UNIVERSIDAD CATOLICA DEL MAULE

Proyecto Tom y Jerry

Programación ECI-121

Profesor: Hugo Araya Carrasco

Nombres: Victor A. Farias y Diego A. Fuentes

05/12/2024

Indice

1

Introducción

Se nos introduce un problema representando como referencia a los protagonistas de la caricatura Tom y Jerry en él nos plantean en términos matemáticos una matriz siendo un arreglo rectangular, cada elemento de la matriz representado por un símbolo que los caracteriza. La solución que implementa será en Python 3 con una entrada de datos externos del programa principal con la característica que tendrá una salida en un archivo (.txt). El programa asegura reglas establecidas para el uso correcto del proyecto.

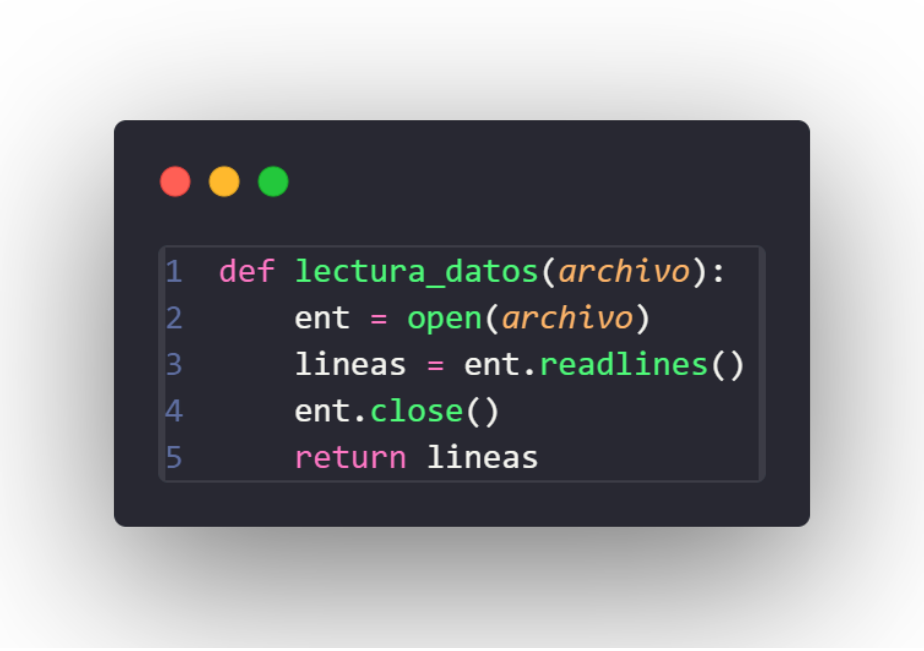
Diseño de la solución

Nuestra solución consta de once funciones, estas funciones presentan varias de ellas anidadas, esto podría tener una versión más optimizada utilizando otras estructuras y técnicas más avanzadas, pero optamos ésta para utilizar conceptos más sencillos y más manejables.

Funciones

Lectura de datos:

Utilizamos la versión más simple para abrir un archivo, al usar este método peligramos de la pérdida de archivos, por eso utilizamos el método close, ¿hay otras maneras?, si, como por ejemplo “with” es más seguro y limpio, ya que no tienes que preocuparte de cerrar el archivo manualmente, evitando posibles errores si el programa se interrumpe, pero optamos esta estructura por la experiencia ya adquirida de usarlo.



```
def lectura_datos(archivo):
    ent = open(archivo)
    lineas = ent.readlines()
    ent.close()
    return lineas
```

Leer las dimensiones:

Una de las funciones clave es leer las dimensiones, nos permite graficar o dimensionar nuestro entorno para resolver los obstáculos que presenta Tom para capturar a Jerry, este método es propenso a errores por ello se implementará en funciones siguientes para alertar sobre lo ya mencionado.

```
1  def leer_dimensiones(linea):
2      dimensiones = linea
   .split()
3      m = int(dimensiones[0
   ])  # Leer las dimensiones
4      n = int(dimensiones[1])
5      return m, n
```

Leer posiciones:

Esto nos permite poder tomar los datos en el archivo.txt, y poder manejarlo esto dará orden a todas las funcione siguientes, el inconveniente sería que nuestra función es estática y solo que el archivo txt debe venir correctamente escrito, de lo contrario o no arrojará error o tomará los datos equivocados entorpeciendo las siguientes funciones.

```python
def leer_posiciones(linea):
    posiciones = linea.split()
    tom = [int(posiciones[0]), int(posiciones[1])]  # Posición de Tom
    jerry = [int(posiciones[2]), int(posiciones[3])]  # Posición de Jerry
    return tom, jerry
```

Leer obstaculo:

Tomando los datos de la función anterior tomará la cantidad de obstáculos que presenta nuestra matriz, junto con la posición de esta.

```python
def leer_obstaculos(lineas, k):
    obstaculos = []
    for i in range(3, 3 + k):
        coords = lineas[i].split()
        obstaculos.append([int(coords[0]), int(coords[1]), int(coords[2]), int(coords[3])])  # Obstáculos
    return obstaculos
```

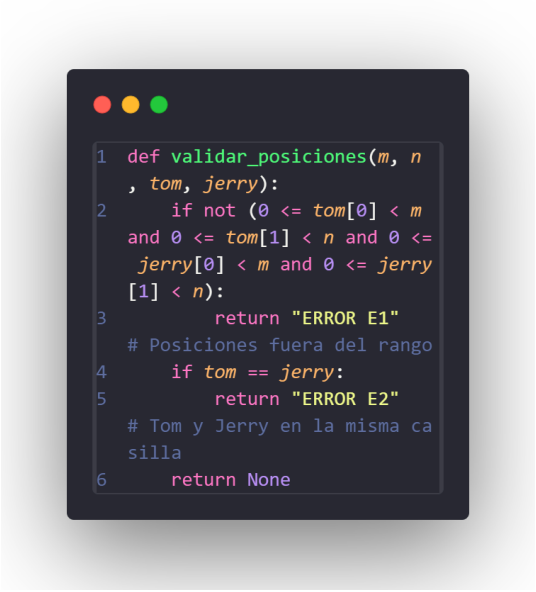
Validar dimensiones:

Para esto se inicia un ciclo con las reglas necesarias para evitar errores si cumple con los requisitos entonces nos devolverá el nombre del error.

```python
def validar_dimensiones(m, n):
    if m <= 0 or n <= 0:
        return "ERROR E0"  # Dimensiones no válidas
    return None
```

Validar posiciones

Al tomar las ubicaciones de Tom y Jerry estas son propensas a tener errores, lo cual para ello las posiciones pasa por una validación si estas cumplen una condición de error, nos retornará el mensaje del error.



```python
def validar_posiciones(m, n, tom, jerry):
    if not (0 <= tom[0] < m and 0 <= tom[1] < n and 0 <= jerry[0] < m and 0 <= jerry[1] < n):
        return "ERROR E1"  # Posiciones fuera del rango
    if tom == jerry:
        return "ERROR E2"  # Tom y Jerry en la misma casilla
    return None
```

Validación de obstáculos:

La validación siendo la más compleja a este punto con una estructura anidada de bucles y condiciones, esto puede llegar a generar confusiones, esto cuenta con más formas de hacerlo, pero esta estructura nos es la más viable y más manejable.

```python
def validar_obstaculos(m, n, obstaculos):
    casillas_obstaculos = []
    for obstaculo in obstaculos:
        x1, y1, x2, y2 = obstaculo
        if not (0 <= x1 < m and 0 <= x2 < m and 0 <= y1 < n and 0 <= y2 < n):
            return "ERROR E3"
        # Coordenadas no válidas
        if x1 > x2 or y1 > y2:
            return "ERROR E4"
        # Vértices no válidos
        for x in range(x1, x2 + 1):
            for y in range(y1, y2 + 1):
                if [x, y] in casillas_obstaculos:
                    return "ERROR E5"
                # Obstáculos solapados
                casillas_obstaculos.append([x, y])
    return casillas_obstaculos

```

Validar posiciones y obstáculos:

El objetivo de esta función es verificar si Tom o Jerry están en en la posición exacta en un obstáculo, el cual no puede pasar y en el caso que este, significa que nos arrojará un error el cual debemos retornar.

```
1  def
   validar_posiciones_en_obsta
   culos
   (tom, jerry,
   casillas_obstaculos):
2      if tom in
   casillas_obstaculos or
   jerry in
   casillas_obstaculos:
3          return "ERROR E6"
   # Tom o Jerry están en un o
   bstáculo
4      return None
5
```

Dibujar casa:

Con esta función nos permitirá dibujar nuestra matriz, tomando todos los datos recolectados y lo datos sobrantes se transformaran en dato vacío representado por la letra “o”.

Tom está representado como “T” y Jerry como “J”, los obstáculos están representados como “x”.

```
1  def dibujar_casa(m, n, tom, jerry,
   obstaculos):
2      casa = []
3      for i in range(m):
4          fila = []
5          for j in range(n):
6              fila.append('o')
   # Casilla vacía
7          casa.append(fila)
8
9      casa[tom[0]][tom[1]] = 'T'
   # Marca posición de Tom
10     casa[jerry[0]][jerry[1]] = 'J'
   # Marca posición de Jerry
11
12     for obstaculo in obstaculos:
13         x1, y1, x2, y2 = obstaculo
14         for x in range(x1, x2 + 1):
15             for y in range(y1, y2 + 1
   ):
16                 casa[x][y] = 'x'
   # Marcar obstáculos
17     return casa
```

Validar todo

Como última capa de verificación utilizamos todas las variables ya creadas para asegurarnos que nuestro programa pueda seguir las reglas ya establecidas.

```python
def validar_todo(m, n, tom, jerry, obstaculos):
    # Validamos dimensiones
    error = validar_dimensiones(m, n)
    if error:
        escribir_salida(error, True)
    else:

# Validamos las posiciones de Tom y Jerry
        error = validar_posiciones(m, n, tom, jerry)
        if error:
            escribir_salida(error, True)
        else:
            # Validamos los obstáculos
            casillas_obstaculos = validar_obstaculos(m, n, obstaculos)
            if isinstance(casillas_obstaculos, str):
# Si hay un error en los obstáculos
                escribir_salida(casillas_obstaculos, True)
            else:

# Validamos si Tom o Jerry están en un obstáculo
                error = validar_posiciones_en_obstaculos(tom, jerry, casillas_obstaculos)
                if error:
                    escribir_salida(error, True)
                else:

# Si todo es válido, dibujamos la casa y escribimos la salida
                    casa = dibujar_casa(m, n, tom, jerry, obstaculos)
                    escribir_salida(casa, False)

```

Escribir salida:

Ya con los resultados obtenidos, los retornamos en un nuevo archivo de salida llamado error.txt que va a escribir errores y el resultado.

```python
def escribir_salida(contenido, es_error):
    sal = open('error.txt', 'w')
    if es_error:
        sal.write(contenido + '\n')
# Escribir el error
    else:
        for fila in contenido:
# Escribir la casa
            for casilla in fila:
                sal.write(casilla)
            sal.write('\n')
    sal.close()
```

Llamar las funciones:

Utilizamos el “if __name__ == ‘__main__’”, utilizamos este para mayor control de llamado de funciones.

```python
if __name__ == "__main__":
    lineas = lectura_datos('casa.txt')
    m, n = leer_dimensiones(lineas[0])
    tom, jerry = leer_posiciones(lineas[1])
    k = int(lineas[2])  # Número de obstáculos
    obstaculos = leer_obstaculos(lineas, k)
    validar_todo(m, n, tom, jerry, obstaculos)
```

Limitaciones

El programa cuenta con las instrucciones para seguir las reglas del formato pero cuenta con las siguientes limitaciones.

Manejo erróneo del archivo:

No maneja escenarios en los que el archivo “casa.txt” esté corrupto o incompleto, más allá de validar los formatos básicos.

Limitación de datos:

No optimiza el procesamiento de obstáculos si el número de estos es extremadamente grande (afectaría el rendimiento).

Soluciones:

1. Leer directamente datos incorrectos sin detener la ejecución.
2. Implementar optimizaciones para manejar grandes cantidades de datos.

Manual de usuario

1. Crear un archivo llamado **casa.txt** con el siguiente formato:
    - **Primera línea:** Tamaño de la casa (número de filas y columnas).
    - **Segunda línea:** Coordenadas de Tom y Jerry.
    - **Tercera línea:** Número de obstáculos.
    - **Siguientes líneas:** Coordenadas de los obstáculos.
2. Ejecutar el programa con un entorno que soporte Python 3.
3. Si los datos son válidos, el programa imprimirá la representación de la casa; en caso contrario, mostrará un mensaje de error.

Conclusiones

El programa cumple con los requisitos establecidos, validando correctamente las dimensiones de la casa, las posiciones de Tom y Jerry, y los obstáculos. Además, genera una representación clara de la casa o un mensaje de error según corresponda.

Durante el desarrollo, se utilizaron estructuras de datos simples y un diseño modular para dividir el problema en subproblemas, lo que facilitó su implementación y comprensión. Sin embargo, se identificaron limitaciones, como la dependencia de un archivo bien formateado y la detección de un solo error a la vez. Estas áreas pueden mejorarse en futuras versiones.

El proyecto permitió reforzar conocimientos en Python, como manejo de listas y validaciones, y destacó la importancia de seguir un flujo claro y organizado al desarrollar soluciones computacionales.