Lucia Bermejo

Informe Parcial 1 – Programación en Entornos Distribuidos

Informe: Máquina Arcade Distribuida en Python

Este proyecto consiste en desarrollar una **Máquina Arcade Distribuida** utilizando **Python**, donde se implementan tres juegos clásicos: **N Reinas**, **Torres de Hanói** y **Recorrido del Caballo** (Knight's Tour). Cada uno de estos juegos funciona como un cliente independiente que se comunica con un servidor central, el cual se encarga de almacenar los resultados de las partidas en una base de datos. La comunicación entre los clientes y el servidor se realiza mediante **sockets TCP/IP**, permitiendo que los juegos se conecten al servidor para enviar los resultados.

El propósito de este proyecto es aplicar conceptos de **programación distribuida** y **concurrencia**, mientras que se gestionan los datos de manera persistente mediante el uso de **SQLAlchemy** y **SQLite**.

Arquitectura del sistema

El sistema está basado en una arquitectura cliente-servidor, en la cual:

- **Clientes**: Son los juegos, cada uno con su propia lógica. Cada cliente puede enviar los resultados de las partidas al servidor central.
- **Servidor**: Es el encargado de recibir los resultados de los juegos y almacenarlos en la base de datos. Además, el servidor está preparado para manejar múltiples conexiones concurrentes, gracias al uso de **hilos**.

La comunicación entre los clientes y el servidor se realiza a través de **sockets TCP/IP**, lo que asegura una conexión estable y eficiente. El servidor, al recibir los resultados, los procesa y los guarda en una base de datos **SQLite** usando **SQLAlchemy**, que permite mapear los objetos de Python a las tablas de la base de datos.

Implementación de los Juegos

Se han implementado tres juegos, cada uno con su propia lógica:

1. **N Reinas**: Un juego clásico de ajedrez en el que se deben colocar N reinas en un tablero de N x N sin que se ataquen entre ellas. Este juego se resuelve mediante un algoritmo de **backtracking**.
2. **Torres de Hanói**: Un juego en el que se deben mover discos de una torre a otra sin violar las reglas (un disco grande no puede estar encima de uno más pequeño). Este también se resuelve usando **backtracking**.
3. **Recorrido del Caballo**: En este juego, un caballo debe recorrer todas las casillas de un tablero de ajedrez de manera que visite cada una exactamente una vez, respetando las reglas del ajedrez. Este juego se resuelve usando un algoritmo de **backtracking** similar al de N Reinas.

Interfaz Gráfica

Lucia Bermejo

Se ha implementado una interfaz gráfica para los tres juegos utilizando **Tkinter**, donde el usuario puede ingresar los parámetros (por ejemplo, el número de discos para el juego de Hanói o el tamaño del tablero para N Reinas), resolver el juego y ver los resultados.

Cada juego tiene su propio conjunto de interfaces gráficas que permiten al usuario interactuar con ellos de manera intuitiva:

- En **N Reinas**, el usuario introduce el tamaño del tablero y, al resolver el juego, se visualizan las posiciones de las reinas en el tablero.
- En **Torres de Hanói**, el usuario puede ver cómo se mueven los discos entre las torres hasta que se resuelve el puzzle.
- En **Recorrido del Caballo**, se muestra el tablero de ajedrez y el recorrido del caballo a medida que va visitando las casillas.

Persistencia de Datos

Los resultados de los juegos se almacenan en una base de datos **SQLite**. Para interactuar con la base de datos, se utiliza **SQLAlchemy**, un ORM que permite mapear las clases de los juegos a tablas de la base de datos. Los resultados incluyen información como el número de discos en el caso de Hanói, el número de movimientos realizados en el caso del Caballo, y el tamaño del tablero en N Reinas.

Concurrencia y Manejo de Hilos

Para garantizar que el sistema pueda manejar múltiples solicitudes simultáneas, el servidor utiliza **hilos (threads)**. Esto permite que, mientras un cliente está resolviendo un juego, otros clientes pueden conectarse y enviar sus resultados sin bloquear el servidor. Además, cada cliente también utiliza hilos para comunicarse con el servidor sin bloquear la interfaz gráfica del juego.

Pruebas y Validación

El sistema fue probado para asegurarse de que todos los juegos funcionaran correctamente. Se verificó que:

- Los resultados se guardaran correctamente en la base de datos.
- Los juegos se resolvieron correctamente para diferentes tamaños de tablero y números de discos.
- La comunicación entre el cliente y el servidor fuera eficiente, sin bloqueos o errores.

Dificultades Encontradas

Una de las principales dificultades fue el manejo de **concurrencia**, tanto en el servidor como en los clientes. Asegurarse de que los hilos estuvieran bien gestionados y que no hubiese condiciones de carrera fue un desafío. Sin embargo, este problema se resolvió utilizando las herramientas adecuadas de **Python** para el manejo de hilos y asegurando que el acceso a la base de datos estuviera sincronizado.

Otra dificultad fue la **gestión de la base de datos** y asegurarse de que la estructura de las tablas estuviera correcta desde el principio. Afortunadamente, **SQLAlchemy** facilitó la creación y el mantenimiento de la base de datos.

Lucia Bermejo

Conclusión

Este proyecto ha muy útil para aplicar y entender conceptos clave de la **programación distribuida**. Se han utilizado **sockets TCP/IP** para la comunicación entre clientes y servidores, **hilos** para manejar concurrencia, y **SQLAlchemy** para la persistencia de datos en una base de datos **SQLite**. Además, se ha integrado una interfaz gráfica interactiva utilizando **Tkinter** para cada uno de los juegos, lo que mejora la experiencia del usuario.

El proyecto es completamente funcional y permite resolver los tres juegos de manera eficiente, almacenando los resultados y proporcionando una forma visual e interactiva de jugar. Esto ha sido un gran aprendizaje en términos de arquitectura de software, concurrencia y diseño de interfaces gráficas.