Geometry Ball Tournament

Distribuerad Nätverksdesign

2015-03-04

Victor Assmundsson – a12vicas

Alexander Milton – b13alemi

Problembeskrivning

Uppgiften tillhanda utgick från ett redan färdigt spel, *Geometry Ball Tournament*, och krävde av studenterna att modifiera detta spel för att tillåta för upp till fyra deltagare över ett distribuerat nätverk.

Spelet är skrivet i Java och går ut på att knuffa en vit boll till motståndarlagets ”vägg” i spelplanen. Varje lag består av ett antal ”skepp” som vardera kontrolleras av en spelare. Spelare kan rotera sitt skepp medurs, moturs samt accelerera skeppet framåt.

För att distribuera spelet måste studenterna använda sina samlade kunskaper om kommunikation, distribuerade system, meddelandeöverföring och dataöverföringsprotokoll för att skapa en mjukvarulösning som tillåter för spelare att kopplas ihop som klienter till spelet.

Lösning

Lösningsmodellen som valts och presenteras i denna rapport utgår från en traditionell klient/server-design, där en spelare startar en personlig instans av spelet som kopplas upp till en värd som håller en global instans av spelet som hanterar och kontrollerar alla uppkopplade spelare. Spelarna är således klienter som är uppkopplade till en server.

Klienterna förmedlar spelarens input till servern som beräknar speltillståndet, men för att ge spelaren en bra spelupplevelse utan lag simuleras speltillståndet även lokalt hos klienten. Inputen skickas via UDP-paket till servern som efter att den beräknat om speltillståndet skickar ut det via UDP till alla anslutna klienter.

Detta gör att klienterna hela tiden får information om var alla objekt i spelet befinner sig, och kan rätta sig efter det. Dock skulle det ge ett dåligt intryck om objekten bara hoppade till nya positioner när det kommer ett nytt speltillstånd. Därför används datan inte som direkt representation utan spelaren ser ett tillstånd som är nära men inte exakt som serverns.

Detta görs med hjälp av surrogat-objekt hos klienterna som simulerar hur de rör sig vid nya speltillstånd, men samtidigt håller koll så att de inte avviker för mycket från servern. Så länge skillnaden är liten kommer surrogaten långsamt förflytta sig mot serverns tillstånd, men om skillnaden skulle bli alldeles för stor ansågs det bättre att ha ett mer korrekt speltillstånd än att spelaren inte ser några “omöjliga” beteenden, varpå objekten teleporteras direkt till sin egentliga position. (Syberfeldt, 2015) (Smed m fl., 2001)

Klient/server-modellen har fördelen att en centraliserad och skyddad exekveringsrymd förenklar distribuerad kommunikation och släpper krav på klienter att kunna upprätthålla kommunikation sinsemellan (likt en *peer-to-peer*-modell) samt ger omfattande skydd mot fusk bland klienterna, då de inte har tillgång till serverns skyddade data.

För att försäkra snabb kommunikation (millisekunder) mellan server och klienter valdes UDP-protokollet, som garanterar snabb överföring på bekostnad av risken för försummelseproblem (felaktig meddelandeordning, multipel eller ingen framkomst av meddelanden). Användningen av JSON underlättade formatering av meddelanden tack vare det smidiga gränssnittet och gjorde det enkelt att skapa lättviktiga datagrampaket.

Alternativa lösningsförslag

För meddelandestrukturen diskuterades om Javas inbyggda serialisering av klasser JSON Simple skulle användas. I slutändan valdes JSON då det ansågs vara lättare att hantera. En potentiell fördel med att använda serialiserbara klasser är att distinktionen mellan olika meddelandetyper kan bli lättare, då kontrollen blir vilken typ av objekt som skickas istället för att kontrollera parametrar.

Istället för *User Datagram*-protokollet hade istället *Transfer Control*-protokollet kunnat användas. I valet av transportprotokoll övervägdes den mest effektiva och pålitliga lösningen, där effektiv definierades som mest benägen att leverera paket snabbt och ofta. TCP har inte den generella egenskapen att leverera paket snabbt och är således potentiellt olämplig för syftet. Förlorade paket har ingen omfattande inverkan på programmets korrekthet eller exekvering, därför lämpade sig UDP betydligt bättre.

Ett alternativt lösningsförslag vore att placera serverinstansen direkt hos en klient. En klient skulle på så vis agera värd för hela spelet, likt många moderna onlinespel som tillåter klienter att skapa sina matcher utan att kräva värdskap av en extern server. Fördelen med detta är att ingen extern värd krävs för att hålla servern och en instans färre att synkronisera skulle eventuellt ge bättre prestanda. Detta skulle dock riskera att medföra vissa orättvisor, i form av kraftigt varierande latens mellan klienterna samt en markant ökad risk för fusk gentemot en global, “låst” server. Värden skulle ha direkt tillgång till servern då den i sådant fall skulle befinnas på dennes personliga system och hen skulle ha större möjlighet att manipulera serverns data. Dessutom skulle datatransportsträckan för värden bli näst intill obefintlig jämfört med de klienter som är uppkopplade till värden över ett nätverk.

Reflektion

Lösningens styrkor

Fuskskydd

Risken för att en spelare lyckas fuska är väldigt låg då servern kör alla beräkningar och berättar hur spelläget ser ut, medan klienterna bara kan säga vad de vill göra, exempelvis köra framåt eller svänga. Detta gör att så länge en spelare inte modifierar de andra spelarnas kommunikation med servern kommer hen inte påverka spelet på ett otillåtet sätt.

Detta verifierades med hjälp av minneseditorn Cheat Engine (Heijnen, 2014), ett program känt som hjälpmedel för att fuska i spel, såväl lokala som distribuerade. Cheat Engine låter bland annat användare precisera specifika minnesadress och manipulera värden samt ta kontroll över spels klockor för att öka eller sänka applikationens hastighet (så kallat “speedhack”).

I ett första test startades en server och en första klient på en dator, vilka var menade att exekvera fullt normalt och tillåta för verifiering av speltillstånden. En andra dator startade en ny klient, medvetet ämnad att försöka fuska, och kopplade upp sig mot servern. Därpå fokuserade användaren av fuskklienten på minnesadressen som höll värdet på fuskklientens lags poängmätare och ändrade det till ett högre värde. Effekten av detta blev en visuell konkurrens på fuskklientens sida. Servern forcerade det korrekta poängvärdet hos klienten och Cheat Engine skrev i respons över det med det högre värdet. Hos övriga klienter och hos servern syntes ingen skillnad, vilket indikerar att klientens försök att fuska inte haft någon effekt.

I det andra testet användes det inbyggda “speedhacket” av fuskklienten med målet att få övertag över andra spelare genom att kunna röra sig snabbare än dem. Resultatet av detta blev att entiteter såg ut att röra sig snabbare än vanligt, men omedelbart korrigerades till sina korrekta positioner i enlighet med serverns uppdateringar. Om något försämrades endast spelbarheten för den fuskande klienten som fick svårare att uppskatta spelets tillstånd när entiteterna på dennes sida utstod en stor mängd jitter. Speedhacking föreföll således i testet som ineffektivt tack vare lösningens design.

Lösningens svagheter

Om en spelare inte har en bra anslutning till servern utan alla paket tar en stund att komma fram föreligger en relativt stor risk för att den spelaren upplever lag, då servern inte beräknar om spelläget när den får paket som skickades innan läget beräknades. Så om ett paket med data till frame 5 kommer fram vid frame 10 kommer inte de 5 frames emellan beräknas om, utan paketets data används för att beräkna frame 11. Detta leder till att spelet har en relativt låg tolerans mot hög latens.

Spelet kan startas i flera instansiera på samma system men går inte att spela lokalt utan kräver flera datorer då varje fönster bara kan styra en spelare, och enbart när fönstret är aktivt. Detta gör att lokalt kan bara en spelare röra på sig i taget. Spelet går inte heller att spela utan att starta en server, vilket ökar minimiprestandan från att bara kunna köra klienten till att kunna köra klienten och servern samtidigt.

Lösningen specificerar inte ett exakt antal spelare utan tillåter ett arbiträrt antal, dock blir meddelandena för stora för att den hårdkodade bufferstorleken ska klara av dem om fler än 9 spelare är anslutna, vilket leder till att ingen av spelarna kan köra spelet utan att starta om servern.

Förbättringsförslag

Ett bra första steg för att förbättra upplevelsen är att implementera så kallad *Time warp* (Mauve m fl., 2002) för att ta hänsyn till spelarens input data vid rätt frame istället för att använda den när den kommer fram, då detta skulle öka toleransen för högre latency och ge fler möjlighet att få en bra spelupplevelse. En sak som eventuellt skulle kunna förenkla processen är att använda logiska klockor istället för systemets klocka.

Lärdomar

Under projektets gång har vidare kunskaper om kommunikation och synkronisering mellan distribuerade system erhållits, samt lärdomar om hur latensproblem mellan klient/server-applikationer kan motverkas och hur fusk från klienter kan förebyggas. Allt detta känns väldigt relevant och matnyttigt inför framtida arbeten som involverar speldesign, säkra realtidslösningar och applikationsutveckling till distribuerade system.

Feedback

Uppgiften var rolig och gav insikt i hur man kan applicera kursens innehåll på en någorlunda verklig situation.

Uppgiftsbeskrivningen var utdaterad och otydlig och är i stort behov av att ses över ordentligt eller göras om. Exempelvis står det i introduktionen till uppgiften att man ska tillhandahålla klienter till både desktop och Android-miljö, och i beskrivningen av hur spelet fungerar stod det att skeppen inte ska kunna bromsa men i underlaget som lämnades ut var en broms implementerad.

Under rubriken “Report tips” står det vilka krav som finns för rapporten, vilket inte är tips för rapporten.

Referenser

Mauve, M., Vogel, J., Hilt, V. & Effelsberg W. (2002). Local-Lag and Timewarp: Providing Consistency for Replicated Continuous Applications. *IEEE Transactions on Multimedia, 6*(1):47-57.

Smed, J., Kaukoranta, T. & Hakonen, H. (2001). Aspects of Networking in Multiplayer Computer Games. *The Electronic Library, 20*(2):87–97.

Heijnen, E. (2014). Cheat Engine (6.4) [Datorprogram]. Hämtad från http://www.cheatengine.org/

Syberfeldt, S. (2015) *Nätverksspel* [PowerPoint-presentation]. Hämtad 4 mars, 2015, från Högskolan i Skövdes webbsida: https://scio.his.se/portal