Tietorakenteiden harjoitustyön Toteutusdokumentti

1. Ohjelman yleisrakenne

Harjoitustyöni aiheena oli kehitellä erilaisia algoritmeja satunnaisten kaksiulotteisten roguelike-tyyppisten luolastojen proseduraaliseen generointiin. Lopputuloksena kehitin kaksi erilaista algoritmia, jotka tuottavat hyvin erilaisia luolastoja. Ensimmäinen näistä perustuu luolaston jakamiseen eri kokoisiin osiin, jolloin huoneiden ja niitä yhdistävien polkujen luominen ei vaadi erityisiä tarkistuksia. Toinen algoritmi puolestaan lähtee liikkeelle yhdestä huoneesta, jonka seinistä satunnaisesti luodaan uusia polkuja sekä huoneita. Ohjelman toteutuksessa kutsun näitä algoritmeja nimillä Dungeon divider ja Dungeon miner.

Dungeon divider lähtee liikkeelle BSP-puumaisesta ajatuksesta, jossa luolaston koko alue laidasta laitaan on puun juuri ja sen solut luolaston ala-alueita jaettuna erilaisilla tavoilla. Juuren lähimmät solut koostuvat siis kahdesta palasesta, jotka ovat saatu halkaisemalla alkuperäinen alue kahteen eri kokoiseen alueseen joko pysty- tai leveyssuunnassa. Nämä ala-alueet puolestaan jaetaan samalla tavalla kunnes alkuperäinen luolasto on jaettu tarpeeksi monta kertaa. Erilaisilla parametreilla on mahdollista säätää jako-operaatioiden määrää sekä tapaa, jolla jako toteutetaan.

Kun luolasto on jaettu tarpeeksi monta kertaa, luodaan jokaisen ala-alueen sisälle huone, joka voi olla pienempi kuin alueensa. Lopuksi yhdistetään ala-alueet toisiinsa poluilla, jotka kulkevat huoneesta toiseen siten, että jokaisesta huoneesta pääsee jokaiseen huoneeseen jotain reittiä pitkin.

Dungeon miner luo siis luolaston heittelemällä uusia huoneita ja polkuja kiinni jo aikaisempien huoneiden tai polkujen seiniin. Tehokkuuden lisäämiseksi jokainen seinä tallennetaan erilliseen pinorakenteeseen matemaattisessa mielessä välinä, esimerkiksi leveyssuuntainen seinä välinä [x1 + 1, x2 - 1] jossa x1 on seinän vasempi x-koordinaatti ja x2 oikea x-koordinaatti. Uutta huonetta tai polkua luodessa otetaan siis pinosta ylin seinäväli, ja valitaan sieltä satunnaisesti jokin koordinaatti, josta lähdetään huonetta tai polkua rakentamaan. Seinien lisääminen pinoon takaa sen, että uusia alueita lähdetään luomaan mahdollisimman kauas alkuperäisestä huoneesta, joka testien perusteella osoittautui tehokkaammaksi tavaksi täyttää koko luolasto huoneilla ja poluilla kuin täysin satunnainen seinien valitseminen.

Algoritmi päättää aluksi uutta huonetta tai polkua luodessa sen tulevat mitat ennen kuin on edes tarkistettu, voiko kyseinen rakenne mahtua siihen kohtaan luolastoa. Jos rakenne ei mahdu, seinä hylätään kokonaan ja mennään seuraavaan. Toinen vaihtoehto olisi toki luoda siihen kohtaan luolastoa niin iso huone tai polku kuin mahdollista, mutta se hidastaisi algoritmia huomattavasti eikä olisi ainakaan esteettisesti yhtään parempi vaihtoehto.
Molemmista algoritmeista löytyy myös mahdollisesti selkeämmät kuvaukset lähteissä.

2. Saavutetut aika- ja tilavaativuudet

Algoritmien tila- ja aikavaativuuksia oli vaikea tarkastella erityisesti Duongeon minerin tapauksessa. Siinä missä Dungeon dividerin aikavaativuus voitiin selvittää tehtyjen jako-

operaatioiden perusteella, Dungeon minerin ajankäyttöön ei vaikuta mikään muu syöte kuin taulukon koko.

Dungeon divider luo luolaston mittojen puitteissa sinne aina saman verran huoneita jako-operaatioista riippuen. Jos siis luolastossa on vain tilaa, luotujen huoneiden määrä $h$voidaan selvittää jakojen $divs$perusteella seuraavasti: $h = 2^{divs}$

Aika-analyysi voidaan siis miettiä joko jako-operaatioiden tai huoneiden lukumäärän pohjalta. Tarkasteltava metodi on `generateDungeon(int divisions)`, joka luo varsinaisen luolaston. Metodi kutsuu varsinaisen työn tekevää metodia `generateDungeon(MapRegion region, int divisions)` rekursiivisesti $h$kertaa, joka koostuu lähinnä vakioaikaisista metodeista. Ainoa poikkeus on `MapRegion getRoomWithPosition(Position position, MapRegion region)`, joka käy puurakennetta läpi binäärihakumaisesti ajalla suurimmillaan ajalla $O(log_2 h)$, mutta pienimmillään vakioajassa riippuen siitä, kuinka syvällä puussa ollaan. Metodia kutsutaan kaikkiaan siis kaikkiaan $h * 4$ kertaa, mutta keskimääräinen toiminta-aika on $O(log_2 \frac{h}{2})$, joten varsinaiseksi aikavaativuudeksi saadaan $O(hlog_2 h)$.

Kyseisessä aikavaativuusanalyysissä oletettiin, että huoneiden ja polkujen tallentaminen taulukkoon ei vaikuta kulutettuun aikaan. Oletus on luonnollisesti väärä, mutta kyseisen analyysin kannalta epäolennaista ja vertailun kannalta turhaa, koska kaikilla algoritmeilla menee taulukon muuttamiseen saman verran aikaa.

Tilavaativuuteen ei ole järkevää ottaa huomioon taulukon vaativaa tilaa. Dungeon divider varaa jokaista huonetta ja aluetta varten tilaa oman olion verran. Lisäksi `generateDungeon(...)` metodin rekursiivisen luonteen vuoksi muistia varataan myös sitä kautta enemmän. Tarkastelua varten tarvitaan uusi muuttuja $regions$eli puun solujen määrä, joka saadaan laskettua $regions = 2^{h-1}-1$. Tilavaativuuteen vaikuttaa siis metodin kutsukerrat, huoneiden lukumäärä sekä puun solujen määrä. `getRoomWithPosition(...)` vaikuttaa sinänsä tilavaativuuteen rekursionsa vuoksi, mutta sen varaama tila voidaan vapauttaa jo luolastoa luodessa. Tilavaativuudeksi saadaan siis $O(2h + 2^{h-1}-1)$eli yksinkertaistetummin $O(h + 2^{h-1})$.

Dungeon minerin aikavaativuuden selvittäminen on vaikeampaa. Algoritmin ajankäyttöön vaikuttaa hyvin monet muuttujat, ja vaihtelua esiintyy paljon. Karkea maksimianalyysi voidaan kuitenkin saada aikaiseksi kuvittelemalla, että algoritmi täyttää koko luolaston niin monella huoneella kuin sinne teoreettisesti mahtuu. Nyt siis h= luolaston korkeus, w= luolaston leveys, S= huoneen minimikoko ja r= huoneiden lukumäärä. Tarkastellaan seuraavaksi metodia `generateDungeon()`, joka generoi luolaston. Metodissa toistetaan looppia kunnes jokaisen luodun huoneen seinät ovat käyty läpi. Toistojen kerta on siis huoneiden lukumäärä kerrottuna kolmella, joka tässä kuvitteellisessa tilanteessa on $r = \frac{hw}{(S+1)^2}$. Näin siis tämän tilanteen aika-analyysiksi saadaan $O(3\frac{hw}{(S+1)^2})$ eli yksinkertaistetummin $o(\frac{hw}{(S+1)^2})$. Vaikka tämä analyysi ei vastaa realistista tilannetta, antaa se kuitenkin jonkinlaisen kuvan algoritmin ajankäytöstä.

Tila-analyysi on puolestaan helpompi, koska varsinaisesti tilaa käytetään vain seinien tallentamiseen pinoon. Tilavaativuudeksi saadaan siis yksinkertaisesti $O(3r+3p)$, jossa p

on polkujen lukumäärä, eli yksinkertaistetusti $O(r+p)$. Pino ei myöskään ole koskaan aivan täynnä, koska sieltä otetaan jatkuvasti alkioita pois. Yksinkertaistettu muoto on siis lähempänä totuutta.

3. Suorituskyky- ja O-analyysivertailu

Suorituskykytesteissä Dungeon divider suoriutuu selvästi paremmin, kun tarkastellaan normaalikokoista 200 * 200 luolastoa. Dungeon divider suoriutui tehtävästä 7 jaolla 3-6:lla millisekunnilla ja 8 jaolla 4-8 millisekunnilla, kun taas Dungeon minerillä vaihtelu oli luokkaa 11-70 millisekuntia. Vasta puhuttaessa 2000 * 2000 kokoisesta luolastosta Dungeon divider alkoi hidastua lähes Dungeon minerin tasolle, jolloin Dividerillä vaihtelu oli suunnilleen välillä 150-200 millisekuntia 15 jaolla ja minerillä 180-260 millisekuntia. Näin isot luolastot eivät kuitenkaan ole hyödyllisiä algoritmien käytön kannalta.

4. Työn mahdolliset puutteet ja parannusehdotukset

Tämänkaltaisessa työssä on parannusmahdollisuuksia käytännössä loputtomasti. Dungeon divideriä voisi kuitenkin hioa vielä selkeästi käytännöllisempään suuntaan, sillä tällä hetkellä sen luomat huoneet ovat välillä oudon pitkiä tai leveitä, sekä huoneiden välille luodut polut voivat joissain tapauksissa lävistää jonkin toisen huoneen. Tämä ei välttämättä ole ollenkaan huono asia, mutta ei aina toivottavaa.

Dungeon miner puolestaan tarvisi puolestaan enemmän säätövaraa siihen, kuinka paljon luodaan polkuja jotka eivät johda mihinkään. Tällä hetkellä myöskään polut eivät voi viedä johonkin jo aikaisemmin luotuun huoneeseen, joka rajoittaa luolaston linkittyvyyttä huomattavasti.

5. Lähteet

Kummatkin algoritmit perustuivat alla esiteltyihin ideoihin, vaikka poikkeavat niistä joissain määrin erityisesti tehokkuussyistä.

Dungeon divider: http://roguebasin.roguelikedevelopment.org/index.php/Basic_BSP_Dungeon_generation
Dungeon miner: http://roguebasin.roguelikedevelopment.org/index.php/Dungeon-Building_Algorithm

Varsinaisesti muita lähteitä ei tullut käytettyä lukuunottamatta epäsuorasti tietorakenteet-kurssin asioita sekä Javan dokumentaatiota.