PRACTICA RMI

MARC GÀNDARA SENTOLL & RAUL GABRIEL FILIP

COMPUTACIÓN DISTRIBUIDA Y APLICACIONES

UNIVERSIDAD DE LERIDA

GEI – Grado en Ingeniería Informática

Índex

1. Introducción

El objetivo de esta práctica era implementar una aplicación RMI para que varios usuarios puedan conectarse a un servidor con la finalidad de jugar a la versión Texas Hold'em de poker.

En el campus virtual se nos daba un esqueleto del programa que teníamos que desenvelejar. A partir de estes esqueleto había que implementar el cliente, servidor, interfaces y hacer algunas modificaciones sobre algunas clases ya implementadas.

Para implementar el servidor-cliente debíamos tener en cuenta el uso importante de los servicios RMI, callbacks entre cliente-servidor, sincronización correcta entre cliente y servidor, etc.

Además del programa, hemos redactado este informe con la finalidad de explica las implementaciones que hicimos, los métodos que tuvimos en cuenta, algunas dificultades… A continuación, se podrá ver la explicación de nuestra solución final.

2. Programación

2.1 Servidor

El servidor o **dealer** es el encargado de crear y gestionar la partida interactuando con los jugadores. Además de gestionar la partida, este permite cambiar diversas opciones del juego. Este permite cambiar:

- *Numero máximo de jugadores por partida*
- *Tiempo de espera antes de empezar la partida*

También hay que tener en cuenta que el dealer tiene que estar **informando** constantemente a los jugadores sobre el estado de la partida. Debe tener en cuenta que mientras haya un mínimo de 2 jugadores, la partida puede continuar. Pero en caso de que no haiga este mínimo, el dealer tendrá que finalizar la partida.

Mientras hay un mínimo de jugadores el **dealer** tiene que hacer lo siguiente:

1. **Repartir y notificar** 2 cartas a cada jugador
2. **Solicitar y esperar** por las apuestas antes de poner cartas en mesa (“Flop”)
3. **Repartir el Flop,** es decir, poner las 3 primeras cartas en la mesa y **notifcar** a los jugadores de las cartas que hay en la mesa.
4. **Solicitar y esperar** por las apuestas del “Flop”.
5. **Repartir el Turn,** es decir, poner la 4 carta sobre la mesa y **notificar** a los jugadores sobre las cartas que hay en la mesa.
6. **Solicitar y esperar** por las apuestas del “Turn”.
7. **Repartir el River,** es decir, poner la 5 y ultima carta sobre la mesa y **notificar** a los jugadores sobre las cartas que hay en la mesa.
8. **Solicitar y esperar** por las apuestas del “River”.
9. **Calcular el ganador**, a través de algunos métodos ya implementados se decide el ranking de quien tiene la mano más poderosa comparando la cartas de cada jugador con las que hay en la mesa.
10. **Anunciar al ganador.** El dealer le **notifica** al ganador y le da los “chips” ganados.
11. **Eliminar perdedores**. El dealer tiene que **notificar** al jugador que se ha quedado sin “chips” de que será eliminado.

2.1.1 Interfaz Servidor

- `void removePlayer(Player player) throws RemoteException` → método que elimina un jugador de la partida
- `void setOptions(TOptions values) throws RemoteException` → método que establece las opciones indicadas por parámetro
- `void StartGame()throws RemoteException, InterruptedException` → método que arranca la partida y mantiene la partida continua
- `void AddPlayer(Player player, PlayerCallback callback) throws RemoteException` → método encargado de añadir un jugador en la partida y comunicarle mediante una callback que esta dentro de la partida
- `void QuitGame()throws RemoteException, InterruptedException` → método que finaliza la partida y notifica a los jugadores mediante callback

2.1.2 Clase *DealerServer*

DealerServer

Para poder arrancar nuestro servidor, primero le tenemos que definir un método básico *DealerServer* el cual creará un HashMap concurrente que contendrá los jugadores y sus correspondientes callbacks. También tendrá que crear una instancia de *GameTexasHoldem*, que es la clase que contiene los métodos necesarios para arrancar la partida.

setOptions

Antes de arrancar el servidor tendrá que indicar las opciones. Estas opciones nos indicarán el número máximo de jugadores y el tiempo máximo de espera antes de arrancar la partida.

En este método tendremos que comprobar que los valores no sean nulos, que el número máximo de jugadores no sea menor que 2 y que el tiempo máximo de espera no sea menor que 0.

En caso de que no se cumplan estos parámetros se indicará un mensaje de error. En caso contrario, se indicará por la terminal del dealer las opciones elegidas.

removePlayer

Antes de poder eliminar un jugador tenemos que comprobar de que este realmente este dentro de la lista de jugadores. Esta comprobación la hacemos a través de la Hash Map *playerCallMap*.

En caso de que el jugador especificado este por parámetro:

1. Imprimiremos por pantalla de que el jugador ha estado eliminado
2. Obtendremos el jugador para eliminarlo de la lista de jugadores
3. Eliminamos el jugador del hash map *playerCallMap*

En caso contrario, indicaremos a través de un mensaje de que el jugador no esta registrado en la partida.

StartGame

Esta función se encarga de juntar las funciones del juego y ejecutarlas en orden. Se encarga desde la distribución de cartas hasta la determinación de los ganadores y la notificación de estos a los jugadores conectados.

Para poder arrancar la partida tendremos que comprobar de que el nombre de jugadores sea mayor que 2. Una vez hemos realizado la comprobación se va a ejecutar lo siguiente:

1. Vamos a obtener la lista de jugadores a partir del hash map de las callbacks y vamos a arrancar la partida.
2. El dealer va a repartir las cartas a los jugadores mediante el método *deal()*
3. Se van a pedir las apuestas pre-flop a los jugadores mediante el método *setBets()*
4. El dealer va a repartir las cartas Flop en la mesa mediante el método *callFlop()*
5. 2ª ronda de apuestas mediante *setBets()*
6. El dealer va a repartir la carta Turn en la mesa mediante *betTurn()*
7. 3ª ronda de apuestas mediante *setBets()*
8. El dealer va a repartir la ultima carta (River) en la mesa mediante *betRiver()*
9. Ultima ronda de apuestas
10. El dealer va determinar cual es el ganador y lo va a imprimir por pantalla

AddPlayer

Para facilitar algunos pasos, primero comprobamos si el jugador ya se encuentra en la partida. En caso de que el jugador se encuentra en la partida, se notificará a través de un mensaje de que el jugador ya se encuentra en la partida.

En caso contrario se seguirán los siguientes pasos:

1. Imprimiremos por pantalla de que hemos registrado el jugador y su correspondiente callback
2. Añadiremos el jugador juntamente con su callback en nuestro hash map *playerCallMap*
3. Notificaremos al jugador, mediante la callback, de que ha estado registrado en la partida.

QuitGame

Para poder acabar la partida, primero hay que eliminar todos los jugadores y después finalizar el proceso.

1. Iteramos sobre los jugadores guardados en *playerCallMap*
2. Para cada jugador se crea un nuevo hilo que elimina al jugador mediante el método *removePlayer*
3. Se inician los hilos y se almacenan en *quitThread*
4. Se espera a que los hilos acaben de eliminar todos los jugadores
5. Se notifica mediante un mensaje de que la partida ha finalizado
6. Finalizamos el programa con un *exit(0)*

2.2 Cliente

El cliente o **jugador** es el representante de los usuarios que juegan a una partida. Esta clase se encarga de interactuar constantemente con el servidor o **dealer** para participar en la partida.

El jugador debe responder a las solicitudes del servidor. En caso de que este no responda correctamente, se le volverá a pedir de que este responda correctamente. El jugador también tendrá asociado un número concreto de "chips" con las que podrá apostar.

Mientras el jugador este dentro de la partida, podrá realizar el siguiente:

1. **Abandonar** la partida. En caso de que el jugador no desea participar en la partida, la puede abandonar en cualquier momento.
2. **Apostar**, es decir, el jugador podrá apostar un número concreto de sus "chips".
3. **Subir apuesta**, es decir, si el jugador quiere hacer una apuesta más grande, puede subir la apuesta siempre y cuando cumpla las condiciones de apuesta.
4. **Igualar apuesta,** es decir, si el jugador no quiere subir la apuesta ni quiere abandonar, este puede igualar la apuesta para seguir en la partida.
5. **Pasar turno**

Para poder apostar el jugador debe de tener en cuenta de que la apuesta que realiza no puede ser menor que cero y no puede ser mayor que su número de "chips".

2.2.1 Interfaz PlayerCallback

- `void registeredClient(IPlayer player) throws RemoteException` → método que nos indica que el cliente se ha registrado correctamente
- `void gameStarted() throws RemoteException` → método que nos indica que la partida ha emepezado. También nos modifica las variables *gameStarted* para indicar que la partida ha empezado
- `void receiveCards(Card[] cards) throws RemoteException` → método que muestra al jugador las cartas recibidas
- `void showTableCards(List<Card> cards)throws RemoteException` → método que muestra al jugador las cartas que hay sobre la mesa
- `boolean isGamePlaying() throws RemoteException` → método que le imprime por pantalla al jugador las opciones que puede triar
- `void messageBet(int currentBet) throws RemoteException` → método que establece las opciones indicadas por parámetro
- `boolean isTurnBet()throws RemoteException` → método que indica si a un jugador le toca apostar o no
- `void sendBetOption(String option) throws RemoteException` → método que notifica la opción escogida
- `boolean isOptionValidated()throws RemoteException` → método que nos comprueba si la opción escogida es valida o no

- `void setOptionValidated(boolean optionValidated) throws RemoteException` → método que nos guarda la opción escogida por el jugador
- `String getBetOption() throws RemoteException, InterruptedException` → método que nos obtiene la apuesta del jugador
- `boolean isBetToDo()throws RemoteException` → método que nos indica si tenemos que apostar "chips" o no
- `void messageToDoBet(int currentBet, IPlayer player) throws RemoteException` → método que nos indica por pantalla la apuesta que tenemos que realizar i los "chips" disponibles del jugador
- `void sendBetAmount(int amount)throws RemoteException` → método que envia la apuesta del jugador al servidor para poder procesarla
- `int getBetAmount(IPlayer player, int currentBet) throws RemoteException, InterruptedException` → método que hace la lectura de la cantidad de apuesta insertada por el jugador
- `void getMessageAfterBet(int playerChips) throws RemoteException` → método que nos muestra las "chips" disponibles del jugador después de la apuesta
- `void finalizeBetTurn()throws RemoteException, InterruptedException` → método que finaliza el torno de apuestas del jugador
- `void invalidChips() throws RemoteException` → método que nos imprime por pantalla que la cantidad de "chips" insertada por el jugador es invalida.
- `void changeValidOption() throws RemoteException` → método que invierte las booleana *validOption*
- `boolean isValidOption() throws RemoteException` → método que nos indica si la boleana *validOption*, o opción escogida, es correcta o no
- `void setTrueConditions() throws RemoteException` → método que nos pone las booleanas *validOption* y *turnBet* a true.
- `void messageIncorrectPass() throws RemoteException` → método que nos imprime por pantalla un mensaje indicando que no podemos pasar de turno si no realizamos una apuesta.
- `boolean isPassed() throws RemoteException` → método que nos indica si el jugador ha pasado de turno o no
- `void messsageCallBet(int currentBet) throws RemoteException` → método que le confirma al jugador que ha igualado la apuesta
- `String getConfirmToEqualOrExit() throws RemoteException, InterruptedException` → método que pide al jugador la confirmación de igualar la apuesta o abandonar
- `void sendWinner(List<IPlayer> winners, GameTexasHoldem game) throws RemoteException` → método que indica a todos los jugadores cual ha sido el jugador ganador

2.2.2 Clase *PlayerCallbackImpl*

gameStarted()

Antes de arrancar la partida hay que indicar por pantalla al jugador que va a empezar la partida. También tendremos que definir una serie de variables booleanas:

- *gameStarted = true* → indicamos que la partida ha empezado
- *turnBet = false* → indicamos que todavía nadie ha empezado el turno de apuestas
- *validOption = false, gameStatus = false* → variables que necesitamos en otros métodos pero inicialmente necesitan estar a *false*

messageBet(int currentBet)

Para que el jugador sepa cuales opciones puede elegir durante el turno de apuestas, le vamos a imprimir por pantalla las diferentes opciones que puede escoger.

Vamos a inicializar la variable *turnBet = true* para indicar que ha empezado el turno de apuestas.

Hay que tener en cuenta que la primera apuesta que se realiza solo va a poder escoger 2 opciones: apostar o abandonar. A partir de la primera apuesta, el jugador puede escoger entre diferentes opciones:

- Pasar turno → siempre y cuando no se haya hecho una apuesta durante una ronda
- Subir apuesta → si el jugador quiere hacer una apuesta más grande
- Igualar apuesta → si el jugador quiere mantenerse en la partida pero no quiere apostar más de lo necesario
- Abandonar

También la indicaremos cual es la apuesta actual para que pueda decidir qué hacer.

getBetAmount(IPlayer player, int currentBet)

Este método es el encargado de obtener la apuesta indicada por el jugador, pero hay que tener en cuenta una serie de condiciones.

Antes de leer la apuesta del jugador vamos a indicar por pantalla la apuesta que ha insertado. A continuación, se van a hacer una serie de comprobaciones para validar la apuesta del jugador:

1. Comprobamos de que el jugador tiene suficientes “chips” para hacer una apuesta
2. Comprobamos de que la apuesta que ha hecho el jugador es igual o superior a la apuesta actual en la mesa

En caso de que no se cumpla alguna de estas condiciones se le va a indicar por pantalla de que no las ha cumplido y se le volverá a pedir a apostar de nuevo.

sendWinner(List<IPlayer> players, GameTexasHoldem game)

Una vez la partida ha llegado a su final, vamos a indicar por pantalla a todos los jugadores cual ha sido el jugador que ha ganado la partida.

Para hacer esto vamos a recorrer todos los jugadores, vamos a obtener el ganador de la lista de ganadores y le vamos a imprimir a cada jugador de la partida el ganador.

También se va a imprimir la mano ganadora, es decir, la combinación entre las cartas de la mesa y las cartas del jugador.

2.3 Otras modificaciones

validateOption(String playerOption, IPlayer player, PlayerCallback callback) → *método implementado en la clase GameTexasHoldem*

Para poder validar la opción escogida por el jugador vamos a necesitar hacer un *switch-case* para comprobar de que esta coincide con alguna de las opciones que se pueden escoger

- **case "AP"** → corresponde a la opción de apostar
- **case "PT"** → corresponde a la opción de pasar turno
- **case "PA"** → corresponde a la opción de subir apuesta
- **case "I"** → corresponde al caso de igualar la apuesta
- **case "A"** → corresponde al caso de abandonar la partida

bet(IPlayer player) → *método implementado en la clase GameTexasHoldem*

La función bet es responsable de manejar las apuestas realizadas por un jugador durante un turno en el juego. Obtiene las callbacks de los jugadores que están guardadas en el HashMap "playerCallMap", donde esto permite comunicar-se con el jugador. Verifica el que el jugador tenga las fichas disponibles y mediante las callbacks recibe e informa de la apuesta del jugador.

callBet(IPlayer player) → *método implementado en la clase GameTexasHoldem*

Esta función permite a un jugador igualar una apuesta realizada por otro jugador en el juego. Facilita que un jugador iguale la apuesta realizada por otro jugador, garantizando que se realice correctamente la igualación de la apuesta y que se mantenga actualizado el estado del juego y del jugador.

3. Conclusión

La finalidad de este documento era realizar una explicación general sobre las implementaciones de las funciones del cliente-servidor juntamente con la lógica del juego dentro del programa.

También hemos comentado las dificultades que hemos tenido al largo de esta práctica. Des de nuestro punto de vista, hemos tenidos varias dificultades en encontrar errores en los tornos de apuestas de los jugadores.

En términos de invocar los jugadores mediante callbacks e implementar hilos en el servidor, consideramos que lo hemos realizado correctamente.

Además de las explicaciones generales también se han incluido algunos métodos auxiliares y modificaciones de clases ya implementadas con la finalidad de facilitar nuestra programación.

En conclusión, consideramos que no tenemos un proyecto implementado completamente correcto, pero hemos conseguido los objetivos básicos para el funcionamiento correcto del juego.