':
                laberinto[fila][columna] = 1
    return laberinto
```

Hacemos las llamadas para:

- parsear el laberinto
- encontrar la entrada del laberinto
- crear un agente en la entrada del laberinto y se marca esa posición como visitada automáticamente
- asignamos a la variable ruta la estructura para guardar la ruta seguida por el agente a la salida, si es que la hay.

```
# Hacemos parsing del laberinto para estandarizar los valores '0' y '1' a 0 y 1
laberinto = parse_laberinto(laberinto)
# Obtenemos la posición de entrada del laberinto
entrada = encontrar_entrada(laberinto)
print(f"Entrada: {entrada}")
# Crear una instancia de la clase Agente
agente = Agente(entrada)
# Marcamos la posición inicial como visitada en la matriz de visitados
visitado[agente.posicion[0]][agente.posicion[1]] = True
ruta = collections.deque([])

# Llamar a la función para encontrar la salida
encontrar_salida(agente, laberinto, ruta)
```

Finalmente la función de encontrar salida

- se establecen los movimientos permitidos
- se comienza con todos los movimientos posibles del agente desde donde se encuentra y lo primero que se revisa es si el agente se encuentra en la salida.
- de no se el caso de estar en la salida, el agente tratará de moverse en la dirección hacia una casilla válida, de encontrar una casilla válida en la dirección actual se hará una llamada recursiva a la función encontrar_salida y de no encontrar un salida tomando ésa dirección se deshace el movimiento o se regresa al agente a la casilla válida anterior y aquí es en donde se hace el backtrack

```
def encontrar_salida(agente, laberinto, ruta_seguida):
    movimientos = ["arriba", "abajo", "izquierda", "derecha"]
    ruta_seguida.append(agente.posicion.copy())  # Almacena la posición
   actual en la ruta
    contador = 0

    for movimiento in movimientos:
        if laberinto[agente.posicion[0]][agente.posicion[1]] == "S":
            print("Encontré la salida :D")
            print("Ruta seguida por el agente:", ruta_seguida)
            exit()

        elif agente.mover(movimiento, laberinto) != 0:
            encontrar_salida(agente, laberinto, ruta_seguida)
            agente.mover(movimiento, laberinto)  # Deshacer el movimiento
            ruta_seguida.pop()  # Deshacer el movimiento en la ruta

        elif agente.mover(movimiento, laberinto) == 0:
            contador += 1
            if contador == 4:
                ruta_seguida.pop()
                if not ruta_seguida:
                    print("No existe una salida :(")
                    exit()
                ultimo = ruta_seguida[-1]
                ruta_seguida.pop()
                back = Agente([ultimo[0], ultimo[1]])
                encontrar_salida(back, laberinto, ruta_seguida)
```

Script 1: Código completo del agente que resuelve un laberinto

```
import collections

laberinto = [
    ["E", 0, 1, 0],
    [1, 0, 1, 0],
    [0, 0, 0, 0],
    [1, 1, 0, "S"]
]

# Creamos una matriz auxiliar de tamaño igual al laberinto, para almacenar los puntos visitados
visitado = [[False] * len(laberinto[0]) for _ in range(len(laberinto))]

class Agente:
    # Constructor del agente en una posición de la matriz
    def __init__(self, posicion):
        self.posicion = posicion  # La posición inicial del agente

    # Función que permite mover al agente dentro de la matriz
    def mover(self, direccion, laberinto):
        x, y = self.posicion
        if direccion == "arriba" and x > 0 and laberinto[x-1][y] != 1 and not visitado[x-1][y]:
            visitado[x-1][y] = True
            self.posicion = [x-1, y]
            return 1
        elif direccion == "abajo" and x < len(laberinto) - 1 and laberinto[x+1][y] != 1 and not visitado[x+1][y]:
            self.posicion = [x+1, y]
            visitado[x+1][y] = True
            return 1
        elif direccion == "izquierda" and y > 0 and laberinto[x][y-1] != 1 and not visitado[x][y-1]:
            self.posicion = [x, y-1]
            visitado[x][y-1] = True
            return 1
        elif direccion == "derecha" and y < len(laberinto[0]) - 1 and laberinto[x][y+1] != 1 and not visitado[x][y+1]:
            self.posicion = [x, y+1]
            visitado[x][y+1] = True
            return 1
        else:
            return 0  # Indica que el movimiento no fue válido

def encontrar_salida(agente, laberinto, ruta_seguida):
    movimientos = ["arriba", "abajo", "izquierda", "derecha"]
    ruta_seguida.append(agente.posicion.copy())  # Almacena la posición actual en la ruta
    contador = 0

    for movimiento in movimientos:
        if laberinto[agente.posicion[0]][agente.posicion[1]] == "S":
            print("Encontré la salida :D")
            print("Ruta seguida por el agente:", ruta_seguida)
            exit()
```

```
50
51          elif agente.mover(movimiento, laberinto) != 0:
52              encontrar_salida(agente, laberinto, ruta_seguida)
53              agente.mover(movimiento, laberinto)  # Deshacer el movimiento
54              ruta_seguida.pop()  # Deshacer el movimiento en la ruta
55
56          elif agente.mover(movimiento, laberinto) == 0:
57              contador += 1
58              if contador == 4:
59                  ruta_seguida.pop()
60                  if not ruta_seguida:
61                      print("No existe una salida :(")
62                      exit()
63                  ultimo = ruta_seguida[-1]
64                  ruta_seguida.pop()
65                  back = Agente([ultimo[0], ultimo[1]])
66                  encontrar_salida(back, laberinto, ruta_seguida)
67
68      print("iiiiii no encontré la salida")
69
70  # Función auxiliar, recibe un laberinto y regresa la posición de entrada
71  def encontrar_entrada(laberinto):
72      for fila in range(len(laberinto)):
73          for columna in range(len(laberinto[0])):
74              if laberinto[fila][columna] == "E":
75                  return [fila, columna]
76      return None
77
78  # Función auxiliar, recibe un laberinto y lo regresa estandarizado
79  def parse_laberinto(laberinto):
80      for fila in range(len(laberinto)):
81          for columna in range(len(laberinto[0])):
82              if laberinto[fila][columna] == '0':
83                  laberinto[fila][columna] = 0
84              if laberinto[fila][columna] == '1':
85                  laberinto[fila][columna] = 1
86      return laberinto
87
88  # Hacemos parsing del laberinto para estandarizar los valores '0' y '1' a 0
        y 1
89  laberinto = parse_laberinto(laberinto)
90  # Obtenemos la posición de entrada del laberinto
91  entrada = encontrar_entrada(laberinto)
92  print(f"Entrada: {entrada}")
93  # Crear una instancia de la clase Agente
94  agente = Agente(entrada)
95  # Marcamos la posición inicial como visitada en la matriz de visitados
96  visitado[agente.posicion[0]][agente.posicion[1]] = True
97  ruta = collections.deque([])
98
99  # Llamar a la función para encontrar la salida
100 encontrar_salida(agente, laberinto, ruta)
```

Script 2: Salida del programa

```
Python 3.11.2 (main, Mar 13 2023, 12:18:29) [GCC 12.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
Entrada: [0, 0]
Encontre la salida :D
Ruta seguida por el agente: deque([[0, 0], [0, 1], [1, 1], [2, 1], [2, 2], [3, 2], [3, 3]])
```