IES Velázquez
Sevilla

Programación de servicios y procesos

Acceso a datos

Hundir la flota Sockets

2° DAM

Óscar Márquez Moya

Contenido

1. Tipo de Sockets.

Para el proyecto se ha optado por utilizar **Sockets TCP**, ya que el protocolo TCP está orientado a la conexión y permite que un flujo de bytes originado en un ordenador se entregue sin errores en el ordenador de destino. Por otro lado, el protocolo UDP, no está orientado a conexión, y prioriza la velocidad de la transmisión sobre la pérdida de paquetes, por lo que los Sockets UDP no son adecuados para este proyecto ya que puede haber perdida de información. De esta manera los Sockets TCP son los más adecuados para nosotros dado que la velocidad no es un requisito indispensable, por la naturaleza misma del juego, y la perdida de información puede ser nefasta.

2. Conexión directa vs. Arquitectura cliente-servidor

A continuación se detallan las ventajas e inconvenientes entre una conexión directa entre clientes (P2P) y la arquitectura cliente-servidor, enfocadas en su utilización en el juego del proyecto:

Ventajas de la Conexión Directa:

1. **Menos carga en el servidor:** En una conexión directa, los clientes se comunican directamente entre sí, lo que puede reducir la carga en el servidor. Lo que puede llegar a ser útil para juegos con una gran cantidad de jugadores.
2. **Menor latencia:** Al evitar el paso por el servidor, las comunicaciones pueden ser más rápidas, reduciendo la latencia en las interacciones entre jugadores.

Desventajas de la Conexión Directa:

1. **Complejidad:** Implementar y mantener una red P2P puede ser más complejo que una arquitectura cliente-servidor, especialmente en términos de manejar la conectividad directa entre clientes y gestionar problemas de seguridad.
2. **Seguridad:** P2P puede presentar retos adicionales en términos de seguridad y protección contra hacks, uso de bots, explotación de vulnerabilidades… ya que el servidor central tiene menos control directo sobre las interacciones entre clientes.

Ventajas de la Arquitectura Cliente-Servidor:

1. **Control centralizado:** Un servidor centralizado permite un control más eficaz sobre la lógica del juego, la sincronización y la seguridad.
2. **Manejo de la lógica del juego:** El servidor central puede ser responsable de la lógica del juego, lo que facilita la consistencia y la prevención contra problemas de seguridad.
3. **Clientes más ligeros:** Permite uso eficiente de clientes ligeros, ya que el servidor realiza el procesamiento principal, permitiendo interfaces

interactivas más amigables para el usuario, y centralizando recursos que permiten a los clientes compartir información de forma simultánea.

Desventajas de la Arquitectura Cliente-Servidor:

1. **Carga en el servidor:** Con un gran número de jugadores, el servidor puede convertirse en un cuello de botella y requerir una infraestructura más robusta.
2. **Mayor latencia:** Debido a que las interacciones entre jugadores pasan por el servidor, puede haber una mayor latencia en comparación con las conexiones directas.
3. **Complejidad en el desarrollo:** La implementación de la lógica del juego y la comunicación entre el cliente y el servidor puede ser más compleja. Además que requiere garantizar la consistencia de la información mediante mecanismos de sincronización, ya que varios clientes pueden operar simultáneamente con los mismos datos.

3. Funcionamiento del programa

Para la explicación del funcionamiento del programa vamos a empezar con el servidor. El servidor publica el puerto desde su hilo principal donde se van a recibir las conexiones, y espera en un bucle **`while(true)`** a que un cliente se conecte. Una vez que un cliente se conecta se instancia un objeto de la clase **`ConexionConCliente`** que extiende **`Thread`** y se ejecuta su hilo mientras que el principal continúa esperando otra conexión.

La clase **`Servidor`** de tipo **`singleton`** se instancia y hace una consulta a la base de datos para obtener las credenciales de todos los usuarios.

En el método **`run`** de **`ConexionConCliente`** haciendo llamadas a métodos de la clase **`Servidor`** se comprueban las credenciales del cliente y si no son correctas cierra la conexión. Si son correctas se consulta en la base de datos las partidas del cliente y se le envían si es que las hay; seguidamente se le envía también una lista de los usuarios conectados, que previamente el servidor ha guardado y actualizado en cada conexión y desconexión, notificando a todos clientes conectados en ese momento. De la misma manera si el servidor registra que se está intentando conectar un cliente ya conectado rechazará la conexión enviado dicho motivo de desconexión.

Una vez el cliente está conectado, su hilo de **`ConexionConCliente`** quedará a la espera de recibir información del cliente, y actuará en consecuencia enviándole la información necesaria en función de lo que haya recibido.

El servidor consta de dos paquetes: uno llamado **`hundirlaflota`** que contiene toda la lógica del juego en diferentes clases, y otro llamado servidor que contiene las clases ya mencionadas **`ConexionConCliente`** y **`Servidor`**, además de las clases **`GestionDatos`** y **`MainServer`**. A continuación se detalla la función que cumple cada una de ellas:

- **`MainServer`**: hilo principal, desde aquí arranca el servidor y se ejecutan los hilos de los clientes.
- **`Servidor`**: guarda los clientes conectados y se encarga de gestionar toda la lógica del servidor: envíos de información a clientes y gestión de la base de datos.
- **`ConexionConCliente`**: hilo de cada cliente conectado a la espera de información, a él llegan las peticiones del cliente y las filtra para que **`Servidor`** las gestione.
- **`GestionDatos`**: clase que contiene todos los métodos que gestionan la base de datos y que **`Servidor`** usará para ello.

Por otro lado, en cuanto al cliente, éste contiene tres paquetes: **`juego`** con la lógica necesaria del juego, **`vista`** con todas las clases de la interfaz y **`cliente`** con las clases que se encargan de la gestión.

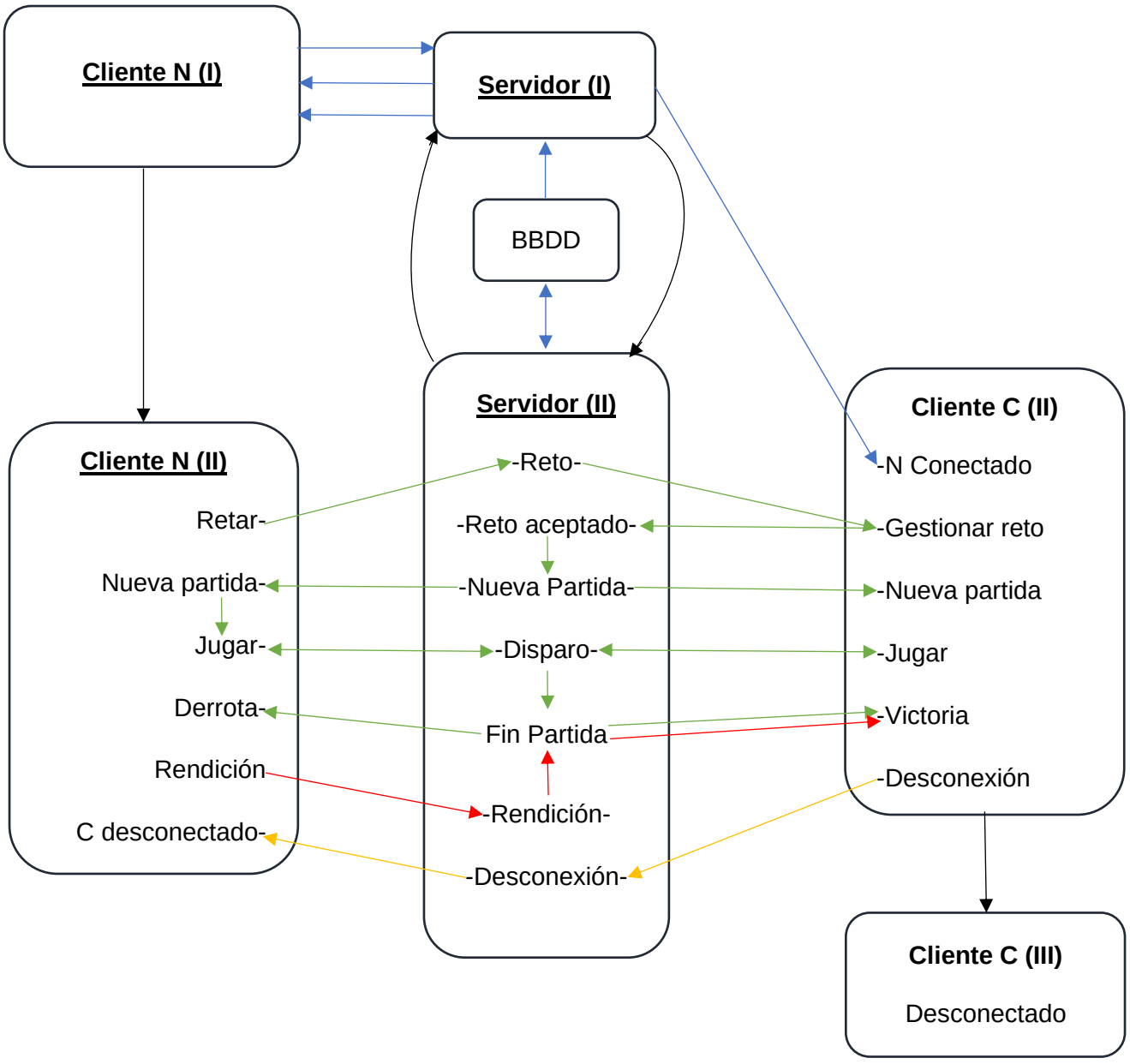
El cliente arranca desde la clase **`Login`** del paquete vista, e instancia el objeto de **`ConexionConServidor`** (del paquete cliente) desde el cual se envían las credenciales del cliente y recibirá el estado de la conexión. Si la conexión tiene éxito se recibirán las partidas del cliente así como los usuarios conectados en ese momento. Seguidamente se ejecutará el hilo de la clase **`ConexionConServidor`** desde el cual se realiza la escucha de la información que le envía el servidor, y se derivará la gestión de la misma a la clase **`GestionJuego`** del paquete **`cliente`**; aquí llega información sobre movimientos del rival, nuevos retos, y actualización de usuarios (rivales) conectados en todo momento.

Desde el hilo principal, la interfaz de usuario, se enviarán las solicitudes al servidor en función de las acciones del usuario, haciendo uso del método **`enviar`** de la clase **`ConexionConServidor`**, ya que la interfaz tiene la instancia de dicha clase como atributo en la mayoría de sus clases.

La clase **`GestionJuego`** se encarga de realizar la mayoría de operaciones en función de lo que recibe del servidor y de las acciones del cliente, y de actualizar la interfaz en tiempo de ejecución.

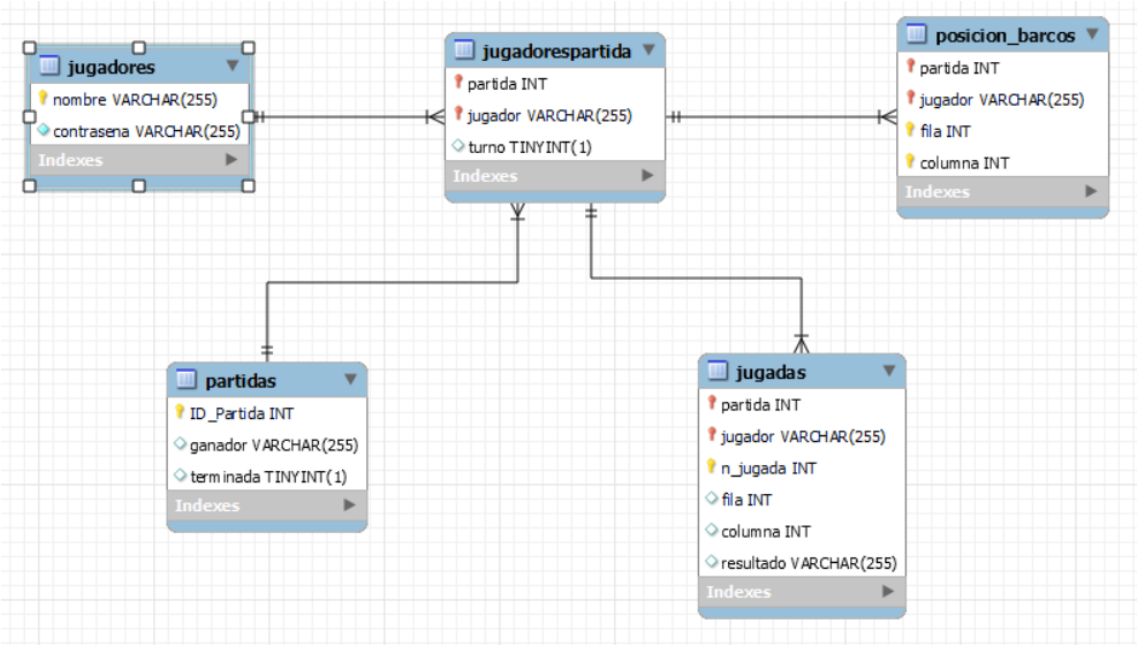
4. Mensajes entre cliente y servidor

A continuación se muestra un diagrama de estados con los mensajes intercambiados entre cliente y servidor, con el ejemplo de dos clientes, uno que se conecta (Cliente N)y otro ya conectado con anterioridad (Cliente C). Nótese que pese a que los clientes tienen todos los mismos estados aquí se han repartido entre Cliente N y Cliente C para simplificar el diagrama, de la misma manera ocurre con los mensajes tanto enviados como recibidos.



5. Base de datos

A continuación se muestra un diagrama relacional de la base de datos diseñada para el proyecto:



6. Notas a tener en cuenta

Para la prueba del programa se han creado los siguientes usuarios: angela, mari, oscar y usuario; cuyas contraseñas son sus mismos nombres. Todos ellos tienen datos de partidas, pero se recomienda usar "mari" que es el que más datos tiene.

Los .jar de cliente y servidor se encuentran en las carpetas sus respectivos proyectos. El cliente usa fuentes e imágenes para la interfaz, lo que hace que si se mueve de la carpeta donde está sin ir acompañado de las que contienen dichas fuentes e imágenes se conectará al servidor pero no podremos visualizar la interfaz; por lo que se recomienda ejecutarlo desde donde está.

En el proyecto del servidor, en las clases **`ConexionConServidor`** y **`MainServer`** hay varios **`System.out.println()`** comentados. Estos se pueden descomentar para ver por consola el arranque del servidor y las conexiones y desconexiones de los clientes.