Examensarbete

HÖGSKOLAN I SKÖVDE

FUSKFÖREBYGGANDE BANDBREDDSOPTIMERING I REALTIDSSTRATEGISPEL

CHEAT PREVENTING BANDWIDTH OPTIMIZAZTION IN REAL-TIME STRATEGY GAMES

Examensarbete inom huvudområdet Datalogi
Grundnivå 30 högskolepoäng
Vårtermin 2016

Patrik Johansson

Handledare: Sanny Syberfeldt
Examinator: Henrik Gustavsson

Sammanfattning

Denna studie fokuserar på att optimera bandbredden i ett nätverksprotokoll som förhindrar fusk inom realtidstrategispel (RTS). Metoden använder sig utav speltillståndsuppdateringar och studien undersöker om kommandoöverföring är mindre bandbreddskrävande än speltillståndsuppdateringar. För att behålla fuskskyddet i protokollet måste kommandon uppdelas för att förhindra klienterna från att få för mycket information.

Experimentet testade dessa två metoder för att se vilken utav dem som är minst bandbreddskrävande. Kommandoöverföring visar väldigt bra resultat, dock fanns det tillfällen som upptäcktes där speltillståndsuppdatering var effektivare. Det som orsakade att kommandoöverföring var mer bandbreddskrävande vid det tillfälle var att kommandoöverföringen skickade flera meddelande vid samma uppdatering samtidigt när speltillståduppdateringen skickades med ett meddelande.

Denna studie gjorde experiment på en enkel version utav ett RTS och det finns många faktorer i ett kommersiellt RTS som kan påverka mätningen. Det skulle vara intressant att utföra denna studie i ett kommersiellt RTS.

Nyckelord: Fuskhantering, Nätverk, Bandbreddsoptimering, RTS

Innehållsförteckning

1 Introduktion ... 1

2 Bakgrund ... 2

2.1 Nätverksarkitekturer och nätverksprotokoll ... 2

2.2 RTS och Fuskhantering ... 2

2.3 Tidigare arbeten ... 3

3 Problemformulering ... 4

3.1 Problembeskrivning ... 4

3.2 Metodbeskrivning ... 5

3.3 Metoddiskussion ... 7

4 Implementation ... 8

4.1 Spelet ... 8

4.2 Nätverk ... 9

4.2.1 Typer av meddelanden ... 9

4.2.2 Överföring av kommandoinput ... 10

4.2.3 Överföring av speltillstånd ... 12

4.3 Förebyggande av fusk ... 12

4.4 Simulering ... 13

5 Utvärdering ... 17

5.1 Presentation av undersökning ... 17

5.1.1 Scenario 1: Enkel rörelse förbi fiendeenhet ... 18

5.1.2 Scenario 2: Diagonal rörelse i en trång miljö förbi en fiendeenhet ... 19

5.1.3 Scenario 3: Snabb rörelse in och ut ur fiendens synfält ... 20

5.1.4 Scenario 4: Slumpmässiga rörelser ... 20

5.2 Analys ... 21

5.2.1 Scenario 1: Enkel rörelse förbi fiendeenhet ... 21

5.2.2 Scenario 2: Diagonal rörelse i en trång miljö förbi en fiendeenhet ... 21

5.2.3 Scenario 3: Snabb rörelse in och ut ur fiendens synfält ... 21

5.2.4 Scenario 4: Slumpmässiga rörelser ... 21

5.3 Slutsatser ... 22

6 Avslutande diskussion ... 23

6.1 Sammanfattning ... 23

6.2 Diskussion ... 23

6.3 Framtida arbete ... 24

Referenser ... 26

1 Introduktion

Realtidsstrategispel (RTS) är en stor spelgenre som oftast använder Peer-to-Peer (P2P) som nätverksarkitektur för spel över internet. Problemet med P2P är att det finns möjlighet för spelarna att använda sig av fuskprogram för att få ett större övertag. Lösningen är att använda sig av klient/server (KS) för att undvika fuskhantering, men problemet med KS är att det kan bli väldigt långsamt. ORTS är en KS lösning som Buro (2002) använder, men problemet är att den skickar speltillståndsuppdateringar som leder till hög bandbredd. (Terrano & Bettner, 2001) skriver att i *Age of Empires* (Ensemble Studios, 1997) används P2P-arkitekturen och kommandouppdateringar istället för speltillståndupdateringar. De ville ha episka krig med många enheter och de menar att det inte skulle kunna gå med speltillståndupdatering då det är för mycket data som behöver skickas över nätverket vid varje uppdatering.

Detta arbete fokuserar på att optimera bandbredden i ORTS metoden genom att använda kommandouppdateringar istället för speltillståndsuppdateringar för att förminska bandbredden. Ett krav är även att fuskskyddet som ORTS har fortfarande ska vara kvar. Att använda sig av hela kommandouppdateringar som *Age of Empires (*Ensemble Studios, 1997) gör, skulle leda till att klienterna får för mycket information om de andra spelarna.

Arbetet mäter bandbreddsskillnaden mellan kommandouppdateringar och speltillståndupdateringar. För att implementera Interest Management (Smed, Kaukoranta & Hakonen, 2002) skalades kommandon ner för att undvika att klienterna skulle få information som de inte var behörighet till, något som kunde leda till högre bandbreddsanvändning än vanlig kommandouppdatering.

Experimentet utförs i ett RTS-spel med olika scenarion för att se hur mycket bandbreddskrävande det är mellan metoderna vid specifika scenarion. Det visade sig att vissa scenarion var det ena metoden effektivare och vissa scenarion var det ombytta roller. Kommandouppdateringar var effektivare vid få händelser och speltillståndsuppdateringar var effektivare vid flera händelser. Det var en faktor i kommandoöverföringen vilket gjorde att det blev väldigt ineffektivt jämfört med speltillståndsuppdateringar, vilket kunde ha lett till kommandoöverföringen kunde varit effektivare.

2 Bakgrund

Datorspel är en väldigt stor industri och spel där spelare tävlar mot varandra är väldigt populära. I sådana spel finns det spelare som fuskar för att få ett övertag. Detta är väldigt vanligt och det är något som utveklarna försöker att kämpa mot. När ärliga spelare möter spelare som fuskar blir det orättvist för dem och de kan tappa lusten att spela.

2.1 Nätverksarkitekturer och nätverksprotokoll.

De flesta multiplayerspel spelas över ett nätverk och de två vanligaste arkitekturerna för nätverkskommunikation är Peer-to-Peer (P2P) och Klient/Server (KS). P2P är en uppsättning bestående av enbart klienter som kommunicerar med varandra och delar på information om speltillståndet. KS har en central server som håller hela speltillståndet och kan dela ut information om speltillståndet till klienterna.

På internet används två protokoll för meddelandeöverföring mellan datorerna; UDP och TCP. (Cerf & Kahn, 1974) är grunderna till TCP. Båda har fördelar och nackdelar som gör att båda protokollen används för olika datorspel beroende på vad spelet kräver. När ett meddelande skickas finns det en risk att meddelandet går förlorat. TCP försäkrar att meddelanden kommer fram till destinationen och att meddelanden kommer i rätt ordning. Ifall ett meddelande försvinner kommer TCP se till att skicka meddelandet igen tills motagaren har bekräftat att meddelandet har kommit fram. UDP garanterar inte leverans av meddelanden. Fördelen med UDP är att det går snabbare att skicka meddelanden för den inte har några kontroller som TCP har. TCP har mer information än vad UDP har vilket leder till att TCP har större grundstorlek än vad UDP har. Ett tomt TCP har 20 bytes jämfört med UDP som har 8 bytes (Diffen 2016).

UDP används i datorspel där uppdateringar av speltillstånd skickas konstant till klienterna. Ifall ett meddelande försvinner är det onödigt att försöka skicka samma meddelande för vid den tiden har speltillståndet uppdaterats och är redo att skicka nästa uppdatering till de andra klienterna. First-person shooter är en genre där det är väldigt viktigt att meddelanden kommer fram snabbt för att spelarna ska kunna reagera snabbt beroende hur speltillståndet ser ut. TCP används i datorspel där det är viktigare att meddelanden kommer fram än att de kommer snabbt. Turbaserade spel är ett exempel på en genre där det är mer lämpligt att använda TCP än UDP.

2.2 RTS och Fuskhantering

Realtidsstrategispel (RTS) är en populär genre inom multiplayerspel. RTS går ut på att bygga upp en bas och producera stridsenheter för att anfalla och försvara sig mot sina motståndare. Målet är ofta att förinta alla fienders byggnader och inte förlora sina egna byggnader. Tre kända RTS spel är *Age of Empires* (Ensemble Studios, 1997), *Warcraft: Orcs & Humans (*Blizzard Entertainment, 1994) och *StarCraft* (Blizzard Entertainment, 1998) som även fått uppföljare som använder sig av spelmekaniken Fog of War. Fog of War är att spelaren bara ser omgivningen där spelaren har sina enheter. Maphack är ett fusk som tar bort Fog of War och visar vart fiendeenheter befinner sig, något som ger ett stort övertag för den fuskande spelaren. RTS har ofta stridsenheter som är bra mot vissa enheter och dåliga mot andra, enligt sten, sax, påse principen. Om fuskaren vet vad motståndaren producerar för enheter kan han kontra med att bygga de enheter som är bra mot motståndarens enheter.

Interest Management är en teknik som Smed, Kaukoranta & Hakonen (2002) beskriver som gör att klienterna bara får information de har rätt till. Servern/klienten räknar ut vart och hur långt bort objekt befinner sig från spelarens egna objekt och ifall objekten är långt bort behöver spelaren inte information om dem. Detta skapar extra arbete för att räkna ut vilka objekt som är relevanta för klienterna men är väldigt effektivt för att minska bandbredden. Detta är även nyckeln till att förhindra att klienter får onödig information om speltillståndet, vilket neutraliserar maphack då det inte går att komma åt mer information än vad som visas för spelaren.

2.3 Tidigare arbeten

Bursztein, Hamburg, Lagarenne & Boneh (2011) skapar ett maphack och förklarar hur processen går till för att få fram information om hela speltillståndet. Deras maphack söker i spelets minne för att hitta den uppbyggda strukturen för speltillståndet. När den hittar strukturen visar den upp speltillståndet på en bild som är en karta med olika färger som representerar olika enheter på kartan. Deras arbete syftar till att försöka kryptera minnet för att förhindra program att komma åt minnet och skapa en bild av speltillståndet. Problemet är dock att datan fortfarande finns kvar på klienten. Även om den är skyddad, kommer det alltid finnas en möjlighet att hitta den ändå.

Chambers, Feng, Feng & Saha (2005) försöker förhindra maphack genom att varje klient skickar data om deras vy de ser till de andra klienterna. Med den vyn kan klienten beräkna vilka enheter som befinner sig inom den vyn och kan skicka bara information som den klienten ser. Detta löser inte exakt problemet maphack då varje klient kan se vad andra klienter har för vy, vilket leder till att man kan få en bild var de befinner sig. Ett annat problem som uppstår är att när en ensam klient simulerar världen kan den fuska genom att simulera på fel sätt, till exempel genom att skapa flera enheter. Ett problem som de löser är att klienterna kan ljuga om hur stor area de ser med en metod som kontrollerar efter matchen.

Buro (2002) skapar metoden ORTS som är en lösning som förhindrar maphack och andra fusk till RTS-spel. Metoden använder sig av KS arkitekturen. Klienterna skickar kommandon till servern, som servern behöver för att simulera speltillståndet. När servern har uppdaterat speltillståndet skickar den ut alla nya positioner och händelser för varje enhet till de behöriga klienterna. RTS-spel har väldigt många enheter och när det blir väldigt många enheter klarar inte servern av att skicka ut alla enheter inom den tiden som är kravet, vilket leder till fördröjningar inom spelet.

3 Problemformulering

RTS-spel spelas oftast över internät. På Internet finns det många oärliga personer som använder sig av fusk för att få ett bättre övertag. Maphack är en typ av fusk som används i RTS genren som gör att spelaren ser vilka enheter fiender har och vart de befinner sig. Maphack är ett program som läser i minnet för att hitta information om fienderna.

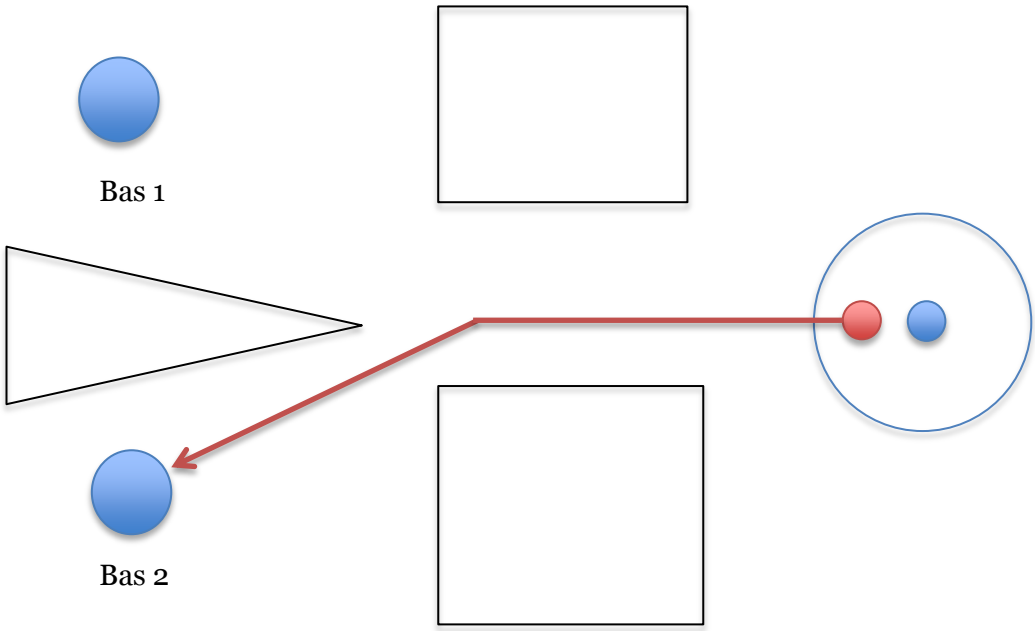
När spel använder en P2P-arkitektur kommer det alltid finnas möjlighet att kunna fuska. För att förhindra maphack i P2P får inte klienterna ha information om varandra, vilket kan leda till att klienten till exempel kan fördubbla sina enheter utan någon annan upptäcker det. Den ända lösningen för att hantera fusk i RTS är att köra spelet i klient/server arkitektur. Genom att ha en nod där simuleringen sker, som ingen av spelarna kontrollerar, kan ingen av klienterna ändra speltillståndet, och klienterna tar bara emot information som är relevant för dem.

3.1 Problembeskrivning

Metoden ORTS (Buro, 2002) en lösning för att förhindra fusk. Problemet med den är att den är väldigt bandbreddkrävande. När enhet *A* har fått kommandot att gå till position *X*, gör servern uppdateringarna och skickar nya speltillstånd till klienterna för varje uppdatering. Detta betyder att servern måste skicka meddelanden till klienterna för varje gång enhet *A* rör på sig som är väldigt bandbreddkrävande. Detta är ett problem när det är många enheter som rör sig samtidigt, då det leder till att nätverket blir belastat och inte hinner skicka all information.

Age of Empires (Ensemble Studios, 1997) använder P2P arkitekturen och använder sig av kommandon som varje klient skickar till varandra som (Terrano & Bettner, 2001) nämner. Med varje kommando som varje klient tar emot kan de simulera sitt eget speltillstånd. Detta sparar mycket bandbredd där ingen av klienterna behöver skicka hela speltillståndet för varje uppdatering. Problemet med *Age of Empires* (Ensemble Studios, 1997) är att varje klient simulerar varje enhet som befinner sig under Fog of War, vilket leder till att maphack kan komma åt information om enheterna.

Detta arbete fokuserar på att skapa en metod för att reducera bandbreddsanvändningen i ORTS. Metoden inspireras av *Age of Empires* (Ensemble Studios, 1997) lösning och använder kommandoöverföringar istället för speltillståndsuppdateringar. Lösningen är att använda sig av kommandoöverföringar över en klient/server arkitektur där servern simulerar hela speltillståndet och klienterna simulerar den delen av speltillståndet som är relevant för dem för att minska bandbreddsanvändningen utan att påverka fuskskyddet som ORTS har. Problemet med att skicka hela kommandon till klienterna är att spelarna kan få för mycket information av kommandot, se figur 1. Lösningen behöver ändras så att servern delar upp kommandon och skickar de delar som är relevanta för klienterna. När klienten har simulerat klart första delen av en enhets kommando måste klienten begära nästa del och detta skapar mer bandbredd men skyddar ifrån maphack. Hypotesen är att skicka uppdelade kommandon till klienterna är mindre bandbreddskrävande än att skicka över speltillståndet.



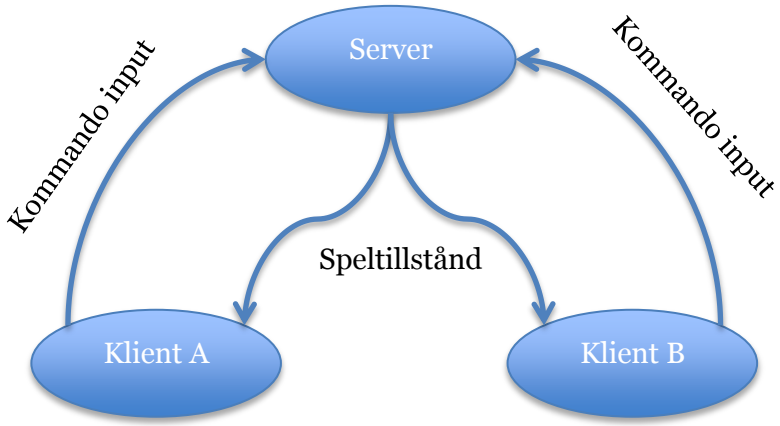
Figur 1 En röd enhet befinner sig inom den blåa spelarens synfält. Om enhetens kommando inte är uppdelat kommer den blåa klienten simulera hela händelsen. Maphack kommer då kunna visa vart den röda enheten är på väg och den blåa spelaren kan då förberedda sig inför ett anfall på bas 2.

3.2 Metodbeskrivning

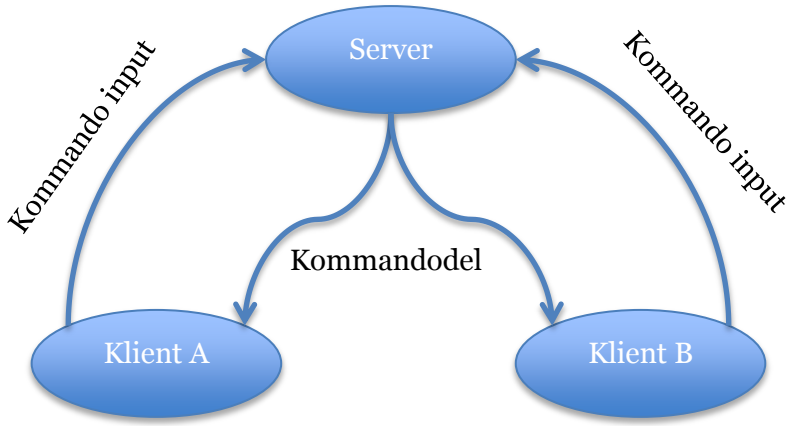
För att pröva hypotesen kommer ett experiment(Wohlin, Ohlsson, Runeson, Regnell, Höst & Wesslén, 2012) utföras i ett RTS-spel där två klienter är kopplade till en server. Spelet kommer att implementeras i en version med ORTS metoden där servern skickar tillstånd och en version där servern skickar kommando och klienterna simulerar spelvärdlen. Genom att ha båda metoderna kommer det vara möjligt att experimentera mellan de två metoderna som är oberoende variabeler som då kommer att påverka beroende variabeln som är mängden data som skickas med den valda metoden och som gör möjligheten att mäta skillnaden mellan metoderna och undersöka om hypotesen är sann.

Experimentet mäter hur många bytes som skickas ifrån servern till klienterna i respektive version för att se hur stor skillnad det är mellan att skicka speltillståndet som metoden ORTS gör och att skicka över enbart kommandon, se figur 1 och 2. För att få samma förutsättningar kommer båda metoderna köras samtidigt för att kunna göra en rättvis mätning.

För att uppnå Interest Management behöver alla kommandon som server skickar att skalas ner till ett stadie som gör att maphack inte kommer åt mer information än vad klienten visar för spelaren. Servern skickar enheternas riktningar vart de är på väg istället för vilken position enheten. När enheten byter riktning eller stannar meddelar servern till klienten att enheten har bytt beteende.



Figur 1 Klienterna skickar kommando input till servern som sedan beräknar speltillståndet och sedan skickar ut information om speltillståndet till klienterna.



Figur 2 Klienterna skickar kommando input till servern och som sedan räknar ut hur mycket av det kommando är relevant för den andra klienten som kommer få uppdelat.

Spelet kommer att använda sig av både UDP och TCP. Kommando input skickas med TCP för att undvika förlorade paket. Servern på ORTS modellen kommer att skicka speltillståndet med UDP för att minska bandbredden samt att det går fortare. Som beskrivs i kapitel *2.1* är förlorade meddelanden inget problem ifall det är speltillstånd som skickas, vilket gör att speltillståndet kommer att skickas med UDP. ORTS modellen kommer då använda sig av TPC och UDP och simuleringsmodellen av bara TCP. Mätningen kommer att ta hänsyn till att de olika basstorlekar på protokollen som kommer att adderas på mätningen.

Experimentet kommer att testa olika scenarion för att se när det skiljer sig mest ifrån båda metoderna.

- Enkel rörelse förbi fiendens enhet. Detta är väldigt grundläggande rörelse, vilket sker väldigt mycket i RTS-spel.

- Diagonal rörelse i en trång miljö förbi en fiendeenhet. Detta är för att testa om uppdelningen av kommandon blir för hög att det leder till högre bandbreddskrävande än vid speltillståndupdateringar.
- Snabb rörelse in och ut ur fiendens synfält ett antal gånger. Detta leder till att servern kommer att behöva skicka ny information varje gång enheterna kommer in i synfältet som kan leda till hög bandbredd.
- Slumpmässiga rörelser. Detta är inte så vanligt förekommande, men kan vara intressant att se skillnaden vid dessa tillfällen.

3.3 Metoddiskussion

Ett problem med den föreslagna metoden är att spelet som implementeras är väldigt förenklat jämfört med ett kommersiellt RTS. Enheternas rörelse blir väldigt stela och kommandon delas upp lika mycket jämfört med ett kommersiellt RTS. Detta påverkar bandbredden när det inte skickas lika många kommandodelar till klienterna.

ORTS använder sig av kollision, dock är det inte relevant för detta problem att implementera det när arbetet handlar om bandbredd. Dock kommer metoden behöva använda sig av en miljö där objekt inte kan vistas sig vid, till exempel berg som inte enheterna kan gå över. Detta är för att få fler uppdelningar av kommandon. Om det var en platt miljö skulle det bara vara raka kommandon som gör att servern inte behöver skicka lika ofta och blir mindre likt ett kommersiellt RTS.

Mätningen kommer att använda sig av tekniken experiment(Wohlin, Ohlsson, Runeson, Regnell, Höst & Wesslén, 2012). Mätningen kommer experimentera på olika scenarier för att se om det blir olika bandbreddskrävande beroende på vilket scenario som utförs. Problemet att använda sig av denna mätteknik är att de scenarier som testas måste vara relevanta för det som ska mätas. De fall som ska utföras i denna studie bör vara sådana fall som efterkommer mycket i ett kommersiellt RTS. En annan teknik skulle vara fallstudie(eng. *Case study*)(Wohlin, Ohlsson, Runeson, Regnell, Höst & Wesslén, 2012). Problemet med fallstudie är att den fokuserar på ett scenario och analyserar vilken metod som skulle vara mest lämpligast att använda för det scenario. I ett RTS förekommer det många olika scenarier och kan inte använda sig av fallstudie för att denna studie behöver titta på flera scenarier. Att använda sig av tekniker som surveyundersökning(Wohlin, Ohlsson, Runeson, Regnell, Höst & Wesslén, 2012) som intervjuar människor och samlar in deras åsikter är inte relevant för denna studie när det är data ifrån två metoder som ska jämföras.

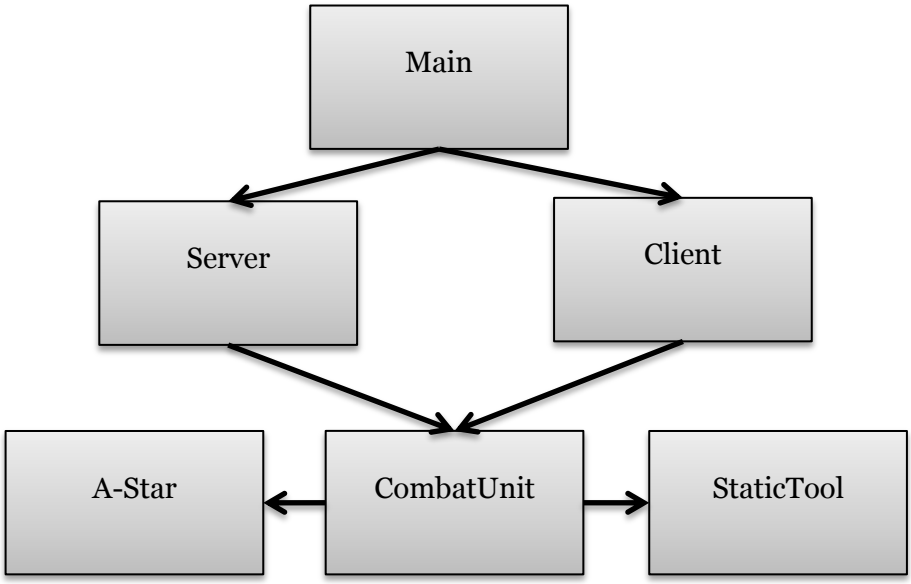
4 Implementation

Detta kapitel beskriver implementationen av spelet för experimentet och de designval som har gjorts. Spelet använder sig av olika typer av meddelanden som skickas mellan klienterna och servern och det förklaras varför de behövs för att kunna göra experimentet. Dessutom beskrivs ett spelscenario med ett exempel på hur en mätning görs.

4.1 Spelet

Spelet utvecklas i programspråket C++ och använder bibloteket Simple and Fast Multimedia Library (SFML Team, 2016). Bibloteket har egenskaper som förenklar utritning av grafik, att skicka nätverkspaket, tidtagarur (klockor), och spela upp ljud. Spelet använder sig av bibloteket främst för grafik och nätverk.

De komponenter som spelet har visas på figur 3. Main startar upp hela programmet med att skapa två instanser av *Client* och en instans *Server*. *Server* och *Client* använder sig utav en gemensam komponent för enheterna för att få samma rörelser. *Server* och *Client* har inga kopplingar mellan varandra, då för data kommuniceras igenom det lokala nätverket och inte programmets allokerade minne.

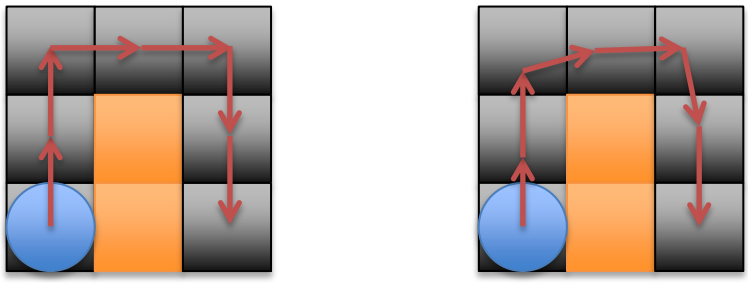


Figur 3 Komponentdiagram

Spelet består av tre fönster där två representerar klienter med var sitt lag, röd och blå, och ett representerar servern. Varje klient visar i sitt fönster alla enheter i sitt lag och varje enhets synfält och varje fiendeenhet som befinner sig inom synfältet. Servern har hela speltillståndet och serverns fönster kommer att visa alla enheter i speltillståndet. Spelet använder inte spelmekaniken ”Fog of War” för att klienterna tar bort motståndarens enheter ur sin version av speltillstånd för att förhindra felaktig information.

Spelet använder sig av vägplaneringalgoritmen A* (Hart, Nilsson & Raphael, 1968) för att styra enheter runt hinder i spelmiljön. A* valdes för den är en av de snabbaste vägplaneringalgoritmerna inom datorspel. Nackdelen med att använda A* är att

vägplaneringalgoritmen är rutnätbaserad vilket gör att rörelserna kommer bli väldigt stela när enheterna rör sig ruta för ruta. En lösning som används i spelet är att när enheten ska gå till nästa ruta är att den inte behöver vara på mitten av rutan för att påbörja nästa destination, se figur 4. Med denna lösning kommer det bli flera olika riktningar som kommer att öka nätverkstrafiken hos kommandoöverföringens metoden. Detta behöver inte vara en nackdel för i ett kommersiellt RTS byter enheterna ofta riktningar vilket gör att resultatet blir likt ifall ett kommersiellt RTS skulle ha experimenterats.



Figur 4 Den blåa enheten tar sig runt det orangea hindret med två olika krav. Den vänstra bilden har enheten kravet att den måste nå mitten innan den påbörjar nästa destination. Den högra bilden har kravet att den måste komma in en fjärde del av rutan innan den påbörjar nästa destination. På vänstra bilden har den tre olika riktningar medan den högra har fem olika riktningar.

4.2 Nätverk

Spelet använder sig av totalt två klienter och en server. Servern använder sig av de två metoderna som ska jämföras, kommandoinput och speltillståndupdatering. Klienterna skickar samma meddelanden till servern oberoende av vilken metod som används. Servern i sin tur skickar olika meddelanden beroende på vilken metod som används. Delkapitel 4.2.1 handlar om de olika typer av meddelanden som används.

4.2.1 Typer av meddelanden

De följande typer av meddelanden används i spelet:

1. **Client-SendMove**. Detta kommando skickas av klienten till servern när den ska flytta på en enhet. Meddelandet innehåller följande information:
    1. Kommandotyp. Detta behövs för att klienten ska veta hur paketet ska läsas och utföras.
    2. Enhetens ID. Specificerar vilken enhet som ska röra sig.
    3. Destination.

2. **Client-SendRemovedUnit**. Detta kommando skickas från klienten till servern för att meddela att klienten inte längre ser en av motståndarens enheter. Varför detta meddelande behövs i spelet förklaras i 4.3.2. Meddelandet innehåller följande information:
    1. Kommandotyp.
    2. Enhetens ID.
3. **Server-SendMove**. Detta kommando skickas från servern till den klient som inte äger enheten ifall den klient har minst en enhet som kan se den. Detta behöver inte skickas till enhetens klient eftersom servern mottog kommandot **Client-SendMove** och klienten vet därför redan vart enheten är på väg. Meddelandet innehåller följande information:
    1. Kommandotyp.
    2. Enhetens ID.
    3. Enhetens lag. Om en ny enhet skall skapas på klienten behöver den kunna skapa enheten i rätt lag.
    4. Position. Denna information skickas för att synkronisera med klienterna om klienterna har beräknat fel när enheten ska stanna.
    5. Enhetens riktning. Riktningen skickas istället för position för att tillåta Interest Management som beskrivet i delkapitlet 2.2. Alla enheter har samma hastighet och hastigheten är alltid konstant, vilket gör att ingen hastighet behöver skickas med.
4. **Server-SendStop**. Detta kommando kommer att skickas när en enhet har stannat på servern och meddelar till båda klienterna att enheten har stannat. Detta är ett måste för den klient som inte äger enheten eftersom den bara vet riktningen och har ingen information om vilken destination enheten har. Enhetens klient får meddelandet för att synkronisera att de stannar på samma position. Meddelandet innehåller följande information:
    1. Kommandotyp.
    2. Enhetens ID.
    3. Enhetens lag.
    4. Enhetens position.
5. **Server-SendGameState.** Detta kommando används när servern ska skicka de förändringarna av speltillståndet till klienterna. När inga förändringar sker kommer det inte skickas något meddelande. Meddelandet innehåller följande information:
    1. Kommandotyp.
    2. Antalet objekt i paketet. Detta behövs för att klienten ska veta hur många enheter den ska hämta ut ur paketet.
    3. Alla enheters ID.
    4. Alla enheters lag.
    5. Alla enheters Positioner.

4.2.2 Överföring av kommandoinput

Vid uppstarten av spelet räknar servern ut vilka enheter som vardera klient ska ha information om och skickar kommandot **Server-SendStop** för att skapa enheterna på klienterna. Klienterna kan skicka ett kommando genom att klicka på en enhet och sedan klicka på den position i miljön dit enheten ska röra sig. När detta sker skickas det ett **Client-**

SendMove till servern som då kommer att utföra kommandot och även beräkna ifall sändarens motståndare ska få informationen.

Servern kommer att, för varje uppdatering för varje enhet som motstånden inte har information om, beräkna när motståndaren behöver ha tillgång till den senaste informationen för den enheten, se figur 5 för psuedokod. Detta gör att servern inte behöver skicka information som klienterna redan har, vilket sparar bandbredd. När en klient inte ser en av motståndarens enheter kommer den skicka kommandot **Client-SendRemovedUnit** och även ta bort enheten på klienten. Detta gör att servern kommer att veta vilken information klienten vet om och kan då skicka informationen igen till klienten när enheten kommer in i klientens synfält. Att lita på klienten är inget problem i det här fallet, vilket förklaras i kapitel 4.3.

```
for each( CombatUnit var )
{
        if( var.Update() == true ) //   true == the CombatUnit has change angle or
        {                          //   stop moving and need to inform the other Clients

                if( EnemyInRange( var ) == true )
                {
                        if( var.Moving == true )
                        {
                                SendMove( var )
                        }
                        else
                        {
                                SendStop( var )
                        }
                }
        }

        if( var->EnemyKnowMe() == false )
        {
                // Each update, see if the unit has been moved to a location so the
                // enemy see the unit.

                if( EnemyInRange( var ) == true )
                {
                        if( var.Moving == true )
                        {
                                SendMove( var )
                        }
                        else
                        {
                                SendStop( var )
                        }
                }
        }
}
```

Figur 5 Pseudokod för Enheterna på Servern. Denna bit av Pseudokod har kontroll på när enheterna kommer in i varandras synfält.

En annan lösning är att även servern räknar ut när en klient inte längre ser en motståndarens enhet. Fördelen med denna lösning är att det inte behöver skickas ett kommando från klienten, vilket minskar nätverkstrafiken. Största nackdelen med denna lösning och varför den inte används är att klienten och servern kan få olika resultat av såna beräkningar. Det kan inträffar att servern och klienten inte har exakt samma positioner på grund av fördröjningar mellan meddelanden. Detta är svårt att upptäcka med ett mänskligt öga, men skillnaden i flyttalsrepresentationer av hastigheter och positioner kan leda till problem. Problemet i detta fall är att servern tror att klienten har information om en enhet trots att klienten har tagit bort enheten ifrån sin representation av spelet. Om servern tror

att klienten har information om enheten kommer servern inte skicka någon information om den, vilket gör att klienten har felaktig information.

När en klient inte ser en motståndarenhet måste den ta bort enheten. Ifall den inte tar bort enheten kan det leda till att klienten ser enheten på fel plats när klienten får överblick över den platsen. Problemet är att när klienten inte har överblick över den platsen, kan det hända att enheten har rört sig ifrån platsen till en annan destination och servern skickar inte information om den rörelsen till motståndaren ifall den inte har någon enhet i närheten. Därför måste klienten ta bort information om den enheten och meddela detta till servern.

Istället för att klienten beräknar vilka enheter som försvinner ur dess synfält skulle servern kunna göra det. Servern skulle behöva skicka kommandon till klienten när enheter försvinner ur dess synfält för att inte få felaktig information. Fördelen med att göra beräkningen på klienten är att beräkningarna fördelas mellan klient och server samt att klienten alltid har färre eller som högst lika många enheter som servern, vilket gör att klienten har färre enheter som behöver jämföras.

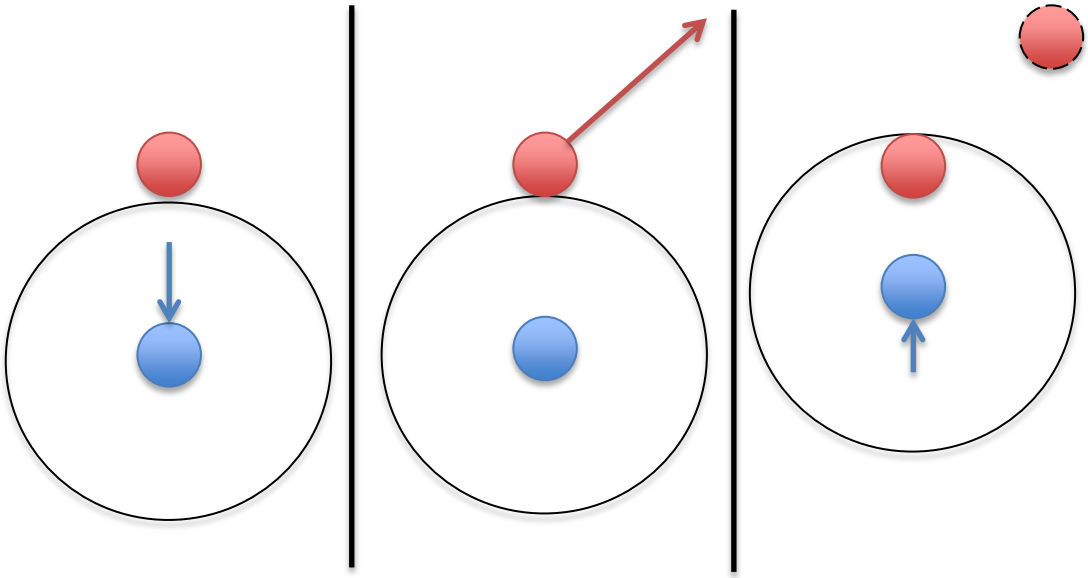
4.2.3 Överföring av speltillstånd

I lösningen med överföringen av speltillstånd, så beräknar servern vilka enheter som respektive klient ska få information om. Klienterna skickar information enligt beskrivningen i delkapitlet 4.3.2. För att minska mängden datatrafik kommer servern spara vilken position den senast skickade till klienterna. Genom att göra det behöver inte servern skicka information om enheter som inte har rört sig.

Servern kommer att beräkna när en motståndares enhet försvinner ur klientens synfält och meddela klienten att enheten ska tas bort, eftersom informationen som klienten har om enheten kan vara felaktig. Anledningen till att inte klienten gör det är att servern gör alla beräkningar och klienten bara tar emot information. Om klienten skulle göra det måste klienten meddela när enheter försvinner för den och det skapar mer nätverkstrafik.

4.3 Förebyggande av fusk

Klienten skickar information till servern när den inte längre ser en motståndares enhet. Detta kan tyckas bryta mot regeln att man inte ska lita på klienten. I detta fall har det ingen betydelse för om klienten väljer att inte skicka **Client-SendRemovedUnit** meddelandet betyder det att klienten kommer att ha felaktig information om enheten, se figur 6. Om klienten skickar meddelandet till servern även om klienten ser den fortfarande, kommer servern bli tillsagd att den enheten inte syns av motståndaren. Nästa uppdatering kommer den kontrollera alla enheter som inte syns av motståndaren om de är inom deras synfält och kommer då få en ny information om den enheten.

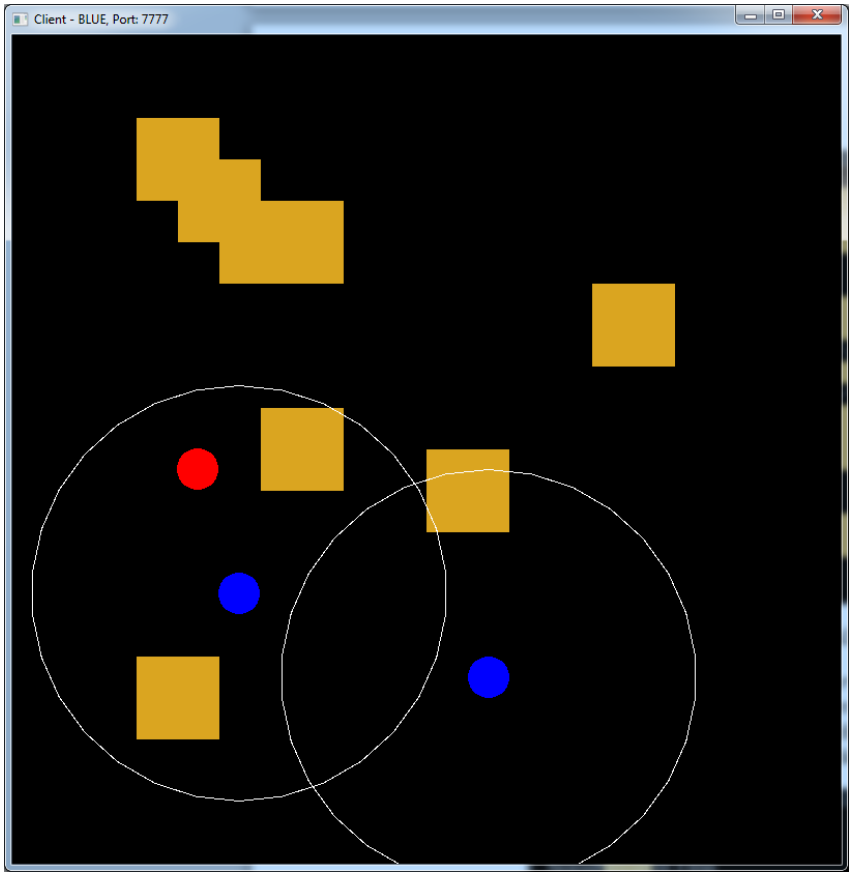


Figur 6 Denna figur visar ett flöde som uppstår när klienten inte skickar **Client-SendRemovedUnit**. Första bilden flyttar den blåa enheten neråt och avbryter meddelandet att den röda försvinner ur den blåas synfält. Nästa bild gör den röda en förflyttning och servern gör en kontroll om den blåa klienten ska ha den informationen. Tredje bilden visar när den blåa enheten gör en förflyttning mot den röda enheten och har kvar den gamla information om den som är då felaktig för den blåa klienten har inte fått informationen om att den rörelsen som den gjorde.

Ett problem som inte hanteras i denna lösning är att enheterna på både servern och klienterna delar samma ID på samma enhet. Detta leder till att användare kan läsa klientens minne för att hitta motståndarenhetens ID och memorera vilka enheter som klienten har sett. När servern skickar att klienten ska se en ny enhet kan användaren kontrollera ifall det är en enhet som klienten har sett tidigare. För att lösa detta problem kan serverns enhet ha två olika ID för varje klient och varje gång en klient ser en fiende kan servern tilldela ett nytt ID för den klienten.

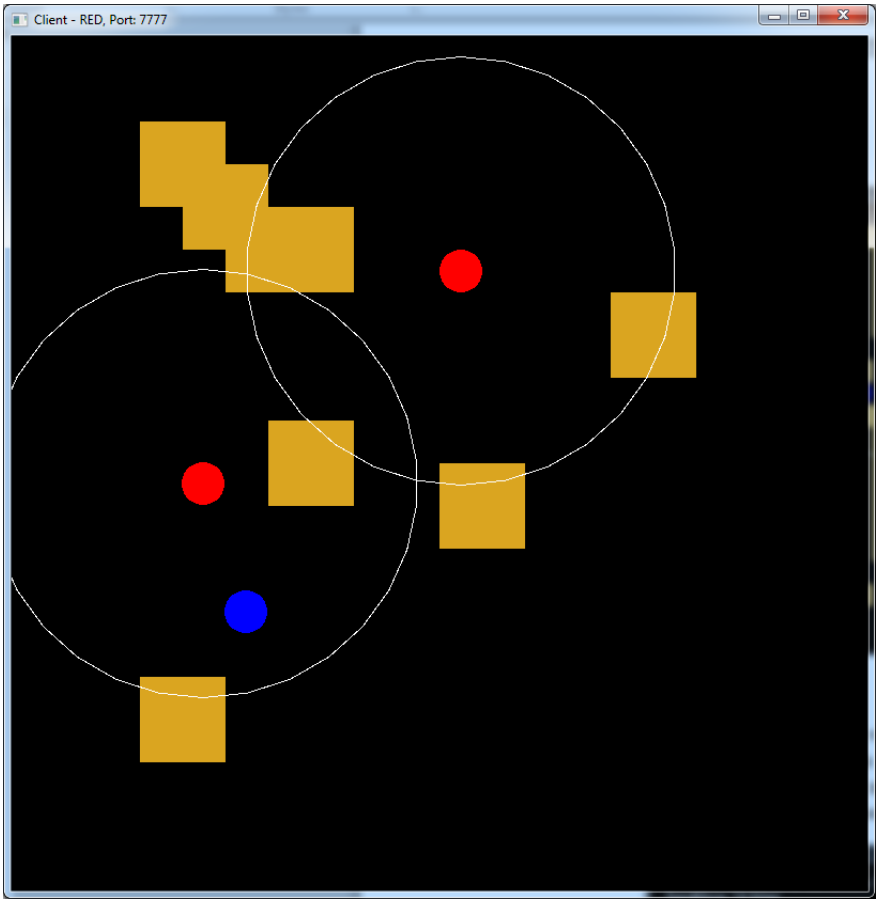
4.4 Simulering

En simulering visas i tre fönster: ett för varje klient och ett för servern. På Figur 7 visas vad den blåa klienten kan se och där syns en röd enhet som befinner sig inom synfältet till den blåa enheten.



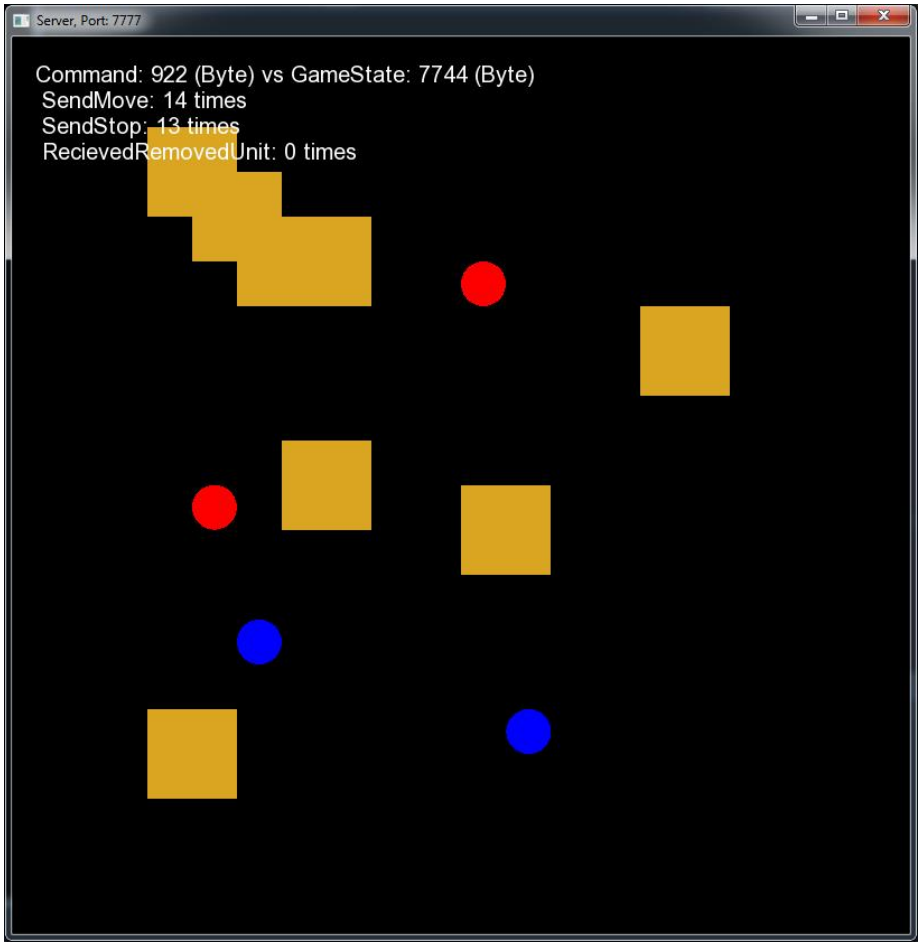
Figur 7 Den blåas klientens vy av speltillståndet

I Figur 8 visas samma spelscenario där en av de blå enheterna inte finns med samt en röd enhet som inte syntes hos den blåa klienten.



Figur 8 Den rödas klientens vy av speltillståndet.

I serverns fönster syns alla enheter, vilket visas i Figur 9. De två siffrorna uppe till vänster visas skillnaden mellan de två metoderna som mäter data som skickas från servern till klienterna. Den högra är kommandoinput metoden och den andra är tillståndsuppdatering metoden och räknar vilka typer av meddelanden som skickas. När en simulering är klar skriver programmet ner resultatet i ett textdokument, vilket gör att det går att simulera flera scenarier samtidigt och samla in data utan att vara närvarande.



Figur 9 Serverns vy av speltillståndet.

5 Utvärdering

5.1 Presentation av undersökning

De scenarier som testats är följande:

- Enkel rörelse förbi fiendeenhet.
- Diagonal rörelse i en trång miljö förbi en fiendeenhet.
- Snabb rörelse in och ut ur fiendens synfält.
- Slumpmässiga rörelser.

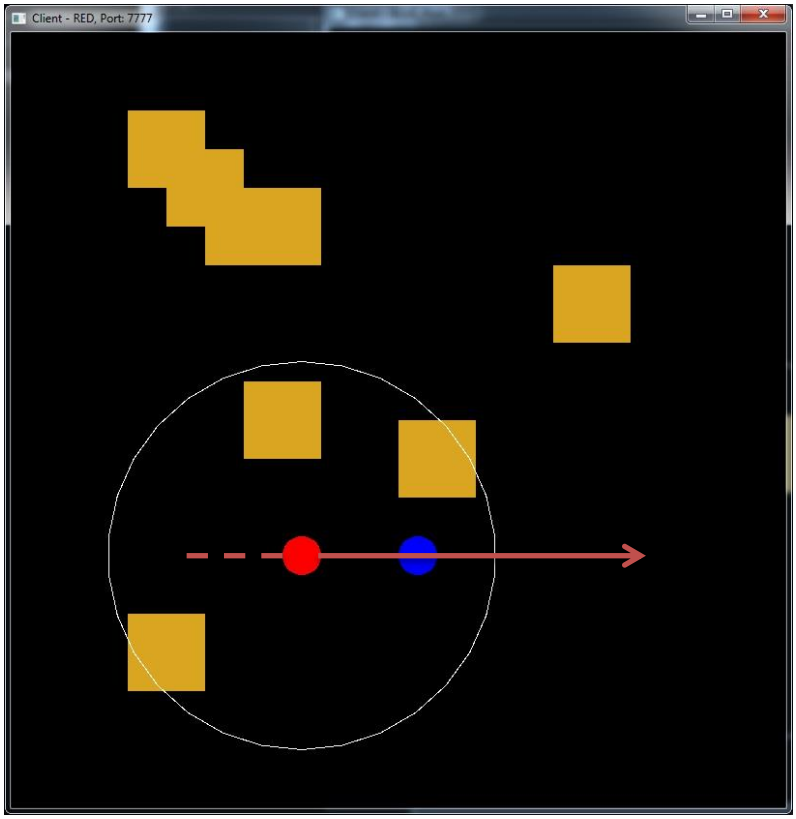
Varje test kördes tio gånger för att få en jämn mätning då det förekom spikar. Spikar i mätningen berodde på att det blev loopar inom nätverket mellan klient och server. När servern skickade information om en enhet till klienten nekade klienten till att den såg den enheten och skickade tillbaka att den inte ser enheten. Detta kunde ske upprepade gånger under kort tid, vilket påverkade mätningen. Dessa mätningar togs bort då en sådan situation inte kan uppstå i en arkitektur där servern har full kontroll över vad klienterna ser, vilket skulle vara fallet i ett fullskaligt realtidsstrategispel.

Det skickas speltillstånd till klienterna 30 gånger per sekund. Varje scenario, förutom scenariot med slumpmässiga rörelser, körs med följande konfigurationer:

- 1 rörlig enhet och 1 stillastående fiende.
- 10 rörliga enheter och 1 stillastående fiende.
- 1 rörliga enhet, 10 fiender som står stilla.
- 10 rörliga enheter och 10 enheter som står stilla.
- Mätningar av scenarier med gradvis ökande (1-10) antal enheter.

5.1.1 Scenario 1: Enkel rörelse förbi fiendeenhet

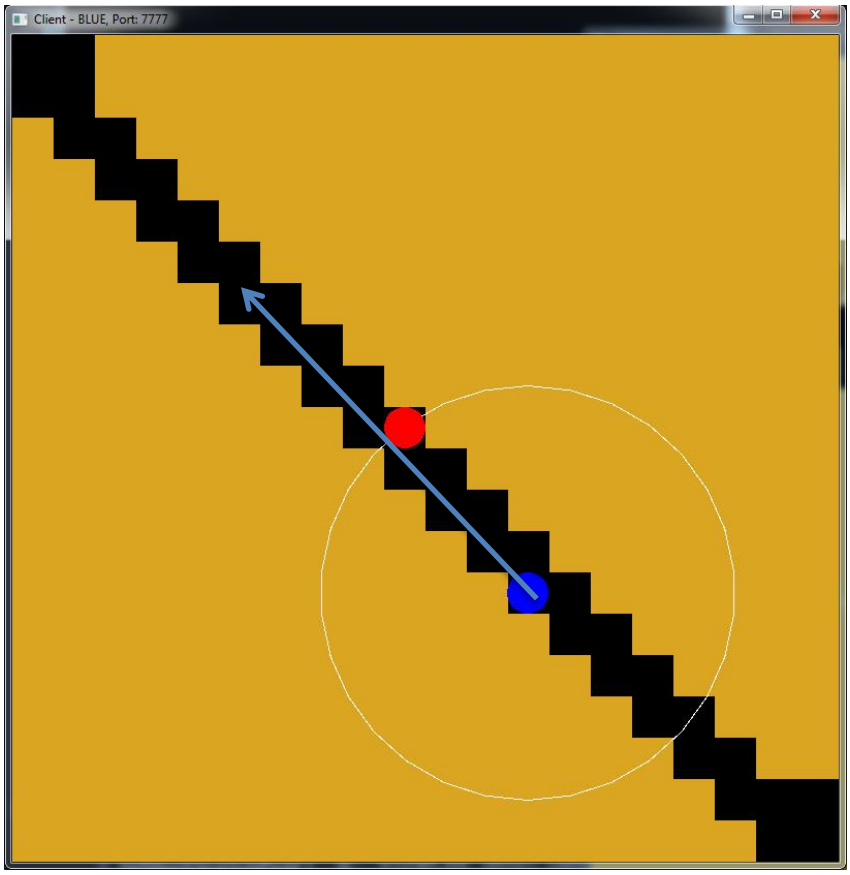
Det första scenariot en enkel rörelse förbi en moståndarenhet, se figur 10. Det är den enklaste rörelsen för kommandoöverföring när det är en endast riktning som sker. Den röda enheten och den blåa enheten startar på positioner som gör att de inte syns på motståndarens klient. Rörelsen sträcker sig till så att båda enheterna inte ser varandra längre. Detta leder till att det kommer skickas information när de är tillräckligt nära samt att klienten kommer att skicka att när enheten försvinner i syn. Mätningen finns i Appendix A för Scenario 1.



Figur 10 Enkel rörelse förbi en moståndarenhet.

5.1.2 Scenario 2: Diagonal rörelse i en trång miljö förbi en fiendeenhet

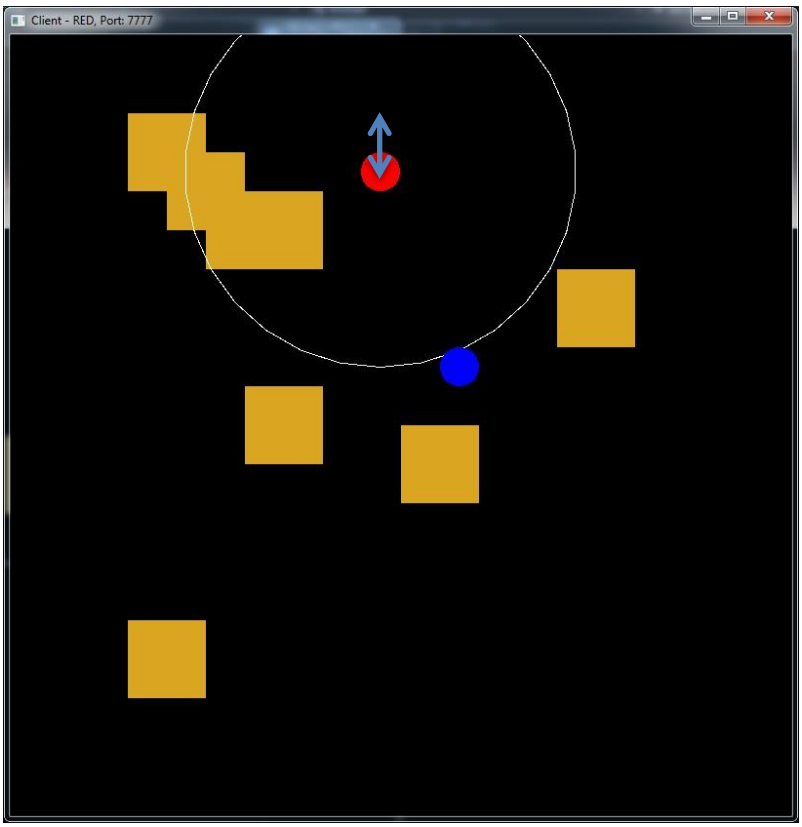
Scenario 2 är som scenario 1 med en trång miljö istället, se figur 11. Detta är till för att testa om det blir högre bandredd i kommandoöverföring när enheten byter riktning som servern måste meddela fienden om. Mätningen finns i Appendix B för Scenario 2.



Figur 11 Diagonal rörelse förbi en moståndarenhet

5.1.3 Scenario 3: Snabb rörelse in och ut ur fiendens synfält

Tredje scenario som testas är när en enhet rör sig snabbt inom ett område fram och tillbaka. När en enhet kommer in i motståndarens synfält skickar servern information till båda klienter i kommandoöverföringensmetoden. Mätningen finns i Appendix C för Scenario 3.



Figur 12 Den röda enheten rör sig snabbt upp och ner för att få effekten att servern behöver skicka mycket data.

5.1.4 Scenario 4: Slumpmässiga rörelser

Experimentet testar även att göra slumpmässiga rörelser för enheterna för att kontrollera skillnaden mellan metoderna vid scenario där enheterna rör sig slumpmässigt. Mätningen finns i Appendix A för Scenario 4.

5.2 Analys

5.2.1 Scenario 1: Enkel rörelse förbi fiendeenhet

I första scenariot där enheter rör sig igenom fiendens synfält, är det väldigt lätt att se att kommandoöverföringsmetoden är mycket mindre bandbreddskrävande än speltillståndsöverföringsmetoden, se tabell 1. När det läggs till 9 stycken extra rörliga enheter ökar den procentuella skillnaden med 400%, se tabell 2. Kommandoöverföringensmetoden ökar mer jämfört med speltillståduppdateringmetodens ifrån tabell 1. Detta beror på att när den rörliga enheten kommer närheten av de stilla stående enheterna skickar servern ett meddelande för varje enhet till klienten och det är väldigt onödigt och det påverkar kommandoöverföringen väldigt mycket.

Tabell 3 är det 10 enheter som står stilla och en enhet som rör sig. Lösningen som skickar speltillstånds data är inte långt ifrån jämfört med tabell 1. Kommandoöverföring har dock ökat mycket för det är många meddelanden som måste skickas per enhet. De stillastående enheterna bör inte skicka med på nytt för nästa speltillstånd skickas de bara om de har rört sig, vilket gör låga siffror jämfört med tabell 2.

Tabell 4 visar när testet körs med 10 rörliga enheter och 10 stillastående enheter och det visar att kommandoöverföringen är fortfarande överlägsen i låg dataöverföring. Det som förbättrades för speltillståndsuppdateringen som gjorde att det blev lägre skillnad är för vid varje uppdatering skickas det med alla förändringar i ett och samma meddelande istället för varje enhet som skickar ett meddelande vid varje förändring som kommandoöverföring gör. Detta är den stor nackdel hos den nuvarande kommandoöverföringsmetoden. Detta leder till som visas på tabell 5, desto mer enheter det är desto sämre blir kommandoöverföring på grund av att det skickas enskilda meddelanden för varje enhet vid samma tillfälle som speltillståndsuppdateringarna skickar alla enheter i ett och samma meddelande.

5.2.2 Scenario 2: Diagonal rörelse i en trång miljö förbi en fiendeenhet

När en enhet ska ta sig igenom en trång miljö ökar kommandoöverföringen jämfört med en öppen miljö, jämför tabell 1 och 6. Detta beror på att det behöver skicka flera kommandoöverföringar för varje ny riktning som sker.

5.2.3 Scenario 3: Snabb rörelse in och ut ur fiendens synfält

På tabell 11 utförs experimentet med att en enhet rör sig in och ut ifrån en fiendes synfält och vid sådana tillfällen börjar kommandoöverföringen närma sig speltillståduppdaterings data. När det är 10 enheter som rör sig in och ut ur en fiendes synfält, se tabell 12, så blir det väldigt stora ändringar. Då börjar kommandoöverföringsmetoden bli mer bandbreddskrävande än speltillståduppdateringsmetoden. Detta beror på att enheterna går in och ut ur fiendens synfält väldigt ofta som leder till att det skickas kommandon lika ofta som speltillståndsuppdateringar och problemet är att kommandon skickas enskilt och ett speltillståndsuppdateringsmeddelande skickar alla enheter i samma meddelande. Testet körde med 30 uppdateringar i sekunden och om det skulle köras i 60 uppdateringar i sekunden skulle istället kommandoöverföringsmetoden varit mindre bandbreddskrävande. Då är frågan, hur många gånger behöver speltillståndsuppdateringar skickas?

5.2.4 Scenario 4: Slumpmässiga rörelser

På de slumpmässiga rörelserna kan det konstanteras att det speltillståduppdateringsmetoden blir effektivare än kommandouppdateringsmetoden desto

mer enheter det är, se Tabell 16 och 17. Detta är samma problem som de andra scenarierna att kommandouppdateringarna skickar ett meddelande per kommando istället för ett meddelande för alla kommando vid den uppdateringen.

5.3 Slutsatser

Hypotesen säger att genom att skicka över uppdelade kommandon till klienterna är mindre bandbreddskrävande än att skicka över speltillståndet. Generellt stämmer denna hypotes i de flesta scenarion som har testats i detta arbete. Men det finns tillfällen då kommandoöverföringsmetoden är sämre och det största problemet är att vid vissa tillfällen där många ändringar sker måste det skickas flera kommandoöverföringsmeddelanden. Detta skulle kunna optimeras genom att skicka flera kommandon i ett meddelande vid varje uppdatering som sker.

Denna studie fokuserar på förflyttning av enheter i ett RTS spel och det finns många andra faktorer som avgör om kommandoöverföringsmetoden är bättre än tillståndsuppdateringar, t.ex. hastigheter, acceleration och explosion av granater. Men denna studie visar att kommandoöverföringsmetoden är mindre bandbredds krävande än tillståndsuppdateringar vid kommunikation av enhetsförflyttning i ett RTS.

Ett problem som uppstår med att skicka flera kommandon i olika meddelanden är att de inte kommer fram samtidigt. När det är väldigt många kommandon som ska skickas kan nätverket bli belastat, vilket leder till fördröjningar. Lockstep (Terrano & Bettner, 2001) är en metod som skulle behöva implementeras för att få en deterministisk simulering på alla klienter. Lockstep delar in simulationen i tidssteg och exekverar kommandon vid samma tidssteg på alla klienter.

Ett problem var att klienten skickar till servern vilka enheter som försvinner ur klientens synfält. När klientens enhet kommer inom fiendeenhetens synfält skickar servern information om den. Vid vissa tillfällen kunde klienten ha dessa två enheter utom synhåll ifrån varandra samtidigt som servern hade dessa enheter inom varandras synfält. Vid detta tillfälle skickar servern till klienten ”Du ser enhet A”. Klienten får detta meddelande och skickar tillbaka ”Jag ser inte enhet A längre”, på grund av att hos klientens är de två enheterna för långt ifrån varandra. Detta kan leda till evighetsloopar och ökad kommandoöverförings data, vilket gav problem i mätningen. Om speltillståndet har två enheter, en i vardera lag, och om servern och klienten inte har 100% identiska speltillstånd är det ineffektivt att ge uppdrag till båda att räkna ut avståndet mellan enheterna. I vissa fall fick de olika resultat och skickade onödiga meddelanden. Tanken var att försöka fördela arbetet till klienten för att undvika belastning på servern, dock var det dumt att ge två noder samma uppdrag.

Skillnaden mellan överföringarna i ett meddelande är att kommandoöverföringen innehåller en riktning. Om metoden för kommandoöverföring av en enhetsförflyttning har samma storlek som ett tillståndsuppdateringsmeddelande för en enhet skulle data mängden för kommandoöverföring bara överstiga datamängden för tillståndsuppdateringar om antalet meddelanden är större. Det förekommer bara om en enhet ändrar riktning 30 gånger per sekund, något som är väldigt sällsynt.

6 Avslutande diskussion

6.1 Sammanfattning

Att skicka uppdelade kommandon till klienterna är mindre bandbreddskrävande än att skicka speltillståndet, som hypotesen säger. I scenariot med störst skillnad mellan metoderna var kommandoöverföring 10 gånger effektivare än speltillståduppdatering. Dock fanns det scenarion där kommandoöverföring var mer bandbräddskrävande än speltillståndsuppdatering, vilket beror på att meddelanden skickas per enhet i kommandoöverföring medan speltillståndsuppdatering skickar alla enheter i ett meddelande.

Mätningen kördes på fyra scenarior:

- Enkel rörelse förbi en fiendeenhet
- Rörelse genom trång miljö
- Enhet rör sig snabbt in och ut ur fiendens synfält
- Slumpmässiga rörelser

I det första scenariot var kommandoöverföring överlägsen vinnare eftersom enheten då bara hade en riktning. I scenariot med trång miljö minskade skillnaden mellan kommandoöverföring och speltillståndsöverföring men kommandoöverföring var fortfarande effektivare. När enheter rör sig in och ut ur fiendens synfält blev det oväntat resultat. Kommandoöverföring översteg mängden data som speltillståndöverföring hade. Detta beror på att när mer än en enhet rör sig in i fiendens synfält skickas det ett meddelande per enhet till klienten för att informera den om vilka enheter den ser. Detta var en faktor som gjorde stor skillnad som inte var beräknad.

6.2 Diskussion

Arbetsutgångspunkten var att förhindra fuskmöjligheter i datorspel. Bursztein, Hamburg, Lagarenne & Boneh (2011) skapade en metod som krypterar data på klienten vilket gör att det blir svårare för program att hitta data i klienten. Chambers, Feng, Feng & Saha (2005) skapade en metod där klienterna skickar information om de lokala enheternas vy och får information ifrån servern om de enheter som befinner sig inom denna vy. Klienterna kan dock utnyttja detta genom att inte tala sanning om vad deras vy är. När en klient får information om deras vy kan den användas för att se vart på kartan fiendens enheter befinner sig. I denna studie hanteras problemet genom att använda Klient/Server arkitekturen där en server har hela speltillståndet och beräknar vilka enheter som ska synas på vardera klient.

I arbetet som beskrivs av Buro (2002) skickar speltillståndsuppdateringar med Klient/Server arkitekturen, något som inspirerat i denna studie. Denna metod är väldigt effektiv för att lösa fuskmöjligheter inom datorspel, problemet med speltillståndsuppdateringar är att de skickas väldigt ofta och använder mycket bandbredd. För att lösa detta problem använder denna studie kommandoöverföring av den typ som Terrano & Bettner (2001) använder i *Age of Empires* (Ensemble Studios, 1997). Servern kan dock inte skicka hela kommandon eftersom klienterna får mer information än vad de har rätt till. Kommandon delas upp i olika steg som servern informerar klienten om. Detta börjar

då likna Client-Prediction (W. Bernier, 2001) där klienterna förutspår vart enheterna kommer att befinna sig innan klienten får nästa information ifrån servern om enheterna. Skillnaden är att klienten beräknar med hjälp utav förgående position, vilken hastighet och riktning som skiljs ifrån uppdelade kommandoöverföring där servern skickar vilken riktning enheterna rör sig.

E-sport scenen är väldigt stor i dagens samhälle och det finns spelare som försöker uppnå toppen genom att använda sig av fusk. Metoder som förebygger fusk gör spelet mer rättvist och mer sportsligt. Metoder som förhindrar fusk kan dock leda till andra problem. Till exempel kan extra servrar krävas, vilket kostar pengar för utvecklarna.

Denna studie testades inte i något kommersiellt RTS spel och kan därför inte visa exakt hur effektivt uppdelade kommandoöverföringar är i ett sådant. Något som kan konstateras är att det är mindre bandbreddskrävande än speltillståndsuppdateringar. Om klienterna vet hela speltillståndet behöver inte kommandon delas upp och servern behöver endast skicka ett meddelande per kommando.

Forskningsetiskt är det viktigt att arbetet är upprepbart. Det är stort möjligt att upprepa studien, men det krävs att servern har samma villkor för när den ska skicka meddelanden till klienterna. Om arbetet utförs på annan hårdvara eller använder ett annat biblotek kan resultatet skilja sig något från denna studie, men resultatet kommer sannolikhet ha liknanden resultat. Den största orsaken till eventuella skillnader är att det inte är deterministiskt när det uppstår spikar i mätningen, se kapitel 5.1. Arbetet använder sig utav öppen programvara som stärker upprepbarheten.

6.3 Framtida arbete

För att vidarutveckla arbetet och göra det mer anpassat för kommersiella RTS är två förbättringar att implementera lockstep samt att skicka alla kommandon som utförs samtidigt i samma meddelande. Lockstep skulle försäkra att två kommandon som utförs direkt efter varandra på en klient, skulle utföras samma steg på andra klienter. Detta skulle minska mängden data som skickas

Om denna studie kunde användas i ett kommersiellt RTS skulle det vara intressant att se om överföring av uppdelade kommandon är tillräckligt snabb. Studien utfördes i en miljö där bara enheter rörde på sig och det finns mycket annan information som behöver kommuniceras i ett kommersiellt RTS som hastigheter och acceleration etc.

Resultaten från detta arbete, i synnerhet kommandouppdelningen, skulle implementeras i ett biblotek för användning i spelmotorer samt även för de vanliga programmeringsspråken som C++, C# och Java etc. Biblioteket kan ha då stöd för hanteringen av nätverkskommunikationen och speltillstånd mellan klient och server.

Det är svårt att hitta användningsområden för uppdelade kommandon i andra områden än spel. Metoden är till för system som har ett antal noder, där en är huvudnoden som har kontroll över all data. Om en av noderna skickar information till huvudnoden som sedan ska skickas vidare till de andra noderna, så är det helt onödigt att dela upp kommandon. Skälet till att dela upp kommandon är för att dölja information som är till för att ge ett spel mer utmaning för spelaren.

Att en server skickar information om vilken riktning enheter är på väg kan användas i t.ex. flygplan vars riktning kan användas för att förutspå vart flygplanet är på väg. En server kan skicka ut information till flygplanen som fungerar som klienter och som meddelar vilken riktning de håller tillbaka till servern.

Referenser

Blizzard Entertainment (1998) StarCraft [Datorprogram] (Blizzard Entertainment)

Blizzard Entertainment (1994) Warcraft: Orcs & Humans [Datorprogram] (Blizzard Entertainment)

Buro, M. (2002). ORTS: A Hack-Free RTS Game Environment. *Lecture Notes in Computer Science*. Vol.2883, ss. 280 - 291. ISBN: 978-3-540-20545-6.

Bursztein, E., Hamburg, M., Lagarenne, J. & Boneh, D. (2011). OpenConflict: Preventing Real Time Map Hacks in Online Games. *Security and Privacy (SP), 2011 IEEE Symposium on*. ss. 506-520. DOI: 10.1109/SP.2011.28

Diffen (2016). TCP vs. UDP. http://www.diffen.com/difference/TCP_vs_UDP [2016-02-18]

Cerf, V & Kahn, R. (1974). A Protocol for Packet Network Intercommunication. *Communications, IEEE Transactions on*. vol. 20, ss. 637 – 648. DOI: 10.1109/TCOM.1974.1092259

Chambers, C., Feng, W., Feng, W. & Saha, D. (2005). Mitigating Information Exposure to Cheaters in Real-Time Strategy Games. *NOSSDAV '05 Proceedings of the international workshop on Network and operating systems support for digital audio and video*. ss. 7 – 12. DOI: 10.1145/1065983.1065986

Ensemble Studio s (1997) Age of Empires [Datorprogram] (Ensemble Studios)

Hart, P.E., Nilsson, N.J. & Raphael, B. (1968) A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4, s. 100–107.

Terrano, M. & Bettner, P. (2001). 1500 Archers on 28.8: Network Programming in Age of Empires and beyond. Tillgänglig på http://www.gamasutra.com/view/feature/131503/1500_archers_on_a_288_network_.php?page=1 (Hämtad 2016-02-11)

SFML Team (2016) Simple and Fast Multimedia Library (2.3.2) [datorprogram]. Tillgängligt på http://www.sfml-dev.org/index.php (Hämtad 2016-04-07)

Smed, J., Kaukoranta, T. & Hakonen. H. (2002). Aspect of networking in multiplayer computer games. *The Electronic Library*. vol. 20, ss.87 – 97. DOI: 10.1108/02640470210424392

W. Bernier, Y. (2001). Latency Compensation Methods in Client/Server In-game Protocol Design and Optimization. Tillgänglig på https://developer.valvesoftware.com/wiki/Latency_Compensating_Methods_in_Client/Server_In-game_Protocol_Design_and_Optimization (Hämtad 2016-05-16)

Wohlin, C., Runson, P., Höst, M., Ohlsson, M.C., Regnell, B. & Wesslén, A. (2012) *Experimentation in Software Engineering*. DOI: 10.1007/978-3-642-29044-2

Appendix A - Scenario 1: Enkel rörelse förbi fiendeenhet

Tabell 1 Resultatet med 1 stillastående enhet och 1 rörlig enhet. Speltillståduppdateringsmetoden skickar 1247% data utav kommandoöverföringsmetoden.

Stillastående enheter:	1	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
368	2796	760%
176	2796	1589%
206	2776	1348%
206	2816	1367%
206	2816	1367%
368	2796	760%
152	2708	1782%
368	2796	760%
206	2816	1367%
206	2816	1367%
	Medelvärde:	1247%

Tabell 2 Resultatet med 1 stillastående enhet och 10 rörliga enheter. Speltillståduppdateringsmetoden skickar 837% data utav kommandoöverföringsmetoden.

Stillastående enheter:	1	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
1520	12520	824%
1520	12464	820%
1466	12572	858%
1376	12612	917%
1574	12456	791%
1520	12628	831%
1490	12372	830%
1520	12500	822%
1520	12516	823%
1466	12480	851%
	Medelvärde:	837%

Tabell 3 Resultatet med 10 stillastående enheter och 1 rörlig enhet. Speltillståndsuppdateringsmetoden skickar 324% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
962	3012	313%
962	2992	311%
962	2992	311%
668	2772	415%
962	3032	315%
908	2952	325%
962	3012	313%
962	2992	311%
962	2992	311%
962	2992	311%
	Medelvärde:	324%

Tabell 4 Resultatet med 10 stillastående enheter och 10 rörliga enheter. Speltillståndsuppdateringsmetoden skickar 397% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
3080	12692	412%
3080	12576	408%
3080	12768	415%
3080	12608	409%
3620	12660	350%
3080	12720	413%
3404	12804	376%
3080	12744	414%
3080	12996	422%
3620	12808	354%
	Medelvärde:	397%

Tabell 5 Resultatet med olika antal enheter. Medelvärdet är ointressant i detta fall då de inte använder sig av samma förutsättningar.

	Stillastående enheter:	1 -> 10	
	Rörliga enheter:	1 -> 10	
	Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
1	206	5096	2474%
2	520	7564	1455%
3	384	7172	1868%
4	1472	11844	805%
5	3784	13548	358%
6	2856	16068	563%
7	3332	17968	539%
8	5536	18672	337%
9	4716	21984	466%
10	5300	23844	450%

Appendix B - Scenario 2: Diagonal rörelse i en trång miljö förbi en fiendeenhet

Tabell 6 Resultatet med 1 stillastående enhet och 1 rörlig enhet. Speltillståduppdateringsmetoden skickar 372% data utav kommandoöverföringsmetoden

Stillastående enheter:	1	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
932	2684	288%
716	2976	416%
716	2856	399%
1148	2976	259%
716	2996	418%
678	2884	425%
716	2996	418%
1094	2976	272%
716	2956	413%
716	2976	416%
	Medelvärde:	372%

Tabell 7 Resultatet med 1 stillastående enhet och 10 rörliga enhet. Speltillståduppdateringsmetoden skickar 199% data utav kommandoöverföringsmetoden

Stillastående enheter:	1	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
6674	13116	197%
6674	13368	200%
6674	13176	197%
6674	13192	198%
6728	13420	199%
6674	13248	199%
6484	13296	205%
6560	13168	201%
6728	13364	199%
6674	13104	196%
	Medelvärde:	199%

Tabell 8 Resultatet med 10 stillastående enheter och 1 rörlig enhet. Speltillståndsuppdateringsmetoden skickar 263% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
1164	3080	265%
1202	3172	264%
1202	3092	257%
1164	3100	266%
1202	3192	266%
1202	3172	264%
1202	3020	251%
1202	3152	262%
1202	3192	266%
1202	3192	266%
	Medelvärde:	263%

Tabell 9 Resultatet med 10 stillastående enheter och 10 rörliga enheter. Speltillståndsuppdateringsmetoden skickar 178% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd(byte)**	**Kommando / Speltillstånd**
7160	12740	178%
7160	13304	186%
8240	13504	164%
7160	12948	181%
7160	13664	191%
8240	13508	164%
7700	13132	171%
7160	12856	180%
7160	13464	188%
7160	13120	183%
	Medelvärde:	178%

Tabell 10 Resultatet med olika antal enheter. Medelvärdet är ointressant i detta fall då de inte använder sig av samma förutsättningar.

	Stillastående enheter:	1 -> 10	
	Rörliga enheter:	1 -> 10	
	Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
1	1202	2316	193%
2	1540	4064	264%
3	2796	5232	187%
4	3004	6412	213%
5	4498	7568	168%
6	6996	8016	115%
7	6146	9564	156%
8	5728	11236	196%
9	6444	12388	192%
10	8780	12844	146%

Appendix C - Scenario 3: Snabb rörelse in och ut ur fiendens synfält

Tabell 11 Resultatet med 1 stillastående enhet och 1 rörlig enhet. Speltillståduppdateringsmetoden skickar 127% data utav kommandoöverföringsmetoden

Stillastående enheter:	1	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
2190	2884	132%
2666	3416	128%
2680	3504	131%
2566	3276	128%
2250	2984	133%
2456	3308	135%
2852	3376	118%
2380	2984	125%
2944	3544	120%
2838	3504	123%
	Medelvärde:	127%

Tabell 12 Resultatet med 1 stillastående enhet och 10 rörliga enheter. Speltillståduppdateringsmetoden skickar 71% data utav kommandoöverföringsmetoden

Stillastående enheter:	1	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
20974	14884	71%
19650	14224	72%
20954	14560	69%
20676	13608	66%
19904	14352	72%
20346	14756	73%
20138	14288	71%
21050	14708	70%
20674	14620	71%
19390	14324	74%
	Medelvärde:	71%

Tabell 13 Resultatet med 10 stillastående enheter och 1 rörlig enhet. Speltillståndupppdateringsmetoden skickar 56% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	1	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
9778	5252	54%
9208	5172	56%
9838	5232	53%
9778	5192	53%
9298	5212	56%
8270	5092	62%
9238	5212	56%
8330	5052	61%
9268	5232	56%
9748	5212	53%
	Medelvärde:	56%

Tabell 14 Resultatet med 10 stillastående enheter och 10 rörliga enhet. Speltillståndupppdateringsmetoden skickar 62% data utav kommandoöverföringsmetoden

Stillastående enheter:	10	
Rörliga enheter:	10	
Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
26018	15748	61%
26282	16076	61%
26234	16232	62%
25282	15028	59%
26178	16356	62%
26860	16276	61%
25158	16172	64%
24688	15708	64%
25778	16268	63%
25066	15724	63%
	Medelvärde:	62%

Tabell 15 Resultatet med olika antal enheter. Medelvärdet är ointressant i detta fall då de inte använder sig av samma förutsättningar.

	Stillastående enheter:	1 -> 10	
	Rörliga enheter:	1 -> 10	
	Kommando (byte)	**Speltillstånd (byte)**	**Kommando / Speltillstånd**
1	2492	3168	127%
2	5300	4856	92%
3	7940	6560	83%
4	9054	6844	76%
5	13468	9896	73%
6	15582	10680	69%
7	17292	12016	69%
8	19834	13424	68%
9	21576	14600	68%
10	25302	16700	66%

Appendix D - Scenario 4: Slumpmässiga rörelser

Tabell 16 Resultatet med 2 rörliga enheter. Speltillståduppdateringsmetoden skickar 1398% data utav kommandoöverföringsmetoden

Antal Enheter:	2	
Kommando (byte)	**Speltillstånd(byte)**	**Kommando / Speltillstånd**
2280	50556	2217%
3594	52124	1450%
2794	50932	1823%
3378	52044	1541%
6578	57320	871%
3770	51536	1367%
6968	58332	837%
2646	50872	1923%
5958	54436	914%
5226	54120	1036%
	Medelvärde:	1398%

Tabell 17 Resultatet med 20 rörliga enheter. Speltillståduppdateringsmetoden skickar 338% data utav kommandoöverföringsmetoden

Antal enheter:	20	
Kommando (byte)	**Speltillstånd(byte)**	**Kommando / Speltillstånd**
109458	340524	311%
101660	354376	349%
98912	351480	355%
103536	344504	333%
100608	349056	347%
100926	349192	346%
106050	352296	332%
101434	349548	345%
106218	351728	331%
106534	353800	332%
	Medelvärde:	338%