

Simulación de un Invernadero Inteligente



Desarrollado por:

-Benjamín Morales

-José Sáez

(Alumnos de ICINF en práctica 1 en Laboratorio CIMUBB Semestre 2022-2)

Tabla de contenido

Introducción	3
Packages Necesarios	4
CODIGO	5
save	6
robot	12
sprites	13
funciones	14
Archivo Principal: GUI	16
Desafíos a Futuro	76

Introducción

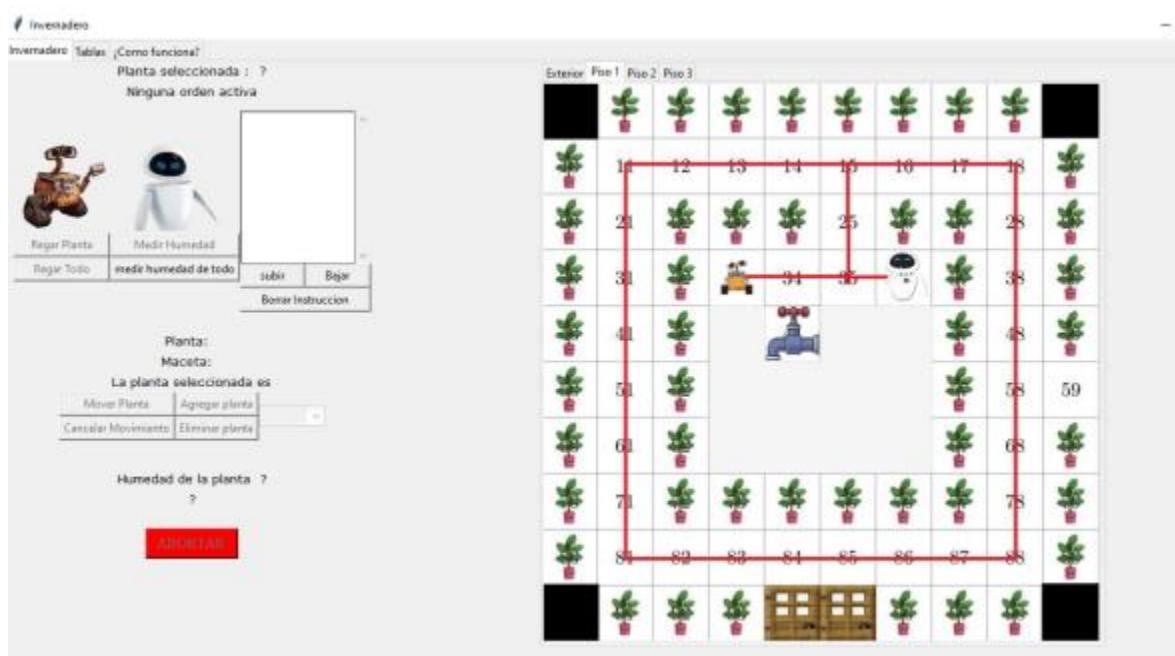
Este manual ha sido desarrollado para explicar el funcionamiento del código escrito para la creación de la simulación de un invernadero inteligente hecho en Python.

A continuación, se indicarán las librerías utilizadas, luego se explicará y mostrara el código del programa, primero el de las funciones que creamos para hacer el código más ordenado, y luego el principal, el cual contiene la lógica del programa y la interfaz gráfica.

Cabe señalar que durante la creación del programa se utilizó la versión 3.10.6 de Python

Para descargar el proyecto diríjase al siguiente enlace:

Para facilitar el entendimiento acá se muestra la interfaz del programa



Packages Necesarios

Se debe ingresar al símbolo del sistema (CMD) e ingresar los siguientes comandos:

pip install pillow

pip install tk

Para utilizar estos debe tener instalado Python y pip, Python se puede descargar en este enlace:

<https://www.python.org/downloads/>

Mientras que la instalación de pip dependerá de su sistema operativo, por lo que acá hay una guía

<https://tecnonucleous.com/2018/01/28/como-instalar-pip-para-python-en-windows-mac-y-linux/>

Pillow sirve para insertar imágenes en la interfaz gráfica, tk es la librería con la cual se construyó la interfaz gráfica, pip sirve para instalar packages de Python.

CODIGO

Durante el transcurso del programa con frecuencia se usará una variable llamada ‘pixel’, esta es un int medido en pixeles que será igual a la altura de la pantalla dividida en 10 (se eliminan los decimales), cada posición de la matriz es de tamaño ‘pixel’x‘pixel’.

El robot encargado de mover la planta es llamado ‘walle’, mientras que el robot encargado de medir humedad es llamado ‘eva’, las imágenes para representarlos también provienen de esa película.

(el subtítulo es el nombre del archivo. Tener cuidado de no confundirse con los nombres de archivo, función y variable. Para facilitar esto cuando se hable del archivo se usarán negritas, con las funciones se usarán comillas dobles y con las variables se usarán comillas simples)

Save

save sirve para que al cerrar el programa se guarden los cambios de movimiento de plantas, así como de las mismas plantas, esto no afecta a la humedad de estas ya que no se podría determinar el tiempo que transcurre al tener el programa cerrado. Para esto usaremos una base de datos interna y el gestor de base de datos sqlite, el cual creará un archivo que contendrá los datos que se necesitaran para iniciar el programa.

El código de la creación de las tablas e inserción de datos iniciales se encuentra en el archivo .txt **creacion de la base de datos**, este se deberá copiar y pegar en un archivo .py y correrlo una sola vez. Sin embargo esto no es necesario ya que en la carpeta del proyecto ya viene un archivo de tipo base de datos que tiene estos datos (**data**). De todas formas a continuación se va a mostrar el contenido del txt.

Cabe destacar que en el txt se utilizan 2 formas distintas para ingresar datos, pero funcionan de la misma forma.

```
import sqlite3

con = sqlite3.connect("data.db")
cur = con.cursor()

con.execute("CREATE TABLE planta(plant_id, expected_humidity, type_plant)")

con.execute("CREATE TABLE planta2(plant_id, expected_humidity, type_plant)")

con.execute("CREATE TABLE mapas(piso,'1','2','3','4','5','6','7','8','9','10')")
```

Primero se abre la conexión con la base de datos **data** y se crea el cursor para ejecutar las acciones. Posteriormente se crean las tablas que contienen los datos que se quieren guardar, los cuales son el número de la planta, la humedad esperada de esa planta y qué tipo de planta es. Para el mapa se guardará el piso al que corresponde y 10 cadenas de texto, en donde la Z simboliza un espacio vacío donde no puede haber plantas (en la interfaz se verá un cuadro blanco en lugar del número de la posición), la P simboliza una puerta, la Y simboliza el grifo, y la X simboliza una planta.

```

cur.execute("""INSERT INTO mapas(piso,'1','2','3','4','5','6','7','8','9','10') VALUES(0,"ZZZZZZZZZZ",
    "ZZZZZZZZZZ",
    "ZZZZZZZZZZ",
    "ZZZZZZZZZZ",
    "ZZZZZZZZZZ",
    "ZZZZZZZZZZ",
    "ZZZZPPZZZZ",
    "",
    "",
    ")""")

cur.execute("""INSERT INTO mapas(piso,'1','2','3','4','5','6','7','8','9','10') VALUES(1, ",
    "",
    "",
    "",
    "ZYZZ",
    "ZZZZ",
    "ZZZZ",
    "",
    "",
    "PP")""")

cur.execute("""INSERT INTO mapas(piso,'1','2','3','4','5','6','7','8','9','10') VALUES(2, "ZZZZZZZZZZ",
    "ZZ      ZZZ",
    "ZZ      ZZZ",
    "ZZ  YZ  ZZZ",
    "ZZ      ZZZ",
    "ZZ      ZZZ",
    "ZZ      ZZZ",
    "ZZZZZZZZZZ",
    "ZZZZZZZZZZ")""")
con.commit()
con.close()

```

Cada una de las 3 opciones del mapa debe ser una lista de 10 datos de tipo String (lista de caracteres), donde cada dato tiene 10 caracteres, esto debido a que dividiremos cada piso en una matriz de 10x10, aunque varios de esos espacios no se usaran por la diferencia de tamaño de los pisos. Hay que recordar que un espacio en blanco cuenta como carácter.

“commit” se usa para subir los datos a la base de datos y “close” para cerrar la conexión con la base de datos.

```

def conectar():
    con = sqlite3.connect("data.db")
    cur = con.cursor()
    return con, cur

def insertarDatos(plant_id, expected_humidity, type_plant):
    con, cur = conectar()
    sentencia = "INSERT INTO planta(plant_id, expected_humidity, type_plant) VALUES(?, ?, ?)"
    datos = (plant_id, expected_humidity, type_plant)
    cur.execute(sentencia, datos)
    con.commit()
    con.close()

def insertarDatos2(plant_id, expected_humidity, type_plant):
    con, cur = conectar()
    sentencia = "INSERT INTO planta2(plant_id, expected_humidity, type_plant) VALUES(?, ?, ?)"
    datos = (plant_id, expected_humidity, type_plant)
    cur.execute(sentencia, datos)
    con.commit()
    con.close()

```

Para los datos de la planta definiremos “conectar” para empezar la conexión y obtener el cursor, para entonces definir funciones para insertar los datos, los “?” se usan para reemplazar variables, al ejecutarse la orden se les asignan los valores que contiene ‘datos’. Igualmente hay que guardar los cambios y cerrar la conexión.

```

numbers1Macetas = [1, 2, 3, 4, 5, 6, 7, 8, 10, 19, 20, 22, 23, 24, 26, 27, 29, 30, 32, 37,
                   39, 40, 42, 47, 49, 50, 52, 57, 59, 60, 62, 67, 69, 70, 72, 73, 74, 75, 76, 77, 79,
                   80, 89, 91, 92, 93, 96, 97, 98]

numbers2Macetas = [23, 24, 25, 26, 32, 37, 42, 47, 52, 55, 57, 62, 67, 73, 74, 75, 76]

for i in numbers1Macetas:
    insertarDatos(i, 0, " ")

for i in numbers2Macetas:
    insertarDatos2(i+100, 0, " ")

```

Se ingresan todas las posiciones donde puede haber una planta, como esto es para la primera vez que se ejecuta el programa y no hay plantas, entonces el resto de los datos se les coloca un dato sin valor para ocupar el espacio.

Con esto terminamos con la creación de tablas y se pasará a ver las funciones para modificar la base de datos.

1	import sqlite3
2	

Primero debemos importar sqlite3

```
3     def conectar():
4         con = sqlite3.connect("data.db")
5         cur = con.cursor()
6         return con, cur
7
8     def insertarDatos(plant_id, expected_humidity, type_plant):
9         con, cur = conectar()
10        sentencia = "INSERT INTO planta(plant_id, expected_humidity, type_plant) VALUES(?, ?, ?)"
11        datos = (plant_id, expected_humidity, type_plant)
12        cur.execute(sentencia, datos)
13        con.commit()
14        con.close()
15
16    def insertarDatos2(plant_id, expected_humidity, type_plant):
17        con, cur = conectar()
18        sentencia = "INSERT INTO planta2(plant_id, expected_humidity, type_plant) VALUES(?, ?, ?)"
19        datos = (plant_id, expected_humidity, type_plant)
20        cur.execute(sentencia, datos)
21        con.commit()
22        con.close()
```

Agregamos lo mismo que estaba en el txt, “conectar” será usado por las demás funciones, a diferencia de los “insertarDatos” que no serán utilizados, pero que pueden llegar a servir en caso de que haya que manipular manualmente las tablas.

```

24     def consultarDatos(plant_id):
25         con, cur = conectar()
26         m = ""
27         resultado = cur.execute("SELECT * FROM planta WHERE plant_id = ?" , (plant_id,))
28         for fila in resultado:
29             m = [fila[0],fila[1],fila[2]]
30         con.close()
31     return m
32
33     def consultarDatos2(plant_id):
34         con, cur = conectar()
35         m = ""
36         resultado = cur.execute("SELECT * FROM planta2 WHERE plant_id = ?" , (plant_id,))
37         for fila in resultado:
38             m = [fila[0],fila[1],fila[2]]
39         con.close()
40     return m
41
42     def consultarDatosM(piso):
43         con, cur = conectar()
44         m = ""
45         resultado = cur.execute("SELECT * FROM mapas WHERE piso = ?" , (piso,))
46         for fila in resultado:
47             m = [fila[1],
48                  fila[2],
49                  fila[3],
50                  fila[4],
51                  fila[5],
52                  fila[6],
53                  fila[7],
54                  fila[8],
55                  fila[9],
56                  fila[10]
57             ]
58         con.close()
59     return m
60

```

“consultarDatos” funciona de forma similar para las 3 tablas, se le da un dato, y devuelve los valores de la fila en la que esta ese dato. Recordar declarar m para evitar posibles problemas de retornar una variable que no existe.

```

61     def actualizarDatos(plant_id, expected_humidity, type_plant):
62         con, cur = conectar()
63         cur.execute("UPDATE planta SET expected_humidity = ?, type_plant = ? WHERE plant_id = ?", (expected_humidity, type_plant, plant_id))
64         con.commit()
65         con.close()
66         print("datos actualizados correctamente")
67         return True
68
69     def actualizarDatos2(plant_id, expected_humidity, type_plant):
70         con, cur = conectar()
71         cur.execute("UPDATE planta2 SET expected_humidity = ?, type_plant = ? WHERE plant_id = ?", (expected_humidity, type_plant, plant_id))
72         con.commit()
73         con.close()
74         print("datos actualizados correctamente")
75         return True
76
77     def actualizarDatosM(piso, mapa):
78         con, cur = conectar()
79         cur.execute("UPDATE mapas SET 1 = ?, 2 = ?, 3 = ?, 4 = ?, 5 = ?, 6 = ?, 7 = ?, 8 = ?, 9 = ?, 10 = ? WHERE piso = ?",
80                     (mapa[0], mapa[1], mapa[2], mapa[3], mapa[4], mapa[5], mapa[6], mapa[7], mapa[8], mapa[9], mapa[10], piso))
81         con.commit()
82         con.close()
83         print("datos del mapa actualizados correctamente")
84         return True

```

Para actualizar los datos de las plantas se les asignaran nuevos datos a la humedad esperada y al tipo de planta, a cual planta se le hace depende del ‘plant_id’ que se le asigne.

Para el caso del mapa se copia y pega en un piso determinado el ‘mapa’ que se le entrega.

robot

```
1 class Robot:  
2  
3     def __init__(self,x,y,piso,planta,key_entrando, destino_final,accion,quieto,id,tomar,orden):  
4         self.x=x #posicion x  
5         self.y=y #posicion y  
6         self.piso = piso  
7         self.planta = planta #la planta a la que se dirige  
8         self.key = key  
9         self.entrando = entrando #si es que entra a la base  
10        self.destino_final = destino_final #hacia donde se dirige  
11        self.accion = accion #si se esta moviendo  
12        self.quieto = quieto #hace que se quede quieto unos momentos al realizar un accion  
13        self.id = id  
14        self.tomar = tomar  
15        self.orden = orden  
16
```

robot solo contiene la clase “Robot”, que es el tipo de objeto que serán los robots que recorran el invernadero.

‘x’ e ‘y’ (int) son la posición del robot en la pantalla usando pixeles como unidad, ‘piso’ (int) es el piso en el que se encuentra el robot, ‘planta’ (int) es el número de posición en la matriz donde se encuentra la planta a la que se dirige, ‘key’ (String) es la dirección en la que se mueve (arriba, abajo, izquierda o derecha), ‘entrando’ (boolean) se usa dentro de la lógica del movimiento para saber si el robot esta entrando o saliendo de su estación,’destino_final’ (int) es la ubicación a la que va el robot, ‘accion’ (boolean) se usa para saber si el robot esta actualmente cumpliendo una orden, ‘quieto’ (int) se usa para retrasar visualmente el movimiento del robot que de otra forma se movería instantáneamente, ‘id’ (int) almacena el identificador de la imagen del robot, ‘tomar’ (boolean/String) sirve para saber en que parte de la orden va el robot, ‘orden’ (String) dice cual es la orden que está ejecutando el robot.

sprites

```
1      from PIL import Image,ImageTk
2
3
4
5
6  def cargarImagen(pixel,opcion):
7      if (opcion=="planta"):
8          img = (Image.open("./assets/planta.png"))
9      if (opcion == "grifo"):
10         img = (Image.open("./assets/grifo.png"))
11     if (opcion == "paredb"):
12         img = (Image.open("./assets/paredblanca.jpg"))
13     if (opcion == "walle"):
14         img = (Image.open("./assets/walle.png"))
15     if (opcion == "eva"):
16         img = (Image.open("./assets/eva.png"))
17     if (opcion == "transparente"):
18         img = (Image.open("./assets/transparente.jpg"))
19     if (opcion=="puerta"):
20         img = (Image.open("./assets/puerta.png"))
21     if (opcion == "arduino"):
22         img = (Image.open("./assets/ESP32.png"))
23     resized_image = img.resize((pixel,pixel), Image.Resampling.LANCZOS)
24     new_image = ImageTk.PhotoImage(resized_image)
25     return new_image
26
```

sprites se usa para añadir imágenes de forma más ordenada, además de evitar tener que escribir el proceso de ajustar el tamaño de una imagen cada vez que se quiere agregar alguna.

PIL es la librería que permite poner las imágenes.

En “cargarImagen” dependiendo de que ‘opcion’ reciba, retornara cierta imagen.

Advertencias:

-Para que funcione debe poseer la carpeta llamada assets, así como los archivos png dentro de esta.

-en caso de que ocurra el error: *Import “PIL” could not be resolved from source*, intente cambiar el entorno/environment. Este error nos ocurrió al utilizar Visual Studio Code, y se resolvió cambiando el entorno de uno de Microsoft store, al que no es de Microsoft Store.

funciones

funciones contiene varias funciones simples que utilizan pocos parámetros, pero que se usan regularmente

```
3 def traducir(x,pixel):
4     valor = int(x/pixel)
5     return valor
6
7 def matriz(x,y,pixel):
8     lugar = traducir(x,pixel) + (10*traducir(y,pixel))
9     return lugar
10
11 def black_matriz(planta,pixel):
12     posicion= (int(planta%10)*pixel),(int(planta/10)*pixel)
13     return posicion
14
```

“traducir” sirve para pasar un valor en pixeles a una coordenada de la matriz.

“matriz” utiliza “traducir” para dado 2 valores en pixeles, determinar en que segmento de la matriz se encuentran.

“black_matriz” es la inversa de “matriz”, dado una posición de la matriz, retorna la ubicación de ese segmento en pixeles.

```
15 def exactitud(x,y,pixel):
16     if (x%pixel==0 and y%pixel==0):
17         return True
18     else:
19         return False
20
```

“exactitud” sirve para controlar que el robot se encuentre exactamente en el inicio de un segmento de una matriz

Advertencia: el inicio de un segmento es la esquina superior izquierda, por lo que si se quiere llevar a vida real probablemente habría que cambiarlo a pixel/2, con el fin de que se use el centro del cuadro como referencia.

```
21 def reemplazar(linea, y, tomar): #linea es la posicion en el arreglo, y es el caracter a cambiar
22     aux2 = list(linea)
23     if(tomar== True):
24         aux2[y] = ' '
25     elif(tomar== "regado"):
26         aux2[y] = 'X'
27     linea = "".join(aux2)
28     return linea
29
```

“reemplazar” se usa para modificar el ‘mapa’, si es que ‘Tomar’ es True entonces elimina una planta, mientras que si es igual a regado entonces agrega una planta.

```
30     def buscar_id(plantas, planta):
31         id = 0
32         for i in range_(len(plantas)):
33             if (planta == plantas[i][0]):
34                 id = plantas[i][1]
35         return id
36
```

“buscar_id” se usa para saber cual es la id de la imagen de una planta, esto con el fin de eliminarla visualmente.

‘planta’ es una lista que su primer elemento es la posición en la matriz, mientras que el segundo es la id de su imagen, ‘plantas’ es una lista que contiene cada ‘planta’ del piso.

id=0 es para evitar el posible error de retornar un nulo.

```
37     def buscar_humedad_posicion(plantas, planta):
38         i = 0
39         for i in range_(len(plantas)):
40             if (planta == plantas[i][0]):
41                 return i
42         return 0
```

Archivo Principal: GUI

```
1 import random
2 import tkinter as tk
3 from tkinter import ttk
4 from tkinter import messagebox
5 from PIL import Image, ImageTk
6 import sprites
7 import mapa
8 import funciones
9 from robot import *
10 import time
11 from functools import partial
12 import save
```

Primero vamos a agregar las librerías que se van a usar más adelante. **random** será usada para generar la humedad de las plantas de forma aleatoria ya que no hay datos reales, **tkinter** será usado para la creación de la interfaz gráfica, **PIL** será usado para añadir imágenes, **time** será usado para que la humedad de las plantas disminuya con el tiempo, y **functools** sirve para añadirle funciones que reciben parámetros a los botones.

```
14 # configuracion de la ventana
15 ventana = tk.Tk()
16 ventana.title("Invernadero")
17 width = ventana.winfo_screenwidth()
18 height = ventana.winfo_screenheight() - 80
19 ventana.geometry("%dx%d" % (width, height))
20 height = int(height / 10)
21 pixel = height
22 while (pixel % 20 != 0):
23     pixel -= 1
24 height = pixel * 10
25 ventana.resizable(False, False)
```

Lo siguiente es configurar la ventana, utilizando los valores de ancho y alto de la pantalla, además de definir el tamaño de 'pixel'.

```

26     # aqui se crean los subventanas
27
28     tabsInvernadero = ttk.Notebook(ventana)
29     tab0 = ttk.Frame(tabsInvernadero)
30     tab1 = ttk.Frame(tabsInvernadero)
31     tab2 = ttk.Frame(tabsInvernadero)
32     tabsInvernadero.add(tab0, text='Invernadero')
33     tabsInvernadero.add(tab1, text='Tablas')
34     tabsInvernadero.add(tab2, text='¿Como funciona?')
35     tabsInvernadero.pack(side=tk.BOTTOM, fill="both", expand=True)
36     frameizquierda = tk.Frame(tab0, relief="sunken")
37     frameinformacion = tk.Frame(frameizquierda)
38     frameizquierda.pack(side=tk.LEFT, fill='y')
39     framePlantaSelect = tk.Frame(frameinformacion)
40     framehumedadplanta = tk.Frame(frameinformacion)
41     framehumedadplanta.pack(side=tk.TOP)
42     frameMovimiento = tk.Frame(frameinformacion)
43     lblplantaselec = tk.Label(framePlantaSelect, text='Planta seleccionada : ', fg="black", font=("Verdana", 10))
44     frameComofunciona = tk.Frame(tab2)
45
46     imagen = tk.PhotoImage(file="./assets/Instrucciones.png")
47     ComocHacelbl = tk.Label(frameComofunciona, image=imagen).pack(fill=tk.BOTH)
48     frameComofunciona.pack(fill=tk.BOTH)
49
50     lblplantaselec.pack(side=tk.LEFT)
51
52     cb1 = ttk.Combobox(frameMovimiento,
53                         values=("Cebolla", "Ajo", "Espinaca", "Alcachofa", "Zanahoria", "Albahaca", "Calabaza", "Remolacha"),
54                         width=8,
55                         state="disabled")
56
57     frameTabs = ttk.Frame(tab0)
58     tab_control = ttk.Notebook(frameTabs)
59     tab00 = ttk.Frame(tab_control)
60     tab01 = ttk.Frame(tab_control)
61     tab02 = ttk.Frame(tab_control)
62     tab03 = ttk.Frame(tab_control)
63     tab_control.add(tab00, text='Exterior')
64     tab_control.add(tab01, text='Piso 1')
65     tab_control.add(tab02, text='Piso 2')
66     tab_control.add(tab03, text='Piso 3')
67     tab_control.pack(side=tk.LEFT)

```

Se crean frames (dividir en zonas) para acomodar de mejor forma los elementos dentro de la interfaz, así como los tabs (pestañas) para cada piso. 'cb1' es un combobox que contiene los posibles tipos de plantas que se puede ingresar al invernadero.

69	inicio = time.time()
----	----------------------

Se empieza a registrar el tiempo

```

79     def callback(event):
80         botonCancelar.config(state="normal")
81         RegarPlanta.config(state="normal")
82         MedirHumedad.config(state="normal")
83         if tab_control.tab("current", "text") == "Exterior":
84             cb1.config(state="readonly")
85         else:
86             cb1.config(state="disabled")
87         global inicio
88         tiempo = time.time() - inicio
89         for i in range(len(stats0)): # reduccion de la humedad de la tierra con el paso del tiempo
90             if (stats0[i] >= 0):
91                 stats0[i] -= 0.005 * (tiempo % 5)
92             if (stats0[i] < 0):
93                 stats0[i] = 0
94
95         for i in range(len(stats)): # reduccion de la humedad de la tierra con el paso del tiempo
96             if (stats[i] >= 0):
97                 stats[i] -= 0.005 * (tiempo % 5)
98             if (stats[i] < 0):
99                 stats[i] = 0
100
101        for i in range(len(stats2)): # reduccion de la humedad de la tierra con el paso del tiempo
102            if (stats2[i] >= 0):
103                stats2[i] -= 0.005 * (tiempo % 5)
104            if (stats2[i] < 0):
105                stats2[i] = 0
106
107        botonBlanco.config(state="normal")
108        inicio += tiempo
109        mouse = funciones.matriz(event.x, event.y, pixel)

```

Se definen “key” y “callback” que son eventos cuando se hace presiona el click del mouse en la zona donde se muestra un piso, dentro de “callback” ira la activación de ciertos botones, así como la reducción de nivel de humedad de las plantas (definida de forma arbitraria).

```

111     # al hacer click en el exterior
112     if (mouse in numbers0 and tab_control.tab("current", "text") == "Exterior"):
113
114         monitor2.config(text=(mouse + 200))
115         monitor2.config(fg="black", font=("Verdana", 14))
116         monitor3.config(text=tipo0[mouse], fg="black", font=("Verdana", 10))
117         Salida1.config(text=mouse + 200, fg="black", font=("Verdana", 14))
118         EliminarPlanta.config(state="normal")
119
120     else:
121         EliminarPlanta.config(state="disabled")
122     if (mouse in numbers0Macetas and mouse not in numbers0 and tab_control.tab("current", "text") == "Exterior"):
123         AgregarPlanta.config(state="normal")
124     else:
125         AgregarPlanta.config(state="disabled")
126     # al hacer click en el primer piso
127     if (mouse in numbers1 and tab_control.tab("current", "text") == "Piso 1"):
128         monitor2.config(text=mouse, fg="black", font=("Verdana", 10))
129         monitorh1.config(text=info[mouse], fg="black", font=("Verdana", 10))
130         monitorh2.config(text=mouse, fg="black", font=("Verdana", 10))
131         Salida1.config(text=mouse, fg="black", font=("Verdana", 10))
132         monitor3.config(text=tipo[mouse], fg="black", font=("Verdana", 10))
133
134     # al hacer click en el segundo piso
135     if (mouse in numbers2 and tab_control.tab("current", "text") == "Piso 2"):
136         monitor2.config(text=(mouse + 100), fg="black", font=("Verdana", 10))
137         monitorh1.config(text=(info2[mouse]), fg="black", font=("Verdana", 10))
138         monitorh2.config(text=mouse + 100, fg="black", font=("Verdana", 10))
139         Salida1.config(text=mouse + 100, fg="black", font=("Verdana", 10))
140         monitor3.config(text=tipo[mouse], fg="black", font=("Verdana", 10))

```

Las variables “numbers” se refieren a los números de las plantas en un piso, el proceso por el que se les ingresa información está más adelante (Piso 0 se refiere al exterior). Como anotación, cuando se habla de los números de planta las del primero piso van del 0 al 99, las del segundo del 100 al 199, y del exterior del 200 al 299.

Se verifica que el click haya sido en una planta dentro de un tab en específico. Y muestra en la interfaz cual fue esa planta.

```

142         # boton MoverPlanta
143         if tab_control.tab("current", "text") == "Exterior":
144             if (mouse in numbers0Macetas) and (mouse not in numbers0):
145                 Salida2.config(text=mouse + 200)
146             if tab_control.tab("current", "text") == "Piso 1":
147                 if (mouse in numbers1Macetas) and (mouse not in numbers1):
148                     Salida2.config(text=mouse)
149             if tab_control.tab("current", "text") == "Piso 2":
150                 if (mouse in numbers2Macetas) and (mouse not in numbers2):
151                     Salida2.config(text=mouse + 100)
152             if (Salida1.cget("text") != "" and Salida2.cget("text") != ""):
153                 MoverPlanta.config(state="normal")

```

Finalmente, si se presiona un lugar donde no hay planta, pero es posible tener una, se mostrará cual es esa posición, pero en un monitor distinto. Con esto termina el contenido dentro de “callback”.

```

156 monitorh0 = tk.Label(framehumedadplanta, text="Humedad de la planta ", fg="black", font=("Verdana", 10))
157 monitorh2 = tk.Label(framehumedadplanta, text="?", fg="black", font=("Verdana", 10))
158 monitorh1 = tk.Label(frameinformacion, text="?", fg="black", font=("Verdana", 10))
159
160 monitorh0.pack(side=tk.LEFT)
161 monitorh2.pack(side=tk.LEFT)
162 monitorh1.pack(side=tk.TOP)
163 monitor1 = tk.Label(frameizquierda, text="Ninguna orden activa", fg="black", font=("Verdana", 10))
164 monitor2 = tk.Label(framePlantaSelect, text="?", fg="black", font=("Verdana", 10))
165
166 framesalidas = ttk.Frame(frameMovimiento)
167 framesalida1 = ttk.Frame(framesalidas)
168 framesalida2 = ttk.Frame(framesalidas)
169
170 frameTipoPlanta = ttk.Frame(framesalidas)
171 monitor3 = tk.Label(frameTipoPlanta, fg="black", font=("Verdana", 10))
172 lblTipoPlanta = tk.Label(frameTipoPlanta, text=" La planta seleccionada es", fg="black", font=("Verdana", 10))
173 framesalida1.pack(side=tk.TOP)
174 framesalida2.pack(side=tk.TOP)
175 lblsalida1 = tk.Label(framesalida1, text="Planta: ", fg="black", font=("Verdana", 10))
176 lblsalida2 = tk.Label(framesalida2, text="Maceta: ", fg="black", font=("Verdana", 10))
177 Salida1 = tk.Label(framesalida1, text="", font=("Verdana", 10))
178 Salida2 = tk.Label(framesalida2, text="", font=("Verdana", 10))

```

Se crean los monitores que se usaron anteriormente, así como algunas líneas de texto que se mostrarán en pantalla.

```

180 # variables:
181
182 numbers0Macetas = [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]
183
184 numbers1Macetas = [1, 2, 3, 4, 5, 6, 7, 8, 10, 19, 20, 22, 23, 24, 26, 27, 29, 30, 32, 37,
185                 39, 40, 42, 47, 49, 50, 52, 57, 59, 60, 62, 67, 69, 70, 72, 73, 74, 75, 76, 77, 79,
186                 80, 89, 91, 92, 93, 96, 97, 98]
187
188 numbers2Macetas = [23, 24, 25, 26, 32, 37, 42, 47, 52, 55, 57, 62, 67, 73, 74, 75, 76]
189

```

Las variables 'numbers_macetas' son las posiciones de la matriz donde se puede poner una planta, la cual como se puede ver varían según el piso.

```

190     lblsalida1.pack(side=tk.LEFT)
191     Salida1.pack(side=tk.LEFT)
192     lblsalida2.pack(side=tk.LEFT)
193     Salida2.pack(side=tk.LEFT)
194     lblTipoPlanta.pack(side=tk.LEFT)
195     monitor3.pack(side=tk.LEFT)

```

Se selecciona el lugar donde aparecen algunos elementos visuales.

```
197     # piso 0 en tab00
198     canvas0 = tk.Canvas(tab00, width=height, height=height)
199     canvas0.bind("<Key>", key)
200     canvas0.bind("<Button-1>", callback)
201     canvas0.pack(side=tk.LEFT)
202
203     # piso1 en tab01
204
205     canvas1 = tk.Canvas(tab01, width=height, height=height)
206     canvas1.bind("<Key>", key)
207     canvas1.bind("<Button-1>", callback)
208     canvas1.pack(side=tk.LEFT)
209
210     # piso2 en tab02
211     canvas2 = tk.Canvas(tab02, width=height, height=height)
212     canvas2.bind("<Key>", key)
213     canvas2.bind("<Button-1>", callback)
214     canvas2.pack(side=tk.LEFT)
215     # piso3 en tab03
216     canvas3 = tk.Canvas(tab03, width=height, height=height)
217     canvas3.bind("<Key>", key)
218     canvas3.bind("<Button-1>", callback)
219     canvas3.pack(side=tk.LEFT)
```

Se crea un canvas para cada piso, que es donde iran las imágenes como la matriz o las plantas.

```

221 # colocar la matriz de fondo y microcontroller
222 img1 = (Image.open("./assets/matriz0.png"))
223 img2 = (Image.open("./assets/matriz1.png"))
224 img3 = (Image.open("./assets/matriz2.png"))
225 resized_image1 = img1.resize((height, height), Image.Resampling.LANCZOS)
226 resized_image2 = img2.resize((height, height), Image.Resampling.LANCZOS)
227 resized_image3 = img3.resize((height, height), Image.Resampling.LANCZOS)
228 matriz0 = ImageTk.PhotoImage(resized_image1)
229 matriz1 = ImageTk.PhotoImage(resized_image2)
230 matriz2 = ImageTk.PhotoImage(resized_image3)
231 canvas0.create_image(0, 0, anchor="nw", image=matriz0)
232 canvas1.create_image(0, 0, anchor="nw", image=matriz1)
233 canvas2.create_image(0, 0, anchor="nw", image=matriz2)
234
235 imgA = ImageTk.PhotoImage(Image.open("./assets/ESP32.png"))
236 panel4 = tk.Label(canvas3, image=imgA)
237 panel4.pack(side=tk.TOP)
238 panel4.config(width=420, height=220)

```

Se colocan las imágenes de fondo para cada piso (la matriz), así como una imagen de un ESP32, la cual irá de fondo en el tercer piso.

```

240 # lista que contiene los espacios de la matriz que son plantas
241
242 numbers0 = []
243 numbers1 = []
244 numbers2 = []
245 # dibujar el mapa
246 mapa0 = save.consultarDatosM(0)
247 x = 0
248 y = 0
249 c = 0
250 new_image0 = []
251 planta = tuple()
252 plantas0 = []
253
254 imagen_planta = sprites.cargarImagen(pixel, "planta")

```

Se definen algunas variables que serán usadas para llenar el canvas con imágenes, así como para registrar las plantas, esto usando la información de la base de datos, primero se hará con el exterior.

```

256     for fila in mapa0:
257         for muro in fila:
258             if muro == "X":
259                 new_image0.append(sprites.cargarImagen(pixel, "planta"))
260                 id = canvas0.create_image(x, y, anchor="nw", image=new_image0[c])
261                 planta = funciones.matriz(x, y, pixel), id
262                 plantas0.append(planta)
263                 numbers0.append(planta[0])
264             elif muro == "Y":
265                 new_image0.append(sprites.cargarImagen(pixel, "grifo"))
266                 canvas0.create_image(x, y, anchor="nw", image=new_image0[c])
267             elif muro == "Z":
268                 new_image0.append(sprites.cargarImagen(pixel, "paredb"))
269                 canvas0.create_image(x, y, anchor="nw", image=new_image0[c])
270             elif muro == "P":
271                 new_image0.append(sprites.cargarImagen(pixel, "puerta"))
272                 canvas0.create_image(x, y, anchor="nw", image=new_image0[c])
273             else:
274                 new_image0.append("")
275                 x += pixel
276                 c += 1
277
278             x = 0
279             y += pixel

```

Se lee ‘mapa’ y se rellena con imágenes según el carácter que se este revisando. Esto lo hace llamando a la función “cargarImagen” en **sprites**, diciéndole que imagen tiene que agregar. ‘x’ e ‘y’ se usan para recorrer visualmente la matriz.

A cada imagen se le asigna una id, para así poder modificarla más adelante.

En el caso de leer una X, se registrara la planta (su número e id de imagen) y se ingresara en la lista ‘plantas’, además se ingresará a ‘numbers’, pero en esta caso solo importa su numero.

```
288     for fila in mapa1:
289         for muro in fila:
290             if muro == "X":
291                 new_image1.append(sprites.cargarImagen(pixel, "planta"))
292                 id = canvas1.create_image(x, y, anchor="nw", image=new_image1[c])
293                 planta = funciones.matriz(x, y, pixel), id
294                 plantas.append(planta)
295                 numbers1.append(planta[0])
296             elif muro == "Y":
297                 new_image1.append(sprites.cargarImagen(pixel, "grifo"))
298                 canvas1.create_image(x, y, anchor="nw", image=new_image1[c])
299             elif muro == "Z":
300                 new_image1.append(sprites.cargarImagen(pixel, "paredb"))
301                 canvas1.create_image(x, y, anchor="nw", image=new_image1[c])
302             elif muro == "P":
303                 new_image1.append(sprites.cargarImagen(pixel, "puerta"))
304                 aux = canvas1.create_image(x, y, anchor="nw", image=new_image1[c])
305             else:
306                 new_image1.append(sprites.cargarImagen(1, "transparente"))
307                 canvas1.create_image(0, 0, anchor="nw", image=new_image1[c])
308                 x += pixel
309                 c += 1
310
311             x = 0
312             y += pixel
```

Se hace lo mismo en el piso 1

```

314     mapa2 = save.consultarDatosM(2)
315     x = 0
316     y = 0
317     c = 0
318     plantas2 = []
319     new_image2 = []
320     for fila in mapa2:
321         for muro in fila:
322             if muro == "X":
323                 new_image2.append(sprites.cargarImagen(pixel, "planta"))
324                 id = canvas2.create_image(x, y, anchor="nw", image=new_image2[c])
325                 planta = funciones.matriz(x, y, pixel), id
326                 plantas2.append(planta)
327                 numbers2.append(planta[0])
328             elif muro == "Y":
329                 new_image2.append(sprites.cargarImagen(pixel, "grifo"))
330                 canvas2.create_image(x, y, anchor="nw", image=new_image2[c])
331             elif muro == "Z":
332                 new_image2.append(sprites.cargarImagen(pixel, "paredb"))
333                 canvas2.create_image(x, y, anchor="nw", image=new_image2[c])
334             else:
335                 new_image2.append("")
336             x += pixel
337             c += 1
338
339             x = 0
340             y += pixel
341

```

Se hace lo mismo en el piso 2

```

342     tiposdeplantas = [("Cebolla", 4.52),
343                         ("Ajo", 2.34),
344                         ("Espinaca", 3.6),
345                         ("Alcachofa", 7.3),
346                         ("Zanahoria", 2.3),
347                         ("Albahaca", 6.7),
348                         ("Calabaza", 3.7),
349                         ("Remolacha", 12.3)]

```

Se define cuales son los tipos de planta que hay, además se les asigna un valor de humedad optimo (se le puso un numero arbitrario)

```

351     stats0 = [] # asignarle humedades a las plantas
352     info0 = []
353     tipo0 = []
354     humedadesperada0 = []
355     c = 0
356     for i in range(100):
357         if (c < len(plantas)):
358             if i in numbers0:
359                 stats0.append(round(random.uniform(0, 2), 3))
360                 info0.append("?")
361                 tipo_rand = (random.choice(tiposdeplantas))
362                 tipo0.append(tipo_rand[0])
363                 humedadesperada0.append(tipo_rand[1])
364                 c += 1
365             else:
366                 stats0.append(-1)
367                 info0.append(-1)
368                 tipo0.append("")
369                 humedadesperada0.append("")
370         else:
371             stats0.append(-1)
372             info0.append(-1)
373             tipo0.append("")
374             humedadesperada0.append("")

```

'stats' es la humedad de la planta, 'info' es la humedad visible de la planta, la cual será un signo de interrogación en caso de que no se sepa, 'tipo' es el tipo de la planta. Estos datos se asocian a cada planta por medio del número de planta, el cual será usado como índice de estas listas.

En caso de que no haya planta en la posición que se está revisando se le asignara un -1 como humedad.

Advertencia: el asignar humedades no sirve, pues no hay ninguna planta presente al iniciar el programa, y en una situación real donde ya haya plantas estas tendrían humedades desconocida, esto es solo para pruebas. Aun así, es importante crear la variable.

```
376     stats = [] # asignarle humedades a las plantas
377     info = []
378     tipo = []
379     humedadesesperada = []
380     c = 0
381     for i in range(99):
382         if (c < len(plantas)):
383             if i in numbers1:
384                 temp = save.consultarDatos(i)
385                 stats.append(round(random.uniform(0, 2), 3))
386                 info.append("?")
387                 tipo.append(temp[2])
388                 humedadesesperada.append(temp[1])
389                 c += 1
390             else:
391                 stats.append(-1)
392                 info.append(-1)
393                 tipo.append("")
394                 humedadesesperada.append("")
395         else:
396             stats.append(-1)
397             info.append(-1)
398             tipo.append("")
399             humedadesesperada.append("")
```

Se hace lo mismo con las plantas del piso 1, solo que en este caso si es posible que haya plantas, por lo que se obtiene el tipo de planta y la humedad esperada de la base de datos. Para el caso del número de planta se obtiene de 'mapa', el cual también es obtenido de la base de datos.

```
401     stats2 = [] # asignarle humedades a las plantas
402     info2 = []
403     tipo2 = []
404     humedadesperada2 = []
405     c = 0
406     for i in range(77):
407         if (c < len(plantas2)):
408             if i in numbers2:
409                 temp = save.consultarDatos2(i + 100)
410                 stats2.append(round(random.uniform(0, 2), 3))
411                 info2.append("?")
412                 tipo2.append(temp[2])
413                 humedadesperada2.append(temp[1])
414                 c += 1
415             else:
416                 stats2.append(-1)
417                 info2.append(-1)
418                 tipo2.append("")
419                 humedadesperada2.append("")
420         else:
421             stats2.append(-1)
422             info2.append(-1)
423             tipo2.append("")
424             humedadesperada2.append("")
```

Se hace lo mismo con las plantas del piso 2, también se obtienen los datos de la base de datos.

```
426 fotorobot = []
427
428 # walle
429
430 # (x,y,piso,planta,key,entrando, destino_final,accion,quieto,id,tomar,orden)
431
432 walle = Robot(3 * pixel, 3 * pixel, 1, 1, "right", False, 0, False, 20, 0, False, 'null')
433 fotorobot.append(sprites.cargarImagen(pixel, "walle"))
434 walle.id = canvas1.create_image(walle.x, walle.y, anchor='nw', image=fotorobot[0])
435
436 # eva
437
438 eva = Robot(6 * pixel, 3 * pixel, 1, 1, "left", False, 0, False, 20, 0, False, 'null')
439 fotorobot.append(sprites.cargarImagen(pixel, "eva"))
440 eva.id = canvas1.create_image(eva.x, eva.y, anchor='nw', image=fotorobot[1])
```

Se crean los robots y sus imágenes.

'fotorobot' recibe el contenido de los archivos png para representar los robots, además de guarda la id de esta imagen.

Los valores 'x' 'y' son las coordenadas en las que los robots aparecerán en la matriz, también se pone la dirección a la que va a empezar su movimiento, el resto de valores son los que tienen cuando están sin ejecutar ordenes o para rellenar hasta que se le asigne un valor real.

```
442 respaldo_planta = []
443 pm = ""
444 abortar = False
```

Se definen algunas variables. 'respaldo_planta' almacena la información de una planta que se esté moviendo, 'pm' se usa para guardar el número de posición donde se va a dejar una planta, 'abortar' es una variable de control que se usará más adelante para el botón de abortar.

Ahora viene el código del movimiento de los robots, sin embargo, ya que estos usan parámetros de botones que están definidos más abajo se explicara en un orden distinto al que dice el código.

```
1699 # botones
1700 probarR = tk.Button(framecalefactor, text="probar movimiento robots",
1701                         command=partial(PruebaMov, mapa0, mapa1, mapa2, walle, eva, pm))
1702 probarR.pack(side=tk.BOTTOM)
1703 moveUpButton = tk.Button(framelisybotones, text="subir", command=Subir)
1704 movedownButton = tk.Button(framelisybotones, text="Bajar", command=Bajar)
1705 BorrarInsW = tk.Button(framelisybotones, text="Borrar Instrucción", command=borrarelemento)
1706 botonBlanco = tk.Button(frameEva, text="medir humedad de todo",
1707                         command=partial(Medir_all, mapa0, mapa1, mapa2, walle, eva, pm))
1708 on_button1 = tk.Button(framebt�s, image=on, bd=0, command=switch1)
1709 on_button2 = tk.Button(framebt�s, image=on, bd=0, command=switch2)
1710 on_button3 = tk.Button(framebt�s, image=on, bd=0, command=switch3)
1711 on_button4 = tk.Button(framebt�s, image=on, bd=0, command=switch4)
1712 open_button = tk.Button(frameinvernadero, text="Abrir", command=open)
1713 mid_button = tk.Button(frameinvernadero, text="Semi-Cerrar", command=mid)
1714 close_button = tk.Button(frameinvernadero, text="Cerrar", command=closed)
1715 RegarPlanta = tk.Button(frameWalle, text="Regar Planta",
1716                         command=partial(Regar_planta, mapa0, mapa1, mapa2, walle, eva, pm), state="disabled")
1717 RegarTodo = tk.Button(frameWalle, text="Regar Todo", command=partial(Regar_all, mapa0, mapa1, mapa2, walle, eva, pm),
1718                         state="disabled")
1719 MedirHumedad = tk.Button(frameEva, text="Medir Humedad",
1720                         command=partial(Medir_humedad, mapa0, mapa1, mapa2, walle, eva, pm), state="disabled")
1721 MoverPlanta = tk.Button(frameminiomover, text="Mover Planta",
1722                         command=partial(Mover_planta, mapa0, mapa1, mapa2, walle, eva, pm), state="disabled")
1723 botonCancelar = tk.Button(frameminiomover, text="Cancelar Movimiento", command=clear, state="disabled")
1724 AgregarPlanta = tk.Button(frameminiagregar, text="Agregar planta", command=Agregar_planta, state="disabled")
1725 EliminarPlanta = tk.Button(frameminiagregar, text="Eliminar planta", command=Eliminar_planta, state="disabled")
1726 botonABORTAR = tk.Button(frameizquierda, text="ABORTAR", state="disabled", command=partial(abort), width=10, height=1,
1727                         foreground="#000000", background="#ff0000",
1728                         font=("Times", 12))
```

Aquí se definen una serie de botones:

‘probarR’ le dice a Walle que vaya a un lugar en específico y vuelva, una prueba de movimiento.

‘moveUpButton’ y ‘movedownButton’ son para cambiar el orden dentro de una listbox que contiene todas las instrucciones. ‘BorrarInsW’ elimina una instrucción de esta listbox.

‘botonBlanco’ activa la orden de medir la humedad de todas las plantas.

Los ‘on_button’ son botones decorativos que se encarga de acciones como comprobar que corre agua, encender el calefactor, entre otros, estos se encuentran en el tab del piso 3. ‘(open/mid/close)_button’ son para regular el grado deertura de la sombrilla.

‘RegarPlanta’ activa la orden de regar una planta.

‘RegarTodo’ activa la orden de regar todas las plantas bajo el nivel de humedad óptimo.

‘MedirHumedad’ activa la orden de medir la humedad de una planta.

‘MoverPlanta’ activa la orden de mover una planta a una ubicación disponible.

‘botonCancelar’ Limpia de la pantalla las posiciones que se han seleccionado. Llama la función “clear()”, la cual es usada también por otros botones.

‘AregarPlanta’ Agrega una planta al piso exterior donde se haya seleccionado. ‘EliminarPlanta’ hace desaparecer una planta del piso exterior.

‘botonABORTAR’ hace que el robot termine su orden actual, así como eliminar todas las ordenes que seguían en la cola.

Algunos dicen partial en el command, esto le entrega parámetros a una función. Las cuales están mas arriba en el código.

```
1476     def Regar_planta(mapa0, mapa1, mapa2, walle, eva, pm):
1477         if (monitor2.cget("text") != ""):
1478             aux = int(monitor2.cget("text"))
1479             if ((aux < 100 and info[aux] != "?") or ((aux > 100 and aux < 200) and info2[aux - 100] != "?")):
1480                 texto = "Regar "
1481                 texto = texto + str(monitor2.cget("text"))
1482                 listbox.insert(tk.END, texto)
1483                 clear()
1484                 if not listbox.get(1):
1485                     Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1486                 else:
1487                     if (aux > 200):
1488                         messagebox.showinfo("Message", "Operacion no valida fuera del invernadero")
1489                     else:
1490                         messagebox.showinfo("Message",
1491                                         "Humedad de la planta desconocida, se recomienda medir la humedad primero")
1492
```

En “Regar_planta” primero se verifica que ya se haya medido la humedad de la planta seleccionada, entonces manda al listbox la palabra Regar + el número de posición que consigue de ‘monitor2’, si es que no hay ninguna orden activa entonces empieza inmediatamente.

```
1494     def Mover_planta(mapa0, mapa1, mapa2, walle, eva, pm):
1495         global posrecogida
1496         global posentrega
1497         posrecogida = str(Salida1.cget("text"))
1498         posentrega = str(Salida2.cget("text"))
1499         mover = "Mover "
1500         a = " a "
1501         texto = (mover + posrecogida + a + posentrega)
1502         listbox.insert(tk.END, texto)
1503         clear()
1504         if not listbox.get(1):
1505             Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1506
```

Con “Mover_planta” obtiene los números de los 2 monitores, ya que se selecciona la posición de la planta así como la del lugar donde hay que dejarla, el resto funciona de forma similar al botón anterior.

```

1508     def Medir_humedad(mapa0, mapa1, mapa2, walle, eva, pm):
1509         if (monitor2.cget("text") != ""):
1510             aux = int(monitor2.cget("text"))
1511             if (aux < 200):
1512                 texto = "Medir Humedad "
1513                 texto = texto + str(monitor2.cget("text"))
1514                 listbox.insert(tk.END, texto)
1515                 clear()
1516                 if not listbox.get(1):
1517                     Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1518                 else:
1519                     messagebox.showinfo("Message", "Operacion no valida fuera del invernadero")
1520

```

“Medir_humedad” funciona igual que “Regar_planta”, solo que no pide que se sepa la humedad.

```

1522     advertencia = True
1523
1524
1525     def Medir_all(mapa0, mapa1, mapa2, walle, eva, pm):
1526         clear()
1527         global advertencia
1528         if advertencia == False:
1529             texto = "Medir humedad de todo"
1530             listbox.insert(tk.END, texto)
1531             if not listbox.get(1):
1532                 Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1533             if advertencia == True:
1534                 messagebox.showinfo("Message",
1535                                     "Esta orden tardara un tiempo elevado, dependiendo de la cantidad
1536                                     advertencia = False
1537

```

“Medir_all” solo entrega una línea de texto al listbox, pero si es la primera vez que se presiona el botón entonces entrega la siguiente advertencia: *Esta orden tardara un tiempo elevado, dependiendo de la cantidad de plantas puede ser de hasta 9 minutos, en caso de querer continuar de todas formas vuelva a seleccionar esta opcion.*

```

1539     advertencia2 = True
1540
1541
1542     def Regar_all(mapa0, mapa1, mapa2, walle, eva, pm):
1543         clear()
1544         global advertencia2
1545         if advertencia2 == False:
1546             texto = "Regar plantas con humedad baja"
1547             listbox.insert(tk.END, texto)
1548             if not listbox.get(1):
1549                 Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1550         if advertencia2 == True:
1551             messagebox.showinfo("Message",
1552                                 "Esta orden puede tardar, en caso de querer continuar de todas formas vuelva a seleccionar esta opcion")
1553
1554

```

“Regar_all” funciona de la misma forma que el botón anterior, dando también una advertencia similar: *Esta orden puede tardar, en caso de querer continuar de todas formas vuelva a seleccionar esta opcion.*

```

1174     # Define los parametros de movimiento
1175     def Orden(mapa0, mapa1, mapa2, walle, eva, pm):
1176         botonABORTAR.config(state="normal")
1177         Salida1.config(text="")
1178         Salida2.config(text="")
1179         text = listbox.get(0)
1180         plantainicio = 0
1181         control = False

```

Más arriba en el código está la función “Orden”, la cual se encarga de preparar lo necesario para el movimiento.

```

1182         if (text[0:5] == "Regar" and text != "Regar plantas con humedad baja"):
1183             walle.orden = "regar"
1184             robo = walle
1185             robo.planta = int(text[6:9])

```

Ya que las órdenes son textos predefinidos, reconoce ciertos caracteres en esto para determinar la orden y posición.

‘robo’ es un objeto de tipo “Robot” que recibe parámetros de eva o walle según corresponda, es la variable que se usa en la función de movimiento.

```

1186     if text == "Regar plantas con humedad baja":
1187         for i in numbers1:
1188             if (info[i] != "?"):
1189                 aux = str(info[i])
1190                 aux = aux[:4]
1191                 aux = float(aux)
1192                 if (aux < humedadesperada[i]):
1193                     control = True
1194                     walle.orden = "regar"
1195                     robo = walle
1196                     robo.planta = i
1197                     monitor1.config(text="regando planta " + str(robo.planta))
1198                     robo.accion = True
1199                     robo.tomar = True
1200                     robo.entrando = False
1201
1202             while (walle.accion == True):
1203                 walle = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1204                 robo = walle
1205                 walle.orden = 'null'
1206
1207             for i in numbers2:
1208                 if (info2[i] != "?"):
1209                     aux = str(info2[i])
1210                     aux = aux[:4]
1211                     aux = float(aux)
1212                     if (aux < humedadesperada2[i]):
1213                         control = True
1214                         walle.orden = "regar"
1215                         robo = walle
1216                         robo.planta = i + 100
1217                         monitor1.config(text="regando planta " + str(robo.planta))
1218                         robo.accion = True
1219                         robo.tomar = True
1220                         robo.entrando = False
1221                         while (walle.accion == True):
1222                             walle = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1223                             robo = walle
1224                             walle.orden = 'null'
1225             if (control == False):
1226                 robo = walle
1227                 messagebox.showinfo("Message", "ninguna planta con humedad conocida necesita r")
1228                 robo.orden = "null"
1229             control = False
1230         elif (text[0:5] == "Mover"):

```

Para regar todo lo que hace es una iteración en la que cada planta cuyo valor de humedad conocido sea menor al ideal se le hará la acción de regarla, asignándole a walle los parámetros para el inicio de la operación, además de la orden de regar. Si es que no es necesario regar ninguna planta entonces 'control' nunca se volverá True, por lo que mostrará un mensaje. Hace una iteración por piso.

```

1230     elif (text[0:5] == "Mover"):
1231         walle.orden = "mover"
1232         robo = walle
1233         plantainicio = ""
1234         control = True
1235         for i in range(5, len(text)):
1236             if text[i] in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]:
1237                 if (control == True):
1238                     plantainicio = plantainicio + text[i]
1239                 else:
1240                     pm = str(pm) + text[i]
1241                     if (text[i] == "a"):
1242                         control = False
1243                     plantainicio = int(plantainicio)
1244                     pm = int(pm)
1245                     robo.planta = plantainicio

```

Para mover una planta, recorre la cadena de texto para reconocer la posición de la planta, la cual será asignada a 'robo.planta', entonces 'control' se vuelve False y se reconoce la posición donde hay que dejar la planta, la cual se le asigna a 'pm' para su uso posterior.

```

1248     elif (text[0:5] == "Medir" and text != "Medir humedad de todo"):
1249         eva.orden = "medir"
1250         robo = eva
1251         robo.planta = int(text[14:17])
1252

```

Medir es similar a regar, solo que robo toma los parámetros de eva.

```

1253     if text == "Medir humedad de todo":
1254         if (plantas or plantas2):
1255             for i in numbers1:
1256                 eva.orden = "medir"
1257                 robo = eva
1258                 robo.planta = i
1259                 monitor1.config(text="midiendo humedad de " + str(robo.planta))
1260                 robo.accion = True
1261                 robo.tomar = True
1262                 robo.entrando = False
1263
1264             while (eva.accion == True):
1265                 eva = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1266                 robo = eva
1267                 eva.orden = 'null'
1268
1269             for i in numbers2:
1270                 eva.orden = "medir"
1271                 robo = eva
1272                 robo.planta = i + 100
1273                 monitor1.config(text="midiendo humedad de " + str(robo.planta))
1274                 robo.accion = True
1275                 robo.tomar = True
1276                 robo.entrando = False
1277             while (eva.accion == True):
1278                 eva = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1279                 robo = eva
1280                 eva.orden = 'null'
1281         else:
1282             messagebox.showinfo("Message", "No hay plantas en el invernadero")
1283             listbox.delete(0)
1284             robo = eva
1285             robo.orden = "null"

```

Medir todo funciona forma similar que regar todo, con una iteración que recorre cada piso y dando ordenes individuales de medir la humedad de esa planta.

En caso de que no hayan plantas se mostrara un mensaje de error, el darle a 'robo' valores de eva y orden null es para evitar errores por usar los datos de 'robo' sin que este tenga esos valores.

```
1287     # evitar que se haga una accion con una planta que ya no esta
1288     plantaerror = False
1289     if (robo.orden == "mover"):
1290         if (robo.planta < 100 and (robo.planta not in numbers1 or pm in numbers1)):
1291             plantaerror = True
1292         if (robo.planta > 100 and robo.planta < 200 and (
1293             (robo.planta - 100) not in numbers2 or (pm - 100) in numbers2)):
1294             plantaerror = True
1295         if (robo.planta > 200 and ((robo.planta - 200) not in numbers0) or (pm - 200) in numbers0):
1296             plantaerror = True
1297     else:
1298         if (robo.planta < 100 and robo.planta not in numbers1):
1299             plantaerror = True
1300         if (robo.planta > 100 and robo.planta < 200 and (robo.planta - 100) not in numbers2):
1301             plantaerror = True
1302         if (robo.planta > 200 and (robo.planta - 200) not in numbers0):
1303             plantaerror = True
```

Se crea la variable de control ‘plantaerror’, ya que existe la posibilidad de que se quiera trabajar con una planta que ya fue movida de esa posición, por lo que primero se revisa en ‘numbers’ que la planta siga ahí, en el caso de mover planta también se revisa que no haya ya una planta donde se quiere dejar la planta.

```

1305     if (plantaerror == False and text != "Medir humedad de todo" and text != "regar plantas con humedad baja"):
1306         monitor1.config(text=listbox.get(0))
1307         # definir que robot se va a mover
1308         robo.accion = True
1309         robo.tomar = True
1310         robo.entrando = False
1311         monitor2.configure(text="")
1312
1313     while (walle.accion == True or eva.accion == True):
1314         if (eva.orden == 'null'):
1315             walle = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1316             robo = walle
1317         else:
1318             eva = movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio)
1319             robo = eva
1320         walle.orden = 'null'
1321         eva.orden = 'null'
1322         pm = ""
1323         listbox.delete(0)
1324         monitor1.config(text="")
1325         global abortar
1326         if not listbox.get(0):
1327             AgregarPlanta.config(state="disabled")
1328             EliminarPlanta.config(state="disabled")
1329             monitor1.config(text="Ninguna orden activa")
1330             botonABORTAR.config(state="disabled")
1331             abortar = False
1332         else:
1333             monitor1.config(text=listbox.get(0))
1334             Orden(mapa0, mapa1, mapa2, walle, eva, pm)

```

Se asignan los parámetros a ‘robo’ para que inicie la orden, entonces entra a un loop de llamar a “movimiento”, esto hace que los valores de los robots cambien ligeramente, y ya que se necesita el cambio visual se hace que quien recibe los parámetros sea el robot original, ya que ‘robo’ vendría a ser una instancia temporal, aunque de todas formas se le deben asignar los parámetros del robot original para volver a llamar a “movimiento” con los datos de ‘robo’.

Una vez que termina “movimiento” se elimina la orden del robot, se elimina la orden del listbox, y se revisa si hay mas ordenes, si es que hay entonces se sigue con la siguiente, si es que no hay mas ordenes se desactivan algunos botones y se desactiva la opción de abortar.

```

1336     elif (text != "Medir humedad de todo" and text != "Regar plantas con humedad baja"):
1337         messagebox.showinfo("Message", 'La operacion: "' + listbox.get(
1338             0) + '" no pudo ser completada, ya que anteriormente se realizo un movimiento que lo impide')
1339         listbox.delete(0)
1340         walle.orden = 'null'
1341         eva.orden = 'null'
1342         pm = ""
1343         if not listbox.get(0):
1344             AgregarPlanta.config(state="disabled")
1345             EliminarPlanta.config(state="disabled")
1346             monitor1.config(text="Ninguna orden activa")
1347             botonABORTAR.config(state="disabled")
1348             abortar = False
1349         else:
1350             monitor1.config(text=listbox.get(0))
1351             Orden(mapa0, mapa1, mapa2, walle, eva, pm)

```

Equí se utiliza el ‘plantaerror’ de antes, si es True entonces se manda un mensaje diciendo que no se puede realizar la operación, y se elimina la orden. Además hace el proceso de que si hay otra orden después la empieza automáticamente.

```

1353     if (text == "Medir humedad de todo" or text == "Regar plantas con humedad baja"):
1354         listbox.delete(0)
1355         if not listbox.get(0):
1356             monitor1.config(text="Ninguna orden activa")
1357             botonABORTAR.config(state="disabled")
1358             abortar = False
1359         else:
1360             monitor1.config(text=listbox.get(0))
1361             Orden(mapa0, mapa1, mapa2, walle, eva, pm)

```

Lo último de “orden” es que en el caso de que haya sido una orden que abarque varias plantas, entonces hace lo de revisar si hay alguna otra orden en el listbox.

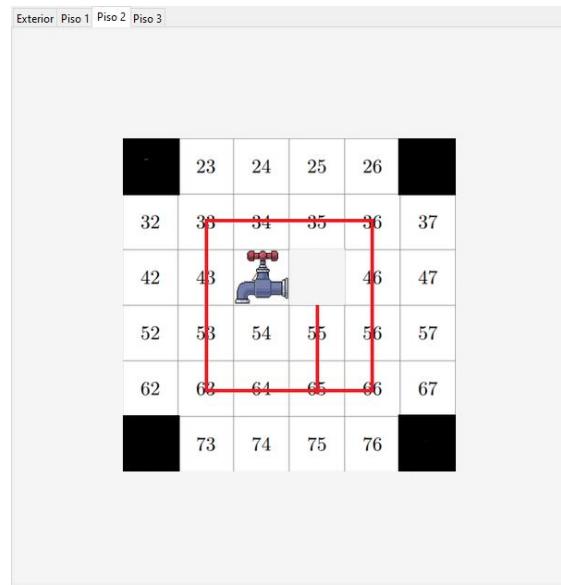
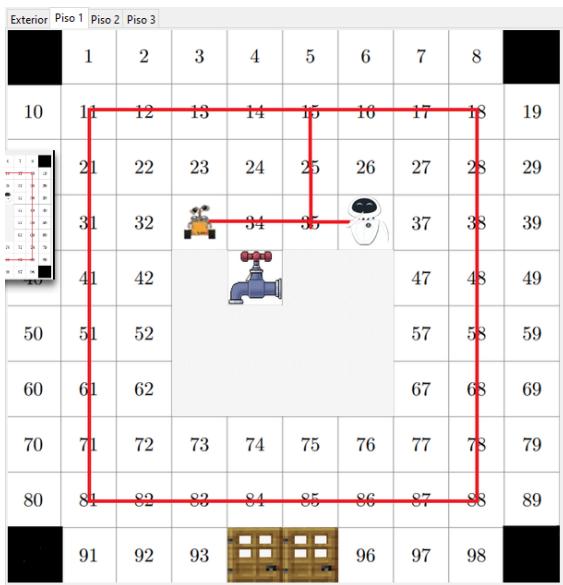
Con esto visto ya podemos volver a la función “movimiento” que está más arriba en el código.

```

447     # movimiento
448     def movimiento(mapa0, mapa1, mapa2, robo, pm, plantainicio):
449         global destino_finalRespaldo
450         global respaldo_planta
451         global abortar
452         if (robo.piso == 1):
453             destino = 18
454             destino1 = 88
455             destino2 = 81
456             destino3 = 11
457         if (robo.piso == 2):
458             destino = 36
459             destino1 = 66
460             destino2 = 63
461             destino3 = 33

```

Primero se definen las esquinas por las que puede pasar el robot en cada piso, esto le sirve para saber cuándo tiene que doblar.



```
462     if (abortar == True):
463         if (robo.tomar == False or robo.tomar == "regado"):
464             if (robo.tomar == False):
465                 robo.tomar = "regado"
466             if (robo.tomar == "regado" and robo.orden == "mover"):
467                 robo.planta = plantainicio
468         else:
469             robo.destino_final = 35
470             robo.tomar = "casa"
```

Luego se confirma que no se haya abortado, en caso de que se presione el botón de abortar la variable ‘abortar’ será True (esta parte está más abajo), lo que hará que ‘robo’ adopte los parámetros que tendría al final de la orden.

```
472     if (robo.piso == 0):
473         auxiliar = funciones.matriz(robo.x, robo.y, pixel)
474         if (robo.tomar != "casa"):
475             robo.destino_final = robo.planta - 10
476         else:
477             robo.destino_final = 35
```

Si el objetivo está en el exterior, entonces si es que va saliendo la posición objetivo es arriba de la planta, si es que va de vuelta entonces el objetivo es la estación de espera.

Cuando la función “matriz” recibe como parámetros ‘robo.x’ y ‘robo.y’ entonces significa que retorna la posición de matriz en la que se encuentra ‘robo’. En este caso se le asigna ese valor a ‘auxiliar’ para no tener que escribir la función cada vez y tener que hacer la llamada.

Hay que recordar que la única orden que involucra al piso exterior es la de mover planta.

```

479     if auxiliar == (robo.destino_final - 200):
480         # al tomar la planta
481         if (robo.tomar == True):
482             aux = int((robo.planta - 200) / 10)
483             mapa0[aux] = funciones.reemplazar(mapa0[aux], (robo.planta - 200) % 10, robo.tomar)
484             id = funciones.buscar_id(plantas0, robo.planta - 200)
485             canvas0.delete(id)
486             robo.tomar = "regado"
487             respaldo_planta = [info0[robo.planta - 200], stats0[robo.planta - 200], tipo0[robo.planta - 200],
488                                humedadesperada0[robo.planta - 200]]
489             info0[robo.planta - 200] = -1
490             stats0[robo.planta - 200] = -1
491             tipo0[robo.planta - 200] = ""
492             humedadesperada0[robo.planta - 200] = ""
493             plantas0.remove((robo.planta - 200, id))
494             numbers0.remove(robo.planta - 200)
495             auxpm = robo.planta
496             robo.planta = pm
497             pm = auxpm
498             if (auxiliar == 84 or auxiliar == 85) and robo.planta < 200:
499                 robo.key = "up"
500
501             if (auxiliar < 84 and (robo.planta >= auxiliar + 210 or robo.planta < 200)):
502                 robo.key = "right"
503
504             if (auxiliar > 85 and robo.planta <= auxiliar + 210):
505                 robo.key = "left"
506

```

'destino_final' es un valor sobre 200 ya que se calcula en base a la planta que se encuentra en el exterior, que como se mencionó anteriormente es mayor a 200. Mientras que auxiliar es menor a 100 ya que se calcula en base a la posición visual dentro de la matriz.

'robo.tomar'==True significa que Walle está recogiendo una planta, por lo que se cambia el texto del 'mapa0' para quitar la X, también busca la id de la planta que está recogiendo para eliminar la imagen de esta. 'robo.tomar' pasa a tener el valor de regado (este valor es debido a que la primera operación creada fue la de regar, y el código de esta podía ser usado para otras operaciones).

También se eliminan los datos de la planta, guardándose antes en un respaldo. Finalmente 'robo.planta' obtiene el valor de 'pm' el cual es la posición donde va a dejar la planta.

Dependiendo de donde se encuentra el robot se le asigna la dirección donde se tiene que mover.

```

507         # al dejar la planta
508     elif (robo.tomar == "regado"):
509
510         aux = int((robo.planta - 200) / 10)
511         mapa0[aux] = funciones.reemplazar(
512             mapa0[aux], (robo.planta - 200) % 10, robo.tomar)
513         posicion = funciones.black_matriz(robo.planta - 200, pixel)
514         id = canvas0.create_image(posicion[0], posicion[1], anchor="nw", image=imagen_planta)
515         planta = (robo.planta - 200), id
516         plantas0.append(planta)
517         robo.destino_final = 35
518         robo.tomar = "casa"
519         info0[robo.planta - 200] = respaldo_planta[0]
520         stats0[robo.planta - 200] = respaldo_planta[1]
521         tipo0[robo.planta - 200] = respaldo_planta[2]
522         humedadesperada0[robo.planta - 200] = respaldo_planta[3]
523         numbers0.append(robo.planta - 200)
524         if (auxiliar == 84 or auxiliar == 85):
525             robo.key = "up"
526
527             if (auxiliar < 84):
528                 robo.key = "right"
529
530             if (auxiliar > 85):
531                 robo.key = "left"

```

Si 'robo.tomar' == regado, significa que el robot esta actualmente dejando la planta en la posición indicada. Es bastante similar al proceso de tomarla, pero al opuesto. Modifica 'mapa0' para añadirle una X, Crea una planta con los datos de respaldo, se le asigna una nueva id, estos datos se asociarán nuevamente al número de planta. Finalmente cambia el valor de 'robo.tomar' a casa, lo que significa que termino la orden y se dirige a su estación de espera.

```

533     elif (robo.destino_final > 200):
534         if (auxiliar == 75 or auxiliar == 74):
535             robo.key = "down"
536
537         elif (auxiliar < robo.destino_final - 200):
538             robo.key = "right"
539
540         elif (auxiliar > robo.destino_final - 200):
541             robo.key = "left"
542
543     elif (robo.destino_final < 200):
544         if (auxiliar == 84 or auxiliar == 85):
545             robo.key = "up"
546
547         elif (auxiliar < 84):
548             robo.key = "right"
549
550         elif (auxiliar > 85):
551             robo.key = "left"

```

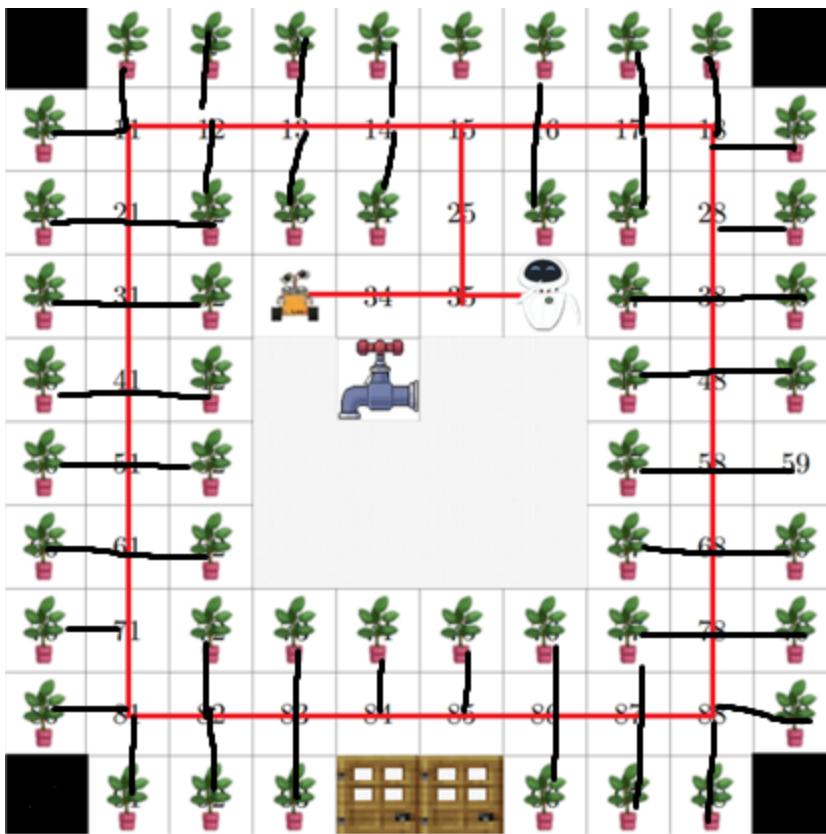
En el caso de que este en camino a su objetivo tomara valores dependiendo de si su objetivo esta fuera o dentro del invernadero.

```

553     if (robo.piso == 1):
554         # elegir el lugar al que se dirigira el robot para llegar a la planta
555         if (robo.tomar == True or robo.tomar == "regado"):
556             if (robo.planta < 9):
557                 robo.destino_final = robo.planta + 10
558             elif (robo.planta % 10 == 9):
559                 robo.destino_final = robo.planta - 1
560             elif (robo.planta > 90):
561                 robo.destino_final = robo.planta - 10
562             elif (robo.planta % 10 == 0):
563                 robo.destino_final = robo.planta + 1
564             elif (robo.planta >= 22 and robo.planta <= 27):
565                 robo.destino_final = robo.planta - 10
566             elif (robo.planta % 10 == 7 and robo.planta != 7 and robo.planta != 97):
567                 robo.destino_final = robo.planta + 1
568             elif (robo.planta >= 72 and robo.planta <= 77):
569                 robo.destino_final = robo.planta + 10
570             elif (robo.planta % 10 == 2 and robo.planta != 2 and robo.planta != 92):
571                 robo.destino_final = robo.planta - 1
572             destino_finalRespaldo = robo.destino_final

```

Esto simplemente significa que el robot interactuara con cada planta desde estas posiciones:



```

574         elif (robo.tomar == False and robo.orden == "regar" and robo.destino_final != 34):
575             robo.destino_final = 35
576
577             if funciones.matriz(robo.x, robo.y, pixel) == robo.destino_final:
578
579                 robo.quieto = 0
580                 aux = int(robo.planta / 10)

```

En la orden de regar ‘robo.tomar’ es False cuando ya se recogió la planta, y se esta camino a regarla, por lo que se acerca yendo a la posición 35.

Para cuando ‘robo’ esta justo en su objetivo, se queda quieto un momento para realismo, y a ‘aux’ se le asigna el valor de la decena de ‘robo.planta’

```

582         # al tomar la planta
583     if (robo.tomar == True and (robo.orden == "regar" or robo.orden == "mover")):
584         aux = int(robo.planta / 10)
585         mapa1[aux] = funciones.reemplazar(mapa1[aux], robo.planta % 10, robo.tomar)
586         id = funciones.buscar_id(plantas, robo.planta)
587         canvas1.delete(id)
588         robo.tomar = False
589     if (robo.orden == "mover"):
590         save.actualizarDatosM(1, mapa1)
591         save.actualizarDatos(robo.planta, 0, " ")
592         respaldo_planta = [info[robo.planta], stats[robo.planta], tipo[robo.planta],
593                             humedadesperada[robo.planta]]
594         info[robo.planta] = -1
595         stats[robo.planta] = -1
596         tipo[robo.planta] = ""
597         humedadesperada[robo.planta] = ""
598         plantas.remove((robo.planta, id))
599         numbers1.remove(robo.planta)
600         auxpm = robo.planta
601         robo.planta = pm
602         pm = auxpm
603         robo.tomar = "regado"

```

Cuando toma una planta para regar solo desaparece visualmente, ya que volverá a su misma posición no hay necesidad de cambiar otras cosas. Mientras que si la orden es mover entonces ocurre lo mismo que en el piso exterior, además de que en la base de datos reemplazará los valores de esa planta por datos sin valor para representar que ya no está ahí, también se modifica el mapa del piso 1 dentro de la base de datos.

Como se dijo anteriormente, con ‘robo.tomar’=False va camino a regar la planta, mientras que con ‘robo.tomar’=regado va a dejar la planta a la nueva posición.

```

607     # al dejar la planta
608     elif (robo.tomar == "regado" and robo.destino_final != 35 and robo.destino_final != 34 and (
609         robo.orden == "regar" or robo.orden == "mover")):
610         aux = int(robo.planta / 10)
611         mapa1[aux] = funciones.reemplazar(
612             mapa1[aux], robo.planta % 10, robo.tomar)
613         posicion = funciones.black_matriz(robo.planta, pixel)
614         id = canvas1.create_image(posicion[0], posicion[1], anchor="nw", image=imagen_planta)
615         planta = robo.planta, id
616         plantas.append(planta)
617         robo.destino_final = 35
618         robo.tomar = "casa"
619         if (robo.orden == "mover"):
620             save.actualizarDatosM(1, mapa1)
621             info[robo.planta] = respaldo_planta[0]
622             stats[robo.planta] = respaldo_planta[1]
623             tipo[robo.planta] = respaldo_planta[2]
624             humedadesperada[robo.planta] = respaldo_planta[3]
625             numbers1.append(robo.planta)
626             save.actualizarDatos(robo.planta, humedadesperada[robo.planta], tipo[robo.planta])
627             for item in TablaPiso1.get_children():
628                 TablaPiso1.delete(item)
629             for i in range(100):
630                 if i in numbers1:
631                     TablaPiso1.insert(parent='', index='end', iid=i, text='',
632                                     values=(i, info[i], humedadesperada[i], tipo[i]))

```

Cuando deja una planta este vuelve a aparecer visualmente y se le crea una nueva id, en ambas ordenes ya esta listo Walle para volver a su estación de espera.

Si la orden era mover entonces se le asignan parámetros a la planta desde 'respaldo_planta' igual que en el piso exterior, pero en este caso también se actualiza la tabla que contiene humedades, esto se hace eliminándola y creándola de nuevo, además los datos de esta planta se agregan a la base de datos, también se modifica el mapa en la base de datos.

```

633             # cuando esta en el grifo
634         elif (robo.destino_final == 35 and robo.tomar != "casa" and robo.orden == "regar"):
635             robo.key = "left"
636             robo.destino_final = 34
637         elif (robo.destino_final == 34 and robo.tomar != "casa" and robo.orden == "regar"):
638             robo.entrando = False
639             robo.destino_final = destino_finalRespaldo
640             robo.tomar = "regado"
641             stats[robo.planta] = humedadesperada[robo.planta] + 0.3
642             info[robo.planta] = str(stats[robo.planta]) + " aproximadamente cuando se rego"
643         for i in range(100):
644             if i in numbers1:
645                 TablaPiso1.delete(i)
646                 TablaPiso1.insert(parent='', index='end', iid=i, text='',
647                               values=(i, info[i], humedadesperada[i], tipo[i]))

```

Si está en camino a regar la planta entonces llegará a la posición 35 como se dijo un poco mas atrás, entonces cuando este ahí se mueve a la izquierda una posición, quedando frente al grifo.

Una vez frente al grifo ‘robo.entrando’ se volverá false, lo que significa que el robot va a salir de la estación. ‘robo.destino_final’ obtiene el valor de ‘destino_finalRespaldo’, el cual se obtiene al inicio de la parte del piso 1 en base a ‘robo.planta’, ‘robo.tomar’ pasa a tomar el valor regado, es decir que va a dejar la planta.

La humedad de la planta pasa a la humedad esperada de su tipo + 0.3, este valor es arbitrario.

En este caso igual se actualiza la tabla para mostrar la nueva humedad.

```

648             # al medir la planta
649         if (robo.orden == "medir" and robo.tomar == True):
650             RegarTodo.config(state="normal")
651             info[robo.planta] = stats[robo.planta]
652             robo.destino_final = 35
653             robo.tomar = "casa"
654         for i in range(100):
655             if i in numbers1:
656                 TablaPiso1.delete(i)
657                 TablaPiso1.insert(parent='', index='end', iid=i, text='',
658                               values=(i, info[i], humedadesperada[i], tipo[i]))

```

Si Eva está midiendo la humedad de la planta cambiara su ‘info’, el cual es el valor de humedad visible, por lo que la tabla se actualizará. Esta operación solo es medir por lo que ‘robo’ tomará parámetros para volver a la estación de espera.

```

661     # walle esta saliendo, se mueve a la derecha para llegar a 35
662     elif (robo.accion == True and robo.tomar != False and (
663         funciones.matriz(robo.x, robo.y, pixel) == 33 or funciones.matriz(robo.x, robo.y, pixel) == 34) and (
664             robo.orden == "regar" or robo.orden == "mover")):
665         robo.destino_final = 15
666
667         robo.key = "right"
668
669     # eva esta saliendo, se mueve a la izquierda para llegar a 35
670     elif (robo.accion == True and robo.tomar == True and funciones.matriz(robo.x, robo.y,
671                             pixel) == 36 and robo.orden == "medir")):
672         robo.destino_final = 15
673
674         robo.key = "left"
675
676
677     # robo sale subiendo hacia 15
678     elif (robo.entrando == False and funciones.traducir(robo.x, pixel) == 5 and funciones.traducir(robo.y,
679                             pixel) > 1 and funciones.matriz(
680                                 robo.x, robo.y, pixel) != 85):
681
682         robo.key = "up"
683
684     # robo entra a la base hacia 35
685     elif ((robo.destino_final == 35 or (
686         robo.destino_final > 100 and robo.destino_final < 200)) and funciones.traducir(robo.x,
687                             pixel) == 5 and funciones.matriz(
688                                 robo.x, robo.y, pixel) != 85 and funciones.matriz(robo.x, robo.y, pixel) != 45):
689
690         robo.key = "down"
691         robo.entrando = True

```

Aquí no hay mucho que decir, pues lo que hace cada fragmento ya está comentado, solo mencionar que ‘entrando’ define si el robot sube o baja cuando está en la columna de los terminados en 5.

```

693         # movimiento en sentido horario
694         elif ((robo.orden == "regar") == (robo.tomar == "regado")) and (robo.planta > 200) or (
695             ((robo.planta % 10 >= 5) and (robo.tomar == "regado" or robo.tomar == True)) or (
696                 (robo.planta % 10 < 5) and (robo.tomar == False or robo.tomar == "casa"))):
697

```

Para optimizar el movimiento usaremos esta condición, básicamente si robo se dirige a una planta cuyo valor en la unidad sea mayor o igual a 5 irá en sentido horario, y si viene desde una planta cuyo valor en la unidad sea menor a 5, se devolverá por sentido horario.

```

693     # movimiento en sentido horario
694     elif ((robo.orden == "regar") == (robo.tomar == "regado")) and (robo.planta > 200) or (
695         ((robo.planta % 10 >= 5) and (robo.tomar == "regado" or robo.tomar == True)) or (
696             (robo.planta % 10 < 5) and (robo.tomar == False or robo.tomar == "casa"))):
697
698         # robo esta en la fila 2, se mueve a la derecha
699         if (funciones.traducir(robo.x, pixel) < destino % 10 and funciones.traducir(robo.y, pixel) == int(
700             destino / 10)):
701
702             robo.key = "right"
703
704         # robo esta en la columna 8, se mueve hacia abajo
705         elif (funciones.traducir(robo.x, pixel) == destino1 % 10 and funciones.traducir(robo.y, pixel) < int(
706             destino1 / 10)):
707
708             robo.key = "down"
709
710         # robo esta en la fila 8, se mueve hacia la izquierda
711         elif (funciones.traducir(robo.x, pixel) > destino2 % 10 and funciones.traducir(robo.y, pixel) == int(
712             destino2 / 10)):
713
714             robo.key = "left"
715
716         # robo esta en la columna 2, se mueve hacia arriba
717         elif (funciones.traducir(robo.x, pixel) == destino3 % 10 and funciones.traducir(robo.y, pixel) > int(
718             destino3 / 10)):
719
720             robo.key = "up"
721

```

Se Usan los valores de los ‘destino’ que fueron definidos al inicio de “movimiento”, los cuales corresponden a las esquinas.

```

722     # movimiento en sentido antihorario
723     elif (((robo.planta % 10 < 5) and (robo.tomar == "regado" or robo.tomar == True)) or (
724             (robo.planta % 10 >= 5) and (robo.tomar == False or robo.tomar == "casa"))):
725         # robo esta en la fila 2, se mueve a la izquierda
726         if (funciones.traducir(robo.x, pixel) < destino % 10 and funciones.traducir(robo.y, pixel) == int(
727             destino / 10)):
728
729             robo.key = "left"
730
731         # robo esta en la columna 8, se mueve hacia arriba
732         elif (funciones.traducir(robo.x, pixel) == destino1 % 10 and funciones.traducir(robo.y, pixel) < int(
733             destino1 / 10)):
734
735             robo.key = "up"
736
737         # robo esta en la fila 8, se mueve hacia la derecha
738         elif (funciones.traducir(robo.x, pixel) > destino2 % 10 and funciones.traducir(robo.y, pixel) == int(
739             destino2 / 10)):
740
741             robo.key = "right"
742
743         # robo esta en la columna 2, se mueve hacia abajo
744         elif (funciones.traducir(robo.x, pixel) == destino3 % 10 and funciones.traducir(robo.y, pixel) > int(
745             destino3 / 10)):
746
747             robo.key = "down"
748
749             if (funciones.matriz(robo.x, robo.y, pixel) == destino3):
750                 robo.key = "down"
751             if (funciones.matriz(robo.x, robo.y, pixel) == destino2):
752                 robo.key = "right"
753             if (funciones.matriz(robo.x, robo.y, pixel) == destino1):
754                 robo.key = "up"
755             if (funciones.matriz(robo.x, robo.y, pixel) == destino):
756                 robo.key = "left"

```

Para el sentido antihorario se modifica ligeramente las acciones, pero la lógica sigue siendo la misma, excepto por el caso en el que se encuentre en una esquina en el que es necesario decirle a que dirección tiene que ir, ya que se usó el sentido horario como base para crear el sentido antihorario.

A partir de ahora el código se vuelve a utilizar en ambos sentidos.

```

758     # Walle esta entrando por la puerta
759     if (funciones.matriz(robo.x, robo.y, pixel) == 94 or funciones.matriz(robo.x, robo.y,
760                           pixel) == 95) and robo.key != "down":
761         robo.key = "up"

```

Si Walle está en la puerta al exterior y no está bajando, entonces tiene que subir.

```

763     if (robo.orden == "regar" or robo.orden == "mover"):
764         # walle se mueve desde 35 a 33
765         if (robo.tomar == "casa" and (
766             funciones.matriz(robo.x, robo.y, pixel) == 35 or funciones.matriz(robo.x, robo.y, pixel) == 34)):
767             robo.destino_final = 33
768             robo.quieto = 23
769             robo.key = "left"
770
771         # walle termina su accion
772         if (robo.tomar == "casa" and funciones.matriz(robo.x, robo.y, pixel) == 33):
773             robo.accion = False

```

Si Walle terminó su orden y está en la posición 35, entonces se mueve a la izquierda hacia 33.

Cuando llega a 33 ‘robo.accion’ pasa a ser False, como se vio anteriormente la iteración de llamar a “movimiento” ocurre mientras ‘robo.accion’ sea True, por lo que esta será la última iteración.

```

775     if (robo.orden == "medir"):
776         # eva se mueve de 35 a 36
777         if (robo.tomar == "casa" and funciones.matriz(robo.x, robo.y, pixel) == 35):
778             robo.destino_final = 36
779             robo.quieto = 23
780             robo.key = "right"
781
782         # eva termina su accion
783         if (robo.tomar == "casa" and funciones.matriz(robo.x, robo.y, pixel) == 36):
784             robo.accion = False

```

Con Eva se hace lo mismo, pero va a la posición 36.

```

786     # robo entra al elevador
787     if (robo.planta > 100 and robo.planta < 200 and robo.tomar == True):
788         if (funciones.matriz(robo.x, robo.y, pixel) == 33 or funciones.matriz(robo.x, robo.y, pixel) == 34):
789             robo.key = "right"
790             if (funciones.matriz(robo.x, robo.y, pixel) == 35):
791                 robo.key = "down"
792
793         # walle sale de la piramide
794         if (robo.orden == "mover" and (funciones.matriz(robo.x, robo.y, pixel) == 84 or funciones.matriz(robo.x, robo.y,
795                                         pixel) == 85) and robo.destino_final > 200):
796             robo.key = "down"
797

```

Para el piso 1 ya solo queda poner el cómo el robot saldrá del piso.

Para subir al segundo piso primero ‘robo.planta’ tendrá que estar entre 100 y 200, primero ‘robo’ va a 35 y entonces baja, con eso estará en la posición del ascensor, el cambio real está más abajo.

Para salir al exterior igualmente ‘robo.planta’ tiene que ser mayor a 200 y ‘robo’ tiene que estar en 84 o 85, entonces baja a la puerta, el cambio real está más abajo.

A partir de ahora viene el piso 2, es bastante similar al piso 1 así que solo se verá de forma general:

```

798     if (robo.piso == 2):
799         # elegir el lugar al que se dirigira el robot para llegar a la planta
800         if (robo.tomar == True or robo.tomar == "regado"):
801             if (robo.planta < 128):
802                 robo.destino_final = robo.planta + 10
803             elif (robo.planta % 10 == 7):
804                 robo.destino_final = robo.planta - 1
805             elif (robo.planta > 172):
806                 robo.destino_final = robo.planta - 10
807             elif (robo.planta % 10 == 2):
808                 robo.destino_final = robo.planta + 1
809             elif (robo.tomar == False and robo.orden == "regar"):
810                 robo.destino_final = 154

```

Se obtiene 'robo.destino_final' en base a 'robo.planta'.

```

812     if funciones.matriz(robo.x, robo.y, pixel) == (robo.destino_final - 100):
813
814         robo.quieto = 0
815         aux = int((robo.planta - 100) / 10)
816
817         # al tomar la planta
818         if (robo.tomar == True and (robo.orden == "regar" or robo.orden == "mover")):
819             aux = int((robo.planta - 100) / 10)
820             mapa2[aux] = funciones.reemplazar(mapa2[aux], (robo.planta - 100) % 10, robo.tomar)
821             id = funciones.buscar_id(plantas2, robo.planta - 100)
822             canvas2.delete(id)
823             robo.tomar = False
824             if (robo.orden == "mover"):
825                 save.actualizarDatosM(2, mapa2)
826                 save.actualizarDatos2(robo.planta, 0, " ")
827                 respaldo_planta = [info2[robo.planta - 100], stats2[robo.planta - 100], tipo2[robo.planta - 100],
828                                     humedadesperada2[robo.planta - 100]]
829                 info2[robo.planta - 100] = -1
830                 stats2[robo.planta - 100] = -1
831                 tipo2[robo.planta - 100] = ""
832                 humedadesperada2[robo.planta - 100] = ""
833                 plantas2.remove((robo.planta - 100, id))
834                 numbers2.remove(robo.planta - 100)
835                 auxpm = robo.planta
836                 robo.planta = pm
837                 pm = auxpm
838                 robo.tomar = "regado"
839

```

Cuando Walle recoge la planta se borra la imagen, y si es para mover se guardan y eliminan sus valores.

Cabe destacar que en el piso 2 se resta 100 a algunas variables, ya que la medida en pixeles en ambos pisos es la misma, pero variables como 'robo.planta' no lo son.

```

843         # al dejar la planta
844     elif (robo.tomar == "regado" and robo.destino_final != 154 and (
845             robo.orden == "regar" or robo.orden == "mover")):
846         aux = int((robo.planta - 100) / 10)
847         mapa2[aux] = funciones.reemplazar(
848             mapa2[aux], (robo.planta - 100) % 10, robo.tomar)
849         posicion = funciones.black_matriz(robo.planta - 100, pixel)
850         id = canvas2.create_image(posicion[0], posicion[1], anchor="nw", image=imagen_planta)
851         planta = (robo.planta - 100), id
852         plantas2.append(planta)
853         robo.destino_final = 35
854         robo.tomar = "casa"
855     if (robo.orden == "mover"):
856         save.actualizarDatosM2(mapa2)
857         info2[robo.planta - 100] = respaldo_planta[0]
858         stats2[robo.planta - 100] = respaldo_planta[1]
859         tipo2[robo.planta - 100] = respaldo_planta[2]
860         humedadesperada2[robo.planta - 100] = respaldo_planta[3]
861         numbers2.append(robo.planta - 100)
862         save.actualizarDatos2(robo.planta, humedadesperada2[robo.planta - 100], tipo2[robo.planta - 100])
863         for item in TablaPiso2.get_children():
864             TablaPiso2.delete(item)
865         for i in range(100):
866             if i in numbers2:
867                 TablaPiso2.insert(parent='', index='end', iid=i, text='',
868                               values=(i + 100, info2[i], humedadesperada2[i], tipo2[i]))

```

Cuando Walle deja una planta se vuelve a crear la imagen, y si la orden era para mover entonces se le ponen valores desde el respaldo.

```

870     # cuando esta en el grifo
871     elif (robo.destino_final == 154 and robo.tomar != "casa" and robo.orden == "regar"):
872         robo.entrando = False
873         robo.destino_final = destino_finalRespaldo
874         robo.tomar = "regado"
875         stats2[robo.planta - 100] = humedadesperada2[robo.planta - 100] + 2
876         info2[robo.planta - 100] = str(stats2[robo.planta - 100]) + " aproximadamente"
877         if (robo.planta > 160):
878             robo.key = "down"
879         elif ((robo.planta - 100) % 10 < 5):
880             robo.key = "left"
881         elif ((robo.planta - 100) % 10 >= 5):
882             robo.key = "right"
883         for i in range(100):
884             if i in numbers2:
885                 TablaPiso2.delete(i)
886                 TablaPiso2.insert(parent='', index='end', iid=i, text='',
887                               values=(i + 100, info2[i], humedadesperada2[i], tipo2[i]))

```

Al regar la planta se le cambian los datos, y se define el camino más corto para dejar la planta.

```
888         # al medir la planta
889         if (robo.orden == "medir" and robo.tomar == True):
890             RegarTodo.config(state="normal")
891             info2[robo.planta - 100] = stats2[robo.planta - 100]
892             robo.destino_final = 35
893             robo.tomar = "casa"
894             for i in range(100):
895                 if i in numbers2:
896                     TablaPiso2.delete(i)
897                     TablaPiso2.insert(parent='', index='end', iid=i, text='',
898                                     values=(i + 100, info2[i], humedadesperada2[i], tipo2[i]))
899
```

Al medir la planta se muestra la humedad en la tabla del piso 2.

```
901         # robo entra hacia el elevador
902         elif ((robo.destino_final > 200 or robo.destino_final < 100) and funciones.traducir(robo.x,
903                                         pixel) == 5 and funciones.matriz(
904                                         robo.x, robo.y, pixel) != 35):
905
906             robo.key = "up"
907             robo.entrando = True
```

Si está en la columna de valor 5 y termino su tarea en el piso 2, entonces sube hacia el elevador.

```
910
911             elif (robo.destino_final == 154 and funciones.matriz(robo.x, robo.y, pixel) == 64):
robo.key = "up"
```

En 54 se puede poner una planta, es un lugar extraño así que tiene su if específico.

```

913     # sentido antihorario
914     elif (((robo.planta % 10 >= 5) and (robo.tomar == "regado" or robo.tomar == True)) or (
915         (robo.planta % 10 < 5) and (robo.tomar == False or robo.tomar == "casa"))):
916         # robo esta en la fila 2, se mueve a la izquierda
917         if (funciones.traducir(robo.x, pixel) < destino % 10 and funciones.traducir(robo.y, pixel) == int(
918             destino / 10)):
919             robo.key = "left"
920
921         # robo esta en la columna 8, se mueve hacia arriba
922         elif (funciones.traducir(robo.x, pixel) == destino1 % 10 and funciones.traducir(robo.y, pixel) < int(
923             destino1 / 10)):
924
925             robo.key = "up"
926
927         # robo esta en la fila 8, se mueve hacia la derecha
928         elif (funciones.traducir(robo.x, pixel) > destino2 % 10 and funciones.traducir(robo.y, pixel) == int(
929             destino2 / 10)):
930
931             robo.key = "right"
932
933         # robo esta en la columna 2, se mueve hacia abajo
934         elif (funciones.traducir(robo.x, pixel) == destino3 % 10 and funciones.traducir(robo.y, pixel) > int(
935             destino3 / 10)):
936             robo.key = "down"
937
938         if (funciones.matriz(robo.x, robo.y, pixel) == destino3):
939             robo.key = "down"
940         if (funciones.matriz(robo.x, robo.y, pixel) == destino2):
941             robo.key = "right"
942         if (funciones.matriz(robo.x, robo.y, pixel) == destino1):
943             robo.key = "up"
944         if (funciones.matriz(robo.x, robo.y, pixel) == destino):
945             robo.key = "left"

```

Igual que en el piso 1 se hace el camino que recorre 'robo' en sentido antihorario

```
947     # sentido horario
948     elif (((robo.planta % 10 < 5) and (robo.tomar == "regado" or robo.tomar == True)) or (
949             (robo.planta % 10 >= 5) and (robo.tomar == False or robo.tomar == "casa"))):
950
951         # robo esta en la fila 2, se mueve a la derecha
952         if (funciones.traducir(robo.x, pixel) < destino % 10 and funciones.traducir(robo.y, pixel) == int(
953             destino / 10)):
954
955             robo.key = "right"
956
957         # robo esta en la columna 8, se mueve hacia abajo
958         elif (funciones.traducir(robo.x, pixel) == destino1 % 10 and funciones.traducir(robo.y, pixel) < int(
959             destino1 / 10)):
960
961             robo.key = "down"
962
963         # robo esta en la fila 8, se mueve hacia la izquierda
964         elif (funciones.traducir(robo.x, pixel) > destino2 % 10 and funciones.traducir(robo.y, pixel) == int(
965             destino2 / 10)):
966
967             robo.key = "left"
968
969         # robo esta en la columna 2, se mueve hacia arriba
970         elif (funciones.traducir(robo.x, pixel) == destino3 % 10 and funciones.traducir(robo.y, pixel) > int(
971             destino3 / 10)):
972
973             robo.key = "up"
```

También se hace en sentido horario. Con eso ya está listo el piso 2.

```
999      elif (robo.orden == "medir"): # movimiento de eva
1000          if robo.key == "left":
1001              robo.x += -pixel
1002              eva = robo
1003              izquierda(eva)
1004              robo = eva
1005          elif robo.key == "right":
1006              robo.x += pixel
1007              eva = robo
1008              derecha(eva)
1009              robo = eva
1010          elif robo.key == "up":
1011              robo.y += -pixel
1012              eva = robo
1013              arriba(eva)
1014              robo = eva
1015          elif robo.key == "down":
1016              robo.y += pixel
1017              eva = robo
1018              abajo(eva)
1019              robo = eva
```

Primeramente, se puede ver que la posición de 'robo' se ajustara cada 20 iteraciones. Dependiendo de 'robo.key' se le sumara o restara el valor de píxel a una de sus coordenadas. Entonces se llama a la función correspondiente que es donde se va a mover el Sprite del robot. (Estas funciones se encuentran debajo de "movimiento")

```

968     elif(robo.orden == "medir"): #movimiento de eva
969         if robo.key == "left":
970             robo.x += -pixel
971             eva = robo
972             izquierda(eva)
973             robo = eva
974         elif robo.key == "right":
975             robo.x += pixel
976             eva = robo
977             derecha(eva)
978             robo = eva
979         elif robo.key == "up":
980             robo.y += -pixel
981             eva = robo
982             arriba(eva)
983             robo= eva
984         elif robo.key == "down":
985             robo.y += pixel
986             eva = robo
987             abajo(eva)
988             robo = eva

```

Se hace lo mismo, pero con 'eva'.

```

1021     if (robo.piso == 0):
1022         # walle entra a la piramide
1023         if (robo.destino_final < 200 and (
1024             funciones.matriz(robo.x, robo.y, pixel) == 74 or funciones.matriz(robo.x, robo.y, pixel) == 75)):
1025             canvas0.delete(robo.id)
1026             robo.id = canvas1.create_image(robo.x, robo.y + (2 * pixel), anchor="nw", image=fotorobot[0])
1027             robo.piso = 1
1028             robo.key = "up"
1029             robo.y += 2 * pixel

```

Ahora viene el eliminar y agregar sprites de los robots, en el caso de que Walle entre al invernadero primero se elimina el Sprite del canvas del segundo piso, inmediatamente se crea la misma imagen en el canvas del piso 1, se le cambia el valor de 'robo.piso'. Para la coordenada Y a 'robo.y' se le suman 2 pixeles para que visualmente baje 2 cuadros.

```

1031     if (robo.piso == 1):
1032         # el robot cambia de piso
1033
1034         if (funciones.matriz(robo.x, robo.y, pixel) == 45 and robo.destino_final > 100):
1035             canvas1.delete(robo.id)
1036             if (robo.orden == "medir"):
1037                 robo.id = canvas2.create_image(robo.x, robo.y, anchor='nw', image=fotorobot[1])
1038             else:
1039                 robo.id = canvas2.create_image(robo.x, robo.y, anchor='nw', image=fotorobot[0])
1040             robo.piso = 2
1041             robo.key = "down"
1042
1043         # walle sale de la piramide
1044         if ((robo.planta > 200 and robo.tomar != "casa") and (
1045             funciones.matriz(robo.x, robo.y, pixel) == 94 or funciones.matriz(robo.x, robo.y, pixel) == 95)):
1046             canvas1.delete(robo.id)
1047             robo.id = canvas0.create_image(robo.x, robo.y - (2 * pixel), anchor='nw', image=fotorobot[0])
1048             robo.piso = 0
1049             robo.key = "down"
1050             robo.y -= 2 * pixel

```

Para cuando desaparece del piso 1 es lo mismo, solo hay que tener cuidado con el cambio de posición.

```

1051     elif (robo.piso == 2):
1052         # robo baja al primer piso
1053         if (funciones.matriz(robo.x, robo.y, pixel) == 45 and (robo.destino_final < 100 or robo.destino_final > 200)):
1054             canvas2.delete(robo.id)
1055             if (robo.orden == "medir"):
1056                 robo.id = canvas1.create_image(robo.x, robo.y, anchor='nw', image=fotorobot[1])
1057             else:
1058                 robo.id = canvas1.create_image(robo.x, robo.y, anchor='nw', image=fotorobot[0])
1059             robo.piso = 1
1060             robo.key = "up"

```

Es lo mismo para el piso 2.

```
1062     return robo
```

Con eso ya terminamos con “movimiento”, solo queda retornar ‘robo’ para que se actualicen los parámetros del robot que este haciendo la acción.

```

1065     # movimiento de la imagen del robot a la izquierda
1066     def izquierda(robo):
1067         if (robo.piso == 0):
1068             canvas0.move(robo.id, -20, 0)
1069             ventana.update()
1070             coords = canvas0.coords(robo.id)
1071
1072             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1073                 canvas0.after(200, izquierda(robo))
1074         if (robo.piso == 1):
1075             canvas1.move(robo.id, -20, 0)
1076             ventana.update()
1077             coords = canvas1.coords(robo.id)
1078
1079             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1080                 canvas1.after(200, izquierda(robo))
1081         if (robo.piso == 2):
1082             canvas2.move(robo.id, -20, 0)
1083             ventana.update()
1084             coords = canvas2.coords(robo.id)
1085
1086             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1087                 canvas2.after(200, izquierda(robo))

```

Como se dijo anteriormente falta el movimiento visual de los sprites de los robots, empezaremos por la función “izquierda”, la cual era llamada cuando ‘robo.key’ era igual a left.

Se utiliza la id del robot para mover la imagen en 20 pixeles a la izquierda (recordad que ‘pixel’ siempre es múltiplo de 20), se actualiza la ventana para que se vean los cambios, se guardan las nuevas coordenadas de la imagen, entonces se verifica si es que ya cambio exactamente de cuadro, en caso de que todavía no lo haga se volverá a llamar a si misma con un retraso de 0,2 segundos para que sea visible y no se vea al robot teletransportándose.

Es lo mismo para todos los pisos, solo cambia el canvas en que se hace.

```

1090     # movimiento de la imagen del robot a la derecha
1091     def derecha(robo):
1092         if (robo.piso == 0):
1093             canvas0.move(robo.id, 20, 0)
1094             ventana.update()
1095             coords = canvas0.coords(robo.id)
1096             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1097                 canvas0.after(200, derecha(robo))
1098         if (robo.piso == 1):
1099             canvas1.move(robo.id, 20, 0)
1100             ventana.update()
1101             coords = canvas1.coords(robo.id)
1102             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1103                 canvas1.after(200, derecha(robo))
1104         if (robo.piso == 2):
1105             canvas2.move(robo.id, 20, 0)
1106             ventana.update()
1107             coords = canvas2.coords(robo.id)
1108             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1109                 canvas2.after(200, derecha(robo))

1110
1111
1112     # movimiento de la imagen del robot hacia arriba
1113     def arriba(robo):
1114         if (robo.piso == 0):
1115             canvas0.move(robo.id, 0, -20)
1116             ventana.update()
1117             coords = canvas0.coords(robo.id)

1118             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1119                 canvas0.after(200, arriba(robo))
1120         if (robo.piso == 1):
1121             canvas1.move(robo.id, 0, -20)
1122             ventana.update()
1123             coords = canvas1.coords(robo.id)

1124             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1125                 canvas1.after(200, arriba(robo))
1126         if (robo.piso == 2):
1127             canvas2.move(robo.id, 0, -20)
1128             ventana.update()
1129             coords = canvas2.coords(robo.id)

1130             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1131                 canvas2.after(200, arriba(robo))

1132
1133
1134     # movimiento de la imagen del robot hacia abajo
1135     def abajo(robo):
1136         if (robo.piso == 0):
1137             canvas0.move(robo.id, 0, 20)
1138             ventana.update()
1139             coords = canvas0.coords(robo.id)

1140             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1141                 canvas0.after(200, abajo(robo))
1142         if (robo.piso == 1):
1143             canvas1.move(robo.id, 0, 20)
1144             ventana.update()
1145             coords = canvas1.coords(robo.id)

1146             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1147                 canvas1.after(200, abajo(robo))
1148         if (robo.piso == 2):
1149             canvas2.move(robo.id, 0, 20)
1150             ventana.update()
1151             coords = canvas2.coords(robo.id)

1152             if (funciones.exactitud(coords[0], coords[1], pixel) == False):
1153                 canvas2.after(200, abajo(robo))
1154
1155
1156
1157
1158
1159

```

Es lo mismo en todas las direcciones, solo cambia hacia donde es el movimiento de 20 pixeles.

A partir de ahora están escritas funciones que ya fueron mostradas más arriba en el documento, así que por favor fijarse en el número de la línea.

```
1162     def clear():
1163         Salida1.config(text="")
1164         Salida2.config(text="")
1165         cb1.config(state="disabled")
1166         botonCancelar.config(state="disabled")
1167         RegarPlanta.config(state="disabled")
1168         MoverPlanta.config(state="disabled")
1169         MedirHumedad.config(state="disabled")
1170         AgregarPlanta.config(state="disabled")
1171         EliminarPlanta.config(state="disabled")
1172
```

La función “clear” desactiva botones y limpia los monitores donde salen números de planta, también desactiva el combobox donde están los tipos de plantas. Estos volverán a activarse al seleccionar una nueva posición que este relacionada.

```
1366     # funcionesBotones
1367     def Subir():
1368         idxs = listbox.curselection()
1369         if (str(idxs) != "(1,)"):
1370             for pos in idxs:
1371                 if pos == 0:
1372                     continue
1373                 listbox.selection_clear(0, tk.END)
1374                 text = listbox.get(pos)
1375                 listbox.delete(pos)
1376                 listbox.insert(pos - 1, text)
1377                 listbox.selection_set(pos - 1)
```

Subir se usa para hacer que una orden en específico suba una posición en el orden de instrucciones. Se obtiene el índice del elemento que haya sido seleccionado usando “curselection”, ya que la orden con índice 0 se está ejecutando actualmente entonces si el índice es 1 no se puede subir más. Funciona de forma que obtiene el String usando el índice y lo guarda en ‘text’, lo elimina y luego pone el valor de ‘text’ en una posición mas arriba.

```
1380     def Bajar():
1381         idxs = listbox.curselection()
1382         if (str(idxs) != "(0,)"):
1383             for pos in idxs:
1384                 text = listbox.get(pos)
1385                 listbox.delete(pos)
1386                 listbox.insert(pos + 1, text)
1387                 listbox.selection_set(pos + 1)
1388
```

“bajar” funciona de la misma forma, pero con +1 en lugar de -1.

```
1390     def borrarElemento():
1391         for i in listbox.curselection():
1392             listbox.delete(i)
```

Para borrar un elemento de la listbox simplemente se usa “delete” en el item seleccionado.

```
1395     def updatecanvas3():
1396         progress['value'] += 20
1397         canvas3.update_idletasks()
1398         if progress['value'] <= 100:
1399             canvas3.after(400, updatecanvas3)
1400
1401
1402     def open():
1403         progress['value'] = 20
1404         canvas3.update_idletasks()
1405         canvas3.after(400, updatecanvas3)
1406         posMalla = "Estado actual de la Malla = abierto"
1407         labelmalla.config(text=posMalla)
1408
1409
1410     def mid():
1411         progress['value'] = 20
1412         canvas3.update_idletasks()
1413         canvas3.after(400, updatecanvas3)
1414         posMalla = "Estado actual de la Malla = intermedio"
1415         labelmalla.config(text=posMalla)
1416
1417
1418     def closed():
1419         progress['value'] = 20
1420         canvas3.update_idletasks()
1421         canvas3.after(400, updatecanvas3)
1422         posMalla = "Estado actual de la Malla = cerrado"
1423         labelmalla.config(text=posMalla)
1424
```

Estos son botones meramente estéticos, lo que hacen es hacer avanzar una barra de carga un poco, luego llaman a “updatecanvas3” para que ese cargue el resto de la barra, aprovechando la función “after”.

```
1437     def switch2():
1438         global is_on2
1439         # Determin is on or off
1440         if is_on2:
1441             on_button2.config(image=off)
1442             is_on2 = False
1443         else:
1444             on_button2.config(image=on)
1445             is_on2 = True
1446
1447
1448     def switch3():
1449         global is_on3
1450         # Determin is on or off
1451         if is_on3:
1452             on_button3.config(image=off)
1453             is_on3 = False
1454         else:
1455             on_button3.config(image=on)
1456             is_on3 = True
1457
1458
1459     def switch4():
1460         global is_on4
1461         # Determin is on or off
1462         if is_on4:
1463             on_button4.config(image=off)
1464             is_on4 = False
1465         else:
1466             on_button4.config(image=on)
1467             is_on4 = True
1468
```

Los switch solo cambian de imagen dependiendo de si es presionado.

```
1470     is_on1 = True
1471     is_on2 = True
1472     is_on3 = True
1473     is_on4 = True
```

Además, empiezan en True.

```

1565     def Agregar_planta():
1566         if (not listbox.get(0)):
1567             nueva = 0
1568
1569             if (cb1.get() == ""):
1570                 messagebox.showinfo("Message", "Debe de seleccionar el tipo de planta")
1571
1572         else:
1573             nueva = int(Salida2.cget("text")) - 200
1574             posicion = funciones.black_matriz(nueva, pixel)
1575             id = canvas0.create_image(posicion[0], posicion[1], anchor="nw", image=imagen_planta)
1576             planta = nueva, id
1577             plantas0.append(planta)
1578             info0[nueva] = "?"
1579             tipo0[nueva] = cb1.get() # cambiar, se debe poner manualmente
1580             stats0[nueva] = round(random.uniform(0, 2), 3)
1581             if (cb1.get() == "Cebolla"):
1582                 humedadesperada0[nueva] = 4.52
1583             elif (cb1.get() == "Ajo"):
1584                 humedadesperada0[nueva] = 2.34
1585             elif (cb1.get() == "Espinaca"):
1586                 humedadesperada0[nueva] = 3.6
1587             elif (cb1.get() == "Alcachofa"):
1588                 humedadesperada0[nueva] = 7.3
1589             elif (cb1.get() == "Zanahoria"):
1590                 humedadesperada0[nueva] = 2.3
1591             elif (cb1.get() == "Albahaca"):
1592                 humedadesperada0[nueva] = 6.7
1593             elif (cb1.get() == "Calabaza"):
1594                 humedadesperada0[nueva] = 3.7
1595             elif (cb1.get() == "Remolacha"):
1596                 humedadesperada0[nueva] = 12.3
1597
1598             numbers0.append(nueva)
1599             AgregarPlanta.config(state="disabled")
1600             clear()
1601
1602         else:
1603             messagebox.showinfo("Message", "espere a que los robots terminen sus funciones")
1604

```

Para el botón de agregar planta primero se advierte que no hay una planta seleccionada si es que en el combobox no se seleccionó algún tipo.

Se crea una nueva planta con humedad aleatoria y posición en el lugar seleccionado del piso exterior. Dependiendo del dato que sea entregado del combobox se le pondrá un tipo y humedad ideal.

Esta operación no se podrá realizar si es que hay alguna orden activa, pues podría generar conflicto crear una planta donde Walle este camino a dejar otra.

```
1606     def Eliminar_planta():
1607         if (not listbox.get(0)):
1608             vieja = int(Salida1.cget("text")) - 200
1609             id = funciones.buscar_id(plantas0, vieja)
1610             canvas0.delete(id)
1611             info0[vieja] = -1
1612             stats0[vieja] = -1
1613             tipo0[vieja] = ""
1614             humedadesperada0[vieja] = ""
1615             plantas0.remove((vieja, id))
1616             numbers0.remove(vieja)
1617             EliminarPlanta.config(state="disabled")
1618             clear()
1619         else:
1620             messagebox.showinfo("Message", "espere a que los robots terminen sus funciones")
1621
```

Para eliminar una planta se obtiene la id de la planta seleccionada, entonces se borra del canvas y listas de plantas y sus estadísticas pasan a ser -1 o Strings vacíos. Esta tampoco podrá ser efectuada mientras hayan robots trabajando.

```
1623     def abort():
1624         global abortar
1625         abortar = True
1626
```

Para el botón de abortar solo se cambia el valor de ‘abortar’, en “movimiento” había un if que hacia que si ‘abortar’ era True entonces los parámetros del robot se volvían como si hubiera terminado la tarea.

```
1628     # FramesVentana
1629     framegeneral = ttk.Frame(frameizquierda)
1630     framelisybotones = ttk.Frame(framegeneral)
1631     frameListbox = ttk.Frame(framelisybotones)
1632     frameminimover = ttk.Frame(frameMovimiento)
1633     frameminiagregar = ttk.Frame(frameMovimiento)
1634     frameWalle = ttk.Frame(framegeneral)
1635     frameEva = ttk.Frame(framegeneral)
1636     framecalefactor = ttk.Frame(canvas3)
1637     frameinvernadero = ttk.Frame(canvas3)
```

Aquí se crean algunos frames para ordenar la interfaz.

```
1639 # Lista de Instrucciones robots
1640 listbox = tk.Listbox()
1641 scrollbar = ttk.Scrollbar(frameListbox, orient=tk.VERTICAL)
1642 listbox = tk.Listbox(frameListbox, yscrollcommand=scrollbar.set)
1643 scrollbar.config(command=listbox.yview)
1644 scrollbar.pack(side=tk.RIGHT, fill=tk.BOTH)
```

Se define la listbox, notar que se tiene que configurar un scrollbar para usarlo en la listbox.

```
1646 # imagenesBotones
1647
1648 on = tk.PhotoImage(file=".assets/on.png")
1649 off = tk.PhotoImage(file=".assets/off.png")
1650
1651 framelbls = ttk.Frame(framecalefactor)
1652 framebtns = ttk.Frame(framecalefactor)
```

Se crean variables que reciban una imagen de on y off, las cuales fueron usadas mas arriba en los botones estéticos.

```

1655     def PruebaMov(mapa0, mapa1, mapa2, walle, eva, pm):
1656         if (not listbox.get(0)):
1657             global abortar
1658             listbox.insert(tk.END, "prueba de movimiento")
1659             robo = walle
1660             robo.orden = "regar"
1661             robo.accion = True
1662             robo.tomar = True
1663             robo.entrando = False
1664             robo.planta = 92
1665
1666         while (walle.accion == True):
1667
1668             walle = movimiento(mapa0, mapa1, mapa2, robo, 1, 1)
1669             robo = walle
1670             if (funciones.matriz(robo.x, robo.y, pixel) == 51):
1671                 abortar = True
1672             walle.orden = 'null'
1673             abortar = False
1674
1675             robo = eva
1676             robo.orden = "medir"
1677             robo.accion = True
1678             robo.tomar = True
1679             robo.entrando = False
1680             robo.planta = 92
1681
1682         while (eva.accion == True):
1683
1684             eva = movimiento(mapa0, mapa1, mapa2, robo, 1, 1)
1685             robo = eva
1686             if (funciones.matriz(robo.x, robo.y, pixel) == 51):
1687                 abortar = True
1688             eva.orden = 'null'
1689             abortar = False
1690             listbox.delete(0)
1691             if (listbox.get(0)):
1692                 monitor1.config(text=listbox.get(0))
1693                 Orden(mapa0, mapa1, mapa2, walle, eva, pm)
1694             else:
1695                 messagebox.showinfo("Message",
1696                                 'esta operacion solo se puede realizar si los robots no estan ejecutando otra orden')

```

“PruebaMov” es para revisar que los robots se muevan bien, es básicamente lo mismo que se hacia en “orden” solo que acá ya están definidos los parámetros, además de que solo se podrá realizar si no hay otra orden activa.

```
1729     # walle Ventana
1730     img1 = ImageTk.PhotoImage(Image.open("./assets/walleHD.png"))
1731     panel1 = tk.Label(frameWalle, image=img1)
1732     panel1.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1733     listbox.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1734     frameWalle.pack(side=tk.LEFT)
1735
1736     # eva Ventana
1737     img2 = ImageTk.PhotoImage(Image.open("./assets/evaHD.png"))
1738     panel2 = tk.Label(frameEva, image=img2)
1739     panel2.pack(side=tk.TOP)
1740     frameEva.pack(side=tk.LEFT)
1741     # generales:
1742     frameListbox.pack(side=tk.TOP)
1743     framelisybotones.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1744
```

Se colocan las imágenes de los robots que servirán para identificar que botón con instrucción pertenece a cual

```
1745     progress = ttk.Progressbar(canvas3, orient=tk.HORIZONTAL,
1746                               length=100, mode='determinate')
1747     posMalla = "Estado actual de la Malla =-----"
```

Esta es la barra de carga que se mencionó antes.

```
1750     lblblanca = tk.Label(framelbls, height=3)
1751     lblcalefactor = tk.Label(framelbls, height=3)
1752     lblbombaAgua = tk.Label(framelbls, height=3)
1753     lblVentana = tk.Label(framelbls, height=3)
1754     lblPuerta = tk.Label(framelbls, height=3)
1755     lblblanca.config(text=" ")
1756     lblcalefactor.config(text="Estado actual calefactor:")
1757     lblbombaAgua.config(text="Estado actual Bomba de agua:")
1758     lblPuerta.config(text="Estado actual Puerta:")
1759     lblVentana.config(text="Estado actual Ventana:")
1760
```

Se crean algunos labels que contendrán el texto que muestra estados del invernadero.

```
1761     lblcalefactor.pack(side=tk.TOP)
1762     lblbombaAgua.pack(side=tk.TOP)
1763     lblPuerta.pack(side=tk.TOP)
1764     lblVentana.pack(side=tk.TOP)
1765
1766     on_button1.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1767     on_button2.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1768     on_button3.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1769     on_button4.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1770
1771     framecalefactor.pack(side=tk.BOTTOM)
1772     framebt�s.pack(side=tk.RIGHT)
1773     framelbls.pack(side=tk.LEFT)
1774     labelmalla = tk.Label(frameinvernadero)
```

Se coloca visualmente los labels, botones y frames.

```
1776     progress.pack(pady=10)
1777     labelmalla.pack(padx=200)
1778     open_button.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)
1779     mid_button.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)
1780     close_button.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)
1781     frameinvernadero.pack(side=tk.RIGHT)
1782     labelmalla.config(text=posMalla)
1783
```

Se coloca visualmente las cosas que tienen que ver con la barra de carga.

```
1791     # listbox
1792     BorrarInsW.pack(side=tk.BOTTOM, expand=True, fill=tk.BOTH)
1793     moveUpButton.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)
1794     movedownButton.pack(side=tk.LEFT, expand=True, fill=tk.BOTH)
```

Se colocan los botones para manipular la listbox.

```
1796 # walle
1797 RegarPlanta.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1798 RegarTodo.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1799
1800 # eva
1801
1802 MedirHumedad.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1803 botonBlanco.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1804
```

Se colocan los botones para darles órdenes a los robots.

```
1807 MoverPlanta.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1808 botonCancelar.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1809 AgregarPlanta.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1810 EliminarPlanta.pack(side=tk.TOP, expand=True, fill=tk.BOTH)
1811 framesalidas.pack(side=tk.TOP)
1812 frameTipoPlanta.pack(side=tk.TOP)
1813 frameminimover.pack(side=tk.LEFT)
1814 frameminiagregar.pack(side=tk.LEFT)
1815 cb1.pack(side=tk.LEFT)
1816
1817 frameTabs.pack()
```

Se colocan algunos botones y frames restantes.

```
1819 TablasFrame = ttk.Frame(tab1)
1820 TablasFrame.pack()
1821
1822 frameTabla1 = ttk.Frame(TablasFrame)
1823 lbltabla1 = tk.Label(frameTabla1, text='Tabla piso 1')
1824 lbltabla1.config(fg="black", font=("Verdana", 10))
1825 lbltabla1.pack(side=tk.TOP)
1826
1827 frameTabla2 = ttk.Frame(TablasFrame)
1828 lbltabla2 = tk.Label(frameTabla2, text='Tabla piso 2')
1829 lbltabla2.config(fg="black", font=("Verdana", 10))
1830 lbltabla2.pack(side=tk.TOP)
```

Se crean los frames de las tablas donde van los datos de las plantas, así como un frame que contiene ambos frames, así como unos labels para identificar cual tabla es cual.

```

1832 TablaPiso1 = ttk.Treeview(frameTabla1, selectmode='browse', height=pixel)
1833 verscrlbar = ttk.Scrollbar(frameTabla1, orient='vertical', command=TablaPiso1.yview)
1834 verscrlbar.pack(side=tk.LEFT, fill='y')
1835 TablaPiso1.configure(yscrollcommand=verscrlbar.set)
1836 TablaPiso1['columns'] = ('plant_id', 'plant_humidity', 'expected_humidity', 'type_plant')
1837

```

Se crea una tabla y su scrollbar, también se definen los datos que posee.

```

1838 TablaPiso1.column("#0", width=0, stretch=tk.NO)
1839 TablaPiso1.column("type_plant", anchor=tk.CENTER, width=80)
1840 TablaPiso1.column("plant_id", anchor=tk.CENTER, width=80)
1841 TablaPiso1.column("plant_humidity", anchor=tk.CENTER, width=80)
1842 TablaPiso1.column("expected_humidity", anchor=tk.CENTER, width=80)
1843 TablaPiso1.pack(side=tk.LEFT, fill=tk.BOTH, expand=False)
1844 frameTabla1.pack(side=tk.LEFT)
1845 frameTabla2.pack(side=tk.LEFT)
1846 TablaPiso1.heading("#0", text="", anchor=tk.CENTER)
1847 TablaPiso1.heading("type_plant", text="Tipo Planta", anchor=tk.CENTER)
1848 TablaPiso1.heading("plant_id", text="Planta", anchor=tk.CENTER)
1849 TablaPiso1.heading("plant_humidity", text="Humedad", anchor=tk.CENTER)
1850 TablaPiso1.heading("expected_humidity", text="Humedad esperada", anchor=tk.CENTER)
1851

```

Se configura el posicionamiento de los elementos dentro de la tabla.

```

1852 TablaPiso2 = ttk.Treeview(frameTabla2, height=pixel)
1853 TablaPiso2['columns'] = ('plant_id', 'plant_humidity', 'expected_humidity', 'type_plant')
1854 TablaPiso2.column("#0", width=0, stretch=tk.NO)
1855 TablaPiso2.column("type_plant", anchor=tk.CENTER, width=80)
1856 TablaPiso2.column("plant_id", anchor=tk.CENTER, width=80)
1857 TablaPiso2.column("plant_humidity", anchor=tk.CENTER, width=80)
1858 TablaPiso2.column("expected_humidity", anchor=tk.CENTER, width=80)
1859 TablaPiso2.pack(side=tk.LEFT, fill=tk.BOTH, expand=False)
1860 TablaPiso2.heading("#0", text="", anchor=tk.CENTER)
1861 TablaPiso2.heading("type_plant", text="Tipo Planta", anchor=tk.CENTER)
1862 TablaPiso2.heading("plant_id", text="Planta", anchor=tk.CENTER)
1863 TablaPiso2.heading("plant_humidity", text="Humedad", anchor=tk.CENTER)
1864 TablaPiso2.heading("expected_humidity", text="Humedad esperada", anchor=tk.CENTER)
1865

```

Se hace lo mismo con la tabla del piso 2.

```
1866     for i in range(100):
1867         if i in numbers1:
1868             TablaPiso1.insert(parent='', index='end', iid=i, text='',
1869                             values=(i, "?", humedadesperada[i], tipo[i]))
1870
1871     for i in range(100):
1872         if i in numbers2:
1873             TablaPiso2.insert(parent='', index='end', iid=i, text='',
1874                             values=(i + 100, "?", humedadesperada2[i], tipo2[i]))
1875
```

Se le añaden los datos iniciales a la tabla.

```
1876     monitor2.pack(side=tk.TOP)
1877
1878     framePlantaSelect.pack(side=tk.TOP)
1879     monitor1.pack(side=tk.TOP)
1880     framegeneral.pack(side=tk.TOP)
1881     framegeneral.config(border=10)
1882     frameMovimiento.pack(side=tk.TOP)
1883     frameMovimiento.config(border=10)
1884     frameinformacion.pack(side=tk.TOP)
1885     frameinformacion.config(border=20)
1886
1887     botonABORTAR.pack(side=tk.TOP)
```

Se añaden visualmente los frames faltantes, el monitor que muestra la planta seleccionada y el botón abortar.

```
1889     ventana.mainloop()
1890
```

Finalmente se añade “mainloop”, que hace que el programa este en un loop constante de mantener la ventana.

Desafíos a Futuro

- Combinarlo con Arduino.
- Optimizar el movimiento al mover una planta.
- Registrar la información de las plantas de forma permanente para su estudio.
- Hacer que el robot puede ejecutar una segunda orden sin necesidad de volver a su estación.
- Hacer que el invernadero actué en base al clima.