

Escuela de Ciencias y Sistemas

Estructura de Datos

Proyecto – Fase 2
Manual Técnico

César André Ramírez Dávila

202010816

Fecha: 03/09/2022

Indice

<i>Introducción</i>	3
<i>Objetivos</i>	3
<i>Requerimientos</i>	4
<i>Creación del programa</i>	5
1. Librerías	5
1.1. Archivos de Cabecera	5
1.2. Archivo de Entrada	6
2. Servidor	6
3. Main.cpp	10
3.1. Conexión Servidor	11
4. Cliente.py	14
4.1. Registro Usuarios	15
4.2. Login	16
4.3. Administrador	18
4.4. Usuarios	19
5. Tablero de Juego	22
6. Salida	24

Introducción

El presente documento describe los aspectos técnicos informáticos del programa. El documento familiariza al desarrollador que hace uso del lenguaje C++, Python y una conexión entre estos 2 lenguajes, con temas como: uso de nodos, listas, ordenamientos, ciclos repetitivos, cargas de archivos con muchos datos, registros, graficas, crear tablero de juegos, jugadas, entre otros.

Aprendiendo como usar el algoritmo de hash de seguro de 256 bits para la seguridad en las contraseñas de los usuarios.

Objetivos

- Instruir el uso adecuado del Sistema de información, para el acceso adecuado en el uso de este, mostrando los pasos de desarrollo del programa, así como la descripción de las funciones y métodos usados para la realizacion del programa.
- Comprender uso de estructuras de datos en el lenguaje C++
- Comprender uso de Interfaces graficas en el lenguaje de Python
- Obtener mayor conocimiento en el lenguaje C++ y Python
- Aprender el uso de glove para la comunicación entre C++ y Python

El presente manual está enfocado en la comunicacion entre lenguaje de programación C++ combinado con Python.

Requerimientos

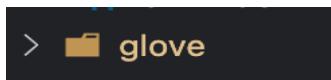
- La aplicación puede ser ejecutada únicamente en distribuciones de Linux y MacOS, debido a que se usa la herramienta glove para la combinación entre esos lenguajes, tener configurado el compilador de C++ en el sistema, en este caso el compilador usado es: mingw-w64.
- IDE recomendado: Visual Studio Code
- Equipo Intel Pentium o superior
- Espacio en el disco duro, al menos 500 mb
- Memoria ram recomendada 4gb (debido al servidor)

Creación del programa

1. Librerías

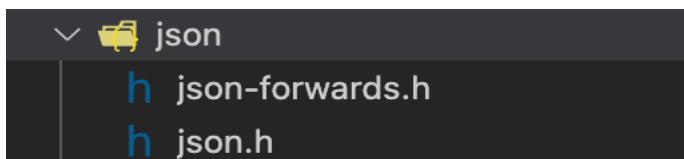
Para poder realizar la comunicación entre C++ y Python se usa la librería glove, esta debe ser clonada en nuestra carpeta del proyecto.

Enlace: <https://github.com/gasparfm/glove>



Las siguientes librerías son requeridas para hacer la carga de un archivo.json y encriptamiento con SHA256.

Se necesita la siguiente carpeta de json dentro del proyecto, para poder hacer uso de la librería.



Librería de SHA256.h



1.1. Archivos de Cabecera

```
#include "./glove/glovehttpserver.hpp"
#include <iostream>
#include <chrono>
#include <thread>
#include <string>
#include <vector>
#include <fstream>
#include "./glove/json.hpp"
#include "C++/jsoncpp.cpp"
#include "C++/ListaCircular.cpp"
#include "C++/ListaArticulos.cpp"
#include "C++/ListaTutorial.cpp"
#include "C++/ArbolB.cpp"
#include "C++/ListaPila.cpp"
```

1.2. Archivo de Entrada

```
{  
    "usuarios": [  
        { "id": "1", "nick": "andre", "password": "card", "monedas": "200", "edad": "20" },  
        { "id": "2", "nick": "luis", "password": "elpana", "monedas": "100", "edad": "40" },  
        { "id": "3", "nick": "angel", "password": "siquea", "monedas": "50", "edad": "21" },  
        { "id": "4", "nick": "pedro", "password": "pedrovrm", "monedas": "350", "edad": "10" }  
    ],
```

Se tiene un arreglo de usuario, este dentro contiene el id, nick, contraseña, monedas y la edad.

```
"articulos": [  
    { "id": "1", "categoria": "legendario", "precio": "500", "nombre": "barco1", "src": "link1" },  
    { "id": "2", "categoria": "epic", "precio": "400", "nombre": "barco2", "src": "link2" },  
    { "id": "3", "categoria": "basico", "precio": "100", "nombre": "barco3", "src": "link3" },  
    { "id": "4", "categoria": "basico", "precio": "300", "nombre": "barco4", "src": "link4" }  
],
```

Se tiene un arreglo de artículos, en el cual dentro contiene el id, categoría, precio, nombre y un enlace a una imagen.

```
"Tutorial": {  
    "ancho": "20",  
    "alto": "30",  
    "movimientos": [  
        { "x": "1", "y": "6" },  
        { "x": "3", "y": "9" },  
        { "x": "5", "y": "1" },  
        { "x": "8", "y": "3" },  
        { "x": "1", "y": "0" }  
    ] }
```

Por último, tenemos un arreglo dentro de otro, donde primero viene las dimensiones del tablero y seguido los movimientos realizados.

2. Servidor

Para realizar la conexión de los lenguajes, es necesario tener archivos header ".h" para la correcta comunicación con glove.

C++ ArbolB.cpp

h ArbolB.h

Ejemplo de archivo .h

Nodo para los usuarios

```
#ifndef NODOUSUARIO_H
#define NODOUSUARIO_H
#include <stddef.h>
#include <string>

class nodoUsuarios{
public:
    std::string nombreuser;
    std::string contra;
    std::string contracifrada;
    int id,monedas,edad;
    nodoUsuarios *anterior;
    nodoUsuarios *siguiente;
    nodoUsuarios(){
        anterior = NULL;
        siguiente = NULL;
        id = 0;
        nombreuser = ' ';
        contra = ' ';
        contracifrada = ' ';
        monedas = 0;
        edad = 0;
    }
private:
};
#endif
```

Creando la lista para usuarios

```
#ifndef LISTACIRCULAR_H
#define LISTACIRCULAR_H

#include "NodoUsuarios.h"
#include <iostream>
using namespace std;

class ListaCircular{
public:
    nodoUsuarios*primero;
    nodoUsuarios*ultimo;
    ListaCircular(){
        primero = NULL;
        ultimo = NULL;
    }
```

Métodos que tendrá la lista

```
void registro_usuario(int id,string nombreuser, string contracifrada, int monedas ,int edad, string contra);
void registro_usuarioJ(int id,string nombreuser, string contracifrada, int monedas ,int edad, string contra);
void lista_usuarios();
void ordenarUsuarioASC();
void ordenarUsuarioDESC();
void ListaUsuarioASC(nodoUsuarios *cabeza);
void ListaUsuarioDESC(nodoUsuarios *cabeza);
void intercambioASC(nodoUsuarios *lado_izq, nodoUsuarios *lado_der);
void intercambioDESC(nodoUsuarios *lado_izq, nodoUsuarios *lado_der);
void HtmLASC(nodoUsuarios *cabeza);
void HtmlDESC(nodoUsuarios *cabeza);
void eliminarCuenta(string userbuscado);
void modificarUsuario(string userb, string nuevouser, string contra, int edad, string cifrada);
string BuscarUser(string nombreuser);
string getUsers();
string Comprobar(string nombreuser);
string Comprobar1(string nombreuser);
string verificarLog(string usuariob, string cifrada);
string Buscar1(string username);
```

```
private:
};

#endif
```

Los métodos para agregar usuarios, son los mismos descritos en el manual de la Fase 1

Los nuevos son: getUsers()

```
string ListaCircular::getUsers() {
    nodoUsuarios*aux = primero;
    string datos = "";
    datos += "{\"usuario\":[";

    while (aux != NULL){
        datos += "{";
        datos+= "\"id\":\"" + to_string(aux->id) + "\",";
        datos+= "\"nick\":\"" + aux->nombreuser + "\",";
        datos+= "\"password\":\"" + aux->contracifrada + "\",";
        datos+= "\"monedas\":\"" + to_string(aux->monedas) + "\",";
        datos+= "\"edad\":\"" + to_string(aux->edad) + "\"";
        datos += "}";

        aux = aux->siguiente;
        if(aux!=primero){
            datos+= ",";
        }
        if(aux == primero){
            break;
        }
    }

    datos += "]";
    return datos;
}
```

Creamos un string con formato json que nos servirá para trabajar con los datos en Python más adelante.

Así sucesivamente con todas las estructuras de datos que se utilizan en el proyecto.

3. Main.cpp

Este archivo será la fuente de comunicación con Python.

```
int contadorusuarios = 0;
ListaCircular ListaUsuarios;
ListaArticulos ListaArt;
ListaTutorial ListTutorial;
ArbolB Arbol;
ListaPila Movimientos;
```

Se define un contador para los usuarios y las listas para almacenar datos creadas previamente.

```
int atoi(std::string s)
{
    try
    {
        return std::stod(s);
    }
    catch (std::exception &e)
    {
        return 0;
    }
}
```

El método llamado atoi nos permite pasar datos de tipo string a formato int.

```
static std::string jsonkv(std::string k, std::string v)
{
    /* "k": "v" */
    return "\"" + k + "\": \"" + v + "\"";
}
```

Esto nos permite retornar una respuesta en formato json.

3.1. Conexión Servidor

```
class Servidor
{
public:
    Servidor()
    {
        string usuario = "EDD";
        string contra = "edd123";
        string encriptado = SHA256::cifrar(contra);
        ///cout<<"encriptado: "<<encriptado<<"\n";

        ListaUsuarios.registro_usuario(contadorusuarios,usuario,encriptado,0,50,contra);
        Arbol.insertar(contadorusuarios,usuario);
        contadorusuarios++;
    }
}
```

Al iniciar el servidor se agrega el usuario administrador.

```
void get(GloveHttpRequest &request, GloveHttpResponse &response)
{
    string rut;
    response.contentType("text/json");
    rut = request.special["Ruta"];
    if (rut.empty())
        response << ListaUsuarios.getUsers();
    else
    {
        response << "{ "
        << jsonkv("status", "ok tengo el get") << ",\n"
        " }";
    }
}
```

Se crea un método get para obtener la ruta del archivo a cargar.

```

void post(GloveHttpRequest &request, GloveHttpResponse &response)
{
    ifstream archivo;
    string ruta;
    string pruebaa;
    string texto;
    string iduser,nombreuser,contra,monedas,edad;
    string idarticoloo, categoriarticulo,precioarticoloo,nombrearticulo,srcarticulo;
    string alt, anch, x1, y1;
    ruta = request.special["Ruta"];
    archivo.open(ruta.c_str(), ios::in);
    if(archivo.fail()){
        cout<<"\nNo se pudo abrir el archivo\n"<<endl;
    }
}

```

Hacemos un método post para la lectura del archivo, similar a la de la fase 1.

```

while (!archivo.eof())
{
    Json::Reader reader;
    Json::Value obj;
    reader.parse(archivo, obj);
    const Json::Value& usuariosJ = obj["usuarios"];
    for (int i = 0; i < usuariosJ.size(); i++){
        iduser = usuariosJ[i]["id"].asString();
        //cout << "\nNick: " << usuariosJ[i]["nick"].asString();
        nombreuser = usuariosJ[i]["nick"].asString();
        //cout << "\nPass: " << usuariosJ[i]["password"].asString();
        contra = usuariosJ[i]["password"].asString();
        string encriptado = SHA256::cifrar(contra);
        //cout<<"El cifrado sha es : "<<encriptado<<endl;
        //cout << "\nMonedas: " << usuariosJ[i]["monedas"].asString();
        monedas = usuariosJ[i]["monedas"].asString();
        //cout << "\nEdad: " << usuariosJ[i]["edad"].asString();
        edad = usuariosJ[i]["edad"].asString();
        std ::string idi = iduser;
        std ::string edadi = edad;
        std ::string monedasi = monedas;
        int iddi = std::stoi(idi);
        int eddi = std::stoi(edadi);
        int monedi = std::stoi(monedasi);
        ListaUsuarios.registro_usuarioJ(iddi,nombreuser, encriptado, monedi, eddi, contra);
    }
}

```

Así sucesivamente con los artículos y el tutorial del Juego.

```

const Json::Value& articulosJ = obj["articulos"];
for (int i = 0; i < articulosJ.size(); i++){
    //cout << "\nID: " << articulosJ[i]["id"].asString();
    idarticuloo = articulosJ[i]["id"].asString();
    //cout << "\nCategoria: " << articulosJ[i]["categoria"].asString();
    categoriarticulo = articulosJ[i]["categoria"].asString();
    //cout << "\nPrecio: " << articulosJ[i]["precio"].asString();
    precioarticulo = articulosJ[i]["precio"].asString();
    //cout << "\nNombre: " << articulosJ[i]["nombre"].asString();
    nombrearticulo = articulosJ[i]["nombre"].asString();
    //cout << "\nSRC: " << articulosJ[i]["src"].asString();
    srcarticulo = articulosJ[i]["src"].asString();
    std ::string iarticulo = idarticuloo;
    std ::string precioarticul = precioarticulo;
    int precioarticulo = std::stoi(precioarticul);
    ListaArt.registro_articulos(categoriarticulo,nombrearticulo,precioarticulo,idarticuloo,srcarticulo);
    //cout << endl;
}

```

```

const Json::Value& tutorialJ = obj["tutorial"];
//cout <<"\nAncho: "<<tutorialJ["ancho"].asString();
anch = tutorialJ["ancho"].asString();
//cout<<"\nAlto: "<<tutorialJ["alto"].asString();
alt = tutorialJ["alto"].asString();
std ::string ancho = anch;
std ::string alto = alt;
int x = std::stoi(ancho);
int y = std::stoi(alto);
ListTutorial.registroTutorial(x,y);
//cout<<"\nMovimientos: ";
const Json::Value& movimientosJ = tutorialJ["movimientos"];
for(int i = 0; i < movimientosJ.size(); i++){
    //cout << "\nX: " << movimientosJ[i]["x"].asString();
    x1 = movimientosJ[i]["x"].asString();
    //cout << " Y: " << movimientosJ[i]["y"].asString();
    y1 = movimientosJ[i]["y"].asString();
    int x = std::stoi(x1);
    int y = std::stoi(y1);
    ListTutorial.registroTutorial(x,y);
}
cout<<"\n";
cout<<"\nArchivo cargado con exito\n"<<endl;
break;
}
archivo.close();

```

```

response << "{ "
    << jsonkv("status", "ok ha sido enviado") << ",\n"
    " }";
return;
}

```

Si el intercambio de datos es correcto, retornamos un mensaje de éxito.

4. Cliente.py

```

ventana = Tk()
ventana.title("Proyecto Fase 2 - 202010816")
ventana.resizable(0,0)
ancho_ventana = 500
alto_ventana = 500
x_ventana = ventana.winfo_screenwidth() // 2 - ancho_ventana // 2
y_ventana = ventana.winfo_screenheight() // 2 - alto_ventana // 2
posicion = str(ancho_ventana) + "x" + str(alto_ventana) + "+" + str(x_ventana) + "+" + str(y_ventana)
ventana.geometry(posicion)
#Botones
btnCargarArchivo = Button(ventana, height=2, width=15, text="Cargar Usuarios", command = abrirArchivo1, background="#B03314", fg="white")
btnCargarArchivo.place(x=180, y=150)
btnRegistrar = Button(ventana, height=2, width=15, text="Registrar Usuario", command=lambda:[ ventana.withdraw(),ventanaLog.deiconify()])
btnRegistrar.place(x=180, y=210)
btnLogin = Button(ventana, height=2, width=15, text="Login",command=lambda:[ventana.withdraw(),ventanaLog.deiconify()])
btnLogin.place(x=180, y=270)
btnSalir = Button(ventana, height=2, width=15, text="Salir",command=cerrar, background="#B03314", font=("Verdana",10))
btnSalir.place(x=180, y=330)
#Labels
labelEditor = Label (ventana, text ="BATALLA NAVAL", font=("Verdana",16), background="#044D9A", fg="white")
labelEditor.place(x=180, y=90)

```

Empezamos con la interfaz gráfica en Python, luego llamamos a la función que tendrá el servidor para el intercambio de datos.

```

base_url = "http://127.0.0.1:8080/"

def abrirArchivo1():
    try:
        global archivo
        archivo = filedialog.askopenfilename(title="Seleccionar archivo", filetypes=[("json", "*.json")])
        #print(archivo)
        prueba = os.path.split(archivo)
        #print(prueba)
        res = requests.post(f'{base_url}/Carga/' + f'{prueba[1]}')
        data = res.text#convertimos la respuesta en dict
        print(data)
        MessageBox.showinfo("Exito!", "Archivo Cargado con exito")
    except:
        MessageBox.showwarning("Alerta", "Debe cargar un archivo")

```

Se tiene la url de conexión y se hace la petición al servidor.

4.1. Registro Usuarios

```
ventanaReg = Toplevel()
ventanaReg.title("Registro Usuario")
ventanaReg.resizable(0,0)
ancho_ventana1 = 500
alto_ventana1 = 500
x_ventana1 = ventanaReg.winfo_screenwidth() // 2 - ancho_ventana1 // 2
y_ventana1 = ventanaReg.winfo_screenheight() // 2 - alto_ventana1 // 2
posicion1 = str(ancho_ventana1) + "x" + str(alto_ventana1) + "+" + str(x_ventana1) + "+" + str(y_ventana1)
ventanaReg.geometry(posicion1)
labelLogin = Label (ventanaReg, text ="Registro", font=("Verdana",16), background="#044D9A", fg="white")
labelLogin.place(x=210, y=80)
labelUser = Label (ventanaReg, text ="Ingrese Usuario", font=("Verdana",16), background="#044D9A", fg="white")
labelUser.place(x=50, y=200)
labelPass = Label (ventanaReg, text ="Ingrese Contraseña", font=("Verdana",16), background="#044D9A", fg="white")
labelPass.place(x=50, y=260)
labelEdad = Label (ventanaReg, text ="Ingrese Edad", font=("Verdana",16), background="#044D9A", fg="white")
labelEdad.place(x=50, y=320)
textoUsuario = Text(ventanaReg, height=2, width=30, fg="white", font=("Consolas", 11))
textoUsuario.place(x=230, y=190)
textoPass = Text(ventanaReg, height=2, width=30, fg="white", font=("Consolas", 11))
textoPass.place(x=230, y=255)
textoEdad = Text(ventanaReg, height=2, width=30, fg="white", font=("Consolas", 11))
textoEdad.place(x=230,y=330)
btnRegistro = Button(ventanaReg, height=2, width=15, text="Registrar", command = lambda:[Comprobar(textoUsuario.get(1,END), textoPass.get(1,END), textoEdad.get(1,END))])
btnRegistro.place(x=180, y=400)
btnRegreso = Button(ventanaReg, height=2, width=8, text="Regresar", command = lambda:[textoUsuario.delete(1.0, tk.END), textoPass.delete(1.0, tk.END), textoEdad.delete(1.0, tk.END)])
btnRegreso.place(x=405, y=5)
ventanaReg.withdraw()
```

Se crea una ventana y luego se llama a la función que nos registrará el usuario en C++.

```
def Comprobar(salida,salida2,salida3):
    res = requests.get(f'{base_url}/Verificar/' + f'{salida}' + "/" + f'{salida2}' + "/" + f'{salida3}')
    data = res.text#convertimos la respuesta en dict

    datos_diccionarioo = json.loads(data)
    #idd1 = datos_diccionario2["Id"]
    estt = datos_diccionarioo["estado"]
    #print(data)

    if estt == "existe":
        MessageBox.showinfo("Problema", "Ya existe un usuario con este Nick")
    if estt == "no existe":
        mandarRegistro(salida,salida2,salida3)
        textoUsuario.delete(1.0, tk.END+"-1c")
        textoPass.delete(1.0, tk.END+"-1c")
        textoEdad.delete(1.0, tk.END+"-1c")
        ventanaReg.withdraw()
```

Haciendo una petición donde verifica si el usuario a ingresar ya existe o no, en caso de que no exista no nos permite crearlo y nos muestra una advertencia, de lo contrario hace el registro de forma exitosa.

```

def mandarRegistro(salida,salida2,salida3):
    res = requests.get(f'{base_url}/Registro/' + f'{salida}' + "/" + f'{salida2}' + "/" + f'{salida3}')
    data = res.text#convertimos la respuesta en dict
    cerrandoRegistro()

def cerrandoRegistro():
    #print("llegando")
    MessageBox.showinfo("Exito!", "Usuario Registrado con exito")
    ventana.deiconify()

```

Llamando a otras funciones para completar el registro.

4.2. Login

```

ventanaLog = Toplevel()
ventanaLog.title("Login")
ventanaLog.resizable(0,0)
ancho_ventana1 = 500
alto_ventana1 = 500
x_ventana1 = ventanaLog.winfo_screenwidth() // 2 - ancho_ventana1 // 2
y_ventana1 = ventanaLog.winfo_screenheight() // 2 - alto_ventana1 // 2
posicion1 = str(ancho_ventana1) + "x" + str(alto_ventana1) + "+" + str(x_ventana1) + "+" + str(y_ventana1)
ventanaLog.geometry(posicion1)
bgL = PhotoImage(file="Python/user.png")
labelPhotoL = Label(ventanaLog, image=bgL)
labelPhotoL.place(x=150,y=30)
labelLogin = Label (ventanaLog, text ="Login", font=("Verdana",16), background="#044D9A", fg="white")
labelLogin.place(x=230, y=200)
labelUser = Label (ventanaLog, text ="Ingrese Usuario", font=("Verdana",16), background="#044D9A", fg="white")
labelUser.place(x=40, y=280)
labelPass = Label (ventanaLog, text ="Ingrese Contraseña", font=("Verdana",16), background="#044D9A", fg="white")
labelPass.place(x=40, y=340)
textoUsuarioL = Text(ventanaLog, height=2, width=30, fg="white", font=("Consolas", 12))
textoUsuarioL.place(x=220, y=270)
textoPassL = Entry(ventanaLog, fg="white", show="*", font=("Consolas", 12), width=30)
textoPassL.place(x=220, y=340)
btnIniciar = Button(ventanaLog, height=2, width=15, text="Iniciar Sesion", command = lambda:[mandarLogin(textoUsuarioL.get(), textoPassL.get())])
btnIniciar.place(x=110, y=410)
btnRegresar = Button(ventanaLog, height=2, width=15, text="Regresar", command = lambda:[ventana.deiconify()])
ventanaLog.withdraw()

```

Creamos la ventana y por cedemos a llamar a la función que nos verifica los datos del login.

```

def mandarLogin(usuario, contra):
    global usuariobusqueda, monedasusuario
    print(usuario)
    print(contra)
    res = requests.get(f'{base_url}/Login/' + f'{usuario}' + "/" + f'{contra}')
    data = res.text#convertimos la respuesta en dict
    print(data)
    if data == "admin":
        ventanaLog.withdraw()
        MessageBox.showinfo("Exito!", "Inicio de sesion correcto")
        ventanaAdmin.deiconify()
    if data == "correcto":
        ventanaLog.withdraw()
        MessageBox.showinfo("Exito!", "Inicio de sesion correcto")
        res = requests.get(f'{base_url}/Log/' + f'{usuario}' + "/" + f'{contra}')
        data = res.text
        datos_diccionario = json.loads(data)
        name = datos_diccionario["nick"]
        passw = datos_diccionario["password"]
        edadd = datos_diccionario["edad"]
        monedd = datos_diccionario["monedas"]
        textoUsuarioC.insert(INSERT, name)
        textoPassC.insert(INSERT, passw)
        textoEdadC.insert(INSERT, edadd)
        usuariobusqueda = name
        monedasusuario = int(monedd)
        ventanaUser.deiconify()
    if data == "incorrecto":
        MessageBox.showinfo("Error!", "Usuario o contraseña incorrectos")
        ventanaLog.deiconify()
    if data == "inexistente":
        MessageBox.showinfo("Inesperado!", "El usuario no existe")
        ventanaLog.deiconify()

```

Hacemos una petición al servidor donde mandamos los datos y comprobamos que existan en la lista guardada.

En este caso como hay un usuario administrador nuestro servidor debe verificar si este es el administrador o un usuario normal, en caso sea administrador abre la ventana para este, sino abre la ventana para usuarios normales.

4.3. Administrador

```
ventanaAdmin = Toplevel()
ventanaAdmin.title("ADMINISTRADOR")
ventanaAdmin.resizable(0,0)
ancho_ventana1 = 500
alto_ventana1 = 500
x_ventana1 = ventanaAdmin.winfo_screenwidth() // 2 - ancho_ventana1 // 2
y_ventana1 = ventanaAdmin.winfo_screenheight() // 2 - alto_ventana1 // 2
posicion1 = str(ancho_ventana1) + "x" + str(alto_ventana1) + "+" + str(x_ventana1) + "+" + str(y_ventana1)
ventanaAdmin.geometry(posicion1)
bg = PhotoImage(file="Python/admin.png")
labelPhoto = Label(ventanaAdmin, image=bg)
labelPhoto.place(x=150,y=30)
btnUser = Button(ventanaAdmin, height=2, width=22, text="Lista de usuarios", command = lambda: [verusuario()], background="white", foreground="black")
btnUser.place(x=160, y=200)
btnArticulos = Button(ventanaAdmin, height=2, width=22, text="Lista de articulos", command = lambda: [verArticulos()], background="white", foreground="black")
btnArticulos.place(x=160, y=250)
btnOrdenAsc = Button(ventanaAdmin, height=2, width=22, text="Usuarios ordenados ascendente", command = lambda: [verusuariosordenadoASC()], background="white", foreground="black")
btnOrdenAsc.place(x=160, y=300)
btnOrdenDesc = Button(ventanaAdmin, height=2, width=22, text="Usuarios ordenados descendente", command = lambda: [verusuariosordenadoDESC()], background="white", foreground="black")
btnOrdenDesc.place(x=160, y=350)
btnTutorial = Button(ventanaAdmin, height=2, width=22, text="Tutorial del juego", command = lambda: [Tutorial()], background="white", foreground="black")
btnTutorial.place(x=160, y=400)
btnCerrarSesionAdmin = Button(ventanaAdmin, height=2, width=22, text="Cerrar Sesion", command = lambda: [ventanaLog.delete()])
btnCerrarSesionAdmin.place(x=160, y=450)
ventanaAdmin.withdraw()
```

Creamos la ventana, pero en este caso el administrador tiene la posibilidad de acceder a todos los reportes, entonces cada función es una petición al servidor.

```
def verusuario():
    res = requests.get(f'{base_url}/Usuarios/')
    data = res.text#convertimos la respuesta en dict
    #print(data)
    im = Image.open('Arbol.png')
    im.show()
```

Para ver a los usuarios en el Árbol B, se hace una petición que devuelve una imagen con el Árbol y se procede a abrir automáticamente.

```
def verusuarioASC():
    res = requests.get(f'{base_url}/UsuariosASC/')
    data = res.text#convertimos la respuesta en dict
    #print(data)
```

4.4. Usuarios

```
ventanaUser = Toplevel()
ventanaUser.title("Principal")
ventanaUser.resizable(0,0)
ancho_ventana1 = 500
alto_ventana1 = 500
x_ventana1 = ventanaUser.winfo_screenwidth() // 2 - ancho_ventana1 // 2
y_ventana1 = ventanaUser.winfo_screenheight() // 2 - alto_ventana1 // 2
posicion1 = str(ancho_ventana1) + "x" + str(alto_ventana1) + "+" + str(x_ventana1) + "+" + str(y_ventana1)
ventanaUser.geometry(posicion1)
bgUser = PhotoImage(file="Python/usuario.png")
labelPhotoUser = Label(ventanaUser, image=bgUser)
labelPhotoUser.place(x=150,y=30)
btnEditar = Button(ventanaUser, height=2, width=15, text="Editar Informacion", command = lambda: [ventanaUser.withdraw(), EliminarCuenta()])
btnEditar.place(x=180, y=200)
btnEliminarCuenta = Button(ventanaUser, height=2, width=15, text="Eliminar mi cuenta", command = lambda: [EliminarCuenta()])
btnEliminarCuenta.place(x=180, y=250)
btnVerTutorial = Button(ventanaUser, height=2, width=15, text="Mostrar Tutorial", command = lambda: [Tutorial()])
btnVerTutorial.place(x=180, y=300)
btnVerTienda = Button(ventanaUser, height=2, width=15, text="Tienda", command = lambda: [MostrarTienda()])
btnVerTienda.place(x=180, y=350)
btnPartida = Button(ventanaUser, height=2, width=15, text="Iniciar Partida", command = lambda: [ventanaObtenerDimensiones()])
btnPartida.place(x=180, y=400)
btncerrarSesionUser = Button(ventanaUser, height=2, width=15, text="Cerrar Sesion", command = lambda: [ventanaUser.withdraw()])
btncerrarSesionUser.place(x=180, y=450)
ventanaUser.withdraw()
```

Creamos la ventana para usuarios normales, con sus distintas opciones, entre ellas la de jugar una partida.

```
def EliminarCuenta():
    res = requests.get(f'{base_url}/Eliminar/' + f'{usuariobusqueda}')
    data = res.text

    #print(data)
    datos_diccionario1 = json.loads(data)
    idd = datos_diccionario1["Id"]
    est = datos_diccionario1["estado"]
    if est == "encontrado":
        res = requests.get(f'{base_url}/Eliminando/' + f'{usuariobusqueda}' + "/" + f'{idd}')
        data = res.text
        mes = MessageBox.askquestion('Eliminar cuenta', '¿Esta seguro de eliminar esta cuenta?')
        if mes == 'yes':
            MessageBox.showinfo('Cerrando Sesion', 'La cuenta ha sido eliminada')
            ventanaUser.withdraw()
            ventanaLog.deiconify()
        else:
            MessageBox.showinfo('Regresar', 'Regresando al menu')
```

Para eliminar la cuenta, se le hace una advertencia al usuario de estar seguro de eliminarla.

```

def MostrarTienda():
    #print("Tienda")
    print(monedasusuario)
    global ides
    res = requests.get(f'{base_url}/Tienda/')
    data = res.text#convertimos la respuesta en dict
    #print(data)
    articulos = json.loads(data)

    idAr = []
    nombresAr = []
    cateAr =[]
    precioAr = []
    for articulo in articulos:
        ides = articulo.get('Id')
        nombr = articulo.get('nombre')
        catt = articulo.get('categoria')
        prec = articulo.get('precio')
        #print(articulo.get('Id'))
        #print(articulo.get('categoria'))
        #print(articulo.get('precio'))
        #print(articulo.get('nombre'))

        idAr.append(ides)
        nombresAr.append(nombr)
        cateAr.append(catt)
        precioAr.append(prec)

    fig = go.Figure(data=[go.Table(
        header=dict(values=['ID','Nombre','Categoria','Precio']),
        cells=dict(values=[idAr,nombresAr,cateAr,precioAr
                           ]))
    ])
    fig.update_layout(title = "Tienda", title_x=0.5)
    fig.show()

```

Mostrado la tienda

```

def Partida():
    global Portaaviones, Submarino, Destructores, Buques
    Portaav = 1
    Subma = 2
    Destruc = 3
    Buq = 4
    # Formula para determinar la cantidad de barcos por tablero
    # B(m) = ((m-1)/10)+1
    dimens = textoDimension.get(1.0, tk.END+"-1c")
    dimen = int(dimens)

```

Antes de empezar la partida se debe crear el tablero de juego, para ello obtenemos las dimensiones para este.

```

if dimen<10:
    MessageBox.showerror("Advertencia", "El Numero minimo para el tablero es de 10")
    if dimen > 10:

```

Si la dimensión es menor a 10 no se permite crear.

```

if dimen==10:
    ventanaObtenerDimension.withdraw()
    textoDimension.delete(1.0, tk.END+"-1c")
    vidas = 3
    B = int((dimen-1)/10)+1
    #print(B)
    Portaaviones = Portaav * B
    Submarino = Subma * B
    Destructores = Destruc * B
    Buques = Buq * B
    print("Portaaviones: ", Portaaviones)
    print("Submarinos : ", Submarino)
    print("Destructores: ", Destructores)
    print("Buques : " , Buques)
    MessageBox.showinfo("Exito", "Tablero creado con exito")
    Llamado(dimen,Buques,monedasusuario,vidas)

```

Si la dimensión es permitida, procede a crear el tablero.

```
def LLamado(dimension,portaa,monedas,vida):
    Busca = prueba(dimension,portaa)
    plt.connect('button_press_event', Busca.on_click)
    plt.ion()
    Busca.Pintar(portaa,monedas,vida)
    plt.draw()
    while Busca.Estado == "":
        plt.pause(0.1)
    plt.ioff()
    plt.show()
```

Graficando el tablero con la librería Matplotlib

5. Tablero de Juego

```
class Casilla:
    def __init__(self):
        self.Visible = False
        self.TieneBuque = False
        self.NumBuquesAdyacentes = 0
```

El tablero lleva la clase Casilla

```

class prueba:
    def __init__(self, tam, numBuques):
        self.Tamano = tam
        self.Tablero = []
        self.Pendientes = tam*tam
        self.Estado = ""
        self.XError = None
        self.YError = None
        for fila in range(tam):
            f = []
            for j in range(tam):
                f.append(Casilla())
            self.Tablero.append(f)
        #print(self.Tablero.append(f))
        num = 0
        while num < numBuques:
            rndx = random.randint(0,tam-1)
            rndy = random.randint(0,tam-1)
            print("Y: ", rndx+1, "X: ", rndy+1)
            #print("X: ", rndy+1)
            #print("X: ", rndx, "Y: ", rndy)
            if not self.Tablero[rndx][rndy].TieneBuque:
                self.Tablero[rndx][rndy].TieneBuque = True
                filaIni = max(rndx-1,0)
                filaFin = min(rndx+1,tam-1)
                colIni = max(rndy-1,0)
                colFin = min(rndy+1,tam-1)
                for i in range(filaIni, filaFin+1,1):
                    for j in range(colIni,colFin+1,1):
                        if i !=rndx or j != rndy:
                            self.Tablero[i][j].NumBuquesAdyacentes += 1
            num += 1

```

Para colocar los barcos de forma aleatoria

```

def Pintar(self,avv,puntos,vida):
    global lblsalida
    global lblpuntos
    global lblvidas
    #doc = open("Python/" + "Prueba" + ".txt", "w")
    if self.Estado == "G":
        #plt.suptitle("Has Ganado :D")
        MessageBox.showinfo("Felicitaciones", "Has ganado el juego ")
    elif self.Estado == "P":
        MessageBox.showinfo("Sigue intentando", "Has perdido el juego ")
        #plt.suptitle("Has Perdido :(")

```

Por último pintamos el gráfico.

```

    lblsalida = avv
    lblpuntos = puntos
    lblvidas = vida
    plt.text(0, self.Tamano + 1.5, "Buques: " + str(avv) , fontdict=None)
    plt.text(4,self.Tamano + 2, "Puntos: " + str(puntos) , fontdict=None )
    plt.text(8,self.Tamano + 2, "Vidas: " + str(vida) , fontdict=None )
    for n in range(self.Tamano+1):
        plt.plot ([0,self.Tamano],[n,n], color="black", linewidth=1)
        plt.plot ([n,n],[0,self.Tamano], color="black", linewidth=1)
    for i in range(self.Tamano):
        for j in range(self.Tamano):
            px = j + 0.5
            py = self.Tamano - (i + 0.5)
            if self.Tablero[i][j].Visible:
                if self.Tablero[i][j].TieneBuque:
                    plt.plot([px], [py], linestyle='None', marker='.', markersize=8, color='teal')
                    #doc.write([px] + '\n')
                else:
                    #if self.Tablero[i][j].NumBuquesAdyacentes != 0:
                    plt.plot([px], [py], linestyle='None', marker='.', markersize=8, color='red')
            else:
                plt.plot([px], [py], linestyle='None', marker='.', markersize=4,color='black')
#doc.close()

```

6. Salida

```

def cerrar():
    MessageBox.showinfo("Adios", "Gracias por usar el programa")
    sys.exit()

```