Examensarbete

HÖGSKOLAN
I SKÖVDE

PROCEDURELL GENERERING AV TVÅDIMENSIONELLA GROTTOR

En jämförelse av genereringstiden, grottmängden och pålitligheten mellan Cellular Automata och Diffusion-limited Aggregation

PROCEDURAL GENERATION OF TWO-DIMENSIONAL CAVES

A comparison of the generation time, cave quantity and availability between Cellular Automata and Diffusion-limited aggregation

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 15 högskolepoäng
Vårtermin 2025

Alex Stenestrand
William Johansson

Handledare: Peter Sjöberg
Examinator: Henrik Gustavsson

Sammanfattning

Denna undersökning gick ut på att implementera olika typer av procedurell generation för grottsystem och att jämföra dem, baserat på tre kriterier. Det här var för att se vilken av två algoritmer som presterar bäst inom generering av enkla tvådimensionella grottor. Algoritmerna valda var Cellular Automata och Diffusion-limited Aggregation (DLA), där de utvärderas utifrån kriterierna genereringstid, grottmängd, samt pålitlighet. Unity (Unity Technologies, 2022), samt programspråket C# användes för att utvärdera och implementera dessa algoritmer.

Slutsatsen som drogs var att cellular automata genererade naturliga grottor på en väldigt bra tid. DLA presterade däremot bättre inom grottmängden och pålitligheten. Fortsättningsvis skulle flera algoritmer kunna undersökas, samt fler utvärderingskriterier implementeras för att få ett bredare perspektiv.

Nyckelord: procedurell generation, cellular automata, diffusion-limited aggregation, grottor, algoritmer

Innehållsförteckning

1 Introduktion

Procedurell generering är en teknik för att skapa innehåll automatiskt. Cano, Tardioli och Mosteo (2024) tar upp spel och simulatorer som exempel på när det här används. Inom spel kan det också användas på många olika sätt och vid många olika tillfällen. Spelet Minecraft (Mojang Studios, 2011) genererar allt från berg till hav med hjälp av procedurell generering, vilket visar möjligheterna procedurell generering har inom spel.

Det finns flera olika algoritmer som hanterar den här procedurella genereringen. I den här undersökningen jämfördes två algoritmer, cellular automata och diffusion-limited aggregation (DLA), med syftet att se vilken av dem som presterade bäst. Algoritmerna användes för att skapa enkla tvådimensionella grottor och jämfördes i tre olika kategorier: tidseffektivitet, grottmängd och pålitlighet. De här kriterierna valdes för att få det mest användbara resultatet inom tidsramen för undersökningen. Fokuset låg i att få svar på vilken av algoritmerna man helst skulle vilja använda när man genererar tvådimensionella grottor.

2 Bakgrund

2.1 Procedurell generering

Enligt Cano, Tardioli och Mosteo (2024) innebär procedurell generering att använda algoritmer i stället för manuella tekniker för att generera innehåll. De här algoritmerna följer satta regler för att till exempel få oförutsägbara utfall, följa ett visst mönster eller automatiskt skapa något från grunden. Cano, Tardioli och Mosteo (2024) tar upp simulering och spelmiljöer som exempel på tillfällen där procedurell generering kan användas. Wattanapornprom et al. (2024) förklarar att genom att använda procedurell generering sparas tid, pengar och arbetsinsats. De lägger till att i spel kan det också användas för att skapa nya upplevelser varje gång man spelar, vilket håller spelare engagerade och intresserade. Att automatisera generationen av innehåll, till exempel miljöer och grafik, inom spel gör det möjligt att skapa oändligt med upplevelser, som annars skulle varit omöjligt manuellt. Det gör det inte bara mer effektivt att implementera, men förbättrar även spelupplevelsen.

2.2 Grottgenerering

I den här undersökningen kommer grottor att genereras. Cano, Tardioli och Mosteo (2024) tar upp att fokuset av procedurell generering för underjordiska miljöer i spel ligger mest i grottor. Grottor definieras som naturliga hålrum i berg, stora nog för människor att inträngа i. Undersökningen kommer skapa tvådimensionella versioner av sådana här grottsystem med hjälp av procedurella genereringsalgoritmer. Målet med algoritmerna är att generera grottor som ser naturliga ut genom skapandet av naturliga former med bara lite repetition av miljön. Kartorna, eller banorna, som genereras ska vara uppdelade i två olika delar. Den ena delen är de faktiska grottorna genererade av algoritmerna och den andra delen är berggrunden utan några hålrum.

2.3 Cellular Automata

Cellular automata uppfanns enligt Manneville et al. (1989) av Jon von Neumann och Stanislaw Ulam. De beskriver cellular automata som ett diskret dynamiskt system med dynamiska variabler på ett rutnät, där variablerna har begränsat antal möjliga olika värden. Värdet på variablerna av cellerna i rutnätet är beroende på värdet av de celler som redan finns i omgivningen. Det här skapar specifika mönster som Antonijevic (2021) framför kan bli använt inom fysik, biologi och datorvetenskap. En av de mest populära och tidiga implementationerna av det här är i spelet Game of Life (John Conway, 1970). Game of Life är ett tvådimensionellt simulationsspel där spelet spelar sig själv efter en inmatning från användaren i början av spelet. Spelet visar tydligt den procedurella genereringsförmågan som cellular automata har.

Atoniuk (2024) tar upp cellular automata som en av de möjliga algoritmerna för att lätt generera en tvådimensionell representation av grottsystem, vilket Johnson et al. (2010) har genomfört. Johnson et al. (2010) använde sig av Moore omgivning (eng. Moore neighborhood), vilket betyder att omgivningen av varje cell är definierad av de åtta omgivande cellerna. Cellerna har även tre möjliga värden eller vad som också kan kallas tillstånd. De här tillstånden är: golv, sten och vägg. Det finns även två

regler som Johnson et al. (2010) fastställde tillsammans med de här tillstånden. Första regeln är att en cell är i tillståndet sten om värdet av omgivningen är lika med T (tröskelvärdet som bestämmer vad som blir till sten, där T = 5), annars är cellen golv. Andra regeln är att en stencell som är bredvid en golvcell blir till en väggcell i stället. Det här ger grottliknande resultat som enligt Johnson et al. (2010) är med låg prestandakostnad.

Van der Linden, Lopes och Bidarra (2014) identifierar fyra egenskaper från metoden av Johnson et al. (2010). Den är effektiv och den kan generera oändligt stora banor. Algoritmen är även enkel och den genererar naturliga grottliknande kartor. De tar också upp hur det är möjligt att generera grottorna i realtid med den här metoden.

2.4 Diffusion-limited Aggregation (DLA)

Den första DLA modellen introducerades av Witten och Sander (1981). Huang och Sumasundaran (1987) tar upp att den var skapad för att simulera formationen av kluster. De förklarar att den genererar en typ av random walk för att emulera diffusion av en partikel. Enligt Huang och Xu (2023) kommer namnet från processen där partiklar håller ihop tillsammans (aggregate) medan de flyttar på sig kaotiskt (diffuse) genom ett medium som är utrustat med någon slags begränsande (limiting) kraft.

RogueBasin (2020) förklarar att den här processen och mönster som skapas av partiklarna kan användas inom generering av banor. Banorna som skapas kan bli naturligt grottliknande, vilket gör den till en passande algoritm för grottgenerering. DLA använder sig också av tillstånd för cellerna, där de som behövs är grotta och berggrund.

Noveltech (2023) tar upp hur grottskapandet med DLA fungerar:

1. Skapa ett rutnät fyllt med berggrund.
2. Skapa ett inledande “seed” och sätt de relevanta cellerna till grottan.
3. Skapa en partikel.
4. Flytta partikeln till en slumpmässig granncell.
5. Repetera steg 4 tills partikeln når grottan.
6. Sätt cellen som var partikelns position innan kollisionen till en grottcell.
7. Börja om från steg 3 och repetera tills satta villkoret har nåtts.

Ek (2017) tar upp hur det centrala klustret som skapas i början av algoritmen gör det möjligt att nå alla golvceller i grottan.

2.5 Flood Fill

Melnyk, Havrylko och Levytska (2021) lyfter fram flood fill, också kallad seed fill, algoritmer som ett sätt att färga förenade pixlar i ett område. De förklarar att den fungerar genom att börja på en vald pixel, för att sedan växa och ändra färgen på pixlarna runt omkring. Det här sker tills det inte finns några pixlar kvar att färga om i området. Färgen i det här fallet kan bytas ut till tillstånd, vilket fungerar med cellular automata och DLA.

Melnyk, Havrylko och Levytska (2021) tar även upp att det finns många olika flood fill algoritmer, men de populäraste använder sig av en fyr-riktad metod, åtta-riktad

metod eller skanningslinjemetod (eng. scanline method). Fyr-riktad betyder att algoritmer sprider sig horisontellt och vertikalt, medan åtta-riktad även gör det diagonalt. Skanningslinjemetoden betyder att den sprider sig rad för rad, antingen horisontellt eller vertikalt. Datastrukturen använd för sådana här algoritmer är oftast en stack eller rekursiv.

2.6 Tidigare forskning

Det finns lite tidigare forskning om jämförelsen av procedurella genereringsalgoritmer med hjälp av ett systematiskt experiment. Ek (2017) jämförde cellular automata, DLA och en annan agentbaserad algoritm i sin jämförelsestudie av procedurell grottgenenering. Algoritmerna var ställda mot varandra gällande tidseffektiviteten, tillgängligheten och golvmängden. Resultatet av studien var att DLA presterade bäst inom tidseffektiviteten och tillgängligheten, men inte lika bra inom golvmängden. Cellular automata presterade överlägset bäst inom golvmängden, där golvet tog upp 62.5% i snitt av all yta av kartan. Agentbaserade algoritmen tog lika mycket tid på sig som DLA, men presterade klart sämst i de andra två kategorierna

Antonijevic (2021) utförde också en jämförelsestudie av procedurell grottgenerering där samma utvärderingsmetoder användes. Antonijevic använde sig dock också av variation, vilket visar hur annorlunda olika delar av banan ser ut. Algoritmerna som jämfördes var cellular automata, random walk och Perlin noise, där random walk gav bäst resultat gällande pålitlighet och variation. Cellular automata gav dock bäst genereringstid, medan Perlin noise presterade bäst i att hantera antalet golvceller på banan.

Jämförelsestudien av Björklund (2016) kollade på genereringen av grottsystem för algoritmerna cellular automata, binary space partitioning (BSP) och shortest path. I den här studien var kriterierna tidseffektiviteten, oanvända celler och poäng från en utvärderingsalgoritm. Den här algoritmen kollade på hur många öppna ytor som finns i grottan. Med hjälp av de här kriterierna kom Björklund fram till att BSP är bäst för att skapa kompakta grottor på kort tid, medan cellular automata täcker störst potentiell yta. Shortest Path ockuperarade minst område av den potentiella mängden utrymme

Eriksson (2022) kollade på procedurell generering av grottsystem inom dataspel. De algoritmer som jämfördes var cellular automata, perlin noise och voronoi (dirichlet tessellation). De kriterier de här algoritmerna jämfördes på var tidseffektivitet, tillgänglighet och variation. Slutsatsen kring algoritmerna var att cellular automata var mest tidseffektiv och voronoi presterade bäst inom tillgänglighet och variation. Eriksson (2022) tar upp att Perlin noise presterade alltid på acceptabel nivå, vilket visar att den har många tillämpningar och brett användningsområde.

3 Problemformulering

Syftet med den här undersökningen är att få svar på vilken av cellular automata och DLA som presterar bäst när det gäller att generera tvådimensionella grottor. Det här avgörs med hjälp av tre olika resultat som kommer ifrån genereringstiden, grottmängden och pålitligheten av grottorna. Togelius (2016) tar upp snabbhet, pålitlighet, styrbarhet, variation och trovärdighet som de fem viktigaste egenskaperna för procedurell genereringsalgoritmer. Av de här fem valdes snabbhet (genereringstid) och pålitlighet för den här undersökningen, då de båda ger tydliga resultat som är lätta att mäta och jämföra. Undersökningen fokuserar på procedurell generering inom spel, då det är där grottor oftast genereras. Resultatet kan dock troligtvis användas i andra sammanhang också.

Genereringstiden, eller tidseffektiviteten, är tiden det tar för algoritmerna att generera grottor. Det här är viktigt inom procedurell generering då det till exempel minskar perioden det tar att ladda kartor i ett spel. Det här påverkar alla spel som använder sig av procedurell generering, även om de bara genererar miljön en gång innan spelet börjar, vilket bland annat Minecraft (Mojang Studios, 2011) gör. Minecraft kan däremot bli mycket mer påverkad av genereringstiden när det spelas online på en server. I det här fallet kan olika miljöer genereras under spelets gång, det vill säga i realtid. Rogue (Epyx, 1980) och The Binding of Isaac (Headup Games, 2011) är också exempel på spel där nya banor genereras under spelets gång. När det här sker är det mycket viktigt att använda sig av en effektiv algoritm för att minska väntetiden. Antalet gånger man genererar kartor, samt kartornas storlek, ökar också behovet av en sådan algoritm. Ju fler och större kartorna är, desto mer tid sparas med hjälp av en tidseffektiv algoritm.

Grottmängden är andelen av kartgenereringen som är en grotta. Det här används för att se hur lika grottorna som genereras av de två algoritmerna blir, samt för att se hur grottmängden ändras beroende på hur mycket yta algoritmerna får att jobba med. Bästa resultatet skulle vara att andelen är konsekvent för alla problemstorlekar. Ek (2017) tar upp att om grottmängden går under 50 % kan det ses som negativt, då en spelare inte har tillräckligt med utrymme. Det här betyder att algoritmerna också borde använda ytan effektivt och inte lämna stora mängder av ytor otillgängliga. Togelius (2016) beskriver pålitlighet som trovärdigheten att banan som skapas är användbar. Det här passar in i definitionen av grottmängden, då algoritmen inte är pålitlig om grottan som genereras bara tar upp till exempel 30 % av kartan.

Pålitligheten i den här undersökningen beskriver om grottan är lösbar eller inte. Det här är viktigt för procedurell generering då man vill ha tillgång till de flesta delar av grottan. Antonijevic (2021) lyfter fram att om det finns en start- och slutpunkt finns det också krav på att spelaren ska kunna gå emellan dem. Grottan i det här fallet är därför misslyckad om den inte är tillräckligt sammankopplad. I praktiken betyder det här att om algoritmen misslyckas behöver den generera en ny grotta från början, tills grottan är lösbar.

Cellular automata och DLA var valda som algoritmer då de båda används inom generering av spelmiljöer. Antonijevic (2021) nämner också att cellular automata är den vanligaste algoritmen inom grottgenerering, vilket gör den ett bra val för den här undersökningen. Togelius (2016) lyfter även fram att cellular automata är ett simpelt och snabbt sätt att generera grottliknande strukturer. DLA används även i den här typen av generering, men inte lika ofta. Det här gör det intressant att se skillnaden mellan de båda algoritmerna när det gäller att generera tvådimensionella grottor. Båda fyller kartan med grottor, men på

olika sätt. Cellular automata använder sig av en typ av brus (eng. noise) som skapas, medan DLA är agentbaserad. Yoshida et al. (2001) lyfter även fram att det mönstret DLA skapar liknar ett träd. Strukturen av träd är väldigt likt grottor, vilket gör DLA passande för grottgenerering.

Frågeställningen som undersöks är: Vilken av procedurell genereringsalgoritmerna Cellular Automata och Diffusion-limited Aggregation presterar bäst i generering av tvådimensionella grottor när det gäller genereringstiden, grottmängden och pålitligheten?

3.1 Metodbeskrivning

Metoden för att få fram resultaten kommer att vara ett systematiskt experiment i spelmotorn Unity (Unity Technologies, 2022) där genereringstiden, grottmängden och pålitligheten mäts på de algoritmer som valts. Anledningen till valet av denna metod är att den ger den mest värdefulla rådatan som kan sedan analyseras och besvara frågeställningen. Eftersom Unity ger stöd till programmeringsspråket C# kommer det att användas i projektet för att implementera algoritmerna, samt experimentmiljön.

För att mäta genereringstiden kommer Unitys inbyggda profiler användas för att spara tiden det tog för algoritmerna att exekvera. Datainsamlingen gjord av Koulaxidis och Xinogalos (2022) är ett exempel på användning av Unitys profiler verktyg. Mätningen av genereringstiden sker i millisekunder.

För att få fram grottmängden används samma sätt som Ek (2017) där antalet grottceller beräknas och jämförs med den totala mängden celler. Grottmängden är procenten av alla celler som är en grotta.

Pålitligheten tas fram med hjälp av en flood fill algoritm. Den kommer att sprida sig i området den hamnar i och om området är under 67 % av alla grottceller så misslyckas den. Resultatet av pålitligheten är chansen att algoritmen lyckas i procent. Det här betyder att om resultatet blir 40 % finns det 40 % chans att flood fill sprider sig genom 67 % eller mer av grottcellerna. Den här procenten valdes eftersom så länge 67 % av grottan är nåbar, är den troligen tillräckligt användbar i praktiken. Liapis, Yannakakis och Togelius (2021) tar upp flood fill som en möjlig metod att utvärdera områden av banor i spel. Det går därför att använda den metoden för att få fram pålitligheten av grottan som genereras.

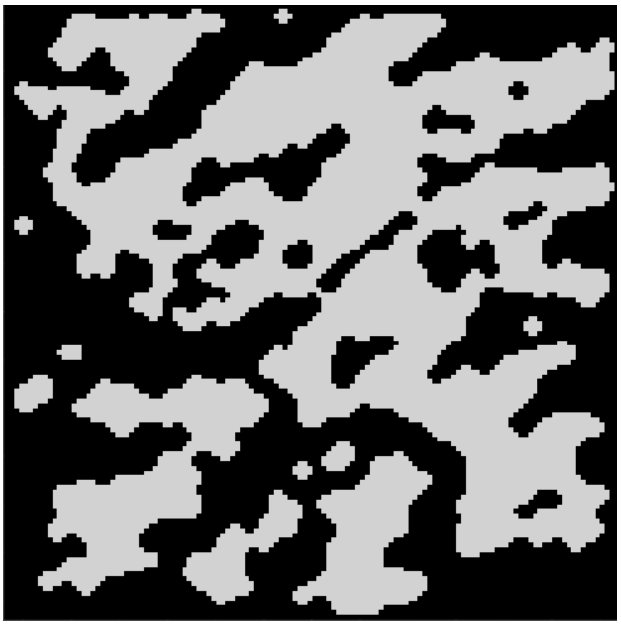
Cellular automata och DLA kommer även skapa kartan grottorna genereras på. Storleken på de här kartorna är 64x64, 128x128 och 256x256. Algoritmerna kommer också att testas 2000 gånger för varje problemstorlek och utvärderingsmetod. Sedan kommer genomsnittet tas från resultaten för att få så användbar data som möjligt. Newsom (2007) tar upp att ju mer tester, också kallat stickprov, som görs, desto mer exakta blir resultaten.

Grottorna som genereras och berggrunden runt omkring kommer att vara en del av svartvit tvådimensionella karta. Grottcellerna kommer att vara vita, medan berggrundscellerna kommer att vara svart. De här olika cellerna bestämmer också de två olika tillstånd som finns, grotta eller berggrund. Grottorna behöver inte vara komplicerade för att få ut den datan som undersöks. De valda algoritmerna används också inom enklare generering, till skillnad från till exempel Perlin Noise som används för landskap.

4 Implementation

4.1 Cellular Automata

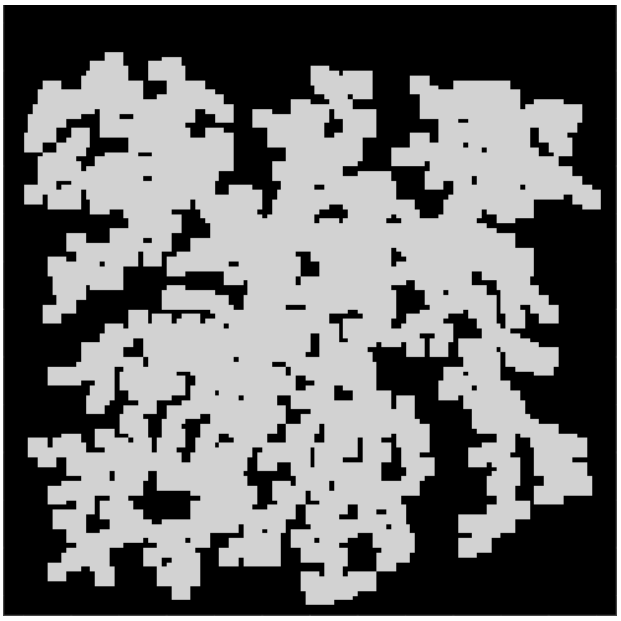
Cellular automata algoritmen implementerades baserat på en youtube video av Sebastian Lague. Algoritmen skapar först en slumpad karta av vita och svarta celler, även kallat en bruskarta (eng. noise map), med ett seed som man själv kan välja eller som kan genereras automatiskt. Denna slump kan justeras för att ändra hur stor chans det är att en ruta är svart eller vit. Algoritmen går sedan igenom alla celler i kartan och ändrar dem till svart eller vit beroende på tillståndet av närliggande celler. Parametrar som användes var 49 % chans att fylla en cell med en svart pixel.



Figur 1 Ett exempel på generering av cellular automata.

4.2 Diffusion-limited Aggregation (DLA)

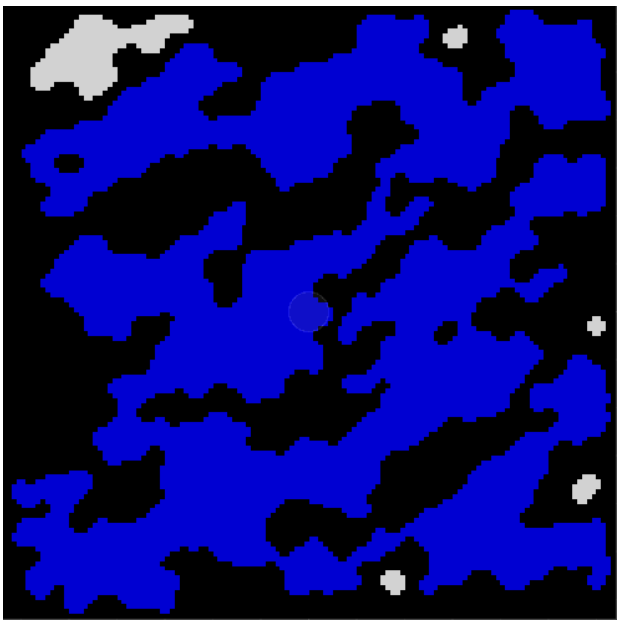
DLA implementerades baserat på en text från Herbert Wolverson. Algoritmen har ändrats för att fungera i C#, implementeringen i texten var följd. Algoritmen skapar först en liten plats i mitten av kartan som framtida partiklar kan kollidera med. Efter det skapas en partikel, eller agent, 1x1 pixel slumpmässigt på kartan. Denna partikel rör sig sedan slumpmässigt tills den kolliderar med en redan existerande partikel. När partikeln har kolliderat så slutar den röra sig och fastnar på plats. Detta fortsätter tills ett viss kriterium har uppfyllts. För att ändra på hur skapandet går till kan vissa parametrar justeras. Det går till exempel att ändra på hur stora pixlarna är när de väl har kolliderat, vilket är möjligt med hjälp av pensel (eng. brush) storlek. Ju större pensel storleken är desto mer celler kan en enda partikel fylla, vilket gör algoritmen mer tidseffektiv men också mindre grottliknande. I den här undersökningen valdes pensel storleken 4 för bästa resultat. En annan parameter var att stoppa algoritmen efter att en viss procent av kartan var grotta, vilket var satt på 50 %. Det går även att implementera så att partiklarna är viktade till att röra sig åt mitten, vilket skulle göra den mer tidseffektiv. Det här var dock inte implementerat.



Figur 2 Ett exempel på generering av DLA.

4.3 Flood Fill

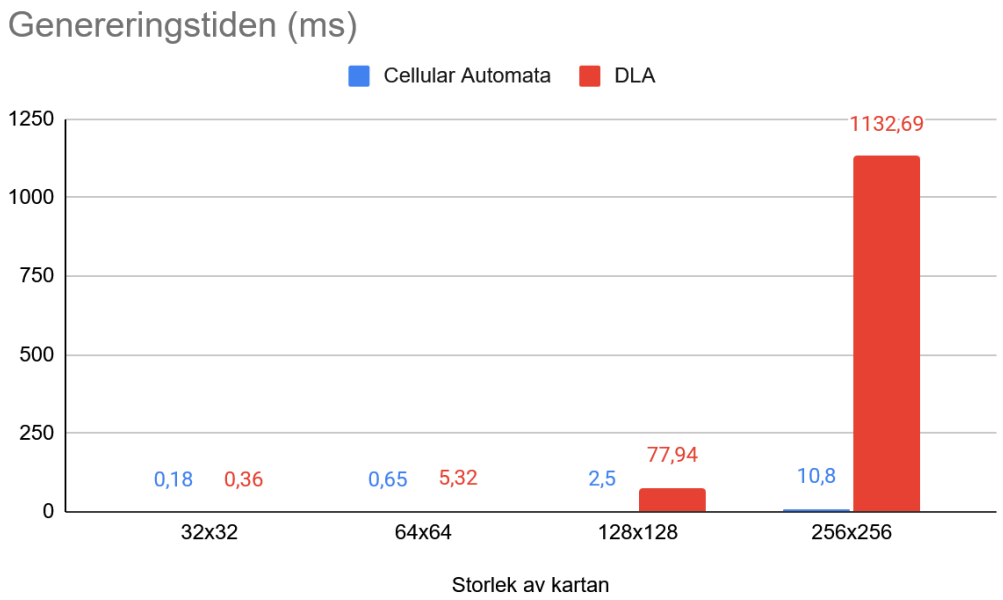
Flood fill algoritmen var baserad på implementationen av Eriksson (2022), men ändringar gjordes för att passa det här experimentet. I det här fallet väljer algoritmen en slumpmässig grottcell och sprider sig i området tills det tar slut. Den använder sig av tillståndet av cellerna för att veta skillnaden mellan grottan och berggrunden, vilket gör det möjligt för den att bara sprida sig genom grottceller. Algoritmen använder sig av en stack datastruktur och är fyr-riktad. En rekursiv metod tar längre tid att exekvera och därför valdes användningen av stack istället. Området algoritmen sprider sig i ändrar den vita färgen till blå.



Figur 3 Ett exempel på cellular automata med flood fill. I det här fallet är kartan pålitlig, eftersom det blåa området är mer än 67 % av grottcellerna.

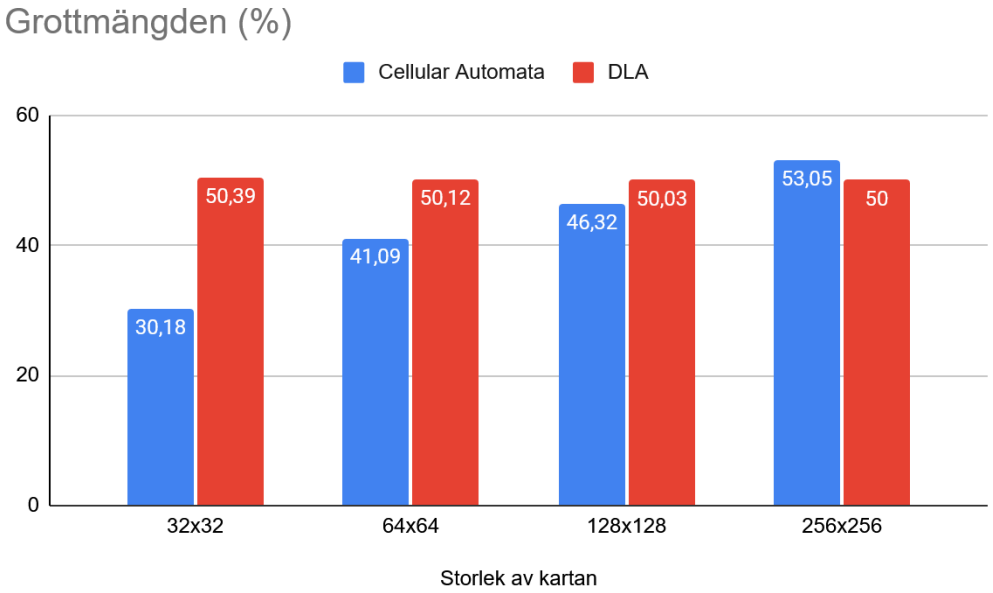
5 Analys

Den första utvärderingsmetoden var att kolla genereringstiden av cellular automata och DLA i millisekunder. Diagrammet visar att cellular automata genererar grottorna mycket snabbare än DLA. På mycket stora problemstorlekar tar det väldigt lång tid för DLA att generera en karta, vilket kan göra den mindre praktisk ifall man söker efter en algoritm som är tidseffektiv. Det här sker på grund av att DLA skapar tusentals agenter som har hög chans att gå på samma cell, medan cellular automata bara kollar en cell en gång.



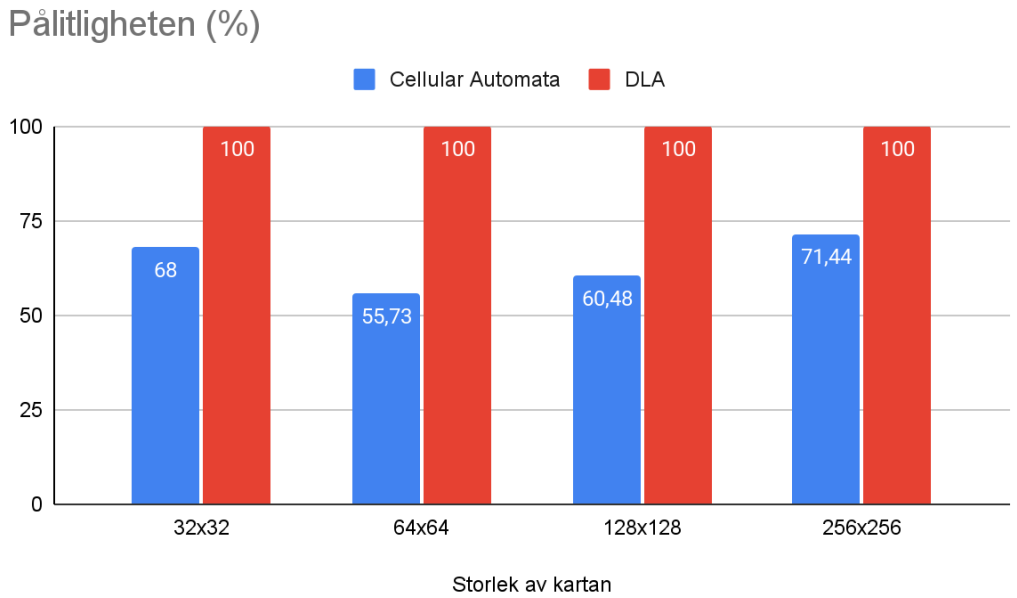
Figur 4 Stapeldiagram av genereringstiden i millisekunder för cellular automata och DLA vid olika storlekar av kartan.

Grottmängden för DLA är mycket lättare att hantera än cellular automata då man kan manuellt sätta stopp för hur stor mängd av kartan man vill ska vara fylld med grotta. Det här betyder att algoritmen fortsätter tills en viss grottmängd är nådd. Vid mindre problemstorlekar kan en cell göra större skillnad och gå över 50 %, men när problemstorlekarna är större gör en cell mindre skillnad. Det här betyder att resultatet av grottmängden blir närmare 50 %. För cellular automata är det inte bestämt innan vilken grottmängd den vill hålla sig runt, utan det är mer slumpmässigt. Det är dock tydligt att ju större kartan är desto mer plats tar grottcellerna upp. I den här utvärderingen var målet att ha tillräckligt med grotta för att det ska vara användbart. Cellular automata når det här målet med större problemstorlekar, men eftersom DLA kan välja grottmängden innan så presterade den bäst.



Figur 5 Stapeldiagram av grottmängden i procent för cellular automata och DLA vid olika storlekar av kartan.

Pålitligheten för DLA är alltid 100 % då den alltid bygger från en punkt i mitten av kartan. Cellular automata är inte garanterad att generera en grotta som är helt sammanhängande, den kan ha oåtkomliga delar av grottan som inte går att komma åt. Cellular automata presterade inte heller särskilt bra inom pålitligheten, men trenden av diagrammet visar att ju större kartan blir desto pålitligare blir den. DLA presterade dock överlägset bättre inom det här utvärderingskriteriet.



Figur 6 Stapeldiagram av pålitligheten i procent för cellular automata och DLA vid olika storlekar av kartan.

6 Sammanfattning och diskussion

6.1 Sammanfattning

Syftet med den här undersökningen var att ta reda på vilken algoritm som presterar bäst när det gäller att generera en enkel tvådimensionell grotta. Algoritmerna som undersöktes var Cellular Automata och Diffusion-limited Aggregation (DLA). Kriterierna de utvärderades ifrån var genereringstiden, grottmängden och pålitligheten.

Cellular automata genererade grottor snabbast och gav väldigt naturliga grottliknande resultat. DLA var mycket långsammare och genererade grottor som inte var lika naturliga, men som fortfarande ser ut som grottor. Det är dock möjligt att ändra parametrar på DLA, till exempel med pensel (eng. brush) storlek. Det här kan öka genereringstiden mycket, men försämrar resultatet av utseendet. DLA var också mer pålitlig och genererade alltid fullt sammanhängande grottor, medan cellular automata inte gjorde det. Vissa grottor var mer eller mindre sammanhängande för cellular automata och det går att ändra på parametrarna för att få mer sammanhängande grottsystem, men det ändrar även på utseendet. Grottmängden för de olika algoritmerna skilde sig också åt, DLA kan bli inställd med parametrar för att sluta vid ett specifikt antal procent av kartan fylld. Cellular automata är däremot inte lika lättstyrd och har ett ungefärligt antal procent av kartan fylld utifrån hur man ställde in parametrarna för slumpen av vita och svarta celler.

6.2 Diskussion

Resultaten kan ses som rimliga utifrån sättet algoritmerna fungerar. DLA behöver skapa flera tusen agenter som går runt för att fylla kartorna, vilket gör algoritmen väldigt tidskrävande. Ursprungligen var större problemstorlekar också med i undersökningen, men DLA var inte snabb nog att hantera dem. Cellular automata behöver bara gå igenom varje cell en gång för att bestämma hur den ska formas, vilket gör den mycket snabbare. En idé för att förbättra prestandan på DLA vid stora kartor är att begränsa området som partiklarna kan röra sig i till ett minimum. Sedan kan det området fördubblas när centrala strukturen som byggs på når ett visst antal celler från kanten av området. Ett annat sätt att förbättra prestandan kan vara att vikta partiklarna så att de rör sig mer åt mitten för att snabbare kollidera med existerande celler, men det här ändrar troligen utseendet också.

Undersökningen har gett resultat som står i kontrast till tidigare undersökningar som utförts i området av Ek (2017). De fick ett resultat som skilde sig från datainsamlingen i denna undersökning, då de kom fram till var att DLA var snabbare än cellular automata. Efter en noggrann undersökning visade det sig att den varianten av DLA som Ek (2017) implementerade inte var samma variant som har implementerats i denna undersökning. Versionen av Ek (2017) använder sig av väggkrock istället för att krocka med redan existerande celler. Det här betyder att algoritmen inte behöver skapa lika många partiklar, men beter sig också mer som en random walk än en faktisk DLA. Cellular automata algoritmen implementerad av Ek (2017) är också annorlunda från den implementerad i den här undersökningen. Implementationen av Ek (2017) är mycket långsammare då de ändrar på algoritmen för att vara helt sammanhängande, vilket kan ha påverkat prestandaresultatet.

Adams och Louis (2017) hade i syfte att generera en spelbar labyrint med hjälp av cellular automata. Deras resultat var att deras metod för algoritmen passade bra med att skapa

labyrinter. Det här kan troligen användas för grottsystem också, då strukturen av labyrinterna skapade är liknande grottor.

Implementationen av cellular automata gör att den inte har så bra pålitlighet, vilket kan göra det svårt att implementera i ett spel eller något annat som kräver hög pålitlighet. DLAs implementation har hög pålitlighet och en mer justerbar grottmängd, vilket gör den bättre anpassad för spel. Dock passar det här inte spel som kräver extra stora kartor då den är väldigt långsam.

Det största problemet med den här undersökningen var att det inte fanns mycket användbar forskning. Mycket av forskning inom omår

6.3 Samhälleliga och etiska aspekter

I den här undersökningen finns det inte mycket koppling till samhälleliga eller etiska aspekter. Syftet var att få fram den algoritmen som presterar bäst i de specifika utvärderingskriterierna valda. Den valda metoden var också ett systematiskt experiment, vilket betyder att det inte var några testpersoner inblandade. Det är svårt att säga om resultatet av experimentet kan bidra till något i samhället.

Wattanapornprom et al. (2024) tar däremot upp att procedurella genereringsalgoritmer, som cellular automata och DLA, kan användas för att spara tid, pengar och arbetsinsats. Cano, Tardioli och Mosteo (2024) beskriver simulatorer byggda med hjälp av procedurell generering för att testa applikationer av olika system som bättre ett val än att göra test i verkligheten. Det här kan användas för att hjälpa samhället på många olika sätt där nya idéer för samhället kan appliceras men måste testas innan. Som allt automatiserat minskar det här dock jobbmarknaden då mycket av det som utfördes av flera personer kan göras av en person istället.

6.4 Framtida arbete

En fortsättning på undersökningen skulle kunna ha mer algoritmer att jämföra, samt fler kriterier de jämförs inom. På kort sikt kan fler kriterier appliceras. Av de fem viktigaste egenskaperna Togelius (2016) tar upp användes bara två i den här undersökningen. Styrbarheten av algoritmerna, variation i genereringen och trovärdighet av det som genereringen vill visa borde också bli analyserat. Intervjuer från testpersoner skulle kunna ge mer insikt i hur verklighetstrogna och naturliga grottorna faktiskt ser ut, vilket besvarar trovärdigheten av grottorna. På längre sikt kan även algoritmer såsom Perlin Noise och Binary Space Partitioning (BSP) bli utvärderade, där BSP är mest intressant då den inte använder sig av brus eller agenter. Det här skulle ge ett bredare perspektiv på användningsområdena och användbarheten av alla algoritmerna.

Det skulle också varit intressant att ändra vad som genereras av algoritmerna. I den här undersökningen genererades bara en väldigt enkel tvådimensionell grotta. Grottan skulle kunnat bli mer komplicerad med fler egenskaper, som till exempel vatten, samt bli applicerad i realtid för att visa användbarheten av till exempel genereringstiden. Den här applikationen skulle kunna ske antingen i ett spel eller i simuleringar.

Referenser

Antonijevic, Filip (2021). *PGG - Processuell Grottgenerering: En jämförelse mellan Cellulär Automat, Random Walk och Perlin Noise*. http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1573330&dswid=-6276.

Ek, Pontus (2017). *Procedurell generering av grottsystem för dataspel: Jämförelse av procedurellt genererade osymmetriska banor*. http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1129961&dswid=5270.

Rogue (1980). Epyx.

The Binding of Isaac (2011). Headup Games. Tillgängligt på Internet: https://store.steampowered.com/app/113200/The_Binding_of_Isaac/

Minecraft (2009). Mojang Studios. Tillgängligt på Internet: https://www.minecraft.net/

Shaker N., Togelius J., Nelson M., (2016). *Procedural Content generation in Games*. Switzerland, Cham: Springer International Publishing.

Cano, L., Tardioli, D. & Mosteo, A. (2024). Procedural generation of tunnel networks for unsupervised training and testing in underground applications. I *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Abu Dhabi, United Arab Emirates 14-18 October 2024, s. 4608-4615. doi: 10.1109/IROS58592.2024.10801552

Wattanapornprom, W,. Wipachainun, C., Lertpinitamonkul, T,. Suksai, W., Rodkaew, Y. & Susutti, W. (2024) Procedural Content Generation for 2.5D Rogue-Lite Games: An Evolutionary Algorithm Approach. *2024 28th International Computer Science and Engineering Conference (ICSEC)*. Khon Kaen, Thailand 2024, s. 1-6. doi: 10.1109/ICSEC62781.2024.10770719

Manneville, P. (1989). XXXXXXXXXXXXXX. I Boccara, N., Vichniac, G. & Bidaux, R. (red.) *Cellular Automata and Modeling of Complex Physical Systems*. Springer Science & Business Media, s. XX-XX.

Conway's Game of Life (1970). John Conway.

Atoniuk, I. (2024). Procedural generation of cave-like tiles with cellular automata and Blender Geometry Nodes. I *2024 25th International Conference on Computational Problems of Electrical Engineering (CPEE)*. Stronie Śląskie, Poland 2024, s. 1-4. doi: 10.1109/CPEE64152.2024.10720415

Johnson, L,. Yannakakis, G. & Togelius, J. (2010). Cellular automata for real-time generation of infinite cave levels. I *2010 Workshop on Procedural Content Generation in Games (PCGames '10). Association for Computing Machinery*. New York, NY, USA 2010, s. 1-4. doi: 10.1145/1814256.1814266

Huang, Y. & Somasundaran, P. (1987). Effects of random-walk size on the structure of diffusion-limited aggregates. *Physical review A*, 36(9), s. 4518-4521. doi: 10.1103/PhysRevA.36.4518

RogueBasin (2020). *Diffusion-limited aggregation*. https://roguebasin.com/index.php/Diffusion-limited_aggregation [2025-03-12]

Melnyk, R., Havrylko, Y. & Levytsk, M. (2021). Defects Detection in PCB with Ground

Plane by Clustering and Flood-fill Algorithms. I *2021 IEEE 4th International Conference on Advanced Information and Communication Technologies (AICT)*. Lviv, Ukraine 2021, s. 14-17. doi: 10.1109/AICT52120.2021.9628899

Togelius, J. (2016). Introduction. I Shaker, N. & Nelson, M.J. (red.) *Procedural Content Generation in Games*. s. 1-15. doi: 10.1007/978-3-319-42716-4_1

Koulaxidis, G., & Xinogalos, S. (2022). Improving Mobile Game Performance with Basic Optimization Techniques in Unity. *Modelling*, 3(2), s. 201-223. doi: 10.3390/modelling3020014

Lague, Sebastian. (2015). *[Unity] Procedural Cave Generation (E01. Cellular Automata)* [video]. https://youtu.be/v7yyZZjF1z4

Liapis, A., Yannakakis, G. & Togelius, J. (2021). Towards a Generic Method of Evaluating Game Levels. I *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 9(1), s. 30-36. doi: 10.1609/aiide.v9i1.12680

Unity (2022). Unity Technologies.

Herbert Wolverson (2019). *Roguelike Tutorial - In Rust*. https://bfnightly.bracketproductions.com/rustbook/chapter_30.html

Noveltech (2023). *How to use the Diffusion Aggregation algorithm to generate 2D maps for dungeons or biomes*. https://www.noveltech.dev/unity-procgen-diffusion-aggregation

Jason T. Newsom (2007). *Lecture 4 Sample Size*. https://web.pdx.edu/~newsomj/pa551/lecture4.htm

Yoshida, A., Haida, T., Matsumoto, E. & Washio, S. (2001). Effects of density distribution of growing frost layer on radiation transfer. *Heat Transfer - Asian Research*, 30(5), s. 439-450. doi: 10.1002/htj.1030

Witten, T. & Sander, L. (1983). Diffusion-limited aggregation. *Physical Review B*, 27(9), s. 5686-5697. doi: 10.1103/PhysRevB.27.5686

Björklund, Oscar (2016). *Kompakthet av procedurellt genererade grottsystem: En jämförelse av procedurellt genererade grottsystem. https://his.diva-portal.org/smash/get/diva2:934642/FULLTEXT01.pdf.*

Eriksson, Carl (2022). *Procedurell grottgenerering inom dataspel: En jämförelse mellan algoritmer. https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1675089&dswid=-7218*

Adams, C. & Louis, S. (2017) Procedural maze level generation with evolutionary cellular automata. I *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, Honolulu, HI, USA, 2017, s. 1-8, doi: 10.1109/SSCI.2017.8285213

van der Linden, R., Lopes, R., & Bidarra R. (2014). Procedural Generation of Dungeons. I *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 1, s. 78-89, March 2014, doi: 10.1109/TCIAIG.2013.2290371