Examensarbe

HÖGSKOLAN I SKÖVDE

DISKRET OCH KONTINUERLIG KOLLISIONSDETEKTERING

En kvantitativ studie i prestandaskillnad

DISCRETE AND CONTINUOUS COLLISION DETECTION

A quantitative study of difference in performance

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 15 högskolepoäng
Vårtermin 2025

Marcus Leidefalk
Patrick Johfur

Handledare: Mikael Thieme
Examinator: Mikael Johannesson

Sammanfattning

Det finns spel inom bland annat realtidsstrategi-genren där stora strider ska simuleras med många soldater som skjuter på varandra. Detta kräver att en stor mängd kollisioner måste detekteras i realtid. Här kan det för spelutvecklare uppstå problem eftersom olika metoder för detektering av kollisioner kräver olika mycket beräkningstid, samtidigt som de olika metoderna har olika stor risk för att missa kollisioner.

Denna kvantitativa studie har jämfört diskret och kontinuerlig kollisionsdetektering mot varandra i tre scenarion i Unity, baserade på tre former av strategispel för att ta reda på vad skillnaden i prestanda blir. Testerna har genomförts på en dator. Resultatet visar att den kontinuerliga metoden är tydligt dyrare i alla tre scenarion.

Framtida arbeten kan utvidga kunskapen på detta område genom att göra test på fler datorer och i andra mer spel-liknande scenarion. En studie som jämför hur många kollisioner som kan missas med diskret kontra kontinuerlig är också intressant.

Nyckelord: Fysik, Unity, Spelprogrammering, Prestanda, Kollisionsdetektering, Informationsteknologi

Innehållsförteckning

1 Introduktion

Förbättring av prestanda är ett viktigt område inom spelutveckling. Spel med dålig prestanda leder till sämre spelupplevelse för användaren, till exempel att spelet uppdateras med en för låg bildfrekvens i ett spel med dålig prestanda och därmed upplevs som frustrerande och svårstyrt, jämfört med ett spel med bra prestanda. Spelutvecklare behöver utveckla spel med bra prestanda för att spelet ska vara kommersiellt gångbart. Det finns olika anledningar till att ett spel kan ha dålig prestanda. En av anledningarna kan vara tidskrävande algoritmer, såsom kollisionsdetektering.

Ett system för kollisionsdetektering tar hand om spelets kollisioner, till exempel för att förhindra att en karaktär kan gå igenom en vägg. Huruvida kollisionsdetekteringen ger en märkbar påverkan på prestanda, beror på hur den är implementerad. Två metoder som används för att implementera kollisionsdetektering är diskret detektering och kontinuerlig detektering. Diskret kollisionsdetektering kontrollerar kollisioner per uppdatering i spelet. Detta är en enkel metod, men har ökad chans att kollisioner kan missas. Kontinuerlig kollisionsdetektering interpolerar mellan två uppdateringar, för att se vilka objekt på banan som objektet har kolliderat med. Denna metod ger ökad chans för att kollisioner upptäcks, men ökar belastningen på processorn för att förutspå framtida kollisioner.

Målet med denna studie är att jämföra prestanda mellan diskret och kontinuerlig kollisionsdetektering. Studien genomförs genom att skapa en testbana där två arméer står mot varandra och skjuter projektiler på den motsatta armén. Det finns tre scenarier i testbanan, skillnaden mellan de olika scenarierna är arméstorleken och de totala projektiler som finns i banan. Varje scenario körs två gånger, en gång för diskreta och en gång för den kontinuerliga metoden. Tidsåtgången för fysik-uppdateringen, vilket då inkluderar kollisionsdetektering, samlas in som data.

2 Bakgrund

I de flesta spel och simuleringar där figurer i en två- eller tredimensionell värld interagerar med varandra behöver deras kollisioner med varandra registreras. Det finns en del spelgenrer där antalet figurer kan bli förhållandevis stort, som t.ex. strategispel som ska simulera stora medeltida fältslag med tusentals soldater som slåss mot varandra, eller i skjutspel där antalet fiender ökar ständigt för att göra det svårare. Ett större antal fiender innebär ett större antal objekt att göra kollisionskontroll mot. Ska spelets fiender dessutom skjuta projektiler så blir varje projektil ett objekt för sig som det måste genomföras kollisionsdetektering på, förutsatt att avfyrade projektiler ska kunna undvikas efter att de avlossats.

I spel som *Risk of Rain 2* (2020), *Total War*-serien (2025) och *Ultimate Epic Battle Simulator* (2023) simuleras stora strider där ett stort antal aktörer och/eller projektiler är inblandade. *Total War*-serien och *Ultimate Epic Battle Simulator* är ett strategispel där spelaren får styra arméer i stora fältslag. Se bild 1 nedan för ett exempel på ett stort fältslag från *Ultimate Epic Battle Simulator*.

Bild 1 *Ultimate Epic Battle Simulator (Wong, 2021)*

Risk of Rain 2 är ett *shooter*-spel där spelaren kan samla på sig olika föremål för att öka sin styrka, vilket kan leda till att spelaren t.ex. kan skjuta ett stort antal fler projektiler, samtidigt som det blir svårare och svårare med fler och fler fiender. Detta kan leda till situationer där en till slut har en spelomgång igång där skärmen är helt fylld av projektiler, explosioner och fiender.

Denna sortens spel ställer därmed krav på spelutvecklaren att använda algoritmer som är tidseffektiva. Det kan handla om att utvecklaren skapar algoritmen själv, eller konfigurerar ett färdigt fysik-system. Ett ökande antal fiender och projektiler i det pågående spelet kan kraftigt öka antalet beräkningar som behöver göras, detta riskerar att sakta ner uppdateringsfrekvensen och få spelet att sakta ner.

I denna studie kommer ordet latens användas för att beskriva tidsåtgång från en uppdatering till en annan, vilket påverkar tiden det tar för input av spelare att bli output på skärmen, och hur pass snabbt bilden på skärmen uppdateras. Detta påverkas av algoritmernas tidsåtgång för att genomföra beräkningar i spelet. Nedan beskrivs ett antal studier som visar varför dålig prestanda, som tar sig i uttryck i form av hög latens, negativt påverkar spelare. Både gällande möjligheten att styra spelet, och även deras självupplevda upplevelse av spelet.

MacKenzie & Ware (1993) visar i sin studie att ett system med hög latens blir svårare att interagera med för människor. Studien är från tidigt 1990-tal, men tekniken som användes då var en datormus, vilket fortfarande används fast i mer utvecklad form. När de testade att öka latensen på test-subjekten så tog de mycket längre tid på sig att genomföra testet och väsentligt fler felaktiga musklick gjordes. Wilcox et al. (2015) visar i sin studie att även inom film så föredrar tittare en högre bildfrekvens. Specifikt så är det målsökande i spel, där precision krävs som mest påverkas av en lägre bildfrekvens (Claypool & Claypool, 2007), detta kan t.ex. vara att sikta och skjuta på ett mål i ett skjutspel, eller att markera olika objekt för att ge orders i ett strategispel där snabba beslut krävs för att vinna.

Claypool et al. (2020) undersökte också effekterna av latens på spelare, med fokus på spel. I deras studie undersöktes en grupp för att mäta hur deras träffsäkerhet påverkades. I studien utvecklades ett spel som hette *Juke!* som gick ut på att spelaren styrde en blå pil och skulle klicka på en röd boll som åkte runt på skärmen. Spelet lade till latens på musens rörelser. Även i denna studie var tid och träffsäkerhet i form av avstånd de två variablerna som mättes. Kanske föga förvånande så tog personerna längre tid på sig och hade sämre träffsäkerhet vid ökad latens.

Liu et al. (2021) gjorde en studie på en grupp spelare med erfarenhet inom spelet *Counter-Strike: Global Offensive*. Där de fick spela med olika hög latens. Deltagarna märkte en betydlig försämring av upplevelsen och deras egna prestation vid högre latens. En tydlig problematik kan då uppstå när det finns spel, som till exempel de som tidigare nämnts här, som blir svårare av att det blir fler aktörer och objekt i spelet att till exempel slåss mot. Detta betyder att svårighetsgraden stiger samtidigt som antalet beräkningar av kollisioner blir fler, och riskerar att sänka bildfrekvensen när det utifrån spelarens perspektiv blir som viktigast att spelet är så responsivt som möjligt.

Två metoder för detektering av kollisioner är diskret detektering och kontinuerlig detektering. Den diskreta metoden går ut på att vid varje uppdatering jämföra olika fysik-objekt med varandra, för att se om de kolliderar med varandra (*Discrete Collision Detection*, 2025). Fysik-objekt eller objekt betyder i denna text digitala representationer av objekt som till exempel soldater och projektiler i ett spel. Den kontinuerliga metoden går ut på att interpolera objekten från föregående uppdatering till nuvarande uppdatering, för att se om de på vägen från den ena uppdateringen till den andra kolliderat med varandra (*Sweep-Based CCD*, 2025). Den förstnämnda metoden kan därmed missa kollisioner som skulle ha skett mellan uppdateringarna. Nedan bild illustrerar skillnaden i de två metoderna.

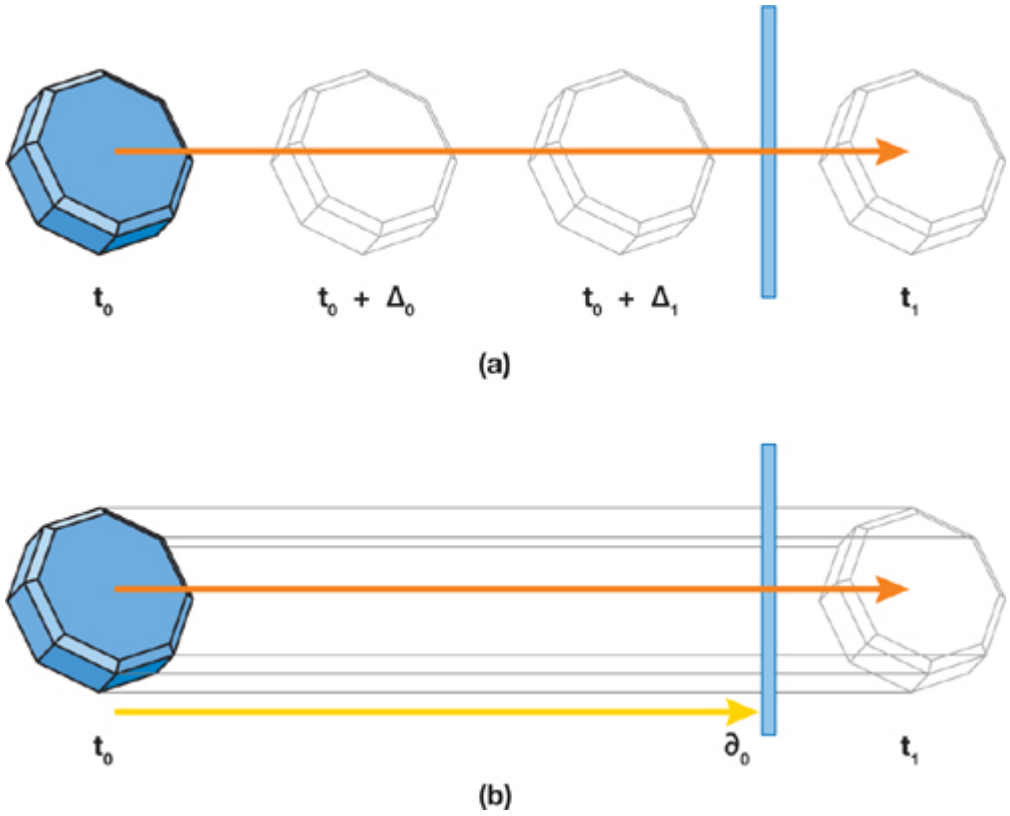

Bild 2 *Illustration av metoder för detektering av kollisioner* (*Collision Detection*, 2025). *(a) är diskret detektering och (b) är kontinuerlig.*

3 Problemformulering

Eftersom kollisionsdetektering av många objekt i spel kan leda till ett ökande antal beräkningar, kan prestandan påverkas. Hög latens och låg bildfrekvens har visats leda till sämre prestation och upplevelse hos spelare som visats av (Liu et al., 2021), vilket kan göra detta till ett kritiskt problem i spel med intensiva strider. Att klicka på saker i spel blir helt enkelt svårare vid högre latens (Claypool et al., 2020), vare sig det handlar om att snabbt välja enheter i ett strategispel, eller träffa rätt i ett skjutspel.

Målet med projektet är att jämföra prestandaskillnader mellan olika metoder för kollisionsdetektering som finns i en högintensiv spelmiljö med många projektiler. För att se vilka scenarion där diskret eller kontinuerlig kollisionsdetektering är bäst lämpade. Med högintensiv spelmiljö menas ett scenario med många kollisioner som sker konstant, i nära anslutning till varandra, vilket lägger en större press på fysik-systemet att hitta alla möjliga kollisioner. I detta specifika fall blir det två armeér som står och skjuter på varandra konstant. Skillnaden i hur pass många kollisioner som upptäcks eller missas är därmed utanför ramen för denna studie. Med prestanda menas här beräkningstiden för fysik-uppdatering, och inte andra områden som till exempel minne eller grafik.

Därmed blir frågan:

> Hur skiljer sig prestandan mellan diskret och kontinuerlig kollisionsdetektering i en högintensiv spelmiljö?

Resultatet av studien ämnar ge beslutsstöd till utvecklare som behöver balansera prestanda och resursförbrukning i simuleringar där kollisionsdetektering behöver genomföras på många rörliga objekt, exempelvis realtidsstrategispel med många soldater som skjuter många projektiler på varandra. Närmare förklaringar av begreppen diskret och kontinuerlig kollisionsdetektering följer i avsnitt 3.1.1.

3.1 Metodbeskrivning

Denna del förklarar den metod som har valts för att genomföra studien. Bland annat den typ av experiment som har valts ut för att samla in data och datorprogram som använts för att utföra experimentet.

3.1.1 Unity

Som en del av experimentet kommer spelmotorn *Unity* (*Unity Engine*, 2024) att användas för att simulera kolliderande fysik-objekt. *Unity* erbjuder färdiga lösningar för att hantera kollisionsdetektering i form av systemen *rigidbody* och *colliders* (*Rigidbody*, 2025). *Rigidbody* är en komponent i *Unity* som ger ett spelobjekt tillgång till fysik-motorn. *Collider* är en komponent i *Unity* som ger ett spelobjekt möjlighet för att utföra kollisionsdetektering. Dessa komponenter kan läggas på ett objekt (*GameObject*) och då hantera objektets kollisionsdetektering och fysikberäkningar. *Rigidbody* erbjuder fyra olika metoder för kollisionsdetektering, nämligen *Discrete*

(diskret kollisionsdetektering), *Continuous* (kontinuerlig kollisionsdetektering mot andra statiska objekt), *Continuous Dynamic* (kontinuerlig kollisionsdetektering mot både statiska och dynamiska objekt) och *Continuous Speculative* (kontinuerlig spekulativ kollisionsdetektering). *Discrete* söker efter kollisioner varje fysik-uppdatering (som heter *FixedUpdate* i *Unity*), vilket medför en risk för att kollisioner som skulle skett mellan två fysik-uppdateringar missas. För att undvika dessa missar finns det *Continuous*, denna metod går ut på att interpolera över sträckan som entiteten färdas mellan fysik-uppdateringarna för att se om det förekommer några kollisioner med andra statiska objekt. Detta är en dyrare beräkning. Den dyraste beräkningen sker i *Continuous Dynamic*, eftersom den utöver statiska entiteter, även kontrollerar entiteter som är dynamiska. I studien undersöks metoderna *Discrete* som benämns diskret och *Continuous Dynamic* som benämns kontinuerlig. Anledningen att just dessa två väljs är eftersom *Discrete* är den minst prestandakrävande och *Continuous Dynamic* är den mest prestandakrävande. Med diskret menas därmed *Discrete* och kontinuerlig menas *Continuous Dynamic*.

3.1.2 Typ av experiment

Metoden för denna forskning är att jämföra olika metoder för kollisionsdetektering hos ett fysik-objekt. Detta blir ett kvasi-experiment som bygger på att jämföra skillnader mellan olika grupper (Borg & Westerlund, 2012, s.14-16). Gruppkategorier för denna forskning blir diskret och kontinuerlig, som är två av de metoder för kollisionsdetektering som kan användas. För att jämföra olika metoder kommer det att finnas två lag av NPC (*non playable character*, icke spelbar karaktär). Målet för en NPC är att skjuta projektiler mot det motsatta laget. NPC:er och Projektilers kollisionsdetektering är inställda på antingen diskret eller kontinuerlig. Genom att ha olika simuleringar där de här två grupperna får skjuta projektiler på varandra kan vi återskapa ett scenario som förekommer i många spel. Detta ger oss möjlighet att mäta prestandan i ett scenario, fast med olika inställningar för att då upptäcka skillnaderna.

En oberoende variabel är en effekt som har skett efter en manipulation, som leder till en orsak (Borg & Westerlund, 2012, s.12). Den oberoende variabeln i denna forskning är antalet fysik-objekt och deras inställning för detektering av kollisioner. Den oberoende variabelns orsak leder till den beroende variabeln, som i detta fall är den kvantitativa data som ska samlas in (Borg & Westerlund, 2012, s.12). Det är i denna undersökning den totala tiden för en fysik-uppdatering (kallas *FixedUpdate* i *Unity*) att beräkna fysik. Mindre tid är bättre prestanda än högre tid (Claypool et al., 2020; Liu et al., 2021; MacKenzie & Ware, 1993).

3.1.3 Testscenarier

I scenarierna kommer de enda spelobjekt som finns att vara NPC:er och projektiler. NPC:er grupperas i varsitt lag i arméformation och är riktade mot det motsatta laget. NPC:er har en skjutfrekvens på 1,6 sekunder då det skapas ett projektil som förflyttar sig mot det motsatta laget. Om ett projektil inte träffar en NPC från motsatta laget kommer projektilen att förstöras efter fem sekunder från tiden den har skapats.

Tre stycken scenarier skapas beroende på antalet NPC i varje lag. Det första scenariot heter Liten med 100 mot 100 som i *Company of Heroes* (2006). Andra är Medium med 250 mot 250 som i *Age of Empires IV* (2021). Den sista heter Stort med 1000 mot 1000 som i t.ex. *Supreme Commander* (2007). Det totala antalet objekt i scenen

samlas också in och presenteras som medelvärde för varje scenario, alltså soldater och projektiler.

Experimentets miljö är en avskalad bana som är skapad i *Unity* spelmotor. Banan är tom för att få färre störande variabler till datainsamling. I banan finns två lag av NPC. Testet ska simulera ett scenario med tre olika arméstorlekar som kan hända i strategispel. Ett exempel på detta scenario är att simulera arméer som skjuter mot varandra, som kan ske i strategispel (*Total War: Shogun 2*, 2011; *The Lord of the Rings: The Battle for Middle-Earth 2*, 2004). Skotten som blir skjutna är projektilerna. Projektilens metod för kollisionsdetektering, som NPC:er skjuter, kommer att vara samma för alla fiender, antingen diskret eller kontinuerlig.

Alla test scenarier kommer att köras på samma bärbara dator med dessa specifikationer:

Processor: 11th Gen Intel(R) Core(™) i5-11400H @ 2.70 GHz 2.69 GHz,

Minne: 32 GB RAM-minne,

Operativsystem: 64-Bit Windows 11 Home

Grafikkort: NVIDIA GeForce RTX 3050 Laptop GPU

Under experimentet ska alla möjliga bakgrundsprogram på datorn stängas av för att inte påverka prestandan, detta kan annars leda till icke-pålitlig data. Prestandan för varje scenario kan mätas med *Unity Profiler*, ett verktyg i *Unity* spelmotor för att mäta prestanda (*Unity Profiler*, 2025). I *Unity Profiler* kan tiden för fysik separeras från resten (*Physics Profiler Module*, 2025), detta ger den kvantitativa data som analyseras.

Den data som har samlats in ska visualiseras med hjälp av spridningsdiagram, låddiagram och andra relevanta visualiseringsmetoder.

Testmiljön utvecklas genom att dels bygga spel-logiken för icke spelbara karaktärer och projektiler, men även att bygga logik för datainsamling. Detta eftersom relevant data måste kunna samlas in och sparas ner på sådant sätt att den går att analysera i kalkylark. Utvecklingen gjordes iterativt genom att testmiljön utvecklades med fokus på att få en färdig testmiljö relativt snabbt. För att sedan kunna köra den, samla in data, analysera datan och dra slutsatser. Genom att försöka göra hela forskningsprocessen på en första implementation av testmiljön så kunde fel och eventuella missar upptäckas tidigt. Det gick att lära sig av felen som upptäckts för att utveckla testmiljön till att passa bättre.

3.2 Metoddiskussion

Eftersom detta projekts målbild är att med kvantitativ data mäta skillnaden i prestanda mellan två algoritmer, så har en iterativ metod bedömts som bäst. Detta då de specifika kraven på applikationens uppbyggnad behövde ändras utifrån resultaten. Kudryashov (2024) ger exempel på design i olika cykler för en webbshop. Metoden har använts genom att testmiljön har utvecklats för att kunna genomföra en pilotstudie, vars resultat sedan använts för att vidareutveckla testmiljön. Genom att göra en pilotstudie kunde resultatet undersökas för att bedöma om det faktiskt samlades in den data som behövdes.

Studien har på grund av begränsad tid och resurser avgränsats till enklare scenarion med färre störande variabler. Detta begränsar dock validiteten i studien delvis eftersom scenarierna inte helt återspeglar hur ett riktigt kommersiellt spel hade sett ut och fungerat.

Valet att endast genomföra studien på endast en dator begränsar generaliserbarheten i studien. Ett exempel är minneshantering som kan förändra insamlat data beroende på hur mycket ram minne en dator har. Resultat bedöms dock ändå vara av intresse för att belysa hur de två metoderna skiljer sig i prestanda.

4 Genomförande av studien

Detta kapitel förklarar hur studien genomfördes, presenterar resultatet av insamlad data och slutligen analyserar resultatet.

4.1 Testmiljö

Kraven för testets scenarier är att det ska finnas två lag av NPC:er, blått och rött lag, som ska skjuta varandra med projektiler. Testerna kommer att skapas i *Unity* version 2022.3.29f1 (*Unity*, 2024).

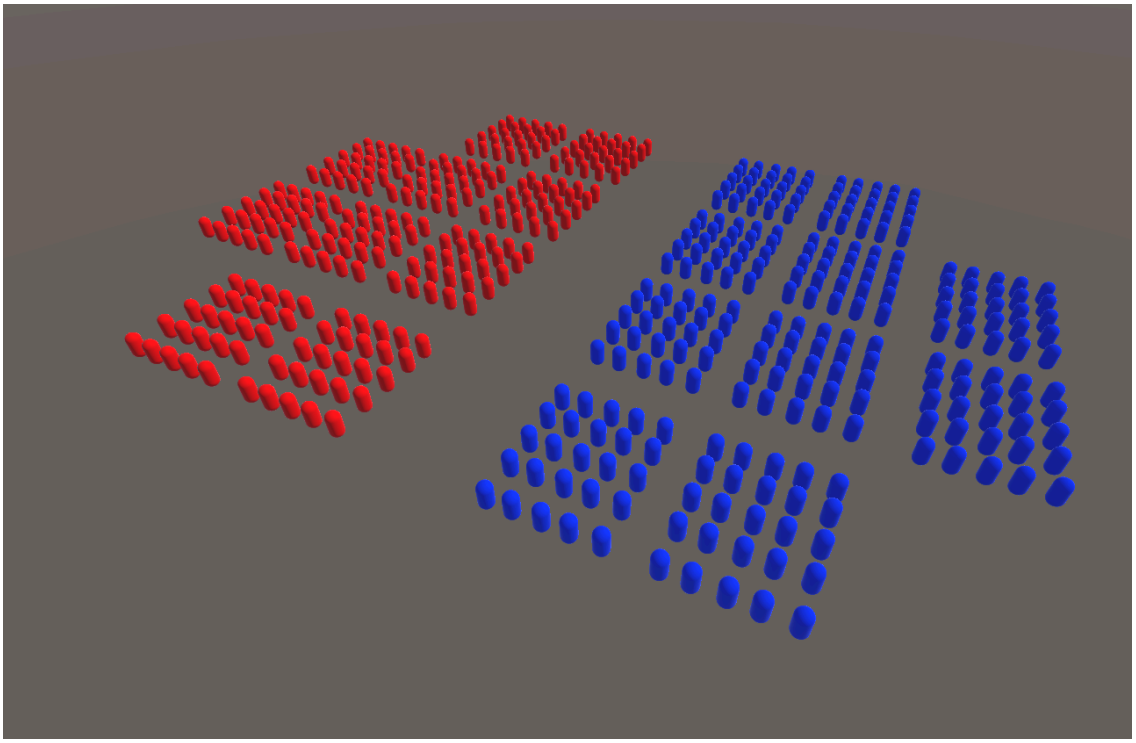

Bild 3 *Från Medium-scenariot. NPC:er uppställda i varsin arméformation*

De båda lagen är riktade mot varandra, blått lag tittar mot rött lag och tvärtom. Meshen för NPC:er var standard-kapslar. NPC:er rör sig från en startpunkt till två olika punkter. Den ena punkten ligger en viss distans från höger om startpunkten och den andra till vänster. NPC:en kommer att röra sig mellan dessa två punkter, fortfarande riktat mot andra lagen. Förflyttning av en NPC sker med fysik-motorn, detta ger möjlighet att mäta prestanda för fysik-objekt, då de manipuleras.

Projektiler är det andra objekt som används för att mäta datan. Meshen för projektilerna är en standard-sfär. Projektilers rörelse sker genom att deras velocitet sätts till deras färdriktning i fysik-motorn. Projektilerna kommer också att bli påverkade av gravitation. Projektilerna är programmerade att förstöra sig själva efter en viss tid om de inte träffar ett objekt, detta för att förhindra att en projektil ska finnas för evigt i scenen. *Layer mask* i *Unity* används för att projektiler inte ska träffa andra projektiler från samma lag (*Unity*, 2024).

Systemet Analyzer.cs har skapats för datainsamling och nyttjar *Unitys ProfilerRecorder* genom Recorder.cs, *ProfilerRecorder* sparar prestanda-data i *Unity* som kan hämtas ut via kod. Kategorin "*Physics.Simulate*" är den mest relevanta kategorin för studien, andra kategorier sparas ner, men *Physics.Simulate* ansågs vara

mest relevant, eftersom den anger total tid för fysik-beräkningar i en uppdatering (*Unity*, 2024). En enskild datapunkt blir därmed en fysik-uppdaterings beräkningstid. Systemet ställs in på en viss tidsfrekvens som gör att 10 datapunkter samlas in per sekund. Eftersom den vanliga uppdateringscykeln körs fler gånger så har en inbyggd timer byggts in för att uppdateringen endast ska ske med denna frekvens. Anledningen till detta är för att data ska kunna samlas in i samma takt. Vare sig det är en liten simulering som går snabbt att simulera, eller att det är en större simulering som tar längre tid.

Analyzer.cs har en samling med Recorder.cs objekt, en för varje kategori och uppdaterar alla dessa objekt löpande genom Update()-funktionen. Efter varje 0,1 sekunder kommer programmet att samla in data om fysik. Totalt kommer det ske 5000 gånger per grupp av stickprov. Anledningen till det är att Joglekar (2016) rekommenderar det som en ideal stickprovsstorlek för att analysera prestanda.

När programmet har gått klart så avslutas simuleringen, då skriver Analyzer-funktionen ner all insamlad data i en .csv-fil. Insamlad data bearbetas i kalkylark för att sedan visualiseras i stapeldiagram och låddiagram.

4.2 Resultat

Extremvärden har tagits bort från datan med hjälp av Jost (n.d.) metod. Genom att beräkna varje grupp av stickprovs yttre gränser, benämnda Outer Fence 1 (OF1) och Outer Fence 2 (OF2) där OF1 är den lägre gränsen och OF2 är den övre gränsen.

$$OF1 = Q1 - 3 * IQR$$

$$OF2 = Q3 + 3* IQR$$

Q1 är den lägre kvartilen, Q3 är den övre kvartilen och IQR = Q3 - Q1 där IQR är det inre kvartilavståndet. Den sista kolumnen i nedan matris anger hur många extremvärden respektive scenario hade av de totalt 5000 datapunkter som samlades in per grupp. För diskret går det att utläsa en tydlig ökning av antalet extrema avvikare från Liten till Stort, men för kontinuerlig metod finns ingen sådan trend. För varje storlek så är samtliga värden betydligt högre för kontinuerlig än för diskret metod.

Tabell 1 *Resultat av testdata*

Grupp av stickprov	Medelantal objekt i scenen	Minimi värde	Lägre kvartil	Övre kvartil	Maxvärde	Medel värde	Median	Extrem avvikare
Diskret Liten	551	0.3168	0.4109	0.4858	0.7021	0.4540	0.4442	11
Kontinuerlig Liten	541	0.4640	0.7059	0.8269	1.2092	0.7675	0.7640	147
Diskret Medium	1,479	0.4311	0.5407	0.6095	0.8190	0.5817	0.5743	49
Kontinuerlig Medium	1,473	0.6918	1.0354	1.1951	1.6491	1.1154	1.1119	35
Diskret Stor	6,239	1.0081	1.3404	1.5997	2.4230	1.4760	1.4762	275
Kontinuerlig Stor	6,180	1.5827	2.0739	2.4537	3.5699	2.2705	2.2668	281

Låddiagram används ofta för att beskriva prisrörelser i finansiella instrument, men används här för att visa spridningen av värden i scenarierna. Varje "sticka" har en kropp med en linje över och under sig. Den övre linjen går upp till det övre värdet i gruppen och den undre linjen går ner till det undre. Linjen mitt i lådan representerar medianen. Extremvärden är inte med i diagrammet. Ju större kroppen är, ju mer olika värden har förekommit. Vilket i detta fall betyder fler olika beräkningstider för detekteringen av kollisioner. En mindre kropp betyder därmed att beräkningstiden har varit mer stabil och värdena har inte skiljt sig så mycket från varandra.

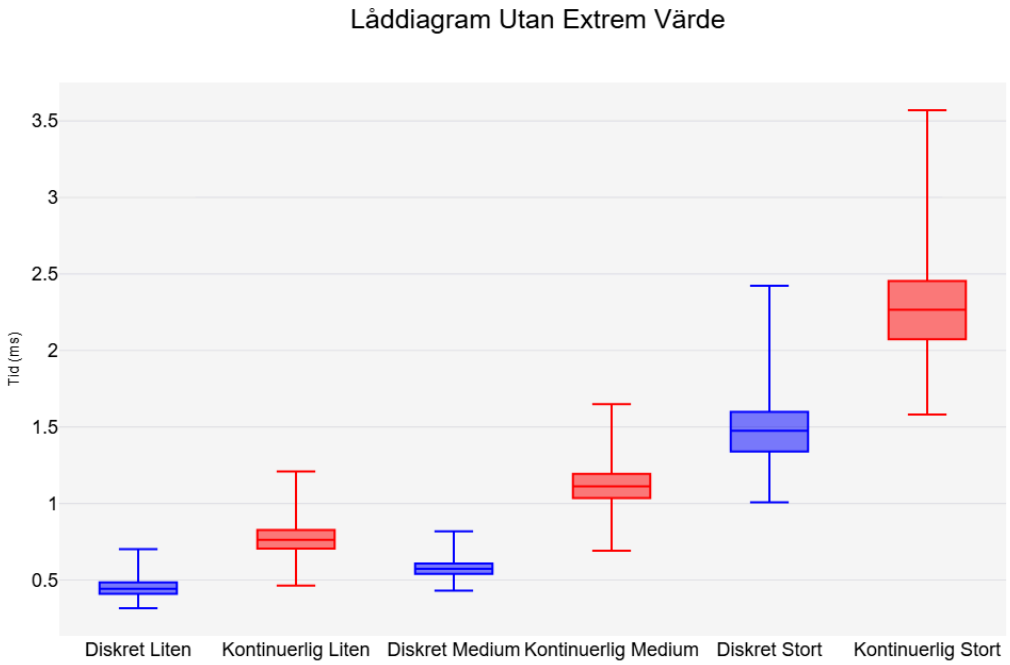

Bild 4 *Låddiagram av beräkningstiden för fysik-uppdateringen utan extremvärden*

Här går det tydligt att se att de större grupperna får högre värden och större kroppar. I samtliga storlekar är diskret den metod som har minst kropp. För Medium är den kontinuerliga metodens kropp ovanför det högsta värdet i diskret, vilket tyder på att majoriteten av alla värden var högre än de högsta värdena i diskret.

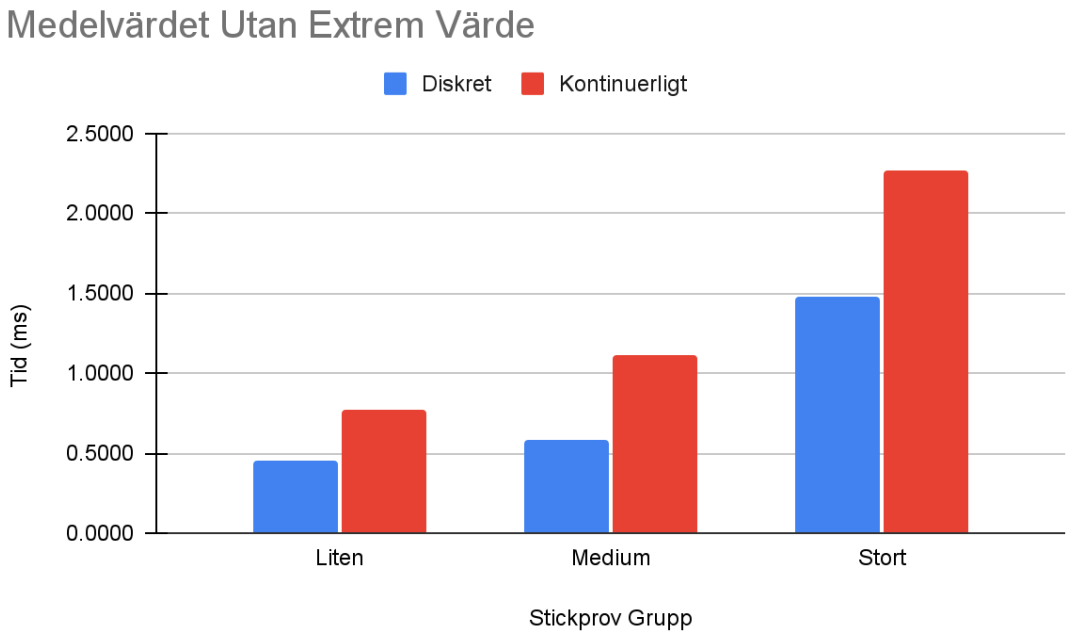

Bild 5 *Stapeldiagram av medelvärdet för fysik-uppdateringens beräkningstid*

Skillnaden i medelvärde mellan de två metoderna är som störst på Medium, där är den kontinuerliga metodens värde 92% större än den diskreta. För Liten är det 69% högre och för Stort är det 54% högre.

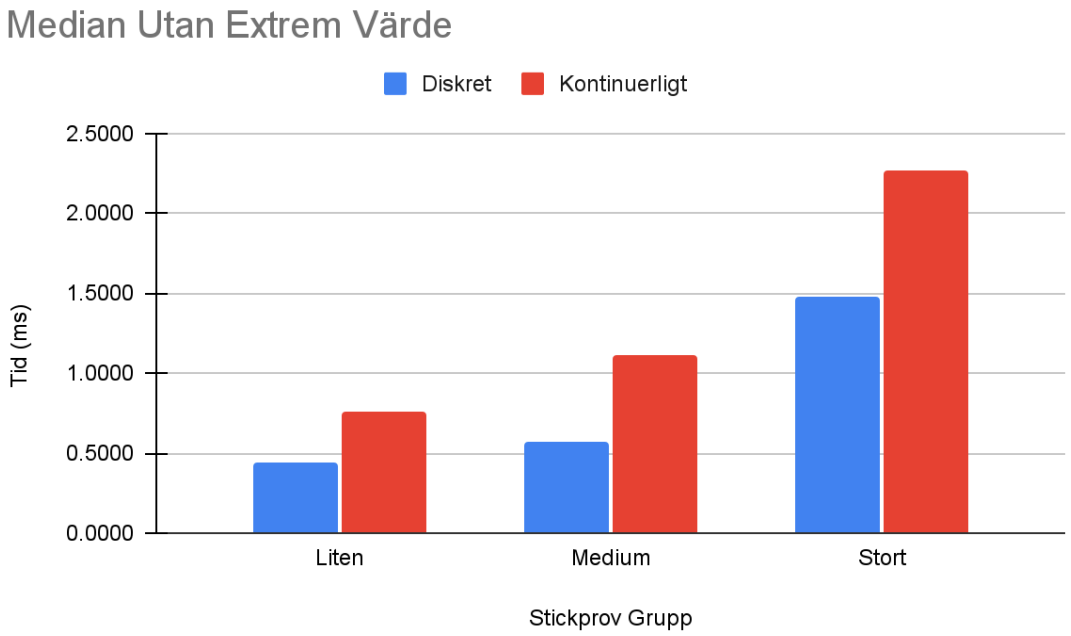

Bild 6 *Stapeldiagram av medianvärdet för fysik-uppdateringens beräkningstid*

Skillnaden i medianvärde är snarlik skillnaden i medelvärde. För Medium är

kontinuerlig 94% större, för Liten är den 72% större och för Stort är den 54% högre.

I testdatan fanns det en del extremvärden som togs bort, i nedan låddiagram är extremvärdena med för att illustrera hur stora extremvärden som förekom. Det förekommer fler extremvärden i de större testerna och i kontinuerlig än i de mindre och i diskret.

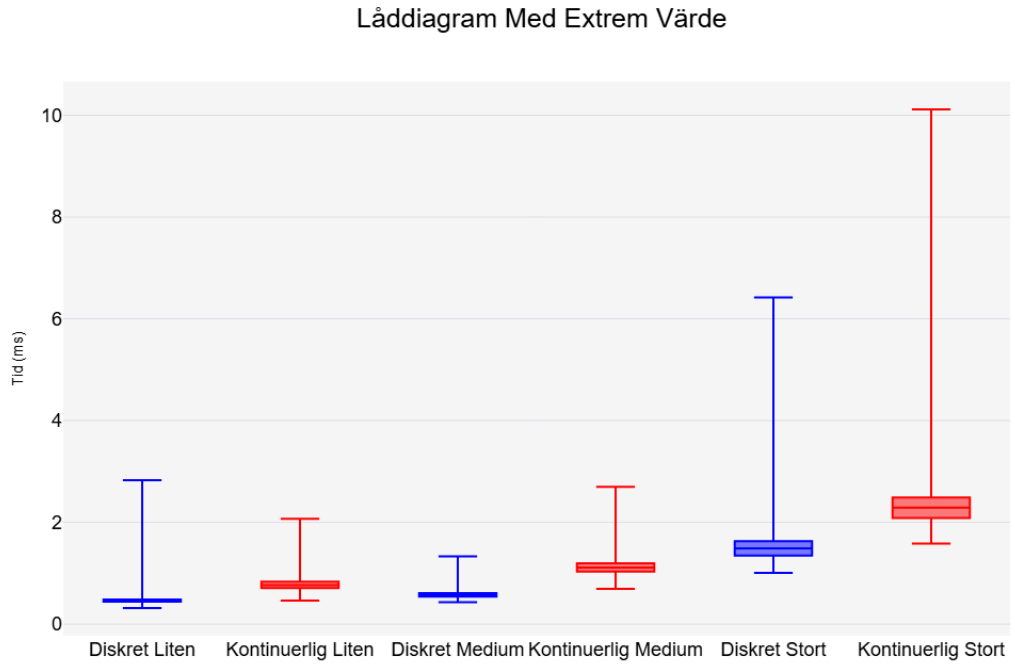

Bild 7 *Låddiagram av beräkningstiden för fysik-uppdateringen inklusive extremvärden*

Trots att lådorna är smala överlag, så är lådorna för den kontinuerliga metoden något större än den diskreta metoden. Lådornas storlek bestäms av det inre kvartilavståndet (Inner Quartile Range, förkortat IQR). Nedan diagram visar IQR för de olika grupperna utan extremvärden.

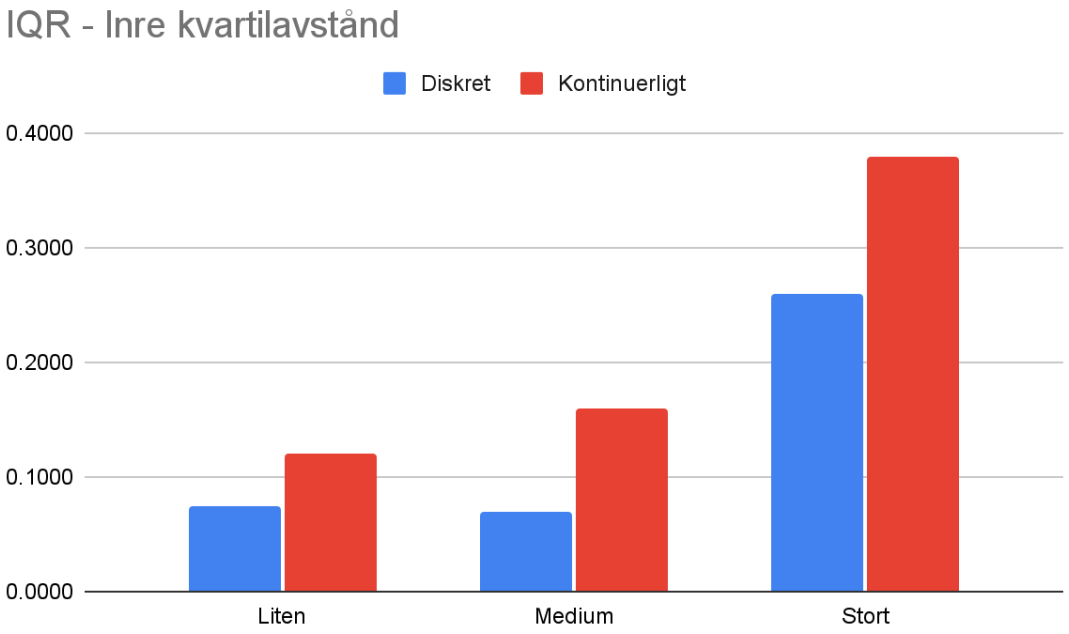

Bild 8 *Stapeldiagram av IQR för fysik-uppdateringens beräkningstid*

Kontinuerlig metod har ett betydligt större IQR för samtliga scenarion, vilket tyder på ett större spann av beräkningstider för just den metoden.

4.3 Analys

Det går tydligt att se i datan att den kontinuerliga metoden är långsammare än diskret, vilket såklart var väntat. I Tabell 1 är samtliga median och medelvärden närmast identiska, vilket tyder på att datan har en symmetrisk sannolikhetsfördelning. Detta betyder att datan inte har någon snedfördelning, alltså att låga värden förekommer lika ofta som höga värden.

Då scenarierna var uppsatta på sådant vis att kollisioner skedde regelbundet på ett förutsägbart vis, så borde inte spikar ske på grund av att det råkat slumpa sig så att för många objekt kolliderar i samma uppdatering, medan andra uppdateringar inte har så många kollisioner. Det borde vara en jämn mängd av konstanta kollisioner. Spikar borde rimligtvis bero på *garbage collection* påkallat av den ständiga instansieringen av nya objekt i scenen i form av nya projektiler. *Garbage collection* är ett system i *Unity* för att frigöra minne som inte är i användning (*Garbage Collector Overview*, 2025).

Något som gör det mer oklart är att de större grupperna inte nödvändigtvis har fler extremvärden. För diskret är det en tydlig ökning i extremvärden från den lägre till den högre. Men för den kontinuerliga metoden minskar mängden extremvärden från låg till medel för att sedan öka igen till den största gruppen. Det är därmed svårt att se något samband mellan storleken på scenarierna och förekomsten av extremvärden, det går dock att tydligt se att kontinuerlig metod har fler extremvärden än diskret i samtliga scenarion, se Tabell 1.

I de tre scenarier som testats i denna studie går det inte att se något samband med att ett ökat antal kolliderande objekt skulle ge en större prestandaskillnad mellan diskret

och kontinuerlig metod. Sett till medianvärde i Bild 6 så var skillnaden som störst för Medium-storleken, följt av Liten och slutligen Stort. Det samma gäller för medelvärde i Bild 5. Resultatet berättar inte varför Medium har större skillnad än de andra scenarierna. Resultatet förklarar inte heller varför extremvärdena i Bild 7 är så stora, men det skedde bara enstaka gånger så det skulle inte ske ofta under en spelsession.

I låddiagrammet i Bild 4 utan extremvärden är samtliga lådor relativt smala, detta tyder på en relativt stabil prestanda då majoriteten av datapunkter ligger runt medelvärdet. En mindre kropp betyder därmed att beräkningstiden har varit mer stabil och värdena har inte skiljt sig så mycket från varandra.

Bild 8 tyder på att den kontinuerliga metoden har mer varierad prestanda än den diskreta metoden, prestandan blir dessutom mer varierad ju större scenario det är. Även här är skillnaden som störst i det mellersta scenariot.

Den kontinuerliga metoden verkar kräva mer prestanda och ha mindre förutsägbar prestanda, med fler spikar. Om en spelutvecklare gör ett större spel, med fler NPC:er och fler projektiler, så är det värt att överväga att använda diskret metod. Opålitlig prestanda på grund av en alltför långsam fysik-uppdatering bör undvikas, då hög latens har en negativ påverkan på spelupplevelsen (Liu et al., 2021). Enskilda missade kollisioner kanske dessutom minska i betydelse i ett stort fältslag. Den procentuella skillnaden mellan de två metoderna kan då vara relevant att ta i beaktande.

I det mindre scenariot kan den kontinuerliga metoden vara att föredra. Till exempel om det är ett strategispel med mindre strider. Där snabba och precisa truppförelser för att undvika projektiler är en viktig del av spelupplevelsen. Då blir en mer exakt kollisionsdetektering viktigare. Resultatet pekar på att den kontinuerliga metoden oftast ligger under 1 ms i den minsta scenariot. Då skulle ett sådant mindre spel kunna ha både tillförlitliga kollisioner och en bra prestanda, vilket tillåter spelarna att göra precisa och snabba klick för att flytta runt sina soldater, vilket har visat sig vara viktigt enligt Claypool & Claypool (2007).

En svaghet med denna undersökning är att denna studie bara gjordes på en dator för att samla in data för analys. Om fler datorer med annan hårdvara hade använts, kunde mer generaliserbar data insamlats för att förstärka hypotesen. Detta skulle leda till olika tider, men algoritmernas komplexitet skulle vara oförändrad. Skillnaden mellan diskret och kontinuerlig borde vara lika, oberoende oavsett vilken hårdvara man kör testet på.

Ett designbeslut angående projektilerna kan ha en påverkan på studiens resultat. Varje gång en NPC skjuter, skapas en ny instans av en projektil och nytt minne allokeras. Detta leder till mer *Garbage Collection*, vilket kräver mer prestanda. Det betyder att varje gång en projektil förstörs måste minnet från den returneras till operativsystemet, detta har en påverkan på prestanda. Ett sätt att undvika detta är *Object Pooling*, detta är en designprincip för att återanvända data istället för att förstöra den, en studie av Renström (2023) visar att *Object Pooling* ger bättre prestanda. I detta fall skulle en NPC återanvända projektiler istället för att förstöra och skapa nya. I vår studie kom några extremvärden upp under testningen, Bild 7, anledning till detta kan vara att *Garbage Collection* aktiverades för att returnera minne från förstörda projektiler. Att använda *Object Pooling* kan mildra dessa extremvärden, men vi har inget bevis på att *Garbage Collection* påverkade extremvärdena med tanke på att datan som analyseras kommer från fysik-delen av

Unity Profiler medan *Garbage Collection* är från processorn.

5 Sammanfattning och diskussion

5.1 Sammanfattning

Med utgång i utvecklares strävan efter att skapa spel eller simuleringar som upplevs som responsiva och snabba för slutanvändaren, utforskar denna studie skillnaden i prestanda mellan två metoder för kollisionsdetektering. I många spel är en del av progressionen att fler objekt introduceras, som till exempel fler fiender och fler projektiler för att göra spelet svårare. Det kan vara bland annat skjutspel eller strategispel där snabba beslut och träffsäkerhet är viktiga för att nå framgång.

Ju fler objekt det är i spelet vars kollisioner måste detekteras, desto mer processorkraft krävs av datorn. Samtidigt som det blir viktigare för spelaren att spelet är responsivt, för att hinna agera. När spel-logikens beräkningar tar för lång tid ökar latensen. En hög latens har en negativ påverkan på spelares prestation och upplevelse av spelet (Liu et al., 2021).

Studien jämför diskret och kontinuerlig kollisionshantering. Diskret innebär att en kontroll av möjliga kollisioner görs i varje uppdatering utifrån objektets position i just den uppdateringen (*Discrete Collision Detection*, 2025). Kontinuerlig kollisionsdetektering innebär att kontrollen görs genom att interpolera objekten över en viss sekvens och se om kollisioner sker (*Sweep-Based CCD*, 2025). Den senare nämnda metoden är dyrare, men har mindre risk för missade kollisioner.

Ett kvantitativt experiment har genomförts där tre simuleringar som liknar tre olika sorters strategispel har körts för de två metoderna för detektering. Två arméer står uppställda och skjuter på varandra. Den minsta scenariot hade 100 soldater i varje armé som skjuter på varandra, den mellersta hade 250 soldater i varje armé och den största hade 1000 i varje armé. Under simuleringen går soldaterna i sidled fram och tillbaka för att vara i rörelse samtidigt som de skjuter. Simuleringen körs tills 5 000 datapunkter har samlats in i varje simulering.

Resultaten från studien visade tydligt att den kontinuerliga metoden var mer krävande och hade mindre stabil prestanda än den diskreta metoden. Den relativa skillnaden i prestanda mellan metoderna är dock inte likadan i de tre simuleringarna. I de två mindre simuleringarna funkade det relativt bra att använda båda metoderna. Resultatet belyser dock skillnaden, vilket kan bidra till en spelutvecklares beslutsfattande gällande val av metod.

5.2 Diskussion

Insamlad data visar att den diskreta metoden ger bättre prestanda än den kontinuerliga. Diskret ska helst vara den första inställningen som sätts på ett fysik-objekt. Diskret har dock större risk för missade kollisioner. Tester kan göras på kollisionsdetektering för att se om diskret metod klarar av spelets kollisions-situationer, om många kollisioner missas så kan diskussion om att ändra dess metod till kontinuerlig göras, för att inte använda för mycket processor-resurser. För spel med få snabba objekt där träffsäkerheten är viktig kan kontinuerlig vara rätt val. För storskaliga simuleringar med många NPC:er kan diskret vara att föredra för att inte riskera dålig prestanda.

Wilcox et al. (2015), Liu et al. (2021) och Claypool et al. (2020) visar alla på hur viktigt en låg latens är för spel, både för spelarens upplevelse och för dennes prestation i spelet. Många spel blir svårare genom att fler och fler fiender och projektiler läggs till, vilket leder till fler saker att ta hänsyn till och reagera på. Det blir då viktigare att spelarens input resulterar i snabb output.

Resultatet för denna studie är användbar för spelutvecklare som vill skapa ett spel med många fysik-objekt i sina banor. Men detta test kan kallas för ett extrem-scenario. Eftersom att alla NPC:erna i banan är nära varandra och konstant skjuter på varandra utan avbrott. Scenarierna skiljer sig därmed troligtvis från scenarier i spel som är utvecklade för kommersiellt syfte. Spelutvecklare kan därmed vara mer villiga att använda den kontinuerliga metoden, då deras spel kanske inte har lika många kollisioner som denna studies scenarier. Det är ett exempel som leder till att studien är användbar för andra spelutvecklare.

Det finns inte en metod för kollisionsdetektering som är perfekt för alla sorters spel, utan en utvecklare bör utgå från vad spelet kräver för att kunna köras med tillräckligt stabil och hög bildfrekvens. Samtidigt kan olika sorters spel ha olika krav på hur exakt detekteringen av kollisioner ska vara. I ett större strategispel med till exempel över 1 000 soldater i varje armé (och då totalt c.a. 6000 fysik-objekt med projektiler inräknat) så kan den kontinuerliga metoden bedömas som för långsam, samtidigt som exakta kollisioner inte nödvändigtvis är lika viktigt. Men i mindre strategispel, som till exempel *Company of Heroes* (2006), detaljstyrs soldaterna i intensiva strider där spelarens styrning kan påverka om soldater hinner ducka eller flytta sig från en inkommande projektil. I ett sådant sorts spel är det färre soldater och projektiler som kommer behöva kontrolleras för kollisioner löpande, samtidigt som det blir viktigare med en exakt och pålitlig detektering av dessa kollisioner. I ett sådant fall kan kontinuerlig kollisionsdetektering vara värt att lägga processorresurser på.

5.3 Samhälleliga och etiska aspekter

Då studien har varit en kvantitativ studie av skillnaden i beräkningstid mellan två algoritmer utan några andra personer inblandade i studien, så bedöms inte någon risk för etiska problem vara relevant. Beaktat studiens tekniska natur bedöms inte heller genus och kulturfrågor vara relevanta heller. Detta baseras på *God Forskningssed* (2024) där de olika aspekterna för att följa god forskningssed har beaktats. I metodbeskrivningen har det vidtagits åtgärder för att vara öppna med hur studien har genomförts för att ge läsaren en bild av förfarandet och därmed vara transparenta.

Studien bedöms vara relevant för utvecklare som skapar spel, simuleringar och andra applikationer. Simuleringar med kollisioner är relevant i många fall där utveckling sker. Speciellt i de fall man skapar spel såsom de som nämnts som exempel tidigare i denna text. Denna studie syftar till att bidra med användbar information till utvecklare som behöver balansera spelbarhet och precision i kollisioner mot prestanda.

5.4 Framtida arbete

För framtida arbete finns det några brister med denna studie som kan mitigeras för att få bättre data. En framtida studie skulle kunna göra tester på flera datorer med olika hårdvara och med olika versioner av programvara. Den skulle kunna visa om denna

studies resultat är generaliserbart. En annan studie skulle kunna vara att göra testerna i en ordentlig spelmiljö istället för en tom bana. Med mer grafik, fler miljöer och mer utvecklad AI skulle testmiljön bli mer lik ett spel. Detta skulle ge data som är mer kopplad till hur det skulle se ut i ett kommersiellt spel. Ett till exempel är att inkludera fler stickprovsstorlekar, till exempel ha fler stickprov i en linjär skala. Att analysera i en linjär skala skulle kunna visa om skillnaden i prestanda har en linjär tillväxt eller inte. Studien skulle även kunna kompletteras med analys av hur minneshantering påverkar beräkningstiden för kollisionerna. För att förstå resultatet bättre kan fler kategorier av data samlas in för att förklara varför Medium stickprovet har större skillnad än Liten och Stor. Data skulle också kunna samlas in för att förklara varför extremvärden uppstår.

Det sista exemplet är att kontrollera hur pass mycket mer den diskreta metoden missar kollisioner, jämfört med den kontinuerliga metoden. Denna studie handlar om att se skillnader i prestanda, men det kan även vara av intresse för spelutvecklare att se hur pass bra metoderna är på att upptäcka kollisioner. Att göra tester för att se hur pass mycket missade kollisioner som sker och/eller är acceptabla, kan ge en mer tydlig bild till när en spelutvecklare ska välja mellan den diskreta och kontinuerliga metoden.

Referenser

Age of Empires IV. (2021). Xbox Game Studios.

Borg, E., & Westerlund, J. (2012). *Statistik för beteendevetare* (3rd ed.). Liber.

Claypool, K. T., & Claypool, M. (2007). On frame rate and player performance in first person shooter games. *Multimedia Systems*, *13*(1), s.3-17. 10.1007/s00530-007-0081-1

Claypool, M., Cockburn, A., & Gutwin, C. (2020). The Impact of Motion and Delay on Selecting Game Targetrs with a Mouse. *AM Trans. Multimedia Comput. Commun. Appl.*, *16*(No.2), s.24. ACM. https://doi.org/10.1145/3390464

Collision Detection. (2025). Epic Games. Hämtad Mars 19, 2025, från https://developer.unigine.com/en/docs/latest/principles/physics/collision/?rlang=cpp

Company of Heroes. (2006). THQ.

Discrete collision detection. (2025). Unity Technologies. Hämtad Mars 19, 2025, från https://docs.unity3d.com/6000.0/Documentation/Manual/discrete-collision-detection.html

Garbage collector overview. (2025). Unity Technologies. Hämtad Mars 6, 2025, från https://docs.unity3d.com/6000.0/Documentation/Manual/performance-garbage-collector.html

God forskningssed 2024. (2024). Vetenskapsrådet.

Joglekar, N. (2016). *How CPU Sampling Works*. Microsoft. Hämtad Februari 19, 2025, från https://devblogs.microsoft.com/devops/how-cpu-sampling-works/

Jost, S. D. (n.d.). *Boxplots and Outliers*. DePaul University. Hämtad Mars 5, 2025, från https://condor.depaul.edu/sjost/lsp121/documents/boxplots.htm

Kudryashov, A. (2024). *Incremental vs Waterfall Model in Software Development*. Cyfrania. Hämtad Februari 7, 2025, från https://cyfrania.com/incremental-vs-waterfall-model

Liu, S., Claypool, M., Kuwahara, A., Scovell, J., & Sherman, J. (2021, Maj 8-13). Lower

is Better? The Effects of Local Latencies on Competitive First-Person Shooter Game Players. *Conference on Human Factors in Computing Systems - Proceedings*, 12. ACM. Hämtad Februari 6, 2025, från https://doi.org/10.1145/3411764.3445245

The Lord of the Rings: The Battle for Middle-earth 2. (2004). EA Games.

MacKenzie, I. S., & Ware, C. (1993, Maj 1). Lag as a determinant of human performance in interactive systems. *CHI '93: Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, 488-493. ACM Digital Library. 10.1145/169059.169431

Physics Profiler module. (2025). Unity Technologies. Hämtad Februari 7, 2025, från https://docs.unity3d.com/Manual/ProfilerPhysics.html

Renström, F. (2023). *Pooling Pattern och Inkrementell Garbage Collections påverkan på prestanda*. Hämtad Mars 6, 2025, från https://his.diva-portal.org/smash/record.jsf?aq2=%5B%5B%5D%5D&c=6&af=%5B%5D&searchType=SIMPLE&sortOrder2=title_sort_asc&query=prestanda+unity&language=sv&pid=diva2%3A1750684&aq=%5B%5B%5D%5D&sf=all&aqe=%5B%5D&sortOrder=author_sort_asc&onlyFullText=false&noO

Rigidbody. (2025). Unity Technologies. Hämtad Februari 7, 2025, från https://docs.unity3d.com/6000.0/Documentation/ScriptReference/Rigidbody.html

Risk of Rain 2. (2020). Gearbox Software.

Supreme Commander. (2007). Microsoft.

Sweep-based CCD. (2025). Unity Technologies. Hämtad Mars 19, 2025, från https://docs.unity3d.com/6000.0/Documentation/Manual/sweep-based-ccd.html

Total War. (2025). Total War. Hämtad Februari 7, 2025, från https://www.totalwar.com/

Total War: Shogun 2. (2011). Sega.

Ultimate Epic Battle Simulator. (2023). Brilliant Game Studios.

Unity (2022.3.29f1). (2024). Unity Technologies.

Unity Engine. (2024). Unity Technologies. Hämtad Februari 7, 2025, från https://unity.com/products/unity-engine

Unity Profiler. (2025). Unity Technologies. Hämtad Februari 7, 2025, från https://docs.unity3d.com/Manual/Profiler.html

Wilcox, L. M., Allison, R. S., Helliker, J., Dunk, B., & Anthony, R. C. (2015). Evidence that Viewers Prefer Higher Frame-Rate Film. *ACM Transactions on Applied Perception*, *12*(4), s.15-26. 10.1145/2810039

Wong, Z. (2021). *Ultimate Epic Battle Simulator 2 Trailer Boasts "Millions" Onscreen*. COG Connected. Hämtad Mars 19, 2025, från https://cogconnected.com/2021/02/ultimate-epic-battle-simulator-2-trailer/