Nombre: Luis Alberto Vargas González.

Fecha: 7/02/2025.

U5 A1: Métodos de llamadas remotas.

Clase: Sistemas Distribuidos.

Maestría en Ciberseguridad.

Descripción de programas.

En esta práctica se implementaron por igual 2 programas, uno llamado gato.py y el otro p2p.py, es destacable que el código de p2p es una reutilización del código implementado en la práctica de la unidad 3 en donde se implementó 2 funciones, una donde se inicia el servidor, y la otra donde se inicia el cliente, por lo que la realización de esta práctica llevo un tiempo y esfuerzo de menor índole. Por lo que considero que explicar de nuevo el código p2p es poco necesario; el código de gato.py lo explicaré más delante.

Código p2p.

```
import socket
import threading
import time

# Lista de nodos (IP, Puerto) en la red P2P
NODOS = [
("127.0.0.1", 56432), # Jugador 1
("127.0.0.1", 56433), # Jugador 2
]

# Función de servidor TCP para recibir mensajes
def iniciar_servidor_tcp(mi_ip, mi_puerto, callback_recibir_mensaje):
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Permitir reutilizar la dirección del socket para evitar el error "Address already
in use"
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
try:
```

```
server_socket.bind((mi_ip, mi_puerto))
server_socket.listen()
print(f"Servidor TCP escuchando en {mi_ip}:{mi_puerto}")
except OSError as e:
print(f"Error al iniciar servidor en {mi_puerto}: {e}")
return

while True:
conn, addr = server_socket.accept()
print(f"Conexión establecida con {addr}")
with conn:
while True:
data = conn.recv(1024)
if not data:
print(f"Cliente {addr} desconectado.")
break
mensaje = data.decode('utf-8')
callback_recibir_mensaje(mensaje)

# Función de cliente TCP con reintentos
def iniciar_cliente_tcp(mi_ip, mi_puerto, mensaje):
print(f"Intentando enviar mensaje: {mensaje}")
for ip, puerto in NODOS:
if (ip, puerto) != (mi_ip, mi_puerto): # No enviarse a sí mismo
intentos = 5 # Número de reintentos
for i in range(intentos):
try:
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as client_socket:
client_socket.connect((ip, puerto))
```

```
client_socket.sendall(mensaje.encode('utf-8'))
print(f"Mensaje enviado a {ip}:{puerto}")
break # Si se envió correctamente, salir del bucle
except ConnectionRefusedError:
print(f"Intento {i+1}/{intentos}: No se pudo conectar a {ip}:{puerto}.
Reintentando...")
time.sleep(2) # Esperar antes de volver a intentar
else:
print(f"No se pudo conectar a {ip}:{puerto} después de {intentos} intentos.")
```

Código gato.py.

```
import tkinter as tk
import threading
```

```
import sys
import time
from p2p import iniciar_servidor_tcp, iniciar_cliente_tcp

# Variables globales
tablero = [["" for _ in range(3)] for _ in range(3)]
turno_actual = "X"
botones = []
mi_ip = "127.0.0.1"
mi_puerto = int(sys.argv[1]) # Tomar el puerto desde la línea de comandos

# Función para actualizar el turno en la interfaz
def actualizar_turno(turno_label):
turno_label.config(text=f"Turno: {turno_actual}")

# Función para manejar movimientos en el tablero
def realizar_movimiento(i, j, boton, turno_label):
global turno_actual
if tablero[i][j] == "":
tablero[i][j] = turno_actual
boton.config(text=turno_actual, state="disabled")

# Enviar movimiento a la red
mensaje = f"{i},{j},{turno_actual}"
iniciar_cliente_tcp(mi_ip, mi_puerto, mensaje)

if verificar_ganador():
turno_label.config(text=f"¡Ganador: {turno_actual}!")
deshabilitar_tablero()
else:
```

```
turno_actual = "O" if turno_actual == "X" else "X"
actualizar_turno(turno_label)

# Función para verificar si hay un ganador
def verificar_ganador():
for i in range(3):
if tablero[i][0] == tablero[i][1] == tablero[i][2] != "":
return True
if tablero[0][i] == tablero[1][i] == tablero[2][i] != "":
return True
if tablero[0][0] == tablero[1][1] == tablero[2][2] != "":
return True
if tablero[0][2] == tablero[1][1] == tablero[2][0] != "":
return True
return False

# Función para deshabilitar el tablero cuando hay un ganador
def deshabilitar_tablero():
for fila in botones:
for boton in fila:
boton.config(state="disabled")

# Función para procesar los mensajes recibidos desde la red
def procesar_mensaje(mensaje):
global turno_actual
print("Mensaje recibido:", mensaje)
try:
i, j, jugador = mensaje.split(",")
i, j = int(i), int(j)
```

```
if tablero[i][j] == "":
tablero[i][j] = jugador
botones[i][j].config(text=jugador, state="disabled")

if verificar_ganador():
turno_label.config(text=f"¡Ganador: {jugador}!")
deshabilitar_tablero()
else:
turno_actual = "O" if turno_actual == "X" else "X"
actualizar_turno(turno_label)
except Exception as e:
print("Error al procesar mensaje:", e)

# Función para crear la interfaz gráfica
def crear_interfaz():
global botones, turno_label
ventana = tk.Tk()
ventana.title(f"Juego del Gato - Jugador {mi_puerto}")
ventana.geometry("300x350")

turno_label = tk.Label(ventana, text="Turno: X", font=("Arial", 14))
turno_label.pack(pady=10)

marco = tk.Frame(ventana)
marco.pack()

botones = []
for i in range(3):
fila = []
for j in range(3):
```

```
boton = tk.Button(marco, text="", font=("Arial", 24), width=5, height=2,
command=lambda i=i, j=j: realizar_movimiento(i, j, botones[i][j], turno_label))
boton.grid(row=i, column=j)
fila.append(boton)
botones.append(fila)

ventana.mainloop()

# Iniciar el servidor en un hilo separado
servidor_hilo = threading.Thread(target=iniciar_servidor_tcp, args=(mi_ip,
mi_puerto, procesar_mensaje), daemon=True)
servidor_hilo.start()

# Esperar antes de empezar para que ambos servidores estén listos
print(f"Esperando a que el otro jugador inicie...")
time.sleep(5)

# Crear la interfaz
crear_interfaz()
```

Este programa inicia definiendo variables importantes, en donde se específica el primer turno del primer jugador, se empieza a dibujar los tableros en donde participan los jugadores, se crea el arreglo de botones, se define la interfaz loopback (127.0.0.1) y se define el puerto, el cual tomará como dato la entrada del usuario, del puerto que seleccione.

Posteriormente se crea la función actualizar_turno en donde se configura para que el primer jugador que inicie sea el que usará las “x” en el tablero.

Después se crea la función realizar_movimiento que permite realizar movimientos a los dos jugadores en donde se define las dos dimensiones del tablero y en ello se configuran los botones, pues cada celda del tablero funciona como un botón por lo que en esta función se define un condicional que manda llamar la función posterior para que se verifique si el jugador 1 o 2 fue el ganador al poder conectar tres “X” o tres “O” y lo anuncia en el tablero.

La función verificar_ganador define un ciclo for el cual verifica todas las celdas para revisar si tienen dichas celdas un signo dentro de ellas, ya sea “x” o “o”, por lo que es importante verificar con las coordenadas posición por posición.

La siguiente función deshabilitar_tablero es la encargada de deshabilitar todos los botones una vez que ellos hayan sido o llenados con un símbolo ya sea “x” o “o”.

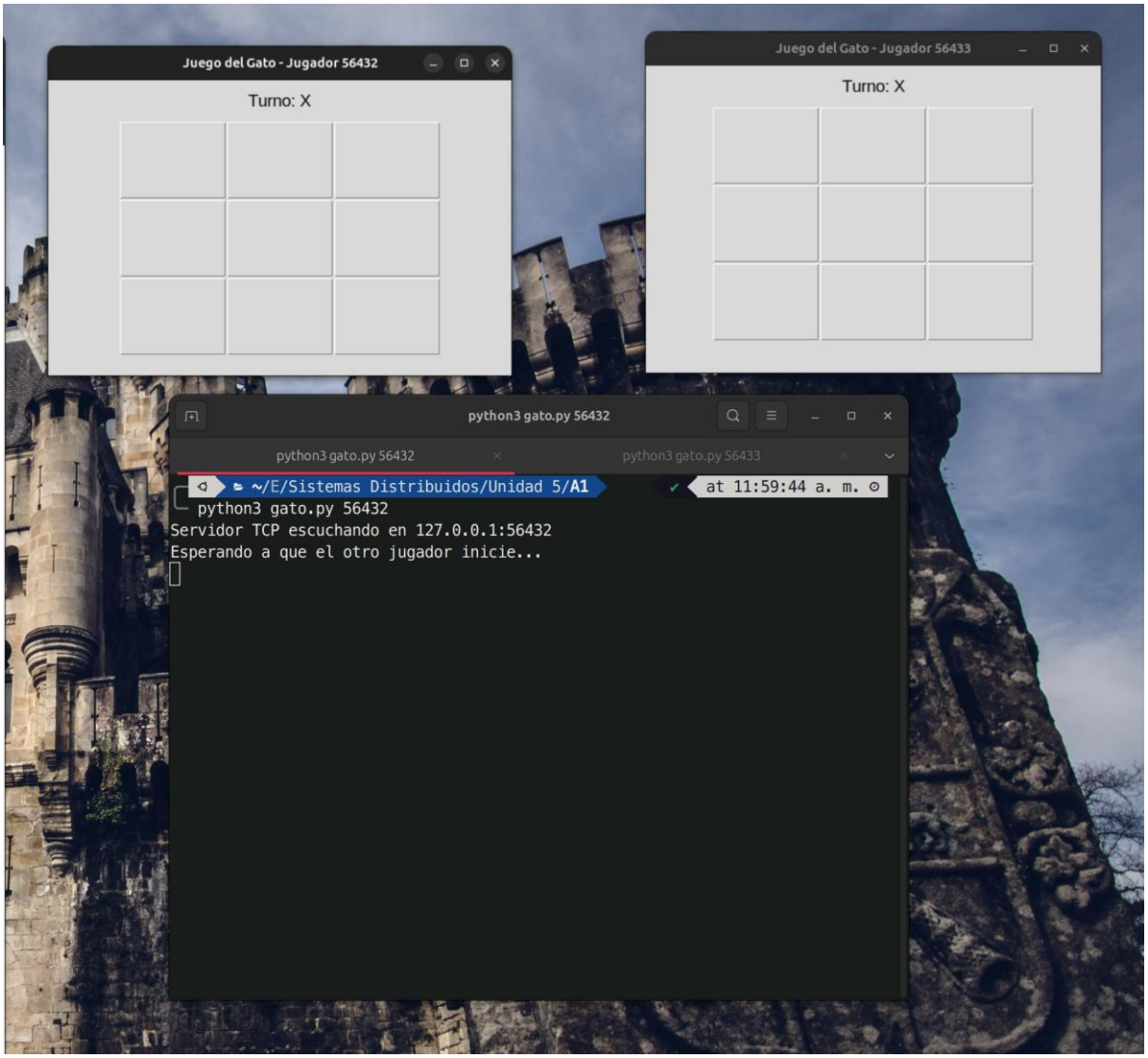
La función procesar_mensaje es la encargada de habilitar el tablero para ambos jugadores cuando sea su turno correspondiente y junto con la función deshabilitar_tablero también se manda a llamar para poder desactivar el tablero al jugador que no le corresponda el turno.

Despues se crea la interfaz del juego, y en esta función se definen el tamaño del tablero, la fuente y tamaño de las letras X y O; y con el método .Frame y. pack se crea el dibujo del tablero 3x3, y al final de esta función se crea un bucle con ventana.mainloop() para mantener abierto el tablero a ambos jugadores.

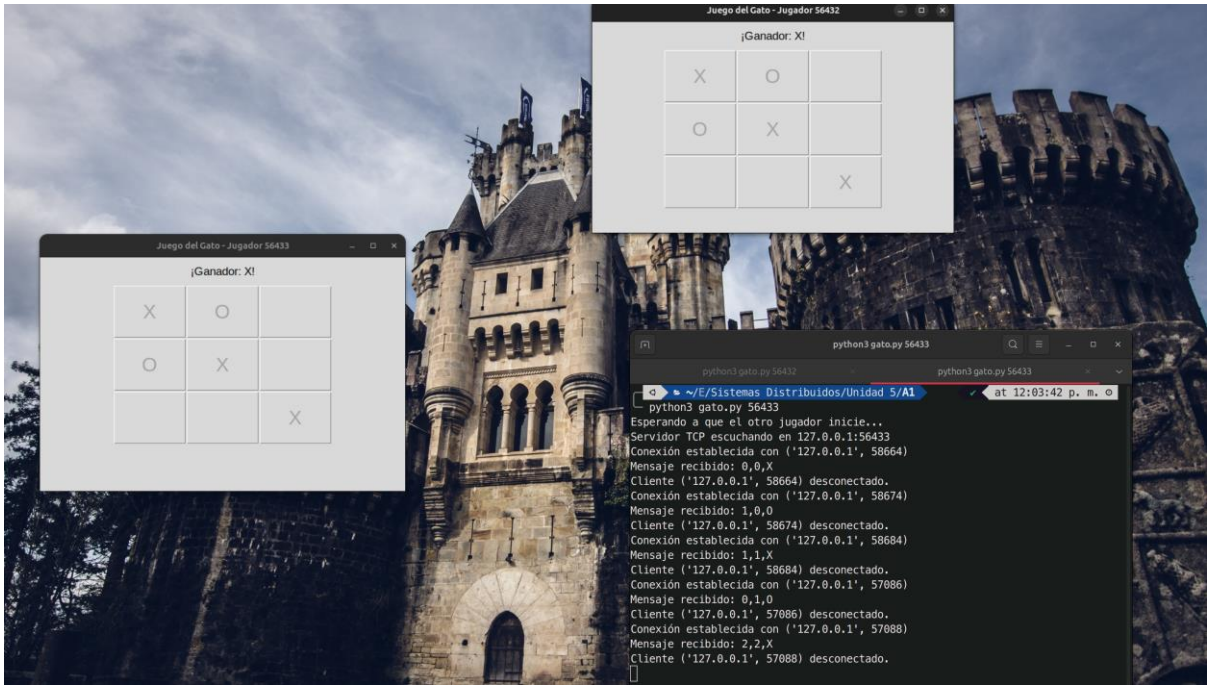
Posteriormente se mandan a crear los hilos que serán los encargados de llamar al método de iniciar servidor y cliente tcp del programa P2P.

Nota: Se habilita la función de sleep() para que el hilo correspondiente del primer jugador espere un tiempo suficiente para que se inicie el servidor y cliente del segundo jugador y se pueda empezar a jugar iniciando con el primer jugador.

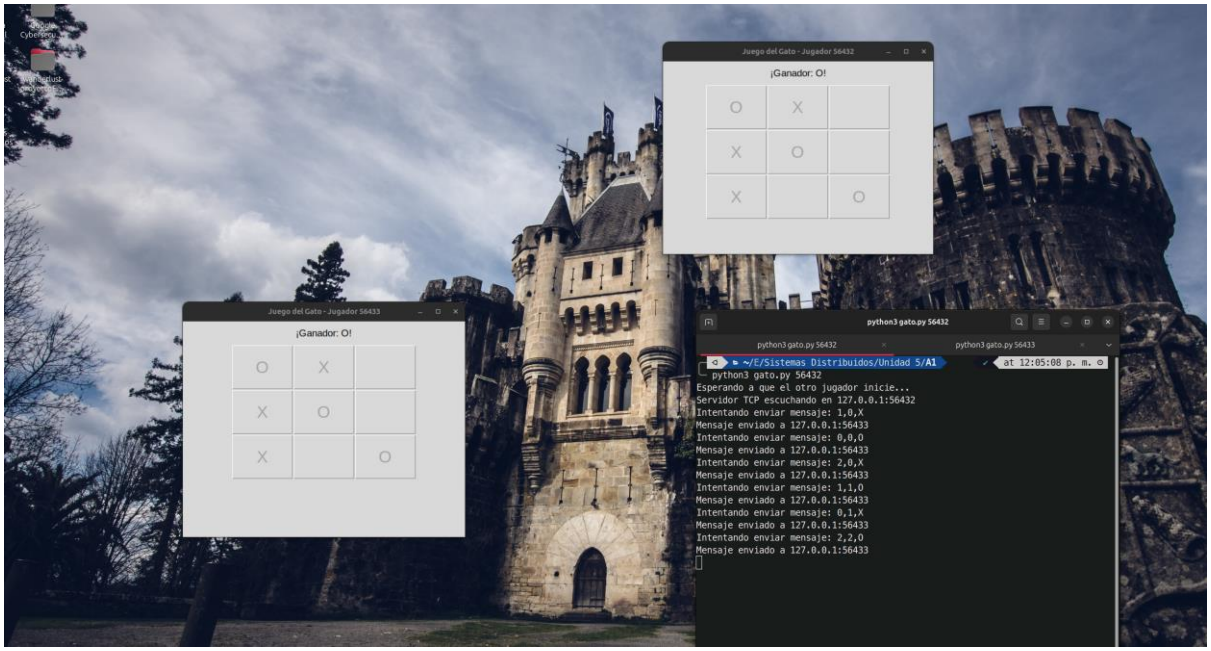
Evidencia de funcionamiento.



Inicio de jugadores



Gana jugador X



Gana jugador O

Conclusiones:

Podemos argumentar que se concluyó con éxito la practica y sus finalidades para poder proyectar el mismo enfoque de sistema distribuido que hemos estado construyendo en las demás prácticas, adicionalmente es destacable argumentar que, se tuvo que implementar la función de sleep() para que el primer jugador estuviera a la espera del segundo jugador para evitar que se cerrara la sesión del 1er jugador cuando se abra la sesión del segundo jugador, el cual fue el principal problema que tuve al crear esta práctica , ya que al iniciar el primero jugador funcionaba correctamente pero al ejecutar el programa para el segundo jugador , había una colisión de puertos a pesar de no estar usando el mismo puerto (56432 y 56433) para el jugador 1 y 2 respectivamente , pues esto ocurría porque el programa reemplazaba el servidor del jugador 1 por el del jugador 2.