Software Distribuït

# Pràctica 1: Client - servidor

UNIVERSITAT DE BARCELONA

Rodrigo Cabezas Quirós — 23843784Y
Abdelkarim Azzouguagh Ouniri — 51288439A

24 de març de 2020

# Índex

## Introducció

L'arquitectura client-servidor és un model de disseny de software en el que les tasques es reparteixen entre el proveïdor de serveis i els demandants, servidor i client respectivament. El client realitza peticions a un altre programa, el servidor, que retorna al client una resposta.

En aquesta pràctica s'implementarà el popular joc de "Batalla naval" seguint el model esmentat. A partir d'un protocol comú i una llibreria de comunicació bàsica es desenvoluparan dues aplicacions: un servidor multifil amb dos modes operatius (un i dos jugadors) i un client amb capacitat per jugar de forma autònoma.

Ambdós programes incorporaran tests unitaris per comprovar el seu correcte funcionament així com la documentació de cada classe i funcions implementades.

## Disseny del servidor

El servidor està estructurat en tres paquets: main, test i utils. Aquets contenen les classes involucrades directament en l'execució del programa, els tests unitaris i la llibreria de comunicació respectivament. El desglossament de les classes és el següent:

- Package main:
  - Classe Datagram: Proporciona un nivell d'abstracció a la llibreria ComUtils. S'encarrega de preparar les trames i enviar-les mitjançant el Socket en el format correcte.
  - Classe Game: Conté tota la lògica del joc. S'encarrega de l'execució joc així com la creació el log entre d'altres coses. S'ha implementat amb la idea de que serveixi tant pel mode d'un jugador com el de dos, evitant crear així una altra classe.
  - Classe GameThread: Implementa la interfície Runnable i es fa servir per executar múltiples partides en diferents fils. El sistema d'execució multifil s'ha implementat d'aquesta forma per facilitar la depuració d'errors durant el desenvolupament. D'aquesta manera la lògica del joc és independent de l'execució multifil i un únic pot ser executat directament des de el fil principal.
  - Classe Server: Conté el main del programa. Gestiona els paràmetres d'entrada i instancia els fils en funció del mode de joc indicat. També crea l'estructura HashMap estàtica que fan servir els fils per accedir als "diners" de cada jugador en els blocs sincronitzats.

- Package test:
  - Classe ComUtilsTest: Tests sobre les funcions de la llibreria ComUtils.
  - Classe DatagramTest: Conté tests sobre les funcions especifiques del servidor relacionades amb la comunicació amb el client.
  - Classe ServerTest: Comprovació dels rangs i número de paràmetres per a la classe principal.
- Package utils:
  - Classe ComUtils: És la llibreria proporcionada en la pràctica 0, ampliada amb funcions que s'han considerat necessàries per a les comunicacions. S'ha modificat la mida de les cadenes de caràcters a 4 (mida de les comandes) i s'ha fet servir la variable per a la transmissió dels missatges d'errors. Altres funcions que s'han incorporat són l'escriptura i lectura de bytes i lectura i escriptura d'espais i caràcters.

## Disseny del client

El client s'ha construït amb la mateixa estructura que el servidor per raons de simplicitat. El desglossament de les classes és el següent:

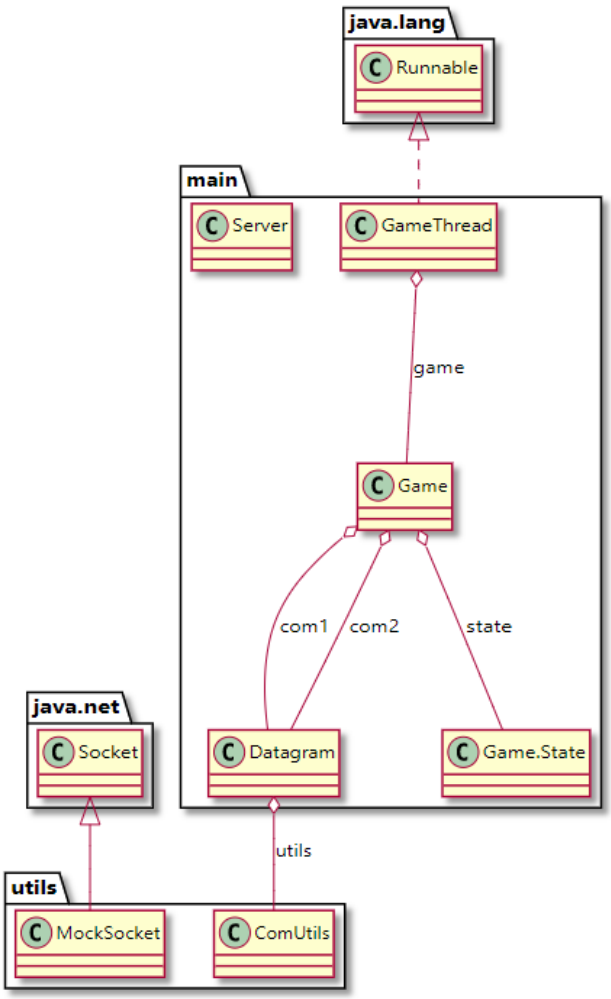
- Package main:
  - Classe client: Programa principal. Realitza la gestió dels paràmetres d'entrada i instancia la nova partida.
  - Classe Datagram: Té el mateix paper que en el cas del servidor, recepció i transmissió de les trames amb el format correcte. Implementa funcions diferents a les trobades a la classe homologa a Server, ja que el client transmet comandes especifiques com per exemple TAKE, EXIT o BETT.
  - Classe Game: Gestiona la lògica de la partida en la part del client. Gestiona les diferents accions en funció de la comanda rebuda i verifica que les entrades manuals tenen el format correcte per se processades pel servidor. Incorpora un mode automàtic, implementat mitjançant un algoritme senzill d'intel·ligència artificial, que juga de forma autònoma una partida.
  - Classe Menu: Processa la entrada del usuari i la prepara per a la seva posterior transmissió. Controla que les entrades del usuari tinguin el format correcte especificat en el protocol.
- Package test:
  - Classe ComUtilsTest: Testeja totes les funcions de la llibreria ComUtils.

- Classe DatagramTest: Tests sobre les funcions de comunicació que fa servir el client per intercanviar informació amb el servidor.
- Classe ClientTest: Comprovació del comportament de la classe principal davant la introducció de paràmetres erronis o fora del rang acceptable.
-

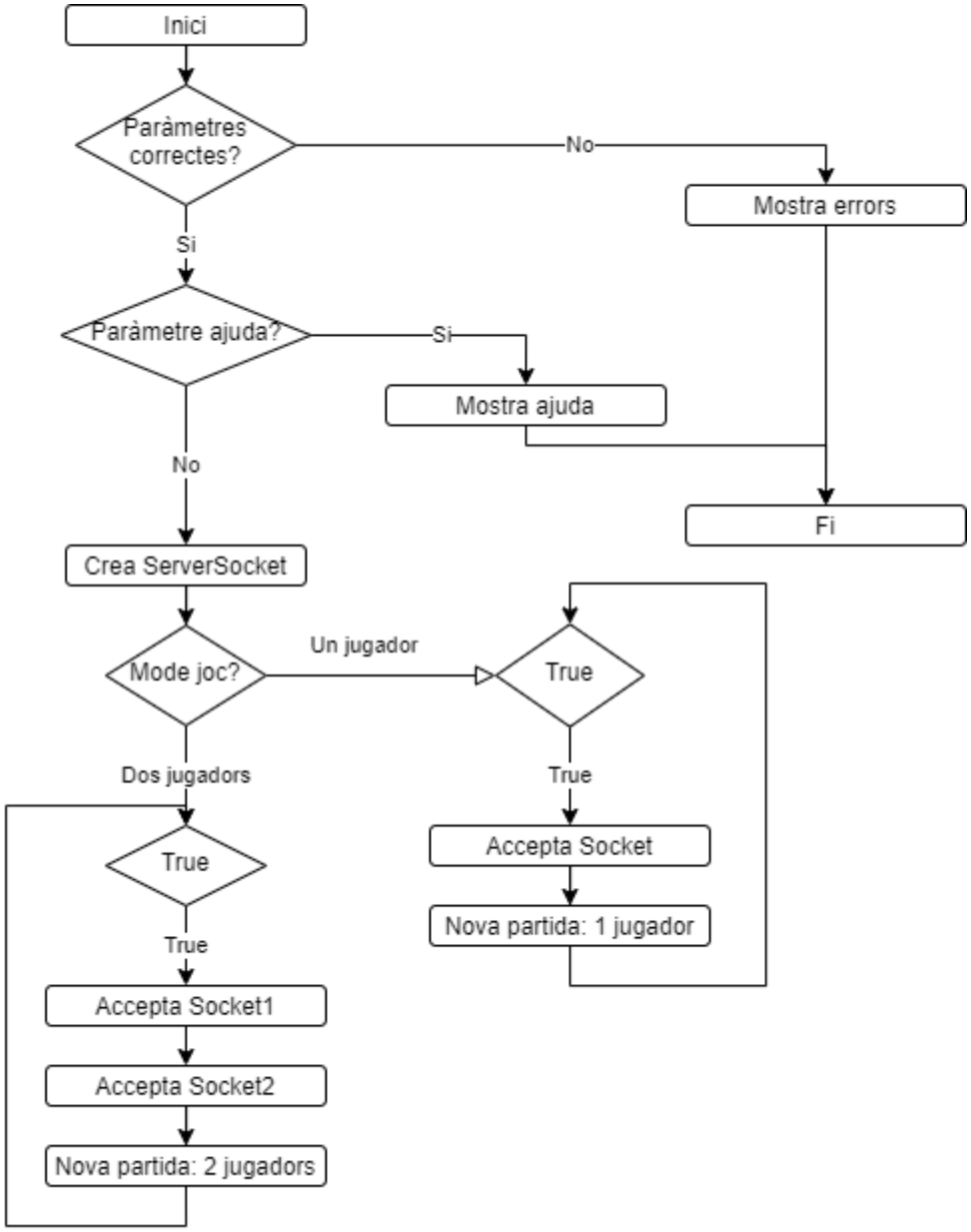
- Package utils:
  - Classe ComUtils: Llibreria similar a la de servidor. Ampliada amb les funcions necessàries per transmetre les comandes que el client transmet.

## Diagrames

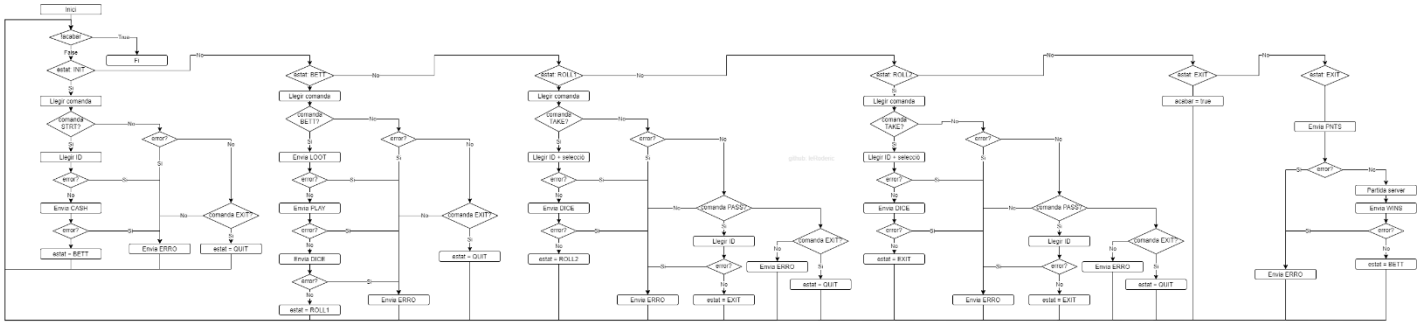
- Diagrama de classes de Server:



- Programa principal Server:



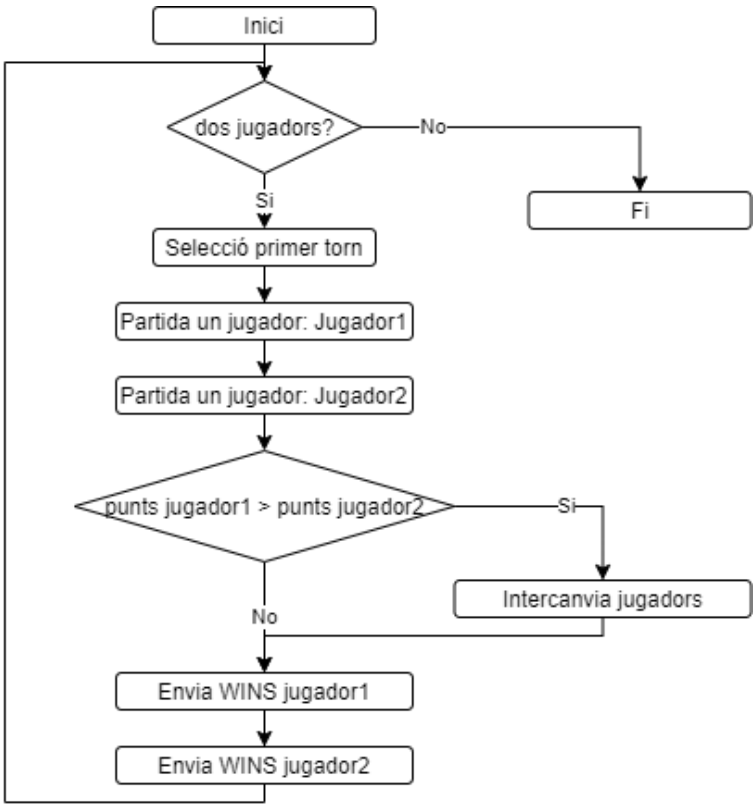
- Lògica partida un jugador:



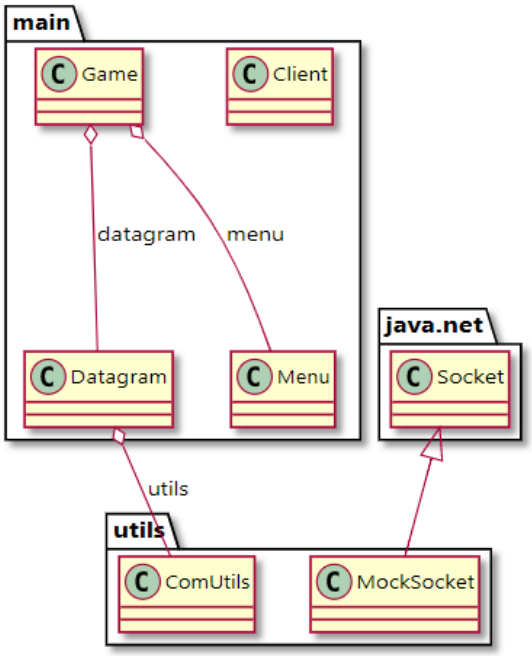
Fer clic aqui per veure la imatge ampliada.

- Lògica partida dos jugadors:

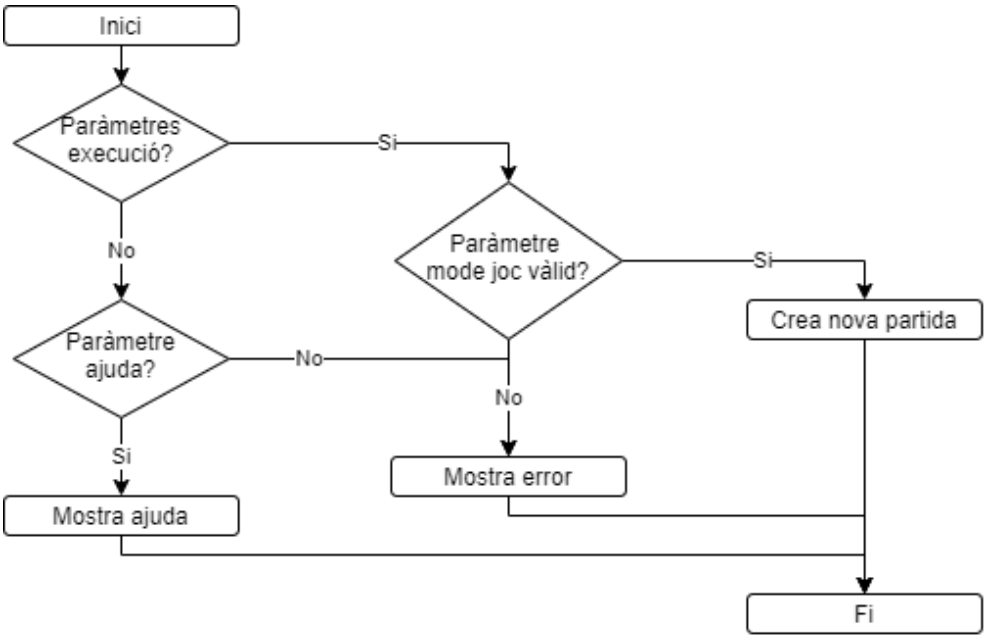
  La nostra implementació del mode de dos jugadors consisteix en dues partides d'un jugador on els jugadors no arriben a jugar contra el servidor. Per tal de simplificar el diagrama i no incloure l'anterior dos cops, hem especificat que es juga una partida d'un jugador amb cada jugador.



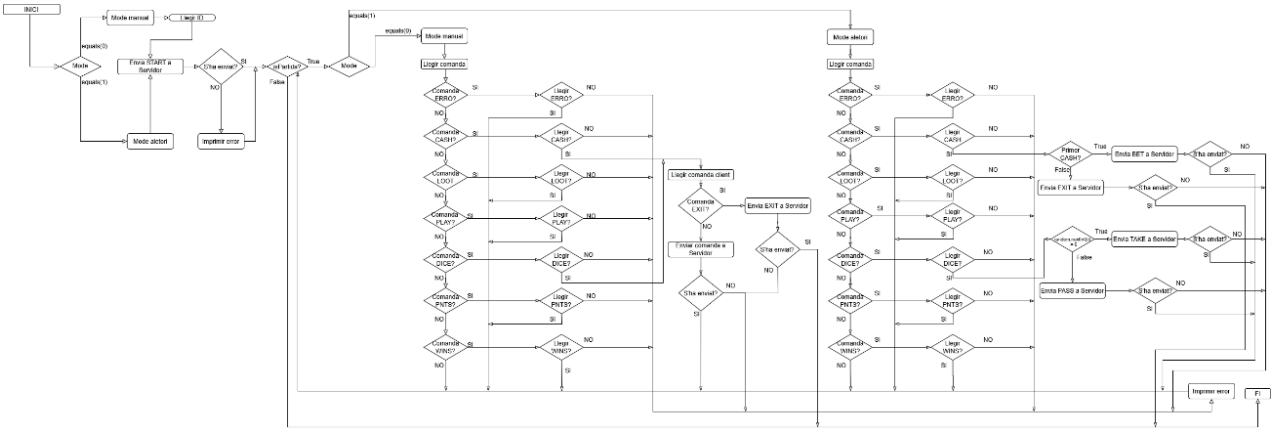
- Diagrama de classes de Client:




- Programa principal Client:



- Diagrama gestió de partida de Client:



Fer clic aquí per veure la imatge ampliada.

## Execució

Donada la suspensió de les activitats acadèmiques presencials a conseqüència de l'epidèmia COVID-19 i que el desenvolupament final de la pràctica s'ha dut a terme a casa en equips amb Windows, on no disposem del mateix programari que a les aules de laboratori, no hem pogut comprovar que la compilació del projecte a Linux i mitjançant Maven sigui correcta.

Tenint això present, l'execució de Client a Linux mitjançant Maven es faria de la següent manera:

```
mvn clean
mvn compile
mvn mvn exec:java -Dexec.mainClass=Client -Dexec.args="-s <IP_SERVER> -p
<SERVER_PORT> -i <PLAY_MODE>"
```

El servidor s'executaria de la següent manera:

```
mvn clean
mvn compile
mvn mvn exec:java -Dexec.mainClass=Server -Dexec.args="-p <SERVER_PORT> -i
<GAME_MODE>"
```

## Sessió de testing

A la sessió de testeig creuat, vam realitzar partides contra tots els servidors dels altres grups. Majoritàriament vam tenir problemes de connexió amb els servidors i no vam arribar a passar de la comanda STRT. Els resultats pel nostre client en els altres casos han estat els següents:

- Servidors A1,A3 i A6: El nostre client executava la partida correctament amb el servidor fins rebre la primera comanda de DICE. Després de revisar el codi de la nostra recepció amb el protocol proporcionat a la pràctica i donat que no hem rebut cap log dels grups implicats, em deduït que el problema hi era en el format en que el servidor enviava la trama DICE.
- Servidor A9: El servidor no enviava la comanda CASH després d'enviar STRT.
- Servidors A12 A, A12 B i A13: No vam poder executar cap partida contra els servidors ja que alguns no hi eren disponibles a les IPs facilitades o les sales on estaven eren plenes.

Pel que respecta el nostre servidor i mirant la graella de la sessió, sembla que tots els grups van poder fer partides contra ell. Tot i així, com molts grups no feien servir la ID del seu grup no hem pogut identificar tots els grups en els logs.

Hem realitzat una modificació al format del nom amb el que es guarden els logs del servidor. Tal i com es demana a la pràctica ("Server-<thread_name>.log"), al realitzar partides de manera successiva el log es sobreescrivia. Per tal d'evitar-ho, s'ha canviat el format dels noms dels logs a "Server-<thread_id>_dd-mm-yyyy_hh.mm.ss.log". Els resultats amb els clients que hem pogut identificar han estat els següents:

- Client A1:
  - Realitza la connexió i funciona bé fins que s'envia el primer DICE i paren l'execució. Log: Server-main-18-03-2020_18-42-56.log
  - Després de fer un altre intent tornen a tancar la connexió donant un error de "*Broken pipe"*. Log: Server-main-18-03-2020_19-06-08.log
- Client A3:
  - Realitzen dos intents de connexió i en ambdós cassos rebem un ID estrany que suposem es degut a la transmissió de l'ID en un format incorrecte. Sabem que és el grup A3 ja que vàrem parlar amb ells durant l'execució. Logs: Server-main-18-03-2020_18-38-38.log i Server-main-18-03-2020_18-39-51.log

- Client A5:
  - No arriba a fer la connexió i dóna error de "*Connection reset"*. Després d'intercanviar feedback amb el grup, ens han dit que el seu client no finalitza el protocol i per tant no executa de manera correcta. Log: Server-main-18-03-2020_19-23-15.log
- Client A2:
  - Executa partida perfectament. Vam identificar el grup a través del xat de Telegram habilitat, doncs l'ID utilitzat no permetia fer-ho. Log: Server-main-18-03-2020_20-02-18.log
- Client A12_A/A12_B:
  - Mateixa problemàtica que amb el client d'A1. Es connecta correctament i executa la partida fins arribar al primer DICE, llavors tanquen la connexió. No sabem ben bé si correspon al grup A o B ja que portaven el mateix ID numèric. Log: Server-main-18-03-2020_20-15-00.log

Gràcies a la sessió de test em pogut detectar diversos errors en les nostres aplicacions. Pel que respecta al client, en mode automàtic i de vegades i en funció de l'estat de la xarxa, envia les comandes massa ràpid. Això provoca que finalitzi la seva execució i el Socket es tanqui, impossibilitant al servidor la lectura de les comandes transmeses i donant un error de "*Connection reset"*. La solució que hem aplicat, i que també es va comentar a les classes de teoria, es aplicar un petit retràs abans d'enviar les comandes crítiques (EXIT per exemple) al servidor.

En el servidor també hem trobat alguns problemes. El log no es guarda quan es produeix un error. Això vàrem solucionar a l'inici de la sessió de test fent que el log es volques cada cop que hi hagués un error. Un altre error del log era que no s'enregistrava la comanda EXIT del client (veure log grup A2), això també s'ha arreglat.

Un altre problema detectat al servidor ha estat que s'entra en un bucle infinit quan es rep una comanda estranya. Per solucionar-ho, em afegit un comptador que acaba la partida després d'un cert nombre de comandes desconegudes.

## Conclusions

Mitjançant la implementació d'aquesta pràctica s'ha aprés a utilitzar els Sockets de Java mitjançant l'API que proporciona així com a crear i gestionar servidors multi-petició amb threads. S'ha programat un servir concurrent amb estats, tot analitzant i comparant les avantatges i desavantatges amb els iteratius o sense estats vists a classe.

També s'han aprés a implementar un protocol a nivell de capa de transport fent servir tant missatges de longitud fixa com missatges que codifiquen la seva longitud.

Finalment, i degut a situació epidèmica viscuda, s'han provat les nostres implementacions en unes condicions reals mitjançant una sessió de test creuada amb la resta de grups des de casa.