```
#LIBRERIA WINCAPTUREFUNCTIONS( TODO LO QUE TENGA QUE CAPTURAR O HACER COSAS CON LA APLICACIÓN DEL BUSCAMINAS) CLICKAR, DETECTAR EL TABLERO, REINICIAR EL JUEGO…



# Carga todas la lista de ventanas para ver cual es el handler del buscaminas, el handler se carga de la siguiente función getHwnd()
def loadWindowsList(hwnd,toplist):
def getHwnd(name):

# Devuelve la posición de la ventana del buscaminas, para detectar el tablero
def getPosition(hwnd, extra):

# captura la imagen del juego
def windowCapture(name):


# Carga todos los simbolitos del 1,2… mina…
def loadTemplate(paths):
# Define los paths de las imágenes y llama a load template
def init_template():


# No hay que ser un genio, carga el tablero, se usa la de "loadBoard" la otra es llamada dentro de load board
def loadBoard(name):
def getBoard(template):




# Para saber el estado de la casilla (1,2,mina,..) compara con las template
def boxCheck(image, template):

# Devuelve una matriz con el estado actual del tablero
def obtainMatrix(board, template):


# Hace click con el ratón en las coordenadas que se le diga de la pantalla
def mouse(x, y, action=False):

# Hace click con el ratón en las coordenadas del tablero
def click_board(x, y):

# Como hacemos click en varios lados, por si la ventana del buscaminas no esta visible, se recupera el foco.
def recover_focus(name):


# Pide al usuario coordenadas y las separa en x e y, ahora que va automático no debería usarse.
def ask4cords():

# Inicia un juego
def init_game():
```

```
    # DEFINICION DE LA CLASE CELDA, NO HAY MUCHO QUE EXPLICAR, SE VAN
      DECLARANDO ATRIBUTOS QUE SE HAN IDO NECESITANDO.

class Cell:
    # Attributes
    discovered = False
    state = 9
    heuristic_value = 0
    undiscovered_neighbours = 8

    def __init__(self, discovered=False, state=9, heuristic_value=-1):
        self.discovered = discovered
        self.state = state
        self.heuristic_value = heuristic_value
        return

    def update_state(self, state):
        self.state = state

    def update_nneighbours(self, nneighbours):
        self.undiscovered_neighbours = nneighbours
```

```
# CLASE TABLERO, AQUI ESTA LA CHICHA
class Board:
def __init__(self, rows, columns, solved=False, dead=False):

# Inicia un nuevo juego
def new_game(self):

# Comprueba si has muerto
def check_dead(self):


# En ella se actualiza el estado de todo el tablero, se llama a las
funciones como calcular vecinos no descubiertos, calcular
heurísticos.. Hay que llamarla al menos una vez tras cada acción
def update_board(self):


# Devuelve la matriz con el estado de la matriz de objetos celda que
posee el objeto Board
def obtain_matrix(self):

#Obviamente aquí se calcula la distancia Mahalanobis. Nah se muestra
en forma de matriz por pantalla pues lo que ponga.
def show_board(self):
def show_discovered(self):
def show_undiscovered_neighbours(self):
def show_heuristic(self):

# Si el estado de la casilla es distinto de 9 (casilla no descubierta)
# actualiza discovered a True, para que sea más cómodo
def calc_discovered(self)
```

```
# Cuenta el número de casillas sin descubrir alrededor de cada casilla
y actualiza su parámetro undiscovered_neighbours. (Parece complicado
porque hay que tener en cuenta que algunas casillas no tienen un lado
pero al final son muchos if else que hacen lo mismo
def calc_undiscovered_neighbours(self):

# Esto sigue la misma estructura que la de arriba pero calcula el
heurístico y lo actualiza.
Heuristico actual: suma de probabilidades de ser mina / numero de
casillas que he sumado (me explico fatal.)
def calc_heuristic(self):


#En base al menor valor del heurístico calculo el "Mejor" siguiente
movimiento.
def next_move(self):
```

```
#Esto está aparte, genera tableros ya resueltos por si quisiésemos
entrenar una red neuronal… (El procedimiento es igual que en
calc_heuristic y en cal_undiscovered_neighbours
def generate_board(rows, col, n_mines):
```

EXPLICACION MAIN

```
def main():
    #Se carga el tablero e inicia un juego
    board = wcf.init_game()

    while True:

        #Actualiza el tablero, todos sus valores
        board.update_board()

        # Calcula el mejor moviento y lo hace
        i, j = board.next_move()
        wcf.click_board(j, i)

        # Si pierde reinicia
        if board.check_dead():
            print("LA PALMASTE")

            #Reinicia el juego y pone todo a cero
            board = wcf.init_game()
            board.new_game()



        # PARA PAUSAR PULSAR EL F6
        state = win32api.GetKeyState(117)
        while state:
            state = win32api.GetKeyState(117)
        gc.collect()
```