

Universidad de San Carlos de Guatemala
Facultad de ingeniería
Ingeniería en Ciencias y Sistemas
ESTRUCTURA DE DATOS



MANUAL TECNICO

Alberto Josue Hernández Armas 201903553

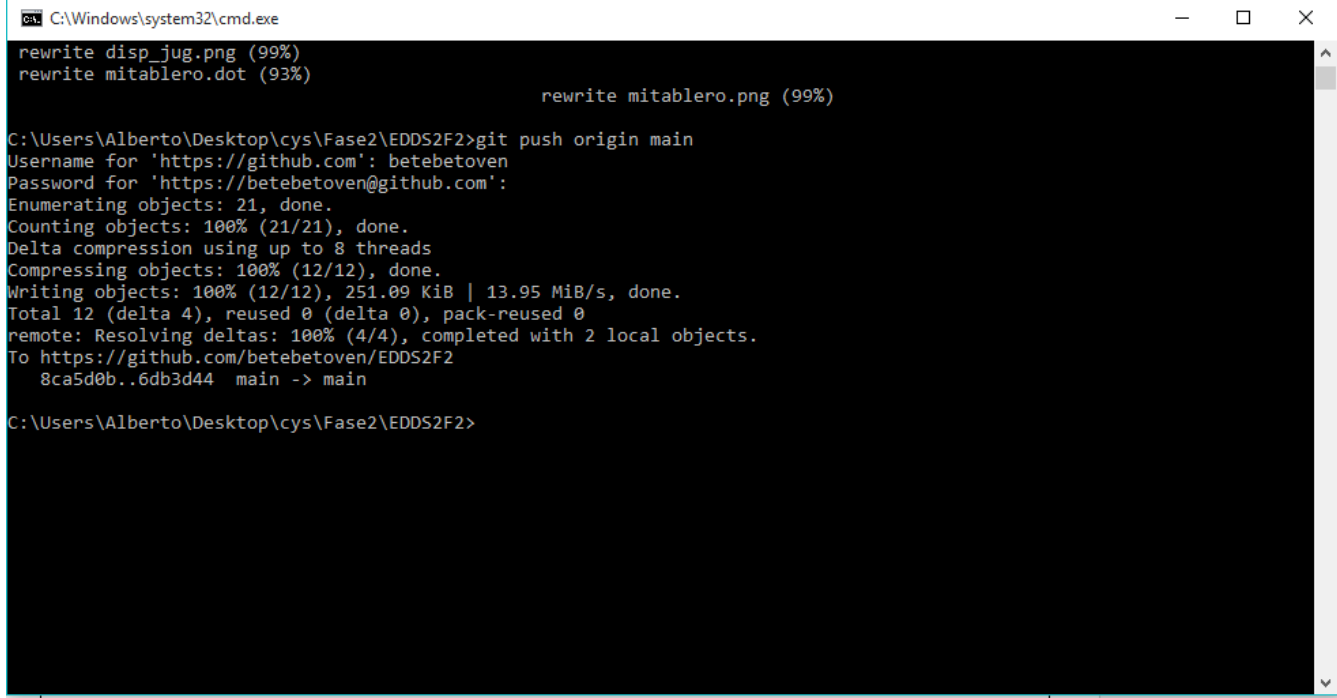
Guatemala 3 de OCTUBRE de
2022

INTRODUCCION

En el presente manual técnico podremos observar cómo es que está diseñado el código y explicando que forman la funcionalidad de una manera general, teniendo como objetivo principal que sea más entendible. Determinando los métodos utilizados y explicando algunas palabras claves refiriéndonos a sus propiedades demostrando cuál es su función dentro de los bloques de código que se nos presentan a continuación.

MANUAL TECNICO

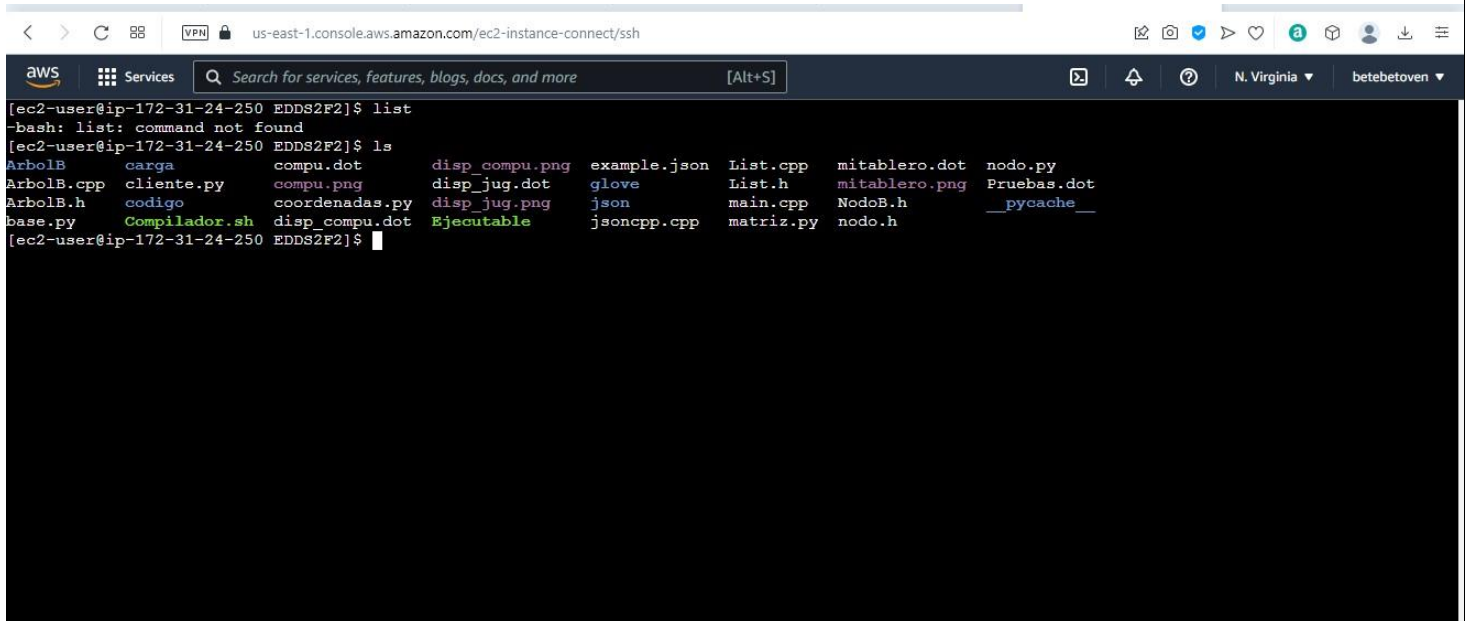
El proyecto se trabajo como una API, de manera que se tiene un servidor que se encarga del manejo de la información, y un cliente que se encarga d ingresarla, recibirla y hasta cierto punto procesarla. En este caso debido a que la REST API era en C++, hacerlo en Windows representaba demasiados problemas y particionar el disco tomaría demasiado tiempo, por lo que se utilizó el servicio de aws, para tener una máquina virtual que sosteniera el servidor en c++,y conectándonos a ella por medio de su IP, para hacer las llamadas desde el cliente.



```
C:\Windows\system32\cmd.exe
rewrite disp_jug.png (99%)
rewrite mitablero.dot (93%)
rewrite mitablero.png (99%)

C:\Users\Alberto\Desktop\cys\Fase2\EDDS2F2>git push origin main
Username for 'https://github.com': betebetoven
Password for 'https://betebetoven@github.com':
Enumerating objects: 21, done.
Counting objects: 100% (21/21), done.
Delta compression using up to 8 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 251.09 KiB | 13.95 MiB/s, done.
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (4/4), completed with 2 local objects.
To https://github.com/betetbetoven/EDDS2F2
  8ca5d0b..6db3d44  main -> main

C:\Users\Alberto\Desktop\cys\Fase2\EDDS2F2>
```

A screenshot of a web browser window displaying the AWS Management Console. The browser's address bar shows the URL 'us-east-1.console.aws.amazon.com/ec2-instance-connect/ssh'. The console interface includes a top navigation bar with the AWS logo, a 'Services' menu, a search bar, and a user profile dropdown. The main content area is a terminal window with a dark background. It shows a command prompt '[ec2-user@ip-172-31-24-250 EDD82F2]\$' followed by the command 'list', which results in an error '-bash: list: command not found'. The user then enters 'ls', which lists various files and directories in a grid-like format. The files include 'ArbolB', 'ArbolB.cpp', 'ArbolB.h', 'base.py', 'carga', 'cliente.py', 'codigo', 'compilador.sh', 'compu.dot', 'compu.png', 'coordenadas.py', 'disp_compu.dot', 'disp_compu.png', 'disp_jug.dot', 'disp_jug.png', 'ejecutable', 'example.json', 'glove', 'json', 'jsoncpp.cpp', 'List.cpp', 'List.h', 'main.cpp', 'matriz.py', 'mitablero.dot', 'mitablero.png', 'nodo.py', 'nodoB.h', 'nodo.h', 'Pruebas.dot', and '__pycache__'. The terminal prompt returns to '[ec2-user@ip-172-31-24-250 EDD82F2]\$'.

Luego de configurar la maquina virtual, se procedio a utilizar la maquina principal como un editor de texto, subiendo el código a github, y haciendo un pull desde la maquina virtual, evitándonos contratar la interfaz grafica de aws y facilitando la modificación de datos. En si aws quedo como un receptor de todos los cambios que se realizaban en github.

Las estructura principales del proyecto se mantienen iguales, ya que el servidor se basa en la fase 2 del proyecto, las estructuras nuevas son El blockchain, lista de adyacencia, y grafos, tabla hash y bendito sea Dios se permitió el uso de Python para esta fase del proyecto.

La tabla hash de implementa de la siguiente manera

La implementación de esta fue que cuando comenzaran a haber colisiones, se cambiara de algoritmo de posicionamiento, además que cuando la tabla hash sobrepasara 80 por ciento de ocupación, esta aumentara su tamaño tal que la ocupación anterior equivalga al 20 por ciento de la ocupación de la nueva tabla.

```

import os

class jacinto():
    def __init__(self):
        self.head = None
        self.last = None
        self.tamaño = 13
        self.ocupacion = 0
        self.porcentaje_de_ocupacion = 0
        for n in range(self.tamaño):
            self.inicia_tamaño()
    def reinicio(self):
        self.head = None
        self.last = None
        self.tamaño = 13
        self.ocupacion = 0
        self.porcentaje_de_ocupacion = 0
        for n in range(self.tamaño):
            self.inicia_tamaño()

    def definir_porcentaje_ocupacion(self):
        self.porcentaje_de_ocupacion = self.ocupacion/self.tamaño
        return self.porcentaje_de_ocupacion

    def inicia_tamaño(self):
        if self.head == None:
            self.head = nodito('')
            self.last = self.head
            self.head.Next = self.last
            self.last.Next = self.head
            self.colisiones = 0
        else:
            k = self.head

            while k.Next != self.head:
                #print(f'acaentra{k}')
                k = k.Next
            k.Next = self.head

```

```

def inicia_tamano(self):
    if self.head == None:
        self.head = nodito('')
        self.last = self.head
        self.head.Next = self.last
        self.last.Next = self.head
        self.colisiones = 0
    else:
        k = self.head

        while k.Next != self.head:
            #print(f'acaentra{k}')
            k = k.Next
        k.Next = nodito('')
        self.last = k.Next
        self.last.Next = self.head

def hasheo(self, cadena):
    general = sum(bytearray(cadena,encoding='utf-8'))
    #print(f'NUMEROHASH____{general}')
    return general

def agregar1(self, id,value):
    posicion = self.hasheo(str(id)+str(value)) % self.tamaño

    k = self.head
    for n in range(posicion):
        k = k.Next

    if k.value == "":
        print(f'SE VA A POSICION____{posicion}')
        k.value = value
        self.ocupacion = self.ocupacion +1
    else:
        print("HUBO OCOLISIOIN")
        f = k
        while f.value != "":

            posicion = posicion + 1

```

```

def retamaño(self):
    while self.definir_porcentaje_ocupacion()*100 > 20.00:
        self.agrega_vacios()
        print(f'PORCENTAJE DE OCUPACION____{str(self.definir_porcentaje_ocupacion()*100)}...%')
    print("retamaño:")
    #print(str(self))
    #self.prettytable(2)
def agrega_vacios(self):
    k = self.head

    while k.Next != self.head:
        #print(f'acaentra{k}')
        k = k.Next
    k.Next = nodito('')
    self.last = k.Next
    self.last.Next = self.head
    self.tamaño = self.tamaño+1
def haseho2(self,numero):
    nuevo_index = ((numero % 3)+1)*self.colisiones
    return nuevo_index

def agrega_inicial(self,id,value):
    posicion = self.haseho(str(id)+str(value)) % self.tamaño
    self.agregar(id,value,posicion)

def agregar(self,id,value,index):
    k = self.head
    for n in range(index):
        k = k.Next
    if k.value == "":
        #print(f'-----')
        print(f' ')
        print(f'[{value}] SE VA A POSICION_{index} que es la posicion {index % self.tamaño}')
        k.value = value

```

La estructura de lista de adyacencia se toma como si fuera una lista de listas por lo cual su implementación es de la siguiente manera

```

from tkinter import messagebox
from nodoll import nodito
import os
class listaenlazada:
    head = None
    tamaño = 0
    general = ""
    def __init__(self):
        self.head = None
        self.tamaño = 0
        self.general = "digraph G\n"+"{label=\"expresion regular\"\n"+"          node[shape = square]\n"+"          node[style = filled]\n"+"          node[fillcolor = \"#EEEEEE\"]\n"
    def agrega(self, x,y):
        if self.head == None:
            self.head = nodito(x)
            self.head.agragaderecha(y)
            return True
        if self.contains(x,y):
            print("si lo agrego")
            return True
    def agrega_simple(self,x):
        if self.head == None:
            self.head = nodito(x)
            self.tamaño = 1
            return True
        k = self.head
        while k.Next != None:
            k = k.Next
        self.tamaño = self.tamaño +1
        k.Next = nodito(x)

```

Y el grafo se implementa de la siguiente manera:

```

def graphvix(self,a):
    self.general = "digraph G\n"+"{label=\"expresion regular\"\n"+"          node[shape = square]\n"+"          node[style = filled]\n"+"          node[fillcolor = \"#EEEEEE\"]\n"
    self.numera()
    self.conexiones()
    self.rank()
    self.general+="\n }"
    messagebox.showerror("entra?",self.general)
    f = open(f'{a}.dot', "w")
    f.write(self.general)
    f.close()
    os.system(f'dot -Tpng {a}.dot -o {a}.png')
    self.general = "digraph G\n"+"{label=\"expresion regular\"\n"+"          node[shape = square]\n"+"          node[style = filled]\n"+"          node[fillcolor = \"#EEEEEE\"]\n"
    aux = self.head
    while aux!=None:
        aux2 = aux
        while aux2.Der != None:
            self.general+=f'\n{str(aux.value)}->{str(aux2.Der.value)}'
            aux2 = aux2.Der
        aux = aux.Next
    self.general+="}"
    fx = open(f'SUB_{a}.dot', "w")
    fx.write(self.general)
    fx.close()
    os.system(f'dot -Tpng SUB_{a}.dot -o SUB_{a}.png')

```

Se toma un header que es parte de una lista principal y este que apunta a varios nodos, en este caso su atributo DER, esto facilita su graficacion.

El árbol Merkle se implementa de la siguiente manera:

```
graph TD
    subgraph "graphviz"
        direction TB
        A["5"]
        B["6 # Hash pairs of items recursively until a single value is obtained"]
        C["7 class MLKjunior():"]
        D["8     def __init__(self):"]
        E["9         self.merkleroot = None"]
        F["10        self.pedro = shasha()"]
        G["11        self.general = \"digraph G\\n\"+\"{label=\\\"EL ARROBA EXPRESA CUANDO SE ITERA CONSIGOMISMO\\\"}\\n\"+\"        node[shape = hexagon]\\n\"+\"        node[style = filled]\\n\"+\""]
        H["12    def merkle(self,hashList):"]
        I["13"]
        J["14        if len(hashList) == 1:"]
        K["15            pe = nodito(hashList[0])"]
        L["16            self.merkleroot = pe"]
        M["17            return pe"]
        N["18        newHashList = ingreso.Head"]
        O["19        # Process pairs. For odd length, the last is skipped"]
        P["20        for i in range(0, (hashList.tamaño)-1, 2):"]
        Q["21            #print(i)"]
        R["22            #print(i+1)"]
        S["23            #print(\"...\")"]
        T["24            newHashList.append(self.hash2(hashList[i], hashList[i+1]))"]
        U["25            self.general+=f'\\n\"{str(hashList[i])[0:7]}\\\" -> \\\"{str(newHashList[-1])[0:7]}\\\"'"]
        V["26            self.general+=f'\\n\"{str(hashList[i+1])[0:7]}\\\" -> \\\"{str(newHashList[-1])[0:7]}\\\"'"]
        W["27        if len(hashList) % 2 == 1: # odd, hash last item twice"]
        X["28            newHashList.append(self.hash2(hashList[-1], hashList[-1]))"]
        Y["29            self.general+=f'\\n\"{str(hashList[-1])[0:7]}\\\" -> \\\"{str(newHashList[-1])[0:7]}\\\"'"]
        Z["30            self.general+=f'\\n\"@{str(hashList[-1])[0:7]}\\\" -> \\\"{str(newHashList[-1])[0:7]}\\\"'"]
        AA["31        return self.merkle(newHashList)"]
        AB["32"]
        AC["33    def hash2(self,a, b):"]
        AD["34        # Reverse inputs before and after hashing"]
        AE["35        # due to big-endian / little-endian nonsense"]
        AF["36        a1 = a"]
        AG["37        b1 = b"]
        AH["38        h = self.pedro.generate_hash(str(a1)+str(b1)).hex()"]
        AI["39        return h"]
        AJ["40    def graphvix(self):"]
        AK["41        self.general+=f'\\n\"{self.merkleroot.value[0:7]}\\\"[fillcolor=\"pink\"] \\n{\"\"}\"'"]
        AL["42        f = open(f'merkle.dot', \"w\")"]
        AM["43"]
    end
```

```
def hash2(self,a, b):
    # Reverse inputs before and after hashing
    # due to big-endian / little-endian nonsense
    a1 = a
    b1 = b
    h = self.pedro.generate_hash(str(a1)+str(b1)).hex()
    return h

def graphvix(self):
    self.general+=f'\\n\"{self.merkleroot.value[0:7]}\\\"[fillcolor=\"pink\"] \\n{\"\"}\"'
    f = open(f'merkle.dot', \"w\")
    f.write(self.general)
    f.close()
    os.system(f'dot -Tpng merkle.dot -o merkle.png')
    self.general = \"digraph G\\n\"+\"{label=\\\"EL ARROBA EXPRESA CUANDO SE ITERA CONSIGOMISMO\\\"}\\n\"+\"        node[shape = hexagon]\\n\"+\"        node[style = filled]\\n\"+\"
```

Obteniendo un nodo en memoria con la información del hash resultante.

Un bloque de blockchain se implementa de la siguiente manera:

Puede observarse que en este mismo bloque se implementa la prueba de trabajo, ya que se instancia al crearse, pero se termina de instanciar hasta que cumple con la prueba de trabajo, no tiene la opción de no cumplirla, ya que la información que toma de las transacciones es necesaria ingresarla al blockchain.

Se implementa como una lista enlazada unidireccional, lo que la hace inmutable es su dependencia en el sha que se genera del árbol merkle y también del sha que se genera de la información del bloque.

```

9 class bloque:
10     def __init__(self, index, transactions, prev, root):
11         self.next = None
12         self.index = index
13         self.hash = None
14         self.transactions = transactions
15         self.timestamp = datetime.now().strftime("%d-%m-%Y:%H:%M:%S")
16         self.prev = prev
17         self.nonce = 0
18         self.rootmerkle = root
19
20     def work_hash(self, proof):
21         pedro = shasha()
22         #SHA256(INDEX+TIMESTAMP+PREVIOUSHASH+ROOTMERKLE+NONCE)
23         sha = ""
24         while True:
25             sha = pedro.generate_hash(f'{str(self.index)}{str(self.timestamp)}{str(self.prev)}{str(self.rootmerkle.value)}{str(self.nonce)}').hex()
26             #print(sha)
27             if str(sha).startswith(proof):
28                 self.hash = str(sha)
29                 return str(sha)
30                 break
31             self.nonce = self.nonce + 1
32
33     #sera que si funciona sin login
34     #ENTRA UNA LISTA ENLAZADA PARA LAS TRANSACCIONES QUE VIENE DE LA LISTA HASH. ALAAAAAAAAAAAAAAAAAAAAA/0 PUEDE SE QUE ENTRE LA LISTA HASH. UNA DE DO

```

150 lines (116 sloc) 5.23 KB

Raw Blame



```
1 from LL import listaenlazada
2 from merkletrie import MLKjunior
3 from bloqueBC import bloque
4 import os
5 import json
6 from sha256 import shasha
7 from tkinter import messagebox
8
9 class blockchain:
10     def __init__(self):
11         self.Head = None
12         self.actual = None
13         self.index = 0
14         self.primerizo = ""
15     def queputas(self,alv,puta):
16         print("que putas"+alv+puta)
17
18
19     def agrega_alv(self,trans,raiz,opcional):
20         if self.index == 0:
21             nuevo = bloque(self.index,trans,"-1",raiz)
22             nuevo.work_hash('00')
23             self.Head = nuevo
24             self.actual = self.Head
25             self.index = self.index +1
26             print(nuevo)
27             return True
28         elif self.index != 0 and self.Head!= None:
29             nuevo = bloque(self.index,trans,str(self.actual.hash),raiz)
30             nuevo.work_hash('00')
31             self.actual.next = nuevo
32             self.actual = nuevo
```

```

33         return True
34
35 def graphviz(self):
36     pepe = listaenlazada()
37     general = "digraph G\n"+"{label=\\"expresion regular\\"\\n"+"      node[shape = hexagon]\\n"+"      node[style = filled]\\n"+"      node[fillcolor = \\"#EEEE\\"]\\n"+"
38     dir_path = r'jsons\\'
39     res = os.listdir(dir_path)
40     if len(res)!=0:
41         for n in res:
42             with open(f'jsons/{n}') as json_file:
43                 data = json.load(json_file)
44                 prefdata = f'\\index:  {data["index"]}\\ntimestamp: {str(data["timestamp"])}\\nnonce:  {str(data["nonce"])}\\nhash:  {str(data["data"])}\\nself_hash"')\\nhash
45                 pepe.agrega_simple(prefdata)
46
47     k = pepe.head
48     while k.Next != None:
49         general+=f'\\n{k.value}->{k.Next.value}'
50         k=k.Next
51
52 #####
53 for n in pepe:
```

A continuación se presenta la implementación de la matriz dispersa, esta se realizó en el lenguaje python. Primero tenemos el nodo que guardará toda la información que el brindemos, así como en c++ se utilizaron las estructuras en modo template para poder reutilizarlas en el contexto que nosotros queramos, python nos permite trabajar las variables con un tipo de dato dinámico, por lo que no permite seguir en la misma línea de pensamiento.

```

15 lines (13 sloc) | 331 Bytes

1  from coordenadas import coordenadas
2
3
4  class nodo:
5      arriba= None
6      abajo = None
7      derecha = None
8      izquierda = None
9      c = None
10     barco = ""
11     def __init__(self, barco,x,y):
12         self.barco = barco
13         self.c = coordenadas(x,y)
14     def __str__(self):
15         return '{c:[' + str(self.c) + '],b:' + self.barco+'}'

```

La ventaja de python es que puede darle una representación string directa a los objetos que creamos, a diferencia de c++.

Implementación de la matriz:

```

1  from tkinter.messagebox import NO
2  from nodo import nodo
3  import random
4  import pyperclip
5  import os
6  class par:
7      x = 0
8      y=0
9      def __init__(self,x,y):
10         self.x = x
11         self.y = y
12     def __str__(self):
13         return '[x:' + str(self.x) + ',y:' + str(self.y)+']'
14
15 class matriz:
16     raiz = nodo("root", -1,-1)
17     dx = 0
18     dy = 0
19     ocupados = []
20     general = "digraph G\n"+"{label=\"expresion regular\"\n"+"          node[shape = circle]\n"+"          node[style = filled]\n"+"          node[fillcolor = \"#EEEEEE\"]\n"
21     espacios = {
22         "pt":4,
23         "sub":3,
24         "dt":2,
25         "b":1
26     }

```

La matriz es una organizacion en la cual se trabajan con nodos ejes, que ayudan a localizar la información por su coordenada, sin instanciar los espacios vacios para evitar un gasto innecesario de la memoria

```
def __init__(self,t ):
    self.dx = t
    self.dy = t
    self.raiz = nodo("root", -1,-1)
    self.ocupados = []
    self.creatodo()

def recursivx(self,rooot, cont, meta):
    if cont == meta:
        print(str(rooot))
        return
    else:
        rooot.derecha = nodo("ejex",cont,-1)
        rooot.derecha.izquierda = rooot
        cont = cont+1
        print(str(rooot))
        self.recursivx(rooot.derecha,cont,meta)

def recursivy(self,rooot, cont, meta):
    if cont == meta:
        print(str(rooot))
        return
    else:
        rooot.abajo = nodo("ejey",-1,cont)
        rooot.abajo.arriba = rooot
        cont = cont +1
        print(str(rooot))
        self.recursivy(rooot.abajo,cont,meta )

def creatodo(self):
    self.recursivx(self.raiz,0,self.dx)
    self.recursivy(self.raiz,0,self.dy)
```

El método de icializacion de encarga de crear los ejes para que la información sea ingresaa de una manera más fácil , el tamaño de los ejes puede ser dinamico si se lo desea, pero en este caso no es asi.

El ingreso a la matriz es por medio de coordenadas, además de que ingresa automáticamente los barcos.

```
65     def ingresar(self,x,y,barco):
66         for n in self.ocupados:
67             if(n.x == x and n.y == y):
68                 return False
69         if(x>=self.dx or y >= self.dy or x<0 or y < 0):
70             return False
71         nuevo_nodo = nodo(barco,x,y)
72         print("ingresando: "+str(nuevo_nodo))
73         ahora = self.raiz
74         while(ahora.c.x != x):
75
76             ahora = ahora.derecha
77
78         while(ahora != None):
79
80             if(ahora.abajo == None and ahora.c.y < y):
81                 ahora.abajo = nuevo_nodo
82                 ahora.abajo.arriba = ahora
83
84             elif(ahora.abajo!= None and ahora.abajo.c.y >y and ahora.c.y < y):
85                 aux = ahora.abajo
86                 ahora.abajo = nuevo_nodo
87                 ahora.abajo.arriba = ahora
88                 ahora.abajo.abajo = aux
89                 ahora.abajo.abajo.arriba = ahora.abajo
90             ahora = ahora.abajo
91         ahora = self.raiz
92         #ahora toca de lado de y para ingresar en x
93         while(ahora.c.y != y):
94
95             ahora = ahora.abajo
96
97         while(ahora != None):
98             if(ahora.derecha == None and ahora.c.x < x):
99                 ahora.derecha = nuevo_nodo
```

También se pueden eliminar nodos sin afectar su entorno:

```

111         ahora = ahora.derecha
112     def eliminar(self,x,y):
113         if(x>self.dx or y> self.dy):
114             return False          #SOLO SE UTILIZAN DOS CASOS EN LA ELIMINACION YA QUE DE FIJO
115         bandera = False          #SIEMPRE VA A TENER NODO A LA IZQUIERDA Y ARRIBA, POR LO QUE PUEDE FACTORIZARSE
116         for n in self.ocupados:#A UN CASO GENERAL
117             if(n.x == x and n.y == y):
118                 bandera = True
119         if bandera == True:
120             ahora = self.raiz
121             while(ahora.c.x != x):
122                 ahora = ahora.derecha
123             while(ahora.c.y != y):
124                 ahora = ahora.abajo
125             ahora.arriba.abajo = ahora.abajo
126             if(ahora.abajo != None):
127                 ahora.abajo.arriba = ahora.arriba
128             ahora.izquierda.derecha = ahora.derecha
129             if(ahora.derecha != None):
130                 ahora.derecha.izquierda = ahora.izquierda
131             ahora= None
132         for n in self.ocupados:
133             if(n.x == x and n.y == y):
134                 self.ocupados.remove(n)
135         return bandera
136

```

El resto de métodos es para la realización del videojuego.

Luego tenemos la creación, instancia e implementación de la interfaz grafica, que es donde se llevara a cabo todas las implementaciones y conexiones con el servidor.

```
15 from matriz import matriz
16
17
18 from PIL import ImageTk, Image
19
20 tablero_jugador_global = None
21 tablero_computadora_global = None
22 tablero_disparos_computadora_global = None
23 tablero_disparos_jugador_global = None
24 direccion = "one"
25 base_url = "http://3.88.228.81:8080/"
26 def entrada():
27     global direccion
28     direccion = askopenfilename()
29     f = open(direccion, "r")
30     return f.read()
31 def carga_masiva(entrada):
32     res = requests.post(f'{base_url}/Lista/{entrada}')
33     data = res.text#convertimos la respuesta en dict
34
35     f = open(f'arbolb.dot', "w")
36     f.write(data)
37     f.close()
38     os.system(f'dot -Tpng arbolb.dot -o arbolb.png')
39
40
41     ver5()
42     print(data)
43 def login(usuario, contraseña):
44     res = requests.post(f'{base_url}/Login/{usuario},{contraseña}')
45     data = res.text#convertimos la respuesta en dict
46     messagebox.showinfo("LOGIN",data)
47     print(data)
48 def editN(nombre):
49     res = requests.post(f'{base_url}/editN/{nombre}')
```


Llamada a creación de blockchain

```
4
5 def display(msg):
6     global BLOCKCHAIN_GLOBAL
7     global TRANSACCIONES_GLOBALES
8     global MERKLE_ROOT_GLOBAL
9     global PREV_DEL_JSON
10    print(msg + ' ' + time.strftime('%H:%M:%S'))
11    if MERKLE_ROOT_GLOBAL != None:
12        BLOCKCHAIN_GLOBAL.agrega_alv(TRANSACCIONES_GLOBALES, MERKLE_ROOT_GLOBAL, PREV_DEL_JSON)
13        time.sleep(5)
14        BLOCKCHAIN_GLOBAL.graphviz()
15
```

```
skis = ""
pedro = shasha()
with open(f'jsons/{n}') as json_file:
    data = json.load(json_file)
    print('\n\n_____')
    print(data)
    merklito = MLKjunior()
    TRANSACCIONES_GLOBALES.head = None
    BLOCKCHAIN_GLOBAL.index = int(data["index"])+1
    PREV_DEL_JSON = str(data["data"]['self_hash'])
    print(data["data"]['transacciones'])
    for n in data["data"]['transacciones']:

        for k in n['skins']:
            HASHTABLE_GLOBAL.agrega_inicial("0", skin(str(k), 0))
            skis+=k
        TRANSACCIONES_GLOBALES.agrega_simple(trans(n["from"], HASHTABLE_GLOBAL.toArray(), HASHTABLE_GLOBAL.total()))
        HASHTABLE_GLOBAL.reinicio()
        shas.append(str(pedro.generate_hash(f'{str(n["from"])}{str(skis)}').hex()))
        print(f'{str(n["from"])}{str(skis)}')
        skis = ""
        construye_MLK()
    nodito = merklito.merkle(shas)
    print(nodito.value)
    #SHA256(INDEX+TIMESTAMP+PREVIOUSHASH+ROOTMERKLE+NONCE
    selfhash = str(pedro.generate_hash(f'{data["index"]}{data["timestamp"]}{data["data"]["hash_prev"]}{nodito.value}{data["nonce"]}').hex())
    #AHORA TIENE QUE AGRAGAR TODO OTRA VEZ

    if(data["data"]["merkle_root"]==str(nodito.value) and selfhash == data["data"]["self_hash"]):
        messagebox.showinfo("agregado", f'INFORMACION EN INDEX {cont} INTACTA')
```

Reingreso de datos de blockchain.

Y todo esto se repite dentro de la función timer, que permite que corra en segundo plano mientras el jugador esta jugando.

```
##Put it into a class
class RepeatTimer(th.Timer):
    def run(self):
        while not self.finished.wait(self.interval):
            self.function(*self.args,**self.kwargs)
            print(' ')
##We are now creating a thread timer and controlling it
timer = RepeatTimer(60,display,['Repeating'])
timer.start() #recalling run
print('Threading started')
mainloop()
time.sleep(10)#It gets suspended for the given number of seconds
print('Threading finishing')
timer.cancel()
```

Fin del funcionamiento del programa.