UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Sistemas Operativos

Grupo 6

Proyecto 1

Integrantes:

Romero Pizano Christian Gustavo

Avila Reyes Iker

Semestre 2025-2

Maestro Gunnar Eyal Wolf Iszaevich

12 de abril del 2025

Introducción

A pesar del tiempo que nos consume la universidad, también nos gusta jugar videojuegos. Una problemática muy recurrente a la hora de conectarse a un juego en línea es que en ocasiones (especialmente en eventos) el ingreso simultáneo de miles de jugadores puede llegar a saturar los servidores del juego llegando a afectar a la cola de ingreso, a causar caídas abruptas, desincronización o un retraso para ingresar al juego; Es por eso que el acceso concurrente debe ser controlado.

En este contexto, podríamos enfrentar problemas como condiciones de carrera donde múltiples jugadores intentan ingresar al mismo tiempo y, si el servidor no protege adecuadamente sus recursos compartidos se producen inconsistencias (por ejemplo, jugadores que reciben recompensas que no deberían o acceso a distintas secciones a las cuales no deberían entrar).

Otro riesgo es la inanición en la que algunos jugadores podrían quedarse por tiempo indefinido en la cola si no se establece un sistema de prioridades o turnos justos. Además, si no se regula el número de conexiones simultáneas el servidor podría sufrir saturación, colapsando por exceso de peticiones y deteriorando el rendimiento para todos los usuarios.

Por estas razones es crucial controlar aspectos como el acceso a la cola de autenticación (evitando así condiciones de carrera) y la asignación ordenada de recursos del juego.

Sin embargo no todas las operaciones requieren un manejo de concurrencia. Un caso claro son las consultas de información ya fija, como cuando los jugadores acceden simultáneamente a datos de solo lectura (estadísticas de personajes, reglas del juego, noticias, entre otros), aquí el orden de las peticiones es irrelevante pues no modifican el estado del sistema.

La clave está en distinguir entre operaciones dependientes a un orden (como la secuencia de entrada y salida al juego) y aquellas donde la concurrencia puede manejarse con mayor flexibilidad, optimizando así el rendimiento sin comprometer la estabilidad del sistema.

Planteamiento del Proyecto

El proyecto consiste en una simulación de un servidor de un juego cooperativo. Digamos que dicho juego acaba de ser publicado, sin embargo, parece ser que los desarrolladores no esperaban recibir tantos jugadores, pues el juego no cuenta con los servidores suficientes para alojarlos a todos, lo cual causa colas de espera muy largas.

Al comenzar la simulación se generan alrededor de 24 jugadores en la cola de espera, y cada cierto tiempo, se siguen metiendo nuevos jugadores. El servidor puede alojar un máximo 12 jugadores, por lo que de los 12 iniciales solo pueden entrar la mitad, y el resto se queda esperando. En este juego, los jugadores deben de capturar siete objetivos: A, B, C, D, E, F y G. Los puntos A, B, D y E están disponibles desde el inicio de la partida, mientras que el objetivo C solo se abre una vez que A y B hayan sido capturados, similarmente, el punto F solo se abre después de que D y E hayan sido capturados. Finalmente, el objetivo G solo se abre cuando C y F hayan sido capturados.

Una vez entrado al servidor, cada jugador tiene la posibilidad de contribuir a la captura de alguno de los objetivos disponibles, sin embargo, debido a que las partidas pueden llegar a ser muy largas, algunos jugadores se terminan cansando y se salen del servidor. Una vez que el objetivo G es capturado, se termina la partida y se reinicia el juego desde el principio. Debido a la grán cantidad de usuarios que los servidores están recibiendo, los desarrolladores implementaron un parche temporal que desconecta a todos los jugadores del servidor cuando se acaba la partida, para también darle a los usuarios en cola la oportunidad de jugar.

Interfaz

Al correr el programa, se abre una ventana con tres secciones:

- La sección izquierda, “Cola”, muestra a todos los jugadores que se están esperando a entrar al servidor. Esta se actualiza cada vez que un jugador entre a la cola o se conecte al servidor.

- La sección central, “Servidor”, muestra el progreso total de cada objetivo, así como cada jugador conectado, el punto en el que se encuentra, y su tiempo de juego restante. Esta se actualiza cada segundo.

- La sección derecha, “Historial”, muestra todos los eventos notables de la simulación: Cada vez que se una un nuevo jugador a la cola, cada vez que se conecte un jugador al servidor, cada vez que se salga un jugador del servidor, cada vez que se capture un objetivo, y cada vez que se termine una partida. Esta se actualiza cada evento.

- En la parte inferior, se muestra un botón que dice “Iniciar Simulación”, el cual comienza la simulación.

Mecanismos de Sincronización

Esta simulación utiliza múltiples mecanismos de sincronización para modelar el acceso concurrente al servidor del juego:

- Hilos (threading.Thread): Cada jugador es representado como un hilo independiente, lo que permite modelar su comportamiento de forma concurrente respecto a otros jugadores.

- Semáforo (threading.Semaphore): Se utiliza un semáforo para controlar el número máximo de jugadores conectados al servidor simultáneamente (máximo 12). Cuando un jugador se conecta, adquiere el semáforo, y lo libera al desconectarse.

- Lock Reentrante (threading.RLock): Protege todas las estructuras compartidas (cola, servidor, objetivos) con un único lock el cual permite que un mismo hilo adquiera el lock múltiples veces
  ¿Por qué RLock y no Lock?
  Se utiliza porque permite que un mismo hilo adquiera el lock múltiples veces sin deadlocks, además de que es necesario cuando métodos como actualizar_estado_jugadores llaman a otros que también requieren el lock (`elegir_objetivo_disponible por ejemplo).

Lógica de Operación

Un minuto en la simulación es equivalente a un segundo en la vida real. Una vez iniciada la simulación y después de que se generen los 24 jugadores iniciales, se añade un nuevo jugador a la cola cada 4 minutos. Cada jugador tiene un nivel de desempeño que determina cuánto progreso puede realizar para capturar objetivos, este puede variar de 0.5 (jugador casual) a 2 (jugador experto). Cada jugador tiene también un límite de tiempo que está dispuesto a jugar, el cual varía de 15 a 45 minutos. Ya dentro del servidor, los jugadores eligen alguno de los objetivos disponibles y comienzan a contribuir a su captura. Cada minuto, el jugador añade 1% (multiplicado por su nivel de destreza) al nivel de progreso de su objetivo, y se decrementa su tiempo de juego restante. Cuando un objetivo llega al 100%, este se cierra y los jugadores buscan otro objetivo que capturar. Si el tiempo de juego de un jugador llega a cero, este se desconecta del servidor. Además, si el punto final es capturado y, por ende, se acaba la partida, se desconectan todos los jugadores actuales en el servidor y se reinician los objetivos a su estado inicial.

Elementos compartidos y mecanismos de concurrencia

- self.jugadores_cola: Lista de jugadores esperando para entrar al servidor.
- self.jugadores_servidor: Lista de jugadores actualmente conectados.
- self.objetivos: Diccionario con el progreso de los objetivos (A-G), donde cada clave es una letra y el valor es el porcentaje de captura.

Se utiliza un RLock para proteger las estructuras anteriores al añadir/remover jugadores de la cola, modificar progresos de captura y al momento en que se conecta o desconecta un jugador .

Interacción entre Hilos:

- Productores/Consumidores: Los jugadores actúan como consumidores de un recurso limitado (el servidor). La cola actúa como buffer.
- Semáforo como barrera de entrada: Solo permite el acceso simultáneo de hasta 12 jugadores.
- RLock: Garantiza coherencia en los datos cuando múltiples hilos intentan modificar estos recursos simultáneamente, evitando condiciones de carrera durante operaciones críticas como la conexión/desconexión de jugadores y la actualización del estado de los objetivos.

Entorno de Desarrollo

Requisitos:
- Python 3.11 o superior (lenguaje empleado para el desarrollo del código).
- Bibliotecas estándar (no se requieren instalaciones adicionales).

Bibliotecas Utilizadas:

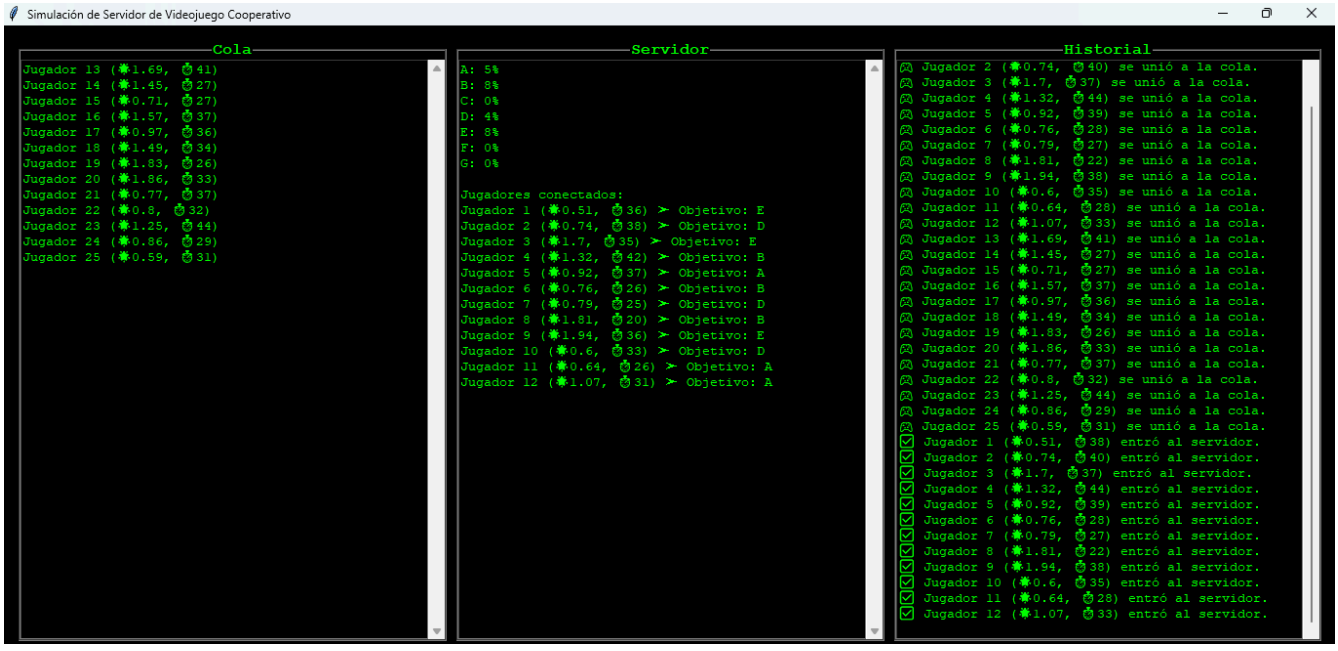
- Threading: Para hilos, semáforos y locks.
- time, random: Para simular el paso del tiempo y generar datos aleatorios.
- tkinter: Para crear la interfaz gráfica.

No se requieren bibliotecas externas para ejecutar el programa, además de que se proporcionará un ejecutable en caso de no contar con los requisitos.

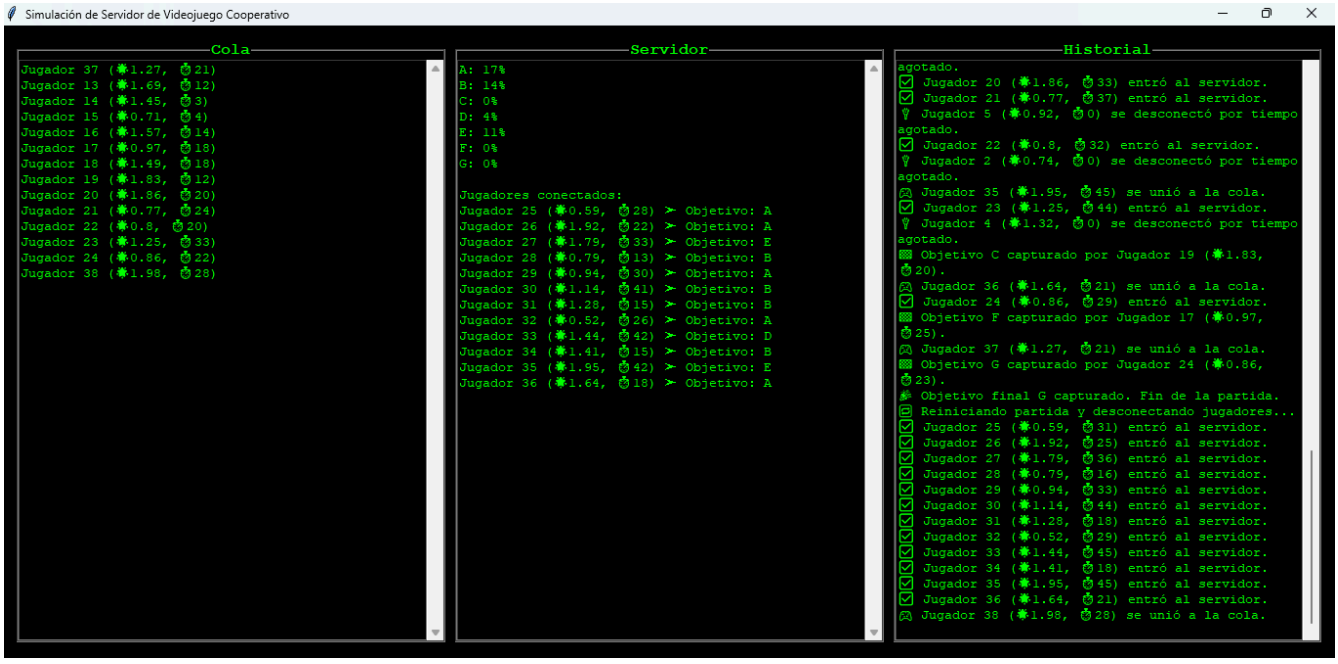
Sistema Operativo de Desarrollo: Fedora.

Ejecución:

Interfaz



Inicio de la simulación al presionar el botón



Actualización del servidor tras finalizar una partida

Conclusiones

Logramos implementar con éxito un simulador de servidor para un videojuego cooperativo que aborda el problema de la concurrencia en sistemas multijugador. Mediante el uso de semáforos para limitar las conexiones simultáneas y un RLock que protege las estructuras de datos compartidas, logramos evitar condiciones de carrera. A su vez fue posible desarrollar una interfaz gráfica amigable y fácil de entender y usar mediante tkinter, donde podemos visualizar de manera clara el funcionamiento del sistema (la cola de espera, el servidor y un historial de eventos).

Los principales aprendizajes incluyeron la importancia de seleccionar mecanismos de sincronización adecuados (como RLock para operaciones anidadas) y la necesidad de balancear flexibilidad con seguridad en el acceso concurrente.

Mediante este trabajo se demostró que mediante el uso de técnicas de programación concurrente bien aplicadas es posible modelar servidores de juego estables incluso bajo condiciones de alta demanda, lo cual podemos dar por sentado como una base para desarrollos más complejos en el ámbito de los sistemas operativos y los videojuegos multijugador de la vida real.