Examensarbete


HÖGSKOLAN I SKÖVDE

# ANVÄNDBARHET VID FÖRDRÖJNINGAR I FYSIKSIMULATIONER ÖVER NÄTVERK

# USABILITY WITH DELAYS IN PHYSICS SIMULATIONS OVER NETWORK

Examensarbete inom huvudområdet Informationsteknologi
Grundnivå 30 högskolepoäng
Vårtermin 2017

Måns Isaksson

# Sammanfattning

Detta arbete undersöker fysiksimulationer över nätverk och vilken effekt detta kan ha i datorspel. För denna undersökning används mänskliga testpersoner. Varje test undersöker två aspekter av den framställda applikationen.

För att undersöka detta mäts det hur spelares uppfattningar kring applikationen varierar med olika mängd fördröjningar. Det mäts också hur deras förmåga att utföra uppgifter förändras.

Resultaten visar att spelares uppfattningar kring ett spel med fördröjd fysik försämras men deras förmåga att avklara spelet är inte påverkat.

# Innehållsförteckning

# 1 Introduktion

Molntjänster inom spel har blivit ett större område på de senaste åren. Med lanseringen av Xbox One (2013) och Playstation 4 (2013) var molntjänster ett argument för att förlänga konsolernas livslängd. 2015 köpte Sony upp tjänsten OnLive vilket nu är känt som Playstation Now. Denna tjänst tillåter ägare av en Playstation 4 att strömma spel från ett serverkluster där de inte behöver uppdatera och rendera spelen lokalt (Sony, 2017).

Ett stort problem som dessa tjänster stöter på är det av fördröjningar. Då hela spelet uppdateras och renderas på en extern server behöver indata från spelarens kontrollinstrument först skickas till den server som har hand om att uppdatera spellogiken för att hanteras innan den slutgiltiga bilden kan renderas ock skickas ut till klienten. Detta introducerar ett flertal extra steg som behöver tas innan spelaren kan se sina handlingar utspela på skärmen där varje steg är mycket känsligt för fördröjningar.

I en intervju med utvecklarna för spelet Crackdown 3 som utvecklas för Microsoft Windows och Xbox One pratar utvecklarna om hur de använder molnkalkyleringar för att hjälpa till med fysiksimulationer (Robertson, 2015). Microsofts sätt att använda sig av molnet är i teorin mindre känsligt för fördröjningar då spelares handlingar kan hanteras och renderas direkt utan att behöva vänta på svar från en extern server. I en miljö som denna kan en extern server hjälpa till med uppdatering av krävande spellogik utan att påverka de aspekter av applikationen som kräver lägre svarstider. Detta arbete har som mål att undersöka hur fördröjningar i fysiksimulationer påverkar spelares möjlighet att avklara uppgifter i spelet samt hur det påverkar deras uppfattningar kring spelet. Detta testas genom att skapa en applikation där fysik uppdateras på en extern server och resultatet av simulationen skickas till klienten för rendering.

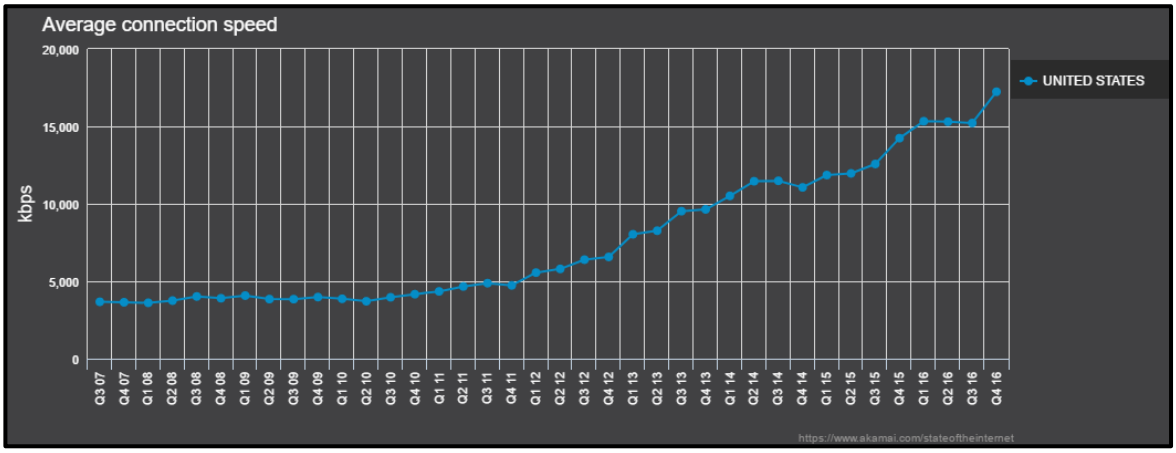
Denna rapport beskriver implementationen av en testmiljö. Denna testmiljö används för uppdatering av fysiksimulationer på en extern server där resultatet av denna simulering kan skickas till en ansluten klient för rendering. För att kunna testa hur denna fysik beter sig i olika mänger fördröjning implementeras en artificiell fördröjning mellan mottagandet av paket från servern och hantering av dessa paket på klienten.

Denna testmiljö används sedan för att utföra en undersökning på mänskliga testpersoner. Denna undersökning samlar in data kring testpersonernas uppfattningar via en enkät. För att mäta testpersonernas prestanda samlas också data om hur fort testpersonerna kan avklara de hinder de möter på inom testmiljön. Denna data analyseras ock reflekteras kring för att bedöma om de mål som arbetet satt upp har nåtts.

# 2 Bakgrund

## 2.1 Nätverk i spel

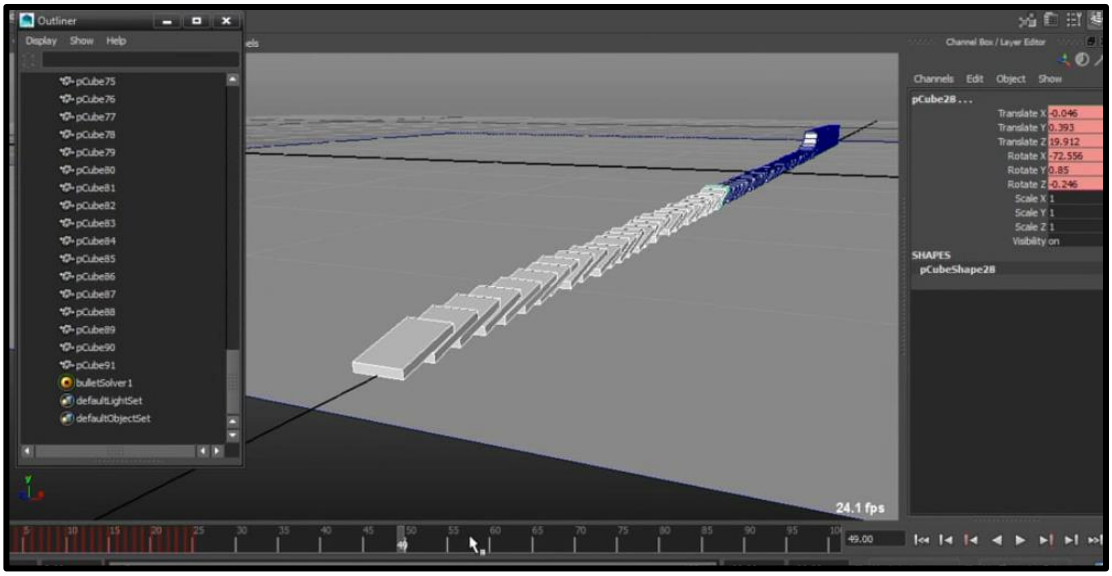
Onlinespel har under de senaste åren växt i popularitet. I en intervju med skaparna av online-spelet *League of Legends* (Kollar 2016) avslöjades det att spelet nått en aktiv användarbas på 100 miljoner spelare per månad. Overwatch, utvecklat av Blizzard Entertainment, som kom ut 2016 har sedan april 2017 30 miljoner användare (PlayOverwatch 2017).




**Figur 1** Denna graf visar den genomsnittliga hastigheten på en bredbandsuppkoppling i USA (Akamai 2017)

Som Figur 1 illustrerar går den genomsnittliga bredbandshastigheten hos användare i USA stadigt upp. I takt med detta har också spel börjat använda mer bandbredd för att skicka data över nätverket. I en studie som jämförde mängden data som skickades i moderna och äldre spel i genren MMORPG uppstod det tydligt att trenden gick mot att använda en större mängd bandbredd (Suznjevic & Matijasevic 2015). Suznjevic och Matijasevic menar på att detta beror på att mängden datapaket som skickas har ökat vilket i sin tur minskar mängden fördröjning i nätverket.

## 2.2 Realtidsfysik och förstörbar miljö




**Figur 2** Fysikmotorn Bullet (bulletphysics, u.å.) används för att förberäkna en fysiksimulation i 3D modelleringsprogrammet Maya

Om fysiksimulationer inte behöver vara dynamiska eller inte interagerar direkt med spelaren går det att förberäkna fysiksimulationen. Som illustrerat i Figur 2 går det använda färdiga fysikberäkningsprogram för att utföra simulationer i förtid för att sedan spara dem i en animation. Detta minskar de kalkylationer som behöver utföras i realtid men tillåter inte animationen att dynamiskt ändra på sig efter spelarens beteenden.

### 2.2.1 Splittring av geometri

I konferensbidraget *Evaluation of real-time physics simulation systems* (Parker O'Brian 2009) undersökes metoder för att simulera förstörbara miljöer i realtid. Denna studie anser att mer dynamiska lösningar för att splittra geometri i förstörbar terräng erbjuder ett mer energiskt och realistiskt resultat än dess statiskt baserade motsvarighet.

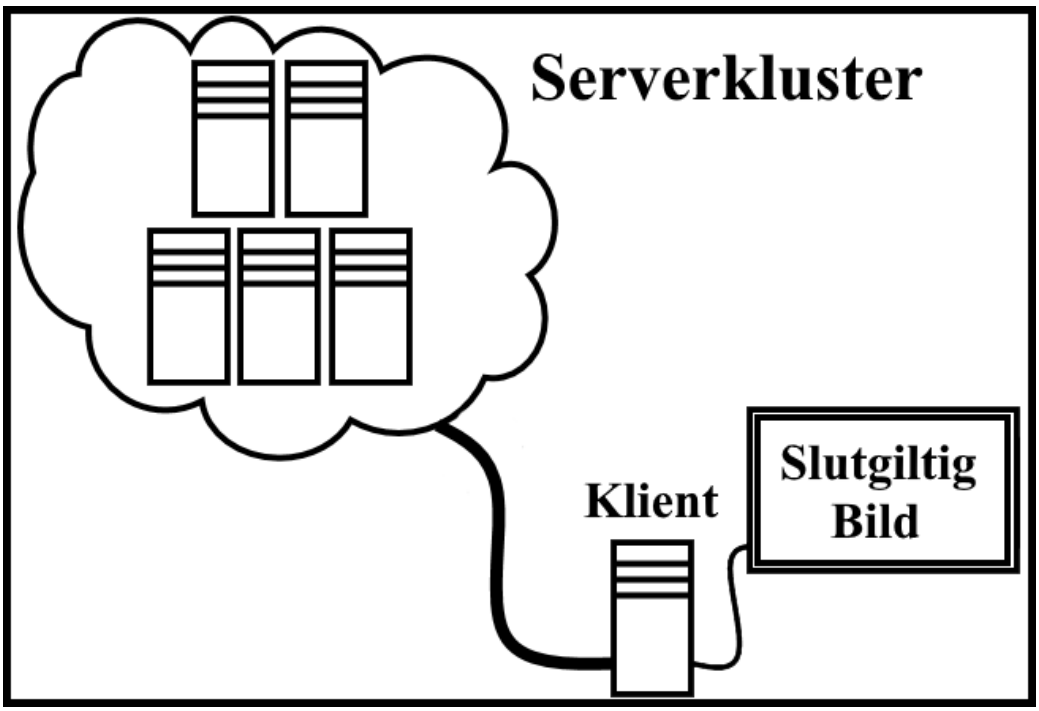
Ett stort område inom realtidsfysik är splittring av geometri vilket är fysikmotorns sätt att simulera hur olika objekt krossas vid stötar. Ett exempel skulle kunna vara hur en glaskula splittras när den slår i marken. Fysikmotorn behöver dela på geometrin som bygger upp objektet för att sedan simulera fysik på alla enskilda delar separat. Att dynamiskt kalkylera hur objekt skall splittras vid kollision kan dock påverka hur väl simuleringen kan skötas i realtid. Parker och O'Brian (2009) påpekar att alla fysikobjekt som befinner sig inom en viss längd av varandra behöver beräkna kollision mot varandra för att se om de kolliderar. Detta leder till att komplexiteten exponentiellt ökar med mängden objekt som läggs till i scenen vilket enligt Parker och O'Brian (2009) är ett stort problem vid splittring av objekt och begränsar mängden splittring som kan ske i en realtids applikation.

## 2.3 Molnberäkning inom spel

### 2.3.1 Spelströmning över nätet

Molnkalkyleringar har under de senaste åren blivit mer relevant inom spelindustrin. Sonys uppköp av molntjänsten OnLive 2015 är ett exempel på hur stora företag är villiga att satsa på

molntjänster inom spel. Denna tjänst, nu känd som Playstation Now (Sony 2017), tillåter alla ägare av ett Playstation 4 att uppdatera och rendera ett spel på ett avlägset moln-kluster där den slutgiltiga renderade bilden strömmas tillbaka till spelaren i realtid.




**Figur 3** Denna bild illustrerar hur ett serverkluster kan användas för att uppdatera och rendera spellogik. Hårdvaran på klientens sida behöver då endast packa upp bilden som skickats av serverklustret

Då dessa molnlösningar kräver att indata från spelarens kontrollinstrument först måste skickas till en avlägsen server blir fördröjningar ett problem. Denna data behöver först skickas till servern som i sin tur hanterar denna indata, rendera en bild och skickar tillbaka resultatet, se Figur 3. Detta leder till att det kan ta tid innan ändringar kan visas på skärmen om anslutningen mellan klient och server har långa fördröjningar.

### 2.3.2 Fördröjningar i molnspel

Då indata från spelaren behöver skickas till en extern server så introduceras fördröjningar. I undersökningen *On Models for Game Input with Delay – Moving Target Selection with a Mouse* (Claypool 2016) testas det hur fördröjningar påverkar spelarens förmåga att utföra uppgifter med olika mängd fördröjningar. Claypool kom fram till att spelarens förmåga att utföra uppgifter försämras linjärt med mängden fördröjningar som introduceras till spelarens inmatningar.

En tidigare studie från Claypool och Finkel (2014) testade den tidigare kommersiella tjänsten OnLive och ett program med öppen källkod vid namn GamingAnywhere som körs över ett lokalt nätverk. Både dessa tester visar på hur både spelarens njutning av spelet samt des förmåga att utföra uppgifter linjärt försämras med mängden fördröjning som introduceras. Claypool och Finkel (2014) menar på att fördröjningar är ett mycket stort problem vad det gäller att strömma spel över nätverk. Dess studie visar på att spelarens förmåga att utföra uppgifter försämras upp till 25% per varje 100 millisekunder av fördröjningar. I ett traditionellt spel hanteras användarhandlingar lokalt av spelmotorn men i ett spel som körs

över nätverket visas handlingarna först på skärmen efter att handlingen skickats till molnet var det behöver hanteras och renderats för att sedan skickas tillbaka i form av en renderad bild.

### 2.3.3 Alternativa molntjänster för spel

I konferensbidraget *A Distributed Architecture for Mobile Digital Games Based on Cloud Computing* (Zamith et. al. 2011) undersöks det hur arbete från uppdateringsloopen av ett spel kan designas för att tillåta delar av spelet att beräknas på molnet. Uppdateringsloop i detta sammanhang refererar till det arbete som behöver utföras vid varje uppdatering av spelet. Detta arbete kan variera beroende på t.ex. spelarens indata eller hur många fysikobjekt som finns på skärmen vid tillfället av uppdateringen.

Genom att avlasta beräkningar som t.ex. artificiell intelligens till en extern part går det att ha komplexa beteenden på mindre kraftfull hårdvara. Detta kan hjälpa med fördröjningar vilket är ett problem som besvärar strömning av hela spelet. Genom att bryta ut delar av spelet som inte är lika beroende av svarstider går det öka prestandan av spelet utan att introducera fördröjningar på spelarens kontrollinstrument.

I en intervju med utvecklarna för spelet Crackdown 3 som utvecklas för Microsoft Windows och Xbox One pratar utvecklarna om hur de använder molnkalkyleringar för att hjälpa till med fysiksimulationer (Robertson 2015). Crackdown 3 är ett flerspelarspel där förstörbar terräng är ett stort fokus. Detta spel använder sig av molnkalkyleringar för att hjälpa till med kalkylationer men istället för att processerna hela spelet i molnet bryts uppdateringsloopen upp. De delar som har hand om att kalkylera den förstörbara terrängen hanteras på molnet och resultatet av fysiksimulationen skickas sedan ut till alla anslutna klienter.

# 3 Problemformulering

Claypool's och Finkel's (2014) påstående om att spelarens förmåga att utföra uppgifter försämras upp till 25% per varje 100 millisekunder av fördröjningar påvisar hur viktigt det är med svarstider inom spel**.** Vad som undersöks med detta arbete är hur avlastning av fysiksimuleringar till en extern part och de fördröjningar det medför påverkar både spelarens förmåga att utföra uppgifter samt dess uppfattning av spelet.

Som Parker och O'Brian (2009) beskriver kan en mer dynamisk metod att simulera fysik få simulationerna att uppträda bättre. Detta kommer dock med en kostnad vad det gäller processorcykler och då spel kräver mer än bara fysiksimulering är det viktigt att simuleringen är snabb så att den inte hindrar uppdateringen av spelet samt rendering av scenen. Detta arbete siktar in sig på att avlasta beräkningar vad det gäller fysik till en extern server som kalkylerar fysiken i realtid och skickar resultatet till en klient som sedan hanterar rendering och uppdatering av spelloopen. Denna studie testar hur fördröjningar av dynamisk fysik påverkar delvis spelarens förmåga att avklara uppgifter samt hur dess uppfattning av spelet förändras vid olika mängd fördröjning. Hypotesen här är att spelares förmåga att utföra uppgifter inte kommer påverkas men att dess upplevelse kommer försämras när fysiksimulationer blir fördröjda.

Då detta är en studie inom mänsklig förmåga att avklara uppgifter under bestämda förhållanden behöver tester utföras med en mängd mänskliga testare. Genom att samla data på ett flertal olika testfall med olika mängd fördröjningar går det skapa en korrelation mellan mängden fördröjning som en spelare utsätts för och hur bra de presterar under denna fördröjning. Problem som uppstår med enkätundersökning är hur spelares svar kan drastiskt variera beroende på hur vana de är vid spel samt vad de tycker om förstapersons skjutarspel. En del av enkäten är därför dedikerad till att samla data kring hur van spelaren är vid att spela förstapersons skjutarspel.

Då det inte finns några spel på marknaden där det går att simulera fördröjd fysik behöver en applikation skapas för utförandet av denna studie. Genom att skapa en specialgjord applikation går det att göra designval som tydliggör problem med fördröjd fysik. Ett exempel på ett sådant designval är att applikationen är mycket fokuserad på interaktion mellan spelare och fysikobjekt. Detta skapar en kontrollerad miljö där utomstående faktorer så som skärmtäckande partikeleffekter och krävande grafik inte påverkar spelarens uppfattningar kring spelet.

## 3.1 Metodbeskrivning

### 3.1.1 Design av applikation

För att testa tesen tas en applikation i form av ett spel fram där fysiken av valda objekt kan simuleras på en extern server. Detta spel utspelar sig som en *First Person Shooter* d.v.s. ett förstapersons skjutarspel. Claypool och Finkel (2014) påpekar att förstapersonsspel är extra känsliga mot fördröjningar vilket leder till att resultaten i denna studie blir mer tydliga. *Evaluation of real-time physics simulation systems* av Adrian Boeing och Thomas Bräunl (2007) analyserar prestandan på sju olika fysiksimuleringsmotorer och kommer fram till att prestandan mellan dessa moderna fysikmotorer har liknande prestanda inom de flesta

områden. Detta påvisar att den fysikmotor som väljs inte har stor påverkan på det slutgiltiga resultatet så länge motorn stödjer simulering av splittrad geometri.

### 3.1.2 Undersökning

I denna studie testas tre olika fördröjningar. Den första fördröjningen som testas är med 0 millisekunder d.v.s. ingen fördröjning. Detta är för att skapa en uppfattning om hur spelare upplever spelet utan någon form av fördröjning. I *Latency and player actions in online games* (Claypool 2006) observeras det att den maximala mängd fördröjning som är acceptabel i ett förstapersonsspel är 100 millisekunder. Den andra fördröjningen som valts är 100 millisekunder och representerar en fördröjning som ligger på den acceptable gränsen.

Den sista fördröjningen är 200 millisekunder och representerar en hög mängd fördröjning. Denna fördröjning är baserad på den högsta mängden fördröjning Claypool och Finkel (2014) undersökte där de kom fram till att spelarupplevelsen inte längre var acceptabel vid denna fördröjning. Deras studier har dock full simulering på servern där denna undersökning endast gör en partiell simulering på servern. Hypotesen är då att denna lösning kan hantera högre mängd fördröjningar utan att påverka spelarens prestanda eller upplevelse till en större grad.

För att mäta spelarens möjlighet att utföra uppgifter med olika fördröjningar byggs en bana där spelaren behöver trycka på ett antal knappar. Dessa knappar är inkapslade i förstörbara objekt där spelaren måste använda sig av explosiva vapen för att förstöra objekten för att kunna nå knapparna. Spelaren blir instruerad att trycka på alla knappar så snabbt som möjligt. Denna tid används som mått för hur bra spelare presterar.

En enkät tas fram för att mäta spelares upplevelse av spelet. Enkäten i denna studie är baserad på undersökningen utförd av Claypool och Finkel (2014). I deras undersökning lät de testpersoner spela ett spel där mängden fördröjningar varierade mellan testpersonerna. För att avgöra spelarupplevelsen använde de sig av en enkät. Denna enkät var uppbyggd av en mängd frågor där svaren gick i en skala mellan 0 – 5 där 0 representerar att upplevelsen var dålig och 5 representerar en bra upplevelse. Summan av alla frågor sedan slås ihop för att få ett värde som representerar spelarens upplevelse.

### 3.1.3 Enkät

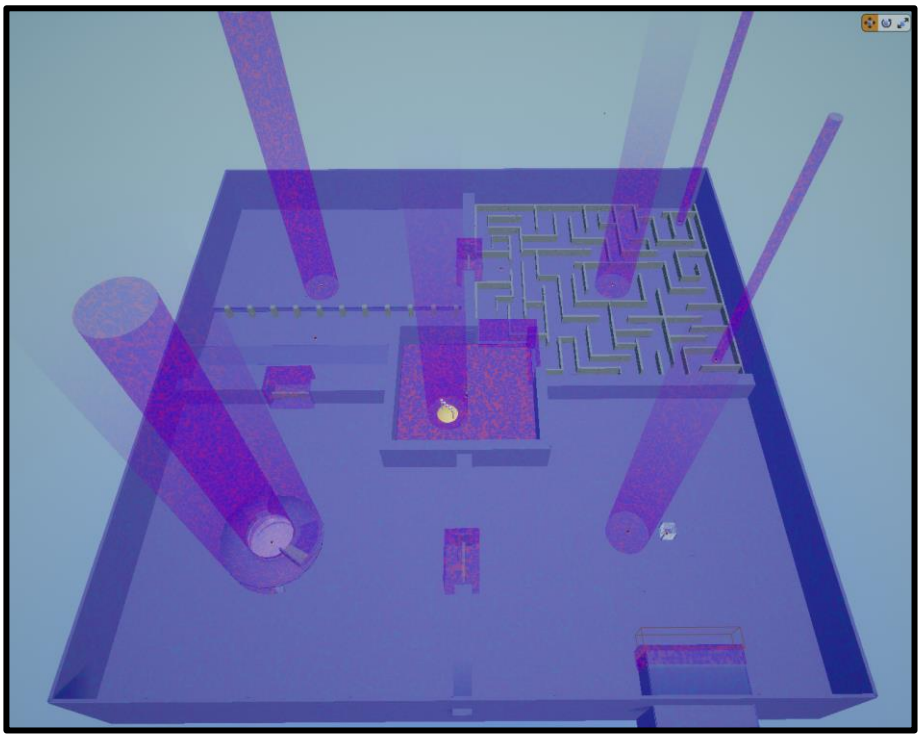
Frågorna i denna studie är uppdelad i två delar. Den första delen är för att mäta varje deltagares åsikter om förstapersons skjutarspel samt hur bra de upplever sig själva att vara på dem. Detta ger ett kontrolvärde då det går att jämföra snittet på olika grupper för att se hur dess erfarenhet inom förstapersonsspel varierar. Den andra delen fokuserar mer på hur deltagarna uppfattade spelet samt dess upplevelse. Dessa frågor ger en subjektiv bild på hur spelare uppfattar spelet med olika mängd fördröjning. Skulle spelare med högre mängd fördröjning t.ex. finna att det var svårare att avklara en uppgift går det att dra en slutsats om att upplevelsen blivit sämre. Alla de frågor som ställs har ett svar som går mellan 1 – 5 för att ge ett kvantitativt värde på hur spelare uppfattar upplevelsen. För den fulla enkäten se Appendix A.

# 4 Projektbeskrivning

## 4.1 Artefakt

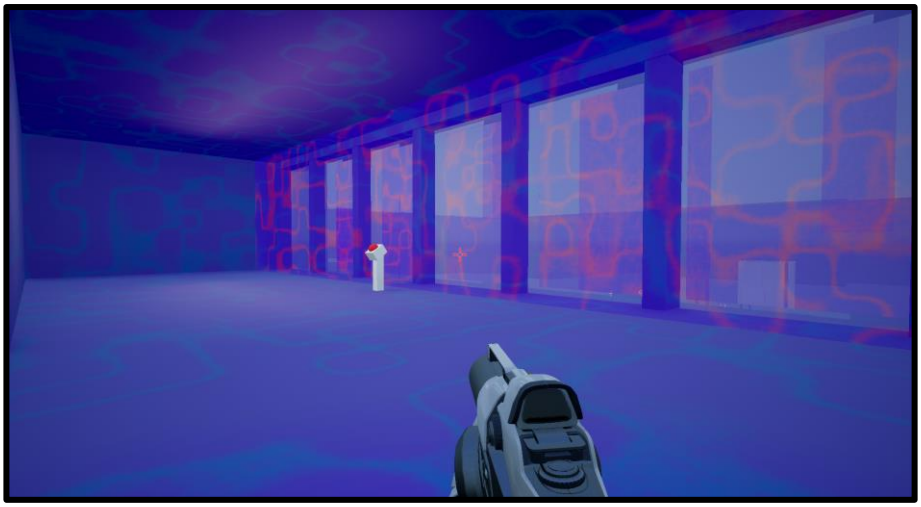
För rendering och hantering av nätverksgränssnitt används spelmotorn Unreal Engine 4. Denna spelmotor använder fysikmotorn PhysX (Epic Games u.å.c) vilken var del av de sju fysikmotorerna som testades av Boeing och Bräunl (2007). Denna fysikmotor är stabil och har inga stora nackdelar i jämförelse med andra stora fysikmotorer och används därför som en grund för fysiksimulationer. Unreal Engine 4 har också ett inbyggt system för att hantera splittring av geometri vilket är essentiellt för att skapa förstörbara miljöer.

Nätverksgränssnittet inom Unreal Engine 4 tillåter full replikering av klientens tillstånd till en extern server (Epic Games u.å.a). Detta leder till att de simulationer som sker på servern kommer utspela sig efter klientens indata. Resultatet av dessa fysiksimulationer kan sedan skickas tillbaka till klienten där denna data hanteras för att skapa en visuell representation av vad servern simulerat utan att göra de faktiska fysiksimulationerna lokalt.



**Figur 4** Testmiljön sen från fågelperspektiv

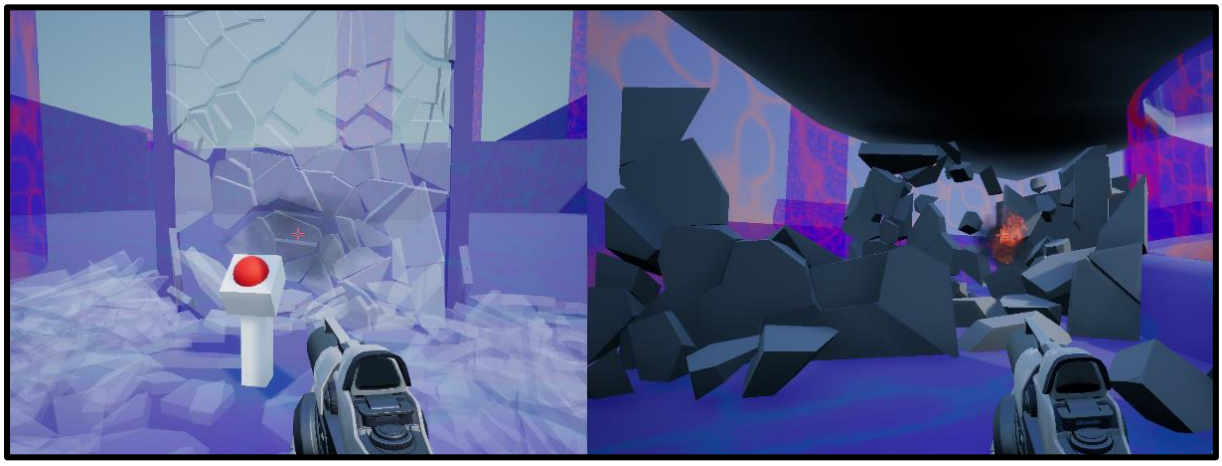
Testmiljön är uppbyggd av sex rum, se Figur 4. Testpersonen rör sig igenom dessa rum i en satt ordning. Varje rum har en eller flera knappar som öppnar ett kraftfält som blockerar antingen en ny knapp eller dörren till nästa rum. För att komma åt dessa knappar behöver testpersonen utföra en uppgift som är direkt anknuten till fysiksimulationer. Testpersonen är utrustad med ett vapen som avfyrar explosiva raketer. Dessa raketer kan användas för att spränga förstörbara objekt. Ett exempel på detta är det första rummet där spelaren behöver spränga ett par glasrutor för att ta sig ut. Ett kraftfält blockerar både spelaren och de skott den avfyrar. Detta betyder att förstörbara objekt som är placerade bakom ett kraftfält inte går att förstöra innan kraftfältet är avstängt.



**Figur 5** Start-rummet där testpersonen får en chans att gå igenom spelets mekanik innan testet börjar

Det första rummet (Se Figur 5) ger testpersonen en chans att lära sig de två största mekanikerna. Den första mekaniken är hur ett kraftfält försvinner när en knapp trycks ner. Detta är extra tydligt i detta rum då knappen är placerad precis framför det kraftfält som försvinner. Bakom detta kraftfält är en mängd glasrutor som blockerar vägen ut ur rummet. Dessa glasrutor är förstörbara och agerar som ett sätt att lära spelaren att vapnet de håller i kan förstöra objekt. När spelaren lämnat det första rummet startar en klocka som mäter mängden tid som passerat. Anledningen till att klockan först startar efter att testpersonen lämnat rummet är då detta rum agerar som ett sätt att låta spelaren bekanta sig med spelet. Denna klocka stoppas när alla pussel är avklarade för att mäta testpersonens prestanda.

Det andra rummet introducerar testpersonen till navigering kring förstörbara objekt. I detta rum är en knapp blockerad av ett par glasskivor. Efter att dessa glasskivor är förstörda behöver testpersonen gå igenom dess spillror för att nå kappen. Glaset används för att testa ett förstörbart objekt som splittras i en stor mängd delar. Som Parker och O'Brian (2009) påpekar så kommer en stor mängd splittror nära varandra skapa ökad komplexitet. Detta lägger stress på fysiksimulationen samt nätverket som behöver skicka information om alla ändringar som skett.



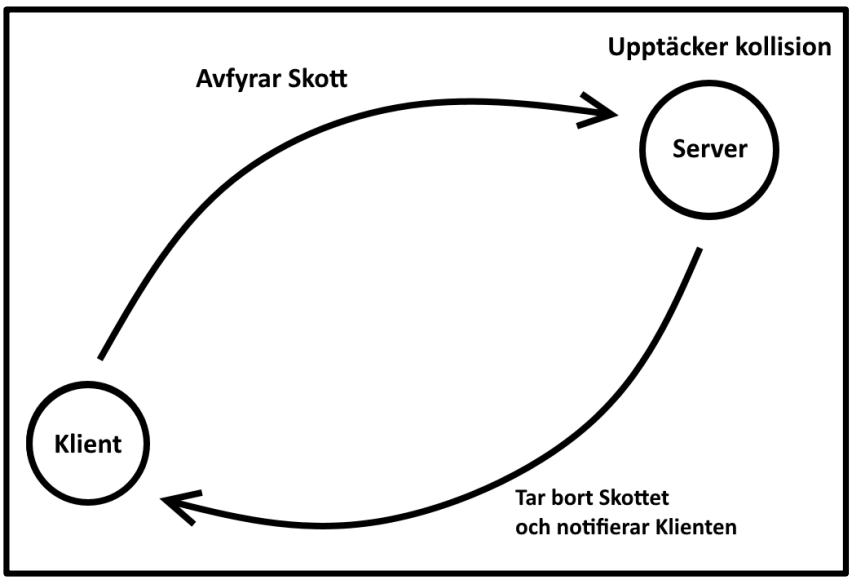
**Figur 6** Jämförelse mellan splittring av glas och splittring av väggar

Utöver en hög mängd glassplitter är det första rummet inte mycket svårt att avklara. Efter att testpersonen tryckt på knappen kan den genast fortsätta till nästa rum. I detta rum behöver spelaren gå upp för en spiraltrappa för att komma åt knappen som öppnar nästa rum. I denna trappa existerar ett flertal väggar som blockerar spelaren från att fortsätta uppåt. Dessa väggar splittras i större delar när de förstörs och testar då hur lätt det är för spelaren att navigera kring spillror av större storlek. Skillnaden mellan splittring av glas och väggar illustreras i Figur 6.

Rum tre är designat att tvinga testpersonen att direkt interagera med förstörda objekt. I detta rum behöver spelaren ta sig upp på en plattform som är för högt upp för att kunna nå genom att hoppa. Testpersonen behöver förstöra en av de pelarna som står bredvid denna plattform och använda de delar som faller av genom att först hoppa upp på dem och sedan upp på plattformen. Det sista rummet, innan målet, är designat för att testa huruvida spelaren väljer att interagera med ett förstörbart objekt eller att gå runt det om möjligt. Detta rum är en labyrint uppbyggt av förstörbara väggar. I detta rum behöver testpersonen ta sig till tre olika knappar i en bestämd ordning. De kan välja att gå igenom labyrinten eller att förstöra väggarna och gå rakt till knapparna. Efter att testpersonen tryckt på den sista knappen i detta rum öppnas det sista rummet som är målet. I detta rum trycker spelaren på en knapp som avslutar testet vilket leder till att timern stoppas och den slutgiltiga tiden visas för testpersonen.

## 4.2 Implementation

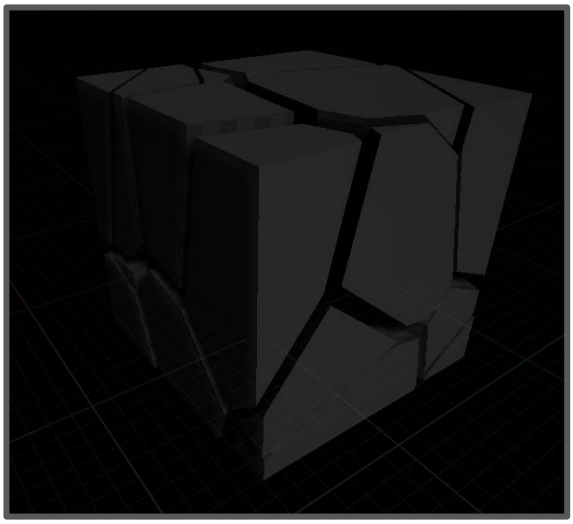
För att lösa replikering av spelarens handlingar användes den inbyggda replikeringen i Unreal Engine 4 (Epic Games u.å.a). Unreal Engine 4 håller en kopia av spelets tillstånd på servern. När klienten utför handlingar skickar den information om dessa handlingar till servern så den kan korrigera sin version av speltillståndet. Dessa handlingar inkluderar rörelse av spelaren och avfyrande av skott.




**Figur 7** Exempel på interaktion mellan klient och server

Figur 7 visar ett exempel på hur interaktionen mellan klienten och servern sker. I detta exempel har klienten avfyrat ett skott och skickar information om detta till servern. Servern

upptäcker sedan att detta skott har kolliderat med ett objekt och behöver sprängas. Servern tar då bort detta skott och skickar information om detta till klienten. Anledningen till varför denna kollisions-beräkning inte sker på klienten är för att servern håller alla beräkningar för förstörbara objekt. Servern kan därför bedöma om ett skott har kolliderat med ett förstörbart objekt och starta fysiksimulationer i reaktion till detta.



**Figur 8** Förberäknad splittring av en kub

Som illustrerat i Figur 8 är de förstörbara objekten uppbyggda av ett flertal förberäknade mindre delar. Detta betyder att när ett objekt splittras kommer splittrorna alltid se likadana ut. Klienten kan därför hålla en egen version av splittrorna och servern behöver endast skicka information om hur dessa splittror skall vara positionerade.

Här visas ett exempel på hur servern skickar information om förstörda objekt till klienten.

```
// Check for chunks that have moved and broadcast them to relevant clients
for (int32 i = 0; i < ChunkCount; i++)
{
	PxRigidDynamic* PxChunkActor = DestructibleComponent->ApexDestructibleActor-
>getChunkPhysXActor(ChunkIdxs[i]);
	if (PxChunkActor)
	{
		FTransform ChunkTransfrom = P2UTransform(PxChunkActor->getGlobalPose());
		FChunkData chunk(ChunkIdxs[i], ChunkTransfrom.GetLocation(),
ChunkTransfrom.Rotator());

		if (!HasChunkMoved(chunk)) // We don't want to send the client info if
the chunk has not moved
			continue;

		uint32 packetID = 0;
		if (auto lastReceivedChunk = ChunkCache.Find(ChunkIdxs[i])) // Check for
id of last chunk sent with this index and add one (Used for knowing if the packet is
old)
			packetID = lastReceivedChunk->packetID + 1;

		chunk.packetID = packetID;
		chunkDataArray.Chunks.Add(chunk);
		ChunkCache.Add(ChunkIdxs[i], chunk); // Cache the sent information
```

```
			if ((i + 1) % 27 == 0) // Max 27 chunks per package (UDP has a "safe
limit" of 512 bytes)
			{
				gameMode->BroadcastUDPPackage(&chunkDataArray);
				chunkDataArray.Chunks.Empty();
			}
		}
}

// Send any remainder chunks
if (chunkDataArray.Chunks.Num() > 0)
		gameMode->BroadcastUDPPackage(&chunkDataArray);
```

När en del av ett förstörbart objekt flyttat på sig lägger servern till delens unika ID, dess position och dess rotation i en lista av förflyttade delar. Servern går igenom denna lista i ett satt intervall och skickar denna information till klienten. Då denna applikation är en realtidsapplikation går servern igenom denna lista 30 gånger i sekunden vilket är tillräckligt för att anses vara realtid utan att ta upp all bandbredd på nätverket. Servern skickar resultatet av denna förflyttning till klienten i form av UDP paket. Paketen delas upp i delar av 500 byte för att minska chansen för packet-loss om paketen blir för stora. När klienten tar emot dessa paket behöver den uppdatera positionen av dess lokala version av de förstörbara objekten.

Här visas ett exempel på datastrukturen som används för att skicka UDP paket.

```
USTRUCT()
struct FChunkData : public FNetData
{
		GENERATED_BODY()

		uint8 ChunkIndex;
		FVector Location;
		FRotator Rotation;

		.
		.
		.

		virtual FArchive& Serialize(FArchive &Ar) override
		{
			Ar << packetID; // We need to know if the packet we received contains
old information since UDP can arrive in the wrong order and we only care about the
newest one
			Ar << ChunkIndex;
			Ar << Location;
			Ar << Rotation;

			return Ar;
		}
};

USTRUCT()
struct FChunkDataArray : public FNetData
{
		GENERATED_BODY()

		FNetworkGUID Recipient;

		TArray<FChunkData> Chunks;
```

```
	.
	.
	.

	virtual FArchive& Serialize(FArchive &Ar) override
	{
		Ar << packetID;
		Ar << DataID;
		Ar << Recipient;
		Ar << Chunks;
		return Ar;
	}
};
```

Koden ovan visar hur UDP paketen som skickas till anslutna klienter ser ut. Dessa paket består av två delar, information om varje del som ska uppdateras och en lista av dessa delar så det går att skicka information om mer än en del i taget. Detta paket har också ett FNetworkID vilket är ett unikt identifieringsnummer inom Unreal Engine som genereras för varje objekt som delas över nätverket. Detta nummer används så att klienten vet vilket objekt som fysiken behöver uppdateras på. Unreal Engine 4 ger också ett gränssnitt för att serilisera datatyper. Detta visas i `Serialize(FArchive &Ar)` funktionen. I denna funktion ges det möjlighet att välja hur datatyper skall packas och packas upp. Det är här möjligt att implementera komprimering av data. Då projektet inte har komprimering i fokus skickas alla datatyper ner utan någon komprimering.

Här visas ett kodexempel på hur klienten hanterar data den får av servern.

```
for (auto It = ChunkMovment.CreateIterator(); It; ++It)
{
	PxRigidDynamic* PxChunkActor = DummyDestructibleComponent-
>ApexDestructibleActor->getChunkPhysXActor(It.Key());

	if (!PxChunkActor)
	{
		UE_LOG(LogTemp, Warning, TEXT("(Client) Tried to update non existing
Chunk, chunk id: %s"), *UKismetStringLibrary::Conv_IntToString(It.Key()));
		return;
	}

	It.Value().TravelAlpha = UKismetMathLibrary::FClamp(It.Value().TravelAlpha +
TickRate * DeltaTime, 0.0f, 1.0f);
	FVector newLocation = UKismetMathLibrary::VLerp(It.Value().StartLocation,
It.Value().TargetLocation, It.Value().TravelAlpha);
	FRotator newRotation = UKismetMathLibrary::RLerp(It.Value().StartRotation,
It.Value().TargetRotation, It.Value().TravelAlpha, true);

	// Tell PhysX that the location of the chunck has moved
	PxChunkActor->setKinematicTarget(U2PTransform(FTransform(newRotation,
newLocation)));
	DummyDestructibleComponent->SetChunkWorldRT(It.Key(), newRotation.Quaternion(),
newLocation);

	FTransform chunkTransform = P2UTransform(PxChunkActor->getGlobalPose());

	// clear the chunck movement if it has reached its desired pose
	if (It.Value().TravelAlpha == 1.f)
		It.RemoveCurrent();
}
```

När klienten tagit emot information om att en del av ett objekt har flyttats så sparar den undan nuvarande position tillsammans med den nya positionen av delen. Klienten utför sedan en operation känd som Lerp (Linear Interpolation) vilket betyder att den utför en stegvis förflyttning mellan dessa två punkter. Då klienten vet att servern skickar paket 30 gånger i sekunden utför den förflyttningen under en trettiondels sekund. Detta är så att klienten hinner utföra förflyttningen i tid för ett nytt potentiellt paket från servern.

Denna kod visar hur fördröjningar i nätverket simuleras.

```
int32 SimulatedNetDelay = CVarSimulatedPhysicsDelay.GetValueOnGameThread();
float delay = FMath::Clamp((float)SimulatedNetDelay / 1000.f, 0.0001f, 999999.f);

FTimerHandle DummyHandle;
GetWorldTimerManager().SetTimer(DummyHandle, FTimerDelegate::CreateUObject(this,
&AUDPReceiver::UpdateChunkData, netPacket.RDActor, netPacket.ChunkDataArray), delay,
false, delay);

.
.
.

// the function used to update the appropriate object on the client
void AUDPReceiver::UpdateChunkData(ARemoteDestructibleActor* rdActor, FChunkDataArray
chunkDataArray)
{
	rdActor->UpdateChunkData(chunkDataArray);
}
```

För att simulera fördröjningar i nätverket så sattes en timer på klientens sida som fördröjde hanteringen av paket. Denna timer kallade på en funktion efter en bestämd mängd fördröjning som i sin tur passerade vidare informationen om de flyttade delarna till det objekt paketet var ämnat för. Mängden fördröjning bestämdes genom användandet av konsolvariabeln `CVarSimulatedPhysicsDelay`. Konsolvariabler är globala variabler som kan bli ändrade via ett grafiskt konsolfönster i Unreal Engine 4 editorn. Detta tillåter för snabb ändring mellan mängden fördröjning viket är essentiellt för detta projekt.

## 4.3 Arbetsprocess

### 4.3.1 Avstängning av fysiksimulering på klient

Det första problemet som behövde lösas var hur fysiksimuleringen av förstörbara objekt behövde stängas av på klienten. Då kollision av förstörbara delar behöver bibehållas på klientens sida går det inte endast att stänga av fysiksimuleringen utan objektet behöver ändras till att vara kinematic. Ett objekt som är kinematic i Unreal Engine 4 bibehåller sin kollision men alla rörelser som utförs styrs av kod eller animation.

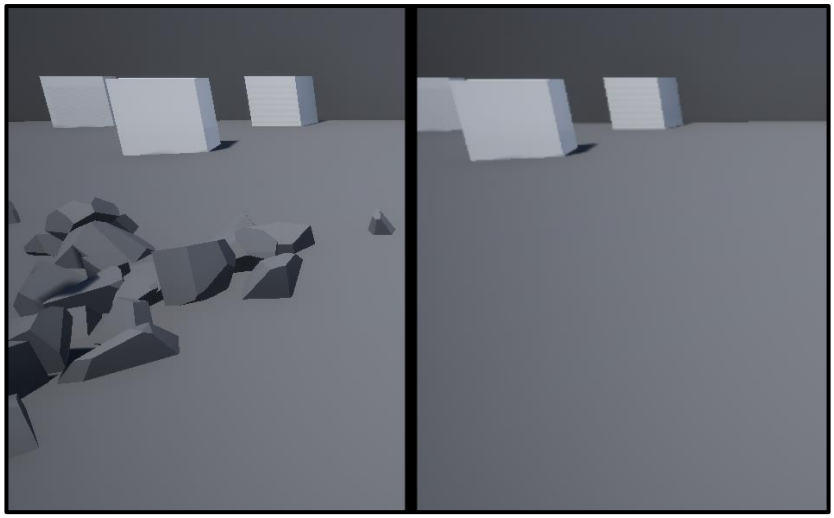
Kod som demonstrerar hur klienten stänger av fysiksimulationer på förstörbara objekt.

```
void UDummyDestructibleComponent::OnComponentFractureEvent(const FVector &HitPoint,
const FVector &HitDirection)
{
	for every chunk
	{
	   PxChunkActor->setRigidBodyFlag(physx::PxRigidBodyFlag::eKINEMATIC, true);
```

```
            PxChunkActor->setActorFlag(physx::PxActorFlag::eDISABLE_SIMULATION, false);
        }
}
```

Detta kodstycket visar hur klienten stänger av fysiksimulation på de förstörda delarna. När servern splittrar ett objekt kallar den på en funktion hos klienten vilket leder till att klienten också vet att ett objekt splittrats. Klienten använder sig av ett specialgjort förstörbart objekt som implementerar funktionen `OnComponentFractureEvent`. Detta event kallas på när ett förstörbart objekt splittras och används för att stänga av fysiken på de förstörda delarna hos klienten.

### 4.3.2 Uppdatering av position på klient



**Figur 9** Denna bild illustrerar hur förstörda delar inte renderas när de fortfarande är i vyn

För att uppdatera positionen av individuella delar användes funktionen `PxChunkActor->setGlobalPose`. Denna funktion uppdaterar position och rotation av en individuell del och löser förflyttning av kollision och delens grafiska representation. Det finns ett problem som denna funktion inte löser vilket är uppdatering av något känt som en bounding box. En bounding box används för att avgöra om ett objekt kan ses av kameran. Som illustrerat i Figur 9 uppdaterades inte denna bounding box och det går i bilden till höger se hur delarna inte renderas även om de är inom kamerans vy.

För att lösa detta användes en ny funktion för att flytta delar. Den funktion som användes var `PxChunkActor->setKinematicTarget`. Denna funktion, likt den som användes innan, flyttade delens kollision och grafiska representation men uppdaterade också bounding boxen. För att denna funktion skulle fungera behövdes en extra flagga sättas på delen när den gick sönder för att tillåta kinematisk förflyttning. För att utföra detta lades flaggan ”`PxChunkActor->setRigidBodyFlag(physx::PxRigidBodyFlag::eUSE_KINEMATIC_TARGET_FOR_SCENE_QUERIES, true);`” till i funktionen som stängde av fysiken när ett förstörbart objekt splittrats.

### 4.3.3 Alternativ implementation för Replikering av fysikdata

För att replikera fysikdata från server till klient utforskades ett flertal olika metoder. Den första metoden som användes var att skicka all data med hjälp av Unreal Engine 4 inbyggda RPC (Remote Procedure Call) (Epic Games u.å.b). En RPC används för att kalla på funktioner

över nätverk. I detta fall användes den för att kalla på en funktion på klienten som uppdaterade all fysikinformation.

Kod som visar hur en RPC funktion är definierad i Unreal Engine 4

```
UFUNCTION(NetMulticast, reliable)
void UpdateChunkInfo_Multicast(const TArray<FChunkInfo> &chunkInfo);
void UpdateChunkInfo_Multicast_Implementation(const TArray<FChunkInfo> &chunkInfo);
```

Som illustrerat i koden ovanför tillåter RPC funktionen parametrar. Det valdes därför att skicka all information angående de förstörda delarna som en parameter i denna RPC funktion. Det problem som uppstod var att mängden data som skickades genom dessa funktioner var för stor. Unreal Engine 4 väljer automatiskt vilken data den bör prioritera för att hålla mängden data som skickas över nätverket till ett minimum. Genom att skicka en så stor mängd data genom dessa funktioner kunde den inte längre prioritera rätt och saker som spelarens rörelse slutade replikeras medan fysikdata skickades. Av dessa anledningar kunde inte RPC användas utan paketen skickades inställt genom UDP. Genom att manuellt hantera paketstuktur och hur de skickades gick det att komma runt de begränsningar som finns i Unreal Engine 4:s RPC.

## 4.4 Pilotstudie

För att utvärdera genomförbarheten av studien utfördes en pilotstudie på två testpersoner. Testpersoner var båda mycket vana vid förstapersonsskjutarspel och hade inga större problem med att avklara testet. Vad som blev tydligt under detta test var hur spelare inte alltid förstod vad för delar som gick att göra sönder. För att lösa detta applicerades en speciell färg på alla objekt som inte gick att förstöra. Enkäten innehöll också ett par frågor som båda testpersonerna hade svårt att förstå vilket ledde till en finjustering av enkäten.

# 5 Utvärdering

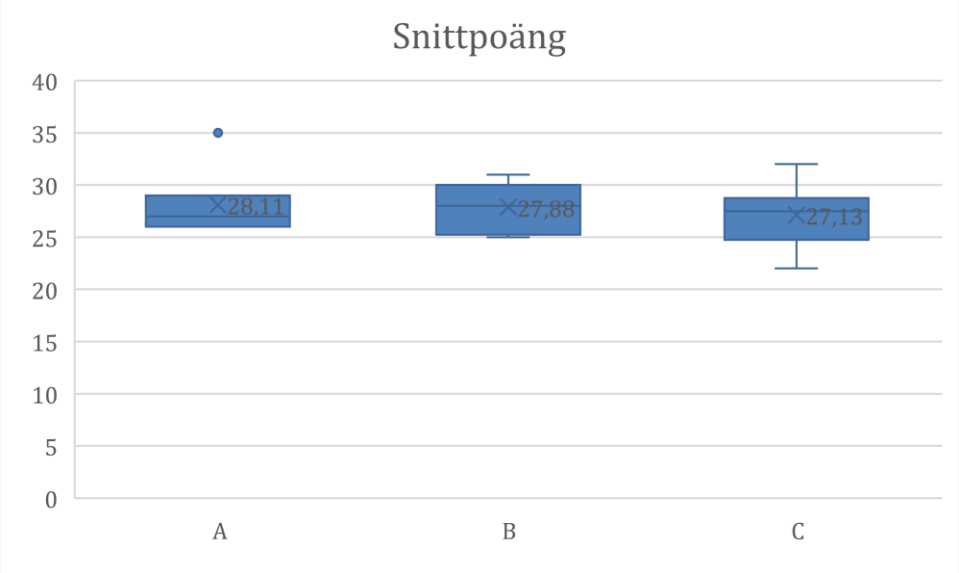
## 5.1 Presentation av undersökning

Undersökningen utfördes på gymnasieskolan Västerhöjd där testerna utfördes under 3 tillfällen utspritt över 3 dagar. Testpersonerna bestod av män i ålder 18 – 20. Då tre olika versioner av applikationen testades delades testpersonerna in i tre olika grupper. Varje person blev del av sin grupp baserat på vilken grupp testpersonen innan var del av d.v.s. att de olika grupperna cyklades igenom för att ge en jämn utspring av testpersoner. Dessa grupper kommer hädan efter refereras till som grupp A, B och C. Totals så utfördes testet på 25 personer där grupp A bestod av 8, grupp B av 8 och grupp C av 9 personer.

Innan testet utfördes av en testperson ändrades inställningar i applikationen beroende på vilken grupp testpersonen var del av. Testerna såg identiska ut för varje grupp, det som ändrades var mängden fördröjning mellan klienten och servern. Grupp A utförde testet utan någon fördröjning, grupp B med 100 millisekunder och Grupp C med 200 millisekunder. Innan testet startade instruerades deltagaren att försöka avklara testet så snabbt de kunde. Detta var på grund av att för varje testperson så mättes tiden det tog för dem att avklara testet. Efter att testet var avklarat fick de svara på en enkät. Denna enkät hade som mål att undersöka hur testpersonernas uppfattningar kring fysiksimuleringen ändrades med olika mängd fördröjning.

## 5.2 Resultat

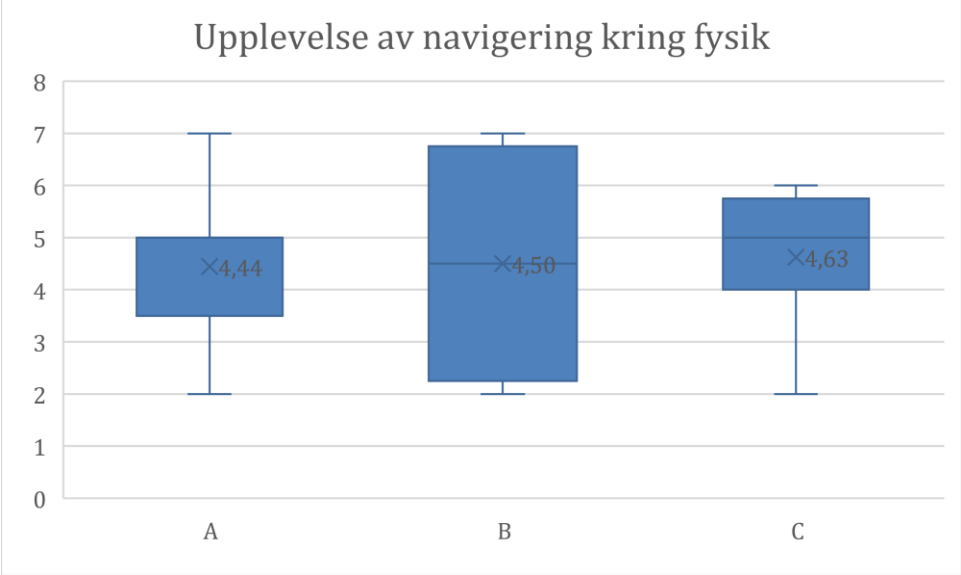
Varje testperson fick svara på en enkät där varje fråga hade ett värde mellan 1 - 5. Detta värde representerar hur testpersonen uppfattade en viss del av applikationen. Grafen i Figur 10 visar de sammanslagna värdena av alla frågor och hur de varierat i de olika grupperna.




**Figur 10** Snittpoäng för de olika grupperna för samtliga frågor (Högre är bättre)

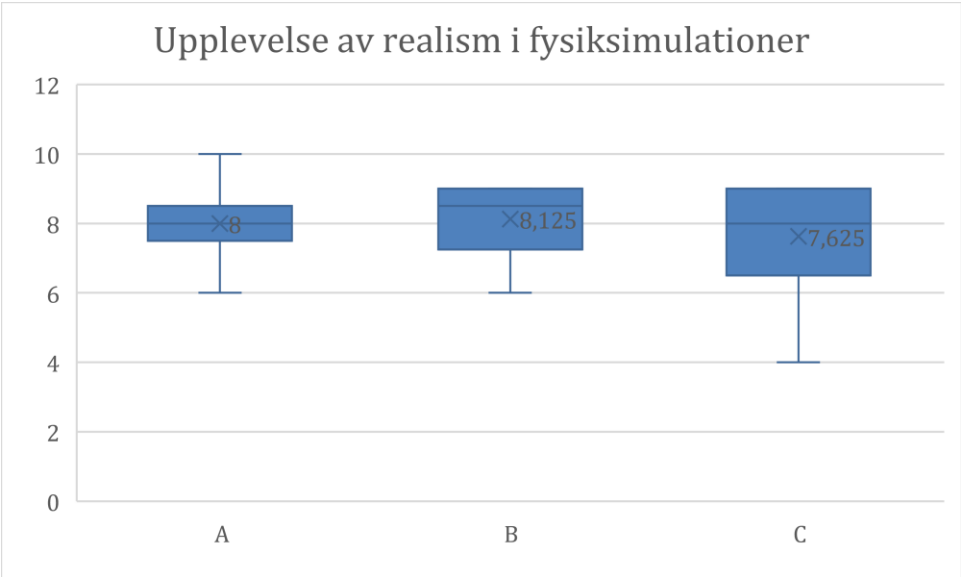
Vi kan i Figur 10 se att uppfattningen inte varierade stort mellan de olika grupperna. Det som tydligt uppstår är hur spridningen på svaren ökar i grupp B och C. Detta kan antyda att

upplevelsen blir mindre konsekvent med ökad fördröjning d.v.s. att grupp A som spelade utan fördröjning fick en mer likvärd upplevelse.




**Figur 11** Samanställning av data som undersöker hur testpersonerna upplevde navigering kring fysik (Högre är bättre)

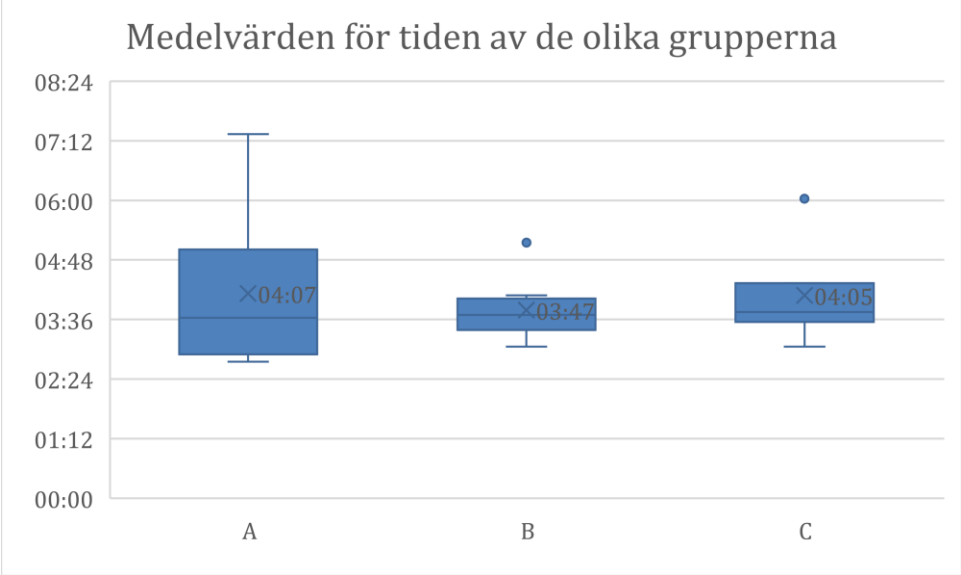
En del av enkäten undersökte hur testpersonen klarade av att navigera kring förstörda objekt. Frågorna från denna del har därför sammanställts till en egen graf som illustreras i Figur 11. Det går i denna graf återigen se att medelvärdet inte skiljer såg stort mellan de olika grupperna. Detta kan antyda att navigering kring fysik inte påverkades stort av fördröjningar i nätverket.




**Figur 12** Sammanställning av data som undersöker hur testpersonerna upplevde realismen i fysiksimulationerna (Högre är bättre).

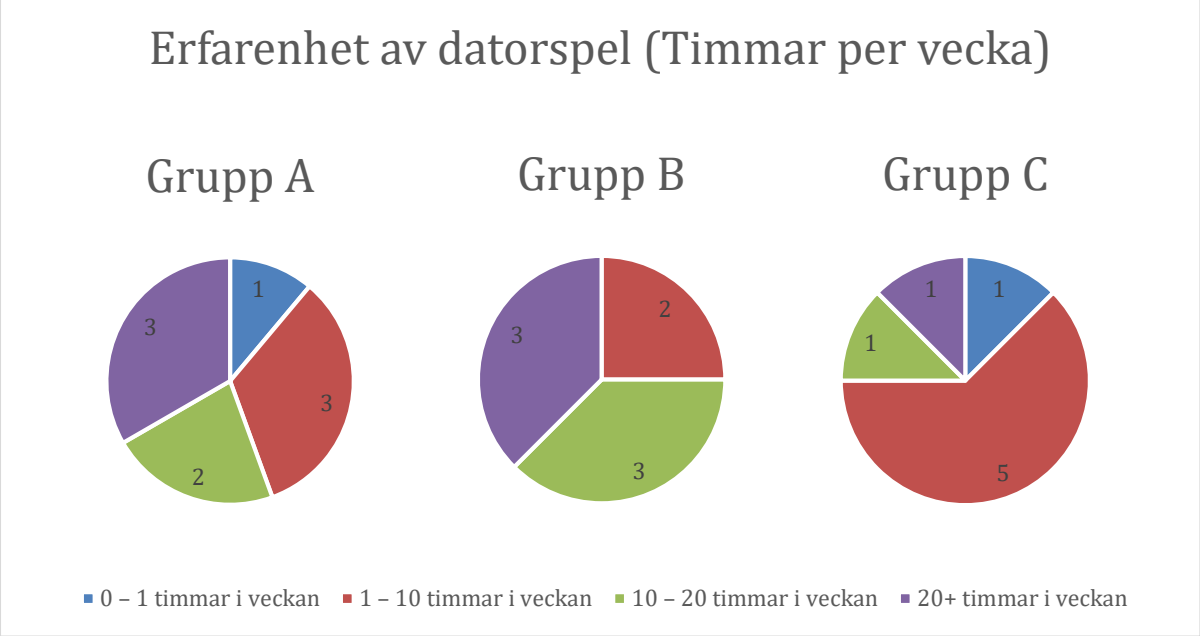
En annan del av enkäten undersökte hur pass testpersonen upplevde att fysiksimulationerna som skedde var realistiska. Frågorna från denna del har sammanställts till en egen graf som illustreras i Figur 12. Likt resultaten från snittpoängen verkar uppfattningen inte variera stort

mellan de olika grupperna. Det går dock i denna graf att se en märkbar försämring i resultaten från grupp C. Det som återigen tydligt uppstår är hur spridningen på svaren ökar i grupp B och C. Detta kan antyda att med ökad fördröjning kan fysiksimulationer ge en tydlig visuell försämring utan att påverka spelarens förmåga att interagera med fysiken. Utöver enkäten mättes också tiden varje testperson tog på sig för att avklara testet. Grafen i Figur 13 visar hur de olika grupperna presterade tidsmässigt i undersökningen.




**Figur 13** Sammanfattning av data som mäter tiden för de olika grupperna (Lägre är bättre)

Vi kan i Figur 13 se att medeltiden mellan de tre olika grupperna inte varierade mycket där grupp C, vilka hade de sämsta förhållanden, slår grupp A vilka hade de bästa. Detta tyder på att fördröjningen i fysiken inte påverkade testpersonernas förmåga att avklara testet.




**Figur 14** Sammanställning av erfarenhet inom datorspel i de olika

Figur 14 visar spridningen av erfarenhet inom dataspel i de olika grupperna. Grafen visar tydligt hur grupp A hade den högsta mängd spridning i erfarenhet vilket kan förklara varför tiderna i denna grupp innehöll både den bästa och den sämsta tiden. Grupp B hade lägre mängd spridning och var i snitt den mest erfarna gruppen. Detta kan förklara varför denna grupp fick den bästa tiden. Grupp C hade en låg mängd spridning men var i snitt den minst erfarna gruppen vilket förklarar varför tiden för denna grupp låg mellan A och B.

## 5.3 Analys

Det gick i enkäten se att testpersonernas uppfattningar kring spelet hade en mycket större spridning när fördröjningarna ökade. Den data representerad i Figur 12 antyder att spelare är mycket duktiga på att se när fysiken är fördröjd. Detta verkar dock inte påverka deras möjlighet att navigera de förstörda objekten vilket kan ses i Figur 11. Denna tes stärks av att testgruppernas tider inte hade stor variation. Som illustrerat i Figur 13 var den grupp med sämst tid den med lägst mängd fördröjning. Med hjälp av den data som visas i Figur 14 går det se att den grupp som hade bäst tid var den med mest erfarna testpersoner. Detta antyder att testpersonernas möjlighet att avklara testen inte var påverkade av fördröjning i fysiken utan den största faktorn var hur erfarna de var av spel sedan innan.

# 6 Avslutande diskussion

## 6.1 Sammanfattning

Målet med detta arbete var att undersöka hur fördröjningar inom fysiksimulationer påverkar spelares uppfattningar samt deras förmåga att prestera. Genom att undersöka detta går det att se hur väl fysiksimulationer kan distribueras över nätverk där fördröjningar är mycket vanliga. Då det avsedda användningsområdet är inom nätverk utvecklades en testmiljö där fysikdata skickades över riktiga nätverksprotokoll. Implementationen av arbetet genomfördes genom att skapa en testmiljö där fysiksimulationer beräknades på en dedikerad server som sedan skickar ut denna fysikinformation till den anslutna klienten. Denna testapplikation innehöll ett flertal delar där testpersonen behövde interagera med fysikobjekt.

Utvärderingen av arbetet utfördes genom att hålla testtillfällen där testpersoner testade det utvecklade programmet. Efter varje testtillfälle fick testpersonen svara på en enkät som utvärderade deras uppfattningar kring spelet. Varje testperson fick också som anvisning att avklara testet så fort de kunde där dess tid mättes för att utvärdera prestandan. Testpersonerna delades in i tre grupper med olika mängd fördröjning. Det kom fram till att prestandan hos testpersonerna inte påverkades nämnvärt vid ökning av fördröjning men deras uppfattningar kring fysiken hade en märkbar försämring.

## 6.2 Diskussion

Likt Claypool och Finkel (2014) så gick det att se en tydlig försämring i testpersonernas uppfattningar kring applikationen när fördröjningen ökade. Deras undersökning noterade en tydlig försämring av upplevelsen vid ökad fördröjning. Det noterades också att vid en fördröjning över 200ms ansågs inte längre spelupplevelsen acceptabel för användare. Arbete som nu utförts noterade också en försämring av upplevelsen men den försämring som noterades var mycket låg och det finns en stor sannolikhet att applikationer hade fortsatt att prestera bra vid en högre mängd fördröjning än 200ms. Claypool och Finkel (2014) antydde också på en 25% försämring av spelarprestanda för varje 100ms av fördröjning jämfört med denna studie som inte kunde hitta någon korrelation mellan fördröjning och spelarprestanda.

Denna studie hade som mål att undersöka om fördröjningar i fysiksimulation över nätverk påverkade dels spelares uppfattningar kring spelet samt dess förmåga att avklara spelet. Med hjälp av den data som samlats gick det att konkludera att spelarens förmåga att avklara spelet inte påverkades med dessa fördröjningar men det gick att se en märkbar försämring i testpersonens uppfattningar kring applikationen med ökad fördröjning. Detta arbete har därav besvarat de frågor som angavs. Även om den metod som använts gav tillfredsställande svar så bör det anmärkas att andra metoder kan användas som kan ge likvärdigt svar. Ett exempel hade varit att använda sig av redan existerade applikationer och undersöka hur fördröjningar i delar av dessa påverkade spelare.

### 6.2.1 Begränsningar

Det resultat som denna studie kommit fram till kan bero på ett flertal faktorer. Ett exempel är testmiljön som använts. Testmiljön hade till exempel kunnat inkludera en klocka för att öka stressfaktorn i testet. Detta hade kunnat leda till att spelaren blivit mer frustrerad om fysiken inte agerat som förväntat. Denna testmiljö var också begränsad till första persons vy och det

går därför inte att dra någon slutsats om hur andra perspektiv hade påverkats av fördröjningarna.

Många designbeslut som togs kan ha påverkat det slutgiltiga resultatet. En helt annan uppsättning av pussel som testpersonerna hade behövt lösa hade kunnat ge olika resultat. Hade det till exempel skapats pussel där spelaren var tvungen att interagera med fysiken på en mer komplex nivå så är det mycket möjligt att fördröjningar i nätverket hade blivit mycket mer tydliga för spelaren. Exempel på ett sådant pussel kan vara en del där spelaren måste plocka upp och flytta på delar manuellt och därav öka interaktionen mellan fysik och spelare. För att få bättre resultat hade ytterligare studier behövts genomföras med olika uppsättningar av pussel.

Det är också viktigt att påpeka att testgrupperna bestod av en homogen grupp av studenter. Det går därför inte dra någon slutsats om hur personer i andra grupper av samhället hade presterat i testet. Storleken på grupperna var också mycket begränsad vilket kan leda till mindre precis data. För att eliminera de oförutsägbara variabler som introduceras med nätverk simulerades både server och klient på samma maskin med en artificiell fördröjning mellan dess kommunikation. Denna simulerade fördröjning representerar inte fullt hur fördröjningar sker över riktiga nätverk. Ett exempel är hur fördröjningen i denna studie aldrig varierade utöver den bestämda fördröjningen medan i ett riktigt nätverk har fördröjningar en tendens att fluktuera. För att ge ett mer tillämpat resultat skulle studien behöva utföras över ett riktigt nätverk där server och klient simuleras på olika maskiner.

### 6.2.2 Etiska aspekter

En aspekt som alltid behöver tas hänsyn till när ett test utförs på människor är lärande effekten. För denna studie valdes detta att undvikas genom att endast tillåta varje testperson att utföra testen en gång. Alternativa tillvägagångssätt hade kunnat applicerats där varje testperson fick utföra testet ett flertal olika gånger med olika mängd fördröjning där de olika resultaten sedan jämfördes med varandra. I ett test som detta finns alltid risken att testpersonen lär sig hur testet går till och det skulle därför behövas skapas nya pussel för varje testomgång. För att undvika urvalsfel valdes versionen som spelades delvist slumpmässigt. Testpersonerna fick inte heller veta vilken grupp de tillhörde eller ens att det existerade olika grupper. Som med alla frivilliga studier fanns problemet där de testpersoner som var mer intresserade av testämnet var mer villiga att utföra testet. Detta ledde till att den större delen av testpersonerna var mycket erfarna inom dataspel vilket han ha förvrängt resultaten.

Storleken på testgrupper var mycket begränsad vilket gör den insamlade data mindre pålitlig. Det finns också möjlighet att testpersonerna klurade ut vad arbetet handlade om innan de utförde testet genom konversation mellan testpersoner vilket kan leda till att dess svar på enkäten ändrades. Även om arbetet utförts efter bästa förmåga kan det fortfarande tillkomma oupptäckta misstag. Dessa misstag är dock samma för alla i testet och bör därför minimera dess effekt på det slutgiltiga resultatet.

All forskning presenterad i denna studie vilket inte var utförd av författaren av denna rapport refererar till dess ursprungliga upphovsman. Det vill också nämnas att alla test utförda var helt frivilliga där testpersonerna hade rätt att avsluta testet när de ville. De var också helt medvetna om att detta var fallet. Ingen skada tillfogades på testpersonerna. Alla tester utfördes under uppsyn av personal inom anläggningen. Denna studie går bra att återupprepa

då all den data som samlats samt enkäten som använts finns redovisad i Appendix A och B. Centrala delar av applikationen beskrivs också i implementationsdelen av denna rapport.

### 6.2.3 Samhällelig nytta

Molntjänster är en stor industri med stor potential att växa inom de närmaste åren och ett arbete som detta lägger grunden för att skapa mer distribuerade applikationer inom spel. Det går redan idag att ta vara på molnets kapacitet vilket till exempel görs av Microsofts Crackdown 3 (Robertson, 2015). Genom att dela upp en applikation så att olika delar kan distribueras på olika delar av nätverket går det att placera de arbetsuppgifter som kräver låg responstid nära klienten och de andra delarna på noder längre bort. Detta kan leda till en smart distribuerad applikation som kan kalkylera de mindre viktiga, men fortfarande tunga, arbetsuppgifterna på ett stort serverkluster långt bort men fortfarande behålla hög responstid genom att beräkna de essentiella arbetsuppgifterna nära klienten.

Ett arbete som detta kan också hjälp till med att skapa mer realistiska simulatorer genom att tillåta simulering på externa superdatorer för de delar av applikationen som inte är lika beroende av responstid. Ett exempel på ett tillfälle som detta är hur det skulle gå att simulera hur en byggnad faller ihop när den brinner vilket kan användas inom brandsimulationer för brandmän. I ett fall som detta är det okej om resultatet av simulationen kommer lite senare då ett scenario som detta inte är mycket beroende av responstid.

## 6.3 Framtida arbete

Då testerna utfördes på en grupp med mycket begränsad storlek hade detta arbete behövt testas på en större mängd testpersoner för att stärka denna rapports slutsatser. Testgruppen var också mycket homogen och representerar därför inte beteendet av alla grupper av samhället. Det hade därför varit ett bra framtida arbete att utföra testerna igen med en större, mindre homogen, testgrupp.

Denna studie testade endast simulering av fysik i relativt enkla förhållanden. Förstörbara objekt hade ett förberäknat sätt att splittras på vilket drastiskt minskade komplexiteten av arbetet. Ett framtida arbete hade kunnat vara att applicera en dynamisk metod för att splittra fysikobjekt. I ett läge som detta skulle serven kunna hantera belastningen av att splittra geometrin och sedan skicka resultatet till klienten.

Något som tas upp i diskussionen (kapitel 6.2) är hur simuleringen endast sker lokalt. Ett framtida arbete hade kunnat vara att testa denna applikation över olika nätverk för att se hur olika förhållanden påverkar resultaten. Ett exempel på ett sådant test är om fluktuerande fördröjningar är värre för spelaren än en hög konstant fördröjning. Det hade också gått att designa denna applikation för att fungera i ett distribuerat nätverk. I ett nätverk som detta kan olika noder i nätverket ansvara för delar av fysiksimulationen. Detta betyder att all simulering inte är beroende av prestandan av en centraliserad nod. Ett exempel på ett nätverk som detta är ett peer to peer nätverk där varje ansluten klient har hand om en liten del av fysiksimulationerna.

Det skulle också vara mycket intressant att se om det går att fördröja andra delar av applikationen. Ett exempel på detta skulle vara om en extern server skulle kunna hjälpa till med kollisionsberäkningar eller hantera mer komplex fysik som till exempel fysikbaserade fordon. Om flera delar av applikationen skulle kunna brytas ut ger det möjlighet att testa dess

potential inom en molntjänstmiljö där de olika delarna av applikationen kan distribueras på olika delar av ett serverkluster.

# 7 Referenser


Boeing, A., Bräunl, T. (2007). Evaluation of real-time physics simulation systems. *Computer graphics and interactive techniques in Australia and Southeast Asia* (ss. 281-288 ). University of Western Australia, Perth: ACM.

bulletphysics. (u.å). *bulletphysics*. Hämtat från bulletphysics.org: http://www.bulletphysics.org/mediawiki-1.5.8/index.php/Maya_Dynamica_Plugin [2017-06-20]

Claypool, M. (2006). Latency and player actions in online games. *Communications of the ACM - Entertainment networking*, Volym 49, utgåva 11, Sidor 40-45.

Claypool, M. (2016). *On Models for Game Input with Delay — Moving Target Selection with a Mouse*. Worcester, MA, USA: IEEE.

Claypool, M., Finkel, D. (2014). *The effects of latency on player performance in cloud-based games*. MA, USA: IEEE.

Epic Games (u.åa). *Actor Replication*. Hämtat från: unrealengine.com: https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/index.html [2017-07-14]

Epic Games (u.åb). *PRCs* Hämtat från: unrealengine.com: https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/RPCs/index.html [2017-07-14]

Epic Games (u.åc). *Physics Simulation* Hämtat från: unrealengine.com: https://docs.unrealengine.com/latest/INT/Engine/Physics/ [2017-07-14]

Kollar, P. (2016). *THE PAST, PRESENT AND FUTURE OF LEAGUE OF LEGENDS STUDIO RIOT GAMES*. https://www.polygon.com/2016/9/13/12891656/the-past-present-and-future-of-league-of-legends-studio-riot-games [2017-09-13]

Parker, E.G., O'Brien, J.F. (2009). *Real-time deformation and fracture in a game environment*. *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (ss. 165-175). New Orleans, Louisiana: ACM.

PlayOverwatch (2017). More than 30 million players have charged into Overwatch! Thanks for grouping up with us, heroes. We couldn't ask for a better team. [twitterpost], 12 maj. https://twitter.com/PlayOverwatch/status/858002935820582912 [2017-04-28]

Robertson, J. (2015). *How Crackdown 3 uses the cloud to make whole cities destructible* https://arstechnica.com/gaming/2015/08/how-crackdown-3-uses-the-cloud-to-make-whole-cities-destructible/ [2015-06-08]

Sony. (2017). *Playstation*. Hämtat från playstation.com: https://www.playstation.com/sv-se/explore/playstation-now/ [2017-07-14]

Suznjevic, M., Matijasevic, M. (2015). *Trends in evolution of the network traffic of Massively Multiplayer Online Role-Playing Games. 2015 13th International Conference on Telecommunications (ConTEL)*, Graz, Austria: IEEE. [2015-09-03]

Zamith, M., Joselli, M., Walter, E., Clua, G., Montenegro, A., Leal-Toledo, R.C.P., Valente, L. (2011). *A Distributed Architecture for Mobile Digital Games Based on Cloud Computing*. IEEE.

# Appendix A – Enkät

Frågor markerade med en * i början är inverterade och dess poäng inverterades innan de lades med i beräkningarna.

## Exjobb 2017 Enkät

*Required

1. Grupp * Mark only one oval.

   - A
   - B
   - C

2. Tid *

3. Hur många timmar per vecka spenderar du på att spela datorspel? * Mark only one oval.

   - 0 – 1 timmar i veckan
   - 1 – 10 timmar i veckan
   - 10 – 20 timmar i veckan20+ timmar i veckan
   - Option 4

4. Hur stor del av denna tid spenderar du på förstaperson-skjutarspel (FPS)? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Spelar aldrig FPS | | | | | | Spelar endast FPS |

5. Hur roligt tyckte du spelet va? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Mycket tråkigt | | | | | | Mycket rolig |

6. Hur skulle du beskriva kontrollschemat? * Mark only one oval.

1 2 3 4 5

| | | | | | | |
|---|---|---|---|---|---|---|
| Mycket dåligt | ◯ | ◯ | ◯ | ◯ | ◯ | Mycket bra |

7. * Blev du någon gång under testningen frustrerad? *
   Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Blev mycket frustrerad en eller flera gånger | ◯ | ◯ | ◯ | ◯ | ◯ | Blev inte alls frustrerad |

8. * Hur svårt tycket du det var att klara av spelet? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Mycket svårt | ◯ | ◯ | ◯ | ◯ | ◯ | Inte alls svårt |

9. Hur funktionellt skulle du säga att detta spel var? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Många delar/hela spelet var icke funktionella | ◯ | ◯ | ◯ | ◯ | ◯ | Spelet var i sin helhet mycket funktionellt |

10. Hur realistisk upplevde du simuleringen av fysiken? *
    Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Mycket orealistisk | ◯ | ◯ | ◯ | ◯ | ◯ | Mycket realistisk |

11. Hur realistiska upplevde du explosionerna av skotten? *
    Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Mycket orealistiska | ◯ | ◯ | ◯ | ◯ | ◯ | Mycket realistiska |

12. * Hur svårt tyckte du det var att navigera kring förstörda objekt? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Mycket svårt | ◯ | ◯ | ◯ | ◯ | ◯ | Inte alls svårt |

13. * Undvek du under någon del av spelet att skjuta/interagera med förstörbara objekt? * Mark only one oval.

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Ja, interaktionen var opålitlig så jag undvek detta så ofta jag kunde | | | | | | Nej, fanns ingen anledning att göra detta |

14. Vad tror du denna studie undersöker? (Frivillig)

Powered by Google Forms

# Appendix B – Enkätsvar

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | 20+ timmar i veckan | 4 | 5 | 5 | 5 | 5 | 5 | 4 | 3 | 4 | 5 | 03:23 |
| B | 10 – 20 timmar i veckan | 4 | 4 | 5 | 5 | 5 | 5 | 4 | 4 | 2 | 4 | 03:38 |
| C | 10 – 20 timmar i veckan | 5 | 5 | 5 | 5 | 5 | 4 | 5 | 3 | 3 | 4 | Trasig Timer |
| A | 10 – 20 timmar i veckan | 4 | 5 | 5 | 3 | 4 | 5 | 5 | 5 | 3 | 4 | 03:38 |
| B | 20+ timmar i veckan | 5 | 4 | 4 | 5 | 5 | 4 | 4 | 4 | 2 | 5 | 03:50 |
| C | 1 – 10 timmar i veckan | 1 | 2 | 4 | 5 | 5 | 5 | 4 | 3 | 5 | 5 | 04:20 |
| A | 0 – 1 timmar i veckan | 1 | 3 | 4 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 07:20 |
| C | 1 – 10 timmar i veckan | 3 | 4 | 4 | 5 | 5 | 4 | 4 | 4 | 5 | 1 | 03:44 |
| A | 20+ timmar i veckan | 4 | 4 | 5 | 5 | 4 | 5 | 3 | 3 | 4 | 5 | 02:45 |
| B | 1 – 10 timmar i veckan | 2 | 4 | 4 | 2 | 2 | 4 | 2 | 2 | 2 | 3 | 05:09 |
| B | 1 – 10 timmar i veckan | 2 | 3 | 4 | 5 | 4 | 5 | 4 | 3 | 4 | 5 | 03:45 |
| C | 1 – 10 timmar i veckan | 4 | 4 | 5 | 5 | 4 | 2 | 2 | 3 | 3 | 4 | 04:09 |
| A | 20+ timmar i veckan | 4 | 4 | 4 | 1 | 5 | 4 | 3 | 3 | 3 | 4 | 02:53 |
| B | 10 – 20 timmar i veckan | 3 | 4 | 5 | 3 | 4 | 4 | 3 | 4 | 4 | 4 | 04:05 |
| C | 1 – 10 timmar i veckan | 2 | 4 | 4 | 3 | 3 | 5 | 4 | 4 | 2 | 5 | 03:33 |
| A | 1 – 10 timmar i veckan | 3 | 3 | 5 | 5 | 4 | 5 | 3 | 3 | 3 | 5 | 04:14 |
| B | 20+ timmar i veckan | 4 | 3 | 3 | 4 | 5 | 5 | 4 | 5 | 5 | 5 | 03:22 |
| C | 20+ timmar i veckan | 3 | 4 | 5 | 4 | 5 | 5 | 3 | 3 | 4 | 4 | 03:03 |
| A | 1 – 10 timmar i veckan | 2 | 4 | 4 | 3 | 3 | 4 | 2 | 2 | 3 | 4 | 05:02 |
| B | 10 – 20 timmar i veckan | 4 | 5 | 4 | 5 | 5 | 5 | 4 | 4 | 5 | 5 | 03:28 |
| C | 0 – 1 timmar i veckan | 1 | 3 | 4 | 1 | 3 | 4 | 4 | 1 | 3 | 5 | 03:45 |
| A | 1 – 10 timmar i veckan | 4 | 5 | 3 | 5 | 4 | 5 | 3 | 4 | 4 | 3 | 02:55 |
| B | 20+ timmar i veckan | 3 | 3 | 5 | 4 | 5 | 4 | 4 | 4 | 3 | 2 | 03:03 |
| C | 1 – 10 timmar i veckan | 4 | 4 | 3 | 5 | 3 | 3 | 3 | 4 | 3 | 3 | 06:02 |
| A | 10 – 20 timmar i veckan | 4 | 3 | 2 | 4 | 4 | 4 | 4 | 3 | 2 | 3 | 04:59 |