Conway’s Game of Life

Ohjelmoinnin peruskurssin harjoitustyö  
Kerttulin Lukio 1.4.2021  
Tehnyt: Artturi Kerminen

SIsällysluettelo

[Ohjelman esittely 1](#_Toc68194831)

[Kuvaus ja analyysi 1](#_Toc68194832)

[Ratkaisuperiaate 2](#_Toc68194833)

[Ohjelman koodirakenne ja optimointi 2](#_Toc68194834)

[Ohjelman aikakompleksisuus ja optimointistrategiat 2](#_Toc68194835)

# Ohjelman esittely

## Kuvaus ja analyysi

Ohjelma perustuu edesmenneen John Conwayn laatimaan soluautomaattin nimeltään [Conway’s game of life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life), joka julkaistiin vuonna 1970. Kenttä koostuu soluista, joita on x \* y määrä. Simulaatio perustuu Conwayn tarkasti laatimien sääntöjen mukaan, jotka kuuluvat:

1. Mikä tahansa elossa oleva solu, jolla on vähemmän kuin kaksi elossa olevaa naapuria kuolee
2. Mikä tahansa kuollut solu, jolla on tasan kolme elossa olevaa naapuria, muuttuu eläväksi
3. Kaikki muut solut kuolevat ja kaikki jo kuolleet solut pysyvät kuolleina seuraavassa sukupolvessa

Jotta ohjelma ei olisi yksipuolinen, se pyytää pelaajalta neljä syötettä:

1. Kentän koko (leveys, korkeus)
2. Elossa olevan solun väri
3. Kuolleen solun väri
4. Sukupolvien elinaika (kuinka kauan kestää, että uusi sukupolvi generoidaan)

joiden perusteella kenttä ja sen solut muodostuvat.

Kuten mainittu, ohjelma simuloi kenttää John Conwayn sääntöjen mukaan. Näitä sääntöjä on kuitenkin todella helppo muokata.

Toinen yleinen sääntösetti on Nathan Thompsonin Highlife, jonka säännöt kuuluvat:

1. Solu syntyy, jos sillä on kolme tai kuusi naapuria
2. Solu selviytyy seuraavaan sukupolveen, jos sillä on kaksi tai kolme naapuria

Soluautomaateille on tyypillistä sisältää erilaisia toistuvia kuvioita. Näiden kuvioiden eri tyyppejä ovat:

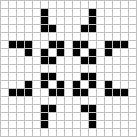
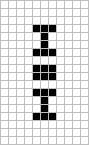
* ”Still lifes” (asetelma), eli kuvioita, jotka eivät muutu sukupolvesta toiseen
* ”Oscillators” (oskillaattori), jotka palaavat alkuperäiseen tilaan finiittisen määrän sukupolvia jälkeen
* ”Spaceships” (avaruusalukset), jotka liikuttavat itseään kentän läpi

Näistä toistuvista kuviosta Conway’s game of life -sääntöjen mukaan yleisimpiä ovat:

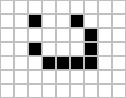
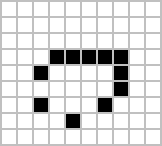
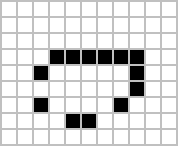
Asetelmat:

Oskillaattorit:

Avaruusalukset:

Joitain samoja kuvioita voi myös esiintyä Highlife -säännöissä, koska ne ovat hyvin samanlaisen Conway’s game of life -sääntöihin.

## Ratkaisuperiaate

Aivan ensimmäiseksi pääohjelma tulostaa komentokehotteeseen ohjelman nimen ja pienet käyttöohjeet. Tämän jälkeen pääohjelma kutsuu input\_handler tiedostossa olevan funktion – handle\_input() – joka pyytää käyttäjältä tarvittavat tiedot simulaation aloittamiseen. Kyseinen funktio samalla tarkistaa onko pelaajan antamat parametrit sopivassa muodossa, jotta ohjelma ei menisi rikki.  
Seuraavaksi pääohjelma alustaa PyGame -kirjaston, jotta ohjelma voisi renderöidä kuvan näytölle. Tämän jälkeen pääohjelma tekee uuden instanssin CellGrid -luokasta ja antaa käyttäjän syöttämät arvot argumenteiksi.

Seuraavaksi pääohjelmassa alkaa ns. Game Loop, jossa tapahtuu kaikki ohjelman toistuvat operaatiot, kuten:

* Näppäimistön syöttöjen havaitseminen
* Tekstin päivittäminen ikkunassa
* Kentän solujen päivittäminen
* Renderöintikutsu PyGamelle

Ohjelman suorittamisen aloittaa ehtolause, joka kutsuu init() -funktion.

# Ohjelman koodirakenne ja optimointi

## Ohjelman aikakompleksisuus ja optimointistrategiat

Tällä hetkellä ohjelman suorittama algoritmi, joka laskee seuraavan sukupolven kaikkien solujen uudet tilat, on hyvin hidas. Jos syötän kentän kooksi 200 \* 200, eli 40 000 solua, on ohjelman suoritus hidasta ja käyttäjän syöttämä sukupolven elinaika ei enää pidä paikkaansa hitauden takia. (Ohjelman algoritmin laskuaika > sukupolven elinaika).

Tällä hektkellä funktioiden *update\_neighbours(self)****,*** *render\_grid(self)* ja *update\_population(self)* aikakompleksisuus on suurinpiirtein , jossa N on kentän leveys ja M on kentän korkeus. Tämän algoritmin suoritusaika nousee hyvin, kun kentän kokoa kasvaa edes vähän molemmilla akseleilla.

Optimointitapoja on tässä tapauksessa monenlaisia. Listaan tapoja, joita tulee mieleen ja joita olen tutkinut.

* Lista, jossa on kaikki solut, joita on muutettu. Tällä tavoin ei tarvitsisi iteroida koko kentän läpi.
* Koodin pyörittäminen suoraa grafiikkakortilla. Tämän voi toteuttaa suhteellisen helposti varjostimilla. Tämän voisi toteuttaa vaikka Unity -pelimoottorilla ja Unityn Compute shader -varjostinta käyttäen.
* Koodin monisäikeistys Pythonin threading -kirjastolla.
* Hashlife -algoritmin käyttäminen. Hashlife on nopein Conway’s game of lifen populaatioden laskualgoritmi, mitä on keksitty. Sen toiminta perustuu Quad tree -rakenteeseen ja hajautustauluihin.
* Koodin kirjoittaminen ja suorittaminen matalalla tasolla, esimerkiksi C++ ja OpenGL. Tämä on itselleni mieluisin vaihtoehto ja toteutan sitä parhaillaan.

Esitin vain viisi erilaista optimointitapaa, mutta niitä on varmasti enemmän. Kaikkien mahdollisten optimointien listaaminen ei kuitenkaan ole tärkeätä.