TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Markus Männil

Lainefunktsiooni kokkulangemise algoritmi laiendus mängumootorile Godot

Bakalaureusetöö (9 EAP)

Juhendaja: Jaanus Jaggo

Tartu 2024

Lainefunktsiooni kokkulangemise algoritmi laiendus mängumootorile Godot

Lühikokkuvõte:

Laine funktsiooni kokkulangemise algoritm on protseduurilise sisu loome algoritm, millega on võimalik luua pilte, maailmu ja muud sisu mängudele. Töös luuakse Godot mängumootorile laiendus selle algoritmi kasutamiseks kolmemõõtmeliste maailmade genereerimiseks. Seda algoritmi optimeeritakse ja viiakse läbi jõudlustestimine, et hinnata selle sobivust mängu kestel sisu loomiseks.

Võtmesõnad:

lainefunktsiooni kokkulangemise algoritm, Godot mängumootor, protseduuriline sisu loomine, optimeerimine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaat-juhtimisteooria)

Wave function collapse algorithm plugin for Godot game engine

Abstract:

The Wave Function Collapse algorithm is a procedural content generation algorithm used to create images, worlds, and other content for games. In this work, a plugin for the Godot game engine is developed to utilize this algorithm for generating three-dimensional worlds. The algorithm is optimized and performance-tested to evaluate its suitability for generating content during gameplay.

Keywords:

wave function collapse, Godot game engine, procedural content creation, optimisation

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus 5

2. Lainefunktsiooni kokkulangemise algoritm 7

2.1 Tekstuuri süntees 7

2.2 Tükkide põhine laine funktsiooni kokkulangemise algoritm 8

2.3 Kolmemõõtmeline laine funktsiooni kokkulangemise algoritm 9

2.4 Mudeli süntees 12

3. Godot mootori laiendamine 14

Laine funktsiooni kokkulangemise algoritm 2D laiendus 15

4. Implementasioon 16

4.1 Algoritm 19

4.2 Andmestruktuurid 21

4.2.1 Külgnevate objektide massiivi täitmine 21

4.2.2 Võimaluste vähendamine 22

4.3 Algoritmi optimeerimine 23

5. Testimine 26

5.1 Testimine võrel suurusega 10 x 5 x 10 26

5.2 Testimine võrel suurusega 20 x 10 x 20 27

5.3 Testimine võrel suurusega 40 x 10 x 40 28

5.4 Testimise tulemused 29

Kokkuvõte 31

Viidatud kirjandus 32

Lisad 33

I. Lähtekood 33

II. Kasutusjuhend 34

III. Litsents 38

Sissejuhatus

Arvutimängude loomine on suur töö, sest on vaja palju erinevat sisu, näiteks maailmu, tegelasi ja muid mänguobjekte, mis muudavad mängu põnevaks. Uuringud näitavad, et populaarsed mängud muutuvad üha suuremaks ja vajavad rohkem inimesi, kes neid arendaks. Näiteks üheksakümnendatel oli populaarsete mängude, nagu "Commander Keen" ja "Doom" meeskonnas 5 või 6 arendajat, aga juba järgmisel kümnendil ulatusid meeskonnad sadade inimesteni. Suuremates mängudes nagu "World of Warcraft" on üle 30 000 eseme ja enam kui 1400 unikaalset maailma asukohta. [1]

Mängu suuruste kasvades on kasvanud ka mängijate ootused. Selleks, et väike meeskond suudaks turul konkureerida, tuleb leida alternatiivseid võimalusi, et mängule sisu luua. Üheks võimaluseks on kasutada protseduurilist genereerimist, ehk luua mängule sisu algoritmide abil. Kuigi ka sellise süsteemi loomine nõuab palju tööd, siis läbimõeldult lahendus võib säästa palju aega ja lisada mängule variatsiooni.

Näiteks mängus "Minecraft" luuakse plokkidest koosnev maailm, mis on iga kord unikaalne. See on ka üks põhjus, miks see mäng on müünud üle kolme saja miljoni koopia ja on ühtlasi kõige enam müüdud mäng maailmas.[1]

Mängus "No Man's Sky" genereeritakse terve universum protsetuuriliselt, ning kokku on seal 2^64 unikaalset planeeti. Neid kõiki ei jõuaks keegi kunagi avastada, kuid ühtlasi võib iga mängija leida oma planeedi, millel tema esimesena käinud on.[2]

Lainefunktsiooni kokkulangemise algoritm (wave function collapse algorithm) edaspidi WFC on piirangute põhine protseduurilise loome algoritm, mis loob etteantud tükkide ja nende vaheliste piirangute abil maailmu, pilte ja mängu tasemeid. Tema eeliseks teiste protseduuriliste loome algoritmide ees on modulaarsus ja laiendatavus. See tähendab, et sinna saab lihtsalt uusi tükke juurde lisada, ilma et peaks algoritmi ennast muutma. Samuti on selle reeglite defineerimine intuitiivne. [2]

---

[1] https://www.windowscentral.com/gaming/minecraft/minecraft-crosses-300-million-copies-sold-as-it-prepares-to-celebrate-its-15th-anniversary
[2] https://www.nomanssky.com/

Käesolevas töö eesmärgiks on luua Godot mängumootorile lainefunktsiooni kokkulangemise algoritmi laiendus, millega saab genereerida kolmemõõtmelisi maailmu. Samuti on eesmärgiks luua sellele kasutajaliides, mille abil saab algoritmile reeglid ja objekte ette anda.

Godot mängumootor valiti just selle pärast, et tegemist on vabavaralise ja üha populaarsust koguva mootoriga. Näiteks on sellega loodud mängud "Dome Keeper", "Blastronaut" ja "Brotato". Samuti on teistel populaarsetel mängumootoritel juba olemas laiendused 3D-s WFC kasutamiseks näiteks Unity mootorile on tööriist Tessera[3] ja Unreal mootorile Procedural Environment Generator (WFC) Basic[4].

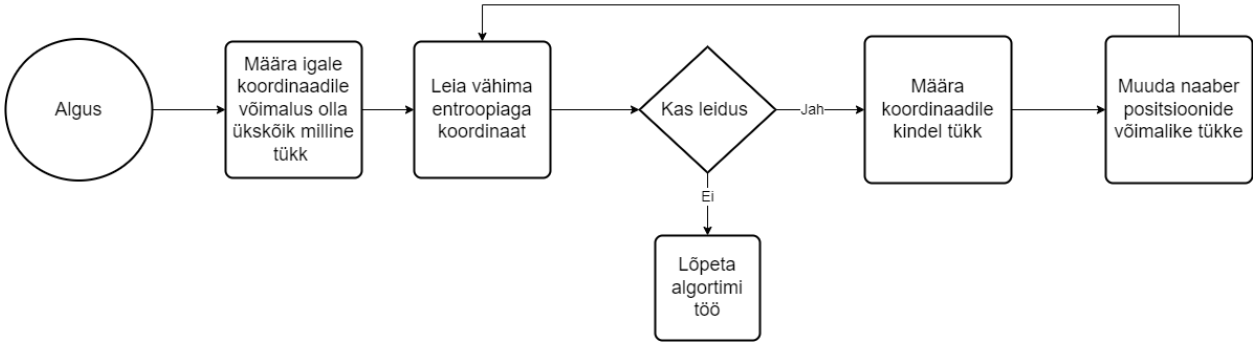
Töö teises peatükis käsitletakse WFC algoritmi tööpõhimõtet ja vaadeldakse erinevaid viise selle implementeerimiseks. Kolmandas peatükis kirjeldatakse Godot mootorit ja sellele laiendamist. Neljandas peatükis vaadeldakse lähemalt algoritmi implementeerimise detaile ja optimeeritakse seda. Viiendas peatükis testitakse algoritmi jõudlust erinevatel andmestikel ja võrestike suurustega. Kuuendas peatükis tehakse kokkuvõte ja antakse soovitusi algoritmi täiendamiseks.

[3] https://assetstore.unity.com/packages/tools/level-design/tessera-pro-161077
[4] https://www.unrealengine.com/marketplace/en-US/product/procedural-environment-generator-wfc-basic

2. Lainefunktsiooni kokkulangemise algoritm

Lainefunktsiooni kokkulangemise algoritm on piirangute põhine protseduurilise generatsiooni algoritm, mis põhineb samanimelisel kvant füüsika nähtusel. Algoritm genereerib defineeritud tükkidest ja nende vahelistest piirangutest pilte, maailmu või mängu tasemeid.

Algoritmi üldine töökäik on järgmine: Esmalt luuakse ruudustik ning määratakse selle igale koordinaadile võimalus olla ükskõik milline tükk. Seejärel valitakse madalama entroopiaga, vähima võimalike tükkidega, koordinaatide seast üks ning määratakse sellele kindel tükk ning vähendatakse naaber koordinaatidele sobivaid võimalike tükke. Jätkatakse ruudustikule tükkide määramist, kuni igale koordinaadile on määratud kindel tükk. [2]

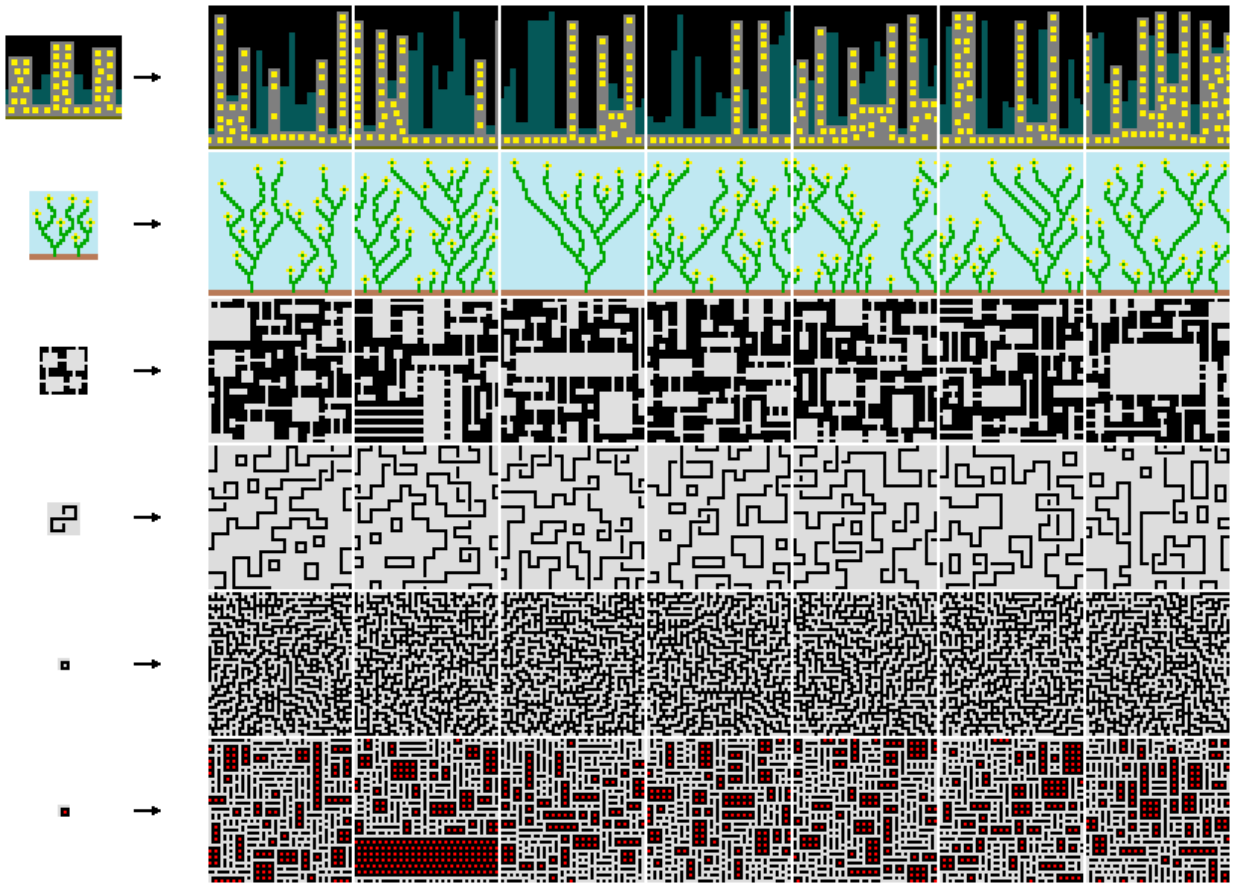


Joonis 1. Lainefunktsiooni kokkulangemise algoritmi töökäik.

2.1 Tekstuuri süntees

Maxim Guminit loetakse WFC loojaks, kes pani 2016. aastal *Githubi* üles WFC repositooriumi. Algoritm oli algselt loodud tekstuuri sünteesimiseks, mis lõi sisestatud pildile sarnase pildi. Tükkide defineerimiseks, leitakse sisestatud pildilt kõik N x N osad, ning määratakse, kuidas need saavad kõrvuti asetseda.[5]

[5] https://github.com/mxgmn/WaveFunctionCollapse



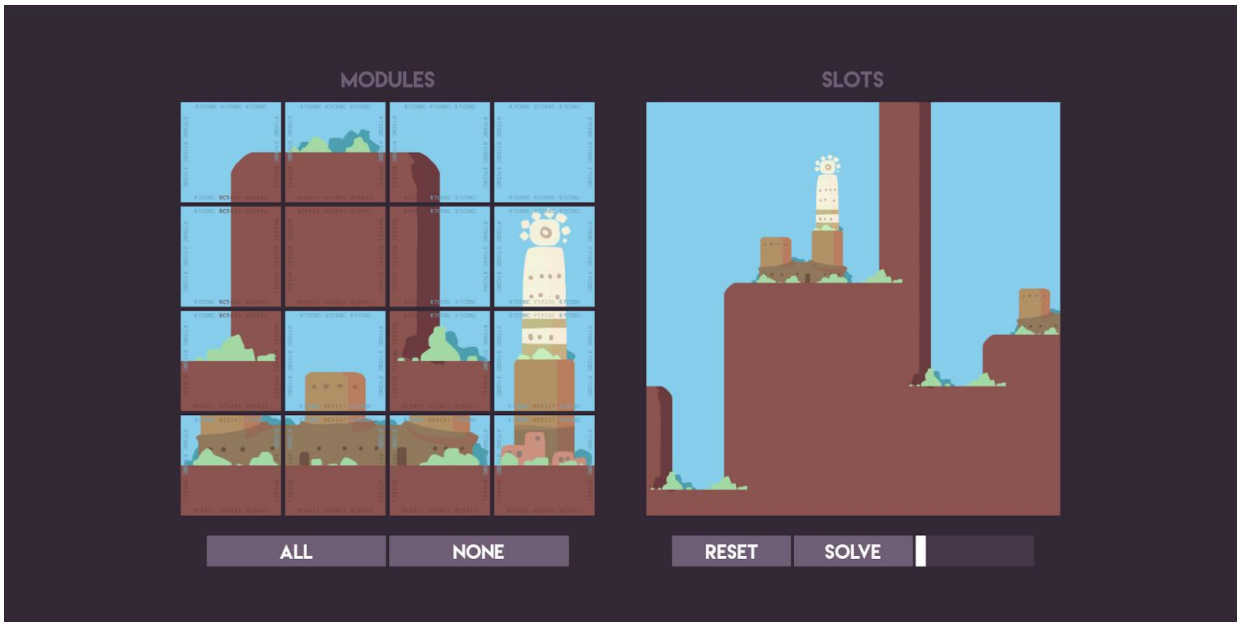
Pilt 1. Max Cumini algoritmi näited, kus vasakul on sisestatud pildid ja paremal nende põhjal loodud pildid.

2.2 Tükkide põhine laine funktsiooni kokkulangemise algoritm

Maxim Gumini tegi ka teise versiooni algoritmist, millele pildi asemel antakse ette tükid ja kuidas need võivad kõrvuti asetseda. Igale tüki küljele kõikide teiste lubatud tükkide määramine, on aegavõttev protsess. Seepärast määras ta tükkidele sümmeetria tüübid ehk ette defineeritud reeglid, mis määravad kuidas erinevate sümmeetria tüüpidega tükid saavad kõrvuti asetseda. Seega pidi ta igale küljele kõigi lubatud tükkide määramise asemel, määrama igale tükile ühe sümmeetria tüübi. Selle meetodi eeliseks on suurem kontroll algoritmi tulemuse üle, sest kasutaja saab täpsemini määrata tükkide vahelisi seoseid.[6]

[6] https://github.com/mxgmn/WaveFunctionCollapse

Oskar Stålberg tegi tükkide põhisest ka populaarse demo “Wave”[7], mis visualiseeris kuidas WFC pilti genereerib. Tükkide vahelisteks reegliteks kasutas ta serva värve, kus iga tüki igal serval on kolm värvi, mida saab ühendada samade värvidega.



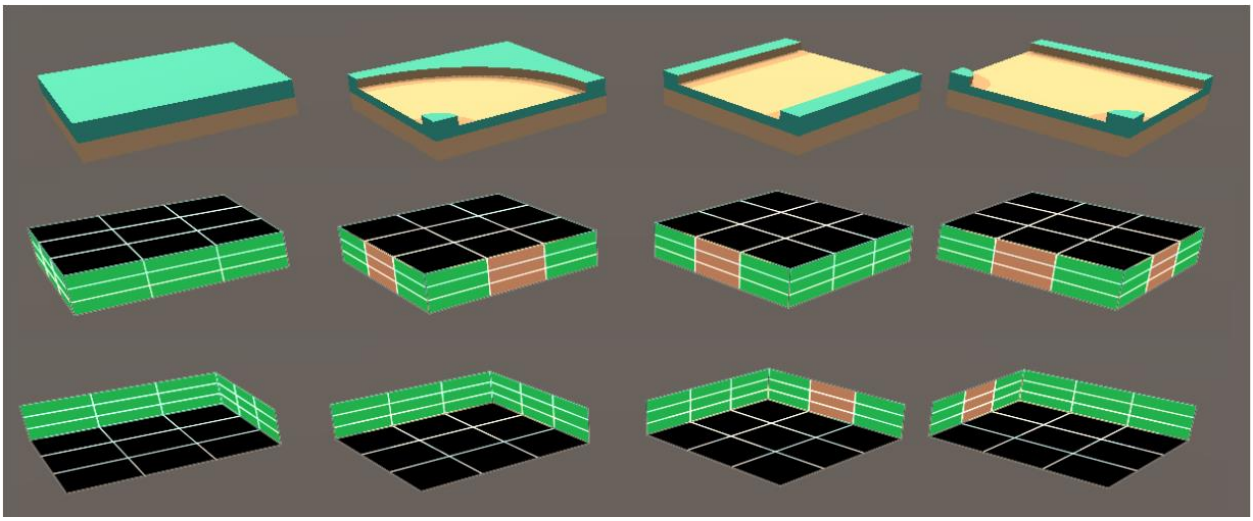
Kuvatõmmis 1. Oskar Stålbergi demo „Wave“

2.3 Kolmemõõtmeline laine funktsiooni kokkulangemise algoritm

Kui algselt rakendati WFC algoritmi vaid 2D kujutiste loomisel, siis nüüdseks on WFC-d kasutatud ka 3D maailmade loomisel.

Boris the Brave tegi tööriista “Tessera”, millega saab WFC-d kasutada Unity’s. Tema tööriista eesmärk on teha algoritmi kasutamine lihtsamaks ja kergemini kontrollitavaks. Objektidele reeglite määrimiseks jagab tööriist iga objekti külje 3x3 tükiks. Neid külje tükke saab värvida ning samamoodi värvitud külgedega objekte saab panna kõrvuti. [3]

[7] https://oskarstalberg.com/game/wave/wave.html



Pilt 2. „Tesseras“ objektide külgede värvimine [3]

Oskar Stålberg on kasutanud WFC-d enda mängudes “Bad North”[8] ja “Townscaper”[9].

“Bad North” on strateegia mäng, mis kasutab WFC, et luua saari, mida mängija peab kaitsma. Mängus peab olema võimalik iga saare punktist liikuda iga teise punkti, seega lisas ta algoritmile piirangu, et tükke valitaks nii, et oleks võimalik liikuda iga punkti vahel. [10]

[8] https://www.badnorth.com/
[9] https://store.steampowered.com/app/1291340/Townscaper/
[10] https://konsoll.org/talks/developing-the-bad-north-look/

Pilt 3. "Bad North" mängus genereeritavad saared.[11]

"Townscaper" on linna ehitamise mäng, kus mängijad saavad ruudustikule vajutades lisada ja eemaldada klotse. Klotside lisamisel või eemaldumisel muudab WFC klotsi välimust, olenevalt teda ümbritsevatest klotsidest. Mäng kasutab klassikalise ruudustiku asemel ebaregulaarset ruudustiku, mis muudab kasutaja poolt loodavad maailmad unikaalsemaks.

[11] https://twitter.com/OskSta/status/1131228226901172226

Kuvatõmmis 2. Pilt mängust „Townscaper“ [12]

2.4 Mudeli süntees

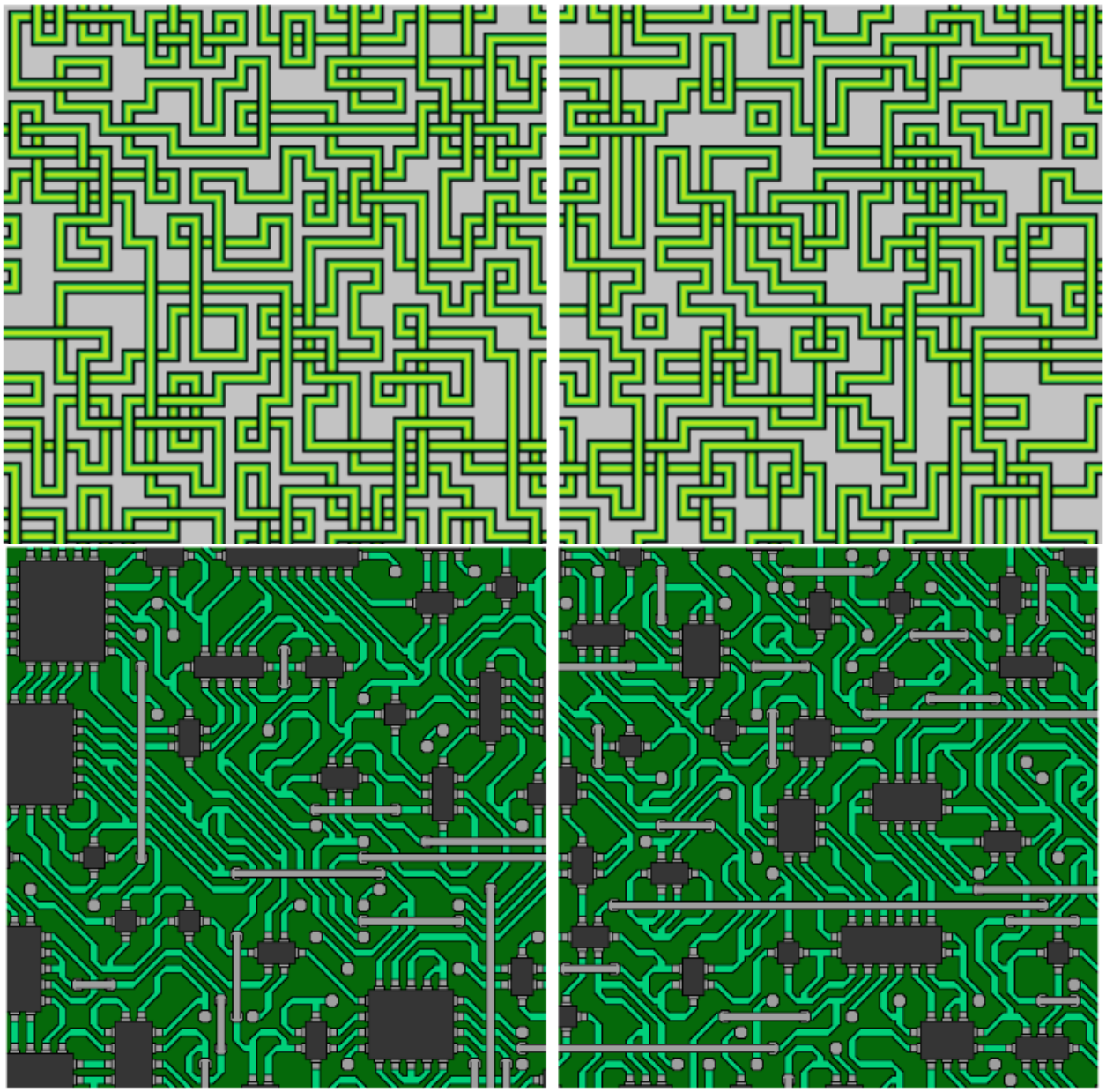
Mudeli süntees on 2007. aastal Paul Merrelli poolt loodud piirangute põhine protseduurilise genereerimise algoritm. [4]

Algoritm on väga sarnane WFC-le ning Merrel on väitnud, et nad on sama algoritmi erinevad versioonid [5]. Samas mudeli sünteesi peamine eesmärk oli luua kolmemõõtmelisi maailmu erinevalt WFC-st mida algselt kasutati tekstuuri sünteesiks. [4][5]

Suurimaks erinevuseks nende vahel on järgmise positsiooni valimine. Mudeli süntees käib positsioonid järjest läbi, kuid WFC valib positsioone väikseima entroopia järgi ehk väikseima võimalike objektide järgi. [5]

Mudeli süntees ei lahenda ka tervet ruudustiku korraga, vaid teeb seda tükkhaaval, ehk kui tükis tekib vastuolu alustatakse uuesti ainult selle tükki lahendamist, mitte tervet ruudustiku. See muudab suuremate ruudustike puhul algoritmi lahendamise kiiremaks. [5]

[12] https://store.steampowered.com/app/1291340/Townscaper/



Pilt 4. Vasakud pildid on mudeli sünteesiga loodud pildid ja paremal lainefunktsiooni kokkulangemise algoritmiga loodud pildid, samal sisendil. [5]

3. Godot mootori laiendamine

Godot on avatud lähtekoodiga mängumootor, millega saab arendada nii 2D kui ka 3D mänge. Mootor kasutab skriptimiseks GDScripti, mida saab mugavalt kasutada Godot redaktoris. Godot on ka visuaalne arenduskeskkond, millega saab lihtsalt ja kiirelt muuta mänguobjekte.

Godot kasutab stseenide põhist süsteemi ning neid stseene saab kergesti taaskasutada. Stseenid koosnevad tippudest, mis moodustavad puu struktuuri, kus igal tipul on kindel tüüp ja funktsioon. Näiteks *Sprite* tüüpi tipp joonistab pildi ja *Collider* tüüpi tipp võimaldab kahe objekti kokkupõrkeid tuvastada. Samuti saab tipule lisada skripti, millega on võimalik selle tipu funktsionaalsust laiendada.

Andmete hoidmiseks kasutab Godot ressursi faile. Need failid võivad olla tippude küljes või salvestatud kettale. Sellised ressursid on näiteks tekstuurid ja kokkupõrke kujud (*CollisionShape*), mida *Sprite* ja *Collider* tüüpi tipud kasutavad.

Godot mootorisse on integreeritud Godot raamatukogu (*Godot Asset Library*), mis koondab kokku vabavaralised Godot laiendused ja võimaldab neid otse mootorist alla laadida. See muudab laienduste jagamise lihtsaks ning säästab arendajate aega.

Kasutaja saab ka ise Godot redaktorit laiendada, et muuta arendamist ja testimist mugavamaks. Selliseid lisasid on võimalik laiendusteks pakkida ja teiste kasutajatega jagada läbi Godot raamatukogu.[13]

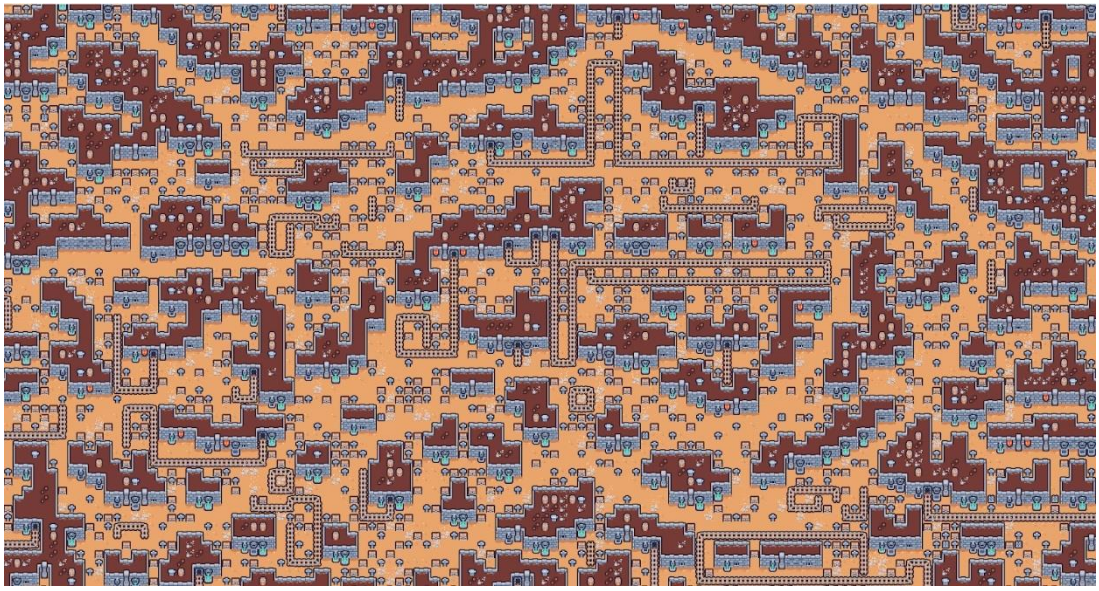
Käesolevas töös kasutatakse Godot versiooni 3.5. Valiti just see mootori versioon, et ka arendajad, kes kasutavad Godot 3.5-te saaksid laiendust kasutada. Laiendust on ka kergem uuendada Godot 3.5-elt Godot 4-le, kuna enamus funktsioone mis on olemas Godot versioonis 3.5 on ka olemas Godot versioonis 4, kuid mitte vastupidi.

[13] https://godotengine.org/

Laine funktsiooni kokkulangemise algoritm 2D laiendus

Godot raamatukogus on üks töötav WFC laiendus, mis on loodud Aleksi Bondi poolt[14]. Tema laiendus rakendab üldist piirangute põhist ülesannete lahendajat, mis lahendab etteantud ülesande defineeritud muutujate ja nende vaheliste reeglitega. Sellele kirjutas ta WFC 2D "ülesande" ehk määras piirangute põhisele ülesande lahendajale, kuidas on defineeritud objektid, reeglid ja mida tehakse ühel sammul.

Objektid ja nende vahelised reeglid määratakse näite järgi. Laienduses on ka implementeeritud tagasi astumine (*backtracking*), ehk kui tekib olukord, kus ühtele koordinaadile ei saa panna ühtegi objekti siis võetakse eelmine valik tagasi ning proovitakse valida teist objekti, jätkatakse edasi tagasi astumist, kuni enam vigu ei teki.

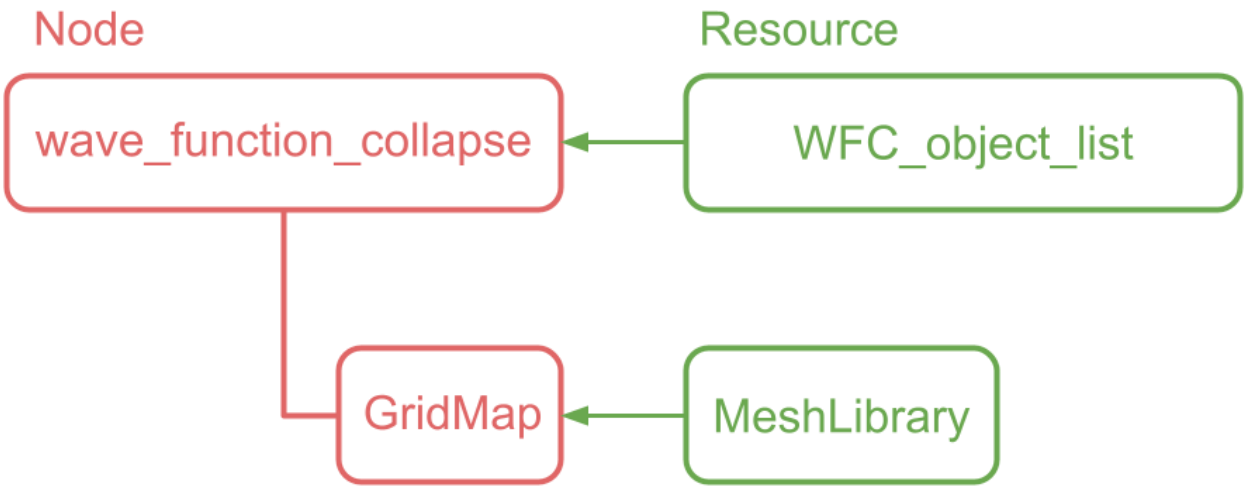


Kuvatõmmis 3. Aleksi Bondi WFC laiendusega loodud pilt

[14] https://godotengine.org/asset-library/asset/1951

4. Implementasioon

Selleks, et laiendust kasutada tuleb kõigepealt lisada Godot stseeni *wave_function_collapse* tipp. Selle tipu alamtipuks tuleb lisada *GridMap*. Neile tippudele tuleb ka luua vastavalt *WFC_object_list* ja *MeshLibrary* ressursi failid, milles laiendus oma andmeid talletab. Seda struktuuri on kujutatud Joonis 1, kus punasega on tähistatud stseenipuusse lisatavad tipud ja rohelisega nende külge pandavad ressursi failid.

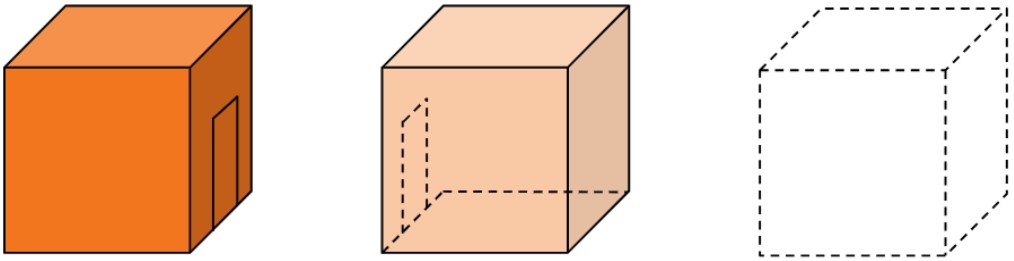


Joonis 1. Laienduse struktuur

Kui nüüd redaktoris valida *wave_function_collapse* tipp, siis avaneb *wave function collapse* aken, mida on kujutatud Kuvatõmmis 4:

1. *Add object* nuppu vajutades lisatakse *WFC_Object_listi* uus *WFC_object*, milles hoitakse objekti reegleid, 3D-mudelit (*mesh*) ja pööret (*rotation*). Objekti reegleid hoitakse iga külje kohta eraldi massiivis, *WFC_object_listis* defineeritud reeglite indeksitena (vt. Joonis 4). Pööre näitab kuidas 3D objekti on ümber oma keskpunkti pööratud.
2. *Edit rules* nupuga saab lisada ja muuta reegleid, mida saab määrata objektide külgedele. Reeglid määravad kas kaks objekti saavad külgepidi kõrvuti olla, ehk kui kahe objekti vastaskülgedel on sama reegel siis need kaks objekti võivad asetseda kõrvuti. Lisatud reeglitele saab määrata värvi, nime ja erisuse, mis aitab neid editoris eristada. Mõnikord on tarvis defineerida reegleid, nii et objekt ei saa iseenda kõrval olla, kuid sobib teiste sama tüüpi objektidega. Näiteks Joonis 2 on kaks uksega objekti ja üks tühi objekt. Kui tahaks ühendada mõlemad uksed tühja kastiga, nii et nad omavahel ei ühenduks, siis

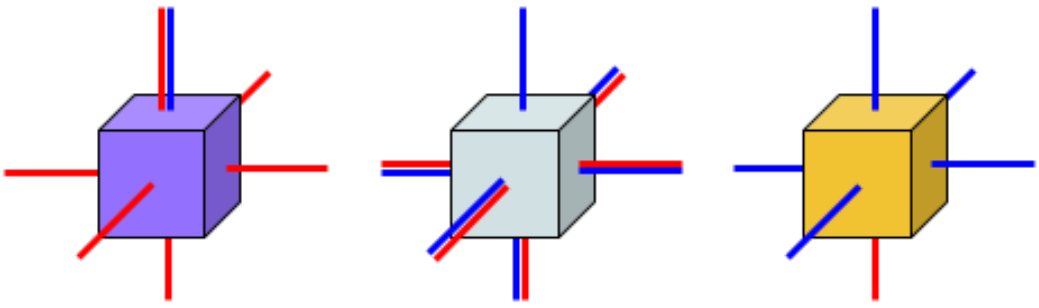
tuleks luua mõlemale küljele eraldi reegel, mis ühenduks tühja objektiga. Selle tulemusena oleks palju reegleid ning keerulisem aru objektide vahelistest ühendustest.



Joonis 2. Kaks uksega objekti ning tühi objekt

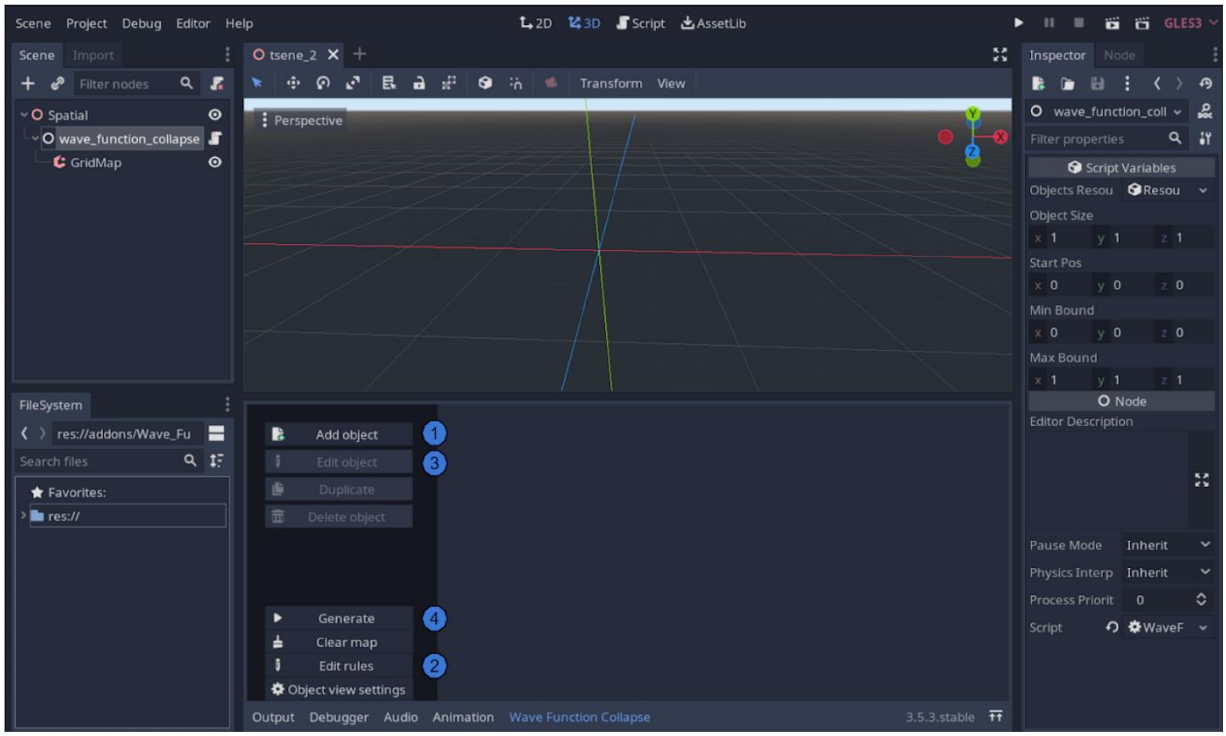
Selle jaoks on võimalus reeglile määrata erisus, mis keelab sama reegliga külgi ühendada. Seega, selle asemel et lisada mõlemale küljele eraldi reeglid, saame mõlemale lisada ühise reegli, mis ühendub tühja kastiga ja erilise reegli, mis keelab omavahel ühenduda.

3. Edit object nupuga (Kuvatõmmisel pole nupp aktiivne, kuna pole lisatud ühtegi objekti) saab lisatud objektide külgedele määrata reegleid. Näiteks Joonis 3 on defineeritud kolm objekti (lilla, hall ja kollane) ja kaks reeglit (punane ja sinine). Lilla objekti ülemisele küljele on määratud punane ja sinine reegel ning ülejäänud külgedel on ainult punased reeglid. Hallile objektile on ülemisele küljele määratud sinine reegel ja ülejäänud külgedele sinine ja punane reegel. Kollase objekti alumisele küljele on määratud punane reegel ja ülejäänud külgedele sinised reeglid.

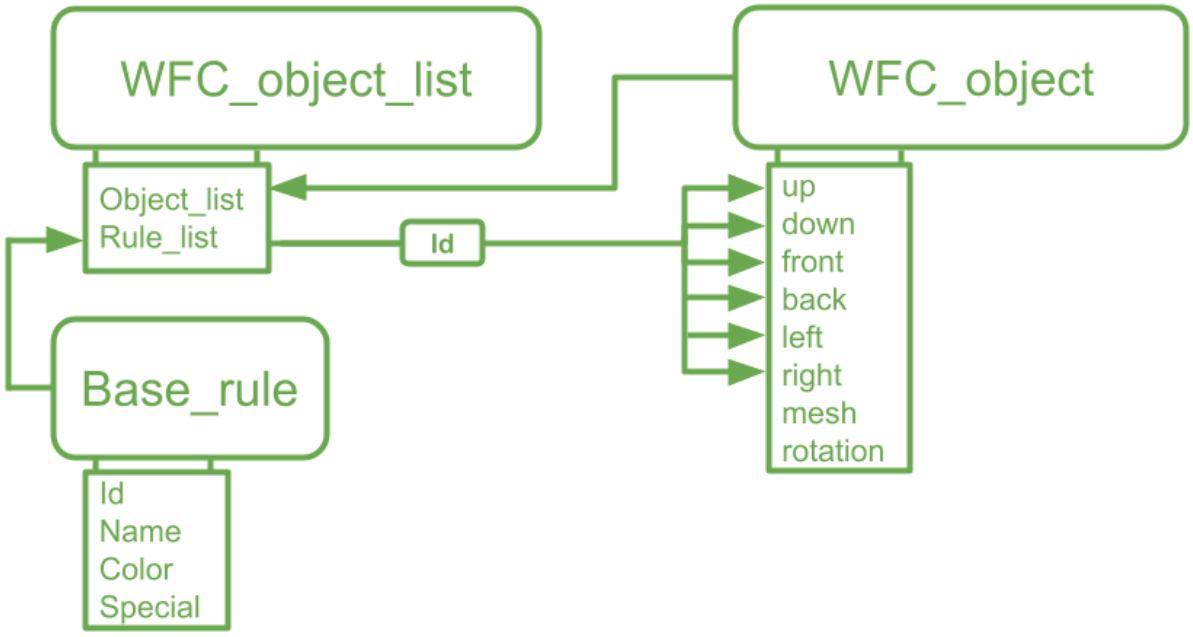


Joonis 3. Defineeritud objektid ja nendele määratud reeglid

4. *Generate* nupuga saab mängu käivitamata jooksutada algoritmi, mis teeb objektide ja reeglite seadistuste testimise kergemaks.



Kuvatõmmis 4. Redaktoris avanenud aken

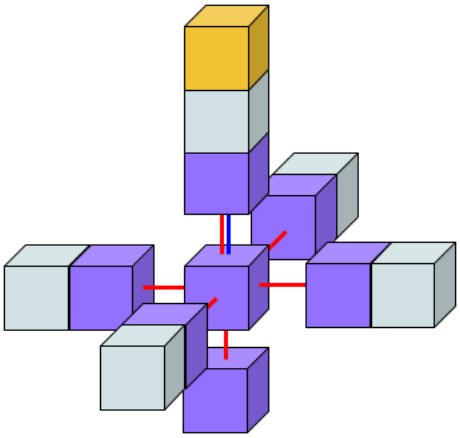


Joonis 4. Andmete paiknemine ressursides

4.1 Algoritm

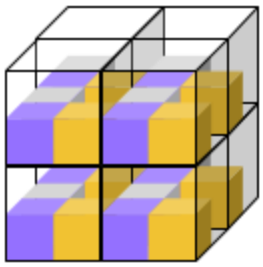
Kui kasutaja on loonud objektid ning lisanud nende külgedele reeglid ning algoritm käivitatakse, siis esmalt tühjendatakse *GridMap*. Seejärel luuakse külgnevuste massiiv, kus on iga vaadeldava objekti külje jaoks järjend objektidest, mis sobivad selle külje kõrvale.

Joonis 5 on näha lilla objekti lubatud külgnevused, vastavalt Joonis 3 defineeritud reeglitele. Ülemise külje kõrvale sobivad kõik objektid, kuna neil kõigil on alumisel küljel punane või sinine reegel. Keskmiste külgede kõrvale sobivad ainult lilla ja hall objekt, sest nendel on keskmistel külgedel punased reeglid ning kollasel objektil on ainult sinine reegel. Alumisele külje kõrvale saab panna ainult teist lillat objekti, sest hallil ja kollasel objektil on ülemisel küljel ainult sinine reegel.



Joonis 5. Lilla objekti lubatud külgnevused

Järgmiseks luuakse kolmemõõtmeline võre, mille mõõdud määrab kasutaja. Igale võre koordinaadile talletatakse kõik objektid, mis saavad selle võre koordinaatidel olla. Esialgu määratakse igale koordinaadile kõik objektid, sest need on kõik vaatlemata ja puuduvad piirangud millist objekti valida. Joonis 6 on kujutatud 2x2x2 võre, kus igal koordinaadil saab olla kõik kolm objekti.

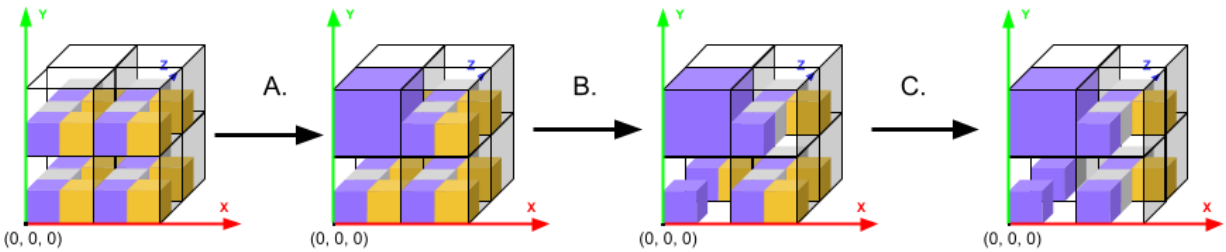


Joonis 6. 2x2x2 võre, kus igale positsioonile on määratud kõik võimalused.

Seejärel hakatakse võre koordinaatidele ükshaaval objekte määrama. Selleks valitakse vähima entroopiaga võre koordinaat, ehk koordinaat millel on kõige vähem sobivaid objekte. Esimesel korral on kõigil koordinaatidel entroopia võrdne ning alguskoordinaadi määrab kasutaja.

Sellele koordinaadile, kõigi sobivate objektide seast, valitakse üks juhuslik objekt ning teised võimalused eemaldatakse. Pärast seda uuendatakse ka võre naaberkoordinaatide võimalike objekte, nii et jääksid kehtima külgnevusreeglid. Kui selle tulemusena vähenes mõne naabri võimalike objektide arv, siis vaadatakse üle ka tema naabrid, kuni kõik muutunud naabrid on uuendatud.

Joonis 7 on kujutatud koordinaadile kindla objekti määramist. Sammul A. määratakse koordinaadile (0, 1, 0) lilla objekt. Selle tulemusena muudetakse sammul B. koordinaadi naabrite reegleid. Alumise naabri, koordinaadil (0, 0, 0), võimalikeks objektideks jääb ainult lilla, kuna see on ainus objekt, mis saab olla lilla objekti all. Teistel naabritel, koordinaatidel (0, 1, 1) ja (1, 1 ,0), jääb alles võimalus olla hall või lilla objekt. Seejärel sammul C. kontrollitakse naabrite naabrid üle. Koordinaadil (1, 0, 0) ja koordinaadil (0, 0, 1) kaob ära võimalus olla kollane objekt.



Joonis 7. Positsioonile kindla objekti valimine ja naabrite võimaluste vähenemine

Kui kõik objektide naabrid on uuendatud, jätkatakse koordinaatidele objektide määramist, kuni igale koordinaadile on jäänud ainult üks objekt. Lõpuks kui kõik igale koordinaadile on valitud kindel objekt, paigutakse *GridMapile* sisse 3D mudelid, mis vastavad valitud objektidele.

4.2 Andmestruktuurid

Algoritm kasutab oma tööks kolme massiivi: objektid, külgnevad objektid ja võre.

- Objektide massiivis hoitakse *wfc_objecte*.
- Külgnevate objektide massiivis hoitakse iga objekti külgnevusreegleid, ehk kuidas objektid võivad kõrvuti asetseda. Iga objekti külje kohta (ülemine, alumine, esi-, taga-, parem ja vasak külg) talletatakse 64-bitine täisarv, mille iga bitt vastab kindlale objektile objektide massiivis (*bit mask*). Joonis 5, lilla objekti külgnevustele vastab massiiv [7, 1, 3, 3, 3, 3], mis on binaarkujul [111, 001, 011, 011, 011, 011].
- Võre on kolmemõõtmeline massiiv mille iga koordinaat hoiab sobivaid võimalike objekte. Ühel koordinaadil hoitakse võimalike objekte 64-bitise täisarvuna, näiteks kui ühel koordinaadil saab olla objekt indeksiga üks või kaks, vastab see bittidele 110, mis on täisarv 6.

4.2.1 Külgnevate objektide massiivi täitmine

Külgnevus reeglite leidmiseks tehakse iga objekti iga külje ja vastaskülje (ülemine ja alumine külg, esi- ja tagakülg, parem ja vasak külg) vahel loogikatehe *and*. Kui leidub ühine reegel, ehk tulemus ei ole 0, siis muudetakse vastav bitt üheks.

Joonis 8 on kujutatud lilla objekti parema külje külgnevuste leidmist. Algselt on lilla objekti parema külje külgnevused binaarkujul 000. Seejärel tehakse loogikatehe *and* operatsioon lilla parema külje reeglite ja vasaku külje reeglite vahel.

$$01 \,\&\, 01 = 01$$

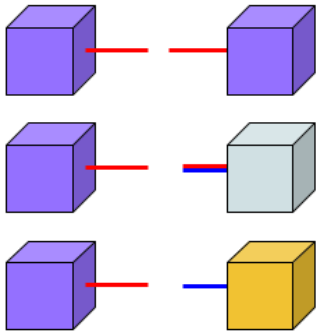
Kuna operatsiooni tulemus ei ole 0, siis üks bitt kattub ja lilla objekt lisatakse parema külje külgnevustele, mis on binaarkujul 001. Seejärel tehakse sama operatsioon halli objektiga.

$$01 \,\&\, 11 = 01$$

Ka selle operatsiooni tulemus ei olnud 0, seega lisatakse hall objekt parema külje külgnevustele, mis on binaarkujul 011. Viimaseks korratakse operatsiooni kollase objektiga.

$$01 \,\&\, 10 = 00$$

Seekord on tulemus 0, seega kollast ei lisata ja lilla objekti parema külje külgnevusteks jääb binaarkujul 011, mis on täisarvuna 3.

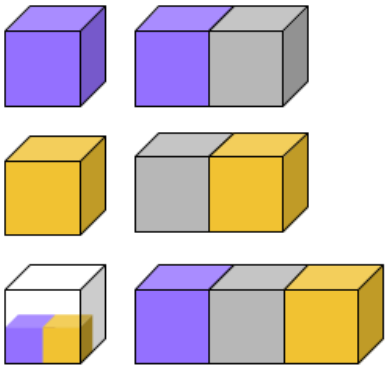


Joonis 8. Lilla objekti külgnevuste leidmine

4.2.2 Võimaluste vähendamine

Pärast seda kui on koordinaadile valitud kindel objekt, tuleb selle naabreid uuendada, et reeglid jääks kehtima. Seda protsessi nimetame võimaluste vähendamiseks. Selleks vaadatakse järjest läbi kõik valitud koordinaadi naabrid. Iga naabri jaoks leitakse võimalikud objektid naabri suunas, kui koordinaadil on võimalus olla mitu objekti, leitakse nende võimalike objektide ühend naabri suunas.

Näiteks Joonis 9 saab lilla objekti paremale küljele panna nii lillat kui halli objekti. Kollase objekti kõrvale saab panna kollast ja halli objekti. Seega kui koordinaadil on võimalikeks lilla või kollane objekt, siis selle paremale külje kõrvale panna nii lillat, kollast kui halli objekti.

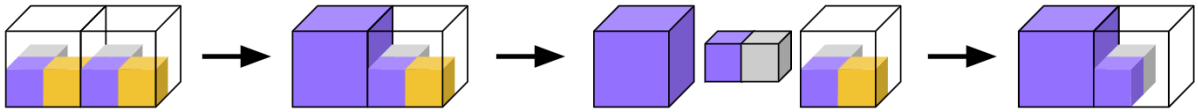


Joonis 9. Lilla ja kollase objekti ühiste võimaluste leidmine

Ühend leitakse koordinaati võimalike objektide suunaga lubatud objektide vahel loogikatehte *or* tegemist. Joonis 9 koordinaadil ühendi leidmine arvutuslikult on:

$$011 \mid 110 = 111$$

Kui on leitud lubatud objektid naabri suunas, siis leitakse selle naabri lubatud ühisosa. See leitakse nende vahel loogikatehte *and* tegemist, ning määratakse naabri võimalikeks. Joonis 10 on kujutatud parema naabri võimaluste vähendamist. Sammul A määratakse vasakule objektile lilla objekt ning alustatakse naabrite võimaluste vähendamist. Sammul B leitakse vasaku objekti kõik lubatud võimalikud paremale. Sammul C leitakse vasaku objekti paremale lubatud võimalike ja parempoolse objekti võimalike ühisosa. Seega jääb parempoolse objekti võimalikeks lilla ja hall objekt.



Joonis 10. Naabri võimaluste vähendamine

Joonis 10 leitud ühisosa arvutatakse järgmiselt:

$$011 \ \& \ 111 = 011$$

4.3 Algoritmi optimeerimine

Vähima entroopiaga positsiooni leidmine on aeglane protsess, kuna tuleb läbi vaadata kõik võre koordinaadid ja leida nende seast vähim. Seda tuleb teha igal võre koordinaadil ning seetõttu muutub algoritm aeglasemaks, mida suurem on võre.

Algoritmi jõudluse parandamiseks loodi ka teine versioon algoritmist, milles kasutatakse võre asemel entroopia põhjal järjestatud massiivi. Kui hoida seda massiivi järjestatud kogu algoritmi vältel saab vähima entroopiaga elemendi lihtsasti leida.

Selles massiivis hoitakse iga elemendi kohta nelja väärtust:

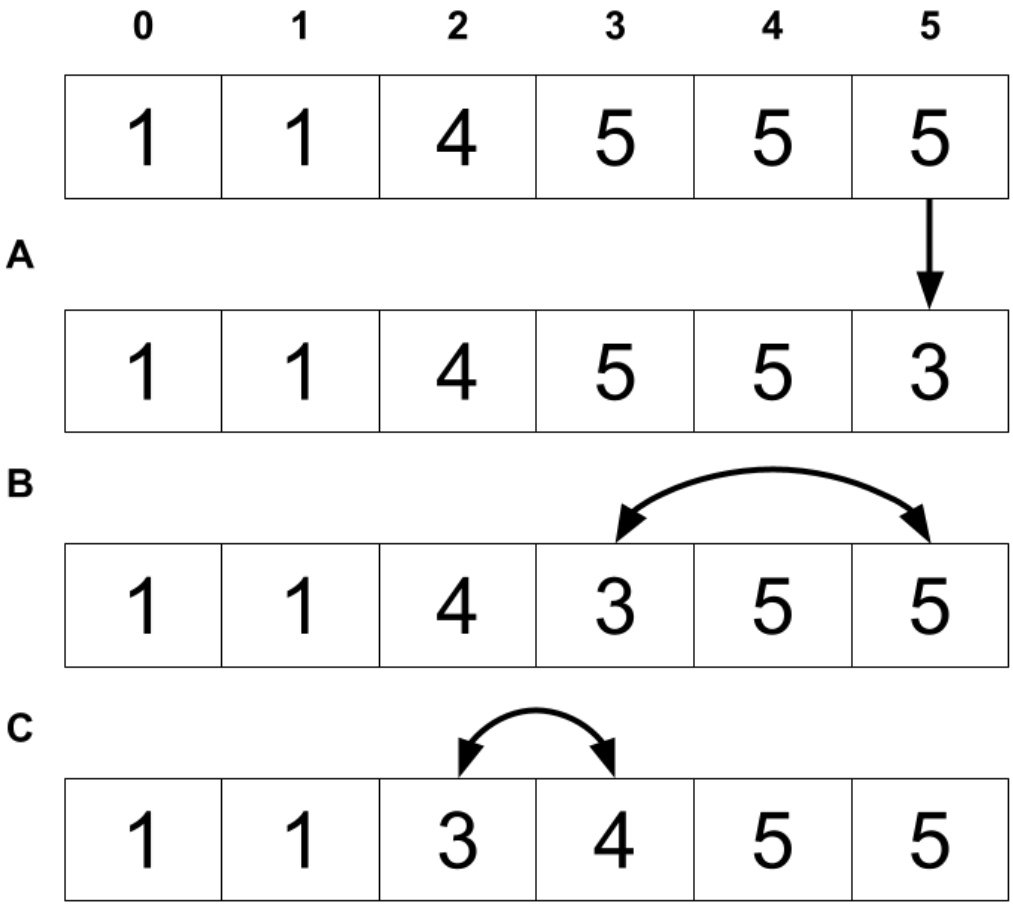
- koordinaat - viitab kohale maailmas,
- 64-bitine täisarv võimalikest objektidest mis sobivad koordinaadile,
- entroopia väärtus on võimalike objektide arv massiivis, et seda ei peaks iga kord uuesti arvutama,
- viited naaberelementidele selles massiivis, mis külgnevad selle koordinaadiga.

Algoritmi käivitamisel täidetakse see massiiv kõigi koordinaatidega ja sobivateks objektideks määratakse igale koordinaadile kõik objektid ja lisatakse kõigile massiivi elementidele ka

viited naaberelementidele. Seejärel segatakse massiivi elemendid ära, et võimaldada hiljem koordinaate juhuslikult valida. Kuna alguses on igal positsioonil võimalus olla ükskõik milline objekt, siis on massiiv entroopia järgi juba järjestatud. Kasutaja määratud algus positsioon viiakse massiivis esimeseks ning defineeritakse pointer, mis sellele indeksile viitab.

Pärast iga objekti määramist liigutatakse pointerit ühe võrra edasi, nii et massiivi algusesse jääksid kõik määratud ja lõppu määramata koordinaadid. Sammuti muudetakse massiivi lõpus olevate elementide asukohti pärast naabrite uuendamist ja entroopia ümberarvutamist säiliks järjestatus.

Algoritmi sammul võetakse pointeri kohalt esimene vähima entroopiaga veel määramata koordinaat. Sellele koordinaadile valitakse kõigi sobivate objektide seast üks juhuslik objekt. Valitud koordinaadi entroopia muudetakse üheks ja pointerit liigutatakse ühe võrra edasi. Seejärel uuendatakse kõiki selle positsiooni naabreid ja vajadusel vähendatakse nende võimalike objektide arvu ja entroopiat. Kui positsiooni entroopia vähenes, siis, positsioonide massiivi järjestatusse säilitamiseks, liigutatakse positsioon õigesse kohta massiivis. Selleks kontrollitakse, kas temast eelmisel indeksil oleva elemendi entroopia on suurem, kui on siis leitakse kõige esimene selle entroopiaga positsiooni indeks ning vahetatakse need omavahel ära. Vahetamist jätkatakse kuni eelmise elemendi entroopia on väiksem, kui kontrollitava oma. Joonis 11 on kujutatud selle protsessi läbimist. Sammul A muutub indeksil 5 oleva positsiooni entroopia viiest kolmeks. Sammul B vahetatakse indeksitel 5 ja 3 olevad positsioonid, kuna indeksil 3 oli esimene entroopiaga viis positsioon. Sammul C vahetatakse indeksitel 2 ja 3 olevad positsioonid, sest entroopiaga neli on ainult üks positsioon.



Joonis 11. Massiivi entroopia järgi järjestatuna hoidmine.

Pärast kõigi muutunud naabrite uuendamist jätkatakse positsioonide valimist massiivist pointeri järgi, kuni pointer on jõudnud massiivi lõppu, ehk kõikidele positsioonidele on alles jäänud üks võimalik objekt.
Selline lähenemine teeb algoritmi kiiremaks, kuna esiteks ei pea käima läbi tervet kaarti iga algoritmi sammu korral vaid saab valida kiiresti massiivist õige positsiooni. Teiseks on praktikas koordinaadid, mille entroopiat vähendatakse ja massiivis liigutatakse, massiivi alguses. Sest üldiselt valitakse massiivis lähestikku asuvaid koordinaate, sest vähima entroopiaga koordinaatidega naabrid on lähedase entroopiaga, seega ei pea neid palju liigutama. Kolmandaks, kuna andmeid hoitakse ühemõõtmelises massiivis, siis on nad mälus lähestikku ja neile saab kiiremini ligi.

5. Testimine

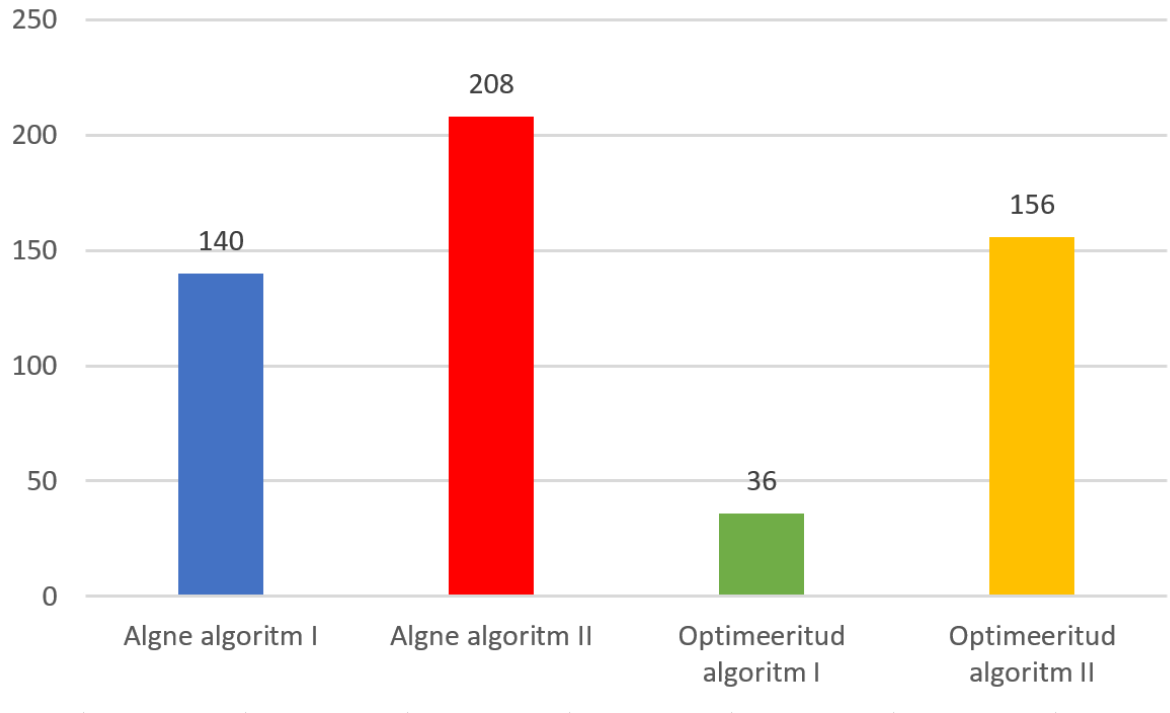
Jõudluse testimine viidi läbi arvutiga millel on Intel ® Core™ i5-1135G7 protsessor.

Testimist tehti kahes osas: kõigepealt testiti mõlemat algoritmi andmestikul, milles oli kaks objekti ja kaks reeglit, seejärel viidi testimine läbi andmestikuga milles oli 14 objekti ja 2 reeglit. Mõlemas osas, testiti algoritmi kiirust kolme võre suurusega:

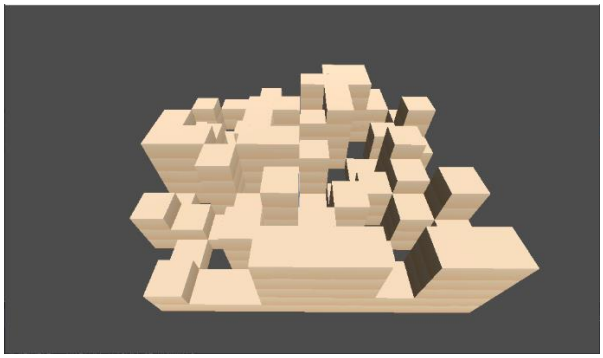
10 x 5 x 10, 20 x 10 x 20 ja 40 x 10 x 40, milles oli vastavalt 500, 4 000 ja 16 000 koordinaati. Et minimeerida erinevatel käivitustel tekkivat viga, korrati testimist iga võre suurusega 20 korda ning võrreldi keskmist ajakulu millisekundites.

5.1 Testimine võrel suurusega 10 x 5 x 10

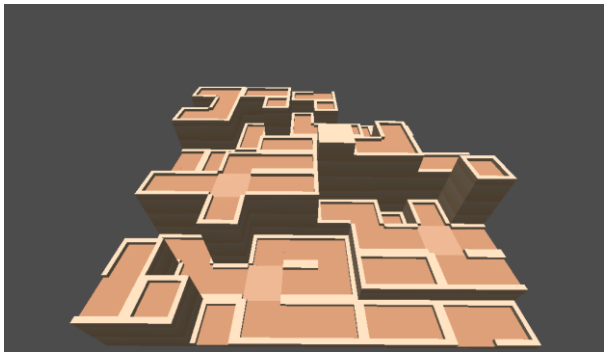
Joonis 12 on kujutatud algoritmide keskmised tööajad 10 x 5 x 10 võrel. Algse algoritmi keskmine tööaeg esimesel andmestikul oli 140 ms (sinine tulp) ning teisel andmestikul 208 ms (punane tulp). Optimeeritud algoritmi keskmine tööaeg esimesel andmestikul oli 36 ms (roheline tulp) ning teisel andmestikul 155 ms (kollane tulp). Optimeeritud algoritm oli esimesel andmestikul mitu korda kiirem algsest algoritmist, kuid teisel andmestikul oli keskmiste aegade vahe väiksem.



Joonis 12. Keskmised tööajad millisekundites 10 x 5 x 10 võrel.



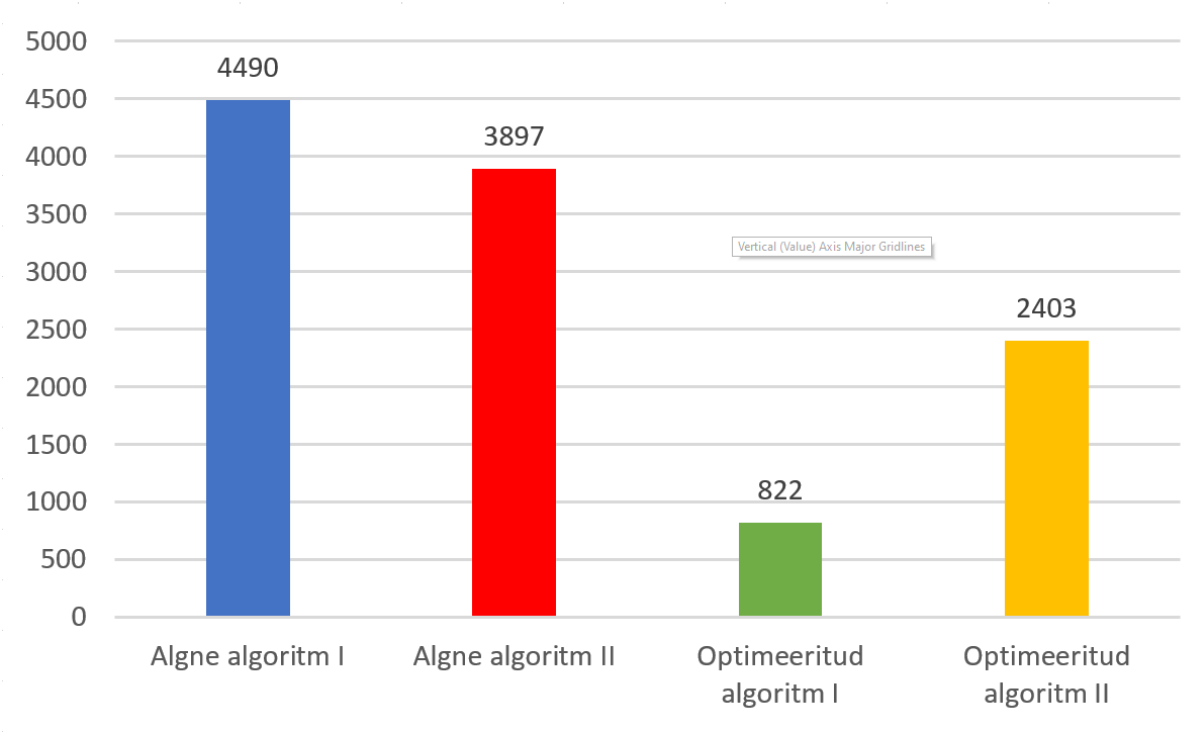
Kuvatõmmis 5. 10 x 5 x 10 maailm esimesel andmestikul.



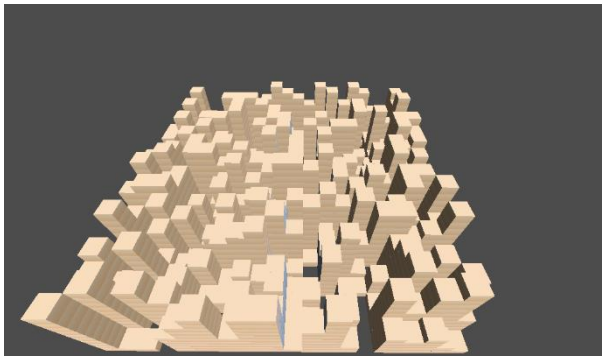
Kuvatõmmis 6. 10 x 5 x 10 maailm teisel andmestikul.

5.2 Testimine võrel suurusega 20 x 10 x 20

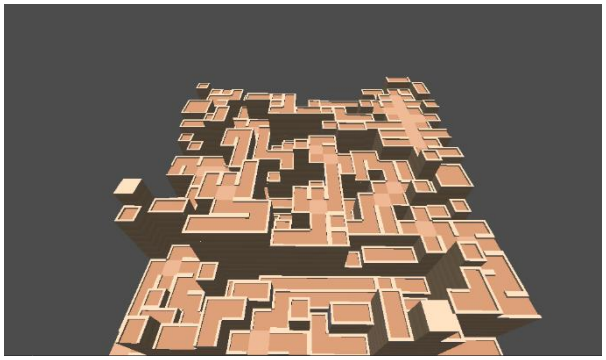
Joonis 13 on kujutatud algoritmide keskmised tööajad 20 x 10 x 20 võrel. Algse algoritmi keskmine tööaeg esimesel andmestikul oli 4490 ms (sinine tulp) ning teisel andmestikul 3897 ms (punane tulp). Optimeeritud algoritmi keskmine tööaeg esimesel andmestikul oli 822 ms (roheline tulp) ning teisel andmestikul 2403 ms (kollane tulp). Optimeeritud algoritm oli mõlemal andmestikul kiirem, kui algne algoritm. Kuid huvitaval kombel oli algne algoritm kahe objektiga andmestikul aeglasem kui 14 objektiga. See tuleneb sellest, et esimesel andmestikul on reeglid defineeritud nii, et koordinaadile objekti määramine naabrite võimalusi eriti ei vähenda. Seevastu teise andmestiku reeglid mõjutavad naabrite võimalusi rohkem, mille tulemusena saab igal algoritmi sammul rohkem koordinaate korraga määratud. Seetõttu ei ole vaja teise andmestikuga nii tihti entroopiat arvutada kui esimese andmestikuga. Kuna optimeeritud algoritmil on entroopia määramine ülikiire, siis väike andmestik hoopis kiirendab protsessi.



Joonis 13. Keskmised tööajad millisekundites 20 x 10 x 20 võrel.



Kuvatõmmis 7. 20 x 10 x 20 maailm esimesel andmestikul.

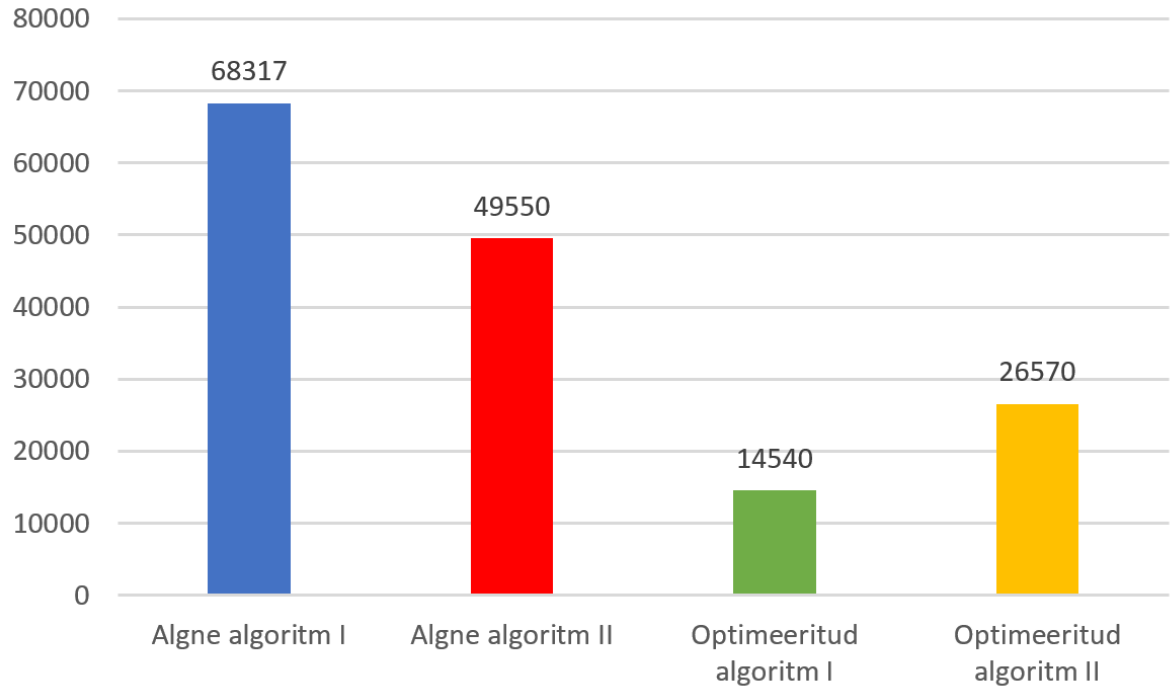


Kuvatõmmis 8. 20 x 10 x 20 maailm teisel andmestikul.

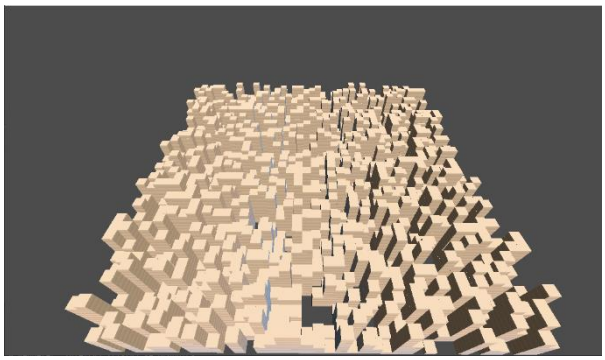
5.3 Testimine võrel suurusega 40 x 10 x 40

Joonis 14 on kujutatud algoritmide keskmised tööajad 40 x 10 x 40 võrel. Algse algoritmi keskmine tööaeg esimesel andmestikul oli 68 317 ms (sinine tulp) ning teisel andmestikul 49 550 ms (punane tulp). Optimeeritud algoritmi keskmine tööaeg esimesel andmestikul oli 14 540 ms (roheline tulp) ning teisel andmestikul 16 570 ms (kollane tulp). Ka seekord oli optimeeritud algoritm ligikaudu poole kiirem algsest algoritmist, kuid teisel andmestikul võttis

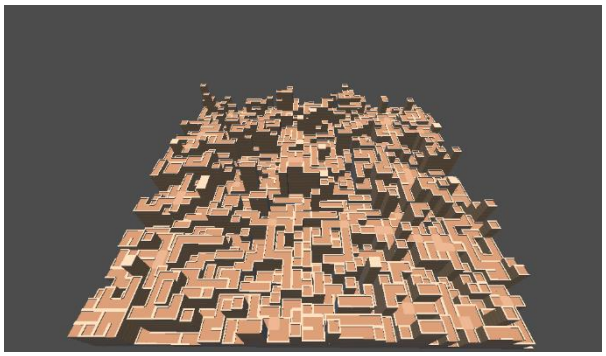
siiski rohkem kui 26 sekundit aega. Sellise suurusega võrel on mõistlik genereerida maailm enne mängu käivitamist valmis, nii et mängija ei peaks mängu ajal algoritmi järel ootama.



Joonis 14. Keskmise tööajad millisekundites 40 x 10 x 40 võrel.



Kuvatõmmis 9. 40 x 10 x 40 maailm esimesel andmestikul.



Kuvatõmmis 10. 40 x 10 x 40 maailm teisel andmestikul.

5.4 Testimise tulemused

Testimine näitas, et optimeeritud algoritm on kiirem kui algne algoritm. Eriti kiire oli optimeeritud algoritm väiksel andmestikul, sest entroopia muutumist toimub vähe seega järjekorra muutmist peaaegu ei toimugi. Godot *profilerit* uurides leiti, et kõige ajakulukam funktsioon optimeeritud algoritmil on ühe naabri reeglite uuendamine, kuna seda funktsiooni

kutsutakse kõige tihemini välja. Kuna GDScript on interpreteeritav keel, siis tsüklid ja rekursioon on aeglane, siis tuleks tulevikus kirjutada ümber GDNative liidesena, mis põhineb C++ keelel.

Kokkuvõte

Käesolevas töös uuriti lainefunktsiooni kokkulangemise algoritmi kasutamist mängudele sisu loomiseks. Samuti implementeeriti laiendus Godot mängumootorile, millega saab luua kolmemõõtmelisi maailmu, kasutaja enda määratud mudelite ja reeglite põhjal. See laiendus sisaldab ka kasutaja liidest, millega mudeleid ja reegleid lihtsasti konfigureerida. Algoritmi ennast võib kasutada, nii redaktoris ilma mängu käivitamata kui ka mängu jooksmise ajal.

Seda laiendust optimeeriti ja viidi läbi jõudlustestimine, millega hinnati selle sobivust mängu käigus sisu loomiseks. Testimisest selgus, et optimeeritud algoritm on väikesete võrestike suurustega piisavalt kiire, et seda mängu jooksul maailma loomiseks kasutada, kuid suurema võrestiku korral, tuleks eelistada maailma ette genereerimist.

Rakendus on saadaval nii Godot raamatukogus kui ka Git repositooriumis MIT litsensi all. Tulevikus on plaanis portida laiendus ka Godot versioonile 4, milles on uusi võimalusi selle kiiremaks muutmiseks. Samuti võib luua GDNative või GDExtension versiooni, mis põhineks C++ keelel ja oleks tõenäoliselt oluliselt kiirem.

Viidatud kirjandus

[1] Hendrikx M., Meijer S., Velden J., and Iosup A. Procedural content generation for games: A survey. ACM Transactions on Multimedia Computing, Communications, and Applications. Feb 2013 , Volume 9, Issue 1, Article No 1, 1–22. DOI : https://doi.org/10.1145/2422956.2422957

[2] Møller T., Billeskov J. Expanding Wave Function Collapse with Growing Grids for Procedural Content Generation. Master thesis. Aalborg University Copenhagen, 2019 . https://www.researchgate.net/publication/334416222_Expanding_Wave_Function_Collapse_with_Growing_Grids_for_Procedural_Content_Generation

[3] Newgas A. Tessera: A Practical System for Extended WaveFunctionCollapse. In Proceedings of the 16th International Conference on the Foundations of Digital Games (FDG '21). Aug 2021, Article 56, 1–7. DOI: https://doi.org/10.1145/3472538.3472605

[4] Merrell P. Example-based model synthesis. In Proceedings of the 2007 symposium on Interactive 3D graphics and games, Apr 2007, 105-112. DOI : https://doi.org/10.1145/1230100.1230119

[5] Merrell P, Comparing Model Synthesis and Wave Function Collapse. 2021. https://paulmerrell.org/wp-content/uploads/2021/07/comparison.pdf

Lisad

I. Lähtekood

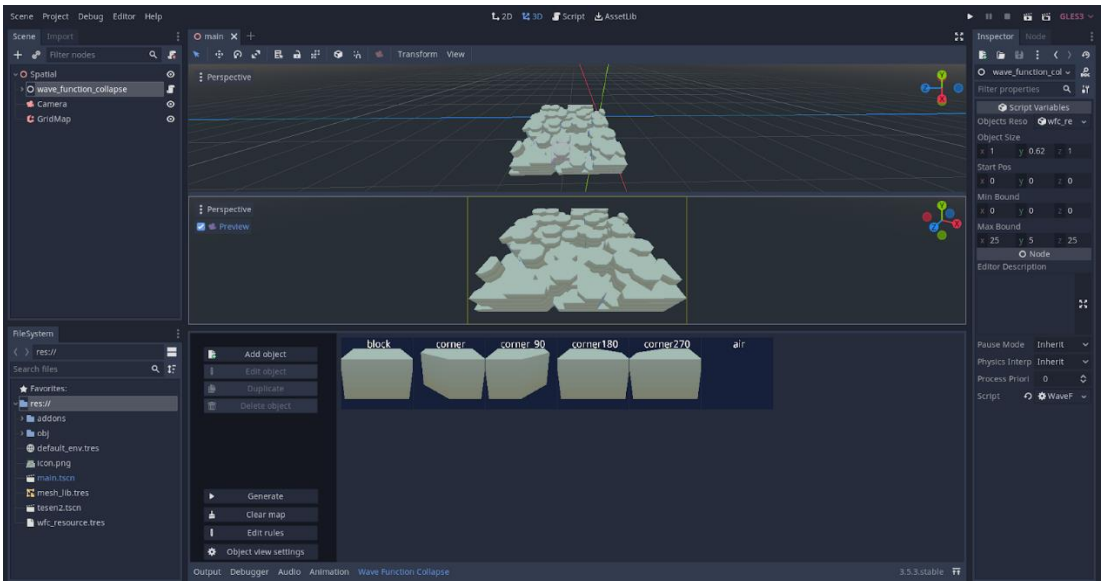
Lähtekoodi ja sellega seonduvaid faile leiab repositooriumist: https://github.com/MarkusMannil/WaveFunctionCollapse3DPlugin

Ning Godot raamatukogust leiab laienduse: https://godotengine.org/asset-library/asset/2888

II. Kasutusjuhend

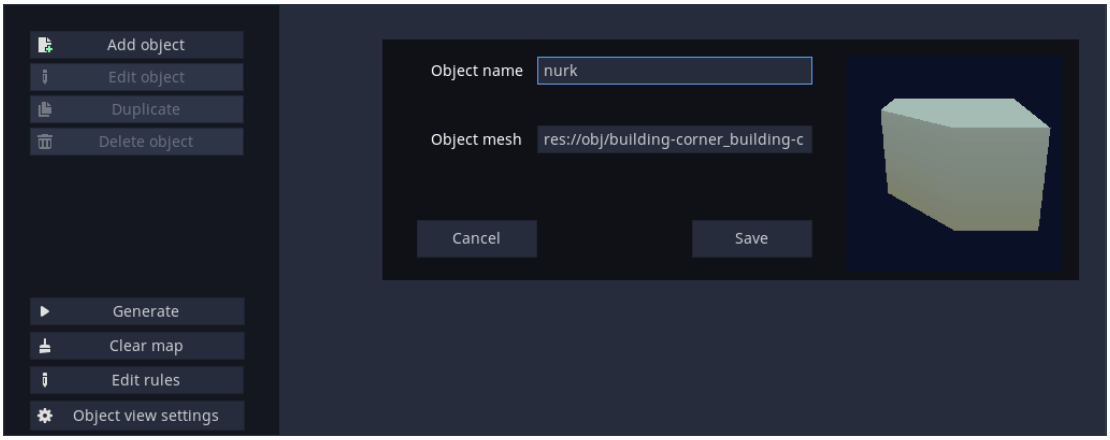
Kui WFC laiendus on Godot raamatukogust alla tõmmatud, siis tuleb esiteks see aktiveerida. Selleks tuleb valida ülemistelt ribalt *project* → *project settings* → *plugins* ning määrata WFC laiendus aktiivseks.

Seejärel saab 3D stseeni lisada *Wave function collapse* tippu. Tipule tuleb lisada alamtipp GridMap, kuhu hakatakse defineeritud objekte asetsema. Neile tippudele tuleb ka luua vastavalt *WFC_object_list* ja *MeshLibrary* ressursi failid, milles laiendus oma andmeid talletab. Seejärel valides *Wave function collapse* tipp, avaneb all *Wave Function Collapse* Aken.



Objektide lisamine

Objekti lisamis (*add object*) nuppu vajutes saab lisada uue objekti, milleks küsitakse kasutajalt nime ja võrestiku (*Mesh*). Objekti lisamiseks tuleb peab sisestama nime, kuid võrestiku võib jätta tühjaks, et luua tühja objekti.



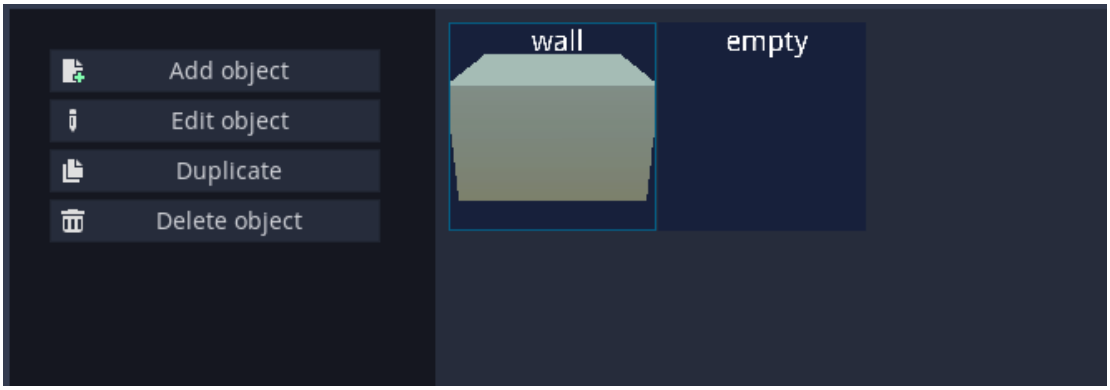
Reeglite muutmine

Reeglite muutmise (*edit rules*) nupuga saab lisada ja muuta reegleid, mida saab objekti külgedele määrata. Objekti külgedele lisatud reeglid lubavad samade reeglitega vastaskülgi kõrvuti asetseda

Reeglitele saab määrata värvi ja nime mis aitab neid eristada. Samuti saab reeglile lisada erisuse, mis keelab külgedel kõrvuti olla, isegi kui teised külje reeglid seda lubaksid.

Objektide vaade

Kõiki lisatud objekte kuvatakse paremal. Objektile vajutades valitakse objekt ja muutuvad aktiivseks muuda (*edit*), kopeeri (*duplicate*) ja kustuta (*delete*) objekt nupud.

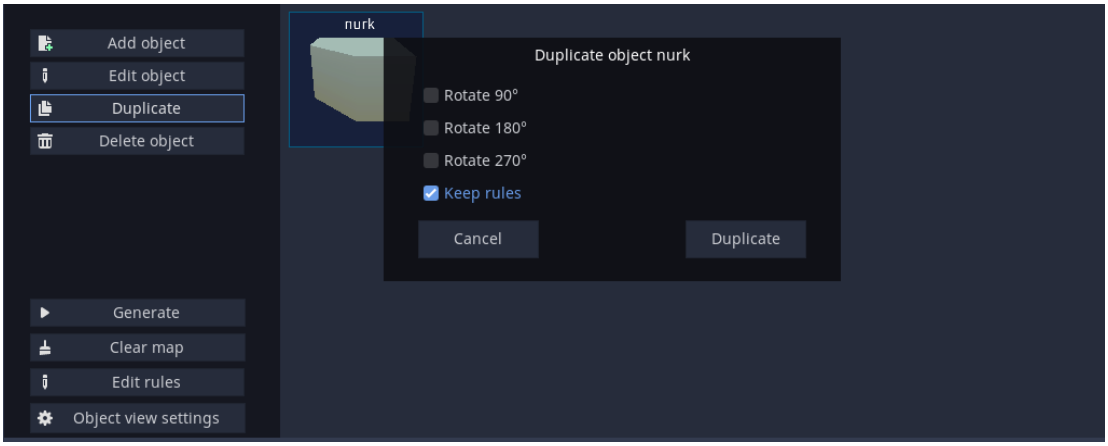


Objektide muutmine

Kui on valitud objekt ning vajutada muuda objekti (*edit object*) nupule, siis avaneb valitud objekti muutmise vaade. Selles saab lisada objekti eri külgede alt vaadata ja lisada neile reegleid mis on varem loodud reeglite lisamis vaates. All paremal (*connecting tiles all*) näidatakse ka kõiki objekte, millega vaadeldav külg saab ühenduda.

Objektide kopeerimine

Kopeeri nupuga saab valitud objektist luua koopia. Kui valida mõni pööre, siis koopiat pööratakse selle pöörde võrra ning valides mitu pööret, luuakse iga pöörde kohta uus koopia. Kui säilita reeglid (keep rules) on valitud, siis koopiatele jäävad reeglid alles, ning pööretega koopiate reegleid liigutatakse vastavalt ringi.



Objektide kustutamine

Kustuta objekt (*delete object*) nupuga kutsutakse valitud objekt.

Algoritmi jooksutamine redaktoris

Genereeri nupule (*generate*) vajutades käivitatakse algoritm, ning pannakse võred võrekaardile.

Kaardi tühjendamine

Puhasta (*clear map*) nupule vajutades eemaldatakse kõik võrestikud võrekaardilt.

III. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, Markus Männil

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose „Lainefunktsiooni kokkulangemise algoritmi laiendus mängumootorile Godot„ mille juhendaja on Jaanus Jaggo, reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.
2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.
3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.
4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Markus Männil

15.05.2024