Examensarbete

HÖGSKOLAN I SKÖVDE

WAVE FUNCTION COLLAPSE I SANDBOX SURVIVAL SPEL

En jämförelse av olika varianter

WAVE FUNCTION COLLAPSE IN SANDBOX SURVIVAL GAMES

A comparison of different variants

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 15 högskolepoäng
Vårtermin 2025

Niklas Johansson

Handledare: Mikael Thieme
Examinator: Sanny Syberfeldt

Sammanfattning

I denna undersökning jämförs olika varianter av wave function collapse i sammanhanget av världsgenerering i sandbox survival spel. I de större spelen i genren så består världen av block arrangerade i ett rutnät vilket leder till den intressanta frågan: Är det möjligt att generera en sådan värld med wave function collapse?

Undersökningen försöker besvara frågan genom en serie mindre experiment där varje experiment undersöker varsin variant av algoritmen i sammanhanget. För att utvärdera varianterna jämförs de med originalet genom att använda olika mätvärden. Resultatet av undersökningen visar att det är möjligt att använda wave function collapse i detta sammanhang. Det finns dock vissa begränsningar som behöver lösas innan algoritmen kan användas fullt ut för ändamålet.

Nyckelord: Wave Function Collapse, Sandbox, Survival, Procedurell Generering

Innehållsförteckning

1 Introduktion ... 1

2 Bakgrund ... 2

2.1 Wave function collapse ... 3

2.2 Sandbox Survival ... 6

3 Problemformulering ... 8

3.1 Metodbeskrivning ... 9

3.1.1 Mätvärden ... 10

3.1.2 Experiment ... 11

3.1.3 Testmiljö ... 11

4 Genomförande ... 13

4.1 Implementation ... 13

4.1.1 Original ... 13

4.1.2 Tillbakagång ... 16

4.1.3 Dynamiska vikter ... 16

4.1.4 Val av rutor ... 17

4.2 Resultat ... 17

4.2.1 Original ... 18

4.2.2 Tillbakagång ... 22

4.2.3 Dynamiska vikter ... 23

4.2.4 Val av rutor ... 25

4.3 Analys ... 27

5 Sammanfattning och diskussion ... 29

5.1 Sammanfattning ... 29

5.2 Diskussion ... 29

5.3 Samhälleliga och etiska aspekter ... 31

5.4 Framtida arbete ... 31

Referenser ... 32

1 Introduktion

Wave function collapse är en algoritm för att procedurellt generera bilder. Algoritmen har fått stor uppmärksamhet vilket har lett till att den har tillämpats i andra sammanhang än att generera bilder. Spelutvecklare visade ett speciellt stort intresse för algoritmen vilket har lett till att den började användas för att generera spelinnehåll. Algoritmen har sina brister vilket har lett till en del forskning kring att förbättra olika aspekter av algoritmen (Alaka & Bidarra 2023; Cheng, Han & Fei 2020; Christie et al. 2024; Kim et al. 2019; Nie et al. 2024; Punia, Priyanka & Sharma 2023; Sandhu, Chen & McCoy 2019). Denna forskning har lett till att olika varianter av algoritmen presenterats. Det är dessa varianter som är fokuset för denna undersökning.

Algoritmen sätts också i ett sammanhang som i detta fall är sandbox survival spel. De största spelen i denna genre bygger på världar som är uppbyggda av block i form av ett rutnät. Detta leder till möjligheten att använda algoritmen för att generera dessa världar. Undersökningen utforskar då om algoritmen kan vara ett alternativ till de traditionella brus som ofta används i sammanhanget. För att besvara frågan jämförs tre olika varianter av algoritmen med originalet i detta sammanhang.

Undersökningen består av en serie mindre experiment som undersöker var sin variant av algoritmen. Varje experiment mäter skillnaden mellan originalet och varianten som undersöks. En del utmaningar upptäcktes även vid implementeringen av originalvarianten som också beskrivs i undersökningen. De varianter som undersöks är tillbakagång, dynamiska vikter samt olika sätt att välja rutor inför observation.

Analysen av resultatet ifrån dessa experiment visar att det fanns vinster med alla varianter. Dock var inte alla fallen helt gratis och vissa negativa effekter upptäcktes. Resultatet visar att algoritmen kan användas för att generera världen i sandbox survival spel. Det finns dock många aspekter av algoritmen som behöver förbättras för att den skall kunna utnyttjas fullt ut. Forskningen kring algoritmen presenterar en del lösningar, men det finns fortfarande stora möjligheter att bedriva mer forskning.

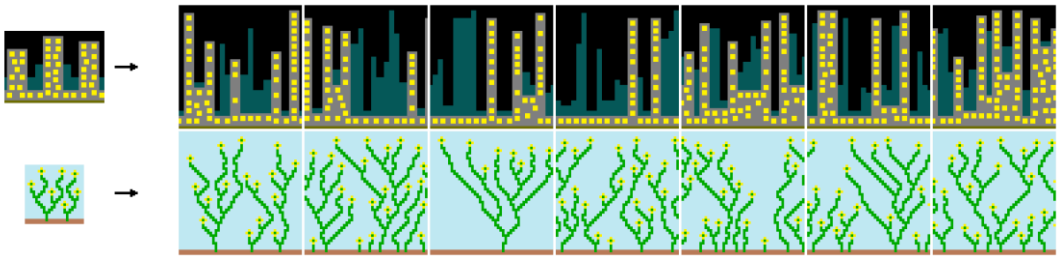
2 Bakgrund

En aspekt av spelutveckling som kan ta lång tid är skapandet av spelens innehåll. Detta har lett till utforskandet av metoder och tekniker för att skapa spelinnehåll med algoritmer i stället för mänskligt arbete. Algoritmerna sparar tid och gör det möjligt att skapa större mängder spelinnehåll på bekostnaden av att resultatet kan bli enformigt eller tråkigt. Detta område kallas för procedurell generering vilket är ett stort område som kan användas för att skapa många olika typer av spelinnehåll. Procedurell generering kan även användas för att skapa spelinnehåll under spelets gång vilket leder till unika möjligheter. Genereringen kan då användas för att ge spelaren mycket större variation och därmed mycket mer omspelbarhet. Ett spel som verkligen utnyttjar detta är Minecraft (2011). En skärmdump av spelet kan ses i Figur 1. Mycket intresse finns för området och en hel del forskning har bedrivits för att utveckla nya eller förbättra existerande metoder.

Figur 1: En skärmdump av spelet Minecraft (2011).

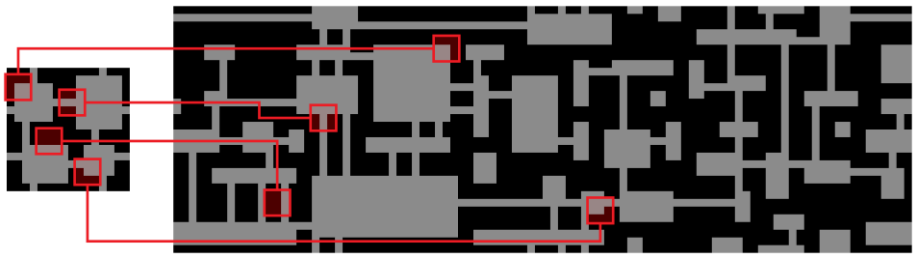
2.1 Wave function collapse

En algoritm för procedurell generering är wave function collapse (WFC). Algoritmen skapades av Gumin (2016) och är baserad på model synthesis (Merrell & Manocha 2011) men inspiration togs även ifrån kvantfysiken som ledde till namnet. WFC går ut på att genom ett exempel skapa en mängd mönster samt regler för hur mönstren får sättas ihop för att sedan generera en ny bild med liknande egenskaper. Algoritmen jobbar dock inte direkt med bilder utan använder sig av rutnät (tilemaps på engelska) där varje ruta har en specifik typ. Rutnätet kan sedan visualiseras genom att fylla i varje ruta beroende på rutans typ. Både exemplet och resultatet är ett sådant rutnät. Algoritmen jobbar därför inte med pixlar eller färger och det är upp till utvecklaren att visualisera rutnäten. Mönstren som skapas utav exempelrutnätet är en kvadratisk del av rutnätet där alla mönster har samma storlek. Exempel på resultat av algoritmen kan ses i Figur 2.



Figur 2: Exempel på nya bilder som genererats utifrån exempel. Bilden är tagen ifrån Github sidan för originalalgoritmen (Gumin 2016) och har modifierats för att bättre passa detta dokument.

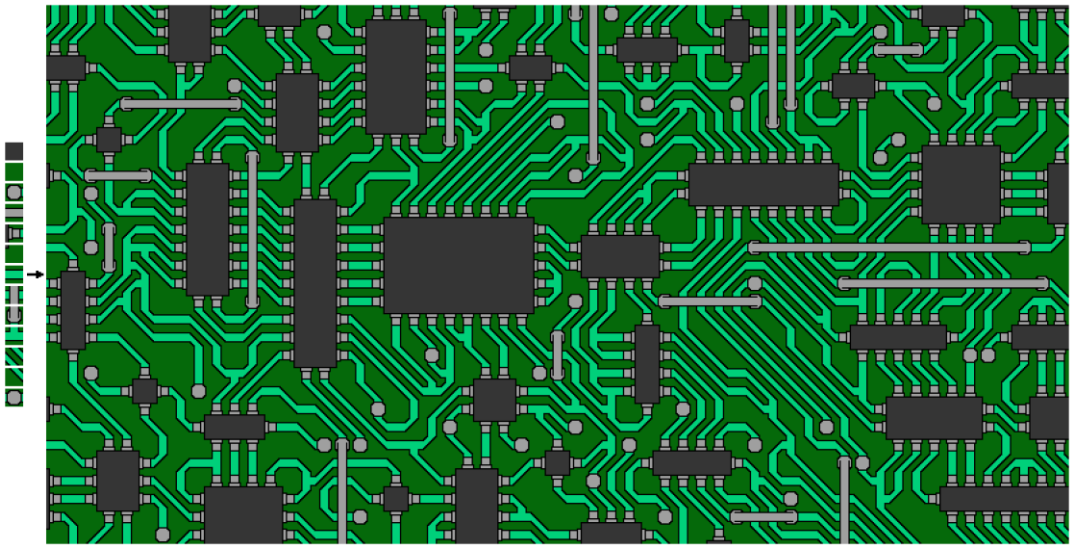
Det finns två olika varianter av algoritmen som skiljer sig i hur mönstren och deras regler skapas. I den överlappande varianten går algoritmen stegvis igenom alla rutor i exempelbilden och för varje ruta skapas ett mönster som innehåller rutan samt nästkommande rutor. Hur många rutor som inkluderas beror på mönstrets storlek. Storleken på mönstren i denna variant är alltid större än ett vilket betyder att mönstren överlappar varandra. För att sedan skapa reglerna används helt enkelt grannarna för varje mönster. Mönster får då endast sättas ihop om mönstren sitter ihop i exempelrutnätet. Detta leder också till en indirekt viktning av mönstren. Om ett mönster är vanligt i exempelrutnätet kommer det också vara det i resultatet och tvärt om. Relationen mellan exempelbilden och resultatet kan ses i Figur 3.



Figur 3: En bild som visar relationen mellan exemplet och resultatet. Bilden är tagen ifrån Github sidan för originalalgoritmen (Gumin 2016) och har modifierats för att bättre passa detta dokument.

Den andra varianten är den enkla rutbaserade varianten där mönstren är av storlek ett. Ett mönster blir då detsamma som typen av den ruta som mönstret avser. Ett exempel på denna variant kan ses i Figur 4. Ett exempelrutnät kan även användas här, men då kommer mönstren inte att överlappa. Det blir dock mer praktiskt att lagra mönstren och deras regler i ett dataformat (Gumin 2016), där originalet använder XML. Detta leder dock till att den indirekta viktningen av mönstren i exempelrutnätet förloras. Vikterna specificeras därför med direkta värden tillsammans med mönstren i dataformatet.

Den enkla rutbaserade varianten är inte nödvändigtvis sämre eller bättre än den överlappande varianten. Varianten är helt enkelt ett annat sätt som algoritmen kan utföras på. Den resulterar i en enklare implementation och mer kontroll över regler och vikter. Nackdelen blir att tid behöver spenderas på att lista ruttyper, regler och vikter i ett dataformat. Grunden för den enkla rutbaserade varianten blir då en samling av ruttyper, regler för vilka ruttyper som får vara bredvid varandra samt vikter för alla ruttyper. Det är denna variant som utgör originalalgoritmen i denna undersökning. Begreppet mönster kommer också att ersättas helt med begreppet ruttyper då de i denna variant inte innebär någon skillnad. Härnäst kommer en beskrivning av WFC algoritmen utifrån denna variant.



Figur 4: Exempel på ett resultat av den enkla rutbaserade varianten. Bilden är tagen ifrån Github sidan för originalalgoritmen (Gumin 2016) och har modifierats för att bättre passa detta dokument.

Första steget i algoritmen är att ladda ruttyperna samt deras regler och vikter. I originalalgoritmen görs detta genom att läsa en XML fil. Efter att detta laddats klart skapas ett rutnät där varje ruta har en lista med olika typer som får lov att väljas för den rutan. Till en början initialiseras rutornas valmöjligheter med alla ruttyperna. Efterhand kommer valmöjligheterna att minska och till slut finns det bara ett val för alla rutor. Detta blir då resultatet av algoritmen och kan sedan visualiseras till en bild.

Efter att rutnätet har initialiserats börjar en loop som slutar när algoritmen är klar eller en motsägelse har upptäckts. Det första steget i loopen är att hitta nästa ruta att observera. Detta kan göras på lite olika sätt. Ett sätt är att köra rad för rad. Ett annat sätt är att ta den ruta med minst entropi där entropi är ett nummer som beräknas utifrån vikterna av de valmöjligheterna som rutan har. Entropi i detta sammanhang baseras på Shannon entropi som kommer från informationsteorin. I praktiken fokuserar entropi på att välja de rutor med minst valmöjligheter där vikterna av valmöjligheterna också tas med i beräkningen. Rutorna som väljs måste också ha mer än en valmöjlighet då de annars inte behöver observeras.

Om inga rutor hittas betyder det att alla rutor endast har en valmöjlighet och algoritmen är klar. Om en ruta hittas så går algoritmen vidare till observationssteget. Här väljs en av de valmöjligheter som rutan har på ett slumpmässigt sätt. Valet görs med hänsyn till valmöjligheternas vikter. Efter att valet gjorts tas resterande valmöjligheter bort ifrån rutans lista. Till sist propageras den nya informationen till resterande rutor. Detta görs genom att kontrollera vilka valmöjligheter som nu enligt reglerna inte kan väljas i närliggande rutor. De valmöjligheter som inte kan väljas tas bort och leder till ytterligare propagering.

När all ny information har spritt sig över rutnätet kontrolleras det om det finns en motsägelse. En motsägelse inträffar om någon av rutorna har blivit av med alla valmöjligheter. Det går då inte att fortsätta och algoritmen börjar om genom att återställa rutnätet. Om ingen motsägelse hittas, så börjar loopen om med att hitta nästa ruta att observera. Som tidigare nämnts avslutas algoritmen när alla rutor endast har en valmöjlighet. Pseudokod för algoritmen kan ses i Figur 5.

```
1   Ladda ruttyperna, reglerna och vikterna
2   Initialisera rutnätet med alla valmöjligheter
3   Repetera:
4       Hitta en ruta med fler än en valmöjlighet
5       Om en ruta inte hittades:
6           Klar
7       Observera rutan genom att välja en valmöjlighet
8       Propagera ändringar till närliggande rutor
9       Om propageringen resulterade i en motsägelse:
10          Starta om
```

Figur 5: Den enkla rutbaserade varianten i pseudokod.

Utvecklare har visat ett stort intresse för WFC (Karth & Smith 2017) vilket har lett till att den har adapterats till andra användningsområden än att generera bilder. Den har till och med använts för att skapa poesi (O'Leary 2017). Spelutvecklare var speciellt intresserade av algoritmen och algoritmen blev snabbt anpassad för att generera spelinnehåll som till exempel banor eller världar. Spel som använder algoritmen inkluderar Bad North (2018), Townscaper (2021) och Caves of Qud (2024). Med sin popularitet har även en del forskning kring algoritmen gjorts.

En artikel beskriver olika strategier för att öka kontrollen för en designer (Sandhu, Chen & McCoy 2019). Strategierna involverar bland annat olika sätt att hantera vikterna samt icke lokala begränsningar. En liknande undersökning har också gjorts som bland annat föreslår sätt att öka den globala kontrollen (Cheng, Han & Fei 2020). En annan artikel beskriver ett intressant koncept av att skapa en hierarki av ruttyper i stället för en platt lista (Alaka & Bidarra 2023). Detta skapar unika möjligheter som till exempel att kunna markera områden såsom vatten, by eller väg och låta algoritmen lista ut vad de skall innehålla samt hur de hänger ihop.

Flera projekt har fokuserat på att dela upp resultatet i bitar som sedan kan delvis genereras samtidigt och i isolation (Christie et al. 2024; Nie et al. 2024). Detta för att minska mängden av konflikter och möjliggöra parallell generering. I den ena artikeln presenteras också en klassificering av den samling av ruttyper som används av algoritmen där de genom matematik bevisar egenskaper hos klasserna (Nie et al. 2024). För att spara ännu mer tid har även olika varianter av tillbakagång studerats (Christie et al. 2024) samt att implementera algoritmen med bitoperationer (Punia, Priyanka & Sharma 2023).

Det har också forskats på att kombinera WFC med andra tekniker som till exempel genetiska algoritmer (Bailly & Levieux 2023) samt traditionellt brus (Büyükşar, Yıldız & Demirci 2024). Ett annat projekt har översatt algoritmen till att fungera med grafer (Kim et al. 2019). Detta leder till att rutorna inte behöver vara i form av ett rutnät och kan ha olika många noder som grannar. Det har även forskats på att undersöka algoritmen i olika sammanhang som till exempel pusselspel (Kim, Seo & Kang 2024). En annan undersökning som inspirerat detta projekt undersöker algoritmen i sammanhanget av roguelite spel (Wang & Chang 2024).

2.2 Sandbox Survival

Sandbox survival är en genre som är en kombination av de två genrerna i dess namn. Den första genren ger spelaren frihet att utforska och leka runt i en värld där endast fantasin sätter gränsen. Genom att låta spelaren manipulera världen genom att till exempel bygga fritt, förstöra terrängen eller placera dekorationer blir spelet mer öppet för kreativitet. Den andra genren fokuserar på att utmana spelaren att överleva. Det kan handla om att behöva samla resurser, utforskning av farliga områden, bli starkare genom tillverkning av utrustning eller slåss mot fiender som försöker hindra dig. Exempel på klassiska spel inom genren är Minecraft (2011) och Terraria (2011).

Då denna undersökning fokuserar mest på tvådimensionella spel kommer Terraria (2011) att beskrivas i mer detalj. Spelet är ett så kallat sidescroller spel vilket betyder att spelaren ser spelet ifrån sidan. Kameran följer spelarens karaktär som befinner sig i en värld bestående av ett rutnät där varje ruta innehåller olika typer av block. Spelaren kan placera och ha sönder blocken vilket leder till att terrängen helt kan modifieras till spelarens tycke. En skärmdump ifrån spelet kan ses i Figur 6. Målet med spelet är att samla resurser, tillverka ny utrustning och besegra stora bossar men spelaren ges även friheten att vara kreativ i till exempel hur dess hem skall se ut.

Figur 6: En skärmdump av spelet Terraria (2011).

3 Problemformulering

Det problem som undersöks i denna undersökning handlar om att sätta en befintlig algoritm i ett nytt sammanhang. I detta fall är det algoritmen wave function collapse som sätts i sammanhanget av världsgenerering i sandbox survival spel. Problemet inspireras av en liknande undersökning som sätter wave function collapse i sammanhanget av roguelite spel (Wang & Chang 2024). Sammanhanget av sandbox survival spel begränsas ytterligare till de spel där världen består av fyrkantiga block arrangerade efter varandra i ett tvådimensionellt rutnät. Motivering för begränsningen är att det blir lättare att arbeta med algoritmen samt att visualisera resultatet om andra dimensioner och former på block bortses ifrån. Begränsningen resulterar heller inte i att resultatet av denna undersökning blir mindre relevant. Många typer av spel i flera genrer använder sig av denna typ av värld. Den är framför allt vanligast i tvådimensionella spel. Ett spel som har denna typ av värld och som utgör sammanhanget i denna undersökning är Terraria (2011).

Valet av wave function collapse som algoritm gjordes då den använder sig utav ett helt annat tillvägagångssätt än de brus som traditionellt sett har använts i sammanhanget. Detta leder till nya intressanta utmaningar och nya egenskaper hos resultatet. Användandet av algoritmen i sammanhanget skulle kunna resultera i mer intressanta världar samt bättre kontroll för utvecklaren. Originalvarianten i denna undersökning är som tidigare nämnt den enkla rutbaserade varianten. Motiveringen till att just denna valdes är för att den är lättare att implementera samt att den passar bättre för den typ av spel som undersöks. Det blir dock mindre intressant att bara undersöka denna variant. Det har ju skett en del forskning kring att förbättra olika aspekter av algoritmen som har lett till andra varianter som också bör utforskas.

Genom att tillämpa olika modifikationer eller påbyggnader uppstår nya varianter av algoritmen som kan undersökas. De modifikationer eller påbyggnader av algoritmen som undersöks i denna undersökning är de som är mest relevanta för sammanhanget. Mer specifikt är det tillbakagång, dynamiska vikter samt olika sätt att välja rutor som undersöks. De två förstnämnda baseras på tidigare forskning som introducerar koncepten (Christie et al. 2024; Sandhu, Chen & McCoy 2019) medan den sista involverar de sätt som finns med i originalet. Det behövs även relevanta mätvärden för att kunna utvärdera de varianter som undersöks. I denna undersökning används exekveringstid, blockdistribution och antalet motsägelser. Med en förklaring av båda delarna av problemet blir problemformuleringen i form av en frågeställning: Hur påverkar olika varianter av wave function collapse världsgenereringen i sandbox survival spel?

3.1 Metodbeskrivning

Den metod som valts för att undersöka problemet är en serie av mindre experiment. Andersson (2012, s. 83) definierar ett experiment som det strategiska manipulerandet av ett system för att skapa en organiserad respons för att kunna besvara en specifik fråga. Metoden valdes då fokuset ligger på de kvantitativa aspekterna av genereringen i stället för upplevelsen av eventuella spelutvecklare eller spelare av spelet. Systemet i detta fall blir världsgenereringen och manipulerandet sker genom att tillämpa en modifikation eller påbyggnad. Frågan som besvaras varierar mellan experimenten men svaren av dessa bidrar till besvarandet av den övergripande frågeställningen. Metoden leder också till praktisk erfarenhet och kunskap som kan ligga till grund för andra spelutvecklare. Det är även samma metod som användes i den undersökning som inspiration har tagits ifrån (Wang & Chang 2024).

Varje experiment undersöker var sin variant som består av originalvarianten tillsammans med en modifikation eller påbyggnad och följer samma struktur. Först implementeras originalvarianten samt den variant som experimentet undersöker. Implementationerna representerar tillståndet av systemet före och efter manipulationen. Olika mätvärden används sedan för att kunna se hur manipuleringen har påverkat systemet. De kvantitativa mätvärdena är olika för varje experiment då inte alla mätvärden är intressanta att mäta för den variant som undersöks. Detta för att syftet med varje experiment är att undersöka hur en variant påverkar en specifik aspekt av algoritmen. Det mätvärde som valts för varje experiment är då det som bäst fångar eventuella skillnader inom denna aspekt. Till exempel blir blockdistributionen inte relevant för tillbakagång då den skulle vara oförändrad. Skulle blockdistributionen däremot användas i det tredje experimentet så är det inte omöjligt att en viss skillnad uppstår. Denna skillnad är dock inte relevant för experimentets syfte som undersöker hur många gånger som algoritmen startar om. En tabell med varianterna som undersöks samt deras mätvärden kan ses i Figur 7.

Variant	**Mätvärde**
Tillbakagång	Exekveringstid
Dynamiska vikter	Blockdistribution
Olika sätt att välja rutor	Antal motsägelser

Figur 7: De varianter som undersöks samt deras mätvärden.

De mätvärden som skall användas i experimentet mäts sedan för båda varianterna av algoritmen. För att kunna visualisera den data som samlats in presenteras den i ett sammanslaget och överskådligt format. Detta möjliggör en analys av eventuella skillnader mellan varianterna. I vissa experiment används KDE grafer för detta ändamål. KDE eller kernel density estimation är en typ av graf som likt ett histogram beskriver distributionen av data men gör detta med en kontinuerlig kurva. Valet av att använda KDE grafer inspirerades av undersökningen som tittade på sammanhanget av roguelite spel (Wang & Chang 2024).

Ett potentiellt problem med detta upplägg kan vara att det valda mätvärdet inte visar den förändring som skett. Valet av mätvärde blir då avgörande för att få ett intressant resultat. Andersson (2012, s. 176) beskriver också hur det är viktigt att se upp för bakgrundsfaktorer som kan leda till förändringar som inte var ett resultat av experimentets manipulering. Som ett exempel skulle exekveringstiden för en algoritm kunna påverkas av hur belastad processorn är med andra uppgifter. Stor vikt behöver därför läggas på att minimera dessa bakgrundseffekter i de mätningar som görs.

3.1.1 Mätvärden

De mätvärden som används i experimenten är exekveringstid, blockdistribution och antal motsägelser. Valet av dessa mätvärden baseras på de varianter som experimenten undersöker. Alla mätningar görs sedan på två olika storlekar på världen för att se om storleken har någon effekt på mätningen. De storlekar som används är 32x32 och 64x64. Anledningen till att endast två storlekar har valts är på grund av de typer av diagram som används för att visualisera den data som samlas in. De typer som valts resulterar i att det behövs ett diagram för varje storlek. Två storlekar valdes därför för att kunna se om storleken har någon effekt men också för att minimera antalet diagram. För varje mått och storlek på världen så mäts sedan varje mätvärde ett hundra gånger.

Exekveringstid är den tid som algoritmen använder för att utföra sitt jobb. Den mäts genom att beräkna skillnaden mellan en absolut tid före och efter exekveringen. Flera mätningar (100 stycken) utförs sedan för varje storlek på världen för att minska bakgrundseffekterna som Andersson (2012, s. 176) beskrev. Mätningarna presenteras genom att använda en KDE graf för varje storlek på världen.

Blockdistributionen handlar om hur många block av varje typ som finns med i resultatet. För att mäta distributionen räknas först antalet block av varje typ ut. För varje blocktyp beräknas sedan den andel av alla block som var av den typen. Skillnaden mellan detta värde och den vikt som blocktypen hade ifrån början blir felet för denna typ. Felet är absolutbeloppet av skillnaden då måttet inte tar hänsyn till om typen är över- eller underrepresenterad. Ett medelvärde för felen hos alla typer räknas sedan ut och det är detta värde som är blockdistributionen för världen. Dock med endast en mätning kan det bli svårt att avgöra vad för effekt som manipuleringen har gett (Andersson 2012, s. 176). Därför mäts distributionen på flera (100 stycken) världar som genererats. För att visualisera dessa mätningar används återigen KDE grafer. Risken med detta mått är att finare detaljer kan gå förlorade, men fördelen är att det ger en bättre överblick av blockdistributionen i sin helhet.

Det sista mätvärdet som är antalet motsägelser används för att se hur många gånger algoritmen måste starta om på grund av en motsägelse. Antalet motsägelser räknas sedan för flera (100 stycken) exekveringar. Detta för att kunna se hur ofta olika mängder av motsägelser påträffats. Mätvärdet påverkas till stor del av den valda blocktypssamlingen men kan också påverkas av olika varianter av algoritmen vilket gör måttet relevant för undersökningen. Antalet motsägelser presenteras genom ett stapeldiagram med staplar för olika antal. Höjden på staplarna representerar sedan hur många gånger som det antalet förekommer.

3.1.2 Experiment

De varianter som undersöks i experimenten är originalvarianten tillsammans med tillbakagång, dynamiska vikter och olika sätt att välja rutor. Tillbakagång beskrivs i tidigare forskning (Christie et al. 2024) och är ett sätt att minska antalet omstarter av genereringen. Detta genom att stega tillbaka lite och se om det går att välja andra alternativ som inte ger en konflikt. Det relevanta mätvärdet här är exekveringstid då färre omstarter kan leda till snabbare generering.

Dynamiska vikter innebär att vikterna av de olika blocken beräknas om undertiden som genereringen sker. En variant på detta beräknar om vikterna runt om ett utplacerat block av en viss typ (Sandhu, Chen & McCoy 2019). Den variant som används i denna undersökning beräknar dock om alla vikter efter varje utplacering. Detta för att ge en bättre distribution av blocktyper för världen som genereras. Om en typ av block är underrepresenterad får den en större chans att väljas. Om typen är överrepresenterad minskar chansen. Distributionen av blocktyper är speciellt viktigt i den genren som undersöks då spelaren troligtvis behöver vissa typer av block för att kunna spela spelet på ett vettigt sätt. Mätvärdet som används i detta fall är blockdistribution.

Det sista experimentet undersöker hur valet av rutorna som skall observeras påverkar antalet motsägelser för den valda blocksamlingen. I originalalgoritmen kan rutorna väljas med hjälp av entropi eller genom att välja rutorna i ordning rad efter rad. I detta experiment undersöks båda alternativen där både ovanifrån och ner samt nerifrån och upp studeras för rad för rad alternativet. Syftet med experimentet är att identifiera det sätt som resulterar i minst antal motsägelser. Det blir viktigt att minimera antalet motsägelser eftersom de påverkar exekveringstiden negativt då varje motsägelse innebär förlorad tid. Det är till och med möjligt att få flera motsägelser i rad vilket påverkar exekveringstiden ytterligare. Måttet för detta experiment är antalet motsägelser för varje sätt att välja nästa ruta att observera.

3.1.3 Testmiljö

Testmiljön som experimenten utförs i är baserad på programmeringsspråket Java. Valet av språk eller plattform bör dock ha minimal effekt på de mätningar som experimenten utför så länge det är moget och väl etablerat. Detta för att undersökningen jämför på ett relativt sätt genom att använda samma testmiljö för alla experimenten. Samtliga experiment använder alltså språket Java och utförs på samma dator. Motiveringen för att just Java valdes är att språket inte är ett lågnivåspråk vilket leder till att tid kan sparas vid implementeringen som i stället kan spenderas på annat. Nackdelen med språket kan vara att det blir svårare att mäta exekveringstiden då språket använder sig utav skräpsamling. Det är också viktigt att ha i åtanke att den absoluta exekveringstiden kan variera kraftigt beroende på hårdvara, språk eller andra aspekter av testmiljön. Detta enskilda värde blir då mindre användbart än det relativa värde som en jämförelse ger.

För att kunna använda wave function collapse behövs också blocktyper, reglerna för dessa samt deras vikter. I denna undersökning används Terraria (2011) som utgångspunkt. Spelet är dock väldigt rikt på innehåll med flera hundratals olika blocktyper. Denna undersökning använder därför endast en liten delmängd av dessa blocktyper. Blocksamlingen som används är dock fortfarande relativt representativ då flera olika typer av saker har inkluderats. För att visualisera blocktyperna används originaltexturerna från spelet. Reglerna för hur blocktyperna får sättas ihop samt vikterna av blocktyperna är sedan baserade på en normal genererad värld i Terraria (2011). Som tidigare nämnt finns det också forskning på klassificering av blocktypssamlingar där vissa egenskaper har bevisats för klasserna (Nie et al. 2024). Denna undersökning baserar inte valet av regler eller blocktyper på att passa in i någon sådan klass. Detta kan leda till negativa eller positiva effekter men är inte det som undersökningen fokuserar på.

Alla experimenten har utförts på en dator avsedd för hemmabruk med operativsystemet Windows 10. Processorn är en ”Intel Core i7-14700KF 3.40 GHz” och mängden RAM som installerats är 32 GB. Versionen på java som använts är ”Java 21.0.1 2023-10-17 LTS”. Under experimentens utförande har övriga applikationer minimerats för att få så bra mätningar som möjligt.

4 Genomförande

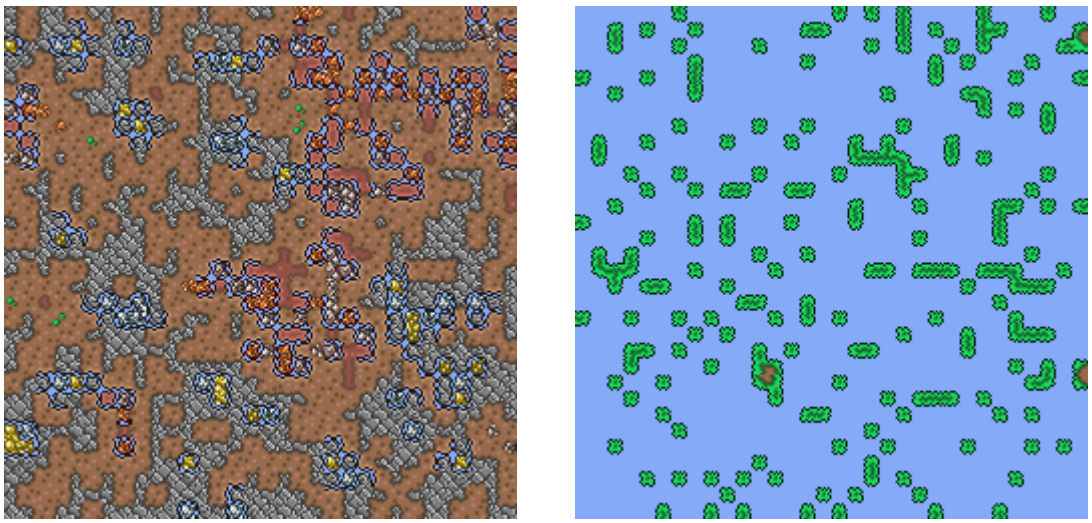
Härnäst beskrivs genomförandet av undersökningen. Först beskrivs implementationen av algoritmens varianter och deras mätningar. Den data som samlats in presenteras sedan som ett resultat som i sista delen analyseras. Även genomförandet och resultatet av originalvarianten finns med. Detta för att ha något att jämföra med men också för att beskriva de intressanta utmaningar och upptäckter som gjordes under genomförandet.

4.1 Implementation

Algoritmen implementerades genom att översätta referensimplementationen (Gumin 2016) till Java. Endast mindre skillnader förekommer som underlättar implementeringen av varianterna samt gör det lättare att arbeta med algoritmen. Exempel på skillnader är att implementationen är lite mer objektorienterad samt att entropin inte lagras och beräknas på nytt varje gång. De varianter som undersöks implementerades sedan i en sammanslagen algoritm. Detta i stället för att implementera de olika varianterna var för sig vilket skulle öka komplexiteten och tidsåtgången vid insamlingen av data. De olika varianterna av algoritmen utgörs av olika inställningar som slår på eller av delar av algoritmen. Med detta upplägg kan det bli så att även om en inställning är avslagen kan den påverka resultatet. Det kan till exempel handla om att extra kontroller görs om inställningen är på eller av som påverkar exekveringstiden. Dessa effekter borde dock vara minimala och är ett litet pris att betala för det tidssparandet och flexibiliteten som en ihopslagen algoritm kan ge.

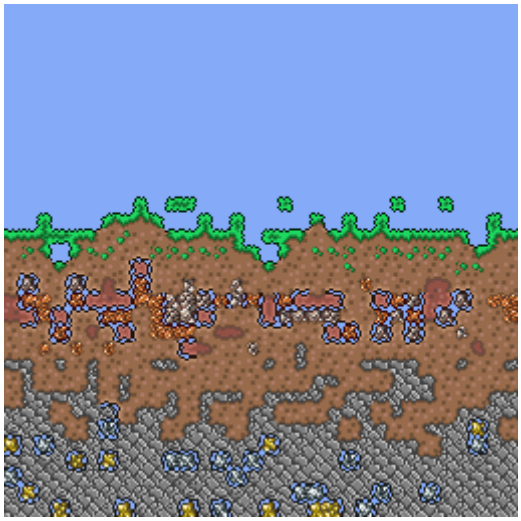
4.1.1 Original

Originalalgoritmen implementerades enligt tidigare beskrivning och resultatet av denna implementation kan ses i Figur 8. Det går tydligt att se hur resultatet inte liknar en representativ Terraria (2011) värld. Resultatet liknar mer brus och det verkar vara svårt för genereringen att få med de olika delarna av världen. I de flesta fallen genereras endast en del av världen som till exempel delen över ytan eller delen under ytan. Det blev då självklart att detta blir en utmaning på vägen till en representativ värld.



Figur 8: Två olika resultat av originalalgoritmen. I det vänstra resultatet finns mest underjordiska block. I det högra resultatet finns endast gräs och luft.

För att lösa detta problem skapades en vertikal distribution av vikten för varje blocktyp. Det vill säga för varje rad i rutnätet har blocktyperna en andel av dess vikt där summan av alla andelarna blir lika med blocktypens ursprungliga vikt. Den slutgiltiga vikten av en blocktyp beror därför på vilken höjd som blocktypen skall placeras på. Det går då att specificera både hur vanlig blocktypen är och hur vanligt blocktypen är på specifika höjder. Resultatet av detta kan ses i Figur 9. Världen är betydligt mer lik en representativ värld, men kan fortfarande bli bättre. Även om världen nu består av de lager som finns med i spelet blir lagren i sig fortfarande mer likt brus. Lösningen har då bara delvis löst problemet och ytterligare en lösning behöver implementeras.

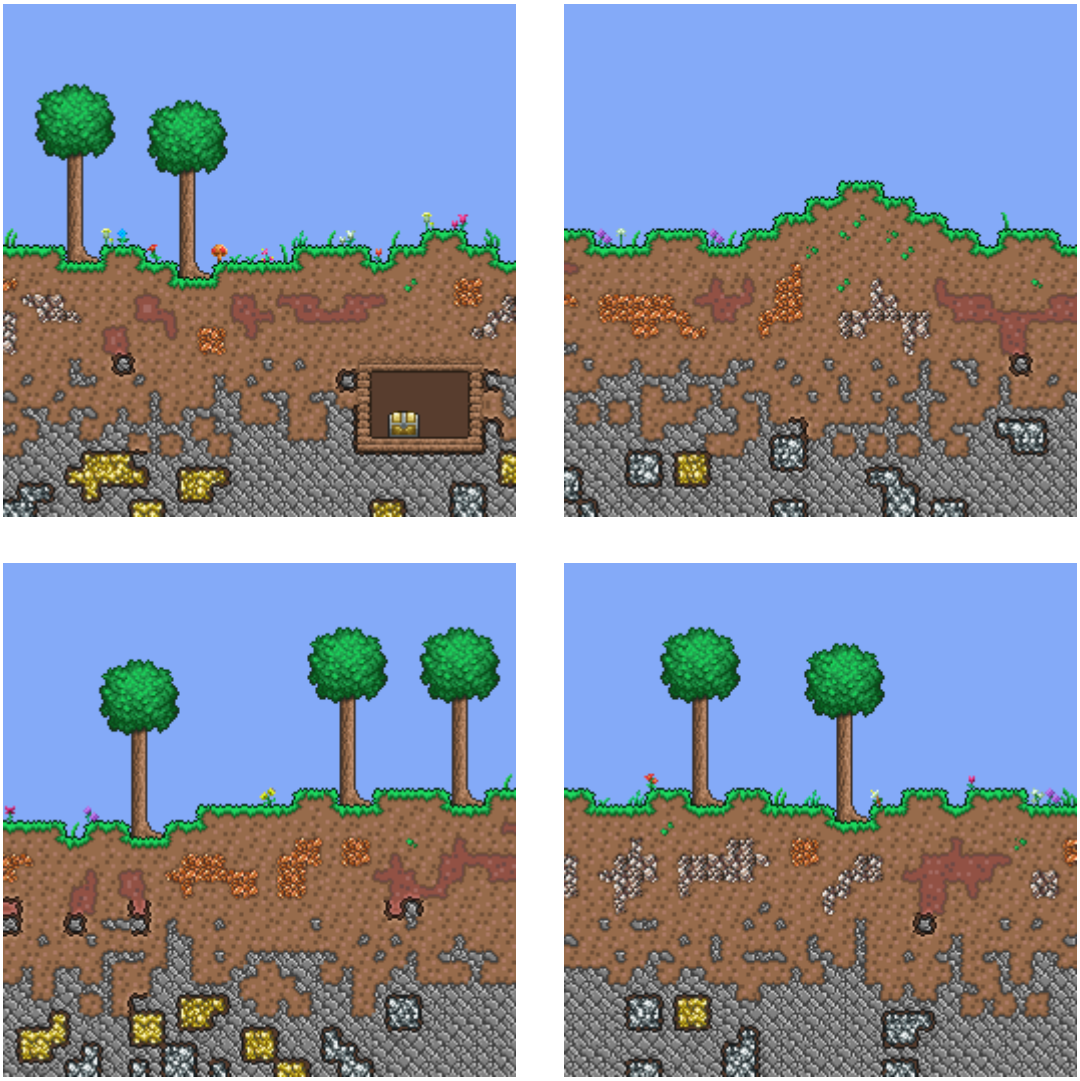


Figur 9: Resultatet av den vertikala distributionen av vikterna.

Problemet är att det inte finns något sätt att specificera hur tjocka klumparna av vissa blocktyper skall vara. Ofta blir det bara små klumpar med ett fåtal block. Det blir också svårt att skapa större och mer komplexa saker som underjordiska rum eller träd.

Dessa problem löstes genom att modifiera blocktyperna och deras regler. Ett antal grupper av blocktyper skapades där blocktyperna i en grupp motsvarar samma blocktyp i resultatet men har olika regler. Detta möjliggör en större räckvidd av kontroll som tidigare inte var möjligt. Nästa steg blev att använda denna teknik för att skapa klumpar som alltid har en viss minsta storlek. Detta gjordes genom att med regler skapa en sorts kedja av blocktyper. Om kedjan påbörjas måste den fortsätta och på så vis uppnås en viss minsta storlek på klumparna. För att undvika raka streck så skapades helt enkelt inte de länkar där ändarna befann sig i motsatta riktningar. En viss anpassning av reglerna gjordes också för hur länkarna får vara placerade bredvid varandra vilket resulterade i mer runda klumpar.

På ett liknande sätt så skapades underjordiska rum. Rummets kanter är kedjor av olika blocktyper som måste påbörjas och avslutas i rummets hörn. Genom att variera antalet länkar i kedjorna kan dimensionen på rummet styras. Träd skapades också med denna teknik. Dessa var helt enkelt en mängd blocktyper där varje blocktyp representerar en del av trädet. Strukturen på trädet specificeras sedan med regler som endast tillåter en korrekt montering av bitarna. Till sist skapades även lite dekorationer som består av enkla blocktyper och i vissa fall en liten grupp av blocktyper som liknar hur träden är konstruerade. Detta användande av blocktyper och regler blir nödvändigt då algoritmen i sig har låg kontroll utanför varje ruta. Resultatet av dessa ändringar kan ses i Figur 10. Efter alla modifieringar som gjorts bedömdes världen vara tillräckligt representativ för att kunna användas som grund vid jämförelser av andra varianter av algoritmen.



Figur 10: Det slutgiltiga resultatet av originalalgoritmen efter modifikationerna.

4.1.2 Tillbakagång

Tillbakagången implementerades genom att skapa en stack där varje element sparar alla ändringar som gjorts under propageringsfasen. Detta resulterar i möjligheten att kunna återgå till tillståndet före en propagering. Varje element kan också ha en referens till den ruta som observerades innan propageringen samt vilket alternativ som valdes. När en motsägelse sedan inträffar så hämtas nästa element i stacken. Med den sparade informationen återställs tillståndet till innan propageringen. Om elementet inte innehåller någon observation börjar tillbakagången rekursivt om igen. Innehåller däremot elementet en observation så tas det valda alternativet bort ifrån valmöjligheterna för den observerade rutan och en ny propagering sker. Skulle propageringen misslyckas betyder det att alternativet som togs bort var det enda valet och tillbakagången börjar rekursivt om igen även här. Då propageringen i tillbakagången inte är ett resultat av en observation kommer denna återställas och tillbakagången påbörjar nästa element i stacken. Skulle det i något fall inte finnas mer element i stacken betyder det att det inte finns någon korrekt lösning och algoritmen avbryts.

Måttet som används för att utvärdera tillbakagång är exekveringstid. Tiden mättes med Javas inbyggda tidsfunktion ”System.nanoTime()”. Denna funktion returnerar det nuvarande värdet i nanosekunder för den högupplösta tidskälla som kommer ifrån den Java Virtual Machine som exekverar programmet. För att mäta exekveringstiden för algoritmen sparades detta värde före och efter ett funktionsanrop till algoritmen. Skillnaden mellan dessa värden blir exekveringstiden. Värdet omvandlades sedan till millisekunder.

4.1.3 Dynamiska vikter

För att implementera dynamiska vikter lagras det antal block som har observerats samt hur många av varje blocktyp som har observerats. Denna information används sedan i funktionen som returnerar vikten för en blocktyp. Först räknas andelen ut genom att dividera antalet av blocktypen som har observerats med det totala antalet observerade block. Om blocktypen inte ännu har observerats sätts denna andel till noll. Blocktypens ursprungliga vikt subtraheras sedan med denna andel och resultatet adderas med ett. Detta värde multipliceras med blocktypens ursprungliga vikt för att få den slutgiltiga vikten. Detta resulterar i att om blocktypen är underrepresenterad blir vikten större och om den är överrepresenterad blir vikten mindre. Detta sker på ett linjärt sätt och anledningen till att ett adderas är att entropin förutsätter att inga vikter är noll. Adderingen gör det omöjligt att vikten blir noll oavsett hur överrepresenterad blocktypen är.

Det mått som mäter skillnaden i detta fall är blockdistribution. Efter att ett resultat har producerats av algoritmen beräknas blockdistributionen enligt den tidigare beskrivningen av mätvärdet. Förhoppningen är detta fel blir mindre med dynamiska vikter.

4.1.4 Val av rutor

De tre olika sätten att välja rutor inför en observation implementerades genom enkla if satser som styrs av en inställning. De tre olika alternativen väljer sedan nästa ruta på olika sätt. I det ena alternativet väljs rutorna med entropi enligt referensimplementationen (Gumin 2016). De två andra alternativen väljer rutorna i ordning och rad för rad. En väljer rutorna ifrån vänster till höger och uppifrån och ner medan den andra väljer rutorna ifrån höger till vänster och nedifrån och upp.

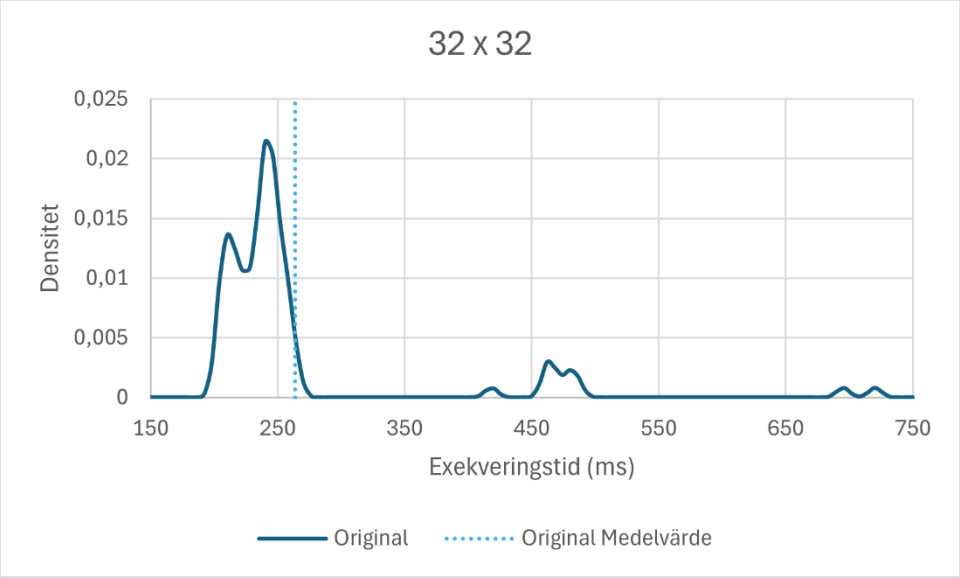
Det mått som används för att utvärdera de olika alternativen är antalet motsägelser. Då tillbakagång inte används kommer algoritmen i vissa fall att stöta på en motsägelse och för varje gång detta händer ökas ett tal. Det är detta tal som måttet avser.

4.2 Resultat

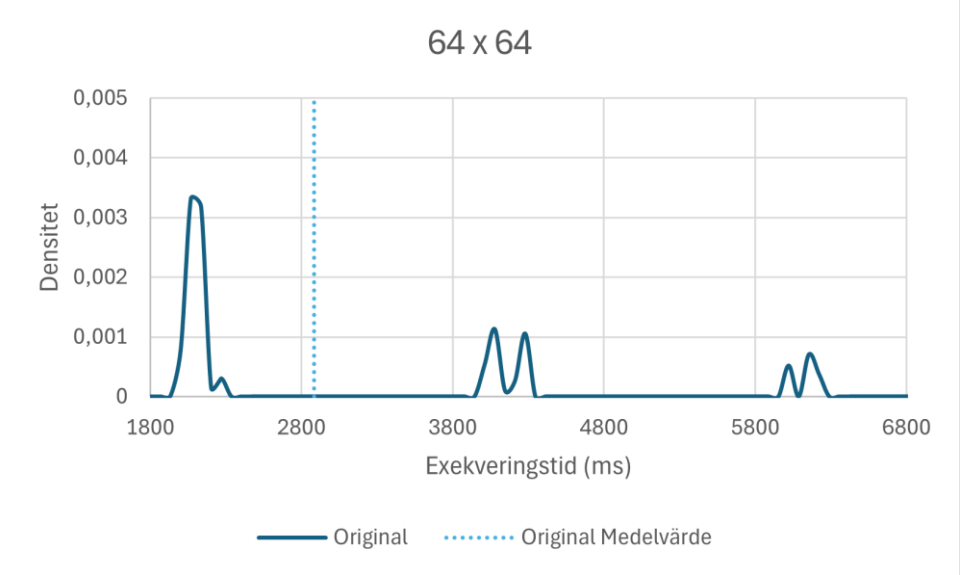
Mätningarna i alla experimenten gjordes en efter en i serie under samma exekvering. Vid varje iteration av en mätning användes iterationens index som frö till genereringen. Detta gör det möjligt att jämföra alla mätvärden eftersom de använder samma serie av frön. De diagram som sedan använts för att presentera dessa mätningar är KDE grafer och stapeldiagram. När det gäller KDE graferna så representerar x-axeln de möjliga värden som en mätning kan resultera i och y-axeln representerar densiteten för x-värdena. I detta sammanhang representerar densitet en relativ sannolikhet. Detta betyder att en dubbelt så hög densitet resulterar i att sannolikheten för att detta värde förekommer är dubbelt så stor. Den andra typen av diagram är stapeldiagram som endast används för att presentera antalet motsägelser. Här skapas en stapel för varje unikt antal som inträffat där höjden av stapeln representerar hur många gånger detta antal inträffats. Varje sätt att välja rutor på får sedan sin egen uppsättning av dessa staplar där det exakta antalet motsägelser visas ovanför varje stapel. Härnäst presenteras resultatet av alla experimenten.

4.2.1 Original

I Figur 11 och Figur 12 kan exekveringstiden för originalalgoritmen ses. För det mesta ligger exekveringstiden nära samma antal millisekunder men det finns även några små kullar med ett högre antal millisekunder. Den större storleken verkar också vara stabilare i de fall som inte kräver omstart.

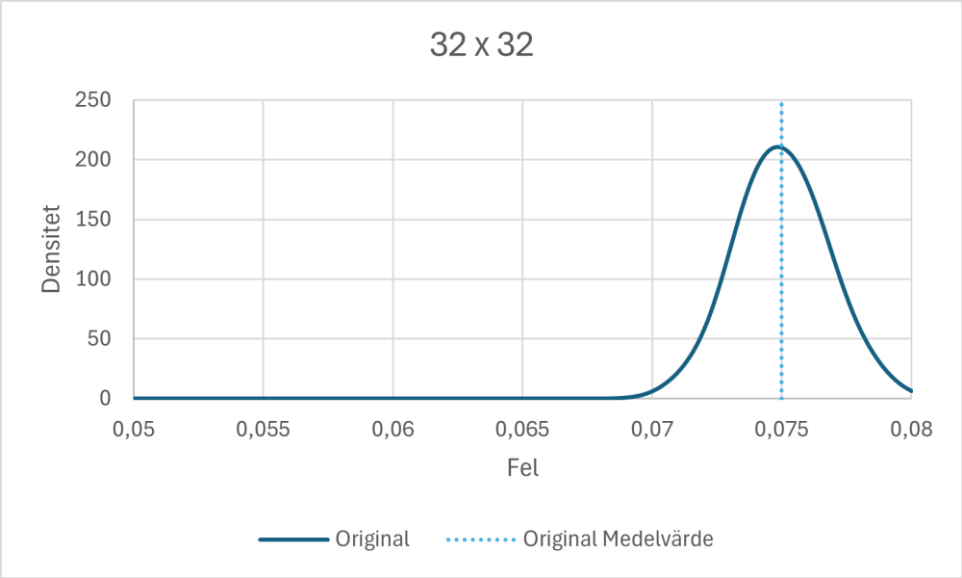


Figur 11: Exekveringstiden för världar av storleken 32 i form av en KDE graf.

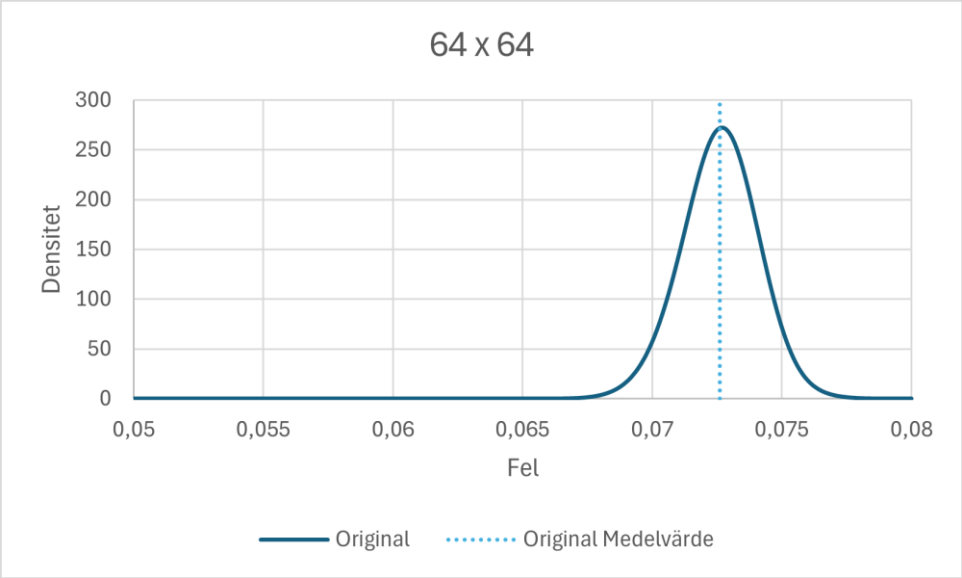


Figur 12: Exekveringstiden för världar av storleken 64 i form av en KDE graf.

I Figur 13 och Figur 14 syns blockdistributionen för originalalgoritmen. Båda storlekarna har samma form på kurvan och verkar vara stabila i sina blockdistributioner. Det som kan läggas märke till är att den större världen har en bättre blockdistribution.

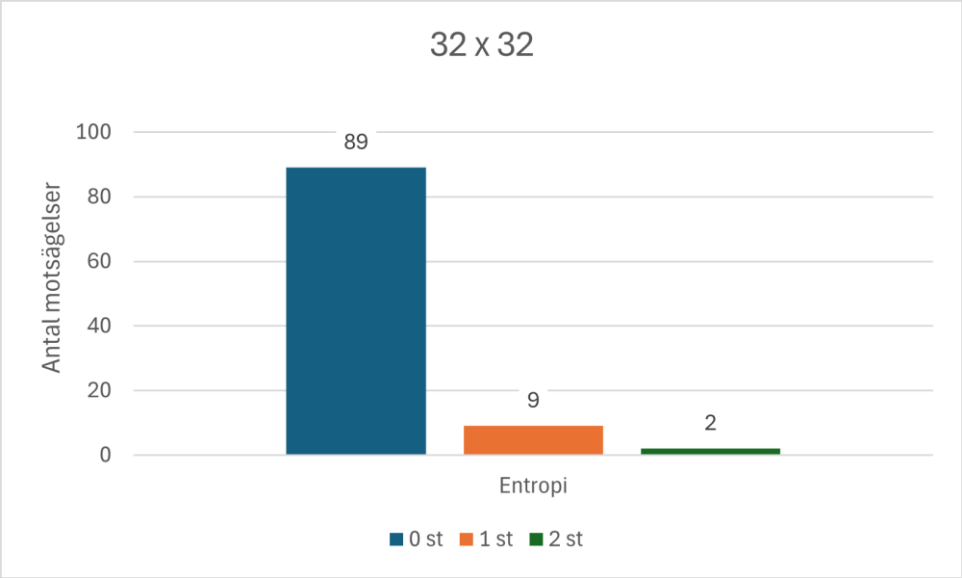


Figur 13: Blockdistributionen för världar av storleken 32 i form av en KDE graf.

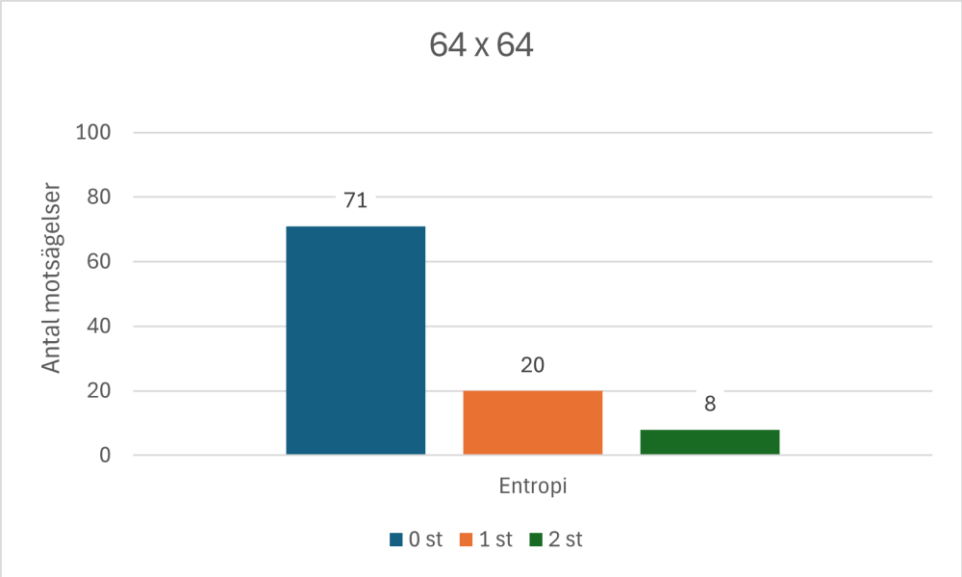


Figur 14: Blockdistributionen för världar av storleken 64 i form av en KDE graf.

Till sist kan antalet motsägelser för originalalgoritmen ses i Figur 15 och Figur 16. Det sätt som användes för att välja rutor inför observation i detta diagram är entropi. Det visar sig att en betydande andel av exekveringarna stötte på motsägelser. Diagrammen visar också att hos de större världarna är andelen betydligt större.

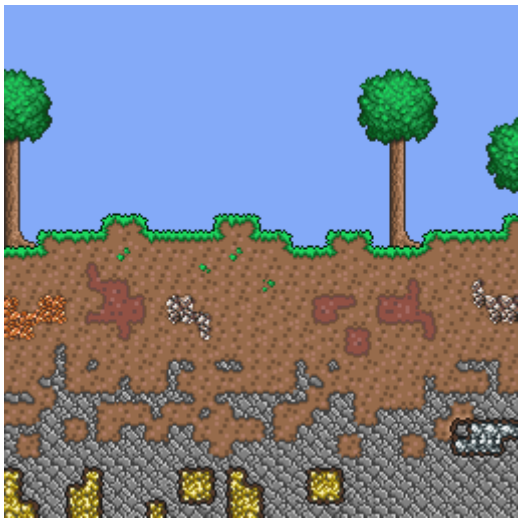


Figur 15: Antal motsägelser för världar av storleken 32 i form av ett stapeldiagram.



Figur 16: Antal motsägelser för världar av storleken 64 i form av ett stapeldiagram.

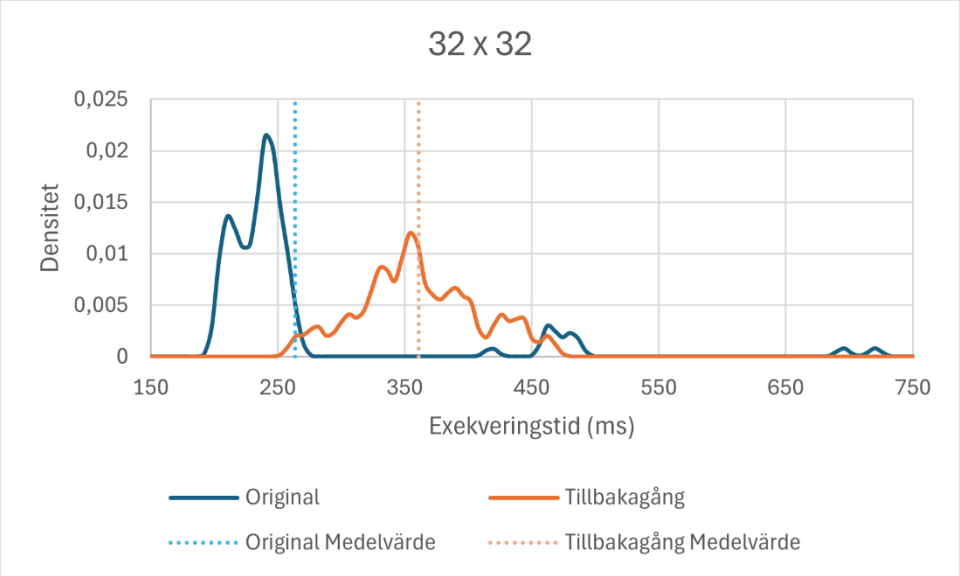
Utöver mätvärdena upptäcktes även tvivelaktiga resultat där ett exempel på ett sådant finns i Figur 17. Ett träd har påbörjats men är för nära marken för att kunna få plats.



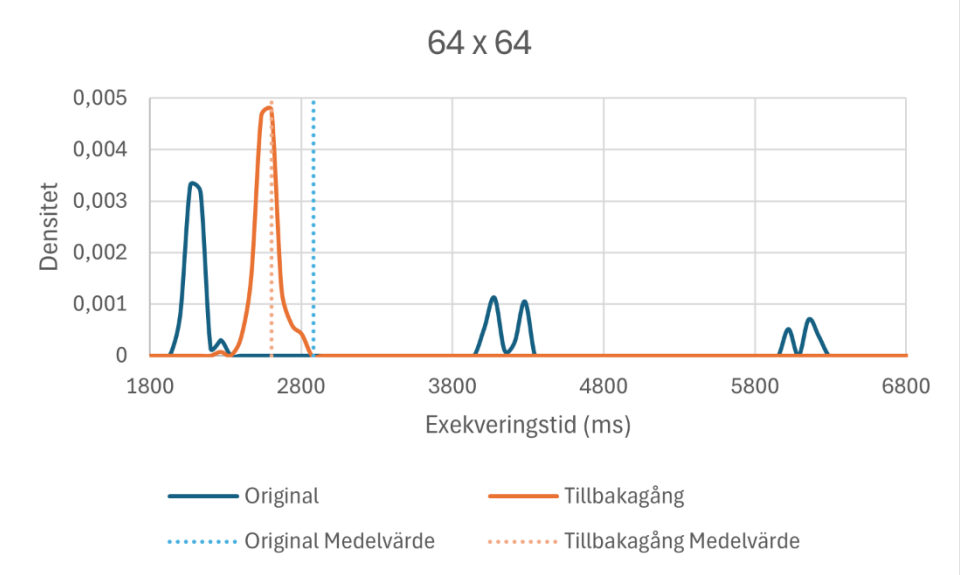
Figur 17: Ett exempel på de tvivelaktiga resultaten som upptäcktes.

4.2.2 Tillbakagång

För tillbakagång mättes exekveringstiden som kan ses i Figur 18 och Figur 19. Diagrammet visar att de kullar med högre antal millisekunder har försvunnit. Spiken som utgjorde majoriteten av fallen har nu också spridits ut över ett större intervall som har förflyttat sig åt det håll med högre exekveringstid. Samma förändring kan ses i båda storlekarna av världen.



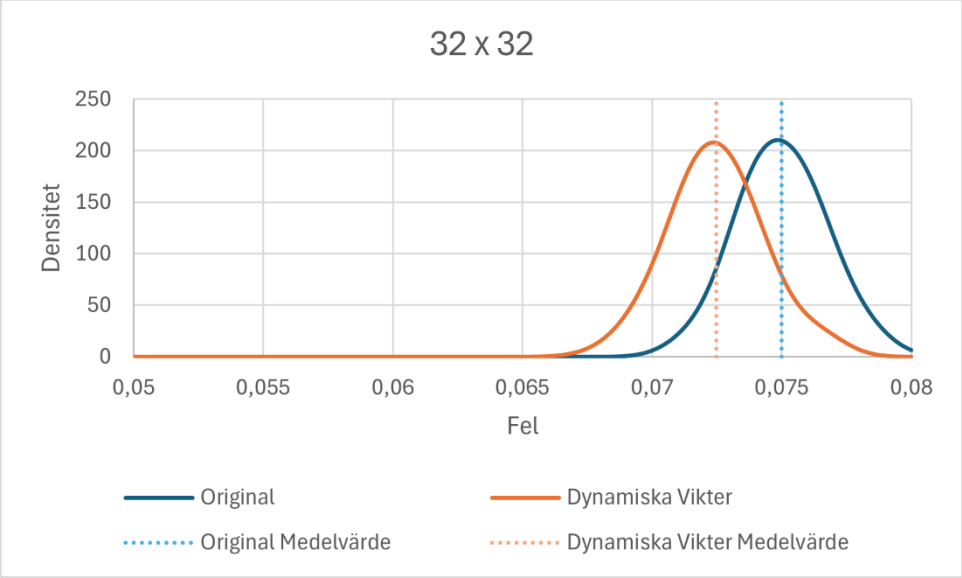
Figur 18: Exekveringstiden för världar av storleken 32 i form av en KDE graf.



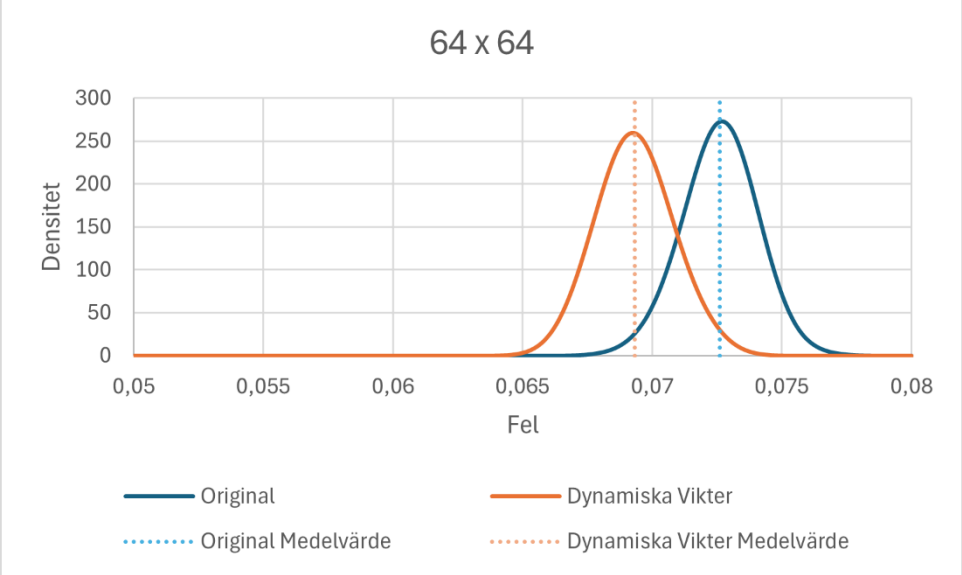
Figur 19: Exekveringstiden för världar av storleken 64 i form av en KDE graf.

4.2.3 Dynamiska vikter

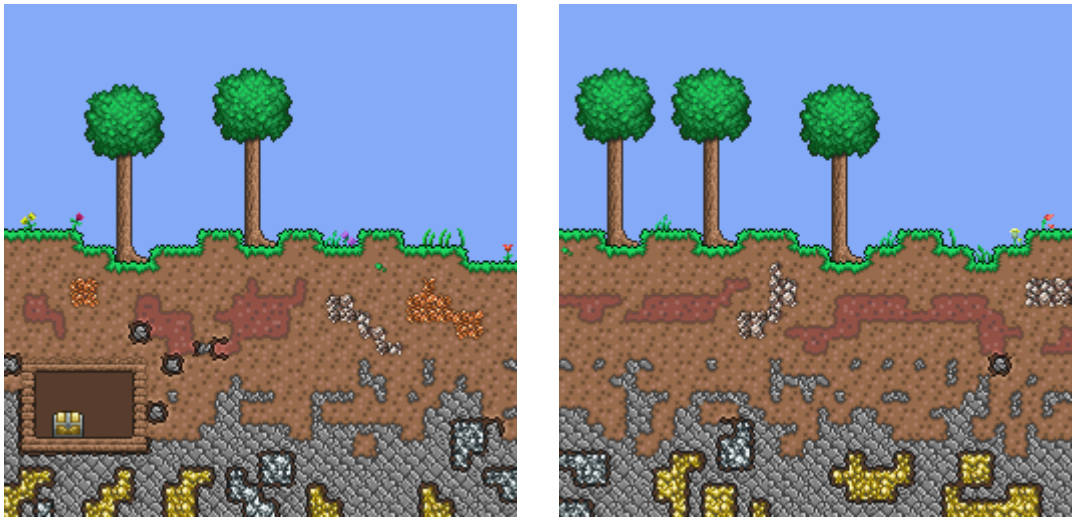
I experimentet med dynamiska vikter mättes blockdistributionen som kan ses i Figur 20 och Figur 21. Det blir tydligt att de dynamiska vikterna har minskat felet i blockdistributionen. Utöver detta verkar det inte finnas några större skillnader. Exempel på resultat finns i Figur 22. Inga tydliga skillnader kan ses mellan storlekarna.



Figur 20: Blockdistributionen för världar av storleken 32 i form av en KDE graf.



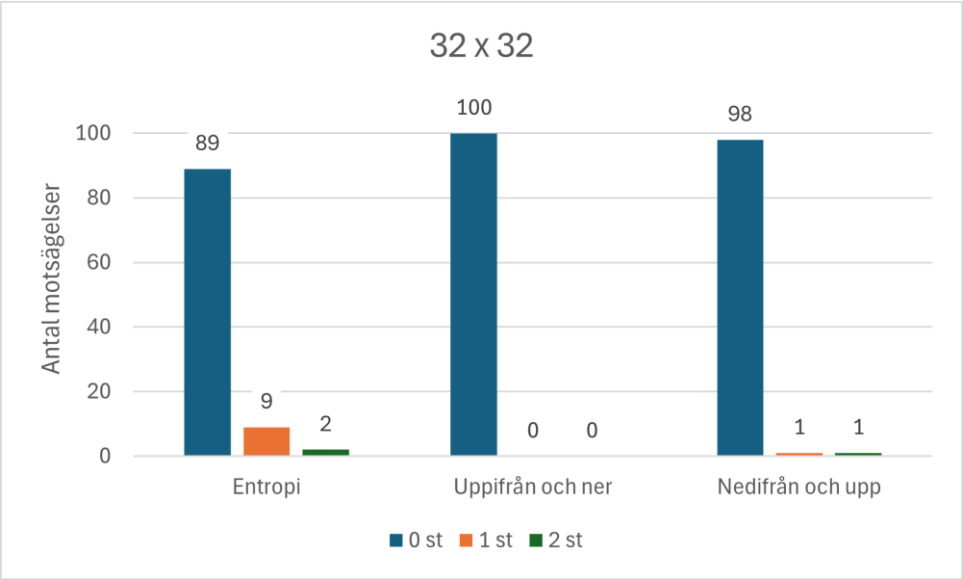
Figur 21: Blockdistributionen för världar av storleken 64 i form av en KDE graf.



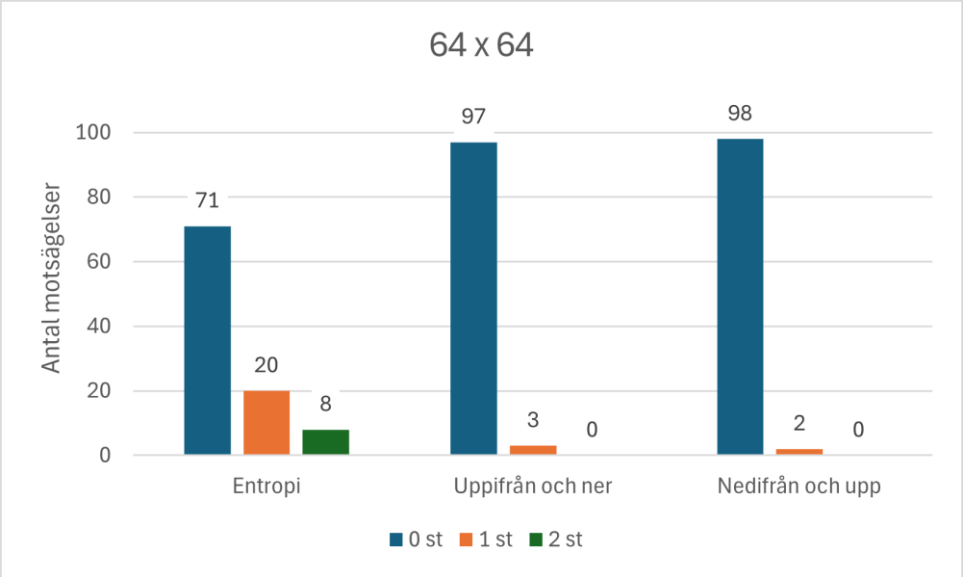
Figur 22: Exempel på resultat med (höger) och utan (vänster) dynamiska vikter.

4.2.4 Val av rutor

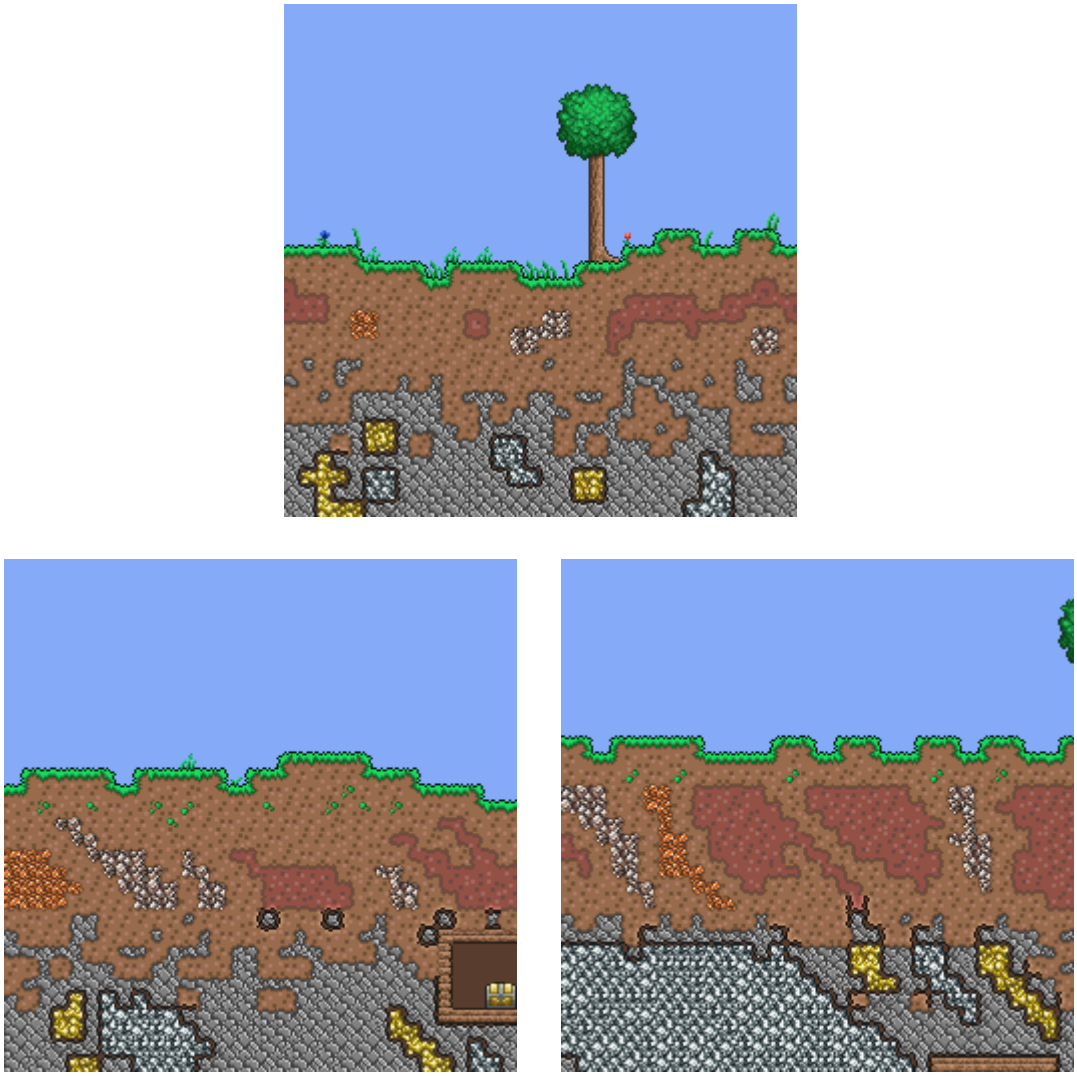
Slutligen återstår de tre olika sätten att välja rutor som skall observeras. För detta experiment mättes antalet motsägelser som kan ses i Figur 23 och Figur 24. Det blir tydligt att de sätt som väljer rutorna i ordning har betydligt färre antal motsägelser. Detta gäller för båda storlekarna. I Figur 25 kan exempel på resultatet hos de tre sätten ses. I det sätt som börjar uppifrån verkar det bildas diagonala klumpar medan i den som börjar nedifrån har klumparna spårat ur helt.



Figur 23: Antal motsägelser för världar av storleken 32 i form av ett stapeldiagram.



Figur 24: Antal motsägelser för världar av storleken 64 i form av ett stapeldiagram.



Figur 25: Exempel på resultat med de tre olika sätten att välja rutor. Entropi (överst), uppifrån och ner (vänster) och nedifrån och upp (höger).

4.3 Analys

Originalalgoritmen visade sig behöva anpassas för att resultera i en någorlunda representativ värld. Detta beror på att algoritmen endast har väldigt lokal kontroll. Det blir då svårt att skapa lagren och andra strukturer som mineral klumpar. Problemet bemöts även i forskningen där olika strategier för mer kontroll presenteras. I en artikel undveks problemet genom att med algoritmen sätta ihop vertikala segment av världen (Nie et al. 2024). Eventuella lager av världen finns då med i dessa vertikala segment. En annan metod använder distansregler som gör det möjlig att specificera hur nära blocktyper får vara varandra (Cheng, Han & Fei 2020). Andra strategier för att lösa problemet som forskats på är att beräkna om vikterna samt att ta bort valmöjligheter i visst område (Sandhu, Chen & McCoy 2019). De lösningar som användes i denna undersökning blev att distribuera vikterna vertikalt samt tekniken att ha en grupp blocktyper som representerar samma block i resultatet. Lösningarna ökade representativiteten markant men innebär också ökad komplexitet. Eftersom det redan krävs en del arbete med att sätta ihop blocktyperna, reglerna och vikterna kan det hela bli tidskrävande. De traditionella brusen är mycket bättre på att skapa större organiska strukturer men WFC utmärker sig när det gäller detaljerna och mer strukturerade former.

Exekveringstiden för originalalgoritmen visade sig vara för det mesta inom samma intervall men det fanns även vissa kullar med högre antal millisekunder. Efter att ha sorterat den data som samlats in och jämfört exekveringstiden med antal motsägelser kan slutsatsen dras att dessa kullar representerar exekveringar som stötte på motsägelser och behövde starta om. Detta låter rimligt då en omstart betyder bortkastad tid som fortfarande tas med i mätningen. Det verkar heller inte finnas någon större skillnad mellan de två storlekarna bortsett ifrån den förlängda exekveringstiden. Den enda skillnaden skulle kunna vara att den större storleken verkar stabilare. Det kan dock vara svårt att bedöma om detta verkligen är fallet eftersom graferna inte avser samma intervall.

När det gäller blockdistributionen blev felet mindre för den större världen. Detta kan förklaras med att ju större världen är desto större blir antalet block som skall observeras. Det blir då samma situation som när slumpmässiga värden väljs ifrån en viss distribution. Ju fler värden som väljs desto bättre representerar samlingen av värden den distribution som värdena hämtas ifrån. Antalet motsägelser visade sig också bero på storleken. Vid större världar blev motsägelserna fler. Detta skulle kunna förklaras med att det blir fler tillfällen att måla in sig i ett hörn ju större världen är.

Det upptäcktes även tvivelaktiga resultat där träd påbörjades men där de var för nära marken för att få plats. Detta beror på att algoritmen inte tar hänsyn till reglerna utanför arbetsytan. Skulle genereringen fortsätta utåt ifrån den redan färdiga världen skulle en motsägelse inträffa. Kanske skulle det kunna gå att upptäcka sådana fel och utlösa en motsägelse.

I det första experimentet undersöktes tillbakagång. Resultatet av denna variant blev att kullarna i exekveringstiden försvann. Detta var förväntat då kullarna representerar fall som stötte på motsägelser. Då varianten inte startar om utan går tillbaka ett steg i taget tills det går att fortsätta sparas tid. Resterande fall fick i stället motsatt effekt i form av en tidsökning. Detta kan förklaras med att algoritmen inte kan förutse om en motsägelse kommer ske och behöver därför spara informationen som används till tillbakagången vid alla exekveringar. Detta styrks också av den forskning som tillbakagången baseras på (Christie et al. 2024). Det kan bli svårt att helt undvika en tidsökning.

I nästa experiment testades dynamiska vikter. De mätningar som gjordes visade att för båda storlekarna så blev felet mindre. Om förbättringen är märkbar är svårt att säga. I de exempel som visades ses ingen markant skillnad. Detta kan bero på att slumpgenereringen blir annorlunda då vikterna ändras. Det finns dock ett par uppenbara skillnader såsom att fler träd har skapats och att det underjordiska rummet har försvunnit. Detta skulle kunna vara en indikation på att träden var underrepresenterade och det underjordiska rummet var överrepresenterat.

Slutligen återstår experimentet som räknade antalet motsägelser. I resultatet blir det tydligt att antalet blir betydligt färre i båda sätten som väljer rutorna rad för rad. Detta kan förklaras med att det blir färre hörn eller konkava delar som sedan resulterar i motsägelser. När rutorna väljs rad för rad blir dessa situationer minimala. Dock blir det en stor skillnad i resultatet för de olika sätten. Det resultat med bäst kvalitet är entropi. För de övriga bildas diagonala artefakter eller så spårar genereringen ut helt. Detta beror på att entropin väljer rutor lite här och där vilket skapar oregelbundna strukturer som måste existera tillsammans. I de övriga sätten väljs rutorna i ordning och resultatet tvingas inte till att skapa samma oregelbundna strukturer. Delar av dessa slutsatser stämmer också överens med en jämförelse som skaparen av model synthesis gjorde där WFC jämfördes med model synthesis (Merrell 2021). I jämförelsen beskrivs det hur entropi resulterar i fler motsägelser. Slutsatsen i jämförelsen drogs ifrån den data som samlades in. Det som dock inte stämmer överens med jämförelsen är att kvaliteten skulle vara den samma oavsett på vilket sätt som rutorna väljs. I denna undersökning blev det stor skillnad i kvaliteten. Det skulle kunna vara så att skillnaden i kvaliteten beror på den blocksamling som används.

5 Sammanfattning och diskussion

Härnäst kommer en sammanfattning av undersökningen, följt av en diskussion om projektet i ett större sammanhang. Därefter beskrivs samhälleliga och etiska aspekter, som slutligen leder till tankar kring framtida arbete.

5.1 Sammanfattning

I denna undersökning sätts wave function collapse algoritmen i sammanhanget av världsgenerering i sandbox survival spel. Wave function collapse eller WFC är en algoritm som procedurellt genererar bilder utifrån mönster och regler. Algoritmen har fått stor uppmärksamhet och har använts till fler saker än att generera bilder. Det har också skett en del forskning kring att förbättra olika aspekter av algoritmen. Detta har lett till flera olika varianter av algoritmen. Denna undersökning fokuserar på tre av dessa varianter som fokuserar på olika delar av algoritmen. För att kunna utvärdera varianterna jämförs de med originalvarianten. Vid jämförelsen används olika mätvärden som fångar upp eventuella skillnader. Den frågeställning som undersöks är: Hur påverkar olika varianter av wave function collapse världsgenereringen i sandbox survival spel?

För att besvara frågeställningen utfördes tre experiment där varje experiment jämför en av varianterna. Vid implementeringen av originalvarianten upptäcktes dock utmaningar som behövde lösas innan experimenten kunde påbörjas. Det visade sig att algoritmen behövde anpassas för världen som skulle genereras. Experimenten utfördes sedan genom att samla in kvantitativa data med hjälp av de olika mätvärdena. Denna data presenterades sedan med hjälp av olika diagram. En analys av resultatet visade att det fanns vinster i alla tre varianter. I vissa fall var dock inte vinsterna helt gratis och andra aspekter av algoritmen försämrades.

5.2 Diskussion

Då tidskomplexiteten för WFC är kvadratisk blir det svårt att generera de stora världar som finns i sammanhanget. Detta är anledningen till att inte större världar användes i denna undersökning. Det tar helt enkelt för lång tid att genera stora världar. För att bemöta detta problem behövs nya strategier som inte används i denna undersökning. I forskningen kring algoritmen hittas flera sådana strategier. Den främsta av dem är att dela upp världen i mindre bitar och generera dessa var för sig. Detta möjliggör parallell generering av bitarna men resulterar också i att inte hela världen behöver kastas bort vid en motsägelse. I forskningen hittas två projekt som presenterar metoder för att göra just detta (Christie et al. 2024; Nie et al. 2024). Exekveringstiden skulle även kunna förbättras genom att optimera algoritmens implementation. I en undersökning implementeras algoritmen med hjälp av bitoperationer (Punia, Priyanka & Sharma 2023). I det stora hela kan det nog dock bli svårt att undkomma den kvadratiska tidskomplexiteten.

Ett annat stort problem blev bristen på mer global kontroll. Flera anpassningar behövde göras för att få en relativt representativ värld. Det var svårt att skapa de lager som världen består av samt att skapa större strukturer i dessa. I analysen beskrevs forskandet kring olika sätt att ge utvecklaren mer kontroll. Problemet skulle också kunna undvikas genom att använda algoritmen på en högre nivå. I stället för att generera världen på blocknivå så skulle WFC kunna användas för att sätta ihop större moduler. Världen skulle då bli mindre intressant då varje modul finns med flera gånger. Ett exempel på ett spel som sätter ihop större moduler är The Lord of the Rings: Return to Moria (2023). En del av kartan i spelet kan ses i Figur 26. Spelet använder inte en rutbaserad värld och det är inte känt om spelet använder WFC men konceptet blir det samma. Andra varianter skulle kunna vara en hybrid lösning där WFC delvis används för världsgenereringen. Ett exempel på detta använder WFC för att generera en övergripande regionskarta som sedan kombineras med traditionellt brus (Büyükşar, Yıldız & Demirci 2024). Sättet använder sig då utav de bästa delarna av de båda algoritmerna. Regionskartan gör det uniforma bruset mer intressant medan bruset inte kräver lika mycket förberedelse som WFC.

Figur 26: En skärmdump av kartan i spelet The Lord of the Rings: Return to Moria (2023) som visar hur spelet är uppbyggt av stora moduler.

Härnäst beskrivs hur användbar denna undersökning är i andra sammanhang. Det skulle till exempel kunna handla om andra spel eller spelgenrer. Likt den undersökning som inspirerade denna undersökning (Wang & Chang 2024) så blir slutsatsen att det kan bli svårt att föra över resultat i andra sammanhang. Detta eftersom andra spel och spelgenrer har sina utmaningar och blocksamlingen som används kan variera stort. Resultaten är dock inte helt irrelevanta för andra sammanhang då liknande problem och lösningar givetvis kan dyka upp även utanför detta sammanhang. Denna undersökning kan därför vara en utgångspunkt för att göra liknande undersökningar i andra sammanhang. Det är dock inte säkert att alla delar av undersökningen blir relevanta men de kan ändå inspirera andra projekt. Detta projekt har ju på ett liknande sätt inspirerats av en annan undersökning som undersökte ett annat sammanhang. Resultaten i denna undersökning styrks också delvis av annan forskning vilket ökar undersökningens trovärdighet. Detta minskar riskerna med att använda dessa resultat i andra projekt.

5.3 Samhälleliga och etiska aspekter

Den samhälleliga nytta som undersökningen skulle kunna resultera i är bidragandet till forskning samt sparande av både tid och pengar för andra utvecklare. Då algoritmen sätts i ett nytt sammanhang kan denna undersökning leda till att mer forskning bedrivs. Procedurell generering används även utanför nöjesindustrin och forskning inom detta område kan därför resultera i samhällelig nytta. Undersökningen kan också vara till nytta för utvecklare som kan se hur algoritmen skulle kunna fungera i sammanhanget utan att de själva behöver spendera tid eller pengar på detta. Denna typ av forskning kan då leda till att fler utvecklare vågar testa nya saker vilket leder till en mer varierad spelindustri.

5.4 Framtida arbete

För att fortsätta arbetet som presenterats i denna undersökning skulle fler varianter kunna undersökas med potentiella nya mätvärden. Speciellt vore varianter där världen delas upp i mindre bitar värt att undersökas. Det skulle då vara möjligt att skapa större världar som blir mer representativt för sammanhanget. Det kan också vara intressant att undersöka hybridvarianter som mixar traditionellt brus med WFC. Mer tid skulle också kunna läggas på att utöka den begränsade mängd blocktyper som användes. Detta kan ge fler insikter i hur algoritmen behöver anpassas samt hur skalbar algoritmen är i sammanhanget. I denna undersökning undersöktes också bara den enkla rutbaserade varianten. Den kan vara värt att undersöka om den överlappande varianten är lämplig i sammanhanget och i sådana fall vad skillnaden blir.

Om området utökas till att omfatta hela forskningsområdet kring algoritmen så finns det många olika aspekter som kan forskas vidare på. Det är uppenbart att en av algoritmens brister är den begränsade globala kontrollen. Viss forskning har presenterat förslag på olika lösningar, men området är fortfarande öppet för nya framsteg. Ett stort område som behöver forskas på mer är hur hybrid lösningar kan skapas där WFC samarbetar med till exempel traditionellt brus. Det hade kanske gått att generera till exempel byggnader med WFC och resten med brus. Ett annat intressant koncept skulle kunna vara om WFC användes som en efterbehandlingseffekt. Världen skulle då genereras i två steg. Först genereras terrängen med hjälp av brus. Sedan läggs byggnader till med WFC. Algoritmen skulle då behöva ta hänsyn till vilka block som redan är utplacerade i ett sorts bakgrundslager. Det skulle då vara möjligt att dra nytta av WFC utan att behöva generera hela världen med algoritmen. Bakgrundslagret skapar också globala restriktioner som algoritmen följer vilket leder till att den begränsade kontrollen inte blir ett lika stort problem. Som avslutande ord så finns det stor potential med WFC men det finns också en hel del begränsningar som behöver bemötas vilket resulterar i ett forskningsområde med stora möjligheter för fortsatt forskning.

Referenser

Andersson, Ö. (2012). *Experiment! : planning, implementing, and interpreting*. Wiley.

Alaka, S. and Bidarra, R. (2023) 'Hierarchical Semantic Wave Function Collapse', *ACM International Conference Proceeding Series* [Preprint]. doi:10.1145/3582437.3587209.

Bad North (2018). Plausible Concept. Tillgänglig på Internet: https://www.badnorth.com/.

Bailly, R. and Levieux, G. (2023) 'Genetic-WFC: Extending Wave Function Collapse With Genetic Search', *IEEE Transactions on Games, Games, IEEE Transactions on, IEEE Trans. Games*, 15(1), pp. 36–45. doi:10.1109/TG.2022.3192930.

Büyükşar, O., Yıldız, D. and Demirci, S. (2024) 'Enhancing wave function collapse algorithm for procedural map generation problem', *Nigde Omer Halisdemir University Journal of Engineering Sciences / Niğde Ömer Halisdemir Üniversitesi Mühendislik Bilimleri Dergisi*, 13(3), pp. 806–814. doi:10.28948/ngumuh.1361413.

Caves of Qud (2024). Freehold Games. Tillgänglig på Internet: https://www.cavesofqud.com/.

Cheng, D., Han, H. and Fei, G. (2020) *Automatic Generation of Game Levels Based on Controllable Wave Function Collapse Algorithm*. Cham: Springer International Publishing. doi:10.1007/978-3-030-65736-9_3.

Christie, R. *et al.* (2024) 'Procedurally Generating Large Synthetic Worlds: Chunked Hierarchical Wave Function Collapse', *2024 39th International Conference on Image and Vision Computing New Zealand (IVCNZ), Image and Vision Computing New Zealand (IVCNZ), 2024 39th International Conference on*, pp. 1–6. doi:10.1109/IVCNZ64857.2024.10794452.

Gumin, M. (2016). *Wave Function Collapse Algorithm*. https://github.com/mxgmn/WaveFunctionCollapse [2025-02-03]

Karth, I. and Smith, A.M. (2017) 'Wave Function Collapse is constraint solving in the wild', *ACM International Conference Proceeding Series*, Part F130151. doi:10.1145/3102071.3110566.

Kim, H. *et al.* (2019) 'Automatic Generation of Game Content using a Graph-based Wave Function Collapse Algorithm', *2019 IEEE Conference on Games (CoG), Games (CoG), 2019 IEEE Conference on*, pp. 1–4. doi:10.1109/CIG.2019.8848019.

Kim, H., Seo, B. and Kang, S. (2024) 'Puzzle-Level Generation with Simple-tiled and Graph-based Wave Function Collapse Algorithms', *IEEE Transactions on Games*, pp. 1–11. doi:10.1109/TG.2024.3368017.

Merrell, P. (2021) 'Comparing Model Synthesis and Wave Function Collapse' Tillgänglig på Internet: https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf [Hämtad Mars 18, 2025].

Merrell, P. and Manocha, D. (2011) 'Model Synthesis: A General Procedural Modeling Algorithm', *IEEE Transactions on Visualization and Computer Graphics, Visualization and Computer Graphics, IEEE Transactions on, IEEE Trans. Visual. Comput. Graphics*, 17(6), pp. 715–728. doi:10.1109/TVCG.2010.112.

Minecraft (2011). Mojang Studios. Tillgänglig på Internet: https://www.minecraft.net.

Nie, Y. *et al.* (2024) 'Nested Wave Function Collapse Enables Large-Scale Content Generation', *IEEE Transactions on Games, Games, IEEE Transactions on, IEEE Trans. Games*, 16(4), pp. 892–902. doi:10.1109/TG.2024.3377637.

O'Leary, M. (2017). *Oisín: Wave Function Collapse for poetry*. https://github.com/mewo2/oisin [2025-02-07]

Punia, Y., Priyanka and Sharma, T.. (2023) 'Enhancing Wave Function Collapse Algorithm using Bitwise Operations', *2023 International Conference on Computer, Electronics & Electrical Engineering & their Applications (IC2E3), Computer, Electronics & Electrical Engineering & their Applications (IC2E3), 2023 International Conference on*, pp. 1–5. doi:10.1109/IC2E357697.2023.10262419.

Sandhu, A., Chen, Z. and McCoy, J. (2019) 'Enhancing wave function collapse with design-level constraints', *ACM International Conference Proceeding Series* [Preprint]. doi:10.1145/3337722.3337752.

Terraria (2011). Re-Logic. Tillgänglig på Internet: https://terraria.org.

The Lord of the Rings: Return to Moria (2023). Free Range Games. Tillgänglig på Internet: https://www.returntomoria.com.

Townscaper (2021). Oskar Stålberg. Tillgänglig på Internet: https://www.townscapergame.com/.

Wang, B.-Y. and Chang, C.-L. (2024) 'Wave Function Collapse Algorithm in Roguelite Game Level Design: An Analysis of Strategies and Challenges', *2024 10th International Conference on Applied System Innovation (ICASI), Applied System Innovation (ICASI), 2024 10th International Conference on*, pp. 359–361. doi:10.1109/ICASI60819.2024.10547926.