Pelin algoritmit

Ryhmä en tiedä

Tuomas Kyttä H3408

Arttu Heinonen H3425

Joonas Katainen H3248

Maaliskuu 2014

Ohjelmistotekniikan koulutusohjelma

Tekniikan ja liikenteen ala

Sisältö

[1 Johdanto 2](#_Toc405374349)

[2 Algoritmin tehokkuus & erityispiirteet 2](#_Toc405374350)

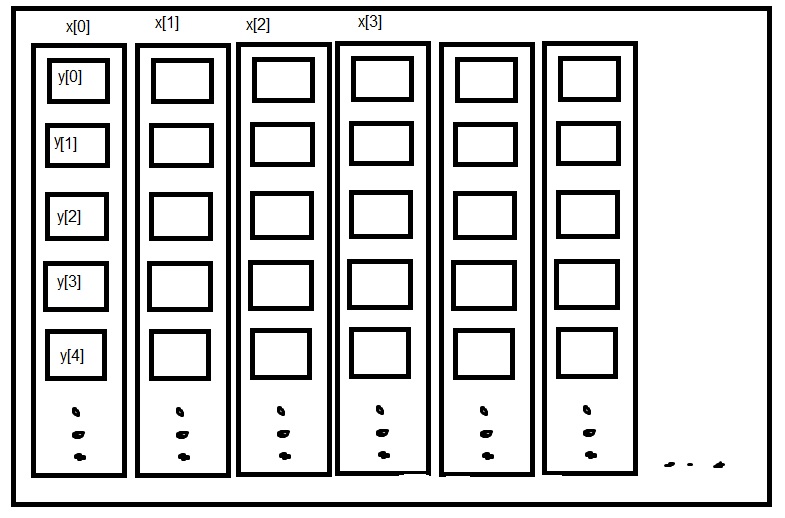
[3 Testiaineiston generointi & testaus 5](#_Toc405374351)

[4 Yhteenveto 5](#_Toc405374352)

# Johdanto

Olio ohjelmointi 2 kurssilla teemme kolmen hengen ryhmässä 2d tasohyppely peliä. Kartan koon kasvaessa peli alkoi kuitenkin hidastua johtuen piirtämisestä ja törmäystarkastelusta. Aloimme pohtia tähän ratkaisuja ja hyväksyimme Kytän esittämän idean kartan paloittelusta eri ruutuihin x ja y akselille. Tällöin tekisimme piirron ja törmäystarkastelut vain pelaajan lähellä oleville ruuduille. Päämääränä oli, ettei kartan koko tai vihollisten lukumäärä vaikuttaisi pelin ajettavuuteen vaan ainoastaan alku käynnistykseen.

# Algoritmin tehokkuus & erityispiirteet

Alussa kartta parsittaisiin kartta editorissa käyttäjän tekemistä paloista merge tilassa. Käyttäjällä on tällä hetkellä kyky uudelleen rakentaa kartta ja tallentaa se. Tallennettu kartta avataan peli tilassa ja parsitaan kolmiulotteiseksi vektoriksi x- ja y akselin mukaan kuvan 1 tavoin.

Kuva 1

Pseudo-koodina algoritmi jolla mappi rakennettiin on seuraavanlainen:

Map()

{

kerroin = 0

koko = 100

avaa haluttu teksti tiedosto

kun on rivejä

{

jos rivi on --

{

heitä mapObjects vektori yRivi vektorin jatkoksi

kerroin++

tyhjää mapObjects vektori

jos kerroin on yhtäsuuri kuin koko

{

heitä yRivi vektori xRivi vektorin jatkoksi

muuta kerroin nollaksi

tyhjää yrivi vektori

}

}

muulloin

{

parsi saatu rivi kolmeen osaan: nimi, x koordinaatti ja y koordinaatti

vertaile minkä objectin pointteri luodaan mapObject vektoriin ja anna sille sijainti

}

}

sulje tiedosto

Tässä vaiheessa kartan paloittelun etuja ei havaita edes teoreettisesti. Kartan luominen hidastuu verrattuna siihen että kaikki esineet luotaisiin osoittimina vain yhteen vektoriin. Edut heijastuvat vasta kartan, pelaajan ja vihollisten päivittämisessä sekä piirrossa.

Pelaaja on aina ruudun keskellä, kamera seuraa pelaajaa. Tämä mielessä pseudo-koodi kartan piirtoon on seuraavanlainen

piirrä()

{

x = kameran keskusta().x / ruudun leveydellä

y = keskusta().y / ruudun korkeudella

for (int xx = -1; xx < 2; ++xx)

for (int yy = -1; yy < 2; ++yy)

if (x + xx >= 0 && y + yy >= 0 && x + xx < xRivi.size() && y + yy < xRivi.size())

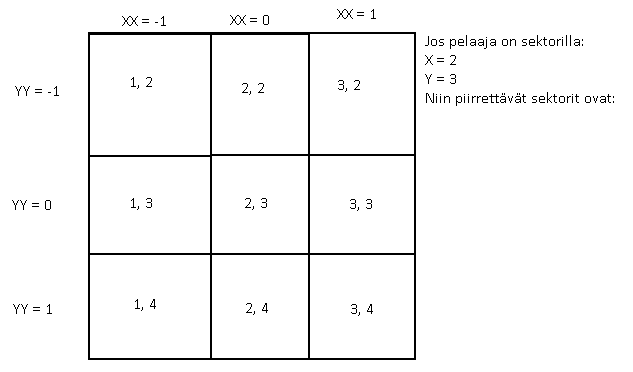
for (int i = 0; i < xRivi[x + xx][y + yy].size(); ++i)

piirrä(xRivi[x + xx][y + yy][i]->shape);

}

Tällöin maksimissaan 9 pelaajaa lähinnä olevaa ruutua piirretään sen sijaan että koko kartta piirrettäisiin kerralla. Tässä joudumme tekemään kaksi laskua ja 4+4+12+ruudussa olevien kappaleiden määrän verran vertailuja, mitkä ovat kevyitä tehdä.

Samanlaista logiikkaa käytettiin pelaajan päivittämisessä, vihollisten piirtämisessä ja päivittämisessä.



Kuva

# Toteutus

Aloitimme testaamaan edellä mainittua kartanluonti algoritmia katsomalla mitä eri kokoja kartanluontiin pystyi laittamaan ja kauan ne veivät aikaa. Ensimmäiseksi selvitimme maksimi arvon, kuinka monta karttapalaa karttaan pystyi asettamaan, ennen kuin kääntäjä lopettaa koodin ajamisen. Minimiarvo on itsestään selvästi 1x1 ruudukko, joka koostuu vain yhdestä kartanpalasta. Testasimme aina neliön muotoisilla kartoilla. Meidän lopetusehtomme oli ohjelman käynnistyminen ja kartan valmistuminen pelikäyttöön.

Testasimme tämän jälkeen vihollisten määrän vaikutusta ruudunpäivitykseen. Teoreettisesti vihollisten määrällä ei pitäisi olla vaikutusta ruudunpäivitykseen juuri lainkaan.

# Testiaineiston generointi ja testaus

Kartan generointia peliympäristöön testattiin mittaamalla aikaa manuaalisesti sekuntikellolla ja vihollisten vaikutusta ruudunpäivitykseen mitattiin peliin ohjelmoidulla ruudunpäivitys mittarilla. Seuraavassa taulukossa nähdään mittaustulokset:

//tulokset

# Yhteenveto