Trabajo practico final

Laberinto

Redes de Datos

Ing. Diego Azcurra – Ing. Damián Santos – Lic. Gustavo Fernandez

UNLA

2018

|  |  |  |
| --- | --- | --- |
| DNI | Nombre | Evaluación Individual |
| 31794620 | Tisera Lucas L. |  |
| 40946043 | Mathov Camila |  |
| 40190709 | Mateus Nicolas |  |
|  |  |  |
|  |  |  |
| Evaluación Trabajo |  | |

## Objetivo

Desarrollar una aplicación que permita jugar en un laberinto bajo el esquema de cliente y servidor.

## Descripción

## Laberinto

El laberinto es un área que cuenta con una única entrada y una única salida. Desde esa entrada surgen varios caminos de los cuales al menos uno permite llegar a la salida. Para salir, primero es necesario encontrar una llave. A su vez, a lo largo del laberinto se podrá encontrar oro. El oro sirve para “pagar” a los guardias que vigilan los caminos. Si nos cruzamos con un guardia y no tenemos oro, nos matará y tendremos que empezar de nuevo.

El laberinto es una matriz cuadrada de una determinada dimensión X. En cada posición, se podrá encontrar un carácter que especificará:

* E: Entrada.
* P: Pared.
* C: Camino.
* K: Llave.
* S: Salida.
* O: Oro.
* G: Guardia.

## Objetivo

El objetivo del juego es encontrar la salida recolectando la mayor cantidad de oro posible.

## Arquitectura

El juego se desarrollará bajo una arquitectura cliente servidor. La aplicación cliente será el jugador y se conectará al servidor para solicitar todos los elementos del juego.

## Desarrollo del juego

Para poder jugar, el cliente deberá conectarse al servidor mediante un usuario y contraseña.

El servidor posiciona al jugador en la casilla indicada con la letra E. En todos los casos, el servidor va revelar solo partes del laberinto: matrices cuadradas armadas desde un parámetro Y (desde la ubicación del jugador, se arma un cuadrado avanzando Y casillas en todos los sentidos). Desde esa posición, el jugador podrá moverse siguiendo los casilleros marcados con C. Al posicionarse en la nueva ubicación, solicitará al servidor que revele las casillas correspondientes.

Por ejemplo, tomemos una parte de un determinado laberinto:

000000000111111

123456789012345

01|PPPPPPPPPPPPPPP

02|ECCCCPPPCCCCCCP

03|PPPPCPPPCPPPPPP

04|PPPPCCCCCPPPPPP

05|PPPPCPPPCCCCCCP

06|PPPPCPCCCPPCPPP

07|PPPCCPPPCCCCPPP

El juego comienza en la posición (2,1). Suponiendo Y= 2, el servidor en este caso va a revelar únicamente:

000

123

01|PPP

02|ECC

03|PPP

04|PPP

Desde ahí, en este caso sólo podrá moverse en sentido horizontal a la posición (2,2). Para esto, el cliente enviará un comando de movimiento a la derecha y el servidor responderá con la nueva matriz:

0000

1234

01|PPPP

02|ECCC

03|PPPP

04|PPPP

La secuencia continúa en búsqueda de oro, de la llave y de la salida del laberinto.

## Requerimientos

Se pide desarrollar un sistema cliente/servidor que permita:

1. Servidor:
   * El servidor deberá permitir loggear jugadores.
   * Deberá administrar el juego. Recibirá comandos desde los jugadores y enviará respuestas.
2. Cliente:
   * Deberá poder conectarse a un servidor ingresando IP y puerto del mismo.
   * Luego de recibir la posición inicial del jugador, deberá interpretar los comandos ingresados por teclado, enviarlos al servidor y actuar en consecuencia.
3. Protocolo:
   * Todos los comandos que se intercambian entre cliente y servidor deberán respetar un protocolo definido por cada grupo. Como mínimo, se deberán contemplar los siguientes campos:
     + Encabezado:
       - Tipo de comando (Conexión, Movimiento, Tomar (para el oro o las llaves), Pagar, Abrir Puerta, etc).
       - Tamaño del mensaje: salvo que se decida trabajar con tamaños fijos.
     + Datos:
       - Datos de los distintos mensajes en el formato que decida el grupo.
   * El protocolo deberá ser documentado detalladamente. Tomar como referencia algún estándar de la industria (HTTP, FTP, telnet…). Por ejemplo:
     + - <https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto>
       - <https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP>
4. Seguridad:

El sistema deberá implementar al menos dos servicios de seguridad informática. El grupo seleccionará qué servicios y con qué implementaciones.

1. Consola cliente:

Se deberá desarrollar otra aplicación cliente que permita interactuar con el servidor mediante mensajes de pregunta y respuesta en formato texto (de la misma forma que lo hace el juego pero en forma manual).

El protocolo definido tendrá una cantidad de operaciones a las que se podrá acceder mediante un esquema de request/response. Esta aplicación deberá permitir que el usuario ingrese por consola un comando para que el cliente lo envíe al servidor y muestre la respuesta obtenida.

Por ejemplo, si el protocolo define que el mensaje de login es:

LOG|us:Damian|pass:Damian

Y la respuesta es:

LOG|200|ok

El usuario podrá ingresar esa cadena y deberá recibir la respuesta esperada. Si algún dato está mal, deberá recibir un código de error.

## Presentación

10/11/18: **Entrega del TP**.

17/11/18: **Recuperatorio del TP**.

**Condiciones de entrega**.

El trabajo entregado deberá contener:

* Una copia impresa del enunciado del trabajo práctico (TODO este documento, incluyendo los anexos).
* Una copia impresa de la estrategia de resolución del trabajo práctico. Es un texto descriptivo de cómo se estructuró la aplicación, como se separaron las capas, relaciones entre las entidades, es decir, todo aquello que consideren significativo para explicar la resolución del trabajo.
* El detalle del protocolo utilizado.
* El detalle de los servicios de seguridad informática aplicados y la forma en que fueron implementados.
* Una copia impresa de todos los archivos del proyecto. Poner como encabezado de cada hoja el nombre del archivo. Pueden imprimirse dos páginas por hoja, en formato horizontal.
* Un CD conteniendo en formato digital todos los puntos anteriores y el proyecto completo.
* La presentación deberá ser en un folio o carpeta, en forma prolija y debidamente identificada. Los CDs deberán contener el número de grupo y el nombre y los padrones o documentos de identidad de cada uno de los integrantes y deberán estar correctamente adjuntos al resto del trabajo práctico de forma tal que no puedan perderse. Además, deberá incluirse **todo** el proyecto desarrollado (**la carpeta completa** generada por el IDE, con los archivos del proyecto y el código fuente) incluyendo los archivos de pruebas.
* Todas las hojas deben estar numeradas.

El incumplimiento de cualquiera de las normas de entrega implicará la desaprobación del trabajo práctico.

Metodología de evaluación:

La Evaluación de los trabajos prácticos contará con una etapa grupal y una individual.

* Grupal: Se realizará un conjunto de pruebas sobre el trabajo presentado por los alumnos en presencia de los mismos. Se deberá aprobar la totalidad de las pruebas. En caso de que una prueba falle, los alumnos podrán intentar corregir el código mientras dure la evaluación.
* Individual: Se realizará una evaluación individual oral o escrita para cada alumno. Los temas a evaluar podrán ser, por ejemplo: preguntas teóricas sobre el contenido de la materia, preguntas sobre el trabajo práctico, codificación de alguna primitiva o modificación del trabajo práctico, etc.

La nota final del trabajo se calculará en función de las notas obtenidas en forma grupal e individual. La nota grupal será el promedio entre la primer presentación y el recuperatorio (en caso de necesitarlo). Por este motivo, SOLO deberán presentarse aquellos grupos que hayan concluido TODO el trabajo práctico ya que no se harán evaluaciones parciales.

## Revisiones

Se agregó el punto 5 dentro de los requerimientos.

Resumen

El proyecto consiste en crear un juego, basado en un Laberinto. La arquitectura del mismo, cuenta con un cliente y un servidor, desarrollados ambos en python. \*\*\*capas de entenderse entre si, definiendo un protocolo entre ellos.

1) Arquitectura

1a) El servidor

Desarrollamos un servidor, capaz de atender multiples usuarios. Este se comunica con el cliente a travez de Sockets, haciendo uso del protocolo TCP/IP. El socket que recibe las peticiones de los clientes es “(IP del servidor, 6666)” una vez establecida la conexión, se genera un Socket

“(IP del Cliente, #Puerto)” el cual oficia de canal de comunicación entre el Servidor y el cliente que realizo la peticion.

Los sockets estan configurados como No bloqueantes, para poder atender mas de un cliente al mismo tiempo.

Una vez que un cliente se conecta, el socket generado para la comunicación se agrega a una lista. El servidor queda en un bucle atendiendo las nuevas peticiones y esperando datos de los clientes, generando las respuestas adecuadas a cada interacción.

Cuando un cliente envia un dato vacio, el servidor interpreta que ya no hay mas que comunicar y este procede a cerrar la conexión (el socket) y removerlo de la lista.

1b) El cliente

El cliente, crea un socket “(IP del server, 6666)” el cual sirve para comunicarse con el servidor. Segun los comandos que ingrese el usuario, comunica a travez del socket las acciones que desea realizar. Por cada accion, tiene una devolucion que la muestra en la consola.

1c) Consola-Cliente

Se desarrollo una consola, para interactuar con el servidor. Esta se conecta automaticamente al servidor y le facilita un prompt al usuario, para que pueda mandar los comandos con el correspondiente formato.

2) Protocolo

Definimos un protocolo personalizado, adaptado a nustras necesidades, el cual enviara Strings con un formato similar al JSON. Los datos que se quieran comunicar deberan estar dentro de un paréntesis y en dupla *nombre:valor.* Para enviar mas de una dupla las separamos con comas “,”. Debe tambien llevar un encabezado, en el cual, el valor del campo indica cuantos campos (sin incluir el mismo) contiene el mensaje. En caso de que no coincida el valor del encabezado con la cantidad de campos del mensaje, se considera como mensaje invalido.

Ejemplo: “(login:2,usser:roberto, password:1234)”

2.1) El cliente realiza la peticion de conexion

- Cliente, envia peticion

- Servidor acepta la conexión entrante.

Aca es donde el cliente se encuentra conectado con el servidor.

2.2) El cliente envia los datos de loggin que ingreso el usuario

**- Cliente, envia usuario y contraseña, en un string con el formato**

**“(login:2,usser:*ussername*,password:*password*)”**

**- Servidor, responde: “(login:1,valido:*true/false*)”**

2.3) El cliente envia una peticion, pidiendo el menu de opciones que debera mostrarle al usuario.

**-Cliente, “(menu:1, estado:principal)”**

**-Server, “(menu:2,dato:*las opciones*,estado:principal)”**

En este momento el usuario debe escoger alguna de las opciones mostradas en pantalla.

Caso de selección de la opcion 1) Cargar mapa.

**-Cliente, “(menu:2,comando:*1*,estado:principal)”**

**-Servidor, “(menu:2,dato:*menu de mapas*,estado:mapas)”**

Caso de selección de la opcion 2) Ver Instrucciones.

**-Cliente, “(menu:2,comando:*2*,estado:principal)”**

**-Servidor, “(menu:2,dato:*instrucciones*,estado:instrucciones)”**

Caso de Seleccion de la opcion 3) Creditos.

**-Cliente, “(menu:2,comando:*3*,estado:principal)”**

**-Servidor, “(menu:2,dato:*creditos*,estado:creditos)”**

2.3.1) Carga de Mapas

Una vez seleccionada la opcion de mapas el cliente, pedira que se le pase el mapa seleccionado por el usuario.

**-Cliente, “(menu:2,comando:[1/*2]*,estado:mapa)”**

**-Servidor, “(mapa:3,dato:*mapa[1/2].txt*, rango:*rango,* pos:*pos*)”**

2.3.2) Desarrollo del juego.

Llegado a este punto, el usuario debera resolver el laberinto, sorteando los obstaculos que lo esperan, moviendose con los comandos preestablecidos.

**Comandos: ["arriba", "abajo", "izquierda", "derecha", "agarrar", "salir", "w", "a", "s", "d", "e", "q"]**

**-Cliente, “(juego:1,comando:*comando*)”**

**-Servidor, “(valido:*[true/false]*,posX:*posX*, posY:*posY*,[error:*error*],[aviso:*aviso*], [finalizado:*true/false*])”**

En caso de que llegue a concluir el juego, cuando el cliente recibe el parametro “finalizado”, tendremos la opcion de Volver a jugar o Finalizar el programa.

2.3.3) Finalizar

Con esta accion cerramos la conexión entre el cliente y el servidor

**-Cliente, “” (notese que en este caso no enviamos el casiJSON)**

3) Seguridad:

Todas las comunicaciones que se realizan entre el cliente y servidor, una vez establecida la conexión, estan protegidas por medio de encriptacion de datos con el algoritmo AES.

El segundo metodo de seguridad, esta establecido por medio de un Loggin :D

4) Codigo Fuente:

Para correr el server y el cliente, usamos python 3.7 con los con los modulos estándar y el modulo pycryptodome

Codigo Fuente del servidor: servidor.py

import socket

import os

import time

import json

from jugador import Jugador

from Crypto.Cipher import AES

import base64

cfgTimeout = 0

lstComando = ["arriba", "abajo", "izquierda", "derecha", "agarrar", "salir",

"w", "a", "s", "d", "e", "q"]

secret\_key = 'a15fg7s9h75q17a8'.encode()

def encriptar(mensaje):

mensaje = mensaje.rjust(768)

cipher = AES.new(secret\_key, AES.MODE\_ECB)

msjCriptado = base64.b64encode(cipher.encrypt(mensaje.encode()))

if(len(msjCriptado) > 1024):

raise Exception("La encriptacion sobrepaso los 1024 bytes")

return msjCriptado

def desencriptar(mensajeEncriptado):

print("mensajeEncriptado: ", len(mensajeEncriptado))

cipher = AES.new(secret\_key, AES.MODE\_ECB)

msjDesencriptado = cipher.decrypt(base64.b64decode(mensajeEncriptado)).strip().decode()

print("msjDesencriptado: ", msjDesencriptado)

return msjDesencriptado

def checkJSON(jObjeto):

try:

dato = json.loads(jObjeto)

cantidad = 0

debeHaber = 0

for i in dato:

if(cantidad == 0):

debeHaber = dato[i]

cantidad = cantidad + 1

if(cantidad - 1 != debeHaber):

return False

return True

except ValueError:

print("No soy jayson")

return False

def run\_server():

os.system('cls')

sServer = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = ("localhost", 6666)

sServer.bind(server\_address)

sServer.listen(1)

sServer.settimeout(cfgTimeout)

print("Servidor en escucha")

salir = False

lstSCliente = []

lstClientAddress = []

lstJugadores = []

sCliente = None

cAddress = None

while salir is not True:

try:

# Aca Aceptamos las conexiones de los clientes y validamos usuarios

sCliente, cAddress = sServer.accept()

sCliente.settimeout(cfgTimeout)

if(sCliente is not None):

lstSCliente.append(sCliente)

lstClientAddress.append(cAddress)

lstJugadores.append(Jugador(sCliente, cAddress, "./Mapas/"))

sCliente = None

cAddress = None

except (socket.timeout, BlockingIOError):

pass

try:

atenderJugadores(lstJugadores)

finally:

# Cerrando conexion

# print("Me aseguro que cierra la conexion MAthov")

# sCliente.close()

# salir = True

pass

def atenderJugadores(lstJugadores):

for jugador in lstJugadores:

# Recibe los datos en trozos y reetransmite

try:

data = desencriptar(jugador.sock.recv(1024))

print("dato recibido: ", data)

if(data):

if (checkJSON(data)):

dicServer = json.loads(data)

print("dicServer: ", dicServer.get("login"))

if(dicServer.get("login") is not None):

preMsg = {}

preMsg["login"] = 1

preMsg["valido"] = jugador.crearJugador(dicServer)

sendMsg = json.dumps(preMsg)

msgEncript = encriptar(sendMsg)

jugador.sock.sendall(msgEncript)

elif(dicServer.get("menu") is not None):

com = dicServer.get("comando")

if(com is None or com in ["q", "salir"]):

# Menu Principal

preMsg = {}

preMsg["menu"] = 2

preMsg["dato"] = jugador.generarMenu()

preMsg["estado"] = "principal"

preMsg = json.dumps(preMsg)

print("Le mando el menu")

jugador.sock.sendall(encriptar(preMsg))

elif(dicServer.get("estado") == "principal"):

preMsg = {}

preMsg["menu"] = 2

if(com == "1"): # Menu MAPAS

preMsg["dato"] = jugador.generarMapas()

preMsg["estado"] = "mapas"

print("Le mando el menu de MAPAS")

if(com == "2"): # INSTRUCCINES

preMsg["dato"] = jugador.generarInstrucciones()

preMsg["estado"] = "instrucciones"

print("Le mando INSTRUCCINES")

if(com == "3"): # CREDITOS

preMsg["dato"] = jugador.generarCreditos()

preMsg["estado"] = "creditos"

print("Le mando CREDITOS")

print("Esto es lo que mando (comando): ", preMsg)

preMsg = json.dumps(preMsg)

jugador.sock.sendall(encriptar(preMsg))

if(dicServer.get("estado") == "mapas"):

preMsg = {}

preMsg["mapa"] = 4

preMsg["dato"] = jugador.traerMapa(com)

preMsg["rango"] = jugador.rango

preMsg["pos"] = jugador.pos

preMsg["oro"] = jugador.cantOro

preMsg = json.dumps(preMsg)

jugador.sock.sendall(encriptar(preMsg))

elif(dicServer.get("juego") is not None):

if(dicServer.get("comando") in lstComando):

if(dicServer.get("comando") in ("q", "salir")):

preMsg = {}

preMsg["menu"] = 2

preMsg["dato"] = jugador.generarMenu()

preMsg["estado"] = "principal"

preMsg = json.dumps(preMsg)

jugador.sock.sendall(encriptar(preMsg))

else:

preMsg = {}

preMsg["juego"] = cantParametros = 0

error, aviso, remp = jugador.controlarComando(dicServer.get("comando"))

if (error != ""):

if(error == "Muerte"):

preMsg["gameOver"] = "Perdiste :c"

else:

preMsg["error"] = error

cantParametros += 1

if(aviso != ""):

if(aviso == "Ganaste"):

preMsg["gameOver"] = "Ganaste :D"

else:

preMsg["aviso"] = aviso

cantParametros += 1

if(remp != ""):

preMsg["remplazo"] = remp

cantParametros += 1

preMsg["pos"] = jugador.pos

preMsg["oro"] = jugador.cantOro

preMsg["juego"] = cantParametros + 2

preMsg = json.dumps(preMsg)

jugador.sock.sendall(encriptar(preMsg))

else:

sendErr = {}

sendErr["juego"] = 3

sendErr["error"] = "Comando invalido"

sendErr["pos"] = jugador.pos

sendErr["oro"] = jugador.cantOro

errMsg = json.dumps(sendErr)

jugador.sock.sendall(encriptar(errMsg))

else:

sendErr = {}

sendErr["Error"] = 1

sendErr["Causa"] = "No sos un tipo valido de datos"

errMsg = json.dumps(sendErr)

jugador.sock.sendall(encriptar(errMsg))

else:

print('no hay mas datos', jugador.address)

jugador.sock.close()

lstJugadores.remove(jugador)

print("Cerrada la Conexion")

except (socket.timeout, BlockingIOError):

pass

time.sleep(0.05)

class DatoInvalido(Exception):

def \_\_init\_\_(self, valor):

self.valor = valor

def \_\_str\_\_(self):

return repr(self.valor)

"""

Documentar checkJSON

"""

if \_\_name\_\_ == '\_\_main\_\_':

run\_server()

Codigo Fuente del servidor: jugador.py

import os

RANGOVISTA = 4

class Jugador(object):

"""docstring for Jugador"""

def \_\_init\_\_(self, sCliente, clientAddress, carpetaMapas):

self.sock = sCliente

self.address = clientAddress

self.hasLlave = False

self.cantOro = 0

self.pos = None

self.rango = int(RANGOVISTA / 2)

self.usuario = ""

self.password = ""

self.valido = False

self.estado = "Desconectado"

self.mapa = ""

self.lstMapa = []

self.mapas = []

self.carpetaMapas = carpetaMapas

def generarCreditos(self):

return("Creditos a:\nCamila Mathov\n"

"Nicolas Mateus\nLucas Tisera")

def crearJugador(self, cadena):

#self.usuario, self.password = cortarEncabezado(cadena)

usvalido = False

self.usuario = cadena.get("ussr")

self.password = cadena.get("password")

if(validarUsuario(self.usuario, self.password)):

usvalido = True

return(usvalido)

def generarMenu(self):

return("Elegi Una Opcion\n1)Cargar mapa\n"

"2)Ver Instrucciones\n3)Creditos")

def traerMapa(self, mapa):

self.mapa, self.lstMapa = cargarMapa(self.carpetaMapas, self.mapas[int(mapa) - 1])

self.pos = posInicio(self.lstMapa)

return (self.mapa)

def generarMapas(self):

self.mapas = cargarListaMapas(self.carpetaMapas)

mandar = ""

for i in range(len(self.mapas)):

mandar += str(i + 1) + ") " + self.mapas[i].capitalize() + "\n"

return(mandar[:-1])

def generarInstrucciones(self):

return ("AWSD para moverse\nE para agarrar oro (O) o la llave (K) cu"

"ando te encuentres arriba\nN para salir\nEl objetivo es llega"

"r a la salida (S) habiendo conseguido la llave antes\nEl G e"

"s el guardia\nsi tratas de pasar por donde esta el sin tener"

" oro moris\nP = Pared C = Camino E = Entrada")

def controlarComando(self, comando):

nP = self.pos

error = aviso = remplazo = ""

if comando == "a":

nP = (self.pos[0], self.pos[1] - 1)

elif comando == "d":

nP = (self.pos[0], self.pos[1] + 1)

elif comando == "w":

nP = (self.pos[0] - 1, self.pos[1])

elif comando == "s":

nP = (self.pos[0] + 1, self.pos[1])

elif comando == "e":

if self.lstMapa[nP[0]][nP[1]] == "K":

self.hasLlave = True

self.lstMapa[nP[0]][nP[1]] = "C"

remplazo = nP

aviso = "Llave encontrada"

elif self.lstMapa[nP[0]][nP[1]] == "O":

self.cantOro += 1

self.lstMapa[nP[0]][nP[1]] = "C"

remplazo = nP

aviso = "Oro encontrado"

else:

aviso = "No hay nada que agarrar"

error, aviso, remplazo = self.controlarNPos(nP, aviso, remplazo)

if error == "":

self.pos = nP

return error, aviso, remplazo

def controlarNPos(self, pos, aviso, remplazo):

error = ""

if controlarFinDeMapa(pos, len(self.lstMapa), len(self.lstMapa[0])):

error = "Fin del Mapa"

elif self.lstMapa[pos[0]][pos[1]] != "C":

if self.lstMapa[pos[0]][pos[1]] == "P":

error = "No se pueden atravesar las paredes"

if self.lstMapa[pos[0]][pos[1]] == "G":

if self.cantOro <= 0:

error = "Muerte"

else:

self.cantOro -= 1

self.lstMapa[pos[0]][pos[1]] = "C"

remplazo = pos

aviso = "Le pagaste al guardia"

if self.lstMapa[pos[0]][pos[1]] == "S":

if self.hasLlave:

aviso = "Ganaste"

else:

error = "La salida esta cerrada"

return error, aviso, remplazo

"""

Funciones que utiliza la clase

"""

def controlarFinDeMapa(pos, lenX, lenY):

return (pos[0] >= lenX or pos[0] < 0 or pos[1] >= lenY or pos[1] < 0)

def validarUsuario(usuario, password):

s = open("usuarios.txt", "r")

for linea in s.readlines():

ussr, contra = cortarEncabezado(linea)

if(ussr == usuario and contra == password):

return True

return False

def cortarEncabezado(cadena):

usuario = cadena[6:cadena.find("|")]

password = cadena[cadena.find("|") + 6:].strip()

return (usuario, password)

def cargarListaMapas(carpeta):

m = []

try:

for archivo in os.listdir(carpeta):

nombre = os.path.join(carpeta, archivo)

if os.path.isfile(nombre):

if nombre.endswith(".txt"):

m.append((nombre.split(".txt")[0]).split(carpeta)[1])

except FileNotFoundError:

pass

return m

def cargarMapa(carpeta, mapa):

lista = []

mandar = ""

try:

f = open(carpeta + mapa + ".txt", "r")

if f.mode == 'r':

fl = f.readlines()

for x in fl:

mandar += str(x.strip()) + ","

lista.append(list(x.strip()))

except FileNotFoundError:

print("Archivo no encontrado\n")

return mandar[:-1], lista

def posInicio(lista):

for x in range(len(lista)):

try:

return x, lista[x].index("E")

except ValueError:

pass

print("Mapa corrupto\n")

if \_\_name\_\_ == '\_\_main\_\_':

print(validarUsuario("camila", "1234"))

print(validarUsuario("camOTORONO", "1234"))

Codigo fuente del Cliente: cliente.py

import socket

import os

import json

from Crypto.Cipher import AES

import base64

estado = "Desconectado"

secret\_key = 'a15fg7s9h75q17a8'.encode()

def encriptar(mensaje):

mensaje = mensaje.rjust(768)

cipher = AES.new(secret\_key, AES.MODE\_ECB)

msjCriptado = base64.b64encode(cipher.encrypt(mensaje.encode()))

if(len(msjCriptado) > 1024):

raise Exception("La encriptacion sobrepaso los 1024 bytes")

return msjCriptado

def desencriptar(mensajeEncriptado):

cipher = AES.new(secret\_key, AES.MODE\_ECB)

return cipher.decrypt(base64.b64decode(mensajeEncriptado)).strip().decode()

def checkJSON(jObjeto):

try:

dato = json.loads(jObjeto)

cantidad = 0

debeHaber = 0

for i in dato:

if(cantidad == 0):

debeHaber = dato[i]

cantidad = cantidad + 1

if(cantidad - 1 != debeHaber):

return False

return True

except ValueError:

print("No soy jayson")

return False

def crearConexion():

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Conecta el socket en el puerto cuando el servidor esté escuchando

server\_address = ('localhost', 6666)

print('conectando a ', server\_address)

sock.connect(server\_address)

return sock

def run\_cliente():

global estado

salir = False

while salir is not True:

mensaje = {}

if(estado == "Desconectado"):

sock = crearConexion()

os.system("cls")

estado = "Deslogueado"

elif(estado == "Deslogueado"):

print("Ingresa Usuario:")

usuario = input()

print("Ingresa password:")

password = input()

mensaje["login"] = 2

mensaje["ussr"] = usuario

mensaje["password"] = password

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

os.system("cls")

estado = "EsperandoLogin"

else:

data = desencriptar(sock.recv(1024))

if checkJSON(data):

data = json.loads(data)

if(estado == "EsperandoLogin"):

if (data.get("valido") is True):

estado = "Conectado"

mensaje["menu"] = 1

mensaje["estado"] = "principal"

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

else:

estado = "Deslogueado"

print("Usuario y/o contraseña incorrecto\n")

elif(estado == "Conectado"):

os.system("cls")

if(data.get("mapa") is not None):

mapa = []

rango = None

for linea in data.get("dato").split(","):

mapa.append(list(linea))

rango = int(data.get("rango"))

mensaje["juego"] = 1

mensaje["comando"] = "d"

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

estado = "Jugando"

elif(data.get("dato") is not None):

print(data.get("dato"))

eleccion = input()

mensaje = {}

mensaje["menu"] = 2

mensaje["comando"] = eleccion

mensaje["estado"] = data.get("estado")

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

elif(estado == "Jugando"):

os.system("cls")

if(data.get("juego") is not None):

if(data.get("gameOver") is None):

pos = int(data.get("pos")[0]), int(data.get("pos")[1])

imprimirMapa(rango, pos, mapa)

print("\n Oro Actual: " + str(data.get("oro")))

if(data.get("error") is not None):

print(" " + data.get("error"), end="\n")

if(data.get("aviso") is not None):

print(" " + data.get("aviso"), end="\n")

if(data.get("remplazo") is not None):

mapa = remplazar(data.get("remplazo"), mapa)

print("\n\n ¿Que desea hacer? ", end="")

comando = input().lower()

mensaje["juego"] = 1

mensaje["comando"] = comando

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

else:

print(data.get("gameOver") + "\n\n")

estado = "Conectado"

mensaje["menu"] = 1

mensaje["estado"] = "principal"

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

os.system("pause")

estado = "Conectado"

else:

estado = "Conectado"

mensaje["menu"] = 1

mensaje["estado"] = "principal"

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

estado = "Conectado"

else:

# No JSON

mensaje["Error"] = 1

mensaje["Causa"] = "No sos un tipo valido de datos"

mensaje = json.dumps(mensaje)

sock.sendall(encriptar(mensaje))

sock.close()

def remplazar(json, lista):

lista[int(json[0])][int(json[1])] = "C"

return lista

def imprimirMapa(r, pos, lista):

# Imprimir posicion

linea1 = linea2 = linea3 = " "

for y in range(len(lista[0])):

if y < 10:

linea1 += "0"

else:

linea1 += str(int(y / 10))

linea2 += str(y % 10)

linea3 += "\_"

print(linea1 + "\n" + linea2 + "\n" + linea3)

# Imprimir mapa

for x in range(len(lista)):

if x < 10:

print("0", end="")

print(str(x) + " |", end="")

for y in range(len(lista[x])):

if (x >= (int(pos[0]) - r) and x <= (int(pos[0]) + r) and

y >= (int(pos[1]) - r) and y <= (int(pos[1]) + r)):

if x == pos[0] and y == pos[1]:

print("J", end="")

else:

print(lista[x][y], end="")

else:

print(" ", end="")

print("")

if \_\_name\_\_ == '\_\_main\_\_':

run\_cliente()

Anexo – Correcciones

|  |  |  |
| --- | --- | --- |
| Prueba | Resultado | Comentario |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Anexo – Correcciones (copia para los alumnos)

|  |  |  |
| --- | --- | --- |
| Prueba | Resultado | Comentario |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |