blocksUD

July 22, 2019

Clase Transacción

In [1]:
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Define los elementos que componen una transacción.
"""

class Transaccion(object):

    def __init__(self, emisor, receptor, cantidad):
        """
        Creación de una transaccióna
        :param emisor: Hash de quién envía
        :param receptor:  Hash de quién recibe
        :param cantidad: valor de la transacción
        """
        self._emisor = emisor
        self._receptor = receptor
        self._cantidad = cantidad

    def getEmisor(self):
        """
        Obtiene el valor actual del hash del emisor
        :return: Hash del emisor
        """
        return self._emisor;

    def getReceptor(self):
        """
        Obtiene el valor actual del hash del receptor
        :return: Hash del receptor
        """
        return self._receptor;

    def getCantidad(self):
        """
        Obtiene el valor de la transacción
```

```
            :return: Valor de la transacción
            """
            return self._cantidad;

        def __str__(self):
            """
            Genera un string con la información de la transacción
            :return: representación en String de una transacción
            """
            return str(self._emisor) + " " + str(self._receptor) +  " " + str(self._cantida
```

Clase Bloque

```
In [3]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        import time
        import hashlib
        import json

        """
        Define la estructura del bloque.
        """

        class Bloque(object):

            def __init__(self, indice, minado, hashBloque, transacciones, marcaTiempo=None):
                """
                Inicialización del bloque
                :param indice: Valor entero que representa la posición de cada bloque en la ca
                :param minado: Número entero usado en el proceso de minería
                :param hashBloque:  hash del bloque actual en la cadena
                :param transacciones: Conjunto de transacciones del bloque
                :param marcaTiempo: Instante de tiempo de creación del nuevo bloque
                """
                self._indice = indice
                self._minado = minado
                self._hashBloque = hashBloque
                self._transacciones = transacciones.copy()
                self._marcaTiempo = marcaTiempo or time.time()

            def gethashBloque(self):
                """
                Obtiene el valor actual del hash del bloque
                :return: Hash del bloque
                """
                return self._hashBloque;

            def setHashBloque(self):
```

```
        self._hashBloque = Bloque.generarBloqueHash(self)

    @staticmethod
    def generarBloqueHash(self):
        """
        Genera Hash la información del bloque
        :return: Hash construido a partir de la información del bloque
        """
        return hashlib.sha256(str(self).encode()).hexdigest()


    def __str__(self):
        """
        Genera un string con la información del bloque
        :return: representación en String del bloque
        """
        return "{} , {} , {}  , {}".format(self._indice, self._minado, self._hashBloque

    def __repr__(self):
            return json.JSONEncoder().encode(self)

    def consultarBloque(self):
        a = ""
        a = a + str(self._indice) + " " + str(self._minado) + " " + str(self._hashBloqu
        a = a + "Transacciones del bloque";
        for i in range(len(self._transacciones)):
            a = a + "\n" + str(self._transacciones[i]);
        a = a + "\nFin Transacciones del bloque";
        return a;
```

Clase Cadena de Bloques

```
In [ ]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        """
        En esta clase almacena la cadena de bloques
        """

        from cadena.Bloque import Bloque
        from cadena.PoolMinado import PoolMinado
        import json
        import copy

        class CadenaBloques(object):

            def __init__(self):
                """
                Inicialización de la blockchain
```

```
    :param _cadena: Estructura de datos tipo array para almacenar la cadena de blo
    """
    self._cadena = []


def getCadenaSerializada(self):
    """
    Serializa la cadena de bloques en un documento JSON
    :return conjunto de bloques en formato JSON
    """
    salida = ""
    for i in range(len(self._cadena)):
        salida = salida +"\nInicio de Bloque\n"
        salida = salida + str(self._cadena[i].consultarBloque());
        salida = salida + "\nFin Bloque " + str(i) + str(". \n");
    return salida

def bloqueGenesis(self,trs):
    """
    Crea el primer bloque en la cadena. Es el bloque seminal.
    """
    self.nuevoBloque(2, Bloque.generarBloqueHash(0), trs.copy())

def nuevoBloque(self, minado, hashBloque,trans):
    """
    Construye e inserta el siguiente bloque de la cadena
    :param minado Método de minado del bloque
    :param hashBloque Hash generado para el bloque
    :return Bloque nuevo bloque generado
    """
    bloque = Bloque(
        indice = len(self._cadena),
        minado = minado,
        hashBloque = hashBloque,
        transacciones=trans.copy()
    )
    self._cadena.append(bloque)
    return bloque


def obtenerUltimoBloque(self):
    """
    Devuelve una referencia al último nodo de la blockchain
    @return último nodo
    """
    return self._cadena[-1]

@staticmethod
```

```
def bloqueEsValido(nuevoBloque, ultimoBloque):
    """
    Verifica bajo cuatro reglas básicas la validez de un Bloque
    Regla 1: Secuencia ordenada de bloques por valor del índice
    Regla 2: Secuencia ordenada de bloques por fecha de creación
    Regla 3: Hash diferente entre el último bloque de la cadena y el nuevo bloque
    Regla 4: Bloque correctamente minado
    @return True si el bloque es válido, False en otro caso
    """
    if ultimoBloque._indice + 1 != nuevoBloque._indice:
        return False
    elif nuevoBloque._marcaTiempo <= ultimoBloque._marcaTiempo:
        return False
    elif ultimoBloque.gethashBloque != nuevoBloque.gethashBloque:
        return False
    elif not CadenaBloques.esPrimo(nuevoBloque.minado + ultimoBloque.minado):
        return False
    return True



def validarBlockchain(self):
    """
    Verifica la integridad de la cadena de bloques.
    para este procedimiento se valida que el índice del siguiente bloque sea
    menor en una unidad del bloque anterior y que el hash de cada bloque sea
    correcto según la política de minado establecida, números primos en este caso.
    @return True si la cadena esta bien formada, False en otro caso
    """
    bloqueAnterior = self._cadena[0]
    for i in range (1,len(self._cadena)):
        bloqueActual = self._cadena[i]
        if (bloqueAnterior._indice + 1) - bloqueActual._indice != 0:
            print("indice malo")
            return False
        if not PoolMinado.esPrimo(bloqueAnterior._minado + bloqueActual._minado):
            return False
        bloqueAnterior = copy.copy(bloqueActual)

    return True



def __repr__(self):
    return json.JSONEncoder().encode(self._cadena)

def __str__(self):
    """
```

```
        Genera un string con la información de la cadena
        :return: representación en String del nodo
        """
        a = "";
        for i in range(len(self._cadena)):
            a += str(self._cadena[i]) + "\n";
        return a;
```

Clase Nodo Minero

```
In [ ]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        """
        Define la estructura de un nodo responsable de minar la cadena de bloques
        """
        import time
        import copy

        class NodoMinador(object):

            def __init__(self, direccion, puerto, descripcion, usuariosMineros):
                """
                Creación de un nodo para la minería de blockchain
                :param _direcccción: Dirección ip del nodo minero
                :param _puerto:  puerto TCP/UDP donde se publica el servicio
                :param _descripcion: información adicional del nodo minador
                :param _fechaCreacion: fecha de adición del nodo a la red
                :param _cadenaBloques: Recibe una copia de la cadena de bloques
                :param _hashRateNodo : Capacidad de resolución de hash por segundo del nodo
                :param _usuariosMineros: Conjunto de usuarios participantes en el nodo minero
                """
                self._direccion = direccion;
                self._puerto = puerto;
                self._descripcion = descripcion;
                self._utilidad = 0;
                self._cantidadBloquesMinados = 0;
                self._fechaCreacion = time.time();
                self._hashRateNodo = 100;
                self._cadenaBloques = [];
                self._usuariosMineros = copy.copy(usuariosMineros);


            def actualizarCadena(self, nuevaCadena):
                """
                El proceso de minería requiere que el nodo actualice la blockchain
                :param: Cadena de bloques actualizada
                """
                self._cadenaBloques = copy.copy(nuevaCadena)
```

```
    def infoCadena(self):
        """
        Genera un string con la información básica de la cadena de bloques
        :return: una cadena con el reporte del total de nodos de la cadena
        """
        return "La cadena tiene {} bloques".format(str(len(self._cadenaBloques)))

    def __str__(self):
        """
        Genera un string con la información de un nodo de minería
        :return: representación en String del nodo
        """
        return "{} , {} , {} , {} , {}".format(self._descripcion , str(self._direccion
```

Clase Pool de Minería

```
In [ ]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        """
        Define la organización para los nodos que van a minar la cadena.
        """
        from random import random
        from multiprocessing.pool import ThreadPool
        import numpy as np
        from random import randint


        class PoolMinado(object):

            def __init__(self):
                """
                Inicialización del pool de minería
                :param _nodos: conjunto de nodos mineros
                :param _hasRate Simula la capacidad de computo del pool de mineria
                :param _recompensaPorBloque cantidad que se recompensa a los nodos que resuelv
                :param _dificultad  valor usado para simular el incremento de dificultad del p
                """
                self._nodos =  [];
                self._hashRate = 0;
                self._recompensaPorBloque = 10;
                self._dificultad = 1000;

            def adicionarMinero(self, minador):
                """
                Agrega  un nodo minero al conjunto de nodos minadores de la blockchain. Si el
                el hashrate del pool ylo iguala o supera el 60% de la dificultad, esta última
```

```
        20%
        @param minador: Nuevo nodo minero
        @return True si el nodo de mineria se puede adicionar, False en otro caso
        """
        self._nodos.append(minador)
        self._hashRate += minador._hashRateNodo
        if self._dificultad * 0.6 < self._hashRate:
            self._dificultad *= 1.2;

        return True

    @staticmethod
    def pruebaDeTrabajo(indiceUltimoMinado):
        """
        Algoritmo de minado por prueba de trabajo.
        En esta versión se implementa una prueba basada en números primos, en la cual
        valor de minado del bloque actual y del último bloque de la cadena debe ser un
        @param indiceUltimoMinado, Valor generado por la minería para el último bloque
        @return minadoNuevo Valor de minado obtenido para el nuevo bloque
        """
        minadoNuevo = indiceUltimoMinado + 1
        while not PoolMinado.esPrimo(minadoNuevo + indiceUltimoMinado):
            minadoNuevo += 1

        return minadoNuevo

    @staticmethod
    def esPrimo(numero):
        """
        Determina si un número es primo
        @param numero, Número que necesitamos determinar si es primo
        @return True si numero es primo, False en otro caso
        """
        if numero < 2:
            return False
        elif numero == 2:
            return True
        else:
            for x in range(2,numero):
                if(numero % x==0):
                    return False
        return True

    def minarBloque(self,nodoMinero, numeroNodo, output):
        """
        Algoritmo que realiza el minado de un bloque para un nodo en especifico.
        @param nodoMinero, Nodo minero que realizará la prueba de trabajo
        @param numeroNodo, identificador entero del nodo minero en el pool de minería
```

```
        @param output, Parámetro de salida donde se conserva la información de
        la prueba de trabajo ejecutada por el iésimo nodo minero.
        @return minadoNuevo Valor de minado obtenido para el nuevo bloque
        """
        trabajo = 0
        for i in range(self._dificultad - nodoMinero._hashRateNodo + randint(1,10000)):
            trabajo += 1
        ultimo = nodoMinero._cadenaBloques.obtenerUltimoBloque()
        nuevoValorMinado = PoolMinado.pruebaDeTrabajo(ultimo._minado)
        #print("[" + nodoMinero._descripcion + "] ha trabajado " + str(trabajo) + " nu
        output[numeroNodo][0] = numeroNodo;
        output[numeroNodo][1] = nuevoValorMinado;
        output[numeroNodo][2] = trabajo;


    def torneoPorNuevoBloque(self,transacciones):
        """
        Realiza un torneo para la adjudicación del nuevo bloque a un nodo minero.
        El nodo ganador será el que realice mayor trabajo.
        @param transacciones, Bloque de transacciones que tendrá el nuevo bloque de la
        """
        try:
            #print("Total nodos " + str(len(self._nodos)))
            numOfThreads = int(len(self._nodos))
            pool = ThreadPool(numOfThreads)
            output = [];
            for x in range(numOfThreads):
                output.append([0]*3)

            for i in range(numOfThreads):
                pool.apply_async(self.minarBloque(self._nodos[i],i,output))
            pool.close()
            pool.join()
            utilidadObtenida = np.max(output);
            ganador = np.argmax(output)//(numOfThreads-1)+randint(0,1);
            #print("utilidad obtenida " + str(utilidadObtenida))
            self._nodos[ganador]._cantidadBloquesMinados += 1;
            self._nodos[ganador]._utilidad += utilidadObtenida;
            self.minadoBloque(self._nodos[ganador],transacciones);
            nueva = self._nodos[ganador]._cadenaBloques;
            self.sincronizarCadena(nueva);

            totalMinadores = len(self._nodos[ganador]._usuariosMineros);
            for key in self._nodos[ganador]._usuariosMineros:
                self._nodos[ganador]._usuariosMineros[key]._saldo += utilidadObtenida/t
                #print(self._nodos[ganador]._usuariosMineros[key]._saldo);
            #print("=====================================================")
```

```
            except:
                #print ("Error: un hilo ha fallado. ",sys.exc_info())
                #print("-");
                errorhilo = True;
        @staticmethod
        def minadoBloque(nodoMinero, transacciones):
            """
            Realiza el proceso de minado de un nuevo bloque en la cadena
            @param nodoMinero que realiza el intento de minar el bloque
            @param transacciones: Conjunto de Transacciones del nuevo bloque
            """
            ultimo = nodoMinero._cadenaBloques.obtenerUltimoBloque();
            nodoMinero._cadenaBloques.nuevoBloque(PoolMinado.pruebaDeTrabajo(ultimo._minado

        def sincronizarCadena(self,nuevaCadena):
            """
            Realiza una sincronización de la cadena actualizada a todos los mineros
            @param nuevaCadena, Cadena con el nuevo bloque
            """
            for i in range(len(self._nodos)):
                self._nodos[i].actualizarCadena(nuevaCadena);

        def __str__(self):
            """
            Genera un string con la información del pool de minería
            :return: representación en String del nodo
            """
            a = "";
            for i in range(len(self._nodos)):
                a += str(self._nodos[i]) + "\n";
            return a;
```

Clase Usuario

```
In [ ]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-
        import time
        import hashlib
        from cadena.Transaccion import Transaccion
        """
        .. module:: blockChainTest
        """


        class Usuario(object):

            def __init__(self, idUsuario):
                """
```

```
        Creación de un usuario en el sistema
        :param _idUsuario Identificador entero del usuario
        :param hashUsuario:  Código Hash del usuario
        :param saldo: Saldo del usuario
        :param marcaTiempo: Instante de tiempo de creación del usuario
        """
        self._idUsuario  = idUsuario;
        self._saldo = 0;
        self._hashUsuario = 0;
        self._marcaTiempo = time.time();
        """
        Usuario Génesis toma 100 de saldo
        """
        if self._idUsuario == 0:
            self._saldo = 100;


    def gethashUsuario(self):
        """
        Obtiene el valor actual del hash del usuario
        :return: Hash del bloque
        """
        return self._hashUsuario;

    def setHashUsuario(self):
        self._hashUsuario = self.generarUsuarioHash();


    def generarUsuarioHash(self):
        """
        Genera Hash la información del bloque
        :return: Hash construido a partir de la información del usuario
        """
        return hashlib.sha256(str(self).encode()).hexdigest()


    def __str__(self):
        """
        Genera un string con la información del usuario
        :return: representación en String del usuario
        """
        return "{}, {} , {}".format(self._idUsuario, self._hashUsuario, self._saldo)

    def enviar(self, valor, receptor):
        """
        :param receptor: hash del usuario receptor de la transferencia
        :param valor: Cantidad que se desea transferir
        :return: La nueva transacción que será enviada a validación
```

```
            """
            if valor < self._saldo:
                self._saldo -= valor;
                self.recibir(valor, receptor);
                return Transaccion(self._hashUsuario, receptor._hashUsuario,valor);

        def recibir(self, valor, receptor):
            """
            :param receptor: hash del usuario receptor de la transferencia
            :param valor: Cantidad que se ha recibido
            """
            receptor._saldo += valor;
```

Clase Interfaz Gráfica del simulador

```
In [ ]: # -*- coding: utf-8 -*-

        # Form implementation generated from reading ui file 'blockchainsimulator.ui'
        #
        # Created by: PyQt5 UI code generator 5.9.2
        #
        # WARNING! All changes made in this file will be lost!
        import sys
        import os
        from simulador.Simulador import Simulador
        from datetime import datetime
        from PyQt5.QtWidgets import QMessageBox
        from PyQt5 import QtCore, QtGui, QtWidgets
        from PyQt5.QtWidgets import (QApplication, QCheckBox,QAction, QGridLayout, QGroupBox,
                                     QMenu, QPushButton, QRadioButton, QVBoxLayout, QWidget, QS

        class Ui_SimuladorBlockchain(object):
            def setupUi(self, SimuladorBlockchain):
                SimuladorBlockchain.setObjectName("SimuladorBlockchain")
                SimuladorBlockchain.resize(633, 502)
                SimuladorBlockchain.setMouseTracking(True)
                SimuladorBlockchain.setTabletTracking(True)
                self.centralwidget = QtWidgets.QWidget(SimuladorBlockchain)
                self.centralwidget.setObjectName("centralwidget")
                self.labelTitulo = QtWidgets.QLabel(self.centralwidget)
                self.labelTitulo.setGeometry(QtCore.QRect(10, 10, 621, 61))
                self.labelTitulo.setObjectName("labelTitulo")

                self.hsCantidadUsuarios = QtWidgets.QSlider(self.centralwidget)
                self.hsCantidadUsuarios.setGeometry(QtCore.QRect(310, 80, 311, 20))
                self.hsCantidadUsuarios.setMinimum(100)
                self.hsCantidadUsuarios.setMaximum(500)
                self.hsCantidadUsuarios.setPageStep(5)
```

```
        self.hsCantidadUsuarios.setProperty("value", 250)
        self.hsCantidadUsuarios.setOrientation(QtCore.Qt.Horizontal)
        self.hsCantidadUsuarios.setObjectName("hsCantidadUsuarios")
        self.hsCantidadUsuarios.setTickPosition(QSlider.TicksBothSides)
        self.hsCantidadUsuarios.setTickInterval(10)

        self.labelCantidadUsuarios = QtWidgets.QLabel(self.centralwidget)
        self.labelCantidadUsuarios.setGeometry(QtCore.QRect(20, 80, 271, 16))
        self.labelCantidadUsuarios.setObjectName("labelCantidadUsuarios")

        self.labelNodosMineros = QtWidgets.QLabel(self.centralwidget)
        self.labelNodosMineros.setGeometry(QtCore.QRect(20, 130, 271, 16))
        self.labelNodosMineros.setObjectName("labelNodosMineros")
        self.hsPoolMineria = QtWidgets.QSlider(self.centralwidget)
        self.hsPoolMineria.setGeometry(QtCore.QRect(310, 130, 311, 20))
        self.hsPoolMineria.setMinimum(2)
        self.hsPoolMineria.setMaximum(5)
        self.hsPoolMineria.setSingleStep(1)
        self.hsPoolMineria.setPageStep(5)
        self.hsPoolMineria.setProperty("value", 3)
        self.hsPoolMineria.setOrientation(QtCore.Qt.Horizontal)
        self.hsPoolMineria.setObjectName("hsPoolMineria")
        self.hsPoolMineria.setTickPosition(QSlider.TicksBothSides)
        self.hsPoolMineria.setTickInterval(1)

        self.labelmaxtrans = QtWidgets.QLabel(self.centralwidget)
        self.labelmaxtrans.setGeometry(QtCore.QRect(20, 170, 271, 51))
        self.labelmaxtrans.setWordWrap(True)
        self.labelmaxtrans.setObjectName("labelmaxtrans")
        self.hsTransaccionesBloque = QtWidgets.QSlider(self.centralwidget)
        self.hsTransaccionesBloque.setGeometry(QtCore.QRect(310, 190, 311, 20))
        self.hsTransaccionesBloque.setMinimum(10)
        self.hsTransaccionesBloque.setMaximum(30)
        self.hsTransaccionesBloque.setPageStep(5)
        self.hsTransaccionesBloque.setProperty("value", 20)
        self.hsTransaccionesBloque.setOrientation(QtCore.Qt.Horizontal)
        self.hsTransaccionesBloque.setObjectName("hsTransaccionesBloque")
        self.hsTransaccionesBloque.setTickPosition(QSlider.TicksBothSides)
        self.hsTransaccionesBloque.setTickInterval(3)

        self.labelBloques = QtWidgets.QLabel(self.centralwidget)
        self.labelBloques.setGeometry(QtCore.QRect(20, 250, 271, 16))
        self.labelBloques.setObjectName("labelBloques")
        self.hsTotalBloques = QtWidgets.QSlider(self.centralwidget)
        self.hsTotalBloques.setGeometry(QtCore.QRect(310, 250, 311, 20))
        self.hsTotalBloques.setMinimum(1000)
        self.hsTotalBloques.setMaximum(3000)
        self.hsTotalBloques.setPageStep(5)
```

```
self.hsTotalBloques.setProperty("value", 1500)
self.hsTotalBloques.setOrientation(QtCore.Qt.Horizontal)
self.hsTotalBloques.setObjectName("hsTotalBloques")
self.hsTotalBloques.setTickPosition(QSlider.TicksBothSides)
self.hsTotalBloques.setTickInterval(100)

self.labelArchivoSalida = QtWidgets.QLabel(self.centralwidget)
self.labelArchivoSalida.setGeometry(QtCore.QRect(20, 320, 271, 16))
self.labelArchivoSalida.setObjectName("labelArchivoSalida")

self.textArchivoSalida = QtWidgets.QPlainTextEdit(self.centralwidget)
self.textArchivoSalida.setGeometry(QtCore.QRect(300, 310, 321, 31))
self.textArchivoSalida.setObjectName("textArchivoSalida")

self.progressProceso = QtWidgets.QProgressBar(self.centralwidget)
self.progressProceso.setGeometry(QtCore.QRect(20, 380, 601, 23))
self.progressProceso.setProperty("value", 0)
self.progressProceso.setObjectName("progressProceso")

self.botonProcesar = QtWidgets.QPushButton(self.centralwidget)
self.botonProcesar.setGeometry(QtCore.QRect(110, 420, 121, 31))
self.botonProcesar.setObjectName("botonProcesar")
self.botonSalir = QtWidgets.QPushButton(self.centralwidget)
self.botonSalir.setGeometry(QtCore.QRect(410, 420, 121, 31))
self.botonSalir.setObjectName("botonSalir")
SimuladorBlockchain.setCentralWidget(self.centralwidget)

self.menubar = QtWidgets.QMenuBar(SimuladorBlockchain)
self.menubar.setGeometry(QtCore.QRect(0, 0, 633, 20))
self.menubar.setObjectName("menubar")
self.menuMen = QtWidgets.QMenu(self.menubar)
self.menuMen.setObjectName("menuMen")

self.salir = QAction('Salir', self.menuMen)
self.menuMen.addAction(self.salir)
self.salir.triggered.connect(self.accionSalir)


self.menuAyuda = QtWidgets.QMenu(self.menubar)
self.menuAyuda.setObjectName("menuAyuda")

self.ayuda = QAction('Ver ayuda', self.menuAyuda)
self.menuAyuda.addAction(self.ayuda)
self.ayuda.triggered.connect(self.verTutorial)

SimuladorBlockchain.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(SimuladorBlockchain)
self.statusbar.setObjectName("statusbar")
```

```
        SimuladorBlockchain.setStatusBar(self.statusbar)
        self.menubar.addAction(self.menuMen.menuAction())
        self.menubar.addAction(self.menuAyuda.menuAction())

        self.retranslateUi(SimuladorBlockchain)
        QtCore.QMetaObject.connectSlotsByName(SimuladorBlockchain)
        self.botonSalir.clicked.connect(self.accionSalir)
        self.botonProcesar.clicked.connect(self.accionProcesar)

    def retranslateUi(self, SimuladorBlockchain):
        _translate = QtCore.QCoreApplication.translate
        SimuladorBlockchain.setWindowTitle(_translate("SimuladorBlockchain", "Ventana
        self.labelTitulo.setText(_translate("SimuladorBlockchain", "<html><head/><body
        self.hsCantidadUsuarios.setAccessibleName(_translate("SimuladorBlockchain", "|
        self.labelCantidadUsuarios.setText(_translate("SimuladorBlockchain", "<html><he
        self.labelNodosMineros.setText(_translate("SimuladorBlockchain", "<html><head/
        self.hsPoolMineria.setAccessibleName(_translate("SimuladorBlockchain", "|"))
        self.labelmaxtrans.setText(_translate("SimuladorBlockchain", "<html><head/><boo
        self.hsTransaccionesBloque.setAccessibleName(_translate("SimuladorBlockchain",
        self.labelBloques.setText(_translate("SimuladorBlockchain", "<html><head/><body
        self.hsTotalBloques.setAccessibleName(_translate("SimuladorBlockchain", "|"))
        self.labelArchivoSalida.setText(_translate("SimuladorBlockchain", "<html><head/
        self.textArchivoSalida.setPlainText(_translate("SimuladorBlockchain", "< nombre
        self.botonProcesar.setText(_translate("SimuladorBlockchain", "Procesar"))
        self.botonSalir.setText(_translate("SimuladorBlockchain", "Salir"))
        self.menuMen.setTitle(_translate("SimuladorBlockchain", "Menú"))
        self.menuAyuda.setTitle(_translate("SimuladorBlockchain", "Ayuda"))


    def accionSalir(self):
        mbox = QMessageBox()
        mbox.setWindowTitle('Simulador Blockchain')
        mbox.setText("Gracias por usar la aplicación")
        mbox.setIcon(QMessageBox.Information)
        mbox.setStandardButtons(QMessageBox.Ok)
        mbox.exec()
        sys.exit(0)

    def verTutorial(self):
        filename = os.path.join(os.path.dirname(__file__),"tutorial.pdf")
        #print("---"+filename+"---")
        if os.name == "posix":
            os.system("/usr/bin/xdg-open " + filename)
        else :
            os.filestart(filename)
```

```
def accionProcesar(self):
    self.botonSalir.setEnabled(False)
    sm = Simulador("Simulación del comportamiento de una cadena de bloques");
    cantidadUsuarios = int(self.hsCantidadUsuarios.value())
    sm.crearUsuarios(cantidadUsuarios);
    cantidadMineros = int(self.hsPoolMineria.value());
    maxTransaccionesPorBloque = int(self.hsTransaccionesBloque.value());
    totalBloquesCaena = int(self.hsTotalBloques.value())
    fsalida = self.textArchivoSalida.toPlainText()
    fsalida = fsalida.lstrip()
    fsalida = fsalida.rstrip()
    fsalida = fsalida.strip()
    mbox = QMessageBox()
    mbox.setWindowTitle('Simulador Blockchain')
    mbox.setText("Usuarios =" + str(cantidadUsuarios) + "\n" +
                 "Pools =" + str(cantidadMineros) + "\n" +
                 "Transacciones x Bloque=" + str(maxTransaccionesPorBloque) + "\n"
                 "Total Bloques=" + str(totalBloquesCaena) + "\n" +
                 "Archivo de salida=" + str(fsalida))

    mbox.setIcon(QMessageBox.Information)
    mbox.setStandardButtons(QMessageBox.Ok)
    mbox.exec()
    sm.inicializarBlockchain(cantidadUsuarios,cantidadMineros);

    tsimulacion = open(fsalida + '.txt', 'a+');
    instanteInicial = datetime.now()
    sm.simular(maxTransaccionesPorBloque,totalBloquesCaena,cantidadUsuarios,self.pr
    instanteFinal = datetime.now()
    tiempo = instanteFinal - instanteInicial # Devuelve un objeto timedelta
    segundos = tiempo.seconds;
    tsimulacion.write("Simulacion \n");
    tsimulacion.write("cantidadUsuarios " + str(cantidadUsuarios) + "\n");
    tsimulacion.write("cantidadMineros " + str(cantidadMineros)+ "\n");
    tsimulacion.write("maxTransaccionesPorBloque " + str(maxTransaccionesPorBloque)
    tsimulacion.write("totalBloquesCadena " + str(totalBloquesCaena)+ "\n");
    tsimulacion.write("demoro " + str(segundos) + " segundos"+ "\n");
    tsimulacion.write("\nFin de la simulación: demoró " + str(segundos) + " segundo
    tsimulacion.write("Estado de los Nodos Mineros...");
    tsimulacion.write("{} , {} , {} , {}, {}".format(str("Descripción"),str("Direc
    tsimulacion.write(str(sm.verNodosMineros()));
    tsimulacion.write("Estado de los Usuarios"+ "\n")
    tsimulacion.write("{}, {} , {} ".format(str("#id"),str("Direccion Hash del Usa
    tsimulacion.write(sm.verUsuarios()+ "\n");
    tsimulacion.write("Cadena de Bloques\n");
    tsimulacion.write(sm._blockchain.getCadenaSerializada()+ "\n");
    tsimulacion.close();
    mbox = QMessageBox()
```

```
            mbox.setWindowTitle('Proceso Finalizado')
            mbox.setText("Abrir el archivo " + fsalida + '.txt')
            mbox.setIcon(QMessageBox.Information)
            mbox.setStandardButtons(QMessageBox.Ok)
            mbox.exec()
            self.botonSalir.setEnabled(True)

    if __name__ == "__main__":
        app = QtWidgets.QApplication(sys.argv)
        SimuladorBlockchain = QtWidgets.QMainWindow()
        ui = Ui_SimuladorBlockchain()
        ui.setupUi(SimuladorBlockchain)
        SimuladorBlockchain.show()
        sys.exit(app.exec_())
```

Clase Simulador

```
In [ ]: #!/usr/bin/env python3
        # -*- coding: utf-8 -*-

        from random import randint
        from cliente.Usuario import Usuario
        from cadena.CadenaBloques import CadenaBloques
        from cadena.NodoMinador import NodoMinador
        from cadena.PoolMinado import PoolMinado
        from random import randrange
        from random import random
        from random import seed
        from datetime import datetime
        from PyQt5.QtWidgets import QProgressBar

        """
        Define la estructura del bloque.
        """

        class Simulador(object):

            def __init__(self, nombre):
                """
                Creación del escenario final de simulación
                :param _nombre Nombre Asignado al escenario
                :param _usuarios Conjunto de usuarios en la simulación
                :param _blockchain Cadena de Bloques
                :param _pool Conjunto de nodos mineros de la cadena
                """
                self._nombre = nombre;
                self._usuarios = {};
                self._blockchain = CadenaBloques();
```

```
        self._pool = PoolMinado()

    def crearUsuarios(self, cantidadUsuarios):
        """
        Inicializa el conjunto de usuarios de la cadena de bloques
        :param cantidadUsuarios: Conjunto inicial de Usuarios en la Simulación
        """
        for i in range(cantidadUsuarios):
            temp = Usuario(i);
            temp.setHashUsuario();
            self._usuarios[temp.gethashUsuario()] = temp;

    def distribuirUsuariosMineros(self, maxUsuarios, maxMineros):
        """
        Permite distribuir de manera uniforme los usuarios a los nodos mineros
        :param maxMineros: Conjunto de Nodos mineros de la blockchain
        :param maxUsuarios: Total de usuarios en el sistema
        :return arreglo con los usuarios distribuidos para cada nodo minero
        """
        #Se construye una conjunto aleatorio de nodos mineros
        usuariosNodo = [];
        usuariosMineros = {};
        contador = maxUsuarios//maxMineros - 1;
        aux = 0;
        for key in self._usuarios:
            usuariosMineros[key] = self._usuarios[key];
            if(aux < contador):
                aux += 1;
            else:
                usuariosNodo.append(usuariosMineros);
                aux = 0;
                usuariosMineros = {};
        return usuariosNodo;

    def verUsuarios(self):
        """
        Consulta el total de usuarios registrados en un momento dado en la cadena de b
        :return  Representación en string del conjunto de Usuarios en la Simulación
        """
        a = "";
        for key in self._usuarios:
            a += str(self._usuarios[key])+"\n";
        return a;

    def consultarHashUsuario(self, idUsuario):
        """
        Retorna el Hash de un usuario dado su número de identificación
        :param idUsuario identificador entero de cada usuario en el sistema
```

```
        :return Hash de usuario que corresponde al identificador dado
        """
        for key in self._usuarios:
            if(self._usuarios[key]._idUsuario == idUsuario ):
                return self._usuarios[key].gethashUsuario();

    def consultarUsuario(self, hashUsuario):
        """
        Consulta la información de un usuario dado su hash
        :param hashUsuario LLave del diccionario
        :return usuario que corresponde con la llave dada
        """
        return self._usuarios[hashUsuario];

    def inicializarBlockchain(self, maxUsuarios, cantidadMineros):
        """
        Creación del bloque génesis de la cadena y asignación de nodos mineros
        :param cantidadMineros, Total de nodos Mineros para la cadena
        :param maxUsuarios: Total de usuarios en el sistema
        """
        emisor = self.consultarUsuario(self.consultarHashUsuario(0));
        receptor = self.consultarUsuario(self.consultarHashUsuario(0));
        transacciones = [];
        transacciones.append(emisor.enviar(100,receptor.gethashUsuario()));
        self._blockchain.bloqueGenesis(transacciones);
        usuariosNodo = self.distribuirUsuariosMineros(maxUsuarios,cantidadMineros);
        self.creacionMineros(cantidadMineros,usuariosNodo);

    def creacionMineros(self, cantidadMineros, usuariosNodo):
        """
        Adiciona los nodos mineros a la blockchain. Cada nodo minero tendrá asociado
        un grupo de usuarios, quienés recibirán recompensa por su esfuerzo de minado.
        :param cantidadMineros, Total de nodos Mineros para la cadena
        :param usuariosNodo, Distribución de usuarios mineros para cada nodo Minero
        """
        for m in range(cantidadMineros):
           address = "127.0.0." + str(1+randint(1,250));
           puerto = 1025+randint(1000,10000);
           descripcion = "Minero " + str(m);
           minerito = NodoMinador(address, puerto,descripcion,usuariosNodo[m]);
           minerito.actualizarCadena(self._blockchain);
           self._pool.adicionarMinero(minerito);

    def simular(self, maxTransaccionesPorBloque,totalBloquesCaena, cantidadUsuarios,ba
        """
        Realiza una simulación de evolución de la blockchain con los parámetros dados.
        :param cantidadUsuarios, Total de usuarios activos en la cadena
        :param cantidadMineros, Total de nodos Mineros para la cadena
```

```
        :param usuariosNodo, Distribución de usuarios mineros para cada nodo Minero
        """
        seed();
        transacciones = [];
        emisor = self.consultarHashUsuario(0);
        receptor = self.consultarHashUsuario(1);
        cantidad =  self.consultarUsuario(emisor)._saldo * random();
        transacciones.append(self.consultarUsuario(emisor).enviar(cantidad, self.consul
        self._pool.torneoPorNuevoBloque(transacciones);
        cantidadPosibleReceptores = 1;
        for i in range(1,totalBloquesCaena):
            transacciones = [];
            if cantidadPosibleReceptores == cantidadUsuarios-1:
                cantidadPosibleReceptores = 1;
            #Se construye una conjunto aleatorio de transacciones
            for tr in range(1+randrange(maxTransaccionesPorBloque)):
                seed();
                if cantidadPosibleReceptores == cantidadUsuarios-1:
                    cantidadPosibleReceptores = 1;
                intemisor = self.seleccionarUsuario(cantidadPosibleReceptores,-1);
                intreceptor = self.seleccionarUsuario(cantidadPosibleReceptores,intemiso
                emisor = self.consultarHashUsuario(intemisor);
                receptor = self.consultarHashUsuario(intreceptor);
                cantidad =  self.consultarUsuario(emisor)._saldo * random();
                #print("Cantidad receptores " + str(cantidadPosibleReceptores)+ " recep
                #print(".", end="");

                transacciones.append(self.consultarUsuario(emisor).enviar(cantidad, seli
                cantidadPosibleReceptores = cantidadPosibleReceptores + 1;
            instanteInicial = datetime.now()
            #print("len transacciones = " + str(len(transacciones)));
            self._pool.torneoPorNuevoBloque(transacciones);
            instanteFinal = datetime.now()
            tiempo = instanteFinal - instanteInicial # Devuelve un objeto timedelta
            segundos = tiempo.microseconds
            #print("Torneo " + str(i) + " " + str(segundos) + " microseconds");
            if barraProgreso is not None:
                barraProgreso.setValue(1 + ((100*i) // totalBloquesCaena))
            else :
                print("Bloque " + str(i))



    def seleccionarUsuario(self,cantidadPosibleReceptores, diferentea):
        seed();
        x = randint(0,cantidadPosibleReceptores+1);
        while x == diferentea:
            x = randint(0,cantidadPosibleReceptores+1);
```

```
        return x;

    def verNodosMineros(self):
        """
        Consulta el total de nodos mineros registrados en la cadena de bloques
        :return pool de mineros de la blockchain
        """
        return self._pool;

    def logEjecucion(self, file, mensaje):
        file.write(mensaje + "\n")
```

SIMULADOR DEL COMPORTAMIENTO DE UNA CADENA DE BLOQUES
VERSIÓN 1.0

CONFIGURACIÓN DEL ENTORNO

Sistema Operativo linux

1. Python 3.6.7 instalado. (https://www.python.org/downloads)

```
python3 --version
```

2. Paquetes de Instalación PIP

```
sudo apt-get install python3-pip
```

3. Instale el paquete NumPy (https://docs.scipy.org/doc/numpy/user/install.html)

```
python3 -m pip install --user numpy
```

4. Instale PyQt5

```
pip3 install PyQt5
```

5. Descargue el repositorio desde github: BlocksUD

```
https://github.com/devappsud/blocksUD
```

6. Descomprima el archivo blocksUD-master.zip

```
unzip blocksUD-master
```

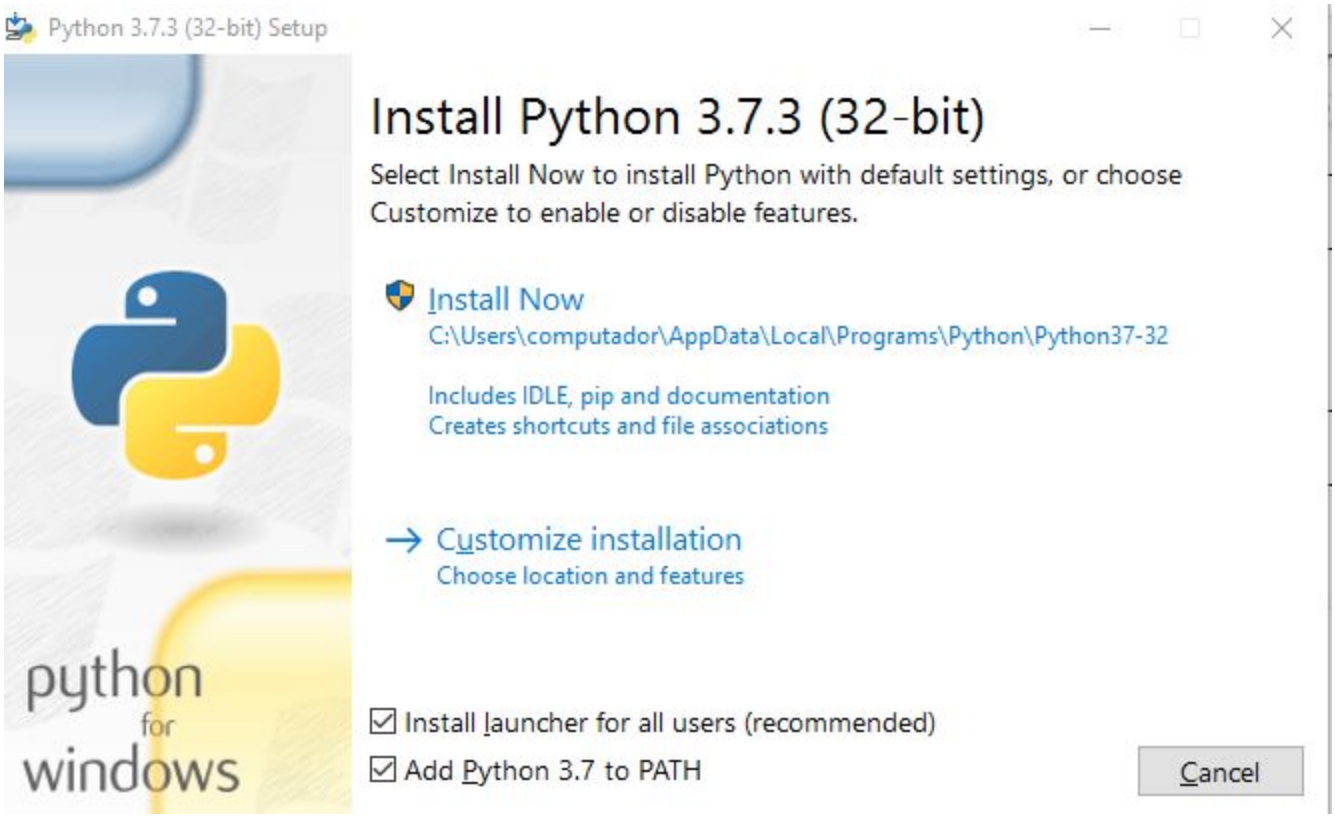
7. Ingresar al directorio de la aplicación

```
cd blocksUD-master
```

Sistema Operativo Windows

1. Python 3.6.7 instalado. (https://www.python.org/downloads

Durante la instalaciòn seleccione: add Python 3.x to path
python3 --version



REINICIE EL SISTEMA OPERATIVO

2. Paquetes de Instalación PIP

Contenido en el paquete de instalaciòn

3. Instale el paquete NumPy (https://docs.scipy.org/doc/numpy/user/install.html)

Sìmbolo del sistema python -m pip install --user numpy

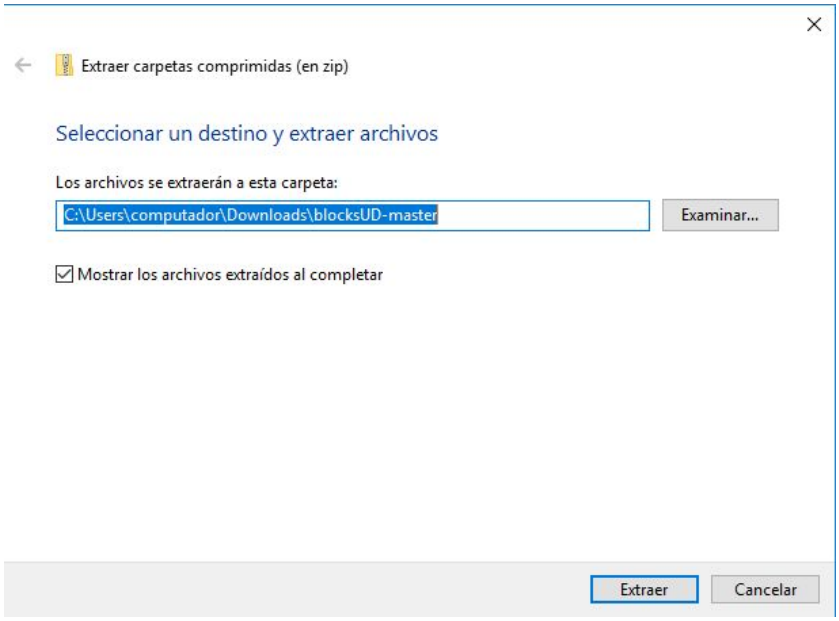
4. Instale PyQt5

pip3 install PyQt5

5. Descargue el repositorio desde github: BlocksUD

https://github.com/devappsud/blocksUD

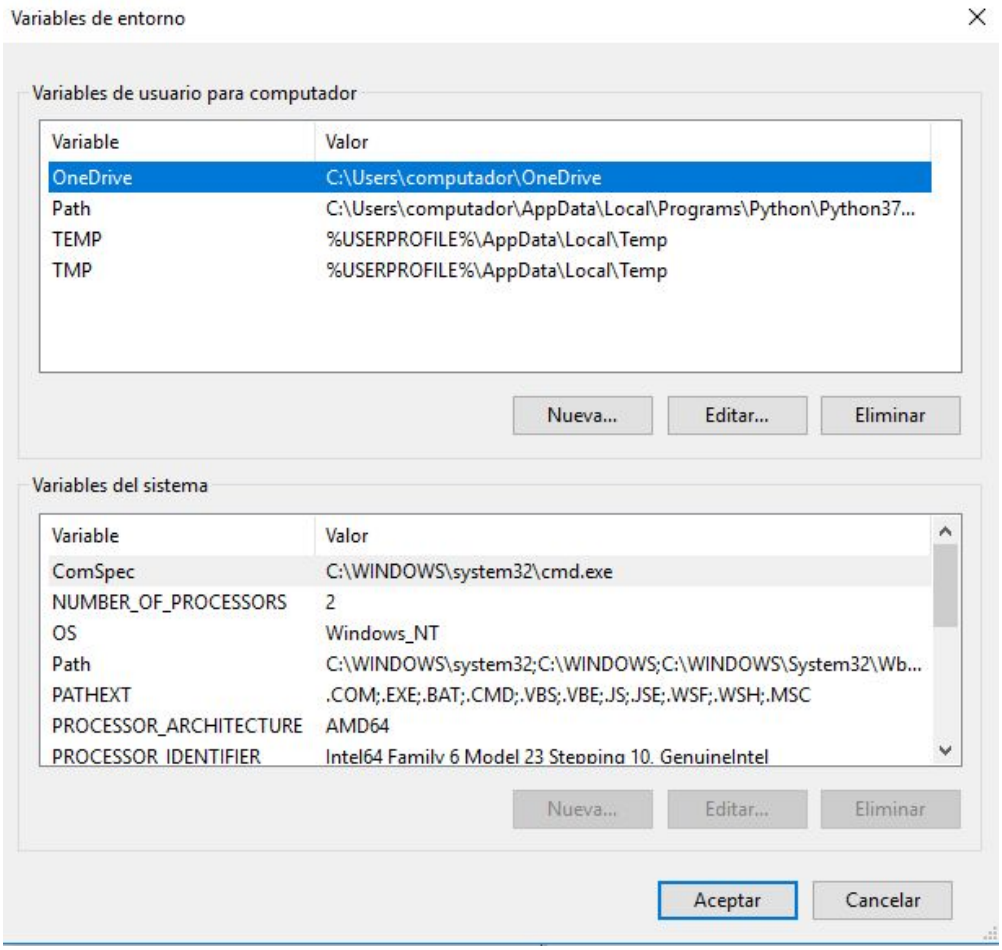
6. Descomprima el archivo blocksUD-master.zip

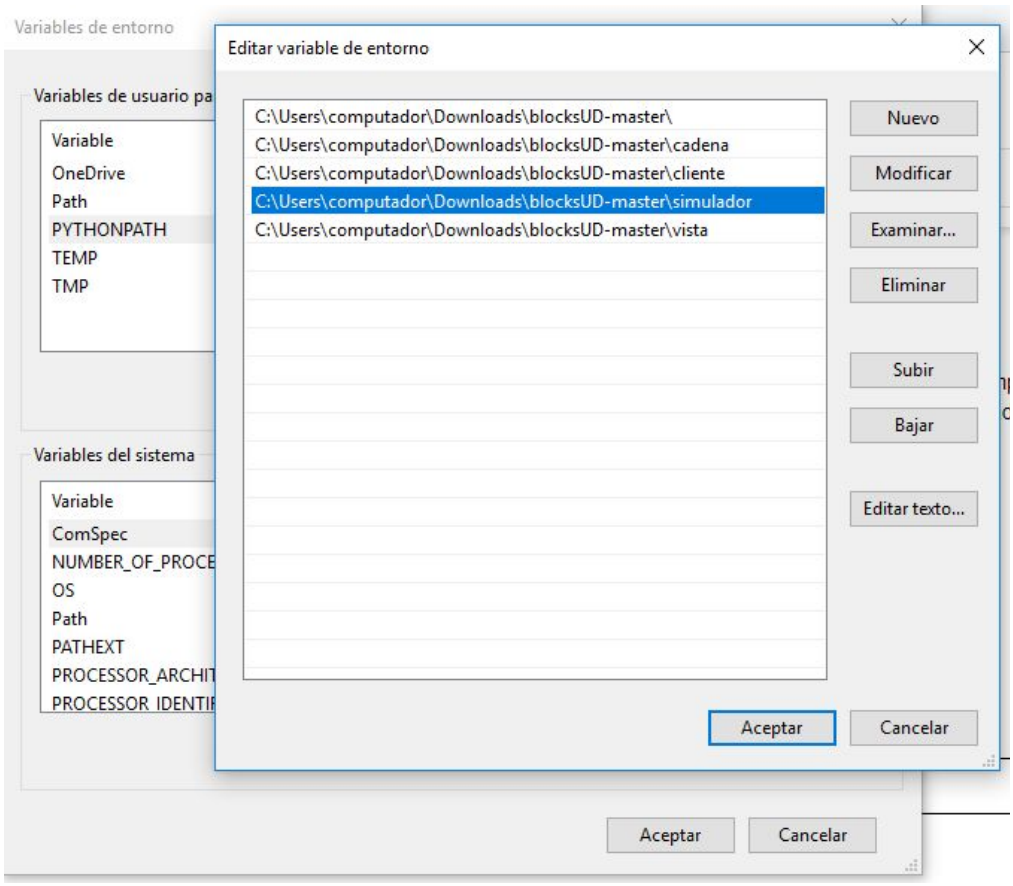


7. Ingresar al directorio de la aplicación

```
blocksUD-master
```

8. Adicione la variable PYTHONPATH con la ruta donde se descargó el proyecto, agregando la ruta de cada uno de los subdirectorios.


Variables de entorno
Variables de usuario para computador
Variable
Valor
OneDrive
C:\Users\computador\OneDrive
Path
C:\Users\computador\AppData\Local\Programs\Python\Python37...
TEMP
%USERPROFILE%\AppData\Local\Temp
TMP
%USERPROFILE%\AppData\Local\Temp
Nueva...
Editar...
Eliminar
Variables del sistema
Variable
Valor
ComSpec
C:\WINDOWS\system32\cmd.exe
NUMBER_OF_PROCESSORS
2
OS
Windows_NT
Path
C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wb...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 23 Stepping 10, GenuineIntel
Nueva...
Editar...
Eliminar
Aceptar
Cancelar



EJECUCIÓN

1. Versión Consola

run_windows.bat

Al finalizar el proceso los resultados de la simulación quedan registrados en el archivo de salida. Este archivo puede ser procesado con la herramienta de análisis de datos de preferencia del usuario.

2. Versión Gráfica

run_windows_grafico.bat

Al finalizar el proceso los resultados de la simulación quedan registrados en el archivo de salida (ejemplo002.txt, en este caso). Este archivo puede ser procesado con la herramienta de análisis de datos de preferencia del usuario.

AUTORES

- MSc ROBERTO A. PAVA DIAZ
  Docente de Planta - Facultad de Ingeniería
  Universidad Distrital Francisco José de Caldas

- MSc NANCY GELVEZ GARCIA
  Docente de Planta - Facultad de Ingeniería
  Universidad Distrital Francisco José de Caldas

- PhD NELSON ENRIQUE VERA
  Docente de Planta - Facultad de Ingeniería
  Universidad Distrital Francisco José de Caldas

VERSIÓN EN CONSOLA

Correr el archivo: TestSimulador.py

- Digitar uno (1) para una ejecución del programa con valores por defecto, y otro número para ingresar un valor para cada parámetro.
- Luego ingresar el nombre del archivo donde se almacenarán los resultados de la simulación y enter para iniciar el proceso

```
IPython 7.4.0 -- An enhanced Interactive Python.

In [1]: runfile('/datos1/Dropbox/doctorado/semestre1/orientadaServicios/codeUsed/blocksUD2/simulador/TestSimulador.py', wdir='/datos1/
Dropbox/doctorado/semestre1/orientadaServicios/codeUsed/blocksUD2/simulador')

Desea una ejecución con valores por defecto (1 en caso afirmativo, otro numero en caso contrario): 1
Creando la cadena de bloques...Adicionando Bloque Génesis...
Hash usuario génesis: ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2
Imprimiendo Bloque Génesis...

Inicio de Bloque
0 2 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
Transacciones del bloque
None
Fin Transacciones del bloque
Fin Bloque 0.


nombre Archivo de salida [sin espacios, ni caracteres especiales]: prueba00001.txt
```

- La simulación imprime el número de bloque en el cual se encuentra

```
nombre Archivo de salida [sin espacios, ni caracteres especiales]: prueba00001.txt
Bloque 1
Bloque 2
Bloque 3
Bloque 4
Bloque 5
Bloque 6
Bloque 7
Bloque 8
Bloque 9
Bloque 10
```

….

```
Bloque 2988
Bloque 2989
Bloque 2990
Bloque 2991
Bloque 2992
Bloque 2993
Bloque 2994
Bloque 2995
Bloque 2996
Bloque 2997
Bloque 2998
Bloque 2999
```

- En el archivo prueba00001.txt contiene los datos generados por el simulador y puede ser procesado por la herramienta de preferencia del usuario

- Visualización parcial del archivo prueba00001.txt

```
Simulacion
cantidadUsuarios 100
cantidadMineros 5
maxTransaccionesPorBloque 20
totalBloquesCadena 3000
demoro 321 segundos

Fin de la simulación: demoró 321 segundos
Estado de los Nodos Mineros...Descripción , Direccion , Utilidad , Bloques, Usuarios
Minero 0 , 127.0.0.75:5576 , 15449073 , 989 , 20
Minero 1 , 127.0.0.245:5195 , 17105206 , 1145 , 20
Minero 2 , 127.0.0.40:9993 , 3534556 , 373 , 20
Minero 3 , 127.0.0.146:4633 , 3540121 , 378 , 20
Minero 4 , 127.0.0.57:3004 , 1026862 , 115 , 20
Estado de los Usuarios
#id, Direccion Hash del Usario , Saldo
0, ae1bc1b04561fac552570296dfe154094dc4df4b6d4970ca81ec9507046375b2 , 2.4334835358362144
1, 9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 , 41.63953862997468
2, b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 , 387.8173151369244
3, 0890bee71d46a465a64625124eee9a424a66908e11ce283ed4afaf5a9551cef0 , 19.49149548995281
4, 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a , 778.1292528831865
5, 5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 , 32.32717803748672
```

Bloque génesis

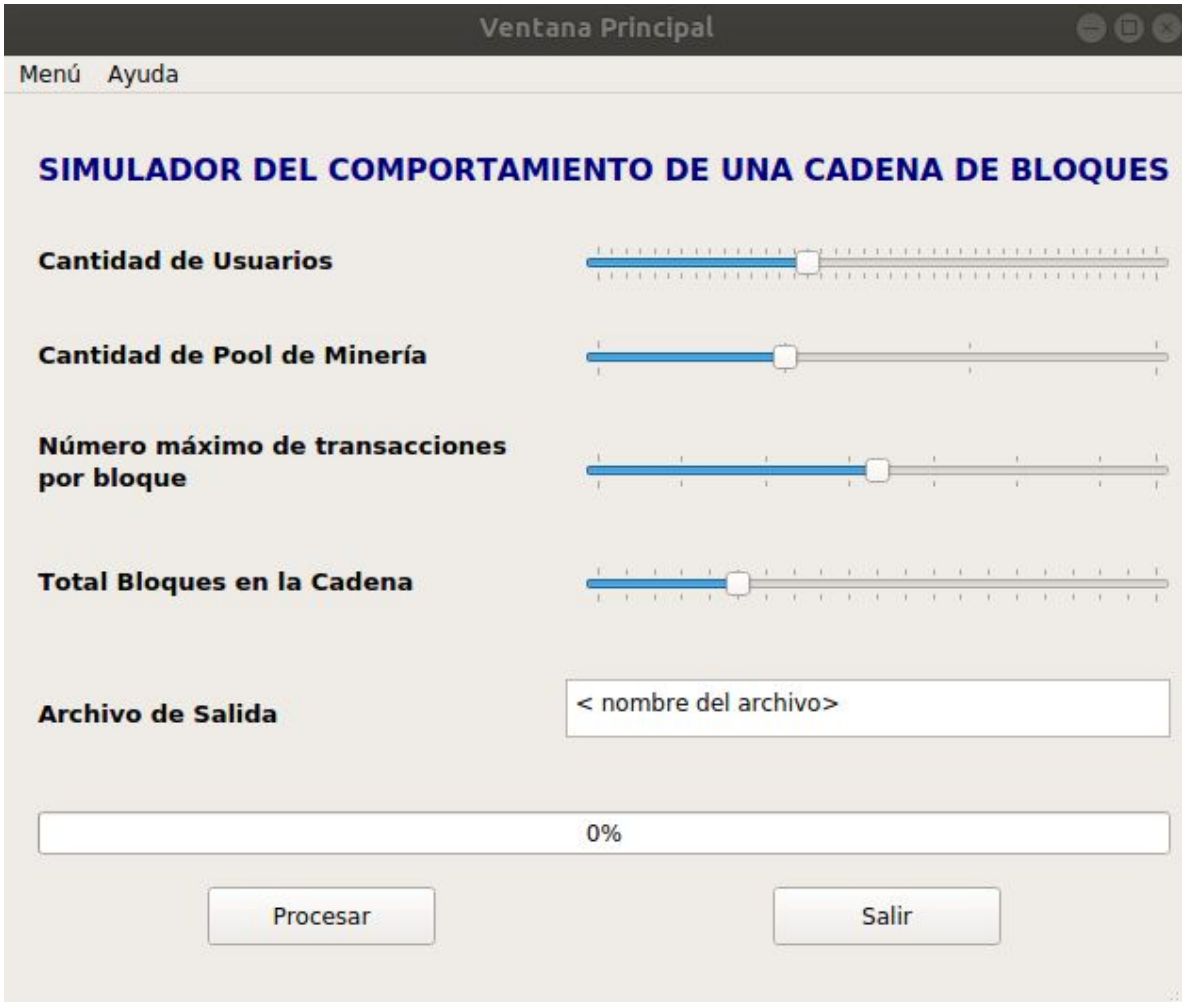
```
Cadena de Bloques

Inicio de Bloque
0 2 5feceb66ffc86f38d952786c6d696c79c2dbc239dd4e91b46729d73a27fb57e9
Transacciones del bloque
None
Fin Transacciones del bloque
Fin Bloque 0.
```

último bloque de la simulación

```
Inicio de Bloque
3000 24180 ef5836b6953cce86200fad71f1c4d497d3ea546e35170f340c90ab916897c70d
Transacciones del bloque
e8a26469e9a9879b4ef23fb12236e77a66b7e1a55b060694537c39fdca4fec8c 643463717c6429de41adbd4c0a6254aae667f91b56050f049d6eb7d147bbcfa9 46.405749361646 29
5fafa6531ad8489b778a631289e5d41cc2a6dee01b2d17dab578b9ecc47abdac 9f3743c16996e33578cdd61c0b2baf35eb81bc3d04207c2ad66aa77b6382e9a1 113.956831179533 84
5d4a62b58203901f4e7502ccbdd5a2e52a234bb44545549fd5c87edef9202363 11446965a28709ff7577c812099db7093ae7a31f8aa8ebe2a81d15ee876b0f16 10.918065948594 593
06aeb9cf521c8e7114647117303ebf837f50bd39b38104ce5e3371bf54d1db61 3862c4ce567a7c413d476da84a2ee4d6937905cf87c0eaffe60b2e4af3e1dc75 53.88462260115935
9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 442581fffd4d604d91b1aaa4fe1cebea4a10019d42f8f7c66ef2840770048295 15.416932938523303
9d5eab463378fe259f33d07590aa6a913a1d099d5a276e353a955cb2da4311f2 839604719f7a03403bda1bea57e369f0ee226a4aa8a092aee337252b330e7ab8 80.16578472943439
192990181fd61e86ffa1e28ec8741a76c1640e816bd75b00e8e7e4499b12c1af 6aece84b2c86c9624574712ba50b4a0a8dfa1fc3d4e46685f55dd82d636e9ba1 76.6080055559965
6aece84b2c86c9624574712ba50b4a0a8dfa1fc3d4e46685f55dd82d636e9ba1 906137d181c223a3a6ab42b56f0363632d5ca32c55a06c17f8e319b5f4d290c9 420.175188661799 95
1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a 9f3743c16996e33578cdd61c0b2baf35eb81bc3d04207c2ad66aa77b6382e9a1 37.925242014557405
d3e0b66ec1e56b92a5c0b1bd7c57540697da963987253413c40b77636707f13c 188ccaec7832607294b07d88109f15299b707dc4817011f95529cf2a6dd711c8 1.352844012573736 4
c1f330b2fa35be43c2d2e695cc0d73ba2c074247ef4e10a82e4a4eca63307c80 b4aec9b924aa7f7d83851c90a7f5142be0cec41c7c46cff2fb89bf0bbf055f67 3.065875536025921 6
b29492fe4bf566a54a5ac57bfb49cd6441b5cc3dbb19b36804da13b6a7495032 643463717c6429de41adbd4c0a6254aae667f91b56050f049d6eb7d147bbcfa9 185.039816638762 4
84abbe76519ebaf94fcf0ecae80e54585edf60fc7a6df983c24f6b6475d644c3 b29492fe4bf566a54a5ac57bfb49cd6441b5cc3dbb19b36804da13b6a7495032 14.714689682828727
1c0f80e789689876618b6f40d93b2c812477ed7d42cfd3666318868314d9be6a 1206cfe53ef4288f00f4c53f5377601c07bdc95f6a5249b95183cc5db54ad93a 774.542189875157 3
Fin Transacciones del bloque
Fin Bloque 3000.
```

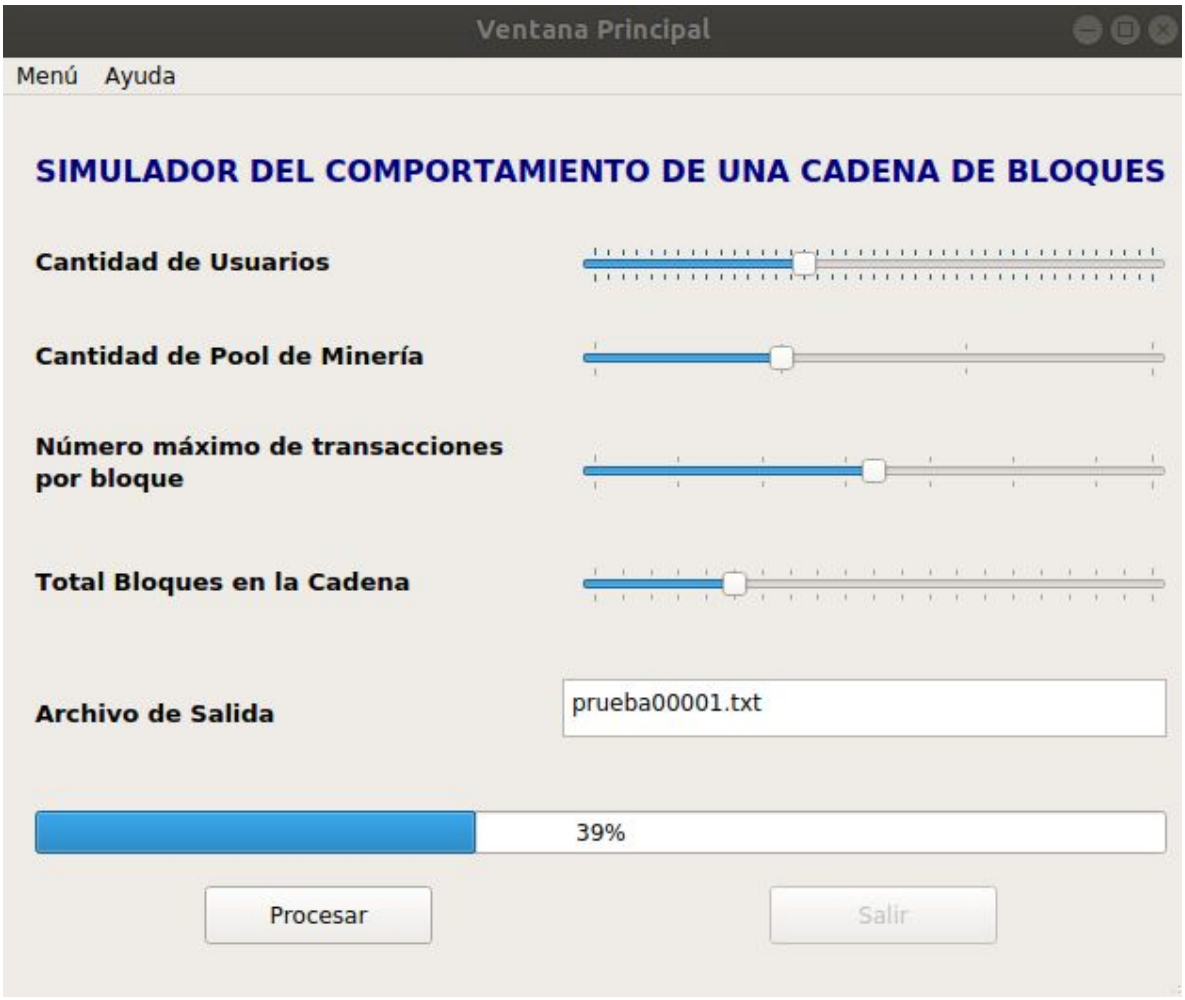
VERSIÓN GUI



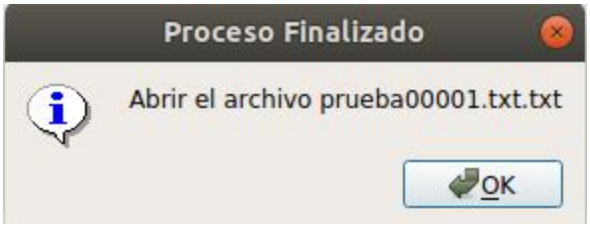
Correr el archivo: blockchainsimulator.py

- Ajustar los controles de la aplicación conforme se desee modificar el valor de los parámetros.
- Luego ingresar el nombre del archivo donde se almacenarán los resultados de la simulación y click sobre el botón procesar para iniciar, aparecerá un mensaje de confirmación resaltando los valores seleccionados.

- La barra de progreso se actualiza cada vez que se ha minado un bloque.



- Al finalizar el proceso se muestra un mensaje indicando que se debe abrir el archivo de salida.



- En el archivo prueba00001.txt contiene los datos generados por el simulador y puede ser procesado por la herramienta de preferencia del usuario. En la sección de consola se visualizó información parcial del archivo de salida.