D0036D Nätverksprogrammering

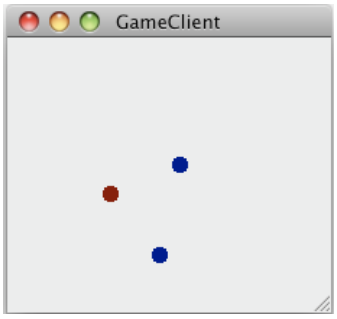
Laboration 3 – Enkelt java spel

Din uppgift är att implementera ett spel (klient/server) i ett antal steg. Dokumentera din lösning i en labbrapport och använd wireshark för att titta på nätverkstrafiken och jämföra de olika lösningarna. Rapporten skall innehålla dokumentation över ditt applikationsprotokoll.

a)
Börja med din spelidé. Designa sedan ditt spel utifrån denna. Sedan ska du tänka igenom nätverstrafiken som måste finnas för att spelet skall fungera. Utifrån detta skapar du ett eget protokoll som definierar meddelandena som ska skickas. Protokollet får inte vara språkspecifikt utan skall kunna implementeras i vilket språk, med nätverksstöd, som helst. Se till att din dokumentation är tillräcklig för att implementera detta protokoll i viket språk som helst?

b)
Implementera ditt spel. Det skall stödja att flera klienter samtidigt kan vara uppkopplade mot servern. All kommunikation ska ske med TCP. Från början är det OK att ange serverns IP adress och använda ett fast portnummer.

Klient
Klienten ska ha ett grafiskt gränssnitt som visar deltagande ”spelare” i en ”spelvärld”. Nedanstående bild visar ett exempel på hur det kan se ut.



I detta exempel visas egen spelare som röd boll medan övriga spelare är blåa. Du kan med fördel använda bilder istället.

Spelplanen kan även innehålla hinder eller andra (dolda?) effekter…

Klienten måste hålla reda på alla spelare som den ska rita ut och samtidigt kommunicera positioner med servern och hantera användarens input. Kontroll av spelaren kan ske via tangentbordet eller via musen. Till servern skickas ett önskemål om att flytta sin spelare. Det är servern som bestämmer om det är tillåtet att flytta till den önskade positionen och klienten ska endast flytta spelare på begäran från servern.

Server
Servern ska hantera flera samtidigt anslutna klienter och vara beredd på att nya spelare ansluter när som helst. Serverns uppgift är att ta emot ett meddelande från en klient, processa innehållet och sedan ev. skicka ut uppdateringar till alla anslutna klienter. Du får själv välja hur din spelplan ska se ut och vad som ska ske vid tex. kollision med annan spelare eller spelplanens gränser. Fundera om det ska finnas hinder, dolda effekter eller annat.

Du får gärna testa ditt spel mellan flera datorer. Det går även bra att köra localhost då alla klienter ändå kommunicerar på olika portar…

c)

I detta moment ska du separera kontroll och data trafiken. Klienter ansluter fortfarande till spelet via TCP och via denna kanal sker all kontroll-signalering medan alla uppdateringar (ex. position) skickas via UDP.

d)
Nu ska servern använda multicast vid uppdateringen av klienterna…
ps. multicast-stöd kan variera på olika nätverk… (tex. att routrar inte joinar grupp)

e)
Nu ska du implementera stöd för IPv6 i ditt spel. En användare ska kunna skriva in en URL (domännamn) som ska översättas till en IP adress. Om det finns en IPv6 adress ska denna användas, annars används IPv4 adressen. IP adressen bör visas på lämpligt ställe i applikationen. Du kan använda ipv6.google.com eller någon annan IPv6 domän för att testa.

f)
Slutligen ska du utöka funktionaliteten med dynamisk upptäckt av tjänster på nätverket för att hitta startade spelservrar. Användaren ska kunna välja spelserver från en lista med alla servrar (namn) på det lokala nätverket.

Servern ska lyssna efter inkommande förfrågningar om tjänster på multicast adressen 239.255.255.250 och port 1900.

Klienten ska sända en förfrågan om en viss tjänst till multicast gruppen och alla servrar som har denna tjänst ska besvara förfrågan med en beskrivning av tjänsten.

Meddelanden skickas i klartext enl. följande:

SERVICE QUERY <service name>

SERVICE REPLY <service name> <instance name> <ip address> <port number>

Byt ut ***<text>*** med lämplig textsträng. I vårt fall ska vår tjänst <service name> sättas till “JavaGameServer” så att vi kan se varandras servrar/tjänster på resp. klient.

En artikel om “dynamic service discovery” finns på:
http://www.developer.com/services/article.php/3728576/Dynamic-Service-Discovery-with-Java.htm