CHALMERS

Battle of Stupid Sorcerers

Hantering av nätverkskommunikation i realtidsspel

Kandidatarbete inom Data- och Informationsteknik

Andreas Arvidsson Mattias Carlsson
Sebastian Lagerman Henning Phan
Kim Strömberg Andreas Wånge

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Kandidatarbete/rapport nr 2013:19
Göteborg, Sverige, Juni 2013

Battle of Stupid Sorcerers
Hantering av nätverkskommunikation i realtidsspel

Andreas Arvidsson
Mattias Carlsson
Sebastian Lagerman
Henning Phan
Kim Strömberg
Andreas Wånge

Examiner: Daniel Cederman

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2013

Abstract

Delays in network communication occur constantly and are virtually unavoidable, but there are some methods that can be used to mitigate this problem. In this report, a study of network management made with the tool Battle of Stupid Sorcerers (BoSS) is presented. BoSS is a network game developed by the project group and is based on a Warcraft III custom map called Warlock. With the help of this tool, methods such as interpolation and extrapolation have been implemented and evaluated. These methods are described individually; first theoretically, followed by their implementation in BoSS. Subsequently followed by an evaluation of the methods' effectiveness and impact on the game; possible problems and risks are also mentioned. Such difficulties are balanced against the methods' advantages.

The group found that many of the problems that exists regarding network communication in real-time games can be compensated for. However, often there's a cost involved in other fields when this is done, and it is necessary to take this into consideration in order to create a good and coherent gaming experience.

Sammanfattning

Fördröjningar i nätverkskommunikation förekommer ständigt och är praktiskt taget oundvikliga, men det finns vissa metoder som kan användas för att hantera detta problem. I denna rapport kommer studier av nätverkshanteringen gjorda med hjälp av verktyget Battle of Stupid Sorcerers (BoSS) att presenteras. BoSS är ett nätverksspel som utvecklades av projektgruppen och är baserat på den spelarskapade Warcraft III-banan Warlock. Med hjälp av detta verktyg har metoder såsom interpolering och extrapolering implementerats samt utvärderats. Dessa metoder beskrivs individuellt; först teoretiskt, sedan hur de blivit implementerade i BoSS. Därefter följer en utvärdering av metodens effektivitet och inverkan på spelupplevelsen; eventuella problem och risker påpekas också. Sådana svårigheter balanseras mot metodens fördelar.

Gruppen kom fram till att många av de problem som manifesterar sig inom nätverkskommunikation i realtidsspel går att dölja eller kompensera för. Detta innebär dock ofta en kostnad inom andra område, och det krävs att man tar hänsyn till detta för att skapa en bra och enhetlig spelupplevelse.

Förord

Rapporten är skriven i samband med ett kandidatarbete som utfördes under våren 2013 vid Institutionen för Data- och Informationsteknik på Chalmers Tekniska Högskola.

Gruppen vill tacka Daniel Cederman för sin tid och hjälp som projekthandledare, Arne Linde (kursansvarig) för hjälp med resurser, Ulf Assarsson för hans hjälp med insikter och tips vid skapandet av spellogiken samt Alexander Poon för hans bilder av trollformler han skapade till vårt projekt. Utöver dessa personer vill gruppen tacka Ann-Marie Eriksson, Calle Carlsson och Keith Comer för deras hjälp med rapportskrivandet.

Vi vill också tacka företagen Valve och ID Software för sin öppenhet med material rörande både nätverkshantering och övriga spelrelaterade koncept. I en marknad med hög konkurrens släpper de information om hur de löste problem och på så sätt hjälper blivande utvecklare.

Kandidatsarbetsgrupp 19

Innehåll

1

Inledning

1.1 Bakgrund

Att konfigurera, implementera och optimera nätverkskommunikation är ett ämne som forskats mycket om sedan internets uppkomst. Ökade krav på applikationer har medfört att denna forskning blivit mer relevant, framför allt inom växande områden såsom videoströmning och spel. I en studie utförd av J. Jacko et. al. [1] kan vi se att användare har idag låg tolerans för applikationer med dålig responstid. Studien är förvisso utförd på websidor, men dessa har ganska låga krav på sig, och redan där uppenbarar sig problem. Att den då kan appliceras på andra typer av nätverksapplikationer är inte ett stort antagande.

Actionspel är en genre som är väldigt intressant att studera då denna typ av spel sker i realtid, vilket betyder strikta krav på låg nätverksfördröjning. Vidare ska applikationen klara att tolka informationen från nätverket, vilken ofta är omformatterad för nätverksoptimering. Dålig hantering av denna nätverkskommunikation kan förstöra spelupplevelsen genom att skapa buggar[1], fördröjningar och/eller inkonsekvenser.

1.1.1 Warcraft III - strategispel med nätverkshantering i realtid

Warcraft III är ett PC-spel utvecklat av spelföretaget Blizzard och släpptes den tredje juli 2002[2] [2]. Spelet tillhör genren RTS (Real Time Strategy) och går ut på att skapa byggnader, arméer och hjältar för att förstöra motståndarens högkvarter.

[1] Fel i datorprogram

[2] Detta är datumet spelet släpptes i USA - det släpptes i Sverige två dagar senare.

Spelet blev fort en bästsäljare med över en miljon sålda exemplar under den första månaden [3]. Bland annat innehöll spelet en revolutionerande cutscene-teknik, djup berättelse och en engagerande kampanj där hjälten växte under spelets gång.

En uppskattad del av produkten var *World Editor*, ett medföljande verktyg som möjliggjorde tillverkning av egna banor. Utöver att vara kraftfullt, är det även användarvänligt - i och med detta kunde vem som helst skapa egna banor och publicera dem på internet. Det var även enkelt att dela med sig av banorna via deras online-system *Battle.net*. Detta har lett till att en mängd populära banor växt fram - så kallade *custom maps*.

Det fanns banor av varierande komplexitet, men majoriteten av dem var enkla i sin uppbyggnad. Ett exempel på en sådan enkel bana är *Warlock*.

1.1.2 Warlock - reaktionsbaserat actionspel

Warlock är en tredjepartsbana till Warcraft III skapad av användarna *Zymoran* och *Demestotenes* och publicerad på *Hive Workshop* [4]. Spelet utspelar sig på en liten arena som krymper med tiden. Runt denna arena finns ett område som gör skada på spelaren vid kontakt. Spelarna slåss mot varandra med hjälp av olika offensiva och defensiva trollformler tills det bara är en överlevande kvar.

En match består av ett konfigurerbart antal rundor. Varje runda börjar med möjligheten att köpa och uppgradera trollformler samt föremål. Därefter påbörjas striden, där varje spelare utnyttjar de förbättringar som erhållits för att besegra motståndarna. I slutet av rundan fördelas poäng baserat på olika kriterier. Spelaren med flest poäng efter att alla rundor har avslutats vinner matchen. För en mer utförlig beskrivning av spelet och dess uppbyggnad, se bilaga C.

1.2 Problem

All kommunikation som sker över nätverk har en viss fördröjning som orsakas av t.ex. avstånd och nätverksstockning[3][5, Sec. II]. Detta kan skapa stora problem för nätverksapplikationer som måste köras i realtid, till exempel Warlock. I dessa typer av applikationer finns många variabler som måste synkroniseras mellan användare snabbt. Det är av stor vikt att möta de förväntningar som spelarna har för att skapa en bra spelupplevelse. Om nätverksfördröjningar blir alltför tydliga kommer det inte bara bli svårare för den drabbade spelaren att styra, utan hela spelet kommer upplevas som orealistiskt och långsamt. Om en spelare ser att en projektil träffar en motståndare, betyder inte det att

[3] Mängden data som skickas över nätverket blir för stor, vilket leder till försämrad/fördröjd överföring.

en annan spelare ser samma händelseförlopp. Det simultana utbytet av stora mängder realtidsinformation med ett flertal klienter, samt den låga toleransen för paketförluster och fördröjningar när alla klienter ska få samma information samtidigt, gör att just realtidsspel har stora krav på effektiv nätverkshantering.

Dessa problem uppkommer givetvis inte bara i spel utan kan även ses i en rad olika applikationer vars information måste skickas till olika användare samtidigt. Strömning av multimedia, till exempel, lider av många liknande problem [6]. Dock tillåts en viss mängd fördröjning i livesändningar, eftersom de inte är interaktiva; användaren kan ligga några sekunder efter utan att det märks. Många av de metoder som beskrivs i rapporten går ändå att använda sig av i andra områden än spel - men ofta går det att finna mer specialiserade metoder.

1.3 Syfte

Syftet med projektet är att undersöka hur man utformar nätverkshantering med fokus på realtidsspel. Med nätverkshantering menas hantering av fördröjningar, paketförluster, tillståndssynkronisering samt minimering av nätverksbelastning.

Målet är att presentera och demonstrera den införskaffade informationen inom detta ämne. Mer specifikt skall de nätverksproblem som uppstår till följd av kraven på snabb kommunikation från ett nätverksspel i realtid resoneras kring. Potentiella lösningar till dessa problem ska implementeras och utvärderas. För att möjliggöra detta, finns ett delmål att utveckla ett grafiskt realtidsspel för flera spelare. Detta spel kommer att fungera som ett verktyg för att testa de olika metoder som är relevanta för actionspel. Ett annat delmål är att presentera metoder och arkitekturer som passar bra till andra typer av spel.

1.4 Utmaningar

Det är stor sannolikhet att problem uppkommer när det gäller att felsöka nätverkskommunikation över internet. Detta är främst på grund av skillnader i routerkonfigurationer hos användare, olika mängd trafik på nätverket, samt brandväggar och andra säkerhetsmjukvaror. Dessa variabler är något som måste tas hänsyn till för att kunna testa de lösningar som presenteras i denna rapport.

En annan utmaning för projektgruppen blir att införskaffa kunskap om verktygen som kommer användas vid utvecklingen av projektet. Det kommer att vara många nya koncept och arbetssätt som komplicerar utvecklingsprocessen. Dessutom är nätverkshante-

ring ett komplicerat ämne; vissa algoritmer som implementeras kommer därför att vara primitiva, för att kunna demonstrera de metoder som bygger på dessa.

1.5 Avgränsningar

Detta projekt innehåller två delmål, där båda kan vara projekt i sig. För att hinna anskaffa relevanta resultat för syftet måste därför avgränsningar specifieras och hållas. Dessa riktas mestadels mot speldelen, då denna huvudsakligen ses som ett verktyg.

1.5.1 Spelprogrammeringen

Främst skall den spellogik som går att finna i det ursprungliga Warlock implementeras. Fokus kommer att ligga på den del av spellogiken som bidrar till att demonstrera nätverkskommunikationen. Detta för att en stor del av logiken agerar likadant när den synkroniseras över nätverket.

Icke essentiella grafiska element har låg prioritet, och därför kommer arbete som i vanliga fall utförs av designers att undvikas. Detta innefattar skapande av texturer, modeller, animationer och partikeleffekter, då detta inte tillför något till projektets syfte.

1.5.2 Nätverksdelen

Klienten ska implementera funktionalitet som hanterar nätverksproblem och ger effekten av en responsiv spelupplevelse. Detta innefattar att dölja fördröjning och förlust av paket. Dessutom skall klienten omvandla serverns relativt låga tillståndsuppdateringsfrekvens till en högre bilduppdateringsfrekvens för att åstadkomma ett mindre hackigt spelflöde och därmed hålla mängden överförd data låg.

1.6 Metod

Den största delen av informationen är hämtad från källor på internet. Där går det att få åtkomst till en stor mängd artiklar inom ämnet som är producerade av forskare, utvecklare, ingenjörer och spelföretag. Vidare har kursböcker i spelprogrammering och nätverkskommunikation givit en stabil teoretisk bas för implementeringarna.

Rapporten kommer inledningsvis att beskriva de verktyg och metoder som använts vid utvecklingen av spelet. Därefter följer två kapitel som rör vid både teori och resultat för spel- respektive nätverksutvecklingen. Varje underavsnitt i dessa kapitel inleds med

teori, vilket följs av projektgruppens utvärdering samt implementering av dessa teorier. Rapportens huvudstoff avslutas med ett kapitel om projektgruppens slutsatser. Här presenteras nettoresultatet av arbetet med tillhörande analys, samt de aspekter som inte undersökts och vad som lämnas åt framtida undersökningar.

De resultat som rapporten presenterar erhölls delvis via egna undersökningar och delvis från externa rapporter. Undersökningarna genomfördes av projektgruppens medlemmar genom att göra subjektiva bedömningar av resultaten dragna utifrån egna erfarenheter av spel. Olika parameterinställningar experimenterades med för att undersöka hur de olika presenterade metoderna uppförde sig.

2

Arbetsgång

2.1 Verktyg

För att inte spendera mer tid än nödvändigt på för projektet irrelevanta lågnivåaspekter, nyttjas en rad etablerade verktyg som sköter versionshantering, informationslagring och spelgrafik. Detta möjliggör mer fokus på projektets syfte och höjer produktiviteten.

2.1.1 Unity

Spelklienten utvecklas med det interaktiva utvecklingsverktyget och spelmotorn Unity. Detta utvecklingsverktyg har ökat enormt i populäritet, och har fått stor uppmärksamhet av framförallt indieutvecklare[1] men även av större företag. Verktyget erbjuder många funktioner som normalt sett kräver en stor utvecklingsgrupp att implementera - Exempel på dessa funktioner är renderingsmotor, animeringsmotor, interaktiv utvecklingsmiljö, resurshantering och färdigt skriptsystem. Några exempel på spelföretag som har använt unity är InXile [7] och Rovio [8].

Fördelar med Unity är dess stora användarbas och all utvecklarinformation samt dokumentation som finns tillgänglig på internet. Att arbeta med Unity är effektivt tack vare enkelheten att använda verktyget. Detta har lett till mer tid och ett större fokus på nätverksaspekterna, då projektgruppen har sparat tid genom att använda Unitys inbyggda funktioner. Som en extra bonus har Unity möjliggjort att programmet kan kompileras till flera olika plattformar - förutom Windows, Mac, Linux och Web finns även stöd för många olika spelkonsoler samt smartphones.

[1] En eller flera utvecklare utan finansiellt stöd av förlag

2.1.2 Git

Git är ett program initialt skapat av Linus Torvalds [9] för distribuerad versionshantering[2], som möjliggör att flera utvecklare kan modifiera samma kodbas utan att riskera att någon ändring blir överskriven. Verktyget betonar hastighet och effektivitet. Varje lokal arbetskatalog är en fullfjädrad datakatalog med komplett revisionshistorik[3]. Git är inte beroende av nätverksåtkomst eller en central server.

2.1.3 C#

Valet av programmeringsspråk för såväl server som klient föll på C#, ett modernt och populärt språk som även stödjs av Unitymotorn. C# bygger officiellt på C++ [10], men är mer likt Java då även C# kompileras till bytekod och körs i en virtuell maskin.

C# utvecklades av Microsoft och designades för att vara ett enkelt, starkt typat, modernt och objektorienterat programmeringsspråk. Språket använder sig av .NET-funktionalitet, vilket innefattar numeriska algoritmer, undantagshantering, säkerhet och annan funktionalitet som inkluderas i klassbiblioteken. Unity använder en multiplattformskompilator för C# kallad Mono som är kompatibel med .NET-ramverket. Användandet av en sådan kompilator medför fördelen att programmet inte är låst till ett operativsystem.

Att använda samma programmeringsspråk på både server- och klientsidan ger märkbara fördelar som underlättar arbetsgången. Dels är serialisering[4] och deserialisering trivialt, dels är delning av kod mellan server och klient avsevärt enklare. Man skriver mindre kod, vilket leder till färre buggar i projektet.

2.1.4 JSON

JSON (JavaScript Object Notation) är ett kompakt och lättviktigt textbaserat format för att lagra samt överföra informationen över nätverket [11]. Det är designat för att vara effektivt att tolka och generera för datorn, samtidigt som det är lättläst samt enkelt att skriva för användare. JSON:s textbaserade format är oberoende av programmeringsspråk, av vilka de flesta moderna sådana har stöd för, antingen officiellt eller inofficiellt (skapat av tredjepartsutvecklare[5]). Det används vanligtvis som ett alternativ till XML och har fördelen att det i många fall är effektivare än XML [12].

[2]Förmågan att tidigare versioner av en kodbas sparas och skillnader mellan versionerna spåras.
[3]Lista med sparade förändringar i kodbasen
[4]En process där ett objekt konverteras till ett format som kan skickas över nätverket
[5]Utvecklare som har inga förbindelser till grund tillverkaren

3

Battle of Stupid Sorcerers

BoSS (Battle of Stupid Sorcerers) är verktyget projektgruppen utvecklat och använt för att utvärdera nätverkshantering (se figur 1). Verktyget är ett spel enligt Wolfgang Kramer definition [13] och är utvecklat i spelmotorn Unity. Syftet med denna produkt är att implementera spellogik från Warlock, vars grundläggande principer finns beskrivna i bilaga C.

(Figur 1: Battle of Stupid Sorcerers)

3.1 Fundamentala spelkomponenter

Verktyget som utvecklats i detta projekt innehåller ett par viktiga fundamentala spelkomponenter som gynnas av en lite mer genomgående beskrivning. I nästkommande stycken förklaras hur BoSS hanterar kollisioner och visualisering, eftersom implementeringen av dessa komponenter inte är helt självklara.

3.1.1 Kollisioner

Kollisionshantering är en av de grundläggande komponenterna inte bara i datorspel utan inom många simulerade miljöer (bland annat inom fysiken och medicinen), samt modelleringsprogram och dylikt. Kollisionshantering delas officiellt in i tre underkategorier:

- Kollisionsdetektering: Fastställer hurvida två eller fler objekt kolliderar i en virtuell miljö.
- Kollisionsfastställande: Avgör exakt vart de involverade objekten kolliderar.
- Kollisionsrespons: Bestämmer vilken åtgärd som ska tas för de kolliderade objekten.

Då spel kan innehålla hundratals rörliga objekt är det viktigt att snabbt kunna utföra kollisionshantering för att minska risken att beräkningarna fördröjer uppritning av bildrutorna. Detta är ett stort ämne där det utförs stora mängder forskning för att kunna hitta effektivare algoritmer. Det har till och med börjat experimenteras med att använda GPUn för att parallellisera beräkningnarna för att tillåta mer objekt i en scen [14]. En naiv algoritm för kollisiondetektering där man alltid kontrollerar alla objekt mot varandra skulle utföra kontroller för varje bildruta, där en kontroll tar varierande tid beroende på objekten och dess geometriska komplexitet.

$$n \cdot m + \binom{n}{2} = n \cdot m + \frac{n(n-1)}{2}$$

Ekvation för antalet beräkningar vid naiv kollisionsdetekteringsalgoritm [15]
n är antalet rörliga objekt och m är antalet statiska objekt

Naiva lösningar fungerar i små spel eller spel där kollision är det enda krävande momentet (dvs avsaknad av andra tidskrävande beräkningar), men skulle aldrig fungera i ett avancerat spel.

För att förenkla kollisionshanteringen i spelet, valde projektgruppen att implementera logiken helt i 2D. Cirklar används som kollisionsgeometri för att undvika komplicerad

beräkningar. Detekteringen är implementerad som den tidigare nämnda naiva lösningen, där alla objekt testas mot alla andra objekt. Detta eftersom implementering av smartare algoritmer kräver bättre datastrukturer och dessa skulle ta orimligt mycket tid för projektgruppen. Dessutom har BoSS inte så mycket beräkningstunga komponenter att en naiv lösning skulle dra ner på kvaliten.

Kollisionsdetektering i 2D visade sig fungera utmärkt, likaså att använda cirklar istället för mer korrekt modellerad geometri. Även om cirklarna inte speglar den verkliga situationen, fungerar de tillfredställande bra.

3.1.2 Visualisering

Visualisering av spelvärlden ger det första intrycket hos spelaren, och det är viktigt att man har en genomtänkt sådan. Det primära syftet för BoSS är att fungera som ett verktyg för testning av nätverkshantering, och därför har inte mycket fokus lagts på visualiseringen. Dock renderas spelet i 3D, vilket tillåter användning av Unitys inbyggda funktionalitet för detta. Eftersom själva logiken är 2D, kan spelet definieras som 2.5D [16].

3.2 Spelets mekanik

Spelmekanik är konstruktioner av regler som syftar till att skapa visst beteende i spelet. När en spelare interagerar med spelet sker detta genom mekaniken.

3.2.1 Matchstruktur

Varje runda i BoSS består av två faser. Den första är en affärsfas som utspelar sig under trettio sekunder då man kan köpa trollformler och föremål för att förbättra sina chanser under resten av matchen. Den andra är arenafasen; denna fas karaktäriseras av ett slag, där målet är att vara den sista kvarstående spelaren. Målet uppnås genom att man med hjälp av sina införskaffade föremål och trollformler besegrar sina motståndare. Då det enbart är en spelare kvar avslutas rundan.

3.2.2 Spelkaraktärer

Varje spelare styr en spelkaraktär, som börjar med en mängd guld och ett antal trollformler. Guldet kan spenderas på att införskaffa och uppgradera trollformler under affärsfasen. Mer guld kan fås genom att man besegrar andra spelkaraktärer under arenafasen.

Spelkaraktären börjar med 100 hälsopoäng, vilket är standardmaxvärdet. Inför varje ny runda återställs dessa till spelarens aktuella maxvärde. Hälsopoängen kan minskas på två sätt: antingen genom att karaktären blir träffad av trollformler kastade av andra spelare (som beskrivs i 3.2.5), eller att den står i lava (som beskrivs i 3.2.3). Hälsopoängen ökar kontinuerligt upp till maxvärdet genom en konstant hälsoåterhämtning.

3.2.3 Spelplanen

Spelets arena omges av ett stort lavaområde, som är farligt för spelare att beträda. Att stå inom detta området medför att spelaren förlorar fyra hälsopoäng per sekund. Allt eftersom arenafasen fortgår kommer arenan minska i storlek. Detta innebär att spelet blir intensivare med tiden tills det att arenan slutligen försvinner.

3.2.4 Rörelser

Rörelserna för de olika dynamiska objekten i spelet följer fysikaliska formler. Vissa är gemensamma för alla dynamiska objekt och vissa är specifika för olika typer av objekt. Alla dynamiska objekt har en position, en extern hastighet, och en intern hastighet.

Den interna hastigheten är en vektor som motsvarar ett objekts hastighet utan yttre påverkningar. För spelkaraktären är den interna hastigheten vektorn från spelarens position till målet, med en statisk magnitud. För trollformler är den interna hastigheten dess normala hastighet riktad mot den punkt den kastades mot. Magnituden på vektorn är specifikt definierad för varje typ av objekt i en JSON-fil.

Den externa hastigheten består av all yttre påverkan på objektet, vilket för spelkaraktärer inkluderar bakåtstötar och andra effekter skapade av trollformler. I nästkommande avsnitt kommer dessa effekter beskrivas mer i detalj.

3.2.5 Trollformler

Det finns ett antal olika typer av trollformler - projektiler vars syfte är att kastas mot andra spelare, områdestrollformler som påverkar en viss spelyta, samt trollformler som påverkar spelaren själv. Alla trollformler har en så kallad *väntetid*. Detta är tidsperioden efter att en trollformel har kastats tills att man kan kasta samma trollformel igen. Den visas för spelaren genom att trollformelns ikon delvis täcks av en skugga och att en siffra visas i hörnet av ikonen som visar antal sekunder tills väntetiden är slut och trollformeln återigen kan kastas. Väntetiden för varje enskild typ av trollformel är definierad i ett JSON-objekt. Förutom väntetid har alla trollformler en livstid. Detta är den maximala tiden en trollformel kan vara aktiv. Om en projektil inte träffat ett annat objekt kommer

den att försvinna efter livstiden tagit slut.

Det tar en halv sekund att kasta en trollformel. Under tiden som en spelkaraktär kastar en trollformel visas en animation. Spelkaraktären kan inte gå och kasta en trollformel samtidigt, när detta sker sätts karaktärens interna hastighet till noll och trollformeln kastas. Däremot kan spelaren kasta en trollformel samtidigt som den påverkas av bakåtstötar eller andra externa krafter. Om en spelare ger ett förflyttningskommando under den tiden då den förbereder sig för att kasta trollformeln så avbryts kastningskommandot och trollformeln kastas inte. Spelkaraktären kan inte kasta mer än en trollformel samtidigt.

Alla trollformler har en ägare; i normala fall är det spelaren som kastade den. Trollformler kan endast kollidera med spelkaraktärer som inte är dess ägare. Projektiler kan kollidera med andra projektiler, men inte när de har samma ägare. Ifall en projektil kolliderar med en sköld som inte ägs av dess ägare så byter projektilen ägare till sköldens värd.

De trollformler som implementerades var de som ansågs vara essentiella för att testa nätverkskommunikationens olika delar.

Figur 2: Eldklot [17]

En projektil som färdas i en rak riktning och skadar det första objektet det kolliderar med. Denna trollformel möjliggör initial och grundläggande testning av linjära rörelsemönster. Ikonen i verktyget som representerar denna trollformeln är figur 2.

Figur 3: Målsökande missil [18]

En projektil som är målsökande mot den närmsta fienden och skadar det första objektet det kolliderar med. Implementering av målsökande missil tillåter testning av ickelinjär objektförflyttning. Ikonen i verktyget som representerar denna trollformeln är figur 3.

Figur 4: Teleportering [19]

Teleportering som namnet syftar på förflyttar en spelkaraktär omedelbart. Tillåter testning av omedelbar förflyttning över längre avstånd. Ikonen i verktyget som representerar denna trollformeln är figur 4.

Figur 5: Sköld [20]

En barriär runtom spelkaraktären som reflekterar fientliga projektiler som kolliderar med den. Detta möjliggör testning av kraftiga riktningsbyten. Ikonen i verktyget som representerar denna trollformeln är figur 5.

Figur 6: Blixt [21]

Blixt är en ögonblicklig trollformel som skadar det första objektet i sin väg. Planerades att utvärdera tillbakaspolning. Ikonen i verktyget som representerar denna trollformeln är figur 6.

Figur 7: Plåga [22]

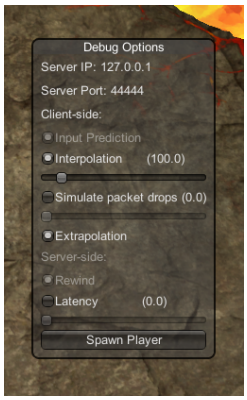
Plåga är en områdestrollformel som knuffar iväg fientliga spelkaraktärer som står nära den karaktär som utför trollformeln. Den första områdestrollformeln för att testa områdeseffekter. Ikonen i verktyget som representerar denna trollformeln är figur 7.

3.2.6 Bakåtstötar

Bakåtstöt sker när en spelkaraktär blir träffad av en trollformel som tillhör en annan spelare. Karaktären trycks då bort av en impuls från projektilen eller mittpunkten för områdestrollformelns effektyta. Hela bakåtstötens fart appliceras med en gång och minskar över tiden genom en friktionskonstant. Storleken på impulsen beror både på trollformeln som orsakar bakåtstötens bakåtstötsmagnitud, och ett värde på extra mottaglighet för bakåtstöt som finns i spelkaraktären. Detta värde ökas varje gång en spelkaraktär tar skada, vilket medför att en spelare som tagit mycket skada under en runda kommer att förflyttas mycket längre än en spelare som inte tagit skada.

Bakåtstöt fungerar enligt förväntningar och förhoppningar, förutom vid ett specifikt fall där ifall en spelkaraktär blir tryckt in i en motståndare. Då är den önskvärda effekten att spelaren själv ska överföra sin kinetiska energi till sin motståndare i en elastisk stöt. Den erhållna effekten just nu är att spelaren kommer att behålla sin hastighet och försiktigt knuffa undan sin motståndare för att sedan fortsätta i sin bana.

3.3 Kontroller



Figur 8: Kontrollpanel

För att kunna testa de nätverkshanteringsmetoder som gruppen har implementerat, skapades en kontrollpanel till verktyget BoSS. Denna panel innehåller inställningar för simulering av paketförlust, möjligheten att aktivera/deaktivera metoder såsom interpolering och extrapolering (vilka beskrivs utförligt i kapitel 4), modifiera parametrar till ovannämnda metoder, samt skapa icke kontrollerbara karaktärer. Paketförluster simuleras genom att nyttja en slumptalsgenerator för att bestämma om ett anlänt paket ska kastas eller inte. Frekvensen detta skall ske bestäms genom en justerbar parameter vars värde kan vara mellan noll och ett, vilket motsvarar noll respektive hundra procents paketförlustchans. Se figur 8.

Med hjälp av kontrollen har gruppen haft möjlighet att experimentera med och utvärdera olika inställningar av nätverkshanteringsmetoder. Karaktärerna som kunde skapas av kontrollen var till stor hjälp för att testa trollformellogiken. Detta eftersom de tillät en ensam utvecklare att testa funktionalitet som i vanliga fall krävde flera klienter.

3.4 Utvärdering av spel

Spelet implementerar inte alla funktioner från Warlock, men är en fungerande prototyp. Den implementerar tillräckligt med spelmekanik för att fylla sin roll som verktyg för att utvärdera nätverkshanteringen. Kommunikation, tolkning av indata, och speltillstånd är implementerade. Detta medför att tester som utvärderar hur dessa koncept påverkas av de hanteringsmetoder som diskuteras i kapitel fyra enkelt kan utföras. Sådana tester har gjorts med hjälp av kontrollen som nämndes i 3.3. Olika inställningar och nivåer av bland annat paketförluster valdes, varefter spelkaraktären manipulerades i enlighet med spelmekaniken. De valda inställningarnas inverkan på de tidigare nämnda koncepten noterades manuellt av testaren.

4

Nätverk

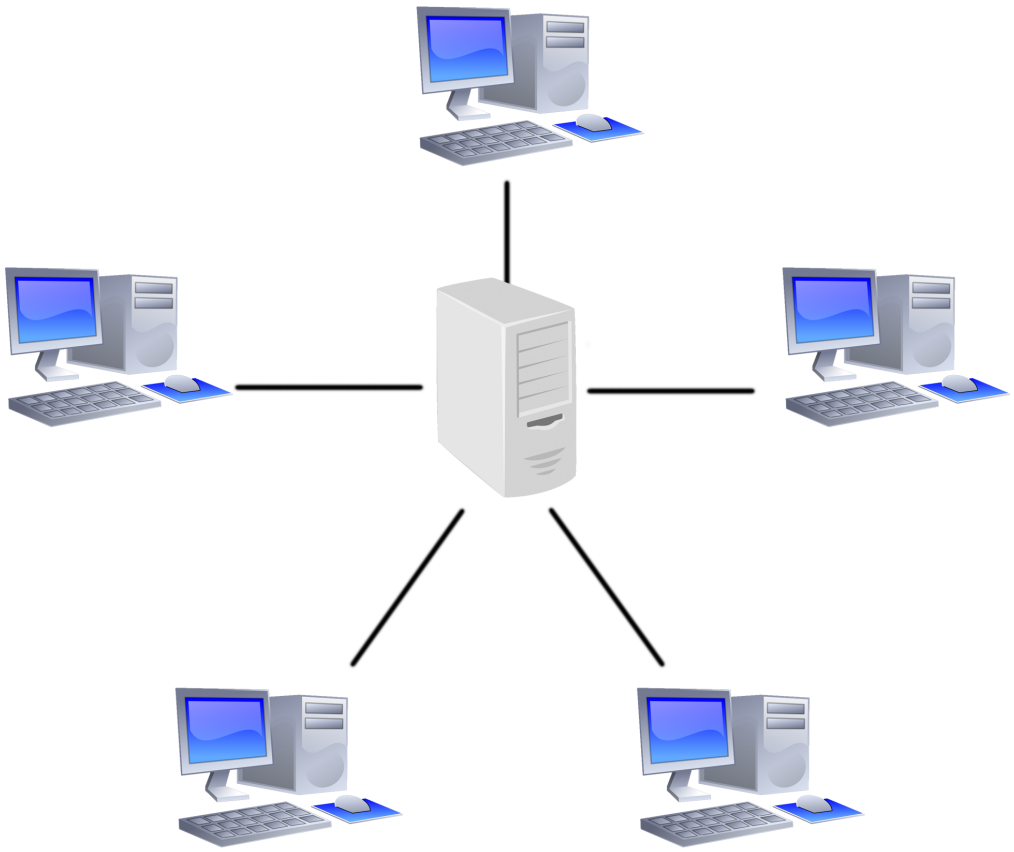
Det finns en mängd olika problem förknippat med hantering av nätverk. Målet är att hitta ett sätt att komma runt de fördröjningar och paketförluster som uppkommer för att kunna erbjuda en kommunikation som ger illusionenen av att ske i realtid.

4.1 Nätverksarkitektur

Termen nätverksarkitektur beskriver de fysiska komponenter, resurser samt relationen mellan dessa i ett nätverk. De flesta arkiteturer som används i moderna nätverksapplikationer bygger på två nätverkstopologier: Klient-server och Peer-to-peer. Dessa fungerar på väldigt olika sätt och var och en har sina för- respektive nackdelar; de utvärderas och en väljs utefter vad som passar bäst för projektets syfte.

4.1.1 Klient-server

Klient-server är en populär arkitektur med en server som sköter kommunikationen mellan olika värddatorer (klienter). Generellt används en auktoritär server, vilket innebär att servern är den som bestämmer vilket speltillstånd som är giltigt



(Figur 9: Klient-server topologin [23, 24])

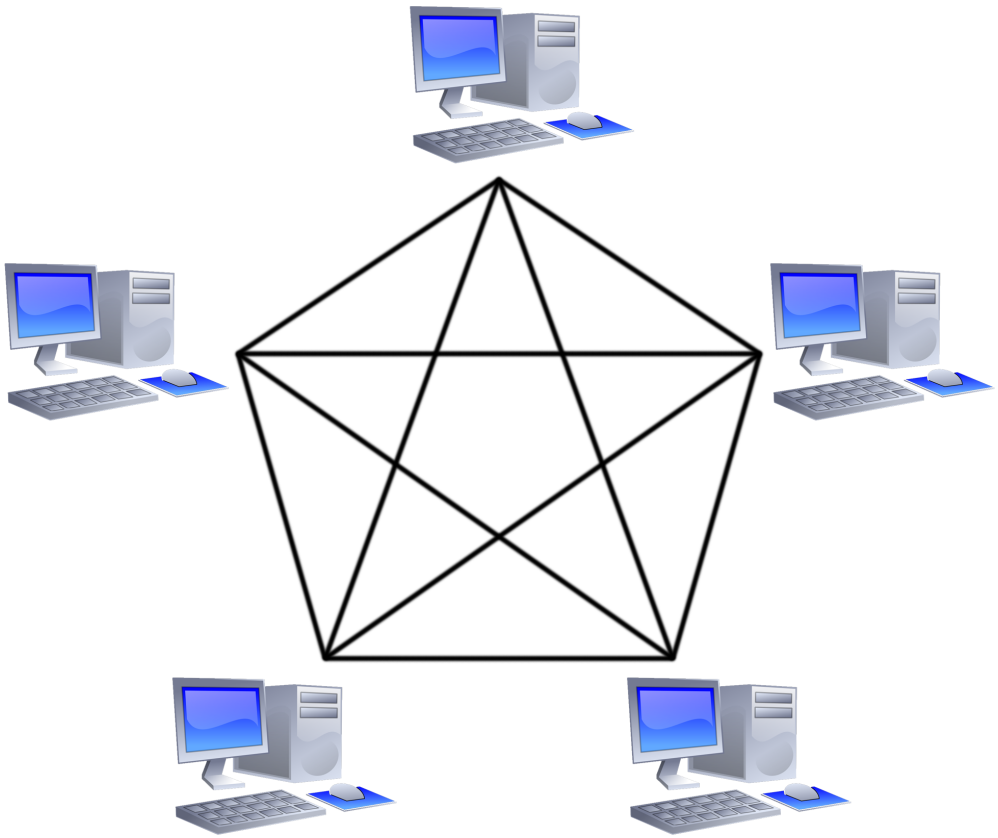
I spelsammanhang skickar varje klient kommandon till servern, som sedan använder dessa för att simulera spelet. Som beskrivet i [25] skickar servern periodiskt spelets tillstånd till alla klienter; detta leder till att klienter aldrig direkt kommunicerar med varandra (se figur 9). Ifall man implementerar all logik och låter alla uträkningar ske på servern, kan klienten bli väldigt simpel då den endast har två funktioner. Den första är att skicka kommandon till servern och den andra är visualisera de tillstånd som anlänt från servern. Denna metod används bland annat av QuakeWorld och Unreal Engine 3 [26], samt även spel baserade på Source-motorn, såsom Half-Life och Counter-Strike [27].

Implementationen av denna arkitektur kan ske på ett par olika sätt. Ett vanligt sätt är att ha en dedikerad server på separat hårdvara. Man kan då distribuera servermjukvaran mellan många servrar för att få en bra täckning över hela världen. Ett billigare och ofta använt alternativ är att använda en lokal server (även kallad *listen server*) [28]. Detta innebär att en klient agerar server och spelare samtidigt. Personen som är värd för matchen, i lobbyn, tilldelas ofta rollen som både server och spelare. Praktiskt fungerar denna server på samma sätt som en dedikerad server - men det tillkommer en del noterbara

effekter. Ett exempel är att nätverkskommunikationen blir beroende av en (ofta mindre tillförlitlig) klient, vilket kan leda till att en match har hög fördröjning som följd av en instabil uppkoppling hos värden.

4.1.2 Peer-to-peer

Peer-to-peer använder en nätverkstopologi där alla noder är direktkopplade med alla andra noder [25, 29]. I denna modell simulerar alla klienter spelet genom att allting sker synkront. Alla kommandon en klient utför sprids till de övriga klienterna, och beräkningar för dessa kommandon sker synkroniserat (se figur 10). Detta innebär att alla klienter har samma väldefinierade tillstånd, givet att alla klienters spelinstanser exekveras på samma sätt, vilket inte är garanterat. Flyttalsberäkningar behöver inte nödvändigtvis ge samma resultat på olika datorer [30], vilket ger bieffekten att beräkningar kan få olika resultat. Eftersom beräkningsresultaten skiljer sig åt, är det omöjligt att garantera samma väldefinierade tillstånd hos alla klienter. Därför måste alla beräkningar vara deterministiska.



(Figur 10: Peer-to-peer topologin [24])

Peer-to-peer som arkitektur brukar anses vara det bästa alternativet i spel där det finns tusentals spelenheter som ska synkroniseras, då spelarna endast kommunicerar med kommandon. Individuella enheters data beräknas då lokalt istället för att överföras över nätverket.

Nackdelar uppkommer dock som följd av att klienterna har auktoritet. Det blir komplicerat att förhindra eventuella fusk, vilket förklaras i kapitel 4.3. Även synkronisering blir icke-trivialt då det är många olika klienter som måste enas om ett gemensamt tillstånd. Som tidigare nämnts är en förutsättning för detta att alla tillståndsberäkningar är deterministiska, annars riskerar man att tillstånden drabbas av *fjärilseffekten* [25]. Fjärilseffekten innebär att ett litet fel i beräkningen propagerar och förvärras vid varje efterföljande beräkning. Om detta sker hos en klient kommer denne till slut få ett helt annorlunda tillstånd än övriga klienter. Synkroniseringsbuggar som dessa har uppkommit i bland annat utvecklingen av Warcraft [31].

4.1.3 Implementering

Nätverksarkitekturen i detta projekt följer klient-server-modellen och är uppbyggd på ett sådant sätt att klienterna endast ritar upp speltillstånd och tar emot användarens kommandon. Detta innebär att det finns en centraliserad auktoritär instans, som utför logikberäkningar och vars tillstånd är det korrekta. Mer information om flödet på klienten och servern finns i bilaga D.

Servern skickar ut tillståndsuppdateringar till alla klienter var 50:e millisekund (20 tillstånd per sekund). Mellan tillstånden kommer servern att simulera spelet med hjälp av kommandon från spelarna.

4.1.4 Utvärdering

Att implementera klient-server arkitekturen bidrog med en rad fördelar till BoSS.

Enkelt att implementera: Att separera renderingen av tillstånd och spellogiken är ett designval som många spel använder sig av - även i spel utan nätverksstöd. Separering av dessa till två olika fysiska enheter var relativt enkelt att implementera. Kommunikationen mellan server och klient innehåller mycket information om speltillståndet, och detta gör det aningen mer komplicerat vid konstruktion och tolkning av paket.

Säkerhet: Eftersom all information passerar igenom servern som fungerar som ett nav är det lätt att förhindra fusk genom att ha kontroller i servern. Detta skulle vara svårare att implementera i peer-to-peer arkitekturen och i kapitel 4.3 tas spelsäkerheten upp i

BoSS med hänsyn till topologin.

Synkronisering: I peer-to-peer krävs det stor försiktighet när man hanterar synkronisering. Om två tillstånd skiljer sig åt mellan klienter, är det mycket svårt att återhämta systemet. Klient-server undviker dessa problem genom att hela tiden skicka hela tillståndet - om två klienter skulle ha två olika tillstånd spelar detta liten roll då nya, korrekta tillstånd ändå snart kommer att anlända.

Central processering: Att ha en central server ger fördelar när det kommer till felsökning - om ett fel uppstår är tanken att loggfiler kommer kunna användas för att hitta felets källa. Om man använder peer-to-peer, måste klienterna kommunicera loggfilerna på annat sätt.

Trots de ovanstående fördelarna, finns det nackdelar med klient-server som peer-to-peer inte är lika drabbad av.

Kostnad: Att upprätthålla en eller flera spelservrar kan ge en signifikant kostnad som peer-to-peer arkitekturen inte skulle drabbas av. Dessa kostnader involverar strömförbrukning, avgifter för t.ex servercenter, nätverksbelastning etc.

Nätverksbelastning: Klient-server i jämförelse med peer-to-peer skickar tillstånd respektive kommandon. Tillståndspaketen är generellt större än kommandon och därför har klient-server modellen högre nätverksbelastning.

Rättvis logik: Ett stort problem med klient-server arkitekturen är att alla klienter inte har ett gemensamt tillstånd. Detta kan orsaka problem när servern ska göra en bedömning av kollisioner. Spelare befinner sig i ett annat tillstånd än vad servern ser, och därför kan positionen spelaren siktar på innehålla en motståndare på klientsidan. På serversidan kan denna dock motståndaren redan ha flyttat på sig (vilket resulterar i en miss). Detta problem kan undvikas med peer-to-peer, då alla spelare ser samma tillstånd.

Slutsats: Det som är den fundamentala skillnaden i peer-to-peer jämfört med klient-server med en auktoritär server är säkerhet och nätverksbelastning. Då peer-to-peer ger den enskilde klienten makten att kunna ta beslut hindrar inget att klienten tar ogiltiga och fördelaktiga beslut för klienten ifråga. Även om peer-to-peer skickar fler paket, i regel en till varje klient, så är paketen oftast markant mindre än ett tillståndspaket som en server skulle skicka och där ligger skillnaden som gör att peer-to-peer har en lägre nätverksbelastning och ämnar sig väldigt bra till RTS-spel. Valet föll på klient-server, mycket på grund av dess simplicitet: ingen synkronisering behövs, och det är svårt att fuska även utan några åtgärder. Detta tillåter gruppen att tidigare fokusera på projektets

mål, istället för kringliggande komponenter.

4.2 Protokoll

Ett protokoll kan likställas med en överenskommelse mellan parter om hur något ska göras. I fallet för kommunikationsprotokoll finns ett behov att skicka information mellan datorer. Detta kräver att informationen är utformad och överförs på ett sätt som alla parter kan tolka och/eller vidarebefodra. Alla informationsöverföringar som följer protokollet kan tolkas och användas av de som förstår protokollet [32].

Det finns många olika protokoll designade med olika användningsområden i åtanke. I detta kapitel introduceras de två som användes för projektet, hur de användes, samt vilka konsekvenser de medförde.

4.2.1 Transmission Control Protocol (TCP)

Transmission Control Protocol, förkortat TCP, är ett protokoll för pålitlig dataöverföring över internet [33]. Protokollet är förbindelseorienterat, vilket innebär att en förbindelse mellan två värdar skapas och upprätthålls tills anslutningen bryts. TCP erbjuder vissa garantier för dataöverföringen; alla paket skall anlända och i rätt ordning till sin destination. Paket som bryter denna invariant skickas om, vilket tar tid. Därför lämpar sig TCP bäst åt de applikationer som prioriterar säker dataöverföring över realtidsöverföring.

4.2.2 User Datagram Protocol (UDP)

User Datagram Protocol, förkortat UDP, är ett protokoll för dataöverföring över internet [34]. Protokollet är, till skillnad från ovannämnda TCP, förbindelselöst och det finns inga garantier att informationen kommer fram.

Bristen på garantier ger protokollet den stora fördelen att informationen förs över snabbare än TCP, då det är mindre information som läggs till för att möjliggöra dessa garantier. Den främsta fördelen är dock att man inte väntar med att skicka resterande paket för att ett paket har tappats. Därför är UDP väldigt lämpligt för realtidsapplikationer såsom videosamtal och actionspel. Det viktigaste i dessa applikationer är att det senaste tillståndspaketet kommer fram så snabbt som möjligt, och om man tappar ett så blir det snabbt inaktuellt.

4.2.3 Implementering

Båda nätverksprotokollen används till olika delar i projektet. De aspekter som inte involverar realtidskommunikation sköts med hjälp av TCP, på grund av ovannämnda anledningar. Detta är t.ex. kontoskapning, inloggning och skapande av en match. För synkronisering av tillstånd används UDP. Om TCP skulle användas även för detta ändamål, skulle tillstånd kunna bli fördröjda i väntan på ett förlorat paket - något som skulle störa spelflödet [35]. Det är bättre att förlora ett tillstånd och kompensera för det, än att vänta på ett tillstånd som med all sannolikhet kommer vara för gammalt för att kunna användas.

4.2.4 Utvärdering

Att blanda TCP och UDP visade sig inte skapa några större problem rent implementationsmässigt, och båda protokollen fungerade utmärkt till de uppgifter de användes till. Man bör dock vara försiktig vid användandet av både TCP och UDP, eftersom TCP kan orsaka problem med UDP-kommunikationen [35] Vi fann detta inte vara något större problem, då de olika protokollen används vid olika skeden i spelet (TCP vid startskärmen, UDP i en match). Vid utökning av spelet är det dock troligt att det skulle märkas, då mer nätverksberoende funktionalitet implementeras.

4.3 Spelsäkerhet

Säkerhet inom spel är ett viktigt ämne, då klienter har möjlighet att läsa och modifiera den information som de delar med sig. Genom detta kan klienten skaffa sig övertag utanför spelets ramar och regler; med andra ord, fuska. Åtkomsten av data är en egenskap av de underliggande protokollen och kan inte göras någonting åt, men konsekvenserna av det kan mitigeras.

En klients inflytande över spelets gång, samt dess tillgång till information, påverkar möjligheten till fusk. När en klient har auktoritet att ta beslut för sig själv kan detta utnyttjas genom att skicka beslut som inte skett, exempelvis så skulle klienten kunna säga att en spelares skott träffade sin motståndare (även om så inte är fallet). Får klienten tillgång till för mycket information kan det utnytjas utanför spelets regler, genom att visa information spelaren inte borde ha tillgång till.

4.3.1 Implementering

Som nämts i kapitel 4.1.4 använder sig BoSS av nätverksstrukturen klient-server. Det medför två tydliga fördelar vad det gäller säkerhet, tack vare att informationen går ige-

nom en central punkt som klienterna inte har fri tillgång till.

För det första sker logikberäkningar på servern, vilket innebär att servern har den slutgiltiga bestämmanderätten och klienter enbart skickar dess spelares kommandon. Detta betyder att klienterna har så lågt inflytande över spelets gång som möjligt; inga beslut tas lokalt. För det andra kontrollerar servern vilken information varje klient får tillgång till. Således kan det centralt säkerställas att klienterna bara får information de är berättigade till.

4.3.2 Utvärdering

klient-server-arkitekturen i BoSS har gjort spelet motståndskraftigt mot fusk. Servern accepterar bara en begränsad mängd kommandon från klienterna, och eftersom servern har det korrekta tillståndet kan den kontrollera om kommandon den får är giltiga.

Det finns dock sätt att fuska. Bland annat kan en klient förfalska paket så att de ser ut att komma från en annan klient. Med detta kan servern fås att tro att den andra spelaren gör ett dåligt, men helt giltigt, drag. På det sättet kan en illvillig spelare förstöra för sina medspelare, och således underlätta för sig själv. Det är även möjligt att fånga upp serverns tillståndsuppdateringar och modifiera och/eller försena dem menade för ens medspelare. Ett annat sätt som alltid är möjligt är att hacka servern, men detta anser vi vara ett utomstående problem som vi valt att ignorera.

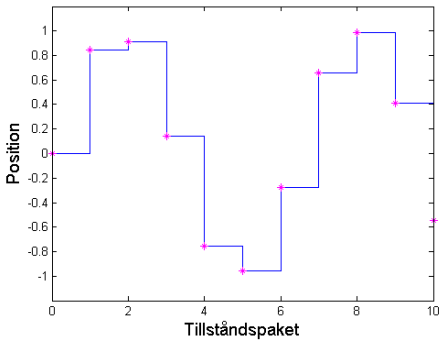
De tidigare fuskexemplen kräver dock kvalificerat arbete, vilket innefattar både att fånga upp specifika paket från nätverket och kolla på samt ändra deras innehåll. För att säkra mot detta krävs kryptering [36]. Eftersom all data måste krypteras och dekrypteras, bidrar detta med en signifikant minskning av överföringshastighet, vilket går emot syftet med projektet. Vi har därför beslutat att ignorera detta potentiella problem till fördel för snabbare överföringshastighet.

4.4 Spelflöde

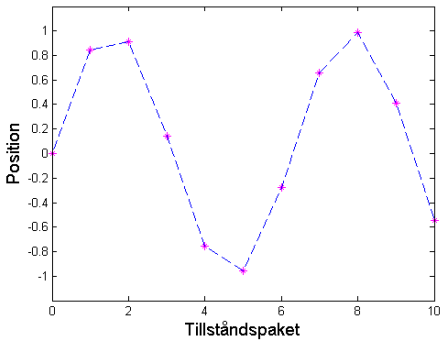
Det är kritiskt att skapa ett spelflöde som upplevs kontinuerligt och mjukt för spelaren. Eftersom man inte vill överbelasta nätverket genom att skicka för många tillståndspaket, kommer klienten att få nya tillståndsuppdateringar mycket mer sällan än bilduppdateringshastigheten. Detta skapar ett hackigt spelflöde, vilket måste motverkas.

4.4.1 Interpolering

Servern skickar spelets tillstånd till alla klienter var 50:e milisekund, vilket betyder att en klient får tjugo uppdateringar per sekund. Om klienten bara ritar en ny bild per tillståndsuppdatering så kan spelflödet bli hackigt. Detta kan lösas genom att alltid skicka så många bilder som klienten kräver för en hög bilduppdateringsfrekvens, men detta skulle öka nätverkstrafiken markant. En bättre lösning är att interpolera mellan tillstånden, vilket enkelt uttryckt betyder att man uppskattar bilder mellan tillståndsuppdateringarna. På detta sättet kan man ha olika tillståndsuppdateringsfrekvens och bilduppdateringsfrekvens. Detta kan relateras till en enkel diskret sinuskurva med få datapunkter som i figur 11. För att komma runt detta kan man använda interpolering [37]. För att få en bättre övergång mellan två tillståndsuppdateringar, räknar man ut alla spelobjekts interpolerade positioner baserat på tiden mellan tillstånden. Om interpolering skulle ske på den diskreta kurvan visad innan, skulle kurvan ta formen av figur 12. Det gäller då att se till att man har ett tillstånd att interpolera till, för att på så sätt alltid få mjuka positionsuppdateringar. Problemet är dock att man inte vet vad framtida tillstånd innehåller för data. Man kommer runt detta genom att fördröja renderingen av tillstånden tills dess att man har två tillstånd att interpolera mellan. Om servern skickar en tillståndsuppdatering var 50:e millisekund, kräver interpoleringen att man fördröjer renderingen minst 50 millisekunder. Detta gör att man alltid har två tillstånd att interpolera mellan.



Figur 11: Icke interpolerade tillståndsändringar



Figur 12: Interpolerade tillståndsändringar

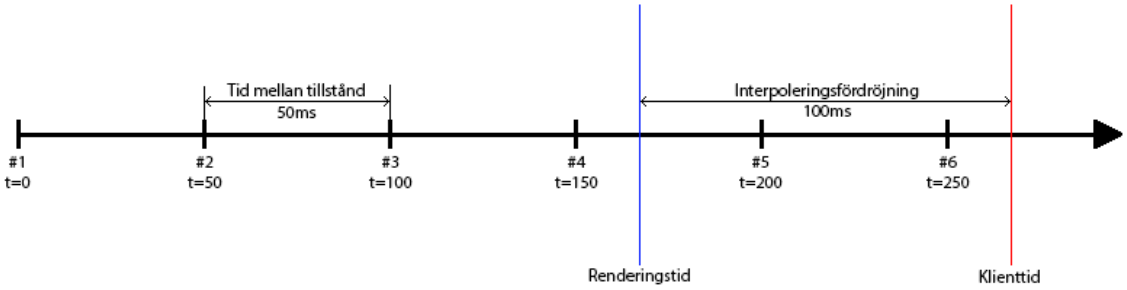
4.4.2 Implementering

Implementeringen av interpolering görs på klientsidan genom att räkna ut den s.k. renderingstiden. Denna tid är helt enkelt nuvarande tid minus interpoleringsfördröjningen. Därefter letar klienten igenom de inkomna tillstånden (vilka sparas i en lista) efter två tillstånd vars tidsstämplar är innan respektive efter renderingstiden. Sedan baseras den nuvarande positionen för alla spelobjekt på medelvärdet av den information som finns i

de två tillstånden.

4.4.3 Utvärdering

Som en följd av att nätverkskommunikationen använder UDP-protokollet, kommer alltid risken att tillståndspaket går förlorade på vägen finnas. För att öka sannolikheten att två tillstånd alltid finns innan renderingen sker på klientsidan, ökar man interpoleringsfördröjningen ytterligare. 100 millisekunder tillåter att ett paket går förlorat om servern skickar ut tillstånden en gång varje 50:e millisekund. Se figur 13, klienten interpolerar mellan tillstånd #4 och #5. Om tillstånd #5 går förlorat, kan klienten fortfarande interpolera mellan tillstånd #4 och #6, tack vare interpoleringsfördröjningen.



(Figur 13: Interpoleringsfördöjning)

Att ha för hög interpoleringstid skapar en negativ spelupplevelse, då spelaren upplever stora fördröjningar i spelflödet. En låg interpoleringstid orsakar hackigt spelflöde. Målet med implementeringen av interpolationsfördröjning är således att hålla en låg interpolationsfördröjning men samtidigt undvika hack i spelflödet till följd av paketförluster. Projektgruppen finner att 100 millisekund räcker för 10-20% paketförlust utan att orsaka för hög fördröjning.

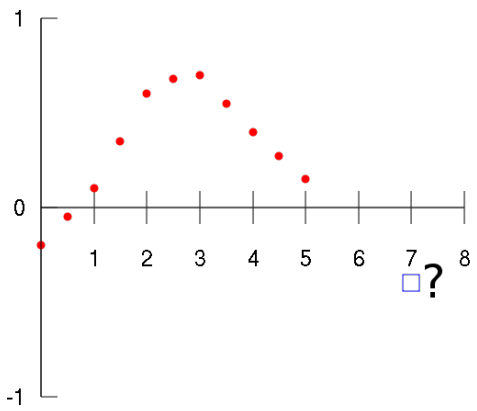
4.5 Paketförlust

Eftersom UDP-protokollet inte erbjuder någon form av garanti att ett paket kommer fram till mottagaren, behöver klienterna ha möjligheten att kompensera för detta.

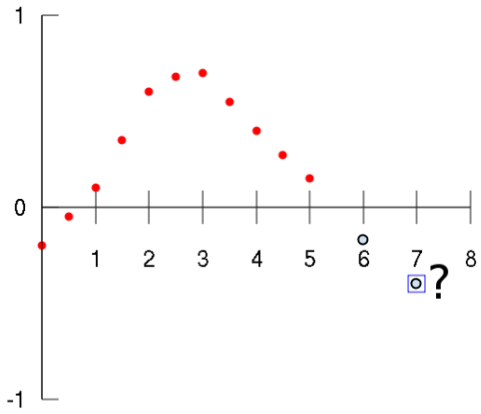
4.5.1 Extrapolering

Extrapolering [37] har stor koppling till interpolering. Om ett eller flera tillståndspaket går förlorade, har klienten inget tillstånd att interpolera till. I det föregående exemplet skulle till exempel tillstånd #5 och #6 kunna försvinna. Då blir alla objekt frysta i de

positioner som beskrivs av tillstånd #4, fram tills dess att ett nytt tillstånd kommer, se figur 14. Detta går att lösa med en interpoleringsfördröjning på 150 millisekunder, men för att undvika onödiga fördröjningar kan man istället använda s.k. extrapolering. Detta går ut på att analysera de senaste tillstånden och göra beräkningar på hur spelobjekten borde röra sig baserat på tidigare ändringar i deras positioner, se figur 15. Under korta tidsperioder är denna metod effektiv, då sannolikheten att ett spelobjekts aktuella rörelse är oförändrad jämfört med tidigare tillstånd är högre ju lägre tidsintervall det handlar om. Ju längre tidsintervallet mellan de två senaste tillstånden blir, desto mer osäker är effekten av extrapolering.



Figur 14: Tillstånd går förlorat [38]



Figur 15: Nya datapunkter skapas utifrån tidigare punkter

Vid extrapolering används såkallad Dead Reckoning[1] för att extrapolera positioner som klienten ännu inte tagit emot information för. Dead Reckoning är en teknik som använder ett objekts position, kurs och hastighet för att förutspå framtida positioner och kan användas för extrapolering genom att kursen och hastigheten uppskattas från input för att uppskatta positionsförändringen som borde ha skett på servern men som servern inte hunnit reagera på genom att skicka ut synkpaket.

4.5.2 Pålitlig UDP-överföring

Pålitlig UDP-överföring är en teknik som kan användas för att garantera att ett viktigt meddelande når till sin slutdestination. För att uppnå denna funktionalitet hämtar man inspiration från TCP och skickar ett bekräftelsepaket på UDP-paketen varje gång ett sådant meddelande tas emot. Istället för att använda TCP:s räddning av paket, skickas

[1] att använda ett objekts position, kurs och fart för att att extrapolera framtida positioner för objektet vid olika tidpunkter

meddelandet med jämna intervall tills en bekräftelse fåtts och paketet tas bort från listan. På grund av att paketen kan fastna i en loop, är detta en teknik som enbart borde användas vid viktiga sändningar av meddelanden som har hög prioritet.

4.5.3 Implementering

Extrapolering implementerades i klienten genom att kolla ifall renderingstiden ligger mellan två mottagna tillståndstidsstämplar. Om det inte finns (p.g.a. paketförlust), körs extrapoleringskoden. Denna kod räknar ut vilken hastighet och rikting objektet rör sig i baserat på de senaste tillstånden. Sedan förflyttas objektet korrekt avstånd enligt denna hastighet och riktning. Klienten antar att objektet färdas i rak riktning i samma hastighet. Denna förutsägelse kan förbättras avsevärt om man tar hänsyn till ännu tidigare tillstånd. Rörelsen kan t.ex. vara cirkulär, och i de fallen kommer extrapoleringen vara lite fel.

Pålitlig UDP-överföring är implementerat så att servern har ett fält per spelare i vilken de pålitliga medelanderna placeras. Det första meddelandet i fältet kommer att skickas ut med jämna intervall, tills dess att den fått en bekräftelse eller att 100 medelanden har skickats. Ifall en bekräftelse mottogs så tas meddelandet bort från listan och nästa medelande börjar skickas; ifall de 100 meddelandet däremot skickas utan bekräftelse så anses den mottagande parten vara frånkopplad. Meddelandet skiljs genom att en byte är reserverad efter sekvensnumret som avgör hurvida det är pålitligt eller inte pålitligt.

4.5.4 Utvärdering

Projektgruppen experimenterade med olika inställningar gällande interpolering i kombination med extrapolering. Vid en interpoleringstid på 100 millisekunder och en simulerad paketförlust på 25% märker man tydliga hack i spelet om extrapolering är avslaget. Om man slår på extrapolering, blir spelflödet genast bättre och små hack sker endast vid snabba förflyttningar i motsatt riktning. Klienten använder sig av s.k. Dead Reckoning [39] för att extrapolera positioner, men mer avancerade metoder kan användas för att få en bättre förutsägelse av icke-linjära rörelser.

Vad det gäller pålitlig UDP-överföring så duger systemet för sitt syfte, men det är värt att nämna att effektiviteten kan förbättras. Exempelvis kan paket skickas kontinuerligt utan någon begränsning av fönsterstorlek. En fördel är att ingen TCP-koppling blandas in för det kan orsaka att UDP paketen förloras [40].

4.6 Fördröjning

Fördröjning är ett svårt problem då man alltid jobbar i fel tempus, på klienten när man utför ett kommande kommer de exekveras i framtiden på serverns sida och då kan alla vilkor ha ändrats under tiden. Servern bör helst jobba i det förflutna, för när ett kommando anländer så är önskan att det redan ska ha exekverats. Projektgruppen vill vara tydlig att inga av dessa metoder har implementerats i BoSS.

4.6.1 Tillbakaspolning

Tillbakaspolning eller *Rewind* är en nätverkshanteringsteknik som används inom spelindustrin för att dölja de konsekvenser av fördröjning som uppstår på klienten. Detta används i spel som Half-Life och Team Fortress enligt [27]. För att implementera tillbakaspolning så måste servern spara en historik över speltillstånd. När servern tar emot kommandon så appliceras de på ett äldre tillstånd istället för det senaste, följt av att de efterföljande tillstånden räknas om.

Tillståndet som kommandot appliceras på ska motsvara tillståndet som ritades upp på klienten då kommandot ifråga rekvirerades. Detta kan identifieras genom att kalkylera storleken på fördröjningen samt interpoleringstiden.

Tillbakaspolning används främst av spel inom FPS (First-person shooter)-genren. Dessa spel bygger vanligtvis på att man snabbt måste sikta på och träffa rörliga mål, vilket ställer höga krav på spelarnas reaktionsförmåga. Ifall tillbakaspolning inte används kommer därför avvikelserna som uppstår mellan server och klient, till följd av fördröjning att medföra stor risk för förödande konsekvenser för spelupplevelsen. Vid höga fördröjningar kan ett skott som ser ut att träffa ett objekt på en klient missa med stor marginal på servern då objektet hunnit flytta på sig. Detta ger en fördel för spelare med låg fördröjning då deras aktuella tillstånds felmarginal är mindre respektive någon med hög fördröjning jämfört med serverns senaste tillstånd. Syftet med tillbakaspolning är att neutralisera denna fördel och därmed ombesörja en mer rättvis spelupplevelse.

Användning av tillbakaspolning medför oundvikligen ett antal sidoeffekter. Exempel på sidoeffekt är att en spelare kan bli träffad av en attack trots att spelaren inte såg ut att befinna sig i en position där detta var möjligt på den lokala klienten. Detta sker då den andra spelaren utfört attacken när den befunnit sig i ett tillstånd där träffen är möjlig. En annan sidoeffekt är en viss susceptibilitet för fusk.

4.6.2 Input prediction

Den förväntade responsen vid ett musklick är att den lokala spelaren ska börja röra sig mot den valda destinationen. När man spelar med hög nätverksfördröjning kan det dock vara irriterande att responsen från spelarens karaktär inte är omedelbar. Detta sker eftersom servern måste ta emot spelarens inmatning, utföra beräkningar av spellogik, för att till sist skicka tillbaka det nya tillståndet till spelaren.

För att undvika detta problem, kan en metod kallad *input prediction* (ibland *client-side prediction*) användas. Denna metod bygger på att klienten ska exekvera samma kod som servern gör, i ett försök att förutspå hur karaktären kommer förflytta sig [25, 27, 37]. För att detta ska fungera krävs det vissa förutsättningar. För det första måste klienten och servern ha tillgång till samma kod; för det andra får inte fördröjningen vara för hög.

Vid input prediction kommer klienten att agera utifrån sin egen förutsägelse och flytta karaktären lokalt till dess nya postion. Därefter jämförs positionen med den korrekta positionen som erhålls från servern efter en kort stund - om förutsägelsen var korrekt händer ingenting. Det händer dock att positionen är felaktig till följd av påverkan av externa objekt. Då korrigeras positionen genom att antingen ändra den rakt av, eller interpolera ändringen över en tid för att dölja förutsägelsen avvikelse.

4.6.3 Implementering

En implementation av tillbakaspolning har förberetts genom att en historik med speltillstånd och spelkommandon skapats som under spelets gång konstant fylls på samt att spellogiken modulariserat så att det finns en funktion som tar ett tillstånd och kommandon som parametrar och räknar ut nästa, det tillåter oss att skicka ett kommando i ett äldre speltillstånd och se hur det skulle påverka nutidens tillstånd. Att räkna om förgångna tillstånd då ett kommando som bör bakåtdateras anländer har inte implementerats då detta visade sig ha signifikant komplexitet i BoSS. I FPS-genren, där tillbakaspolning normalt används, är attacker vanligtvis snabba eller rent av ögonblickliga. Till skillnad från detta är de projektiler som implementerats i BoSS i allmänhet objekt som rör sig enligt fysikaliska lagar och relativt långsamt. Bakåtstötar är i BoSS minst lika viktiga som skada, och en enda trollformelskollision kan orsaka en mycket stor bakåtstöt. Detta leder till att en spelare plötsligt kan ryckas till en helt annan position om en kollision bakåtdateras signifikant utan att spelaren har möjlighet att reagera på kollisionen under de mellanliggande tillstånden.

En annan svårighet är att räkna ut det korrekta tillståndet att applicera inkommande kommandon på. Ifall bakåtdateringen helt enkelt räknas ut ifrån skillnaden mellan kommandots och serverns tidsstämplar och interpoleringstiden så öppnas dörren för fusk.

Enligt [41] är ett möjligt fusk för denna typ av fördröjningskompensation att tidsstämplar manipuleras för framåtblickande fusk. Detta gör att spelaren får mer tid på sig att reagera på händelser vid kritiska spelmoment genom att dess kommandon artificiellt ges förlegade tidsstämplar. Protokoll för att detektera och motverka detta fusk genom buffring av orättvisa kommandon beskrivs också i [41] men dessa protokoll är svårimplementerade och medför att spelare som upplever en tillfällig skarp fördröjningsökning kan falskeligen identifieras som fuskare och därmed missgynnas.

Mer avancerad input prediction är svår att implementera då det inte går att förutsäga mycket mer än att spelkaraktären kommer röra sig i de mönster spelaren klickar i.

4.6.4 Utvärdering

Tillbakaspolning har stora brister i de fall klienterna har signifikant varierande fördröjning, och en potentiell lösning på detta skulle kunna vara att i automatiska spelförmedlingssystem matcha klienter med liknande fördröjningsmagnitud.

För spelare med hög latens kan spelet bli mycket mer responsivt vilket gör det lättare att sikta på spelobjekt. Det skulle vara intressant att utforska detta vidare, speciellt i en miljö där de handlingar som ska tillbakaspolas sker ögonblickligen och påverkar spelet på få sätt, exempelvis FPS-spel där handlingen att skjuta som sker ögonblickligen och har bara två utfall, antingen träffa och skada eller missa. Även input prediction kan förbättra spelflödet mycket. En fungerande implementation av detta hade varit önskvärt för att kunna utföra en gedigen utvärdering.

4.7 Informationsöverföring

4.7.1 Riktningar

Ett spelobjekts riktning kan beräknas av spelklienten genom vektorsubtraktion mellan objektets föregående och nuvarande position. Ett problem uppstår dock när spelkaraktären förflyttas åt ett håll, men springer åt ett annat. Detta uppkommer i fallet då spelkaraktären flyger åt ett håll som följd av en kollision med en trollformel.

För att klienten ständigt ska vara uppdaterad om vilken riktning alla spelkaraktärer springer åt, måste servern bifoga en extra variabel i synkroniseringspaketet. Denna variabel kan i det enklaste fallet vara en enhetsvektor som pekar åt denna riktning. En vektor består dock av två variabler (X och Y), vilket ökar storleken på paketet med 8

bytes per spelare om man använder vanliga flyttal som representation för koordinater. Ett bättre alternativ är att skicka med rotationsvinkeln istället, vilket bara tar hälften så stor plats.

4.7.2 Animationer

För att spelet ska kunna renderas på ett realistiskt och trovärdigt sätt, krävs det att klienten vet om vilka animationer som skall spelas upp och när. BoSS är ett förhållandevis enkelt spel, och därför krävs det endast en byte som representerar vilket tillstånd varje spelkaraktär har. En bit representerar gång (för gå-animationen), en annan kastning av trollformler (kast-animationen). Det finns utrymme att expandera då många bitar är oanvända.

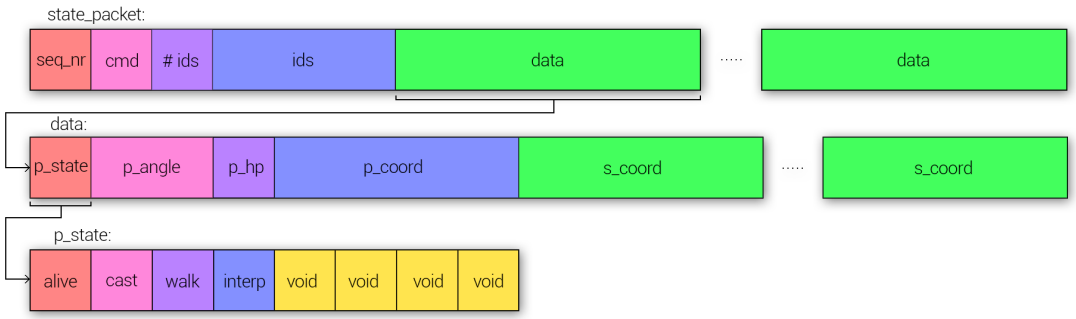
4.7.3 Struktur av nätverkspaket

För att erhålla en så liten storlek på nätverkspaketen som möjligt, måste all information packas tätt på bitnivå. Denna packning måste ske manuellt för att garantera en struktur som är lättviktig och väldefinierad. Inbyggd serialisering av objekt kan därför inte effektivt användas för realtidsöverföring, då dessa medför mycket extra information. Dessutom kräver detta att versionen av det serialiserade objektet stämmer överens med klientens version.

4.7.4 Implementering

Vi miniminerade paketstorleken bland annat genom att kasta om alla variabler som behövde skickas över nätverket till den minsta möjliga datatypen utan en oacceptabelt stor förlust av precision. Bland annat så är spelkaraktärens hälsopoäng ett flyttal (fyra bytes) på serverns sida men skickas över som en byte då den på klienten representeras som ett heltal 0-130 trots att den internt på servern är ett decimaltal.

En ytterligare optimering är att endast aktiva och dynamiska spelobjekt skickas i spelobjekten. Om en spelkaraktärs hälsopoäng är noll och därmed inte finns med på spelplanen så skickas inte data för det objektet till klienterna. Data för trollformler som inte är aktiva på spelplanen för tillfället och därför inte heller ska synas skickas inte heller. Se figur 16.



(Figur 16: Tillståndspaketstruktur)

seq_nr: Sekvensnummer på paketet, används för att kunna jämföra vilken ordning tillstånden har. Användbart då UDP-paketen normalt inte stödjer denna funktion.
cmd: Kommando, används för att identifiera att detta paket är ett tillståndspaket.
#ids: Antalet dynamiska och aktiva spelobjekt (antalet aktiva spelare + antalet aktiva trollformler).
ids: Identifikationsnummer för ett spelobjekten.
data: Information om en spelare samt tillhörande trollformler.

- **p_state**: En byte - varje bit representerar spelarens tillstånd
  - **alive:** 1 om spelaren kan styras, annars 0.
  - **cast:** 1 om spelaren håller på att kasta en trollformel, annars 0.
  - **walk:** 1 om spelaren går, annars 0.
  - **interp:** 1 om spelaren ska interpoleras, annars 0.
  - **void:** Odefinierat tills vidare.
- **p_angle:** Vinkeln spelaren tittar mot.
- **p_hp:** Spelarens liv.
- **p_coord:** Spelarens koordinater (X+Y).
- **s_coord:** Koordinat på en trollformel (X+Y).

4.7.5 Utvärdering

De begränsade fälten som inkluderats i nätverkspaketen ger klienten precis tillräckligt med information för att möjliggöra en snygg uppritning av spelobjekten. Klienten gör lokala beräkningar för att få fram den bristande informationen, så som vilken bildruta i animationen som ska spelas. Unity ger mycket av detta gratis.

Risken med klient-server är att synkroniseringspaketen som skickas ut blir för stora. Detta är något som måste undersökas för att kunna garantera en bra spelupplevelse även

för de spelare som inte har en snabb internetleverantör.

För att garantera att nätverksbelastningen inte blir för stor för varken servern eller klienter, måste vi designa ett scenario som motsvarar det värsta fallet för en match, och räkna ut belastningen för båda parter.

- I matchen finns åtta spelare
- Varje spelare har fyra aktiva trollformler på banan
- Varje spelare exekverar 5 kommandon varje sekund

Vi sätter även en maximal gräns för nätverksbelastningen på serversidan till 10 kB/s nerladdning och 100 kB/s uppladdning, samt 10 kB/s nerladdning och 1 kb/s uppladdning på klientsidan.

Följande formel kan användas för att ge oss den totala storleken på ett tillståndspaket:

$$F(p,s) = 3 + p + s + 14 \cdot p + 8 \cdot s = 15 \cdot p + 9 \cdot s + 3$$

där p är antalet spelare, s är antalet trollformler och F(p,s) är antalet bytes på tillståndspaketet. För vårt scenario får vi 411 bytes för ett enda paket. Eftersom detta paket skickas 20 gånger per sekund, blir den totala nätverksbelastningen (nerladdning) för klienten:

$$20 \cdot F(8, 4 \cdot 8) = 8220 \approx 8kB/s$$

Nätverksbelastningen (uppladdning) för servern blir

$$20 \cdot \cdot 8 \cdot F(8, 4 \cdot 8) = 65760 \approx 66kB/s$$

eftersom servern skickar till åtta spelare. Nätverksbelastningen för klienten (uppladdning) blir:

$$5 \cdot 10 = 50 = 0.05kB/s$$

och serverns nätverksbelastning (nedladdning):

$$5 \cdot 10 \cdot 8 = 400 = 0.4kB/s$$

Resultatet av beräkningarna är att servern har en nätverksbelastning av 0.4 kB/s nerladdning och 66 kB/s uppladdning för server, samt 8 kB/s nerladdning och 0.05 kB/s uppladdning för klient. Detta uppfyller de specifierade kraven i scenariot med god marginal.

Den nuvarande modellen går ut på att varje uppdateringspaket innehåller data om alla entiteter. För att minska storleken kan Delta Encoding användas [42]. Detta innebär att man istället för att skicka ett helt nytt tillstånd endast skickar skillnaden mellan det gamla och det nya tillståndet. Detta är allt som klienten behöver för att rita upp det nya tillståndet eftersom den har tillgång till det gamla. Denna metod har implementerats framgångsrikt av bland annat Quake III [43]. För att kunna skicka det som ändrats behöver servern känna till vad klienten vet, vilket innebär att klienterna måste bekräfta vad de fått. Således, om den mesta datan ändrats kontinuerligt så kan den här metoden öka nätverksbelastningen.

5

Slutsats

5.1 Resultat

Vi fann att de huvudsakliga problemen med kommunikation över nätverket - det vill säga fördröjningar och paketförlust - går mycket väl att kompensera för. Denna kompensation kommer dock ofta med en kostnad inom andra områden. För att dölja fördröjningar måste en del av spellogikkoden köras på klientsidan för att kunna göra kvalificerade gissningar angående objektens positioner. De viktigaste spelobjekten som måste fördröjningskompenseras är de som spelaren har direkt kontroll över, det vill säga själva spelkaraktären. De övriga objekt som är fördröjda påverkar inte spelaren direkt - undantaget är när kollisionsdetektering ska ske. För att öka sannolikheten att en träff på klientsidan kommer resultera i en träff även på serversidan, kan tillbakaspolning användas för att spola tillbaka tillstånden till den punkt där kollisionen borde skett. Användningen av denna metod kan dock öppna upp möjligheter för fusk, då servern blir beroende av individuella klienters tillstånd.

Om ett tillstånd förloras, kan detta ofta kompenseras för genom extrapolering, vilket innebär att erhålla de senaste tillstånden och räkna ut vart spelobjektet borde stå i det förlorade tillstånd. Som ytterligare hjälp mot detta problem kan interpolering användas, vilket fördröjer renderingen av speltillstånden så klienten alltid har ett par tillstånd att arbeta med. Paketförluster är ett större problem för spelarens kommandon som måste ha en garanti att komma fram till servern.

Server-klient är arkitekturen som projektgruppen bedömde vara enklast att implementera, och eftersom modellen bringade tillräckligt med funktionalitet valdes den till BoSS. Alternativet peer-to-peer passar bättre till spel med fler enheter, t.ex. strategispel så som Warcraft III.

5.2 Diskussion

Eftersom Unitymotorn användes för utvecklingen kunde vi enkelt skapa renderingen på klientsidan. Detta medförde dock en liten oförutsedd nackdel, då det blev problematiskt att dela spellogikskod. Unity använder speciella spelobjekt som måste ärva från deras egna klasser, vilket inte kan göras på serversidan. Ett alternativ hade varit att designa en egen spelmotor med hjälp av ett spelramverk. Kodbasen hade då blivit mindre rörig, men det var trevligt att kunna fokusera på nätverkskoden.

När man skapar spel är det bra att skriva mycket modulärt då mycket kod kan användas på andra ställen med små modifieringar - t.ex pålitlig överföring och protokollet för att serialisera och deserialisera UDP paket. Vår icke-kompletta implementation av tillbakaspolning utnyttjar en funktion som tar ett tillstånd och en lista av kommandon och ger oss ett nytt; denna kod hade kunnat användas på klientsidan för att implementera input prediction. Allmänt kan sägas att projektet skulle haft stor fördel av en noggrannare planering kring kodstrukturen, vilket hade lett till implementation och utvärdering av fördröjningshantering - något som saknas i dagsläget.

Både interpolering och extrapolering bidrar till en mjukare spelupplevelse och är nödvändigt då servern inte skickar ett tillståndspaket för varje bildruta. Interpolering är att föredra i förstahand om nätverkskopplingen är stabil och latenstiden låg, då metoden erbjuder en mjukare spelupplevelse i utbyte mot att alla visualiseringar sker aningen fördröjt. Extrapolering skapar en bra spelupplevelse genom att visualisera hur ett förlorat tillstånd borde se ut. Om den förutspår korrekt döljs paketförlusten, om den förutspår fel bör karaktärens position mjukt dras till sin korrekta. Felaktiga förutsägelser är omöjliga att undvika helt och ju längre man extrapolerar från det senaste korrekta tillståndet desto större kan felet bli. Detta reflekteras dåligt i spelet. Rekommendationen är att använda både interpolering och extrapolering, med en så hög interpoleringsfördröjning som möjligt utan att det blir irriterande, och ett extrapoleringsintervall som ger tillräcklig precision utan visuella felaktigheter.

Vidare kan projektgruppen konstatera att det varit enklare om all nätverkskommunikation implementerades via UDP-protokollet (med funktionalitet för leveransgaranti). Nuvarande system fungerar, men att kunna utöka och generalisera nätverkssystemet hade underlättat både tidigare och eventuella framtida arbeten.

5.3 Framtid

Det finns många koncept i denna rapport som inte blivit helt utforskade. Att implementera tillbakaspolning och undersöka metodens för- och nackdelar är något som inte

utförts, trots att det är ett potentiellt viktigt koncept i många spel. Mer advancerade algoritmer för extrapolering och input prediction hade även varit mycket intressant att studera för att kunna tolerera fler paketförluster. Vi har även läst om att använda delta encoding för att minska nätverksbelastningen hos server-klient arkitekturen, något som projektet skulle behöva.

Litteraturförteckning

[1] J. A. Jacko, A. Sears, and M. S. Borella, "The effect of network delay and media on user perceptions of web resources," [Besökt 27 april 2013]. [Online]. Tillgänglig: http://www.tandfonline.com/doi/abs/10.1080/01449290750052688#preview

[2] Blizzard Entertainment, "Warcraft III," [Besökt 30 april 2013]. [Online]. Tillgänglig: http://eu.blizzard.com/en-gb/games/war3/

[3] L. Cieniawa, "The Armchair Empire - PC Reviews: WarCraft III: Reign of Chaos," Aug. 2002, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf

[4] Zymoran, "Warlock 099c," Jul. 2008, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.hiveworkshop.com/forums/maps-564/warlock-099c-92687/

[5] J. Smed, T. Kaukoranta, and H. Hakonen, "Aspects of Networking in Multiplayer Computer Games," [Besökt 6 juni 2013]. [Online]. Tillgänglig: http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/net-aspects.pdf

[6] V. D. Bhamidipati and S. Kilari, "Effect of Delay/ Delay Variable on QoE in Video Streaming," May 2010, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.bth.se/com/mscee.nsf/attachments/5297_Effect_of_Delay_Delay_Variation_on_QoE_in_Video_Streaming_pdf/$file/5297_Effect_of_Delay_Delay_Variation_on_QoE_in_Video_Streaming.pdf

[7] Kickstarter, "Torment: Tides of Numenera," [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.kickstarter.com/projects/inxile/torment-tides-of-numenera

[8] "Unraveling the unending oink," Aug. 2012, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://unity3d.com/gallery/made-with-unity/profiles/rovio-badpiggies

[9] R. McMillan, "After controversy, Torvalds begins work on git," Apr. 2005, [Besökt 20 maj 2013]. [Online]. Tillgänglig: http://www.pcworld.idg.com.au/article/129776/after_controversy_torvalds_begins_work_git_/

[10] J. Osborn, "Deep Inside C: An Interview with Microsoft Chief Architect Anders Hejlsberg," Aug. 2000, [Besökt 7 juni 2013]. [Online]. Tillgänglig: http://www.windowsdevcenter.com/pub/a/oreilly/windows/news/hejlsberg_0800.html

[11] "Introducing JSON," [Besökt 2 maj 2013]. [Online]. Tillgänglig: http://www.json.org/

[12] N. Nurseitov, M. Paulson, R. Reynolds, and C. Izurieta, "Comparison of JSON and XML Data Interchange Formats: A Case Study," [Besökt 25 april 2013]. [Online]. Tillgänglig: http://www.cs.montana.edu/izurieta/pubs/caine2009.pdf

[13] W. Kramer, "What is a Game?" [Besökt 7 juni 2013]. [Online]. Tillgänglig: http://www.thegamesjournal.com/articles/WhatIsaGame.shtml

[14] T. Karras, "Thinking Parallel, Part I: Collision Detection on the GPU," Nov. 2012, [Besökt 20 maj 2013]. [Online]. Tillgänglig: https://developer.nvidia.com/content/thinking-parallel-part-i-collision-detection-gpu

[15] T. Akenine-Möller, E. Haines, and N. Hoffman, *Real-Time Rendering*, 3rd ed. AK Peters Ltd, [Besökt 18 maj 2013].

[16] Wikipedia, "2.5D," [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://en.wikipedia.org/wiki/2.5D

[17] A. Poon, "Eldklot," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[18] ——, "Målsökande Missil," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[19] ——, "Teleportering," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[20] ——, "Sköld," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[21] ——, "Blixt," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[22] ——, "Plåga," 2013, [Skapad för kandidatarbetet, till kandidatarbetet].

[23] "computer," Jan. 2013, [Besökt 6 Juni 2013]. [Online]. Tillgänglig: http://clipart-finder.com/clip?clip=21421

[24] "server mimooh 01r," Jan. 2013, [Besökt 6 Juni 2013]. [Online]. Tillgänglig: http://clipart-finder.com/clip?clip=29848

[25] G. Fiedler, "What every programmer needs to know about game networking," [Besökt 26 april 2013]. [Online]. Tillgänglig: http://gafferongames.com/networking-for-game-programmers/what-every-programmer-needs-to-know-about-game-networking/

[26] Epic Games, Inc, "Client Server Model," [Besökt 25 april 2013]. [Online]. Tillgänglig: http://udn.epicgames.com/Three/ClientServerModel.html

[27] Y. W. Bernier, "Latency Compensating Methods in Client/Server In-game Protocol Design and Optimization," [Besökt 25 april 2013]. [Online]. Tillgänglig: http://web.cs.wpi.edu/~claypool/courses/4513-B03/papers/games/bernier.pdf

[28] A. Armstrong, "The Mammoth Dedicated Server Guidebook," 2009, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.mammothmedia.com.au/~/media/92E27A0F97F34A958D6D4B9F3CD2A64F.ashx

[29] Microsoft, "Peer-to-Peer Topology," [Besökt 25 april 2013]. [Online]. Tillgänglig: http://msdn.microsoft.com/en-us/library/windows/desktop/bb153250%28v=vs.85%29.aspx

[30] G. Obiltschnig, "Cross-Platform Issues With Floating-Point Arithmetics in C++," 2006, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.appinf.com/download/FPIssues.pdf

[31] P. Wyatt, "The making of Warcraft part 3," [Besökt 2 maj 2013]. [Online]. Tillgänglig: http://www.codeofhonor.com/blog/the-making-of-warcraft-part-3

[32] E. Britannica, "protocol," [Besökt 7 juni 2013]. [Online]. Tillgänglig: http://global.britannica.com/EBchecked/topic/410357/protocol

[33] Information Sciences Institute, "TRANSMISSION CONTROL PROTOCOL - DARPA INTERNET PROGRAM PROTOCOL SPECIFICATION," Sep. 1981, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://tools.ietf.org/html/rfc793

[34] J. Postel, "User Datagram Protocol," Aug. 1980, [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://tools.ietf.org/html/rfc768

[35] G. Fiedler, "UDP vs. TCP," Oct. 2008, [Besökt 30 april 2013]. [Online]. Tillgänglig: http://gafferongames.com/networking-for-game-programmers/udp-vs-tcp/

[36] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 6th ed. Pearson Edu, 5 2012, [Besökt 16 maj 2013].

[37] Valve Developer Community, "Source Multiplayer Networking," [Besökt 30 april 2013]. [Online]. Tillgänglig: https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking

[38] "Extrapolation example," [Besökt 7 Juni 2013]. [Online]. Tillgänglig: http://upload.wikimedia.org/wikipedia/commons/8/8f/Extrapolation_example.svg

[39] L. Pantel and L. C. Wolf, "On the suitability of dead reckoning schemes for games," 2002, [Besökt 12 maj 2013]. [Online]. Tillgänglig: http://dl.acm.org/citation.cfm?id=566500.566512

[40] H. Sawashima, Y. Hori, and H. Sunahara, "Characteristics of UDP Packet Loss: Effect of TCP Traffic," [Besökt 18 maj 2013]. [Online]. Tillgänglig: http://www.isoc.org/INET97/proceedings/F3/F3_1.HTM

[41] E. Cronin, B. Filstrup, and S. Jamin, "Cheat-Proong Dead Reckoned Multiplayer Games (Extended Abstract)," 2003, [Besökt 26 april 2013]. [Online]. Tillgänglig: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.124.1611&rep=rep1&type=pdf

[42] N. Samteladze and K. Christensen, "DELTA: Delta Encoding for Less Traffic for Apps," [Besökt 13 maj 2013]. [Online]. Tillgänglig: http://www.csee.usf.edu/~nsamteladze/res/projects/research/delta/lcn-paper.pdf

[43] F. Sanglard, "QUAKE 3 SOURCE CODE REVIEW: NETWORK MODEL (PART 3 OF 5)," Jun. 2012, [Besökt 27 april 2013]. [Online]. Tillgänglig: http://fabiensanglard.net/quake3/network.php

A

Ordlista

Arenafas: Fasen då spelarna försöker besegra varandra genom att knuffa ut varandra i lavan för att minska de andras hälsopoäng till 0.
Bakåtstöt: En effekt en spelkaraktär drabbas av när den kolliderat med en trollformel. Magnituden på bakåtstöten bestäms både av ett värde i trollformeln samt ett värde för bakåtstötssusceptibilitet i spelkaraktären som ökar för varje gång den skadas. Spelkaraktärens värde återställs varje ny runda.
Bekräftelsepaket: Ett paket som skickas för att bekräfta att ett annat paket mottagits.
BoSS: Battle of Stupid Sorcerers.
Bugg: Fel i datorprogrammet.
Client-side prediction: Se Input prediction.
Cutscene: En kort video i spel som används för att utveckla berättelsen mer.
Dead Reckoning: Att använda ett objekts position, kurs och fart för att att extrapolera framtida positioner för objektet vid olika tidpunkter.
Fjärilseffekten: En minimal ändring i ett system ger stor påverkan någon annanstans.
Flyttal: Ett decimaltal i datorn.
Handelsfas: Fasen då spelarna inhandlar trollformler, denna sker innan arenafasen.
Hälsopoäng: Hälsopoäng, dessa motsvarar hälsan spelarens karaktär har och när denna siffran når till noll så är spelaren besegrad. Standardvärdet på det maximala antalet hälsopoäng är 100.
Hälsoåterhämtning: Ökning av hälsa som sker varje tick som spelkaraktären har en hälsopoäng lägre än det maximala antalet hälsa.
Indieutvecklare: En eller flera utvecklare utan finansiellt stöd av publicent.
Input prediction: Nätverkshanteringsteknik som ökar responsitiviteten. Klienten försöker förutspå konsekvenserna av spelarens kommando och ritar upp dem lokalt innan servern kommer med specifikationer hur bilden ska ritas.
JSON: JavaScript Object Notation, en datastruktur.
Klient: Mjukvara som ansluter sig till en spelserver.

Listen server: Servermjukvara som körs på klientens hårdvara.
Lobbyfasen: Innan själva spelet har börjat, nya spelare kan i denna fas ansluta till lobbyn så att de kan delta i spelet.
MOBA: *Multiplayer Online Battle Arena*.
Områdestrollformel: En trollformel vars effekt är i ett specifikt område och som därmed kan påverka flera spelkaraktärer samtidigt. Försvinner inte för att den kolliderar med något annat objekt till skillnad från projektiler.
Projektil: En trollformel som färdas som en projektil och som försvinner då den kolliderar med en spelkaraktär som inte kontrollerar projektilen eller en annan projektil som inte kontrolleras av samma spelkaraktär som kastade projektilen i fråga.
Realtidsapplikation: En applikation som kommunicerar med låg fördröjning.
Rendering: Uppritning av bild.
RPG: *Role-Playing Game*, en spelgenre där spelaren styr en eller flera karaktärer vars förmågor utökas och förbättras under spelets gång baserat på spelarens val.
RTS: *Real Time Strategy*, en genre där man har ett perspektiv ovanifrån och man styr armer i realtid.
Serialisering: En process där ett objekt konverteras till ett format som kan skickas över nätverket.
Server: Datorsystem som betjänar andra system.
Slutfasen: Då spelet är slut.
TCP: Transmission Control Protocol.
Tredjepartsutvecklare: Utvecklare som har inga förbindelser till grund tillverkaren.
UDP: User Datagram Protocol.
Väntetid: En tid efter att en trollformel kastats under vilken samma trollformel inte kan kastas igen.

B

Arbetsfördelning

Projektets medlemmar fördelades initialt i två grupper. Den ena gruppen skulle fokusera på spelprogrammeringen, det vill säga själva spelmotorn och logiken till spelet. Gruppen bestod av:

- Kim Strömberg
- Mattias Carlsson
- Andreas Arvidsson

Den andra gruppens mål var att skapa en server samt nätverkskommunikationen mellan servern och klienten. De personer som blev tilldelade denna uppgift var:

- Sebastian Lagerman
- Andreas Wånge
- Henning Phan

Dessa två grupperna samarbetade starkt då gruppernas implementationer påverkade varandras arbeten. Utöver gruppen man tilldelades i fanns även extra uppgifter:

- Henning Phan - Projektledare
- Andreas Arvidsson - Sekreterare
- Andreas Wånge - Informationsansvarig
- Mattias Carlsson - Rapport
- Kim Strömberg - Rapport
- Sebastian Lagerman - Rapport

Projektledaren: Har den avgörande rösten när gruppen är oenade.
Sekreterare: Antecknar beslut tagna på möten.
Informationsansvarig: Håller sig updaterad om möten och deadlines och informerar gruppen på någon av projektets officiella kanaler.
Rapport: Är extra insatta i rapporten och detaljer angående rapporten.

B.1 Informationssökning

Eftersom syftet med detta arbetet är utforskandet av ett vanligt förekommande problem, har informationsökning varit viktigt för att hitta vad som gjorts tidigare inom området. Sökmotorerna Google, Google Scholar och Summon användes för att leta publikationer som var relevanta för projektet.

C

Spelmekaniken i Warlock

En match i Warlock tar plats på en arena som ständigt krymper med tiden, se figur 17. Runt denna arena finns ett farligt område som skadar spelare vid kontakt. Målet är att spelarna ska försöka döda varandra med hjälp av olika trollformler. För att åstadkomma detta måste man strategiskt skjuta trollformler för att skada motståndare och knuffa ut dem i det skadliga området. Samtidigt måste defensiva trollformler användas för att själv undvika att ta skada.

(Figur 17: Pågående Warlock match)

En runda i Warlock börjar med att alla spelare är immuna mot skada. Under denna tid har alla spelare möjligheten att att spendera guld i affären. Man kan köpa nya trollformler, uppgraderingar och föremål som kommer vara till hjälp för att både döda motståndare och undvika att bli dödad. Spelarna måste strategiskt välja vilka trollformler som ska köpas, då man endast kan ha en trollformel från varenda trollformelgrupp. Olika trollformlar kan passa bättre ihop än andra; därför går det i detta skede att finna en stor del av spelets strategiska aspekt.

Efter ett fördefinierat antal sekunder avslutas affärsrundan för att inleda själva stridsrundan. Spelarna placeras då ut på banan på nytt, med målet att använda de inköpta trollformlerna för att göra skada och knuffa motståndare ut ur banan. Ju fler spelare man dödar eller hjälper till att döda, desto mer pengar får man inför nästa runda.

Affärsrundan och stridsrundan alternerar tills ett visst antal rundor har spelats, varpå resultatet visas upp på skärmen. Varje spelares poäng baseras på en mängd olika faktorer - hur många andra spelare man fick sista attacken på, hurvida man var först med att döda en motståndare, hur länge man håller sig vid liv, samt vem som gjorde mest skada. Den spelare med mest poäng vinner matchen.

Det finns ett gäng olika spellägen man kan välja i spelet. Bland annat finns möjligheten att en spelare utses till en *avatar*. Det blir då alla andra spelares uppdrag att döda denna avatar. Detta blir ofta svårt, då avataren har mycket mer liv och gör mycket mer skada.

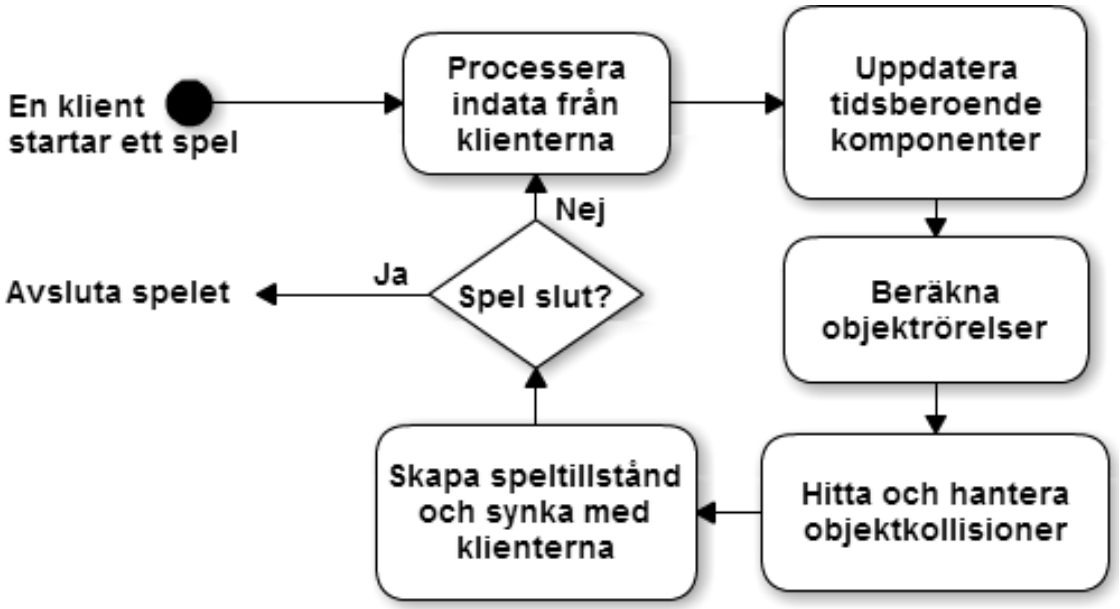
D

Övergriplig beskrivning av spelgång i server och klient

I denna bilaga beskrivs serverns och klientens kod i form av flödesscheman. Syftet är att presentera källkodens logiska exekveringsordning på ett överskådligt sätt. Noder i grafen kommer presenteras av en kort text som beskriver ett block av kods essens. Ur flödeschemat kan man förstå övergripligt hur programmen fungerar.

D.1 Spelgång för server

I figur 18 beskrivs spelgången för servermjukvaran när en klient har startat en match. Vad som händer innan startnoden är inte relevant för rapportens syfte. Programmet löper som standard igenom detta spelgång en gång var 50:e millisekund, vilket motsvarar ett så kallat tick".



Figur 18: Serverns spelgång för en match

En klient startar ett spel: Tre steg måste genomgås i denna nod. Det första steget är att skapa en match genom att låta klienten skicka kommandot CREATE_GAME via den redan upprättade TCP-anslutningen. När servern tar emot detta meddelande, skapas en lobby. Det andra steget är att låta övriga klienter ansluta till denna lobby via kommandot JOIN_GAME. Det tredje och slutgiltiga steget är att en klient startar matchen genom att skicka kommandot START_GAME. Servern initierar då matchen och notifierar alla klienter att spelet har börjat.

Processera indata från klienterna: Alla kommandon som kommer in från klienten kontrolleras för att verifiera att de kan och får utföras.

Uppdatera tidsberoende komponenter: Applikationen uppdaterar den totala tid som gått sedan rundan startade. Därefter uppdateras de komponenter vars tillstånd beror på denna tid. Detta sker var 50:e millisekund.

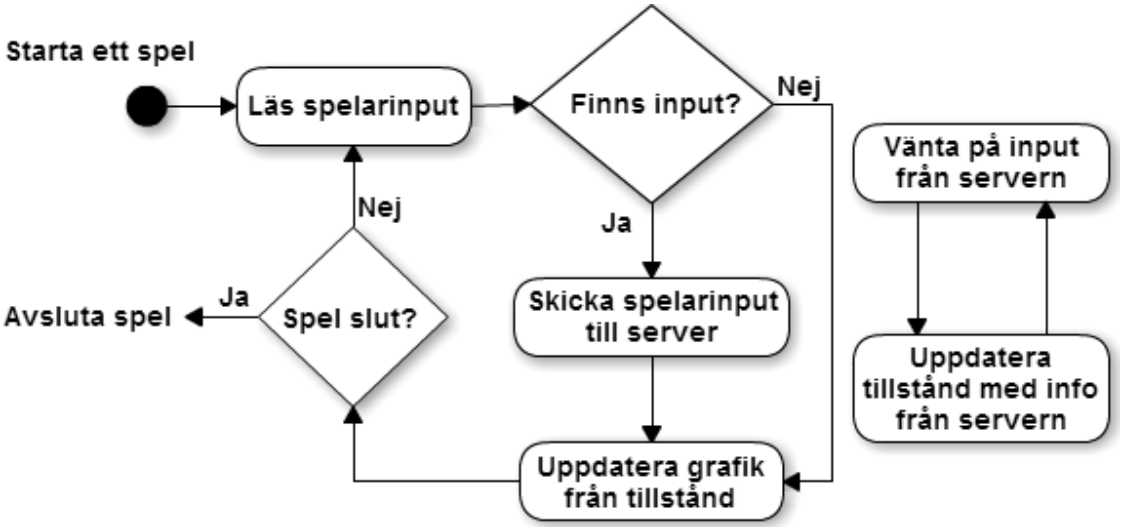
Beräkna objektrörelser: Applikationen går igenom en samling av alla dynamiska spelobjekt och kallar på deras uppdateringsmetoder. I de fall då objektet är aktivt enligt kriterier som varierar på objekttypen, uppdateras objektets position samt dess andra fysikaliska egenskaper såsom fart.

Hitta och hantera objektkollisioner: Applikationen hittar alla objekt som kolliderar i nuvarande tillstånd och utför kollisionshanteringsmetoder beroende på typen av kollision och objekt.

Skapa tillståndspaket och synka med klienterna: Spelets nuvarande tillstånd serialiseras till ett tillståndspaket och skickas ut till alla klienter i matchen.

Är rundan över?: Om det är fler än en spelare i matchen och mindre än två spelare är aktiva är rundan över. I det fall då endast en spelkaraktär är aktiv har spelaren som kontrollerar den vunnit rundan. I de fall då antalet rundor inte ännu nått det fördefinierade maxantalet rundor, återgår spelet till nästa runda.

D.2 Spelgång för klient



Figur 19: Klientens spelgång för en match

I figur 19 visas vad som händer på klienten under en match; på samma sätt som för servern, är det inte relevant för syftet vad som händer innan startnoden.

Starta ett spel: Spelet startar genom att servern skickar ut ett kommando som säger åt klienterna att det är dags att starta. Denna startpunkt till schemat fungerar likvärdigt oavsett om den aktuella klienten är värd för spelet (och således var den som startade det) eller inte.

Läs spelarinput: Applikationen läser ifall spelaren gett någon input sedan senaste iterationen.

Skicka spelarinput till server: Ett paket med kommandot och relevant data som parametrar skapas och skickas till server.

Uppdatera grafik från tillstånd: De grafiska objekten som ritas upp på spelplanen uppdateras enligt de nya värdena i det lokala tillståndet, vilket interpoleras mellan de två senaste tillstånd som servern givit.

Vänta på input från server: Denna och nästa nod är fristående eftersom de utförs asynkront. Uppdateringspaketen från servern kan komma när som helst och tas hand om oberoende av vart spelloopen befinner sig. I denna nod väntar klienten på nämnda paket.

Uppdatera tillstånd med info från servern: Spelets lokala tillstånd uppdateras enligt det paket som erhållits från servern.

E

Egenutvärdering

E.1 Andreas Arvidsson

Allmänt

- Förslagsgivare till projektet.
- Planerat och skapat material inför slutredovisning.
- Skrivit inlägg i tidsloggen efter varje relevant dag.
- Designat gruppens affischer inför presentationen.
- Kommit i tid till och deltagit i möten.
- Varit sekreterare vid möten
- Haft grundläggande demonstration av både Git och Unity.
- Deltagit i diskussioner och givit egna synpunkter

Kodbas

- Skrivit majoriteten av koden på klientsidan, inklusive det grafiska gränssnittet, nätverkskommunikationen samt lokal spellogik.
- Skrivit en del av serverkoden som kommunicerar med klienten.
- Utformat strukturen på nätverkspaketen tillsammans med Andreas och Henning.
- Skapat grundläggande visuella effekter.

- Implementerat de nätverksfunktionaliteter som används på klientsidan, samt simulering av nätverksproblem.

Rapport

Alla har kontribuerat till alla avsnitt i rapporten, men jag känner att jag mest har tillfört information till följande avsnitt:

- Sektion 1.6 - Metod
- Sektion 2.1 - Verktyg
- Sektion 4.2 - Protokoll
- Sektion 4.5 - Interpolering
- Sektion 4.7 - Informationsöverföring
- Sektion 5.1 - Resultat

Utöver detta har jag haft redaktionellt ansvar för följande avsnitt:

- Sektion 5.1 - Resultat
- Sektion 5.2 - Diskussion
- Sektion 5.3 - Framtid
- Appendix D

Utöver dessa punkter har jag även gjort bilder för nätverksarkitekturer, paketstruktur och interpolering. Dessutom har jag tillsammans med Kim Strömberg använt LaTeX för att strukturera rapporten.

E.2 Andreas Wånge

Allmänt

- Skapat presentationen för halvtidsredovisningen och hållit den, tillsammans med Kim Strömberg.
- Opponering av Rally Sport Racing Game med Kim Strömberg
- Hållt gruppen uppdaterad på viktiga moment
- Medverkade med insiktsfulla kommentarer i många intellektuellt givande diskussioner

Kodbas

- Designat en stor del av den inledande serverstrukturen.
- Stått för utformningen av nätverksprotokollen tillsammans med Andreas och Henning.
- Huvudansvarig för servern med Henning.
- Buggsökt nätverksproblem
- Bidragit till WarlocksCommons

Rapport

Alla har kontribuerat till alla avsnitt i rapporten, men jag känner att jag mest har tillfört information till följande avsnitt:

- Sektion 2.4 - Möten
- Sektion 3.3 - Utvärdering av spel
- Sektion 3.4 - Flöde
- Sektion 4.3 - Interpolering

Utöver detta har jag haft redaktionellt ansvar för följande avsnitt:

- Förord
- Abstract
- Sammanfattning
- Sektion 1 - Inledning

- Appendix C

E.3 Kim

Allmänt

- Skapat presentationen för halvtidsredovisningen och hållit den, tillsammans med Andreas Wånge
- Opponering av Rally Sport Racing Game tillsammans med Andreas Wånge
- Ansvarig för att att folk kom i tid (Jag erkänner att jag kanske inte va jättebra på detta)
- Bokat möten med fackspråk
- Varit kontaktperson mellan Daniel (handledare) och gruppen
- Informerat och sett till att alla gått på möten
- Tagit hennings draft av projektplaneringen och gjort den amazing tillsammans med Sebbe
- Väldigt aktiv under diskussioner
- Skrivit massor av texter till affischen (som gruppen fick välja ifrån)
- Annordnade team building tillfällen.
- gjorde Latex tillsammans med Andreas Arvidsson

Kodbas

- Implementerat ett skelett för magier i klienten
- Implementerat fireball på server och klient
- Implementerat ett skelett för kollisioner i servern
- Implementerat ett skelett för kartan och dess funktioner
- Enum för olika magiers typer
- Skelett för JSON
- Gjort tabeller över Magiers skada, cooldown, range osv

Rapport

- Gjort en draft av hela slutrapporten tillsammans med Sebbe
- Tagit ansvar för att folk jobbat med rapporten

- Strukturerat rapporten
- Hållt möten om rapporten (där gruppen gått igenom beslut de tycker om strukturella saker, samt pushat alla att jobba mer med den)
- Redaktionellt ansvar för nätverk tillsammans med Mattias
- Skrivit på alla stycken överallt
- Försökt att hålla isär rapportskrivandet så folk inte jobba på samma stycke

E.4 Mattias Carlsson

Allmänt

- Myntade slogan: "Stupid Sorcerers - Smart Networking"
- Dokumenterat arbetet
- Presentation av slutrapport
- Hjälpt att förbättra språkbruk i rapport och planering

Kodbas

- Kodstruktur och huvudloop
- Kollisionsdetektering
- Kollisionshantering
- Bakåtstötar
- Dynamiska objekts rörelser
- Spelarobjekt
- Trollformelobjekt
- Spelarenan
- Inläsning och serialisering av JSON-objekt
- Skapande av synkpaket
- Synkning av arenatillstånd
- Topbar på klienten

Rapport

Alla har kontribuerat till alla avsnitt i rapporten, men jag känner att jag mest har tillfört information till följande avsnitt:

- Sektion 1.5 - Avgränsningar
- Sektion 3.2 - Spelets mekanik
- Sektion 4.6 - Paketförlust
- Sektion 5.2 - Diskussion

Utöver detta har jag haft redaktionellt ansvar för följande avsnitt:

- Sektion 4.4 - Fördröjning
- Sektion 4.5 - Spelflöde
- Sektion 4.6 - Paketförlust

E.5 Henning Phan

Allmänt

- Tagit fram idén och varit förslagsgivare för projektet
- Planerat och skapat material inför slutredovisning.
- Skrivit inlägg i tidsloggen efter varje relevant dag.
- Bidragit med majoriteten av bilderna till rapporten samt alla trollformler
- Närvarat på alla officiella möten och varit tillgänglig under hela projektet
- Skrev hela utkastet för projektplaneringen
- Presentation av slutrapport
- Drivit konversationen angående säkerhet samt bidragit med idéerna inom området

Kodbas

- Huvudansvarig för servern med Wånge
- Skrivit TCP och UDP testverktyget
- Statistikprogrammet som ska stödja interpoleringsvärdena,
- Skrivit mätning av RTT, se branch Ping
- Skrivit pålitlig UDP överföring
- Refaktoriserat hela Gameklassen så den skapar tillstånd till förmån av Rewind
- Designat en stor del av den inledande serverstrukturen.
- Stått för utformningen av nätverksprotokollen tillsammans med Andreas och Henning.

Rapport

Alla har kontribuerat till alla avsnitt i rapporten, men jag känner att jag mest har tillfört information till följande avsnitt:

- Sektion 2.2 - Arbetsfördelning
- Sektion 2.3 - Informationssökning
- Sektion 4.1 - Nätverksarkitektur
- Sektion 5.3 - Framtid

Utöver detta har jag haft redaktionellt ansvar för följande avsnitt:

- Sektion 2.1 - Verktyg
- Kapitel 3 - Battle of Stupid Sorcerers
- Sektion 3.1 - Fundamentala spelkomponenter
- Sektion 3.2 - Spelets mekanik
- Sektion 3.3 - Utvärdering av spel

E.6 Sebastian Lagerman

Allmänt

- Sett till att alla gått på möten
- Sett till att det skrivits i loggbok varje vecka.
- Planerade upp draften på affischen.
- Kom i tid på gruppens förbestämda arbetstider.
- Planerat och skapat material inför slutredovisning.
- Har haft 100% närvaro på gruppmöten och varit tillgänglig under hela projektperioden.
- Har ägnat mycket tid åt att korrekturläsa rapporten och få den skriven på ett gemensamt språk.
- Skrev om Hennings projektplaneringen tillsammans med Kim Strömberg och Mattias Carlsson.
- Annordnade rum att jobba i under sista halvan av kandidatarbetet.
- Annordnade team building tillfällen.
- Dokumenterat arbete i en tidslog.

Kodbas

- Gjorde så att klienten och servern kunde spara ner meddelande till en log-fil.
- Skapade grunden av UDP lyssnaren för servern som senare vidarutvecklades av Henning Phan och Andreas Wånge.
- Tog fram en draft på hur gränsnittet skulle se ut i spelet.

Rapport

Alla har kontribuerat till alla avsnitt i rapporten, men jag känner att jag mest har tillfört information till följande avsnitt:

- Sektion I - Abstract
- Sektion II - Sammanfattning
- Sektion 1.1 - Bakgrund
- Sektion 1.4 - Utmaningar

Utöver detta har jag haft redaktionellt ansvar för följande avsnitt:

- Sektion 2.1 - Verktyg
- Sektion 2.2 - Arbetsfördelning
- Sektion 2.3 - Informationssökning
- Sektion 2.4 - Möten
- Sektion 4.7 - Informationsöverföring

Jag har dessutom gjort en draft av hela slutrapporten tillsammans med Kim Strömberg.