Luleå Tekniska Universitet **8 oktober 2014**
Laborationsrapport

Laboration 3

D0036D

JavaGameServer & Client

Namn Magnus Björk
E-mail magbjr-3@student.ltu.se

Handledare
Örjan Tjernström

Innehåll

1 Introduktion

Laborationsuppgiften var att programmera ett nätverksspel. Syftet med uppgiften var att man skulle utvidga sin kunskap gällande:

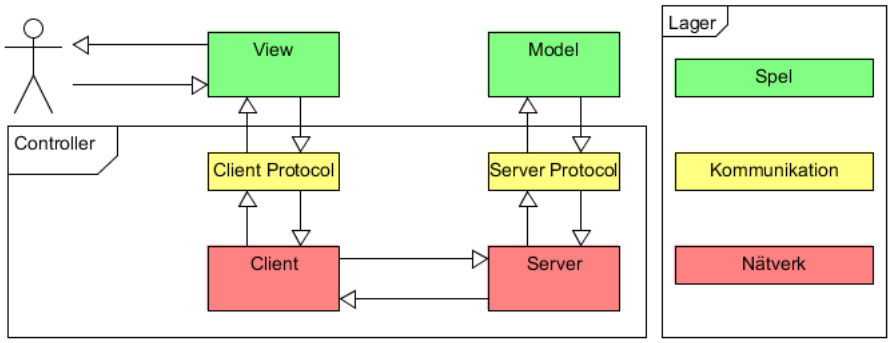
- **Nätverksprotokoll**
  - **TCP**
    Pålitligt protokoll för dataöverföring. Lämpligt att använda vid viktiga tillfällen i spelet. *T.ex. när en klient ansluter eller lämnar en server.*
  - **UDP**
    Opålitligt men snabbt protokoll för dataöverföring. Snabbare än TCP på grund av att det dels inte skickar tillbaka något ACK-meddelande till sändaren samt att UDP har en mindre 'frame'. Lämpar sig bäst för uppdateringar av förflyttningar på grund av sin snabbhet.
- **Sockets**
  Spelet skulle innehålla kommunikation över nätverket, detta sköts av diverse sockets:
  - **DatagramSocket**
    Hanterar trafik av protokollet UDP. Kan skicka unicast till en annan DatagramSocket eller multicast genom att skicka till en multicast-grupp. *T.ex. Önskemål om förflyttning från klient till server.*
  - **MulticastSocket**
    Hanterar inkommande multicast-trafik av protokollet UDP. En MulticastSocket går med i en multicast-grupp för att få ta del av dess trafik. *T.ex. uppdateringar av spelarnas position från server till klienter.)*
  - **ServerSocket**
    Serversocket tar emot trafik av protokollet TCP. Ansluter en klient till servern kan man sedan skapa ett Socket objekt för att hantera strömmen.
  - **Socket**
    Hanterar TCP-trafik. Skapas av ServerSocket på server-sidan. Klienten skapar den socket som tar kontakt med servern.
- **Trådning**
  En del moment måste köras parallellt och då kan man använda trådar. *T.ex. så har varje klient en tråd på servern som den kommunicerar med.*

2 Metod

Min strategi för att lösa uppgiften var att dela upp koden i 3 lager. Detta för att göra det enkelt att lägga på funktioner. *T.ex. Uppgiften att flytta uppdateringsmeddellanden från TCP till UDP var relativt smärtfri, hade spelmekanik varit inblandad i denna del hade det varit betydligt värre.*

- **Nätverk** Nätverkskoden skall endast skicka meddellanden mellan servern och klienten. Ingenting som har med spel-lagret att göra skall finnas här.
- **Spel** Spelets logik och utseende skall endast påverkas av kommunikationslagret av programmet.
- **Kommunikation**
  Det lager som sköter kommunikation mellan Spel- och Nätverks-lagren.

2.1 Design



Jag ville skapa ett dataflöde liknande bilden ovan.

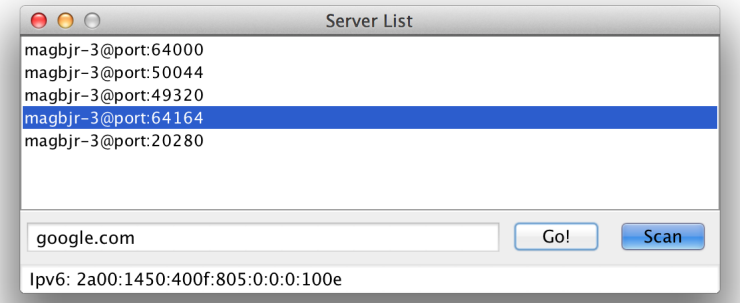
1. Spelaren trycker på en tangent för att flytta sin markör.
2. Klient-protokollet skapar ett meddelande med användarens indata och skickar detta till nätverkslagret.
3. Nätverkslagret skickar nu meddelandet från klienten till servern.
4. Server-protokollet tar emot och bearbetar modellen efter anvisningar i meddelandet. Skapar ett nytt meddelande som skickas tillbaka till klienten via nätverkslagret.
5. Klient-protokollet tar emot meddelandet och ändrar eventuellt vyn som visas för spelaren.
6. Spelaren upptäcker förflyttningen (Om den var giltig) och kan nu ge ny indata om den så vill.

3 Resultat

3.1 Användarhandledning

3.1.1 Serverlista

När en spelare startar en klient möts denne av denna vy:



Här kan man användaren göra följande:

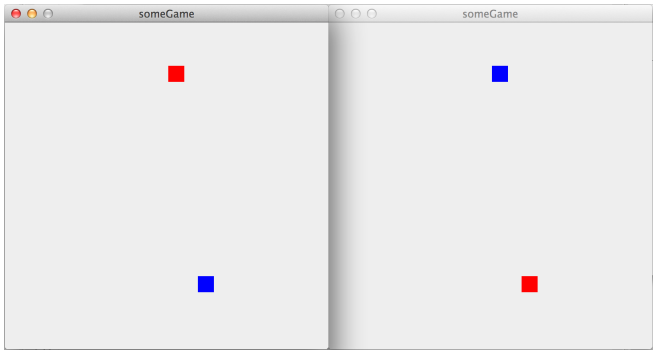
- **Scan**
  Scanna det lokala nätverket efter servrar, hittas en server läggs denne till listan ovan. *Klienten startar 2 trådar, en tråd som skickar ett meddelande till en multicastgroup och en annan tråd som lyssnar efter servrar som svarar på meddelandet. Om en server hittas skickar denne tillbaka en tråd innehållande IP och port nummer. Detta sparas därefter för framtida användande.*

- **Go!**
  I textfältet till vänster om denna knapp kan användaren skriva in en URL. Trycker användaren därefter på knappen 'Go!' så tar programmet reda på IP adressen till denna URL. Finns en Ipv6 adress så är det den som visas. *Programmet använder sig av klassen InetAddress för att ta reda på adressen, sedan används subklassen Inet6Address för att sortera fram Ipv6-adressen.*

- **Serverlista**
  Har nu användaren hittat sig en server denne vill logga in på så kan användaren göra det genom att dubbel-klicka på en server i listan. *Programmet använder nu IP och port nummer det sparat undan för att kontakta servern via TCP och där startar spelprotokollet.*

3.1.2 Spelvy

Spelet har nu börjat, om två spelare är inne på samma server kan det se ut så här:

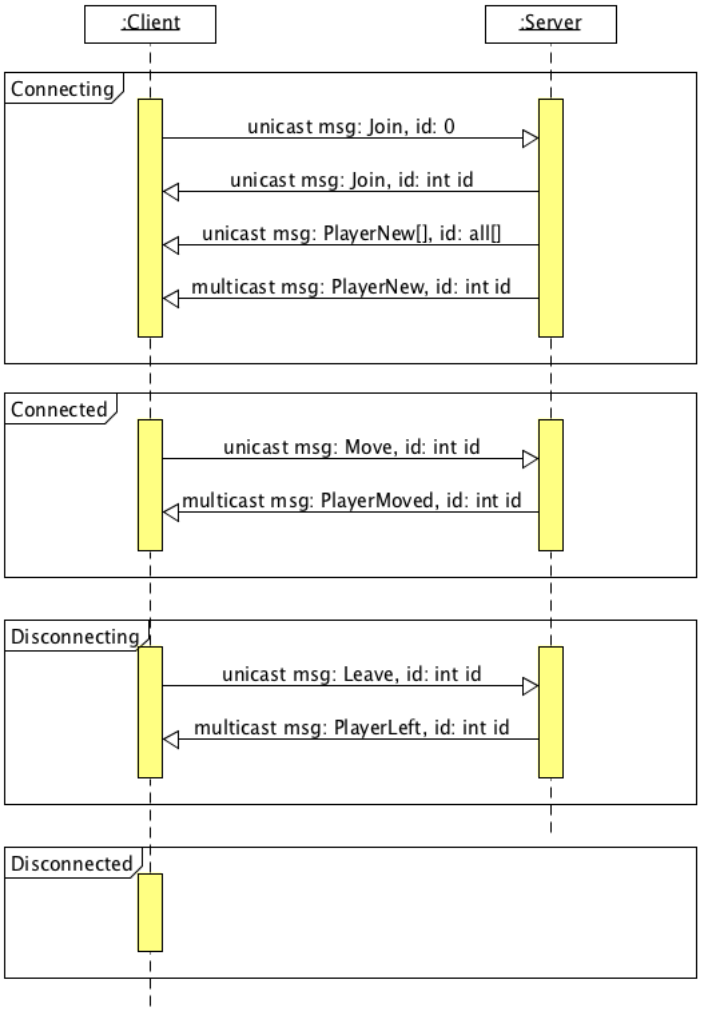


Röd spelare är den klienten styr, den blå är en annan klient. Objekten kan inte förflytta sig utanför spelplanen, de kan heller inte förflytta sig 'över' varandra, de kolliderar alltså.

Skulle en spelare vilja avsluta spelet gör denne det genom att stängar ner fönstret, då öppnas föregående fönster igen och spelaren kan antingen välja en annan server eller stänga även detta fönster. Stänger man serverlist fönstret så stängs applikationen.

För de spelare som stannar kvar på servern försvinner helt enkelt den fyrkant som representerade den spelare som lämnade servern.

3.2 Spelprotokoll

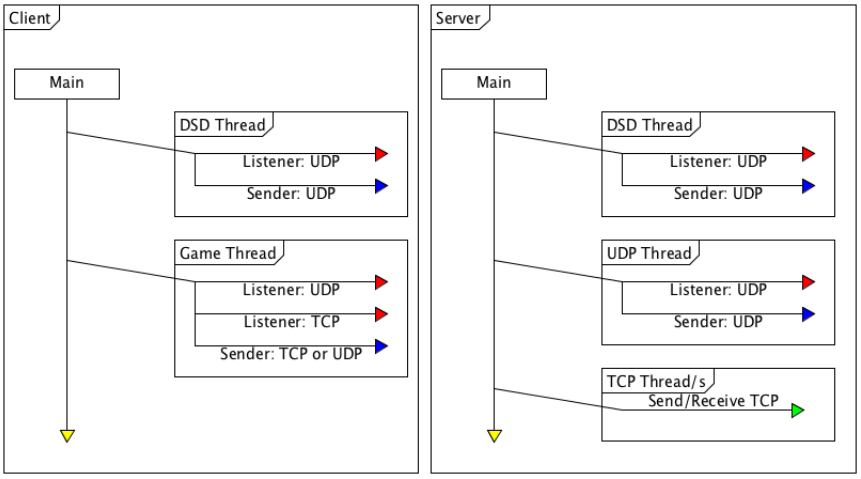


Klienten har en gameLoop som skapar ett meddelande varje runda. Vilket meddelande som skapas styrs av klientens 4 olika stadier:

- **Connecting**
  1. Klienten har id=0 och skickar meddelande *Join* till servern.
  2. Servern svarar med meddelande *Join* och ger klienten även ett id: *'ID'*.
  3. Servern skickar alla nuvarande spelares position och id till den nya klienten: Meddelande *PlayerNew*.
  4. Servern skickar den nya spelarens position till alla spelare på servern, även den nya spelaren: Meddelande *PlayerNew*.
  5. **När klienten upptäcker att den nya spelaren är sig själv (*'ID'*) byter klienten stadium till 'Connected'.**

- **Connected**
  1. Nu kollar klienten om spelaren gett input via tangentbordet, har denne gjort det skapas ett meddelande: *Move* med riktning av förflyttningen samt klientens *'ID'*. Detta skickas till servern via UDP.
  2. Servern tar emot datagrammet från klienten och undersöker om förflyttningen är giltig. Är den det skickas meddelandet: *PlayerMoved* med *'ID'* till samtliga klienter via UDP multicastsocket. Annars skickas ingenting.
  3. Klienten tar emot meddelandet och ritar ut förflyttningen på spelplanen.
  4. **Detta pågår tills dess att spelaren stänger ner spelfönstret, då sätts klientens stadium till 'Disconnecting'.**
- **Disconnecting**
  1. Klienten skickar nu ett meddelande: *Leave* till servern.
  2. Servern raderar alla spår av klienten i modellen och skickar meddelandet: *PlayerLeft* till alla klienter.
  3. Klienten upptäcker att den har lämnat server genom att *'ID'* stämmer överens med sitt eget id.
  4. **Klienten sätter sitt stadium till 'Disconnected'.**
- **Disconnected**
  I detta stadium har finns inte någon koppling mellan servern och klienten. Klienten avslutar nu sina trådar och applikationen startar serverlistvyn igen.

3.3 Trådar



3.3.1 Klient

Klienten startar har följande trådar:

- **DSD: Dynamic Service Discovery**
  Används för att hitta spelservrar genom att skicka strängen SERVICE QUERY JavaGameServer till en muticast-grupp. Sänder ut denna sträng medans knappen Scan"är intryckt. Startar även tråd:
  - UDP lyssnar tråd. Denna tråd tar emot svar från hittade servrar.
- **Game Thread**
  Den tråd spelet körs i. TCP/UDP-ut körs även här. Detta för att endast ett meddelande kommer att gå ut från klienten per cykel av klientens gameLoop. Startar ytterligare två trådar:
  - TCP lyssnar tråd. För att inte blocka multicast över TCP från server trådar så har klienten en egen tråd för TCP in.
  - UDP lyssnar tråd. För att inte blocka klienten så har även UDP-in en egen tråd.

3.3.2 Server

Serverns antal trådar beror på antal klienter anslutna. Trådarna är:

- **DSD Thread**
  Börjar med att lyssna efter klienter som vill hitta servern. Tar en klient kontakt med servern skickas serverns IP och port nummer tillbaka till klienten. Startas vid serverstart, **en tråd per server**.
- **UDP Thread**
  Lyssnar efter förflyttnings-förfrågningar av klienter, skickar tillbaka förflyttning till alla klienter kopplade till servern om förflyttning beviljas. Startas i sammband med serverstart, **en tråd per server** krävs.
- **TCP Thread**
  Sänder och tar emot meddellanden som skickas till servern över TCP. Startas när en klient ansluter till servern, därför krävs **en tråd per ansluten klient**.

4 Diskussion

Laborationen var intressant och givande. Det hade dock varit kul om det hade varit krav på att spelet skulle följa ett specifikt protokoll likt laboration 4. Detta för att kunna testa servrarna/klienterna mot sina klasskamraters.

Datagram*/Packet/Socket
Efter att ha arbetat med TCP strömmar tog det ett tag att ställa om tankesättet till UDP. Men om man tänker DatagramSocket som en brevlåda, DatagramPacket som ett kuvert och Datagram som ett brev så blir det lite klarare. Man skulle kunna tänka sig att man sätter ett tomt kuvert i sin brevlåda och väntar på att en brevbärare stoppar in ett brev i kuvertet, när han gjort det målar han om brevlådan och då vet man att det finns ett brev inuti kuvertet!

4.1 Threadpool

4.2 Synchronize