Examensarbete

HÖGSKOLAN
I SKÖVDE

Procedurell generering av grottsystem för dataspel

Jämförelse av procedurellt genererade osymmetriska banor

Procedural generation of caves for computer games

Comparison of procedurally generated nonsymmetrical levels

Examensarbete inom huvudområdet Datavetenskap
Grundnivå 30 högskolepoäng
Vårtermin 2017

Pontus Ek

Handledare: Mikael Thieme
Examinator: Mikael Johannesson

Sammanfattning

Detta arbete handlar om Procedural Content Generation (PCG) i form av algoritmer som skapar osymmetriska banor, och fokuserar på att ge en översikt av prestanda avseende tid, storlek och tillgänglighetsgaranti. Tre olika algoritmer testades på hur lång tid det tog att skapa en bana, och hur stora dess banor blev, och om dessa banor tillåter att man kan nå alla gängliga områden. En Agentbaserad algoritm, Cellular Automata och Diffusion-limited Aggregation (DLA) studerades avseende dess styrkor och svagheter.

Efter experimentet så drogs slutsatsen att DLA var mest effektiv inom tid och garanti dock skapade den små banor. Cellular Automata lyckades skapa stora rum men kunde de inte godkännas på punkten garanti, och den tog för lång tid att köra. Den Agentbaserade algoritmen misslyckades att skapa banor överhuvudtaget.

Nyckelord: Proceedural Content Generation, Bangenerering, Grottor, Algoritmer

Innehållsförteckning

1. Introduktion

Procedural Content Generation (även känt som PCG) är tekniker, som slumpmässigt skapar innehåll till spel och simulatorer. Detta innehåll kan vara dialoger för spel-agenter, banor samt grafik med mera. Med hjälp av PCG kan man överlåta vissa uppgifter från person till dator. Detta område studeras fortfarande inom olika typer av spel och simulatorer för att kunna skapa en bättre upplevelse till användaren. (Shaker, Togelius & Nelson, 2016)

Arbetet handlar om att skapa en översikt över tre olika algoritmer, som är specialiserade på att skapa osymmetriska banor i realtid. Algoritmerna är Cellular Automata, Diffusion-limited Aggregation och en Agentbaserad algoritm. Dessa banor skapas från algoritmer för att sedan användas till spel eller simulatorer. Ett exempel är spelet Rogue, skapat av Toy och Wichman (ca.1980), där man antar rollen som en äventyrare och beger sig ner för grottor och hålor. Spelet skapar en ny bana i realtid, när äventyraren går till en ny håla eller grotta.

Detta arbete siktar på att ge en övergripande sammanfattning av tre algoritmer, dess funktion och hur väl de presterar. Kriterierna är satta för att ge en bild av hur väl algoritmerna kan skapa spelbara banor inom rimlig tid.

Texten förutsätter att läsaren har en grundläggande förståelse av programmering.

2. Bakgrund

2.1 Procedural Content Generation

Procedural Content Generation genererar innehåll till spel och simulatorer. ”Innehåll” i denna miljö kan beskrivas som objekt man finner i spel eller simulatorer. Exempel skulle vara banor/världar, dialoger m.m.. Dessa tekniker bör även ha en begränsad eller indirekt styrning från användaren. Detta innebär att programmet ska kunna skapa innehåll utan att kräva en person under körning. (Shaker, Togelius & Nelson, 2016)

Ett exempel av PCG är spelet Endless Legend, skapat av Amplitude Studios (2014), vilket använder sig av sådan teknik för att skapa en värld. Till detta brukar algoritmen titta på variabler, som bestämmer hur vissa element ska vara byggda. Exempel skulle kunna vara hur många öar, som ska finnas samt hur stora dessa får vara. Dessa variabler ändras av användaren innan algoritmen börjar arbeta. En skärmdump av spelet finns i Illustration 1

Shaker, Togelius och Nelson (2016) skriver att det finns två olika typer av generering. Den första innebär att algoritmen finns med i slutprodukten och skapar nytt innehåll till spelaren i realtid (Online). Den andra används under utvecklingen av programvaran (Offline). Endless Legend från tidigare och Rogue från introduktionen, använder sig av online-generering.

Genom att använda sig av PCG för att skapa innehåll till spel och simulatorer överlåter man arbetet till datorn, vilket innebär att mer kraft kan användas till annat under utvecklingen. Detta är anledningen till att PCG används enligt Dominic Guay från en artikel skriven av Remo (2008). De Carli, et al. (2011) håller med, men anser att det inte innebär att man kan ersätta en mänsklig konstruktör.

Dessa typer av algoritmer anses fortfarande vara i ett startskede och kan förbättras. Van Der Lindern, Lopes och Bidarra (2013) anser att det borde finnas en mer kraftfull, exakt och rikare styrning av dessa tekniker. I deras arbete kan spelrelaterade variabler bestämma högnivåstyrning över algoritmernas arbete. De ansåg att detta behövdes studeras mer för att kunna förbättra tillgång och styrning.

***Illustration** 1: Skärmdump från spelet Endless Legend*

2.2 Bangenererings-baserad PCG

Ett sätt för bangenerering är cellmatriser. Endless Legend av Amplitude Studios (2014) använder sig av en cellmatris för att skapa sin värld. Genom att sätta olika tillstånd i dessa celler i matrisen, kan en bana skapas. För att skapa osymmetriska grottbaserade banor, anses cellmatriser vara det bättre valet, snarare än banor som bygger på fördefinierade delar.

Innan dessa algoritmer kan presenteras, måste upplägget av cellmatrisen bestämmas. En sådan matris är kvadratisk, och innehållet i varje cell kan representera tre olika saker. De är

- **Yttervägg** fungerar som en ram till matrisen. Den används för att säga till algoritmen att dess cell är ett stopp och kan inte byggas på eller utåt.
- **Vägg** fungerar som en byggsten för algoritmen och kan göras om till en golvcell. Alla väggceller behöver inte vara närliggande till varandra
- **Golvceller** är del av banans struktur. Dessa celler måste vara närliggande till varandra för att garantera att banan är spelbar.

Dessa tre typer ger form till banan och sätter gränser för hur stor cellmatrisen kan bli.

2.2.1 Cellular Automata

Cellular Automata (CA) jobbar genom att undersöka alla celler i matrisen. Matrisen börjar med att ha alla sina celler slumpade mellan golv och vägg. Med 4-5 regeln bestämmer denna teknik vilka celler som ändras eller stanna kvar i sitt initiala tillstånd. Beroende på antalet vägg-celler runtom och vad den nuvarande cellen är, så gäller 4-5 regeln som följande: Om cellen är en vägg och det är 4 eller mindre väggceller runtomkring, så blir den till golvcell. Om cellen är golv och det finns 5 eller flera väggceller, så blir den till väggcell.

Denna cells nya tillstånd sparas i en ny matris. 4-5 regeln itereras genom alla celler i matrisen för att skapa en ny version. Denna nya matris körs sedan igenom med samma procedur ett antal gånger och överlämnar den senaste versionen som en färdig bana. Detta är beskrivningen av CA enligt Roguebasin (2005) och Kun (2012).

Ett problem är att den kan skapa isolerade grott-sektioner, med nekad tillgång för spelaren att utforska. Roguebasin (2005) presenterade en lösning för detta. Flood-fill är en funktion, som letar efter den stora mängden närliggande golvceller och tar bort de resterande golv, som inte är del av den mängden.

2.2.2 Diffusion-limited Aggregation och Agentbaserad algoritm

Dessa två algoritmer använder sig av agentobjekt, som placeras i matrisen och bygger ut banan. Agenterna har funktionen att titta på cellen framför sig och gå till den, om det är lämpligt att ta ett steg frammåt. Dessa två tekniker har några variationer på hur dessa agenter placeras och fungerar.

Diffusion-limited aggregation skapar ett kluster i mitten av matrisen. Denna bas fungerar som en lista av potentiella start-positioner för dessa agenter. När en agent har skapats, får den en riktning och börjar sedan att promenera runt. När den tar ett steg så finns den slumpmässiga möjligheten att den kan ändra riktning. Den sparar alla väggar, som den går på, för att göra om dem till golv när den krockar in i antingen golv eller yttervägg. En ny agent skapas från en annan position i klustret och upprepar detta tills en viss mängd celler är golv. (Roguebasin, 2010)

Central-klustret i DLA garanterar att alla banorna tillåter att alla golvceller nås. Den jobbar tills en viss mängd golvceller existerar på banan, som kan innebära att algoritmen jobbar något för länge. Detta kan spela stor roll, då dessa agenter placeras ut slumpmässigt och kan råka låsa in sig helt.

Den agentbaserade algoritmen placerar endast ut en agent, som utforskar matrisen bestående enbart av väggar. Denna agent tittar på cellen framför sig för att bestämma om den kan jobba vidare. När den tar ett steg framåt blir den föregående cellen till golv. Med detta kollar den möjligheter till två olika egenskaper: att ändra riktning och att bygga ett rum. Dessa två möjligheter ökas eller minskas, beroende på den valda egenskapen. Om den ändrar riktning, så kan den inte bygga rum. Denna agent jobbar tills den krockar in en yttervägg eller golv. (Shaker, Togelius & Nelson, 2016)

Eftersom bara en agent används, så garanterar detta att banan har alla golvceller tillgängliga. Ett problem är storleken på banan samt hur lång tid agenten kan arbeta. Om agenten tar för lite tid på sig, är risken stor att banan är för liten. Om banan är stor, så tar arbetet för lång tid. Slumpen kan innebära att agenten ändrar riktning 3-4 gånger, för att sedan krocka in i en golvcell (går runt i en cirkel).

3. Problemformulering

3.1 Grundläggande frågeställning

Arbetet fokuserade på att genomföra experiment med tre algoritmer relaterat till tre olika kriterier. De data som togs fram användes för att ge en översikt av dessa algoritmer och en slutsats och diskussion runt dessa algoritmer avseende styrkor och svagheter. Kriterierna valdes för att kunna skapa en bild av hur väl de kan prestera inom online bangenerering. Dessa kriterier var:

Tidseffektivitet undersökte hur lång tid varje algoritm använde för att skapa en bana. Inom online-generering vill man att dessa körtider är korta. Det anses vara en styrka om den har korta tider.

Tillgänglighet undersökte om dessa banor har alla sina golvceller närliggande till varandra. För att kunna spela igenom hela banan, måste man kunna nå alla golvceller från alla andra golvceller. Detta kriterium kunde antingen vara godkänt eller icke-godkänt på banorna. För att prestera bra, ska alla banor garantera godkänd tillgänglighet.

Golvmängd anger hur stor mängd celler i banan som är golv. Måttet visar hur många celler som blir golvceller relativt totala antalet celler i en matris. Ett gränsvärde som anses vara lämpligt är 50%, då det kan garantera en bana stor nog för spelaren.

Tre matriser av olika storlekar användes för att skapa en säker slutsats. Till var och en av matriserna skapades 120 banor från algoritmerna. Banornas storlek var 32x32, 64x64 och 128x128. Ändringarna i matrisen skulle ge en bättre bild av hur algoritmernas prestanda, när ytorna ändrades.

Dessa tre algoritmer valdes för deras potentiella egenskaper att skapa osymmetriska banor med hjälp av cellmatriser. Hur de implementeras kommer att beskrivas i senare kapitel.

Den slutgiltiga frågeställningen var följande: ”För bangenereringsalgoritmerna: Cellular Automata, Diffusion-limited aggregation och en agentbaserad PCG; med kriterium tidseffektivitet, tillgänglighet och golvmängd; vilken passar bäst att använda inom spel och simulering med onlinegenerering av osymmetriska banor?” De data som togs fram användes för att formulera en slutsats och diskutera lösningar på svagheter dessa algoritmer har.

Inspiration till detta arbete kommer från tidigare examensarbeten. Evertsson (2014) och Björklund (2016) undersökte olika algoritmer inom PCG och gav en jämförelse inom vissa kriterier. Mer om deras arbete finns på kapitel 4.

PCG studeras fortfarande i sin helhet avseende styrning och prestanda. Det här arbetet implementerar och testar dessa algoritmer samt ger en klar bild av hur väl de genomför sina uppgifter.

3.2 Metodbeskrivning

För att genomföra denna undersökning behövde dessa algoritmer implementeras och krävde även möjlighet att mäta kriterierna. Dessa sätt att ta fram data är inspirerat av tidigare artiklar om PCG skrivna av Evertsson (2014) och Björklund (2016).

För att mäta tidseffektivitet, användes en tidtagarurklass, som anropades när algoritmen började sin generering. Detta tidtagarur stannade när algoritmen var färdig.

För att ta fram golvmängden, räknades alla golvcellerna när matrisen var färdig. Detta värde jämfördes med den totala mängden celler.

En djupetförstsökning användes för att bedöma om alla golvceller var kopplade med varandra. Om den inte kunde hitta alla golvceller, blev banan underkänd.

Spelmotorn Unity skapat av Unity Technologies (2005) användes i detta arbete för att implementera dessa algoritmer. Tillsammans med programmeringsspråket C# kunde man testa teknikerna, och spara resultatet.

Shaker, Togelius och Nelson (2016) presenterade fem kriterier som algoritmer för bangenerering bör uppnå. Det här arbetet fokuserar på två av dem: Tid och tillgänglighet. Resterande kriterier styrning, variation och trovärdighet uteslöts för att begränsa det här arbetets omfattning. Styrning innebär hur mycket man kan manipulera algoritmerna under körning. Variation innebär att resultaten ska vara unika. Trovärdighet handlar om att skapade miljöer uppfattas som äkta.

Kriteriet för golvmängd används för att se hur väl dessa algoritmer jobbar med given yta. Syftet med denna bedömning är att se hur stora banorna kan bli. Om banorna blir små, bedöms det som en svaghet. Det gäller även när procentandelen golvmängd minskade trots att matrisen blev större.

4. Tidigare forskning

Det finns tidigare studier som behandlar samma område. Evertsson (2014) skrev ett arbete om PCG med bangenerering, som använder sig av färdiga rum som kopplades ihop med enkla korridorer. Eftersom dessa delar var fördefinierade så var tillgängligheten av banorna något enklare.

Evertsson (2014) tittade på hur teknikerna Binary Search Partitioning (BSP), Shortest path samt en evolutionär AI kunde prestera. De värden som spelade roll var tidseffektivitet, variationer och garanti att nå alla rum. Hans resultat visade att BSP och Shortest Path var mest lämpliga för spel där genereringen skulle ske online. Algoritmerna använde fördefinierade delar och skapade symmetriska banor. I det här arbetet användes inte dessa algoritmer.

Björklund (2016) jobbade inom samma område med cellmatriser. Han använde BSP, Shortest Path och CA. Kriterierna var hur lång tid dessa algoritmer tog, hur kompakta dess banor blev och mängden oanvända celler.

Björklunds (2016) ansåg att BSP var den mest effektiva algoritmen. Denna algoritm skapade de mest kompakta banorna under dem kortaste tiden. Det här arbetet används samma koncept för att ta fram den totala mängden golvceller, som existerar i dessa banor. Björklund använde enbart CA, som kunde skapa banor med osymmetrisk form. Likt algoritmerna från Evertsson (2014), valdes inte de resterande algoritmerna från Björklund, eftersom de skapar symmetriska banor.

5. Implementation

Denna del av rapporten kommer att presentera artefakt och designval inom arbetet. Till det kommer implementationen av algoritmerna att beskrivas. Därefter diskuteras om denna artefakt var lämplig för studiens genomförande.

Artefakten är ett program som skapar banor med hjälp av de tekniker, som skulle undersökas. Dessa banor studerades utifrån arbetets tre kriterier. Den sparade resultatet i en text-fil, som sedan laddades in i ett kalkylprogram.

5.1 Designval

För att mäta tidsåtgång användes klassen Stopwatch. Denna ansågs vara en smidig lösning för att ta exakt tid, som inte kräver en person som startar och stoppar klockan (se Msdn.microsoft.com inom Stopwatch för mer information om denna klass).

Alla golvceller räknas i varje skapad bana. Denna mängd jämfördes med den totala mängden celler från banan, mätt i procent. Golvmängden sparas som absolut tal och procentvärde.

Djupetförstsökning används för att räkna alla golvceller som är närliggande till varandra. Om den totala mängden golvceller är samma mängd som från tidigare kriterium, så är banan godkänd. Idén till denna test kom från en design presenterat av Weiss (2013).

För att spara alla resultat användes klassen StreamWriter. Denna klass skapar filer, som kan läsas av andra program. Text-filerna är skrivna i tabell-form, vilket gör det smidigt att överföra till kalkylprogram såsom Sun Microsystems (2002) Openoffce Calc. (se Msdn.microsoft,Streamwriter för mer om Streamwriter klassen).

Dessa implementeringar av algoritmerna utgår från beskrivningen i tidigare kapitel. Ändringar anges i följande underkapitel.

5.1.1 Agentbaserad

Denna implementation ville öka golvmängden, när den ansågs vara låg. För att uppnå detta, skapades fler agenter istället för enbart en. Detta innebar också att algoritmen kunde placera ut dessa agenter slumpmässigt upp till en viss mängd. Denna mängd agenter som skapades per bana var 25 st.

En annan skillnad mellan denna implementation och den presenterad av Shaker, Togelius och Nelson (2016) var beteende hos agenterna. Inom denna implementation så dör agenterna endast av yttervägger, medan i det ordinarie konceptet dödades agenten även när den kolliderade med golvceller. Med den nya varianten ökar tillgängligheten. I illustration 2 kan man se ett exempel på hur en bana kan se ut.

5.1.2 Diffusion-limited Aggregation

Denna algoritm jobbar tills en viss mängd golvceller har uppnåtts. Under implementationen ändrades detta till att en viss mängd agenter skulle skapas istället. För att dessa agenter skulle leva längre, så kunde bara ytterväggar avsluta dem. Idén var att begränsa programmets tidsåtgång och minska risken för krasch. Den totala mängden agenter som skapas i denna implementation är 5 st. Se illustration 3 för en skärmdump av en bana från DLA implementationen.

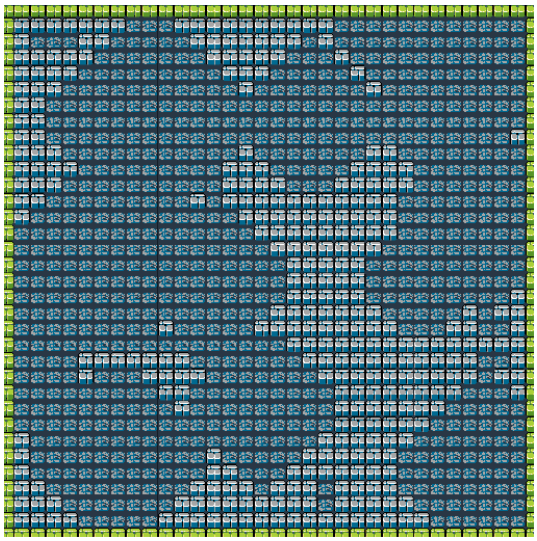

Illustration 2: Bana skapat av den agentbaserade algoritmen.

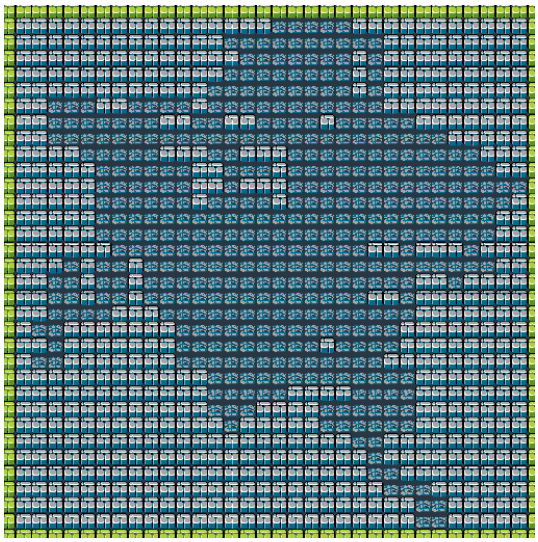

Illustration 3: Bana från DLA

5.1.3 Cellular Automata

Inga ändringar gjordes i ordinarie koncept. Roguebasin (2005) presenterade Flood-fill funktionen, som tar bort små-grupper av golvceller som inte är del utav den största mängden närliggande golvceller. Detta garanterar att alla banorna klarar tillgänglighetskravet. Flood-fill togs inte med i detta arbete då CA skulle bedömas på sina styrkor och svagheter från sitt originalkoncept. Flood-fill som en potentiell lösning till CAs svagheter, att diskuteras längre fram. Ett skärmklipp av CA från artefakten finns på illustration 4.

5.2 Koppling till kriterierna

Artefakten i det här arbetet kan användas till att skapa banor från dessa algoritmer. Den kan även bedöma banorna utifrån kriterierna. Den sparade dessa resultat i text-filer som sedan kunde laddas in kalkylprogram. Hur den tog fram data för dessa kriterier presenterades i början av detta kapitel.

Dessa implementationer hade några ändringar från sina koncept och skulle kunna ge andra resultat än förväntat. Risken bedöms vara liten för påverkan på resultaten.

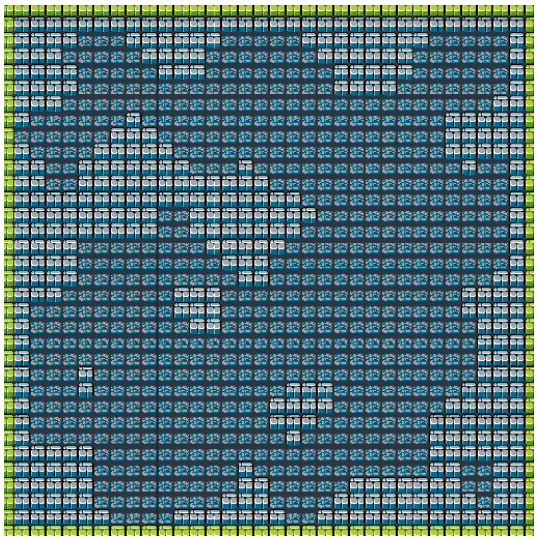

Illustration 4: Bana av CA

6. Utvärdering

6.1 Presentation av undersökning

Denna experimentstudie genomfördes den 25 april 2017, från kl 11.05 till 15.57. Den maskinvara som fanns på datorn var en Intel Core I5-4679k CPU med 3.40 *2 GHz med Operativ-systemet Windows 7 N 64-bit. Det är viktigt att påpeka att annan maskinvara skulle kunna ge andra resultat.

Data presenteras i form av låddiagram eller tabeller. Låddiagram ger en grafisk bild över spridningen av datapunkterna. Själva lådan i dessa diagram representerar de 50% av data som finns nära medelvärdet. Strecken som sticker ut representerar de högre och mindre värden som är utanför detta medelvärde.

6.1.1 Tid

CA tog mest tid på sig inom alla matrisstorlekar. Den använde några enstaka sekunder upp till en hel minut för att skapa en bana. På illustrationerna 5, 6 och 7 så ligger den långt ifrån de andra algoritmerna. Spridningen för CAs tidsåtgång är större än för de andra algoritmerna.

DLA synes vara den mest tidseffektiva på matrisstorleken 32x32 medan för de två större matriserna så blev den agentbaserade algoritmen den mest tidseffektiva. Spridning på den agentbaserade algoritmens tidsåtgång minskade med större matrisstorlek. Detta ses som en styrka då den är mer stabil och förutsägbar inom kriteriet tidsåtgång. (Se Illustrationerna. 5, 6 och 7)

CA hade högst tidsåtgång på grund av sitt sätt att arbeta. Den undersöker alla celler i matrisen med 4-5 regeln, vilket de andra algoritmerna inte gör. När matrisen blir större, så accelererar tidsåtgången.

DLA och den agentbaserade algoritmen hade lägre tidsåtgång för att de ”ritar” i matrisen. Skillnad i arbetssätt beskrevs tidigare. Det fanns tillfällen då spridningen blev större än förväntat. Detta kan bero på livslängden hos agenterna.

Sammanfattningsvis ses DLA och den Agentbaserade algoritmen jobba mest effektiv medan CA tog mera tid på sig.

Illustration 5: Tidmängd (ms) i 32x32

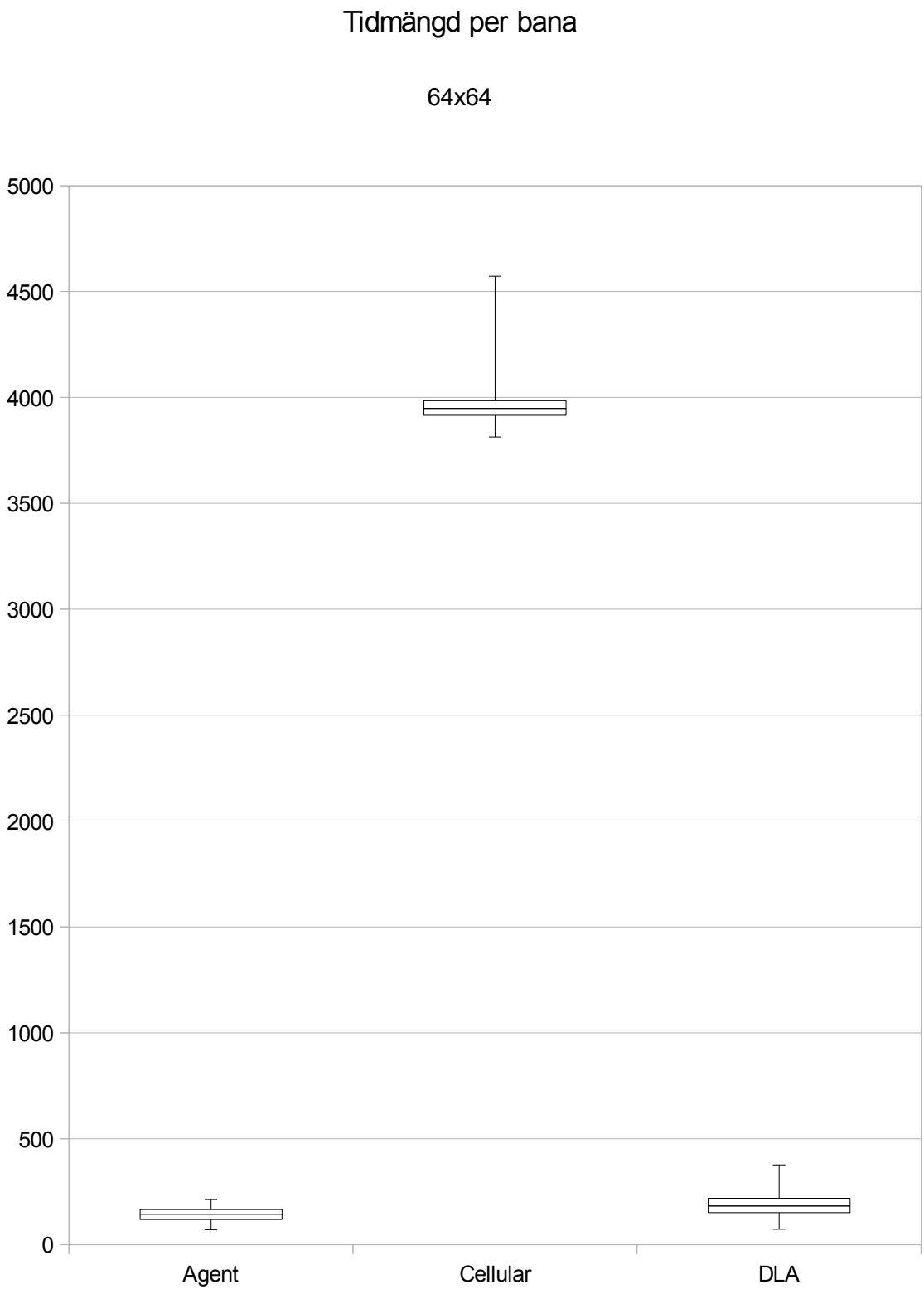

Illustration 6: Tidmängd (ms) i 64x64

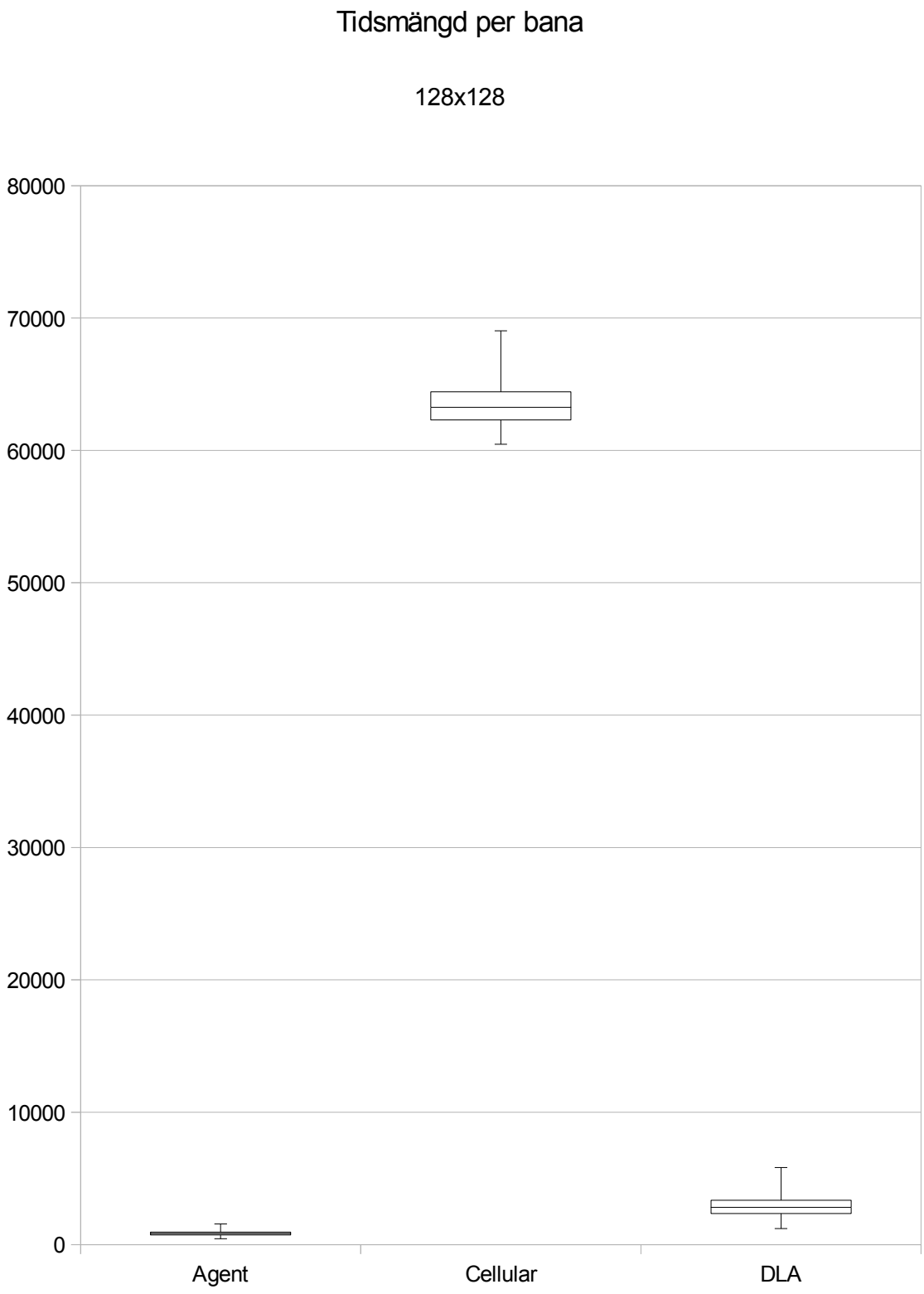

Illustration 7: Tidmängd (ms) i 128x128

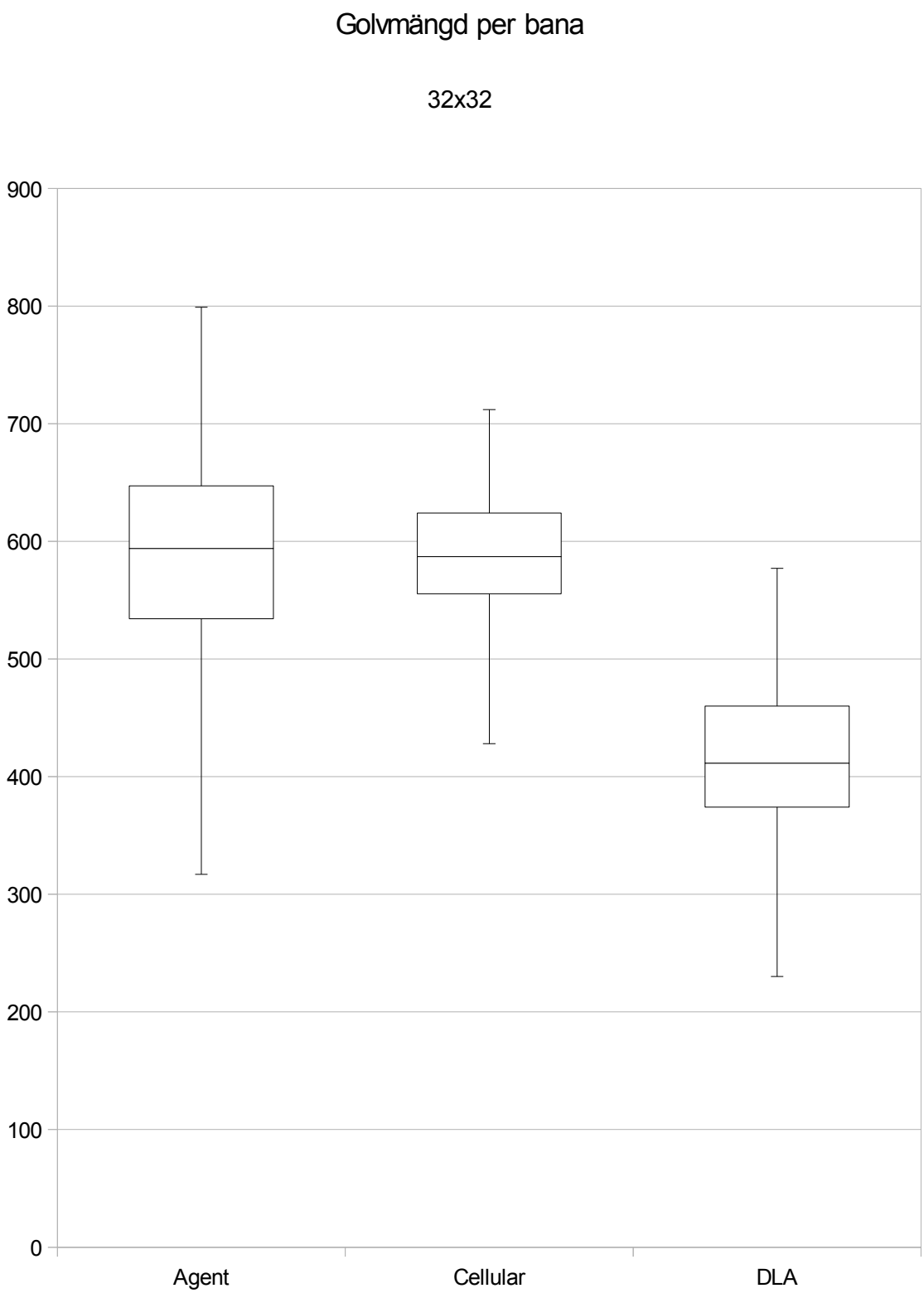

Illustration 8: Golvmängd (Golvceller per bana) i 32x32

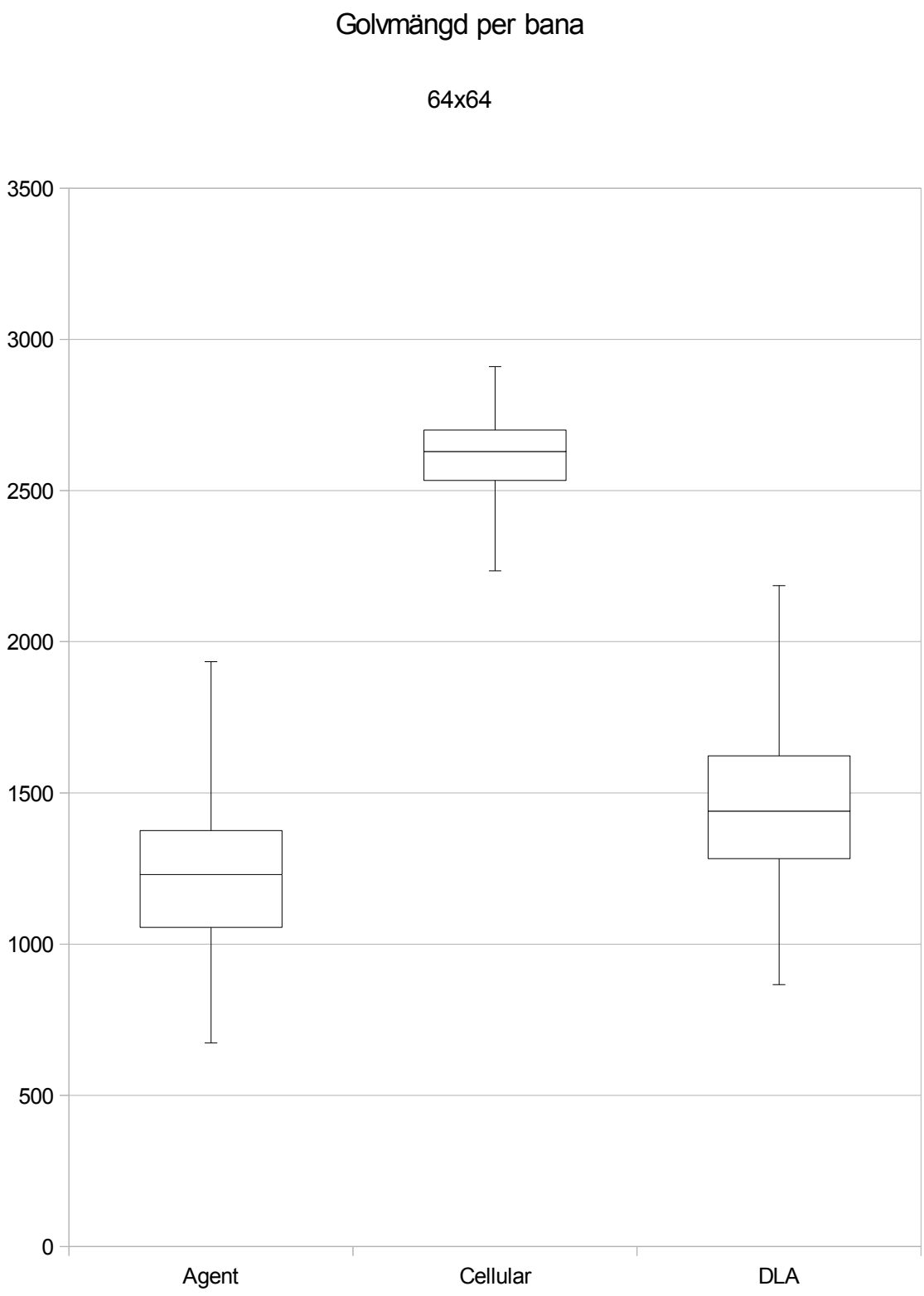

Illustration 9: Golvmängd (Golvceller per bana) i 64x64

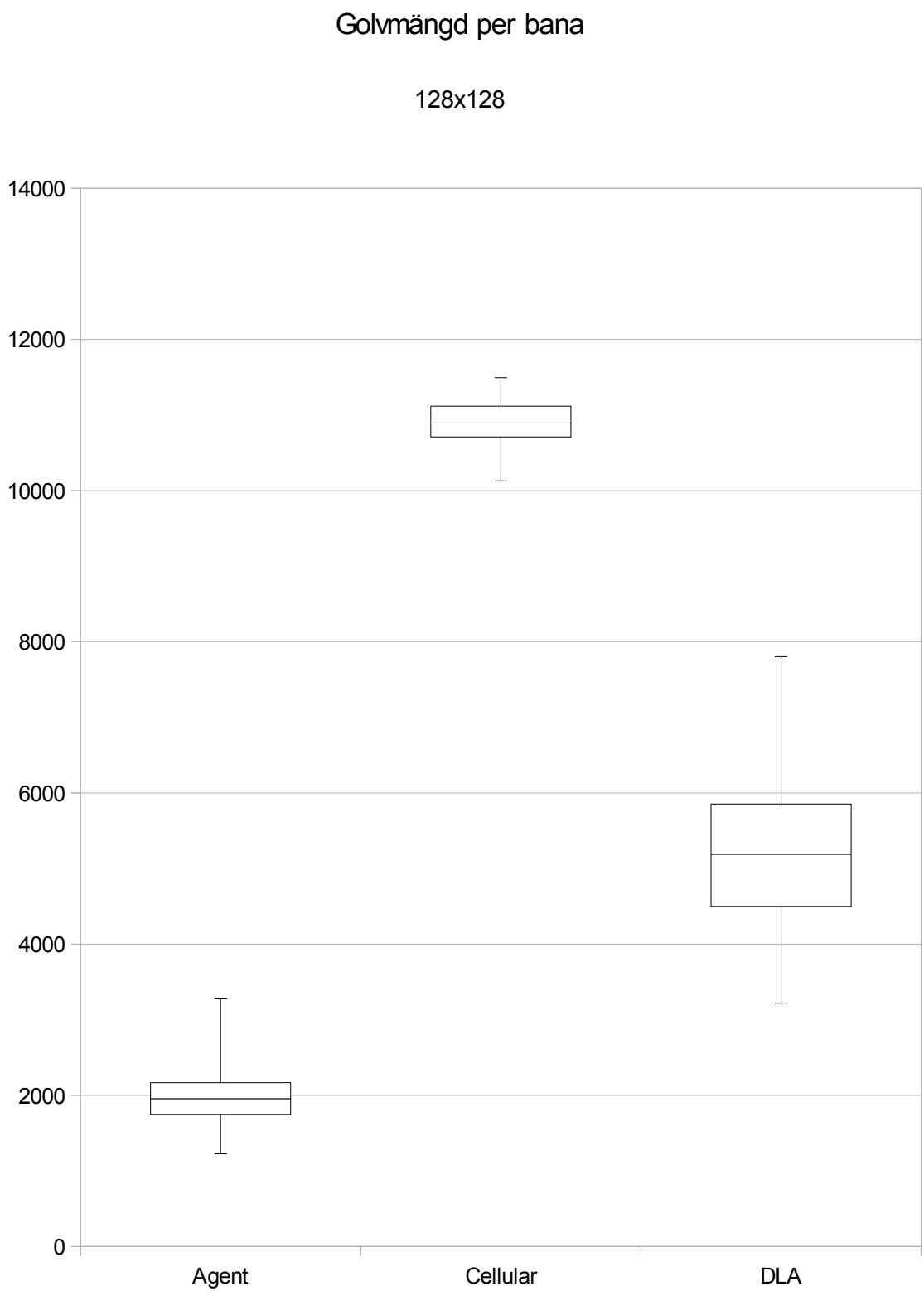

Illustration 10: Golvmängd (Golvceller per bana) 128x128

Golvmängd(% från totala)	Agent	Cell	DLA
32x32:	57,5	57,2	40,8
64x64:	29,9	63,8	35,6
128x128:	11,9	66,5	32

Tabell 1: Golvmängd inom procentvärden

6.1.2 Golvmängd

Från illustration 8 och tabell 1 kan man se att den Agentbaserade algoritmen lyckades ta fram mer golvyta än CA inom 32x32. Med de större matriserna på illustrationerna 9 och 10 så minskades den agentbaserade algoritmen kraftigt medan CA lyckades hålla sig över 50%. Utöver de större matriserna så verkar även dess spridning minska. Detta skulle visa att CA höll sig fast inom en viss gräns för golvyta.

DLA höll sig på 40% men sänkte sig ner till 32% för den största matrisen. Detta ses som en svaghet, då den inte presterar minst 50%.

Varför CA jämt håller sig över 50% av golvceller per bana kan bero på dess skapande av matrisen.

För DLA och den Agentbaserade algoritmen blev dessa resultat sämre än för CA. Detta berodde på mängden agenter och deras livslängd. Mängden är fast (25 st för den agentbaserade och 5 st för DLA) medan livslängden är slumpmässig. Resultaten blir sämre ju större matrisen är.

CA presterade bäst i detta kriterium. DLA och den agentbaserade är svaga i detta kriterium.

6.1.3 Tillgänglighet

Direkt kan man se att DLA presterade bäst i alla matrisstorlekar (se tabell 2). Orsaken är centralklustret, som ger startpositioner till agenterna.

CA lyckas till viss del i den minsta matrisen. Resultatet försämrades påtagligt i större matriser. Dess arbetssätt skapar smågrupper av golvceller som ligger utanför den stora mängden närliggande golvceller.

Den agentbaserade algoritmen misslyckades helt i detta kriterium. Detta beror på att agenternas livslängd blir kort som en följd av algoritmens arbetssätt.

Inom detta område så presterade DLA bäst och ses ha en bra styrka inom detta. CA och den Agentbaserade algoritmen anses inte klara av detta kriterium.

I diskussionen finns förslag till lösningar på svagheter.

Garanti(Godkända %)	DLA	Cellu	Agent
32x32	100	25,8	0,8
64x64	100	2,5	0
128x128	100	0	0

Tabell 2: Resultat inom Tillgänglighet

6.2 Slutsats

Inom denna undersökning nådde CA alltid upp till minst 50% av golvytan i alla matrisstorlekar. CA misslyckas inom tillgänglighet när Flood-fill inte används. Denna algoritmen tog längst tid på sig att arbeta igenom alla celler i matrisen.

Den agentbaserade algoritmen kunde inte garantera att banorna blev spelbara. Dess golvmängd var under 50% golvceller på grund av livslängden på agenterna. Den var mest tidseffektiv.

DLA lyckades garantera att alla banorna var fullt tillgängliga. Algoritmen visade sig vara relativt tidseffektiv. Den hade en låg golvmängd som minskade ytterligare procentuellt när matriserna blev större.

Rangordning för dessa algoritmer är följande: DLA anses vara mest effektiv inom tid och tillgänglighet. Dess enda svaghet är golvmängden. CA skulle komma efter med en högre golvmängd per bana och med högre tillgänglighet än den Agentbaserade. Den kommer efter DLA för att den tog längre tid på sig. Den Agentbaserad misslyckas med tillgänglighet och golvmängd, men jobbar smidigast i tid.

7. Avslutande diskussion

7.1 Sammanfattning

Målet i detta arbete var att undersöka hur väl tre bangenererings-tekniker kunde prestera. En Agentbaserad, Diffusion-limited aggregation och Cellular Automata använder sig av cell-matriser där en cell kan antingen vara golv, vägg eller yttervägg. De två första tillstånden fungerade som byggsten, den sista var gränser för banan. För att bedöma dessa algoritmer, användes tre kriterier: Tid för att se hur länge dessa algoritmer jobbade, golvmängd för att se hur stora dessa banor kunde bli, och tillgänglighet för att se om man kunde nå alla delar. Tre olika storlekar på matriser användes för att se ändringar i beteende i algoritmerna. Dessa storlekar var 32x32, 64x64 och 128x128 celler per bana. Algoritmerna skapade 120 banor var.

Algoritmerna implementerades i C# med Unity som verktyg för att genomföra studien. Programmet skrevs för att skapa banor med dessa algoritmer och även ta fram de data som behövdes för analys. Data sparades i text-filer, som sedan kunde läsas in i ett kalkylprogram. Tidtagarur-klass användes för att ta fram tid. Programmet räknade upp alla golvceller den färdiga banan hade. Djupet-först sökning användes för att se om alla golvceller kunde nås.

Analys av data från studien visade att DLA var snabbast på att skapa tillgängliga banor. Dess svaghet var att banorna hade mindre än 50% golvceller. CA placerades på andra plats då den kunde skapa banor med golvmängden över 50% av den totala mängden celler. Arbetet tog för lång tid och dess banor hade låg tillgänglighet. Den Agentbaserade algoritmen placerades sist, då dess banor gick från 50% till 10% golvmängd när storleken på matrisen ökade. Den hade även lågt tillgänglighet. Den tog kortast tid på sig för att skapa banor.

7.2 Diskussion

7.2.1 Potentiella lösningar

DLAs golvmängd på den minsta matrisen var under 50% och minskade ytterligare när matrisen blev större. Lösningen på detta skulle vara att skapa fler agenter. Detta skulle öka möjligheten till större golvmängd, men algoritmen skulle ta längre tid på sig.

CA tog för lång tid på sig och kunde inte garantera tillgänglighet till hela banan. Flood-fill kan användas för att finna den största mängden närliggande golvceller och ta bort de celler som inte är del av den. Nackdelen med Flood-fill är att det tar längre tid. För att få CA att bli mer tidseffektiv, så skulle multitrådning kunna användas. Problemet med multitrådning är att den kan låsa sig själv.

Den agentbaserade tekniken hade problem med att uppnå en större mängd golvceller på större matriser. En lösning till detta skulle vara att öka mängden agenter. Detta kan leda till att antalet golvceller per bana ökas. En nackdel med detta är att tiden ökar. Dess andra svaghet är att tillgängligheten minskar. En lösning till detta skulle vara att spara en lista av golvcellerna som potentiell start-position för nya agenter. Detta leder till att alla agenterna som skapas är kopplade till golvceller från en tidigare agent.

7.2.2 Sammanhang

Detta arbete bidrog med att ge en översikt på hur väl dessa tre algoritmer presterar. Det ger information till utvecklare av spel o/e simulatorer för att fatta bättre beslut om lämpliga algoritmtekniker. Van Der Lindern, Lopes och Bidarra (2013) ansåg att dessa typer av algoritmer bör studeras mer för att kunna få fram en mer trovärdig värld, och jobba snabbare. Presentation av svagheter i dessa algoritmer fungerar som en start till förbättringar. Mängden tid bestämmer om algoritmen kan ta emot mer arbete för att skapa mer detaljerade banor. Golvmängd per bana kan bli större, vilket bidrar till att banan blir större. Tillgänglighet visar att algoritmen skapar en spelbar bana.

Alternativa användningar för dessa algoritmer skulle vara att skapa grundformer under spelets utveckling. När en viss mängd banor hade skapats så tittar en designer igenom dessa banor och använder de som anses fungera. Därefter finputsar och dekorerar designern sin banan. Dessa banor kan sedan användas i slutprodukten spel eller simulator. Det skulle innebära att tid- och tillgänglighetskriterierna inte är så viktiga, medan golvmängden och dess form är av högre värde. Ytterligare en viktig faktor är om banan är tilltalande att spela. Se kapitel 3.2

Något att uppmärksamma om detta arbete är att algoritmerna enbart skapar banans form. Den dekorerar inte banan med objekt. Detta betyder att tiden ökar om dessa banor dekoreras, beroende på vad det är för spel.

En potentiell felkälla i detta arbete är hur algoritmerna har implementerats. Detta betyder att deras äkta koncept inte prövats och att en modifierad version har använts. Vilket skulle innebära en annan slutsats. Till exempel: Om Flood-fill använts, så skulle CA presterat bättre än DLA. I det här arbetet så användes algoritmernas grundkoncept med så få ändringar som möjligt för att bedöma originalkoncepten.

En annan felkälla kan vara val av operativsystem och spelmotorn Unity. Funktioner inom dessa kan ha påverkat testresultaten på ett okänt/okontrollerat sätt. Om man skulle få samma resultat med andra val, kan inte svaras på i den här undersökningen.

7.2.3 Samhälleliga och etiska aspekter

Enligt Young (2009) är spelberoendet ett växande problem bland barn och tonåringar. För mycket tid ägnas åt spel och för lite till andra aktiviteter (t.ex. studier, hälsa, socialt umgänge). Ett exempel som togs upp var onlinespel där man deltar i en annan värld och upplever nya saker i samspel med andra. Detta håller användarna engagerade i längre perioder. Spel som använder bangenerering i realtid skapar banor som inte tar slut. Algoritmerna som undersöktes kan användas i detta syfte.

Van Rooij, Meerkerk, Schoenmakers, Griffiths och van de Mheen (2009) ansåg att detta kan undvikas om spelutvecklaren ser till att varna användaren för utveckling av spelberoende och hänvisa till tjänster som kan hjälpa.

Att sätta fast vad för effekt detta arbete skulle bidra med inom etiska och samhälleliga vyer kan vara vara svårt. Detta arbete fokuserade på att ta fram en överblick på hur väl dessa bangenereringstekniker presterade. Vad denna information kan användas till, skulle vara att bestämma om dessa algoritmer är lämpliga till projekt som spel och simulatorer. Vad dessa programvaror skulle ge är positiva och negativa effekter för samhället är upp till läsaren att bedöma.

7.3 Framtida arbete

En intressant fortsättning skulle vara att testa dessa tekniker med andra gränser och varianter. Implementationen i detta arbete gav den agentbaserade och DLA en begränsad mängd agenter. Det skulle vara intressant att se om resultaten ändras om DLA och den agentbaserade använts i sina originalkoncept.

I 3.2 beskrevs tre andra kriterier som mäter hur väl en PCG-teknik presterar. Styrning, variation och trovärdighet som inte testas i denna experimentstudie. Variation och trovärdighet kan undersökas med hjälp av enkäter till spelare. För att bedöma styrning kan man ge en öppen styrning av algoritmerna till testpersonerna och efterfråga deras åsikt om de upplevde kontroll över hur algoritmens arbete. Utvärdering av samtliga sex kriterier ger en mer tydlig och rättvisande bild över algoritmernas användbarhet.

Referenser

Amplitude Studios (2014) *Endless Legend* [Datorprogram]. Iceberg Interactive.

Björklund O., (2016) *Kompaktheten av procedurellt genererade grottsystem: en jämförelse av procedurellt genererade grottsystem*.[Examensarbete] Högskolan i Skövde, Datavetenskap, URN: urn:nbn:se:his:diva-9431, Skövde

De Carli D. M., d'Ornellas M.C., Tadeu Pozzer C. & Vevilacqua F. (2011) *A Syrvey of Procedural Content Generation Technique Suitable to Game Development*. SBGAMES '11 Proceedings of the 2011 Brzilian Symposium on Games and Digital Entertainment (Pages 26-35). Universidade Federal da Fronteria Sul. IEEE Computer Society Washington.

Evertsson J. (2014) *Rumsbaserad Bangenerering – En jämförelse av procedurella tekniker*. [Examensarbete] Högskolan i Skövde, Datavetenskap, URN: urn:nbn:se:his:diva-12355, Skövde

Kun J., (2012) *The Cellular Automaton Method of Cave Generation* [Blog]. Tillgänglig via:https://jeremykun.com/2012/07/29/the-cellular-automaton-method-for-cave-generation/. (Senast besökt 10-02-2017)

Msdn.microsoft.com, (inget datum), *StreamWriter.Write Method,* [online]. Tillgänglig via https://msdn.microsoft.com/en-us/library/system.io.streamwriter.write(v=vs.110).aspx (senast besökt: 30-03-2017)

Msdn.microsoft.com, (inget datum), *Stopwatch Class (System.Diagnostics),* [online]. Tillgänglig via https://msdn.microsoft.com/en-us/library/system.diagnostics.stopwatch(v=vs.110).aspx (Senast besökt: 30-03-2017)

Remo C., (2008) *MIGS: Far Cry 2's Guay On The Importance of Procedural Content*. Gamasutra [Online] Tillgänglig via: http://www.gamasutra.com/view/news/112115/MIGS_Far_Cry_2s_Guay_On_The_Importance_Of_Procedural_Content.php. (Senast besökt 10-02-2017)

Roguebasin, (2005). *Cellular Automata Method for Generating Random Cave-Like Levels* [online] Tillgänglig via: http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels (Senast besökt: 10-02-2017)

Roguebasin, (2008). *Irregular Shaped Rooms* [online] Tillgänglig via http://www.roguebasin.com/index.php?title=Irregular_Shaped_Rooms (senast besökt: 10-02-2017)

Roguebasin, (2010). *Diffusion-limited aggregation* [online] Tillgänglig via http://www.roguebasin.com/index.php?title=Cellular_Automata_Method_for_Generating_Random_Cave-Like_Levels (Senast besökt: 10-02-2017)

Roguebasin, (2014). *Random Walk Cave Generation* [online] Tillgänglig via http://www.roguebasin.com/index.php?title=Random_Walk_Cave_Generation (Senast besökt: 29-03-2017)

Shaker N., Togelius J., Nelson M., (2016) *Procedural Content generation in Games*. Switzerland, Cham: Springer International Publishing AG. Sid. 3-10, 31-45

Smith G. (2014). *The Future of Procedural Content Generation in Games. Proceedings of the Experimental AI in Games Workshop*. [Online] Tillgänglig via: http://www.aaai.org/ocs/index.php/AIIDE/AIIDE14/paper/viewFile/9078/9022. (senast besökt: 10-02-2017)

Sun Microsystems, (2002), [Datorprogram] *OpenOffice Calc*., Sun Microsystems.

Toy M., Wichman G., Arnold K., & Lane J. (Ca. 1980) *Rogue* [Datorprogram] .

Unity Technologies (2005) *Unity* [Datorprogram]. Unity Technologies

Van Der Linden R., Lopes R., & Bidarra R. (2013) *Procedural Generation of Dungeons – IEEE Transactions on Computational Intelligence and AI in Games*, IEEE. DOI: 10.1109/TCIAIG.2013.2290371 Tillgänglig via: https://graphics.tudelft.nl/Publications-new/2014/LLB14/Procedural.pdf.(Senast besökt: 10-02-2017)

Van Rooji A.J., Meerkerk G.J., Schoenmarkers T.M., Griffiths M., & van de Mheen D.,(2010). *Video game addiction and social responsibility*. 18. Sid. 489-493.

Weiss M.A., (2013). *Data Structures and Algorithm Analysis in C++*, USA, Addison-Wesley. Sid. 419-420a

Workshop on Procedural Content Generation in Games (2010) *Workshop on Procedural Content Generation in Games* [Online] Tillgänglig via: http://pcgames.fdg2010.org/index.html. (Senast besökt 10-02-2017)

Young K., (2009). *Understanding Online Gaming Addiction and Treatment Issues for Adolescents*., - The American Journal of Family Therapy, 37(5). Sid.355-372.