Escuela Superior de Computo (ESCOM)

Instituto Politécnico Nacional

Paradigmas de Programación

Olivares Cruz Victor Manuel

Práctica 9

Fecha de entrega: 22/05/2024

Instrucciones

Instrucciones

Diseña y desarrolla una clase que represente a un robot, el cual implemente un método que resuelva lo siguiente.

El robot está sentado en la esquina superior izquierda de una rejilla con r filas y c columnas. El robot solo se puede mover en dos dimensiones: derecha y abajo. Algunas de las celdas están bloqueadas para movimiento, por lo que no se puede pasar por ellas. Diseña un algoritmo que encuentre una ruta desde la esquina superior izquierda a la esquina inferior derecha.

Lenguaje:
Python o Java.

Main.py

```python
from robot import Robot

def main():
    grid = [
        [0, 0, 1],
        [0, 1, 0],
        [0, 0, 0]
    ]

    robot = Robot(grid)
    path = robot.find_path()

    if path:
        print("Ruta encontrada:", path)
    else:
        print("No hay ruta disponible.")

if __name__ == "__main__":
    main()
```

Robot.py

```python
class Robot:
    def __init__(self, grid):
        self.grid = grid
        self.rows = len(grid)
        self.cols = len(grid[0])
        self.path = []
        self.visited = [[False] * self.cols for _ in range(self.rows)]

    def is_valid(self, x, y):
        return 0 <= x < self.rows and 0 <= y < self.cols and not self.grid[x][y]
and not self.visited[x][y]
```

```
    def find_path(self):
        if self.dfs(0, 0):
            return self.path
        else:
            return None

    def dfs(self, x, y):
        if x == self.rows - 1 and y == self.cols - 1:
            self.path.append((x, y))
            return True

        if self.is_valid(x, y):
            self.visited[x][y] = True
            self.path.append((x, y))

            # Moverse a la derecha
            if self.dfs(x, y + 1):
                return True

            # Moverse hacia abajo
            if self.dfs(x + 1, y):
                return True

            # Si no se encuentra una ruta, hacer backtrack
            self.path.pop()
            self.visited[x][y] = False

        return False
```

Métodos:

- **__init__(self, grid):**

  Inicializa la rejilla, las dimensiones, la lista path y la matriz visited.

- **is_valid(self, x, y):**

  Verifica si una celda (x, y) es válida: dentro de los límites, no bloqueada y no visitada.

- **find_path(self):**

  Inicia la búsqueda de la ruta desde (0, 0).

  Retorna self.path si se encuentra una ruta, None si no.

- dfs(self, x, y):

  Implementa la búsqueda en profundidad recursiva.

  Marca la celda como visitada y trata de moverse a la derecha y hacia abajo.

  Hace backtracking si no encuentra una ruta y desmarca la celda.

Prueba de ejecución:

```
PS C:\Users\Waffle\Documents\ESCOM 4TO SEMESTRE\Paradigmas\Practica 9> & C:/U
in.py"
Ruta encontrada: [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2)]
```