HÖGSKOLAN SKÖVDE
Institutionen för kommunikation och information

Dynamisk lokal fördröjning och Timewarp

Daniel Lundwall

Examensarbete i datalogi med inriktning mot dataspelsutveckling
30 hp C-nivå, vårterminen 2010

Dynamisk lokal fördröjning och Timewarp

Examensrapport inlämnad av Daniel Lundwall till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Sanny Syberfeldt.

2010-06-08

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Dynamisk lokal fördröjning och Timewarp

Daniel Lundwall

Handledare: Sanny Syberfeldt

Student-email: a07danlu@student.his.se

Sammanfattning

Arbetet ger en lösning till problem som uppkommer vid spel över nätverk orsakade av nätverksfördröjning. Ett vanligt problem är konsistensfel, olika spelare ser olika speltillstånd på sina skärmar. Lokal fördröjning är en teknik för att undvika uppkomsten av inkonsistenser. Genom att låta en spelarhandlings effekt fördröjas lokalt ges handlingen ett försprång över nätverket. Då ökar sannolikheten att båda parterna utför handlingen samtidigt. Timewarp är en teknik som reparerar speltillståndet om en inkonsistens uppstår. Genom att spola tillbaka tiden till den tid då speltillståndet var konsistent och därifrån beräkna ett nytt speltillstånd, baserat på alla efterföljande handlingar, inklusive den som orsakade inkonsistensen. Arbetet har undersökt effekten av en dynamisk lokal fördröjning som anpassar sig till aktuella nätverksförhållanden. I arbetet skapas ett enkelt spel som står till grund för en undersökning om hur teknikerna påverkar spelare. Resultatet är att spelare presterar bättre med en dynamisk lokal fördröjning än en statisk lokal fördröjning.

Nyckelord: Nätverksspel, Timewarp, dynamisk lokal fördröjning

Innehållsförteckning

1 Introduktion

Spel skapas huvudsakligen för att underhålla spelarna. En del av underhållningen kommer från de olika tävlingsmoment som finns i spel. Förr tävlade spelare genom att jämföra poäng alternativt att två personer spelade på samma maskin samtidigt. Idag sker en stor del av tävlandet över nätverk där de tävlande sitter och spelar vid var sin maskin. Dessa maskiner skickar meddelanden över nätverket för att förmedla en spelares handlingar. Ett nätverk kan innefatta fyra maskiner i ett och samma rum så väl som flera miljoner maskiner över internet. Tack vare dessa tekniska framsteg finns det idag stora turneringar och ligor där professionella spelare tävlar om en stor summa prispengar. Detta sätter då ett stort krav på att tekniker för spel över nätverket presterar enligt spelarens förväntningar.

Det största problemet med denna form av spel är de fördröjningar som uppstår i dessa nätverk. Det tar tid för att meddelanden att färdas fram och tillbaka över nätverket. En ytterligare komplikation är att denna fördröjning inte är konstant utan kan variera kraftigt beroende på övrig trafik över nätverket. Detta ger upphov till flera felaktigheter i ett spel. Ponera att en spelare i Bolivia spelandes ett bilspel mot en spelare i Sverige nyss påbörjade en kraftig inbromsning och nu står stilla. Det kommer ta ett tag innan den informationen når spelaren i Sverige och visas på dennes skärm. Detta kan resultera i att den spelaren i Sverige krockar med spelaren i Bolivia, allt på grund av fördröjningen som finns i det delade nätverket.

Lokal fördröjning innebär att spelarens handling fördröjs, från det att denna matar in den till att den sker, både lokalt och globalt sett. På så sätt ges meddelandet ett ”försprång” när det skall skickas över nätverket. Detta ökar sannolikheten att meddelandet kommer fram i tid och där med utförs samtidigt hos de båda parterna.

Timewarp är en algoritm som tar hänsyn till de handlingar som kommer in sent, det vill säga efter att det var tänkt att de skulle utföras. Algoritmen går tillbaka till det tidsteg i simuleringen då handlingen borde ha utförts och låter den och alla efterkommande handlingar beräknas så snabbt som möjligt.

I projektet skapas ett spel vari Timewarp tillsammans med dynamisk lokal fördröjning implementeras och sedan utförs undersökningar av olika egenskaper och hur dessa tekniker påverkar spelarens prestation. För att få ett tillförlitligt resultat har en simuleringsmiljö utvecklats för att efterlikna de egenskaper ett nätverk har. Resultatet mäts baserat på konsistens, spelarens resultat samt ett betyg satt av spelaren.

I kapitel 2 finns den bakgrundsfakta som krävs för att förstå arbetet. Kapitel 3 definierar problemet samt den vetenskapliga metod som står till grund för utvärdering av projektet. Kapitel 4 beskriver systemet, de mätningar som genomförts samt en analys av dessa mätresultat. Kapitel 5 innehåller en sammanfattning samt diskussion av projektet. Där finns även förslag på framtida arbeten.

2 Bakgrund

Detta kapitel syftar till att ge läsaren en faktagrund att stå på under läsningen av denna rapport. Först beskrivs grunderna inom nätverk och även internet. Efter detta beskrivs olika egenskaper, arkitekturer i ett nätverk samt klockor. Sedan ges en formell definition av konsistens och korrekthet. Kapitlet avslutas med en presentation av teknikerna lokal fördröjning och Timewarp.

2.1 Distribuerade System

Coulouris, Dollimore och Kindberg (2005) ger oss följande definition av ett distribuerat system: Ett system där hård- eller mjukvara som befinner sig i ett system av sammankopplade datorer som kommunicerar och koordinerar sina handlingar enbart genom att skicka meddelanden.

Detta är en väldigt generell definition som innefattar allt från internet till ett nätverk inbyggt i en bil.

2.2 Internet

Coulouris et al. (2005) beskriver internet som en världsomspännande sammankoppling av mindre nätverk, dessa ofta av varierande typ. ARPANET var det första storskaliga nätverket. Det står som grund för och utvecklades till det som idag refereras till som internet. Det var från början avsett för kommunikation med elektronisk post. Det visade sig att nätet var kapabelt att leverera dessa väldigt snabbt vilket gjorde det möjligt att skapa diskreta applikationer där två parter kunde agera parallellt trots stora geografiska skillnader. Senare togs ytterligare steg i den riktning vilket resulterade i att det nu var möjligt att köra kontinuerliga applikationer, se kapitel 2.7, över stora geografiska avstånd.

2.3 Nätverksprestanda

Här följer fyra faktorer som påverkar prestandan i ett nätverk:

- *Fördröjning*, från engelskans *latency*. Avser den tid det tar från det att sändningen av ett meddelande påbörjas tills dess att mottagaren har mottagit meddelandet, enligt Tanenbaum (2003)
- *Jitter* är benämningen på den variation av fördröjning som finns i ett nätverk. Denna variation beror på mängden trafik över nätverket, enligt Coulouris et al. (2005).
- *Bandbredd* är storheten som beskriver hur stor mängd data som kan skickas över nätverket inom en given tidsenhet, enligt Coulouris et al. (2005).
- *Paketförlust* avser det bortfall av meddelande när de skickas över nätverket.

Enligt Pantel och Wolf (2004) tar det 100 millisekunder för ljus att färdas från Europa till Australien. Detta betyder att en kabel av optiska fibrer liggandes på fågelsträckan från godtycklig punkt i Europa till godtycklig punkt i Australien tar det ett meddelande minst 100 millisekunder att nå fram. I verkligheten är denna tid längre, då kablar inte ligger optimalt, det behövs förstärkare av signaler, routrar och switchar som behöver tid för att utföra sitt arbete.

Mauve et al. (2004) ger värdena 20 millisekunder inom Europa, 40 millisekunder inom en kontinent och 150 millisekunder för ett världsomspännande nätverk.

Detta visa på goda möjligheter för en bra spelupplevelse över internet. Olika typer av spel kräver olika fördröjningar. Vad det gäller realtidsstrategispel hävdar Sheldon, Girard, Borg, Claypool och Agu (2003) att spelaren märken en försämrad spelupplevelse först när fördröjningen ligger på 500-800ms. Denna typ av spel är inte lika krävande som förstapersonsskjutare.

Armitage (2003) menar att Quake3-spelare som valt en server baserat på dess fördröjning föredrar en fördröjning lägre än 150-180ms. Är fördröjningen högre väljer spelaren att lämna servern.

Ovanstående resultat är tre gånger större än resultatet Quax, Monsieurs, Wim, De Vleeschauwer, och Degrande (2004) kommer fram till när de gör liknande test för ett annat förstapersonsskjutare, nämligen Unreal Turnament 2003. De visar att spelare finner en fördröjning på över 60 millisekunder som störande.

Armitage och Stewart (2004) försökte undersöka effekten av jitter hos spelare över internet. De kom fram till att en spelare med en fördröjning på 150-180 millisekunder sällan upplever jitter på över 30ms, ett värde som inte påverkar spelaren. Förser nätverket spelaren med en tillräckligt låg fördröjning är även jittret så pass litet att det inte påverkar spelarens prestation.

Chen, Huang och Lei (2006) hävdar att paketförlust är en viktig faktor. De använder Shen Zhou Online, ett flerspelar online rollspel, som testbas. De visar att spelare föredrar 200 millisekunder fördröjning och ett paketbortfall på 0.1% över 100 millisekunder fördröjning och 1% paketbortfall.

2.4 Arkitekturer

De nätverk som spel spelas över idag tillhör en av tre kategorier indelad efter deras logiska arkitektur enligt Smed, Kaukoranta och Hakonen (2001). Dessa har alla fördelar och nackdelar.

2.4.1 Klient och Server

Enligt Smed et al. (2001) kopplar ett centralt nav, eller en server, ihop alla spelare, eller klienter. Klienterna har ingen direkt inbördes koppling utan skickar alla meddelanden via servern. Detta gör att meddelanden inte skickas den snabbaste vägen vilket leder till en onödig extra fördröjning. Denna arkitektur skalar väldigt dåligt då servern får mer arbete att utföra för varje ny klient. Det blir dock lättare för administratorn av servern att kontrollera att allt går rätt till dvs. inget fusk och lättare att uppdatera mjukvara osv.

Enligt Brun, Safaei och Boustead (2006) och är denna typ av arkitektur den vanligaste för spel över nätverk. Detta tack vara att den är enkel att förstå och således också skriva kod för. Detta då det finns en instans av spelet som har det korrekta speltillståndet som hela tiden kan användas som referens för de övriga instanserna som ligger ute på klienterna. En nackdel är att om servern kraschar måste spelsessionen avslutas.

2.4.2 Peer to Peer

Smed et al. (2001) beskriver denna arkitektur som en sammankopplat nät av likvärdiga parter där alla har precis samma roll. Alla parter är logiskt direkt kopplade till alla och skickar sina meddelanden till alla parter.

Enligt Coulouris et al. (2005) skalar Peer to Peer mycket bättre än klient och server då för varje ny part som tar del av nätverk ökar också mängden datorer som delar på arbetsbördan. Smed et al. (2001) hävdar dock att denna teknik skalar dåligt på grund av att det inte finns hierarkisk struktur.

Ett problem som Baughman och Levine (2001) lyfter fram är att spelarna lättare kan fuska då det inte finns ett centralt nav som agerar övervakande organ. Fördelen med att inte ha ett sådant nav är att det inte heller finns uppenbar bristningspunkt.

2.4.3 Replikerade serverar

Smed et al. (2001) och Coulouris et al. (2005) beskriver replikerade serverar som en kombination av de två ovanstående arkitekturerna. Ett nätverk av servrar enligt Peer to Peer-konceptet jobbar tillsammans för att förse klienterna med den tjänst de kräver. För ett spel behöver då speltillståndet finnas speglat på alla servrar. Det innebär att om en servers tillstånd förändras måste övriga servrars tillstånd genomgå samma förändring. Detta problem leder lätt till konsistensproblem, det vill säga att servrarnas speltillstånd skiljer sig åt. Replikerande servrar kombinerar flera av de fördelar som de olika arkitekturerna har. Fördröjningen kan sänkas om klienter ansluter till server mot vilken klienten har lägst fördröjning. Det finns inte heller en bristningspunkt. Systemet kan lätt utökas vid behov genom att lägga till fler servrar.

2.5 Klockor

Enligt Coulouris et al. (2005) har varje dator en inbyggd klocka. Detta gör det möjligt för datorer i ett nätverk att samordna sina handlingar. Problemet är dock att dessa klockor inte är perfekta. Om klockorna avläses vid samma tidpunkt hos två olika datorer i ett nätverk behöver dessa inte ge samma svar. Detta på grund av att klockorna driver, från engelskans *drift*. Det vill säga att de går olika snabbt. Om ett nätverk lyckas synkronisera alla klockor skulle det hjälpa föga. Efter en tillräckligt lång tid skulle dessa vara tillräckligt ur fas för att skapa problem för applikationerna.

Coulouris et al. (2005) hävdar att det inte finns tillräckligt praktiska metoder för att synkronisera klockor i ett vanligt nätverk. Den vanligaste lösningen är att låta en central klocka uppdatera klockorna i nätverket. Denna centrala klocka får sin tid från det globala positionssystemet även känt som GPS. När denna centrala klocka skall uppdatera en annan klocka i nätverket kommer det inte bli exakt på grund av de fördröjningar och jitter som finns på nätverket.

2.6 Konsistens och Korrekthet

Konsistens i ett distribuerat system innebär att tillstånden hos alla inblandade parter är identisk vid en given tidpunkt i simuleringen. Mauve et al. (2004) angående konsistens ger oss följande: Vi har en mottagningsfunktion som ser ut så här:

$$R_i(t, o_{j,t^o,t^*}) = \begin{cases} falskt\ om\ o_{j,t^o,t^*}\ mottages\ av\ i\ efter\ t \\ sant\ annars \end{cases}$$

Ekvation 1

Resultatet av Ekvation 1 blir falskt om handlingen *o*, skapad från part *j* vid tiden *t*° där *t** är tiden då operationen skall utföras, mottages av *i* efter tiden *t*, annars sant. Till exempel kan part *i*=1 och part *j*=2, tiden *t* = 5, *o* kan vara en handlingen skjuta, skapad vid *t*°=3 som skall utföras vid *t**=4. Vid dessa värden returnerar funktionen

falskt, handlingen har mottagits för sent, $t^* < t$. Med hjälp av denna funktion kan konsistens definieras som följer:

$$\forall t, i, j \left| \forall t^* \leq t, o_{w,t^o,t^*} \in O \right| R_i\left(t, o_{w,t^o,t^*}\right) \wedge R_j\left(t, o_{w,t^o,t^*}\right) \Rightarrow \left(S_{i,t} = S_{j,t}\right)$$

Uttryck 1

En replikerad kontinuerlig applikation försäkrar konsistens omm Uttryck 1 ger sant. Det vill säga att om vid tiden *t* tillstånden *s* hos part *i* vid tiden *t* är lika med tillståndet *s* hos part *j* vid tiden *t* om båda parterna har mottagit alla handlingar som skall utföras före tiden *t*. Det betyder också att två parter som inte har mottagit alla operationer för att kunna beräkna det korrekta tillståndet fortfarande kan anses vara konsistenta. Utan detta tillägg kan inte konsistens garanteras vid närvarande av minsta nätverksfördröjning. Det betyder även att om alla parter mottagit alla handlingar måste deras tillstånd vara samma. Tiden *t* beror på de olika parternas interna klockor, alltså inte en gemensam klocka. Det spelar ingen roll om de läser av klockorna samtidigt, sett till en gemensam klocka, och då får samma resultat eller inte.

Detta garanterar dock bara att alla parter erhåller samma tillstånd, men inte att det tillståndet är korrekt. Mauve et al. (2004) definierar korrekt tillstånd som det tillstånd som hade nått om alla operationer utförts på en instans av applikationen vid deras respektive t^*. Parten *p* definieras som den virtuella part som håller detta perfekta tillstånd. Då kan korrekthet ges av Uttryck 2:

$$\forall t, i, \left| \forall t^* \leq t, o_{w,t^o,t^*} \in O \right| R_i\left(t, o_{w,t^o,t^*}\right) \Rightarrow \left(S_{i,t} = S_{p,t}\right)$$

Uttryck 2

Alltså, vid godtycklig tid *t* måste parten *i* hålla samma tillstånd som *p* om parten *i* har mottagit alla handlingar som skall utföras före tiden *t*.

2.7 Diskreta och kontinuerliga applikationer

Mauve et al. (2004) nämner två typer av distribuerade applikationer. De diskreta applikationerna baserar sitt tillstånd på användarens handlingar. Detta skiljer sig från de kontinuerliga applikationerna där tillståndet även ändras som en funktion av tiden. Majoriteten av dagens spel tillhör den sistnämnda typen. Fiender i spel rör sig allt eftersom tiden går. Konsistens i diskreta applikationer är ett väl utforskat område. Det samma kan inte sägas om kontinuerliga applikationer.

2.8 Lokal fördröjning

Mauve (2000) föreslår lokal fördröjning som en förebyggande lösning till de temporära inkonsistenser i speltillståndet som uppstår på grund av sent inkommande meddelande. Genom att fördröja alla händelser med en viss tid ges på så sätt meddelandet ett försprång över nätverket och därmed ökar sannolikheten att det kommer fram före den tid då handlingen var avsedd att utföras. Då kommer handlingen utföras samtidigt för de båda spelarna.

Stuckel och Gutwin (2008) samt Mauve et al. (2004) menar att denna fördröjning minskar spelbarheten då spelet lätt kan bli oresponsivt om fördröjningen är för lång. Detta måste dock vägas mot att spelbarheten försämras på grund av det ökade antalet temporära inkonsistenser vid en låg fördröjning. Denna avvägning undersöker Stuckel et al. (2008) och når slutsatsen att spelaren presterar bättre med lokal fördöljning om nätverksfördröjning ligger på 100-500 millisekunder än om applikationen hade varit utan lokal fördröjning.

Mauve et al. (2000) presenterar ett par riktlinjer för att välja ett värde på den lokala fördröjningen. Det måste vara så högt att det avvärjer en stor del temporära inkonsistenser men tillräckligt lågt för att spelaren ska kunna tolerera fördröjningen.

Det lägsta möjliga värdet på den lokala fördröjningen skall vara lika med det maximala genomsnittsvärdet av fördröjningarna mellan parterna. Om parternas klockor inte är synkroniserade skall differensen adderas på detta lägsta värde.

Det högsta möjliga värdet beror på det högsta värdet en spelare kan tolerera. Mauve et al. (2004) hävdar att 80 till 100 millisekunder är omärkbart oavsett applikation. Detta kan te sig högt då Quax et al. (2004) hävdar att 60 millisekunder är för hög fördröjning för ett nätverk. De verkar ännu högre när mar tar i åtanke att eliten inom dataspel valde att inte byta ut deras CRT-monitorer mot nya LCD-monitorer på grund av den sämre respontiden trots att det bara rörde ett tiotal millisekunder.

Mauve et al. (2004) fortsätter med att beskriva de två fall som kan uppstå. Det bästa vore om det lägsta möjliga värdet är mindre än det högsta möjliga värdet. Då kan ett värde mellan de två väljas. Om så inte är fallet får en avvägning göras, låg svarstid men många temporära inkonsistenser alternativt hög svarstid och få temporära inkonsistenser. Mauve et al. (2004) hävdar att detta fenomen sällan sker.

Stuckel et al. (2008) ger också ett förslag för hur den lokala fördröjningen kan döljas för spelaren. Detta genom att låta den lokala applikationen visa en inledningsanimation som sedan leder till handlingen. Till exempel en att ladda upp ett slag. Handlingen skickas direkt till de andra parterna under tiden denna animation visas för den spelare som utfärdar handlingen.

2.9 Timewarp

Timewarp är en algoritm designad för att lösa temporära inkonsistenser. Mauve et al. (2004) använder Timewarp tillsammans med lokal fördröjning som en garanti för att uppnå konsistens och korrekthet i en kontinuerlig applikation i ett distribuerat system om meddelande leverans är garanterat. När en inkonsistens upptäcks, det vill säga när en handling anländer efter den tid den skulle utföras, löser Timewarp denna temporära inkonsistens. Mauve et al. (2004) beskriver algoritmen i fyra steg.

- Steg 1. Vänta en konstant tid *T*. Under denna tid samlas alla operationer, både lokala men också de från resterande parter, in. Dessa lagras i en lista L_i^o för parten *i*. Denna lista är sorterad efter *t**, tiden då operationen skall utföras. Den lägsta nyinkomna *t** lagras som t_w^o.
- Steg 2. Finn t_w^s, tillståndet *s* hos parten *w*, sådant att $t_w^s = \max\{t | (S_{i,t} \in L_i^s) \wedge (t < t_w^o)\}$. Det vill säga finn det tillstånd som rådde precis innan t_w^o. Listan L_i^s håller de beräknade tillstånden och initieras till att bara innehålla S_{P,t_i^I} där t_i^I är tiden då initieringen sker.
- Steg 3. För alla *t* då $t_w^s < t < t^c$, där t^cär den nuvarande tiden, där det finns minst en operation sådant att $o_{k,t^o,t} \in L_i^o$ i stigande ordning. Så finnes S_{i,t_w^s}. Detta tillstånd kallas för iterationens bastillstånd och betecknas med S_{i,t^b}. Från S_{i,t^b} och framåt räknas nya tillstånd ut enligt applikationens regler. Dessa skriver över motsvarande $s_{i,n} \in L_i^s$ där $t^b \leq n < t$.
- Steg 4. Räknar ut s_{i,t^c} och lagrar denna i L_i^s.

Som exempel kan ett spel med två rymdskepp vars mål är att skjuta varandra antas. Ponera att spelsessionen har uppdaterats 100 gånger och där med befinner sig vid

tidsteg 101. Ur spelare etts perspektiv kan algoritmens arbetsgång se ut som följer. Vid steg 1 väntar applikationen 20 millisekunder och samlar in två handlingar. Spelare ett gasar, spelare två skjuter och åker mot spelare ett. Spelare tvås handling borde ha utförts vid tidssteg 95, den har inkommit för sent. I steg 2 laddas tillståndet som rådde före tidssteg 95, det vill säga tidsteg 94. I steg 3 beräknas alla handlingar, som finns i listan av handlingar, som skall utföras efter tidsteg 94 fram till tidssteget före den nuvarande tiden, det vill säga tidssteg 100, i stigande ordning. Vid varje ny beräkning ersätts motsvarande tillstånd i tillståndslistan med det nyligen beräknade tillståndet, det vill säga, det nya tillståndet för t.ex. tidssteg 96 ersätter det gamla tillståndet vid tidssteg 96. I steg 4 beräknas slutligen det nuvarande tillståndet, det vid tidssteg 101. Det är först efter denna uppdatering som spelare ett får se effekten av spelare tvås skott. Detta kan skildras på ett par sätt. Det kan vara så att skottet träffade spelare ett som då borde varit död sen uppdatering 95 men som levt för länge. Då kommer det se ut som spelare ett omedelbart dör. Är skottanimationen kortare än 5 uppdateringar kan spelare ett dö utan att ha sett ett skott. Är det så att skottet inte träffade och animationen är kortare än 5 uppdateringar så ser spelare ett ingen effekt alls av skottet.

Mauve et al. (2004) lyfter fram ett par problem som inte löses av lokal fördröjning och timewarp samt problem som uppstår på grund av teknikerna.

Situationer kan uppstå där spelaren fattade ett felaktigt beslut baserat på den visuella informationen denna hade att tillgång under en temporär inkonsistens. Eftersom informationen var felaktig blir troligtvis även beslutet felaktigt.

Problem kan också uppstå med tanke på att algoritmen inte garanterat blir klar med sitt arbete under samma uppdatering av speltillståndet som den påbörjades.

De lagrade värdena för speltillstånd och operationer kan inte lagras för alltid. De behöver successivt raderas. Finns det en garanterad leveranstid för meddelandena kan listan enkelt rensas allteftersom simuleringen fortskrider. Finns ingen garanterad tid bör en tid väljas som gör det väldigt osannolikt att ett speltillstånd eller en operation som har raderats behövs i algoritmen.

Korrigeringen som sker kan se mycket märklig ut för spelaren. Det kan handla om en spelare återuppstår eller dess position drastiskt förändras. Det finns dock tekniker för att dölja dessa visuella fel. Till exempel visningen av kritiska händelser fördröjas för att på så sätt öka sannolikheten att felen inte inträffar.

3 Problemformulering

Ett problem med Mauve et als (2004) arbete är att den lokala fördröjningen inte tar hänsyn till faktiska förhållanden som råder på nätverket vid en given tidpunkt. Därför skall en dynamisk lokal fördröjning som baserar sig på nätverkets förhållande implementeras. Syftet är att förse spelare i nätverk med en teknik för att hantera fördröjning för att uppnå tillräckligt hög grad av korrekthet och konsistens samt en så låg lokalfördröjning att spelaren skall känna sig nöjd. Denna teknik är förhoppningsvis dynamisk lokal fördröjning i kombination med Timewarp.

Problemet med nätverksfördröjning i spel med avseende på konsistens och korrekthet kan lösas till hög grad med hjälp av en tillräckligt stor lokal fördröjning. Detta leder dock till problem då spelet blir oresponsivt om fördröjningen är för hög. Det finns alltså en avvägning mellan spelbarhet i avseende på svarstid kontra korrekthet och konsistens. Denna avvägning skall i detta arbete lösas genom att ha en dynamisk lokal fördröjning. Arbetet syftar också till att avgöra vilka faktorer som skall påverka mängden lokal fördröjning och dess förändringshastighet.

Den dynamiska fördröjningen kommer att basera sig på antalet temporära inkonsistenser som har skett inom den senaste tiden. Hur stort detta tidsfönster bör vara kommer att undersökas i arbetet. Hur dessa värden kommer påverka den lokala fördröjningen är också ett problem som skall lösas.

Målet är att ha en så låg lokal fördröjning som möjligt vid alla tidpunkter samtidigt som den inte fluktuerar för mycket. Att ha en låg lokal fördröjning förser spelaren med en låg responstid men ger upphov till fler temporära inkonsistenser. Dessa inkonsistenser löses med hjälp av Timewarp. På så sätt kan en låg lokal fördröjning bibehållas samtidigt som långvariga konsistensproblem undviks.

Att undersöka effekten av denna nätverkslösning förutsätter att det är möjligt att kontrollera förhållandet på nätverket. Detta är omöjligt att göra över internet. Därför skall dessa tekniker implementeras i ett litet lokalt nätverk bestående av två datorer, kopplade direkt via en TP-kabel, utan switchar eller routrar som stör, nätverkskort till nätverkskort. Detta för att försäkra att en så låg och statisk fördröjning som möjligt erhålls. Då kan en nätverksimulator implementeras. På så sätt kan fördröjning, jitter och meddelandebortfall kontrolleras för att få ett nätverk som kan användas för att utvärdera diverse situationer.

För att ytterligare utesluta parametrar kommer testpersonerna att möta en datorstyrd motståndare. Poängen är att alltid möta en spelare som gång på gång agerar och presterar enligt ett förutbestämt mönster. Denna motståndare kommer att följa väldigt enkla riktlinjer. Datorspelaren kommer åka mot testpersonens senaste position och när denna kommer inom räckvidd för lasern kommer datorspelaren att avfyra en stråle. Lösningen med datorstyrda spelare återfinns i Liang och Boustead (2006).

3.1 Metodbeskrivning

Ett exempelspel kommer att skapas och användas som testmiljö. Spelet är en tvådimensionell shooter sedd ovanifrån där två spelares rymdskepp möts. Spelarna får poäng för att skjuta sönder den andra spelarens skepp genom att skjuta denna med en laserstråle med begränsad räckvidd. Blir ett skepp träffat av en stråle försvinner detta en kort stund från spelet för att senare återuppstår på en slumpvald plats på spelplanen. Spelplanen är en plan yta utan några hinder. Skärmens kanter kan betraktas som väggar av spelaren. Spelaren kan svänga, accelerera, retardera och

skjuta. Spelet är byggt på en peer-to-peer-arkitektur. Detta spel är en efterliknelse av det spel som Mauve et al. (2004) har skapat för sina experiment. Detta spel ter sig bra för experiment då det är ett enkelt spel med få variabler som kan resultera i felaktig data. Det finns begränsat antal handlingar, vilket gör speltillståndet och listor av operationer enkla att hantera för timewarp. Det finns kritiska händelser, som att skjuta sönder en annan spelare, som tydligt kan avslöja temporära inkonsistenser. Det är enkelt att räkna poäng och se hur bra en spelare presterar. Det är känsligt för all form av fördröjning, att skjuta först är skillnaden mellan vinst och förlust.

Utvärdering och slutsats kommer att basera sig på tre kriterier: konsistens, spelarresultat samt spelarupplevelse.

Konsistensmätning sker genom att räkna antalet temporära inkonsistenser men också under hur lång tid dessa varade. Detta är en enkel mätning tack vare att dessa värden finns att tillgå då de står till grund för Timewarp. Det går enkelt att avgöra under hur lång tid inkonsistensen varade, detta genom att subtrahera tiden då inkonsistensen reparerades med tiden då handlingen skulle varit utförd.

Spelarens resultat mäts genom de poäng denna blir tilldelad, genom att skjuta sönder sin motståndare, minus antalet gånger spelaren själv blir skjuten, under en omgång. Detta är ett objektivt mått på hur bra teknikerna presterar. Högre värde indikerar bättre spelbarhet. Hade fördröjningen på nätverket haft en för kraftig negativ effekt hade spelarna inte lyckats träffa varandra vilket hade resulterat i en låg poäng. Detta följer samma utvärderingsprincip som Liang et al. (2006) använder i sin utvärdering av Quake3 med Timewarp spelat av datorstyrda spelare med skillnaden att i detta fall är bara en av spelarna datorstyrd. Claypool och Claypool (2006) refererar till ytterligare åtta artiklar som använder sig av spelarens resultat eller poäng på olika sätt för att undersöka effekten av fördröjningar på spelarens prestation. Skillnaden är att dessa artiklar undersöker när en spelare presterar sämre på grund av fördröjningarna. Metoden som skall användas försöker undersöka när dessa tekniker gör att spelaren presterar bättre än utan teknikerna. Även då bör spelarens resultat vara ett mått på hur mycket fördröjningen, men också teknikerna för att förminska effekten av fördröjningen, påverkar spelaren.

Intervjufrågor angående spelarens upplevelse kommer att användas för att styrka det föregående mätvärdet. Visar svaren på intervjufrågorna på samma upplevelse som spelarresultatet visar har på så sätt spelarresultaten styrkts ytterligare. Beigbeder, Coughlan, Lusher, Plunkett, Agu och Claypool (2004) visar dock att så inte alltid är fallet. Spelare kan tycka att de spelar dåligt då de statistiskt sett spelar bra.

Efter varje testfall kommer spelaren att bli ombedd att ge ett betyg hur de upplevde fallet. Dessa betyg följer ”Mean Opinion Score”-principen som Dick, Wellnitz och Wolf (2005) introducerar för att utvärdera spelarens upplevelse i nätverksspel. Mean Opinion Score skapades som en subjektiv kvalitetsutvärdering för bild och ljud. Spelaren kan sätta betyg 1-5 för varje testfall enligt de riktlinjer och motiveringar Dick (2005) et al. Presenterar som följer:

1. Oacceptabel miljö, omöjligt att spela.
2. Väldigt irriterande miljö, inte acceptabel.
3. Tydligt bristande miljö, trots det fortfarande acceptabel.
4. Små brister i miljön, väldigt spelbart.
5. Inga märkbara brister i miljön, perfekt spelbarhet.

Spelaren kommer att få spela spelet tre gånger för att täcka in de viktigaste testfallen:

- Referensfallet. Ingen lokal fördröjning och ingen fördröjning eller jitter på nätverket.
- En verklighetstrogen simulering av nätverket, både med fördröjning och jitter. Klienterna använder en statisk lokal fördröjning efter de principer som Mauve et al. (2000) presenterar. Även timewarp kommer att användas.
- En verklighetstrogen simulering av nätverket, både med fördröjning, jitter samt enkel beräkning av dynamisk lokal fördröjning. Även timewarp kommer att användas.

Spelaren kommer inte få veta vilket fall den spelar.

De två sistnämnda testfallen kommer båda ges samma förhållanden på nätverket för att kunna göra en godtagbar jämförelse. Alla de spelare som testar spelet har erfarenhet av nätverksspel.

Alla tester bortser från paketbortfall. Ett meddelande som faller bort skickas igen och kommer då fram senare om det inte hade fallit bort första gången. Det är alltså bara en till faktor, utöver fördröjning och jitter, som gör att meddelandena kommer sent. Den blir redundant för simuleringen. Dock kommer förändringen av nätverksfördröjningen som sker när meddelanden tvingas ta en ny väg mellan parterna att simuleras.

För varje testfall kommer spelaren att spela en träningsomgång som vara i 30 sekunder för att anpassa sig till förhållandena. Samma princip som Stuckel et al. (2008) använder.

Med en enkel beräkning av dynamisk lokal fördröjning är tanken att detta värde skall basera sig på antalet temporära inkonsistenser inom den senaste tiden. Värdet på fördröjningen kommer vara i direkt relation till antalet temporära inkonsistenser. Har det skett många temporära inkonsistenser kommer den lokala fördröjningen att öka och vice versa. Detta kan dock ge upphov till en väldigt fluktuerande lokal fördröjning vilket kan te sig icke önskvärt av spelaren. Målet är att få ett stabilt värde men som fortfarande förändrar sig tillräckligt snabbt för att anpassa sig till förhållandena på nätverket. Då kan värdet vara så lågt som möjligt men samtidigt avvärja en stor det temporära inkonsistenser samt förse spelaren med en så låg lokal fördröjning som möjligt. Dessa beräkningar kommer ske med hjälp av ett styrsystem där inspiration kommer från läran om reglerteknik.

Då den data som skickas endast är spelarens handlingar kommer storleken på paketen vara väldigt små. Då är nätverkets bandbredd inte ett problem. Nätverket som används i denna rapport avser ett fysiskt litet nätverk men logiskt sett skall det efterlikna ett vanligt nätverk och de förhållanden som råder där. Jittret kommer att vara ett helt slumpmässigt värde som adderas på den konstanta basfördröjningen. För att skapa en större varians, så som när paket tvingas ta en ny väg över internet, kommer större och längre ökningar av nätverksfördröjningen också att skapas. Applikationen är skriven för att klara att brukas av två datorer oberoende av deras prestanda. Uppdateringarna kan ta olika lång tid på de olika datorerna utan att något icke önskvärt sker. Då tillåts också timewarp att ta den tid den behöver för att utföra sitt arbete.

4 Genomförande

Detta kapitel beskriver implementeringen av det producerade systemet, redogör för förenklingar samt antaganden.

4.1 Det producerade systemet

Metoden testas genom den producerade applikationen, ett enkelt peer-to-peer spel med timewarp, statisk lokal fördröjning samt dynamisk lokal fördröjning. Detta har implementerats i C# tillsammans med XNA 3.1. Detta då XNA erbjuder gott stöd för nätverkskommunikation.

4.1.1 Grundläggande kod

Koden ligger i två olika klasser, ship samt game. Ship sköter uppdateringen av de individuella skeppen. Game innehåller all spellogik, nätverkskod, utritningskod och inputkod. De grundläggande objekt som existerar är två skepp, en lista med lagrade operationer och en lista med lagrade tillstånd. Både operationer och tillstånd är beskrivna av två structs. En operation definieras med den tid den skall utföras, den tid den blev skapad, vilket skepp som skapade operationen och vilken den faktiska operationen är. Operationen kan vara en av fem olika; svänga höger eller vänster, skjuta samt köra bakåt eller framåt. Listan är sorterad i fallande ordning baserat på tiden då operationen skall utföras. Tillstånd beskrivs av följande faktorer: tid då tillståndet existerade samt position, vinkel, hastighet, poäng, om spelaren skjöt, om spelaren var död och eventuell återstående tid till spelaren skall återuppstå. Där alla förutom den första faktorn är lagrad för båda skeppen. Listan är sorterad i fallande ordning grundat på den tid då tillståndet existerade.

All tid är beskriven kontinuerligt, inte diskret. Tiden som mäts är den riktiga ”väggklockan”, i stället för att låta varje uppdatering vara ett tidssteg. Detta för att vara oberoende på hur snabbt de olika datorerna i nätverket arbetar. Det betyder också att två operationer kan särskiljas trots att de utförs under samma uppdatering. Detta gör att den som utförde en operation först får se effekten av denna.

4.1.2 Beskrivning av spelloopen

För att förstå applikationens arbetsgång beskrivs den grundläggande spelloopen nedan.

1. Tiden då den denna nya uppdatering började lagras.
2. Nätverksfördröjningen beräknas och sätts. Detta görs genom en hjälpfunktion i XNA. Jittret adderas på en basfördjörning.
3. Spelarens input läses av. En operation skapas av denna och läggs till i operationslistan som sedan sorteras.
4. Beräknas ny lokal fördröjning.
5. Om en datorstyrd spelare är aktiverad agerar denna nu.
6. Den operation som skapades i steg 3 skickas över nätverket till motståndaren.
7. Nätverket uppdateras. Det är då all data överförs.
8. Motståndarens operationer mottages och lagras i listan av operationer som sedan sorteras.
9. Om operationen som mottages i steg 8 ankom efter det att den var tänkt att utföras körs timewarp-algoritmen.

10. Spelarna uppdateras. De operationer vars utförningstid ligger mellan den tid då föregående uppdatering började fram till att nuvarande uppdatering började används för att beräkna nya positioner för skeppen.
11. Skottkollision detekteras och hanteras.
12. Det tillstånd som beräknades i steg 10 lagras i listan av tillstånd som sedan sorteras.
13. Gamla operationer och tillstånd tas bort ur de båda listorna.

Nedan följer en förklaring för de mer komplicerade stegen.

4.1.3 Beräkning av lokal fördröjning

Temporära inkonsistenser uppstår på grund av att motspelarens lokala fördröjning är för låg. Målet är att ha en så låg lokal fördröjning som möjligt utan att det uppstår temporära inkonsistenser. I varje uppdateringscykel skickar spelarna ett värde för hur många temporära inkonsistenser som har skett inom ett tidsfönster. Tidsfönstrets storlek är anpassningsbart. Har inga temporära inkonsistenser uppstått inom detta tidsfönster sänks den lokala fördröjningen lite. För att se om den lokala fördröjningen bör höjas undersöks om antalet temporära inkonsistenser inom tidsfönstret överstiger ett visst tröskelvärde; om så är fallet höjs värdet för den lokala fördröjningen. Temporära inkonsistenser kommer i grupp. Därför justeras värdet först efter att effekten av den förra justeringen har visats. Justeras den lokala fördröjningen justeras den igen först när hela tidsfönstret har passerat. På så sätt uppnås också ett mjukare beteende.

4.1.4 Den datorstyrda spelaren

Den ena av spelarna i experimenten är en datorstyrd spelare. Det grundläggande tillvägagångssättet är att köra mot sin motståndare så att denne skall komma inom räckvidd för lasern, då stannar skeppet och avfyrar lasern upprepade gånger till motståndaren har blivit träffad. Kommer motståndaren utom räckhåll kör datorspelaren ikapp igen. Det första steget är att beräkna om och åt vilket håll skeppet skall rotera. En vektor dras från motståndaren till det datorstyrda skeppet. Denna vektor normaliseras och tillsammans med skeppets riktning beräknas en skalärprodukt för att på så sätt veta hur stor vinkel skeppet måste svänga. Sedan beräknas en kryssprodukt baserad på samma vektorer. Resultat visar om skeppet skall rotera med eller moturs samt om det skall köra framåt, backa eller endast rotera. Datorn roterar bara skeppet om vinkeln mellan skeppen är tillräckligt stor. Detta minskar precisionen hos den datorstyrda spelaren, och på så vis sätts svårighetsgraden. Det andra steget är att avgöra om skeppet borde skjuta, detta genom att beräkna avstånden mellan skeppen. Är skeppen för nära varandra kör inte skeppet, det bara roterar.

En nackdel med detta tillvägagångssätt är att det är väldigt dåligt på att hantera den lokala fördöljningen. En människospelare ser hur stor vinkeln är och svänger på känsla. Den trycker inte en gång för att i nästa uppdatering kolla om det var tillräckligt. Så gör datorspelaren då den hela tiden baserar sitt beslut på värden beräknade denna uppdatering och har ingen vetskap om vilka operationer den initierade förra uppdateringen. Det ger problem då resultatet inte kan ses förens vid nästa uppdatering. Det resulterar i att datorspelaren hela tiden svänger med- och moturs om vartannat.

4.1.5 Nätverkskod

Nätverkoden i projektet är på en hög nivå. Detta då XNA sköter lågnivådetaljer. Allt som krävs är att definiera vilka datatyper som skall skickas och i vilken ordning. Sedan sköter XNA resten själv. I varje uppdatering skickas en operation och antalet inkonsistenser som har skett. Har spelaren inte matat in en ny operation skickas en tom operation. Detta kan te sig onödigt men gör det lättare att beräkna rätt tillstånd. Antalet inkonsistenser som uppstår hos motspelaren är det som står till grund för beräkning av den lokala fördröjningen.

4.1.6 Timewarp

Timewarp körs om en operation tas emot efter det att det var tänkt att den skulle utföras. Då ersätts det aktuella tillståndet med det tillstånd som rådde uppdateringen innan det att den sena operationen skulle ha utförts. Alla operationer som följer den sena, samt eventuella operationer som hände under samma uppdatering men före den sena operationen, körs så snabbt som möjligt. Efter att varje operation har körts har ett nytt tillstånd beräknats. Detta tillstånd skall inte alltid sparas permanent i listan av tillstånd då det inte alltid är beräknat på den sista operationen som utfördes den uppdateringen. Det sparas dock temporärt, om ett nytt tillstånd beräknas baserat på en operation som utfördes samma uppdatering skrivs den föregående överskrivningen över. Detta upprepas tills dess att den sista operationen för uppdateringen har körts, då sparas tillståndet permanent. Alla operationer fram till aktuell tid utförs. Det finns dock inte ett sparat tillstånd för den aktuella uppdateringen, därför skapas ett extra tillstånd som bara är till för att skrivas över. Detta skiljer sig lite från den algoritm som Mauve et al. (2004) presenterade. Främst för att ett tillstånd baserar sig på operationer som skall utföras mellan den aktuella uppdateringen och den föregående, detta då kontinuerlig tid används i applikationen.

4.1.7 Uppdatering och beräkning av nytt tillstånd

För att uppdatera skeppens tillstånd söks listan av operationer igenom för att hitta operationer som skall beräknas denna uppdateringscykel. Detta blir lite mer komplicerat då applikationen mäter tid kontinuerligt, till skillnad från diskret. Det vill säga operationers tid anges inte i ett heltal som beskriver den uppdateringscykel den bör utföras i. Det är ett decimaltal med den exakta tiden operationen skall utföras. Därför kollas ett spann mellan den tid då nuvarande uppdateringscykel började och den förra började. Operationen som ligger inom detta spann utförs i stigande ordning, d.v.s. den ordning de inkom i. På så vis klumpas inte operationer ihop utan det blir möjligt att särskilja dem, vilken operation under uppdatering bör utföras först. Efter att varje operation har körts undersöks det om den var en skjutoperation, om så är fallen körs kollisionskoden.

Det kan te hända att en operation i listan är tom. Detta betyder att ingen förändring av inputen har skett. Den tomma operationen är en markör för att tillståndet skall uppdateras och beräknas på de nuvarande värdena utan modifikation från inputen. Det vill säga att skeppet skall fortsätta åka framåt om inget annat har skett. Det kan te sig redundant att fylla listan med tomma operationer men är lättare att genomföra. Alternativet är att man försöker lista ut var mellan operationer som bara en tom uppdatering skall ske. Detta är svårt då applikationen skall kunna köras på olika snabba datorer med samma resultat. Mellan två operationer kan en snabb dator hinna med tio snabba uppdateringar medan en långsammare bara hunnit med tre. Detta

skulle resultera i att världen, däribland friktion, har påverkat ett skepp tio gånger på en dator men bara tre på en annan. Då skulle tillstånden skilja sig åt för de olika parterna.

4.1.8 Kollision

Kollisionsberäkningen är väldigt enkel tack vare datatypen Ray som finns i XNA. En stråle skickas från skeppet. Om denna stråle skär motståndarskeppet har en kollision skett. Om så är fallet tas det träffade skeppet ur spel och den andre spelaren tilldelas ett poäng. Efter en vis tid återuppstår den träffade spelaren och placeras på en till synes slumpvald plats. För att hålla konsistens kan denna punkt inte slumpas fram, då de olika datorerna skulle placera skeppet på olika ställen på planen. Istället baserar sig den nya positionen på spelarnas poäng.

4.2 Genomförda mätningar

Mätningarna är gjorda på två datorer med olika prestanda. För att alla spelare skulle få lika villkor har alla de mänskliga spelarna använt samma dator. I mätningarna har sex mänskliga spelare använts som testpersoner. För att lösa problemet med att spelaren lär sig spelet och därmed presterar bättre för var testfall denna avverkar delades spelarna in i tre grupper. Grupp 1 spelade fallen i ordning ett, två, tre. Grupp 2 spelade fallen i ordning två, tre, följt av ett. Grupp 3 spelade fallen i ordning tre, ett, två. De spelade varje testfall i 60 sekunder. Innan testen spelar de 30 sekunder för träning och anpassning till förhållandet inför varje kommande testfall.

4.2.1 Testfall

De tre testfall som använts är följande:

Testfall 1: Ingen nätverksfördröjning, inget jitter, ingen lokal fördöjning, ingen timewarp. Handlingarna utförs omedelbart hos båda parterna. Detta är idealfallet och på så sätt även ett referensfall.

Testfall 2: Nätverksfördröjning på 60ms, jitter som slumpmässigt adderar mellan 0% och 20% av nätverksfördröjningen varje uppdatering. Eftersom jittret beräknas slumpmässigt kan rent teoretiskt olika testpersoner uppleva olika förhållanden på nätverket. Då jittret beräknas 3000 gånger, 50 gånger per sekund i 60 sekunder, förväntas en genomsnittlig likvärdig total fördröjning. För att efterlikna förändringen av nätverksfördröjningen som sker när meddelanden tvingas ta en ny väg mellan parter så har mellan sekund 10 och 30 samt mellan sekund 40 och 60 ytterligare 20% respektive 30% av den grundläggande nätverksfördröjningen adderats. En statisk lokal fördröjning på 70 millisekunder användes, enligt de riktlinjer som Mauve et al. (2004) presenterar.

Testfall 3: Precis som testfall 2 med tillägget att en dynamisk lokal fördröjning har använts. Tidsfönstret för den dynamiska lokala fördröjningen har varit 500 millisekunder och tröskelvärdet har varit satt till 10% med ett förändringssteg på 100ms.

4.2.2 Mätvärden

För samtliga fall har spelarpoäng mäts och presenteras i två former. Den ena representerar differensen mellan spelarens poäng och den datorstyrda spelarens poäng. Den andra är totalpoängen, summan av de båda spelarnas poäng. Den förstnämnda ämnar att visa hur bra spelaren presterar i relation till datorspelaren under de olika förhållandena som råder i de olika testfallen. Den andra ämnar att visa

en övergripande bild av spelkvaliteten. Var spelet ospelbart resulterade det i en låg totalpoäng, och en relativ hög totalpoäng vid god spelbarhet.

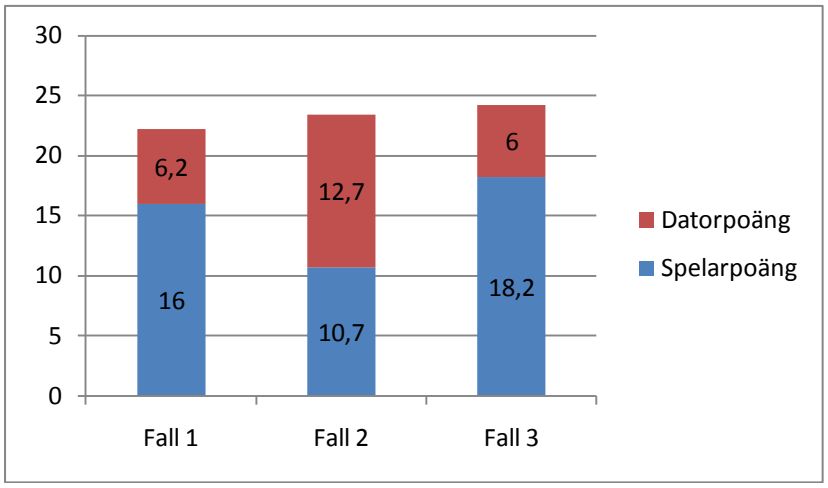
För att styrka främst totalpoängen som ett mått på spelkvaliteten har även ”Mean Opinion Score”-mäts. En skala där spelaren efter varje testfall har betygsatt spelupplevelsen med avseende på nätverk och fördröjning. Skalan sträcker sig från 1 till 5, där 1 står för ett ospelbart förhållande och 5 för ett perfekt förhållande. För utförligare beskrivning se kapitel 3.1.

Utöver det har även värden för antalet temporära inkonsistenser samt tid dessa varade mätts för fall två och tre. Då det i det första fallet inte existerar någon fördröjning över nätverket uppstår det heller inga inkonsistenser och därför mäts de heller inte.

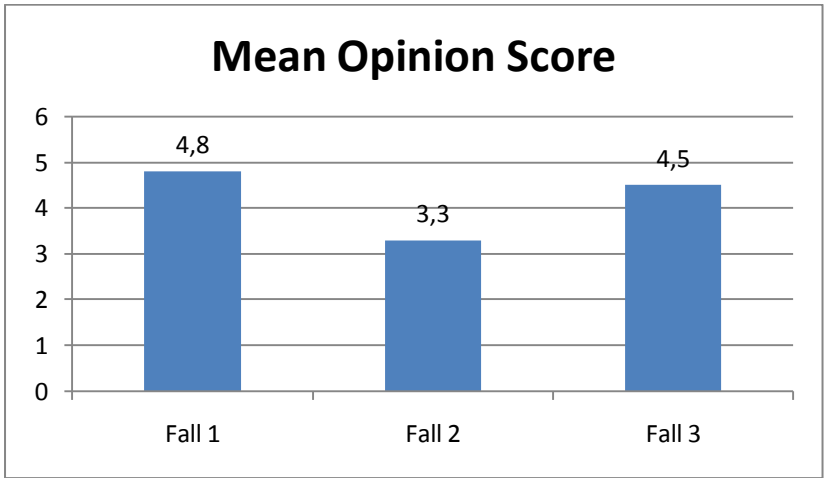
Det kan vid första anblick te sig konstigt att tiden för inkonsistenser kan ligga på 138 sekunder då en spelsession endast varar i 60 sekunder. Det bör förtydligas att det inte är ett mått på under hur många sekunder speltillståndet har varit inkonsistent utan det är summan av hur länge alla inkonsistenser varade.

4.2.3 Resultat

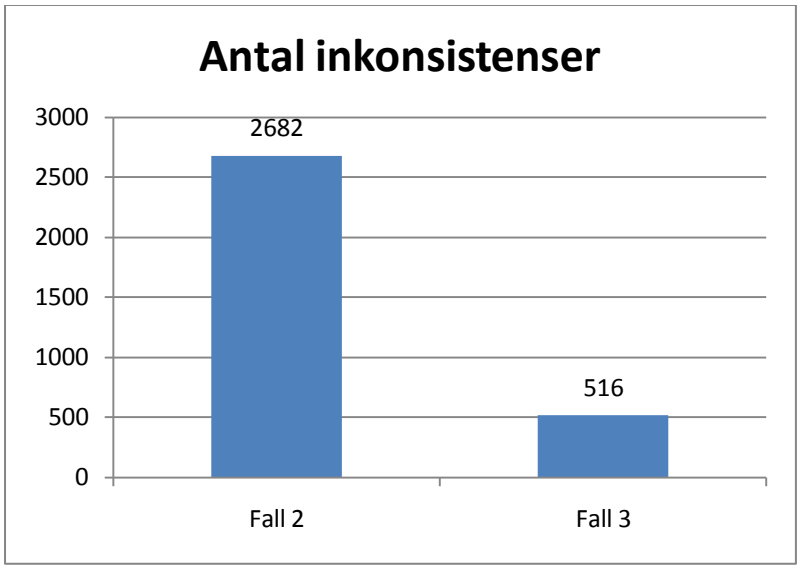
Nedan presenteras resultat från mätningarna.



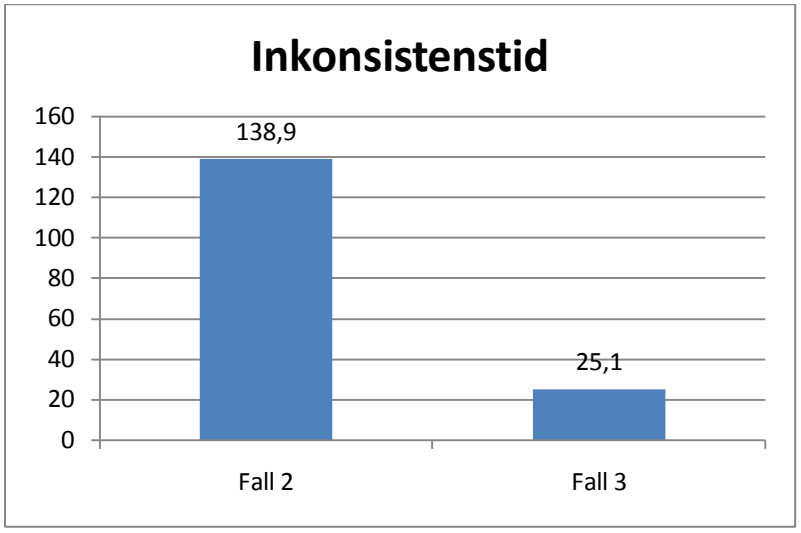
Figur 1 Spelarresultat, visar ett genomsnitt av poängen som uppmäts när de sex testpersonerna har spelat de tre olika testfallen.



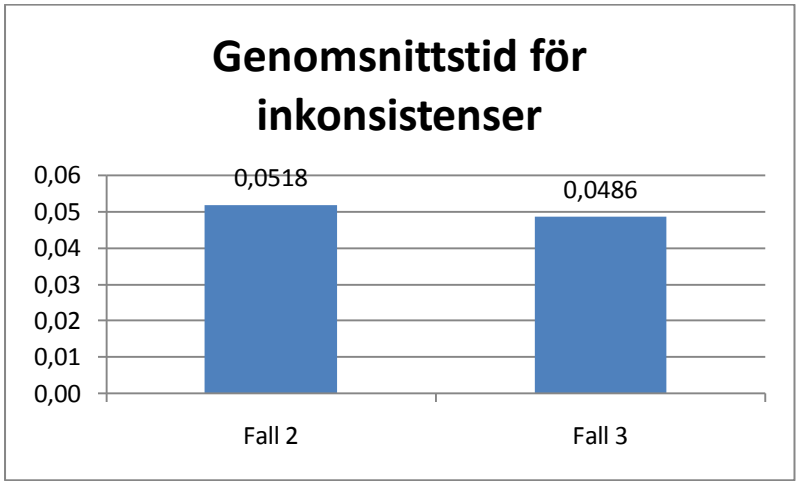
Figur 2 Mean Opinion Score, medelvärdet av de betyg som testpersonerna har givit de olika testfallen.



Figur 3 Antalet inkonsistenser visar ett medelvärde av antalet inkonsistenser som uppkom under testfall två och tre. Då det inte förekommer någon nätverksfördröjning i fall ett sker det heller inga inkonsistenser.



Figur 4 visar genomsnittet av summan av alla inkonsistenser som uppkom i testfall två och tre.



Figur 5 visar genomsnittstiden för de inkonsistenser som uppstod i fall två och fall tre.

4.3 Analys av mätningar

Som kan ses i Figur 1 är totalpoängen i alla fallen väldigt lika. Det skiljer endast ett poäng mellan fall ett och fall två och två poäng mellan fall ett och fall tre. Detta tyder på ett spelbart förhållande i alla fallen.

Som kan ses i Figur 1, skiljer sig den mänsklige spelarens resultat relativt till datorspelarens resultat i de olika fallen. Detta tyder på att någon av parterna gynnas mer än den andra vid de olika förhållandena.

Betraktas Figur 1 och Figur 2 blir det tydligt att det är en större skillnad mellan fall tre och fall två än mellan fall tre och fall ett. Detta tyder på att den dynamiska lokala fördröjningen presterar nästan lika bra som idealfallet och presterar mycket bättre än den statiska lokala fördröjningen.

Den mänskliga spelaren presterar bäst i fall tre och ett, där det endast skiljer 2,2 poäng. Likaså presterar datorspelaren likvärdigt i fall ett och fall tre, det skiljer endast 0,2 poäng. Detta resultat kan däremot inte ses i fall två. Där datorspelaren inte bara presterar bättre än den mänskliga spelaren utan även väldigt mycket bättre än i de andra två fallen. Det skiljer hela 6,5 och 6,2 poäng mellan fall två och fall ett respektive fall två och fall tre för den datorstyrda spelaren.

Tyder detta på att den mänskliga spelaren presterar bättre i fall ett och fall tre än fall två? Eller tyder det på att datorspelaren presterar bättre i fall två än fall ett och tre? Betraktas Figur 2 kan det betyg som de mänskliga spelarna har givit de olika fallen ses. Det bästa, nästan perfekta, fallet är som väntat fall ett. På en klar andra plats finner vi fall 3. Hela 1,2 poäng efter hittar vi fall 2. Den mänsklige spelaren uppfattar förhållandena sämre i det fall som denna presterar sämst i, det vill säga fall två. Detta tyder på att det är den mänskliga spelaren som presterar sämre i fall två i förhållande till de andra fallen och inte att datorspelaren presterar bättre i fall två.

Denna bild blir ännu tydligare om när värdena i Figur 3, Figur 4 samt Figur 5. betraktas. Det sker mycket fler inkonsistenser i fall två än fall tre enligt Figur 3. Som kan beskådas i Figur 4 är den genomsnittliga summan av alla inkonsistenser längre i fall två än fall tre. Figur 5 visar i sin tur att en inkonsistens vara bara lite längre i fall två än fall tre. Dessa tre figurer visar ytterligare på att det råder ett bättre förhållande för en spelare i fall tre än fall två.

Att jämföra fall två och fall tre rakt av är inte korrekt. En viktig parameter som inte är med i mätningarna är värdet för den lokala fördröjningen. I fall 2 är det statiskt och satt till 70 millisekunder, precis enligt de principer som Mauve et al. (2004) ger. I fall tre varierar värdet. För att fallen skall bli jämförbara borde genomsnittsvärdet av den lokala fördröjningen i fall 3 också vara 70 millisekunder. Som kan ses i Figur 3, Figur 4 och Figur 5 sker det inte lika många inkonsistenser, de varar inte lika länge varken summerade eller genomsnittet. Detta tyder på att den lokala fördöljningen för fall 3 har haft ett högre genomsnittsvärde än för fall två. Detta kan även ses när koden för den dynamiska lokala fördröjningen granskas. Värdet för den lokala fördröjningen går inte under det värde valt enligt Mauve et als. (2004) principer. Frågan är vad som jämförs mest? Jämförs en statiskt lokal fördröjning mot en dynamisk lokal fördröjning? Eller jämförs endast två olika värden av lokal fördröjning? Visar mätningarna bara att en spelare föredrar en lite högre lokal fördröjning över en stor mängd inkonsistenser? En sak är klar; denna implementering av dynamisk lokal fördröjning ger färre inkonsistenser och lägre inkonsistenstid än en statisk lokal fördröjning, vilket enligt Figur 2 mänskliga spelare föredrar. Den visar också att

spelaren har inga betydande problem med att anpassa sig till en snabbt förändrande lokal fördröjning.

5 Slutsatser

Detta sista kapitel börjar med en resultatsammanfattning där mätningarna sätts i relation till de mål som sattes upp i kapitel 3. Det följs av en diskussion av metod, argumentation för dynamisk lokal fördröjning och ett resonemang kring dynamisk lokal fördröjning i ett större sammanhang. Kapitlet avslutas med förslag på framtida arbeten.

5.1 Resultatsammanfattning

Arbetets mål, som kan ses i kapitel 3, var att skapa en teknik som förse en spelare i nätverk med en tillräckligt bra miljö för att anse att spelet är spelbart. Denna teknik är dynamisk lokal fördröjning tillsammans med timewarp. Denna lokala fördröjning måste vara så låg att spelaren inte finner spelet oresponsivt samtidigt som den förser spelaren med ett tillräckligt konsistent och korrekt speltillstånd. Att undersöka vad den dynamiska lokala fördröjningen skall baseras på och i vilken utsträckning är även det ett mål. Detta gör uppgiften att skapa ett testspel, en simuleringsmiljö och en datorstyrd spelare till delmål. Nedan följer en sammanfattning av målen:

1. Utveckla dynamisk lokal fördröjning och timewarp.
2. Undersöka vad den dynamiska lokala fördröjningen bör baseras på och i vilken utsträckning.
3. Undersöka hur dynamisk lokal fördröjning tillsammans med timewarp påverkar spelaren.

Enligt mål 1 har en dynamisk lokal fördröjning skapats. Den förändrar den lokala fördröjningen baserat på antalet inkonsistenser som skett hos motparten under ett kort rullande tidsfönster. Har det skett för många inkonsistenser inom tidsfönstret höjs den lokala fördröjningen. Har inga inkonsistenser skett inom tidsfönstret sänks den lokala fördröjningen. Se kapitel 4.1.3 samt kapitel 4.1.6 för utförlig information om implementering.

Mål 2 är ett mål som inte nåddes av arbetet. Ingen undersökning gjordes för att hitta vad en bra dynamisk lokal fördröjning baserar sig på. För undersökningarna i mål 3 valdes godtyckliga värden. Detta faktum underminerar inte resultaten av mål 3. Snarare tvärtom då de visar hur bra bara godtyckligt valda värden presterar relativt en statisk lokal fördröjning.

Mål 3 är det sista och det viktigaste målet. En undersökning baserat på tre fall testade av sex testpersoner utfördes. Resultaten visar att spelaren presterade bättre i en miljö med en genomsnittlig nätverksfördröjning på 60 millisekunder och dynamisk lokal fördröjning än i en miljö med samma nätverksfördröjning och en statisk lokal fördröjning satt till 70ms. Detta baserat på de poäng spelaren uppnådde och det styrks också av det betyg spelaren satte på miljön. Ser man till antalet inkonsistenser som skedde i de olika fallen blir bilden tydligare. Jämför man den dynamiska lokala fördröjningen med en miljö utan varken någon nätverksfördröjning eller lokal fördröjning ser man att det är en mindre skillnad i poäng samt betyg än mellan dynamisk lokal fördröjning och statisk lokal fördröjning. Mätvärdena tyder dock på att genomsnittsvärdet för den dynamiska lokala fördröjningen är högre än den för den statiska. Som en oförväntad biprodukt av testerna syns det att spelare föredrar ett lågt antal inkonsistenser över en låg lokal fördröjning samt att det inte är några problem med att anpassa sig till en snabbt förändrande lokal fördröjning. För mer information se kapitel 4.2 samt 4.3.

Utöver de uppnådda målen har arbetet även bidrag med en testplattform för nätverksspel och olika tekniker. Testplattformen består av tre stora delar:

- Ett enkelt spel där två rymdskepp åker runt i en tvådimensionell värld med avsikt att skjuta varandra. Skrivet i C# med hjälp av XNA 3.1 Se kapitel 4.1.2 för mer information.
- Ett litet nätverk som sammankopplade två datorer. Tack vare XNA 3.1 blev det väldigt enkelt att simulera nätverksfördröjning och jitter. Se kapitel 4.1.5 för mer information.
- En datorstyrd spelare som agerar efter ett par enkla principer. Den styr skeppet mot sin motståndare tills motståndaren befinner sig inom räckvidd för det datorstyrda skeppets laser, då skjuter det upprepade gånger tills det träffar. För ytterligare information se kapitel 4.1.4.

5.2 Diskussion

Det finns ett par punkter i detta arbete som kan förbättras. Främst att de fall som jämförs i undersökningen inte har samma genomsnittliga lokala fördröjning. Vilket väcker frågan om det är konceptet om en dynamisk lokal fördröjning som testas eller endas två olika värden av lokal fördröjning jämförs? Den andra punkten är om den statiska fördröjningen verkligen är satt enligt Mauve (2004) et als. principer. Jittret finns inte med i den beräkningen. Den statiska lokala fördröjningen är troligtvis för låg vilket ger upphov till fler inkonsistenser än om den var satt korret. Den statiska lokala fördröjningen borde ligga på 80 snarare än 70ms. Skulle denna skillnad på 10 millisekunder förändra resultatet? Med största sannolikhet skulle den inte ge en betydande skillnad. Som det är nu sker det mer än fem gånger så många inkonsistenser med den statiska lokala fördröjningen än den dynamiska lokala fördröjningen. Även om det bara skulle ske två gånger så många inkonsistenser så kan samma slutsats av arbetet dras.

Testerna tyder på att en dynamisk lokal fördröjning är att föredra över en statisk lokal fördröjning. Detta då ett konsistent speltillstånd kan erhållas i större utsträckning även om det blir stora förändringar in nätverksfördröjningen. Självklart presterar inte implementeringen av dynamisk lokal fördröjning som är gjord i det här arbetet bättre än en statisk lokal fördröjning vid alla förhållanden. Skulle nätverksfördröjningen nå extremt höga värden som till exempel 500 millisekunder så skulle även den lokala fördröjningen höjas till cirka 500ms. Som litteraturen visar är 500 millisekunder lokal fördröjning inte ett acceptabelt värde för spel av den typen som är skapad i detta arbete. Det kan krävas att man sätter en maxgräns baserad på olika typer av spel.

Det kan tänkas att en dynamisk lokal fördröjning inte är att rekommendera för spel som kräver extrem tillförlitliga responstider som till exempel fightingspel. Där är det inte alltid viktigast att vara först, till skillnad från spelet som har skapats i detta arbete. Det kan röra sig om att göra slaget vid precis rätt tillfälle. Slaget skall inte träffa motståndaren när denna fortfarande har sin guard uppe utan först när guarden har sänkts.

På förhand var det lätt att anta att det största problemet med dynamisk lokal fördröjningen skulle ligga i spelarens oförmåga att anpassa sig till de snabba förändringarna av den lokala fördröjningen. Det finns dock inget i testerna som tyder på att detta har varit ett problem. Även om så faktiskt kan vara fallet så verkar det

vara viktigare för spelaren att det sker få inkonsistenser än att den lokala fördröjningen är statisk.

5.3 Framtida arbete

Ser man till ett kortsiktigt perspektiv är steg ett vid eventuellt vidareutveckling att undersöka vilka värden som den implementerade algoritmen för dynamisk lokal fördröjning presterar optimalt på. Steget efter det är att utarbeta en bättre implementering av dynamisk lokal fördröjning. Detta arbete visar att en enkel algoritm med godtycklig valda variabler presterar bra. Detta faktum gör det intressant att se vad en genomtänkt algoritm kan åstadkomma. Går det att se mönster eller att förutspå ökningar och sänkningar av nätverksfördröjningen? Bör en ökning av den lokala fördröjningen ske lika snabbt som en sänkning? Finns det någon annat att basera algoritmen på än antalet inkonsistenser som sker hos motståndaren? Detta rör tre saker som ett framtida arbete kan undersöka.

För att bättre se hur bra dynamisk lokal fördröjning kan prestera bör en jämförelse mellan dynamisk lokal fördröjning och statisk lokal fördröjning där båda har samma medelvärde för den lokala fördröjningen utföras.

Ett framtida arbete bör också jämföra dynamisk lokal fördröjning med ett fall som har samma fördröjning på nätverket men utan någon teknik för kompensering, det vill säga handlingar utförs när de kommer fram. Efter det kan tekniken jämföras med andra tekniker, så som client-side prediction eller liknande.

I ett långsiktigt perspektiv bör det undersökas om dynamisk lokal fördröjning fungerar bra för fler än två spelare åt gången. Vad skall en sådan algoritm basera sig på? Ett medelvärde av antalet inkonsistenser som finns hos alla parter? Eller på det högsta antalet inkonsistenser som någon part har?

Det kan också vara intressant att undersöka vilken typ av spel som gynnas av dynamisk lokal fördröjning. Det finns säkert vissa typer av spel som gynnas mer än andra. Detta kan bero på vad den högsta gränsen av lokal fördröjning som gäller för just den typen av spel. Kan det även finnas andra faktorer?

I vilken utsträckning spelare klarar att anpassa sig till en skiftande dynamisk lokal fördröjning är också det en sak som bör undersökas. Finns det olika typer av spel som gynnas alternativt lider av en snabbt varierande lokal fördröjning.

Referenser

Armitage, G. (2003) *An experimental estimation of latency sensitivity in multiplayer Quake 3*. (sid 137-141) The 11th IEEE International Conference on Networks

Armitage, G. & Stewart, L. (2004) *Limitations of using real-world, public servers to estimate jitter tolerance of first person shooter games*. (sid 257-262) Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology.

Baughman, N. & Levine, B. N. (2001) *Cheat-proof playout for centralized and distributed online games.* (sid 104-113) In Proceedings IEEE InfoCom 2001.

Beigbeder, T., Coughlan, R., Lusher, C., Plunkett, J., Agu, E., & Claypool, M. (2004) *The effects of loss and latency on user performance in unreal tournament 2003*. (sid. 144-151) Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games.

Brun, J., Safaei, F. & Boustead, P. (2006) *Managing latency and fairness in networked games.* (sid. 46-51) Communications of the ACM Volym 49 nr 11.

Chen, K., Huang, P. & Lei, C. (2006) *How sensitive are online gamers to network quality?* (sid. 34-38) Communications of the ACM Volym 49 nr 11.

Claypool, M. & Claypool K. (2006) Latency *and player actions in online games.*

(sid. 40-45) Communications of the ACM Volym 49 nr 11.

Coulouris, G., Dollimore, J. & Kindberg, T. (2005) *Distributed Systems: Concepts and Design* (sid. 2-3, 34-38,49, 70), Pearson Education Limited.

Dick, M., Wellnitz, O., & Wolf, L. (2005) *Analysis of factors affecting players' performance and perception in multiplayer games*. (sid. 1-7) Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games.

Liang, D. & Boustead, P. (2006) *Using Local Lag and Timewarp to Improve Performance for Real Life Multi-player Online Games.* Artikel 37. Proceedings of 5th ACM SIGCOMM workshop on Network and system support for games.

Mauve, M. (2000) *Consistency in replicated continuous interactive media*. (sid. 181-190) Proceedings of the 2000 ACM conference on Computer supported cooperative work.

Mauve, M., Vogel, J., Hilt, V. & Effelsberg, W. (2004) *Local-lag and Timewarp Providing Consistency for Replicated Continuous Applications*. (sid. 47-57). IEEE Transactions on Multimedia, 6,

Pantel, L & Wolf, L. C. (2004) *On the suitability of dead reckoning schemes for games.* (sid. 79-54). Proceedings of the 1st workshop on Network and system support for games.

Sheldon, N., Girard, E., Borg, S., Claypool, M. & Agu, E. (2003) *The effect of latency on user performance in Warcraft III*. (sid 3-14) Proceedings of the 2nd workshop on Network and system support for games.

Smed, J., Kaukoranta, T. & Hakonen, H (2001) *Aspects of networking in multiplayer computer games*. (sid 74–81) Proceedings of International Conference on Application and Development of Computer Games in the 21st Century.

Stuckel, D. & Gutwin, C. (2008) *The effects of local lag on tightly-coupled interaction in distributed groupware*. (sid. 447-456) Proceedings of the 2008 ACM conference on Computer supported cooperative work.

Tanenbaum, S. (2003). *Computer Networks*. (sid. 81) Pearson Education Limited.

Quax, P., Monsieurs, P., Wim, L., De Vleeschauwer, D., & Degrande, N. (2004) *Objective and subjective evaluation of the influence of small amounts of delay and jitter on a recent first person shooter game*. (sid 152-156) Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for games.