INFORME PROYECTO 3.

Fabian Garcia , Marcelo Vidal, Álvaro Figueroa.

24 de junio de 2024.

INFO 1139, Ingeniería Civil en Informática.

Victor Valenzuela.

Introducción

En este proyecto se resolverán las problemáticas planteadas sobre la creación de un mapa gigante y un mini mapa por el cual se pueda navegar y se desplace el mapa grande, el movimiento aleatorio de los 100 robots y que 10 de estos sean capaces de leer la información que está oculta en el mapa para luego enviarla mediante puertos seriales para que los datos enviados a otro archivo .py sean capaces de dibujar una gráfica que visualiza el tipo y cantidad de recurso.

Además de un segundo problema más pequeño en donde se utilizan recursos similares a los utilizados en el proyecto anterior pero esta vez tenemos un fondo el cual se va desvelando con el paso del robot.

Para esto hemos utilizado los recursos disponibles en plataforma como los códigos disponibles de tutorial y apoyo además de los videos de referencia.

Todo esto se ha logrado utilizando diferentes librerías ya vistas además de la librería serial para la conexión de los puertos y la transferencia de datos, también se ha utilizado python 2.7 y el programa programa VSPE (Virtual Serial Port Emulator).

Orden por Funcionalidad

Problema 1.

Problema 2.

Desarrollo Problema 1 (Interfaz del Mapa Grande)

Definición de constantes a utilizar en el proyecto

```
nRes = (960,480); nt_WX = nt_HY = 32; lGo = True
nMIN_X = 0 ; nMAX_X = 6400 ; nMIN_Y = 0 ; nMAX_Y = 480; nMAX_ROBOTS = 100
nMx = nMy = 0; nMAX_ROBOTSsicensa = 10; import serial as sl
nX0 = 19 ; nY0 = 405 ; yd = 0; xd = 0
```

1. Movimiento de los 100 Robots en el mapa Gigante.

a. Creación del mapa gigante;

```
class eCelda(ct.Structure):
    _fields_ = [
        ('nT',ct.c_ubyte), # Tipo de Tile/Baldosa
        ('nF',ct.c_ubyte), # Fila de Mapa
        ('nC',ct.c_ubyte), # Columna de Mapa
        ('nR',ct.c_ubyte), # Tipo de recurso
        ('nQ',ct.c_ubyte), # cantidad de recurso
        ]
```

Comenzamos definiendo la estructura de nuestro mapa.

```
def Init_Mapa(nAncho_X,nAlto_Y):
    for nF in range(0,nMAX_Y / nt_HY):
                for nC in range(0,nMAX_X / nt_WX):
            aMap[nF][nC].nT = ra.randint(1,3)
            aMap[nF][nC].nF = nF
            aMap[nF][nC].nC = nC
             aMap[nF][nC].nR = ra.randint(1,5)
             aMap[nF][nC].nQ = ra.randint(10,50)
    return pg.Surface((nAncho_X,nAlto_Y))
```

Para la inicialización del mapa gigante utilizamos esta función a la cual le entregamos valores cuando la llamamos (nAncho_x y nAlto_Y), estos valores serán el ancho y alto de nuestra surface principal (6400 en X y 480 en Y). Dentro de la función recorremos las filas y columnas y asignamos valores aleatorios desde el 1 hasta el 3 para el tipo de tile, el tipo de

recurso desde el 1 hasta el 5, que seran; agua, acero, carbón, petróleo y aluminio, luego seleccionamos al azar desde el 10 hasta el 50 la cantidad de recurso que habrá en la tile, además de definir la fila y columna

```
def Pinta_Mapa():
    for nF in range(nMAX_Y // nt_HY):
        for nC in range(nMAX_X // nt_WX):
            if aMap[nF][nC].nT == 1:
                sWin.blit(aFig[2], (aMap[nF][nC].nC * nt_WX, aMap[nF][nC].nF *
nt_HY))
            if aMap[nF][nC].nT == 2:
                sWin.blit(aFig[3], (aMap[nF][nC].nC * nt_WX, aMap[nF][nC].nF *
nt_HY))
            if aMap[nF][nC].nT == 3:
                sWin.blit(aFig[4], (aMap[nF][nC].nC * nt_WX, aMap[nF][nC].nF *
nt_HY))
```

Para mostrar en pantalla el mapa recorremos filas y columnas para luego preguntar qué tipo de tile es, y así pintar la figura correspondiente a ese tipo de tile en la posición que corresponda en el mapa.

b. Lógica de movimiento de los 100 robots.

```
class eRobot(ct.Structure):
    __fields__ = [
        ('nF',ct.c_ushort), #Figura
        ('nX',ct.c_ushort), #Pos X
        ('nY',ct.c_ushort), #Pos Y
        ('nR',ct.c_ushort), #Rango (pasos)
        ('dX',ct.c_ushort), #direccion X
        ('dY',ct.c_ushort), #direccion Y
        ('nV',ct.c_ushort), #Velocidad
        ('Recursostiles'), # arreglo para guardar recursos y cantidad
        ('Tilescensados') # arreglo para guardar los tiles censados
    ]
```

Se define la estructura del robot para poder ser utilizados por el mapa gigante.

```
def init_Robot():
    for i in range(0,nMAX_ROBOTS):
        aBoe[i].nF = 1    # Robot Tipo 1 (no censa)
        aBoe[i].nX = (ra.randint(0,nMAX_X - nt_WX) / nt_WX) * nt_WX #
        aBoe[i].nY = (ra.randint(0,nMAX_Y - nt_HY) / nt_HY) * nt_HY #
        aBoe[i].nR = 0
        aBoe[i].dX = 0
        aBoe[i].dY = 0
        aBoe[i].nV = 0
        aBoe[i].Recursostiles = []
        aBoe[i].Tilescensados = []
    for i in range(0,nMAX_ROBOTSsicensa):
        aBoe[i].nF = 2
    return
```

Para inicializar los robots lo primero es recorrer los 100 robots que tenemos, luego les asignamos a todos que sean de tipo .nF = 1, esto significa que serán del tipo que no censa, luego les damos una posición en X y en Y al azar, iniciamos todos sus datos de movimiento en 0 ya que se modifican en nuestra función de movimiento, estos son .nR que son los pasos que dará el robot, .dX que es la dirección en el eje X, .dY la dirección en el eje Y, por último .nV que es la velocidad del robot. Tambien creamos dos arreglos vacios que serán usados para la recogida de datos por parte de los robots censadores y para finalizar la inicialización, de los 100 robots recorremos 10 y los haremos de .nF = 2, esto significa que censarán las tiles

```
def Mueve_Robot():
    for i in range(0,nMAX_ROBOTS):
        aBoe[i].nR -= 1
        if aBoe[i].nR < 0:
            aBoe[i].nR = ra.randint(0,480)
            aBoe[i].nV = ra.randint(1,3)
            nDir = ra.randint(1,5)
            if nDir == 1:
                aBoe[i].dX = +0 ; aBoe[i].dY = -1
            if nDir == 2:
                aBoe[i].dX = +1 ; aBoe[i].dY = 0
            if nDir == 3:
                aBoe[i].dX = +0 ; aBoe[i].dY = +1
            if nDir == 4:
```

```
                aBoe[i].dX = -1 ; aBoe[i].dY = +0
            if nDir == 5:
                aBoe[i].dX = +0 ; aBoe[i].dY = +0
        newX = aBoe[i].nX + aBoe[i].dX * aBoe[i].nV
        newY = aBoe[i].nY + aBoe[i].dY * aBoe[i].nV

        if 0 <= newX < nMAX_X - nt_WX and 0 <= newY < nMAX_Y - nt_HY:
            aBoe[i].nX = newX
            aBoe[i].nY = newY
```

Para el movimiento de los robots implementamos la función Mueve_Robot(). Esta función recorre nuestros 100 robots, vamos decrementando los pasos ("aBoe.nR") del robot en 1 hasta llegar a -1 para luego volver a asignar una cantidad de pasos desde el 0 hasta el 500 y una velocidad del 1 hasta el 3 al azar, luego seleccionamos, también al azar, una de las 5 posibles direcciones, las cuales son Norte, Este, Sur, Oeste y detenido. Luego definimos las nuevas X e Y del robot respectivamente calculando su posición actual más la dirección multiplicando la velocidad, para finalizar la lógica de movimiento comprobamos si el robot está dentro de los límites del mapa y definimos que la posición actual sea la previamente calculada.

2. Funcionamiento del Mini Mapa.

a. Visualización del Mini Mapa.

```
def Pinta_MiniMapa():
    xp = 0; xy = 0
    sWin.blit(aFig[5],(15,400))
    for i in range(0,nMAX_ROBOTS):
        xp = int(923/float(6400)*aBoe[i].nX) + 20
        xy = int(65/float(480)*aBoe[i].nY) + 404
        if aBoe[i].nF == 1:
            sWin.blit(aFig[7],(xp,xy))
        if aBoe[i].nF == 2:
            sWin.blit(aFig[6],(xp,xy))
    return
```

Para mostrar el minimapa y los robots con sus respectivas figuras, se usa la funcion Pinta_MiniMapa(). Comenzamos definiendo dos variables que usaremos luego en la función, estas serán la posición en X e Y del robot en el minimapa. Luego mostramos la figura que corresponde al minimapa en pantalla en las coordenadas 15,400 (X,Y), seguidamente recorremos los 100 robots para convertirlos a escala usando **Interpolación** para cada eje (xy), tomando valores de posicionamiento en el rango en el que se muestran en el minimapa para asegurarnos de que se muestre el punto azul (aFig[7]) para los robots no censadores (aBoe.nF = 1), así mismo con el punto rojo (aFig[6]) para los robots censadores (aBoe.nF = 2) respectivamente.

b. Scroll minimapa

```
def UpDate_Scroll_Mapa(nMx,nMy):
    global xd, yd
    if 20 <= nMx <= 943 and 400 <= nMy <= 469:
        xd = int((nMx - 20) / 923.0 * (nMAX_X - nRes[0]))
        yd = int((nMy - 404) / 65.0 * (nMAX_Y - nRes[1]))
        pg.display.set_caption('[Coord Mapa]-> X: %d - Y: %d' % (xd, yd))
    return xd,yd
```

Para la implementación del desplazamiento del mini mapa utilizamos la función UpDate_Scroll_Mapa(nMx, nMy), esta función recibe la posición del mouse. Para comenzar llamamos a las constantes globales xd e yd, para después detectar si el mouse está en el rango de píxeles en donde tenemos pintado el mini mapa y nuestra subsurface para luego deshacer la interpolación con xd e yd, calculando así las coordenadas actuales del mapa gigante. Para finalizar escribe como título de la ventana de pygames las coordenadas actuales en las que nos encontramos

c. Superficie usada para el Scroll.

```
def Pinta_subMapa():
    global xd,yd,nX0,nY0
    sWin.blit(mapa.subsurface((xd,yd,924,64)),(nX0,nY0))
    return
```

En esta función determinamos la superficie en la que el mouse podrá hacer click para scrollear en el minimapa, para esto creamos una subsurface que estará dentro de nuestra surface de 6400,480 para realizar lo anteriormente mencionado se utiliza la función .subsurface(), en la cual determinamos tanto la posición en X e Y (xd e yd), como la superficie que ocupará en el minimapa(924 de ancho y 64 de alto), luego tenemos nX0 y nY0 que serán las coordenadas en X e Y donde se coloque la esquina superior izquierda del subsurface.

3. Registro de Recursos y Cantidades.

Para hacer que se registren los recursos y cantidades utilizamos el siguiente bloque de codigo dentro de la función Mueve_Robot():

```
def Mueve_Robot():
        ...
        if aBoe[i].nF == 2:
            nF = aBoe[i].nY // nt_HY
            nC = aBoe[i].nX // nt_WX
             tile_actual = aMap[nF][nC]
            if (nF,nC) not in aBoe[i].Tilescensados:
                aBoe[i].Recursostiles.append({
                'recurso': tile_actual.nR,
                'cantidad': tile_actual.nQ
                })
                aBoe[i].Tilescensados.append((nF, nC))
                enviar_datos_serial(i,tile_actual.nR,tile_actual.nQ,nF,nC)
return
```

Nos aseguramos de que para los robots que sean de tipo censador (aBoe.nF = 2) obtengamos la fila y columna actual del robot censador, esto lo hacemos dividiendo su posición actual por 32, que es el tamaño de los tiles, seguidamente definimos que la **`tile_actual`** será la misma tile del mapa. Luego corroboramos de que el tile que estamos censando no haya sido censado, si es así, agregamos el recurso al arreglo de recursos junto

con la cantidad de toneladas que contenga. Finalmente agregamos la fila y columna del tile al arreglo de tiles ya censados del robot, luego llamamos a la función para enviar los datos por serial y le entregamos los datos del robot que censo y los datos de la tile actual.

4. Manejo de datos por serial.

a. Puerto serial de envío de datos

```
conn = sl.Serial(port='COM2', baudrate=9600, timeout=1)
```

En esta variable declaramos la conexión que hará el **envío por serial** con sus respectivas características desde el **mapa con los robots censadores**, en esta parte disponemos del puerto serial `'COM2'` usado en el software **Virtual Serial Ports Emulator**

b. Puerto serial de recepción de datos

```
conn = sl.Serial(port='COM3',baudrate= 9600, timeout=1)
```

Por el otro lado tenemos la conexión desde el código con el **Panel de control** con sus respectivas características, disponiendo del puerto serial `'COM3'` usado en el software ya mencionado anteriormente.

c. Función encargada de enviar los datos en el código del mapa

```
def enviar_datos_serial(robot_id, recurso, cantidad,fila,columna):
    data = [robot_id, recurso, cantidad, fila, columna]
    data = bytearray(data)
```

```
            conn.write(data)
```

Para realizar el envío de los datos tenemos esta función que recibe el id del robot, el recurso y cantidad de este de la tile además del número de fila y columna de la tile, luego definimos que nuestra data será un arreglo con todos estos números enteros. Para realizar correctamente la transferencia los convertimos a un arreglo de bytes que tendrá 5 bytes de largo y para finalizar enviamos la data mediante la conexión serial.

d. Función encargada de recibir los datos por el Panel.

```
def recibir_datos_serial():
    global nMx,nMy,lGo
    with open('Recursos Proyecto #3\Problema 1\data.dat', 'a') as file:
        while lGo:
            if conn.in_waiting > 0:
                data = conn.read(5)
                data = [ord(dato) for dato in data]
                id = data[0]
                recurso = data[1]
                cantidad = data[2]
                fila = data[3]
                columna = data[4]
                datos = 'idrobot:{}, recurso:{}, cantidad:{}, fila:{},
columna:{}\n'.format(id, recurso, cantidad,fila,columna)
                file.write(datos + '\n')
                init_lineas(id, recurso, cantidad)
```

Esta función tiene varias funcionalidades, funciona como nuestro ciclo principal, para escribir la data en el archivo data.dat y para recibir los datos necesarios para pintar las líneas, ahora nos enfocaremos en la recepción de datos, primeramente mientras nuestra conexión está abierta leemos el arreglo de bytes que tiene una longitud de 5 bytes, seguidamente creamos un ciclo para recorrer la data y extraer cada dato para poder formatearlo de hexadecimal a caracteres con el método “ord()”.

5. Visualización de la cantidad de recursos en el panel.

```
class eReg(ct.Structure):
    fields = [
                ('nB',ct.c_ushort), # ID Robot
                ('nPos',ct.c_ushort), # pos en el panel
                ('nAltura',ct.c_ushort), # Altura de la linea en el panel
                ('nR',ct.c_ushort), # Recursos
                ('nQ',ct.c_ushort), # Qty
            ]
```

Para comenzar lo primero es definir la estructura que tendrán los datos de nuestros robots.

```
def init_eReg():
    for i in range(0,nMAX_ROBOTSsicensa):
            aRegs[i].nB = i
            aRegs[i].nPos = (13+(83*i),130)
            aRegs[i].nAltura = (13+(83*i),130)
            aRegs[i].nR = 0
            aRegs[i].nQ = 0
```

Iniciamos a nuestros robots recorriendolos y definiendo que el número de ciclo será el mismo al id del robot, esto hará que el ciclo 0 sea el robot 0, luego el ciclo 1 será el robot 1 y así sucesivamente, luego iniciamos la posición que tendrá la línea en él panel como una tupla con los valores de X e Y, estos se calculan con valores de píxeles calculados y para el eje X se multiplica por el ID del robot, así se mueve hacia la derecha lo necesario para que calce en el panel. Luego iniciamos la altura que tendrá la línea de la misma manera para tener la misma posición, finalizamos definiendo en 0 el tipo de recurso y la cantidad de recurso.

```
def init_lineas(id, recurso, cantidad):
    colores = {1 : (237, 28, 36), #1 rojo
    2 : (0, 162, 232), #2 celeste
    3 : (34, 177, 76), #3 verde
    4 : (63, 72, 204), #4 azul
    5 : (255, 201, 14)} #5 amarillo
    aRegs[id].nAltura = (13+(83*id), (130-(cantidad)))
    aRegs[id].nR = colores.get(recurso)
    aRegs[id].nQ = cantidad
```

Para la inicializacion de las lineas del panel le entregamos el id del robot, el recurso y su cantidad, luego para los colores de las líneas creamos un diccionario con sus valores RGB correspondientes para la altura de las líneas usamos los mismos valores con los que iniciamos, pero al Eje Y le restamos la cantidad de recurso que tengamos en esa celda, para el color usamos la función get() al diccionario para relacionar la key del color con el id del recurso, y finalizamos definiendo que la cantidad .nQ sea la cantidad recibida en la data de serial.

```
def pinta_lineas():
    for i in range(0,nMAX_ROBOTSsicensa):
        pg.draw.line(sWin, aRegs[i].nR, aRegs[i].nAltura, aRegs[i].nPos, width=10)
```

Para mostrar las líneas en pantalla lo que hacemos es recorrer los 10 robots (ciclos) y entregarle a la función draw.line de pygame los siguientes valores; la pantalla principal para pintarla, el tipo de recurso, la altura

que ya calculamos, la posición en el panel y luego el grosor en píxeles de la línea.

6. Escritura de datos en data.dat.

```
def recibir_datos_serial():

    global nMx,nMy,lGo

    with open('Recursos Proyecto #3\Problema 1\data.dat', 'a') as file:

                        ...

                datos = 'idrobot:{}, recurso:{}, cantidad:{}, fila:{},
            columna:{}\n'.format(id, recurso, cantidad,fila,columna)

                file.write(datos + '\n')
```

Recordando nuestra función tenemos las siguientes funcionalidades, en donde abrimos nuestro archivo data.dat utilizando la función with open, luego definimos que nuestros datos serán los datos que extrajimos anteriormente como una cadena de texto utilizando la estructura y formateando los datos como lo requiere python 2.7 para luego escribirlos en el archivo con un salto de línea

7. Carga de imágenes necesarias para la simulación.

a. Metodología carga de imágenes.

```
def Load_Image(sFile,transp = False):
    try: image = pg.image.load(sFile)
    except pg.error as message:
        raise SystemExit(message)
    image = image.convert()
    if transp:
       color = image.get_at((0,0))
       image.set_colorkey(color,RLEACCEL)
    return image
```

b. Carga de imágenes usando la función anterior.

```
def Init_Fig():
    aImg = []
      ...
    return aImg
```

Desarrollo Problema 2 (Robot descubriendo el puzle)

1. Movimiento del Robot.

```
def Mueve_Robot():

    for i in range(0,nMAX_ROBOTS): # Recorrimos todos los Robots

        aBoe[i].nR -= 1      # Decrementamos en 1 el Rango del Robot

        if aBoe[i].nR <= 0:   # Robot termino sus pasos?

            if aBoe[i].nS == 1:

                aBoe[i].nS = 2  # Cambio de estado

                aBoe[i].nR = nR_2 # Robot ESTE nR_2 pasos

                aBoe[i].dX = 1 ; aBoe[i].dY = 0

            elif aBoe[i].nS == 2:

                aBoe[i].nS = 3  # Cambio de estado

                aBoe[i].nR = nR_1 # Robot SUBE nR_1 pasos

                aBoe[i].dX = 0 ; aBoe[i].dY = -1

            elif aBoe[i].nS == 3:

                aBoe[i].nS = 4  # Cambio de estado

                aBoe[i].nR = nR_2 # Robot ESTE nR_2 pasos

                aBoe[i].dX = 1 ; aBoe[i].dY = 0

            else:

                aBoe[i].nS = 1  # Cambio de estado

                aBoe[i].nR = nR_1 # Robot BAJA nR_1 pasos

                aBoe[i].dX = 0 ; aBoe[i].dY = 1



        newX = aBoe[i].nX + aBoe[i].dX * aBoe[i].nV

        newY = aBoe[i].nY + aBoe[i].dY * aBoe[i].nV
```

En esta función se recorren el máximo de robots decrementando en 1 el rango de robot en el eje Y esto hasta que el robot termine sus paso, en este punto el robot cambia de dirección y al mismo tiempo de estado (nS = 2) aumentando sus pasos en el eje X, hasta que termine sus pasos el robot irá hacia arriba en los pasos determinados globalmente (es lo mismo con los demás pasos), cambiando al estado 3 (nS = 3). En el estado 4 (nS = 4) el robot podrá ir hacia el este **(aumentando los pasos en el eje X)** nuevamente en la misma cantidad de pasos, para finalmente volver al **estado inicial.**

```
if 0 <= newX < nRes[0] - nt_WX and 0 <= newY < nRes[1] - nt_HY:
    if newX == 607 and newY == 32:
        Reiniciar_Mapa()

    else:
        aBoe[i].nX = newX
        aBoe[i].nY = newY
```

Esto se repetirá hasta que el robot llegue al final. Una vez pase esto se llamara a la función **"Reiniciar_Mapa()"** la cual se explicara un poco más adelante (funcionamiento 7), no olvidar que antes de esto se actualizan las posiciones del robot como **"newX" y "newY" .**

2. Inicialización de los parámetros del Robot

Para inicializar los parámetros de los robots implementamos la función init_Robot junto a su estructura, que configura los valores iniciales de cada robot.

```
class eRobot(ct.Structure):
    __fields__ = [
        ('nF',ct.c_ushort), #Figura
        ('nX',ct.c_ushort), #Pos X
        ('nY',ct.c_ushort), #Pos Y
        ('nR',ct.c_ushort), #Rango
        ('nS',ct.c_ushort), #Sentido
```

```
        ('dX',ct.c_ushort), #Direccion X
        ('dY',ct.c_ushort), #Direccion Y
        ('nV',ct.c_ushort), #Velociad
        ('nC',ct.c_ushort)
]
```

Definimos al objeto robot con sus respectivos atributos de tipo Ctypes.

```
def init_Robot():
    for i in range(0,nMAX_ROBOTS):
        aBoe[i].nF = 1    # Robot Tipo 1
        aBoe[i].nX = 0    # (RA.randint(0,nRES[0] - nT_WX) / nT_WX) * nT_WX
        aBoe[i].nY = 0    # (RA.randint(0,nRES[1] - nT_HY) / nT_HY) * nT_HY
        aBoe[i].nR = nR_1 # (RA.randint(0,nRES[0] - nT_WX) / nT_WX) * nT_WX
        aBoe[i].nS = 1    # Switch por defecto
        aBoe[i].dX = 0   # Por defecto robot Direccion Este.-
        aBoe[i].dY = 1
        aBoe[i].nV = 1
        aBoe[i].nC = 1
    return
```

Pasos:

- Establecer el tipo de robot (nF).
- Definir la posición inicial en el mapa (nX, nY).
- Asignar el rango inicial de movimiento (nR).
- Configurar el estado inicial (nS).
- Establecer la dirección inicial (Este, dX, dY).
- Configurar la velocidad (nV) y contador de celdas (nC).

3. Inicialización del Mapa

Para inicializar la estructura del mapa del juego implementamos la función Init_Mapa y su respectiva estructura.

```
class eCelda(ct.Structure):
    _fields_ = [
        ('nT',ct.c_ubyte), # Tipo de Tile/Baldosa
        ('nS',ct.c_ubyte), # 0 : No se pinta - # 1 : Si se pinta
        ('nF',ct.c_ubyte), # Fila de Mapa
        ('nC',ct.c_ubyte), # Columna de Mapa
        ]


def Init_Mapa():
    for nF in range(0,nRes[1] / nt_HY):
        for nC in range(0,nRes[0] / nt_WX):
            aMap[nF][nC].nT = 1 # inicializa el mapa con la tilede acero
            aMap[nF][nC].nS = 0 # la tile aparece por defecto
            aMap[nF][nC].nF = nF # Fila de la Celda
            aMap[nF][nC].nC = nC # Colu de la Celda
    return
```

Pasos:

- Recorrer todas las filas y columnas del mapa.
- Establecer el tipo de tile predeterminado (nT).
- Configurar el estado de visibilidad predeterminado (nS).
- Asignar las coordenadas de fila (nF) y columna (nC) a cada celda.
-

4. Visualización de las figuras del robot

```
def Pinta_Robot():
```

```
    for i in range(0,nMAX_ROBOTS): # Iteramos las 8 Figuras del Robot
        if aBoe[i].nF == 1: sWin.blit(aFig[1] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 2: sWin.blit(aFig[2] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 3: sWin.blit(aFig[3] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 4: sWin.blit(aFig[4] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 5: sWin.blit(aFig[5] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 6: sWin.blit(aFig[6] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 7: sWin.blit(aFig[7] ,(aBoe[i].nX,aBoe[i].nY))
        if aBoe[i].nF == 8: sWin.blit(aFig[8] ,(aBoe[i].nX,aBoe[i].nY))
    return
```

Para pintar los robots en la pantalla implementamos la función Pinta_Robot, que dibuja los robots en sus posiciones actuales. Los pasos son:

- Recorrer cada robot y determinar la imagen correspondiente según el tipo (nF).
- Utilizar blit de Pygame para dibujar la imagen del robot en su posición (nX, nY).

5. Visualización del Mapa.

```
def Pinta_Mapa():
    for nF in range(0,nRes[1] / nt_HY):
        for nC in range(0,nRes[0] / nt_WX):
            if  nC == (aBoe[0].nX+1)/nR_2 and nF == (aBoe[0].nY+1)/nR_2:
                if aMap[nF][nC].nT == 1:
                    aMap[nF][nC].nS = 1
            if aMap[nF][nC].nS == 1:
                fondopantalla = (nC * nt_WX, nF * nt_HY, nt_WX, nt_HY)
                sWin.blit(fondo,fondopantalla, fondopantalla)
             else:
      sWin.blit(aFig[0],(aMap[nF][nC].nC*nt_HY,aMap[nF][nC].nF*nt_WX))
```

Para pintar el mapa en la pantalla implementamos la función Pinta_Mapa, que gestiona la visualización del mapa y la interacción de los robots con él. Los pasos son:

1. Recorrer todas las celdas del mapa.
2. Verificar si una celda contiene un robot y actualizar su estado de visibilidad (nS).
3. Dibujar la imagen de fondo o el tile correspondiente según el estado de la celda.

6. Reinicio del Mapa

```
def Reiniciar_Mapa():
    global fondo
    fondo = aFig[ra.randint(9,13)]
    init_Robot()
    Init_Mapa()
```

Para reiniciar el mapa del juego implementamos la función Reiniciar_Mapa, que restablece el estado inicial del juego. Los pasos son:

1. Seleccionar aleatoriamente una nueva imagen de fondo.
2. Llamar a init_Robot para reiniciar los parámetros de los robots.
3. Llamar a Init_Mapa para reiniciar los parámetros del mapa.

Conclusión.

El análisis y desarrollo de los códigos en el Mapa Gigante junto a su minimapa, el panel de control de recursos censados por los robots y el robot que descubre una foto de fondo han permitido crear un sistema eficiente para controlar y monitorear robots en un entorno simulado. Utilizando Pygame para la interfaz gráfica y la comunicación serial para interactuar con los robots entre el mapa gigante y el panel, se ha logrado un sistema robusto y funcional.

Hemos integrado componentes claves para configurar, visualizar y mover los robots en un mapa dinámico. La interfaz gráfica de Pygame permite ver en tiempo real las operaciones de los robots, facilitando la interacción del usuario y la depuración del sistema. Además, la comunicación serial asegura un intercambio de datos eficiente y constante con los robots. El uso de clases y funciones bien organizadas ha resultado en un código fácil de entender y ampliar.