Lauri Lundell

Proseduraalinen luolagenerointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

29.5.2023

Tiivistelmä

Tekijä:	Lauri Lundell
Otsikko:	Proseduraalinen luolagenerointi
Sivumäärä:	23 sivua
Aika:	29.5.2023

Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Pelisovellukset
Ohjaaja:	Lehtori Miikka Mäki-Uuro

Insinöörityössä tehtiin proseduraaliseen generointiin pohjautuva peli ja tutkittiin proseduraalista generointia yleisemmin. Työn tavoitteena oli laajentaa tekijän käsitystä proseduraalisen generoinnin eri menetelmistä. Proseduraalisen generoinnin käyttö sai alkunsa alun perin lautapeleissä, kunnes sitä alettiin 1980-luvulla käyttämään videopeleissä. Proseduraalista generointia voidaan käyttää peleissä erittäin monin tavoin. Joskus se on pelin keskeisin mekaniikka, kun taas joissain peleissä sitä saatetaan käyttää vain kehitysprosessin apuvälineenä.

Työssä kehitettiin Unity-pelimoottoria ja Fedora Linux -käyttöjärjestelmää käyttäen peli, jonka tasot ovat proseduraalisesti generoituja. Pelissä kaivellaan luolan seiniä ja etsitään reittiä alempaan kerrokseen, seuraavaan tasoon, joka on aina hieman aiempaa kerrosta suurempi. Pelialue on neliön muotoinen, pelaaja aloittaa sen keskeltä ja reunoilla on seiniä, joita ei voi tuhota. Kiviseinien seassa on malmeja, joita on sattumanvaraisesti sinne sijoiteltu.

Peliprojekti jäi suppeaksi kehitysajan rajallisuuden takia, mutta proseduraalisia elementtejä saatiin peliin silti. Pelistä puuttuu käyttöliittymä, mutta pelin rajallisuuden takia se ei koitunut kovin suureksi ongelmaksi. Peliin suunniteltiin alun perin vihollisia ja taikoja. Myös malmeille oli tarkoitus kehittää käyttöjä, mutta aikataulu ei riittänyt niiden lisäämiseen.

Avainsanat: Unity-pelimoottori, Linux, proseduraalinen, generointi

Abstract

Author:	Lauri Lundell
Title:	Procedural cave generation
Number of Pages:	23 pages
Date:	29 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Game Applications
Supervisor:	Miikka Mäki-Uuro, Senior Lecturer

This study consisted of a game based on procedural generation, and prosedural generation in games was also studied more generally. The goal of this study was to learn about different ways of using prosedural generation in game development. Procedural generation started to get popular in games in the 1980s. Sometimes it is the core mechanic of the game, but sometimes it might only be used during the game's development. Often it is something between the two extremes.

The game project was carried out using the Unity game engine and a Linux operating system, specifically its Fedora distribution. The levels of the game are procedurally generated caves in a large square area. The player's task is to dig around in the level and try to find the exit to the next level, which is a hole on the ground, found somewhere on the level. The new levels get larger each time the exit is reached. The player always starts from the center of the level, and the walls on the edges of the level are indestructible. Among regular stone wall tiles, there are ores, which are randomly spread across the level.

The game project fell short of plans and ended up very simple and limited, because of the limited development time. However, the core procedural elements can be seen in the end-product. The game is missing a user interface, but since the game is relatively simple, this was not very critical. Spells and enemies were originally planned, but they had to be dropped due to time limitations. The ores were also supposed to have a purpose in addition to visual variation, but this was also dropped in the end.

Keywords: procedural, generation, Linux, operating system, caves

Sisällys

1 Johdanto

Proseduraalinen generointi on pelin sisällön tuottamista algoritmein sen manuaalisesti käsin tuottamisen sijaan. Generoinnille annetaan ennalta määritettyjä sääntöjä, joiden mukaan prosessi tapahtuu. Proseduraalisesti voidaan luoda esimerkiksi pelikenttiä, maaston topologiaa, tekstiä tai vaikkapa hahmoja peliin.

Proseduraalinen pelin sisällön generointi on nykypäivänä todella yleistä pelikehityksen alalla, ja joskus sitä käytetään tavoilla, joita ei välttämättä edes helposti huomaa valmiista pelistä. Jotkut pelit on rakennettu täysin proseduraalisuutensa päälle, kun taas toisissa se on ollut vain työkaluna pelin kehityksessä, osana paljon suurempaa työkalupakkia. Joidenkin pelien maailmat ovat erilaisia joka pelikerralla, kun taas toisissa maailma pysyy muuten samana, mutta monet muut pelin osa-alueet saattavat muuttua pelikerralta toiselle. Proseduraalinen generointi tekee pelimaailmoista ainutlaatuisia ja sallii monien pelimaailmojen suuren koon suhteutettuna pelien kehittäjien määrään. Vaikka proseduraalinen generointi kuulostaa usein erittäin mielenkiintoiselta ja houkuttelevalta, sen käytössä on myös paljon riskejä, jotka pitää osata ottaa huomioon peliä suunniteltaessa ja tehdessä. Proseduraalisesti on helppo luoda tylsä itseään toistava maailma.

Insinöörityössä oli tarkoituksena pohtia ja kokeilla proseduraalisen kenttägeneroinnin menetelmiä, sillä se on erittäin tehokas työkalu oikein käytettynä pelikehityksessä. Insinöörityötä varten tehtiin yksinkertainen luolaseikkailupeli, jossa pelaaja ohjaa sinikaapuista velhoa kaivellen seiniä ja etsien reittiä syvemmälle maan uumeniin. Luolaston kerrokset ovat proseduraalisesti generoituja, ja reitti alempaan kerrokseen voi olla missä tahansa kerroksen kolkassa. Pelissä käytettiin joitain keinoja, joilla varsin yksinkertainen peli saatiin näyttämään hieman mielenkiintoisemmalta, kuin mitä

se on. Raportissa käydään läpi paljon erilaisia esimerkkejä proseduraalisen generoinnin käytöstä peleissä ja sen historiaa.

Työ tehtiin käyttäen Fedora Linux -käyttöjärjestelmää. Raportissa pohditaan myös pelien kehittämisen hankaluuksia ja hyötyjä Linux-käyttöjärjestelmiä käytettäessä ja sitä, miksi useammat pelaajat tai pelikehittäjät eivät käytä sitä Windowsin sijaan. Siinä myös kerrotaan, miten Unity-pelimoottorin saa toimimaan Linuxilla.

Ensin käsitellään projektin taustaa yleisemmin, minkä jälkeen keskitytään peliprojektiin. Projektin taustan yhteydessä läpikäytävät asiat ovat proseduraalisen generoinnin historia, eri tavat käyttää sen menetelmiä ja se, miten pelejä kehitetään Linux-käyttöjärjestelmällä. Peliprojektiosuudessa kerrotaan suunnitelmista, toteutuksesta ja siitä, miten projekti onnistui.

2 Proseduraalinen generointi peleissä

Luvussa käsitellään projektin taustatietoja. Ensin pohditaan eri tapoja käyttää proseduraalista generointia peleissä, minkä jälkeen käydään läpi hieman sen historiaa. Lopuksi kerrotaan erilaisia esimerkkejä toteutuksista peleissä ja sitten vielä, miten proseduraaliseen generointiin pohjautuva peliprojekti saatiin aikaan Linuxilla.

2.1 Proseduraalinen generointi pelikehityksessä

Proseduraalinen generointi peleissä auttaa luomaan paljon pelattavaa sisältöä suhteellisen pienellä vaivalla pelin kehittäjille. Muuten kaikki pelin kentät pitäisi luoda käsin, mihin kuluisi todella paljon aikaa. Proseduraalinen generointi tuo kenttiin ennalta-arvaamattomuutta, joka esimerkiksi parantaa pelin uudelleenpelattavuutta. Proseduraalisin menetelmin tehdyssä pelissä voi olla niin monenlaisia kenttiä, että yksittäinen pelaaja ei koskaan kohtaa jokaista variaatiota.

Joissain peleissä proseduraalinen generointi on erittäin keskeinen mekaniikka ja osa pelin idean ydintä jo heti sen kehityksen alkuvaiheesta. Tällaisia pelejä ovat esimerkiksi Spelunky (2008) ja Rogue (1980), eivätkä ne toimisi ilman proseduraalista generointia, sillä pelimaailman eri elementtien sattumanvaraisuus on erittäin keskeinen osa pelien ideaa. Tietynlaiset pelit vaativat proseduraalista generointia selkeästi jo idean vaatiman sisällön määrän takia. Jos ideana on luoda esimerkiksi kokonainen galaksi täynnä tähtiä ja planeettoja, kaiken käsin tekeminen ei yksinkertaisesti ole vaihtoehto. (1, s. 3-4.)

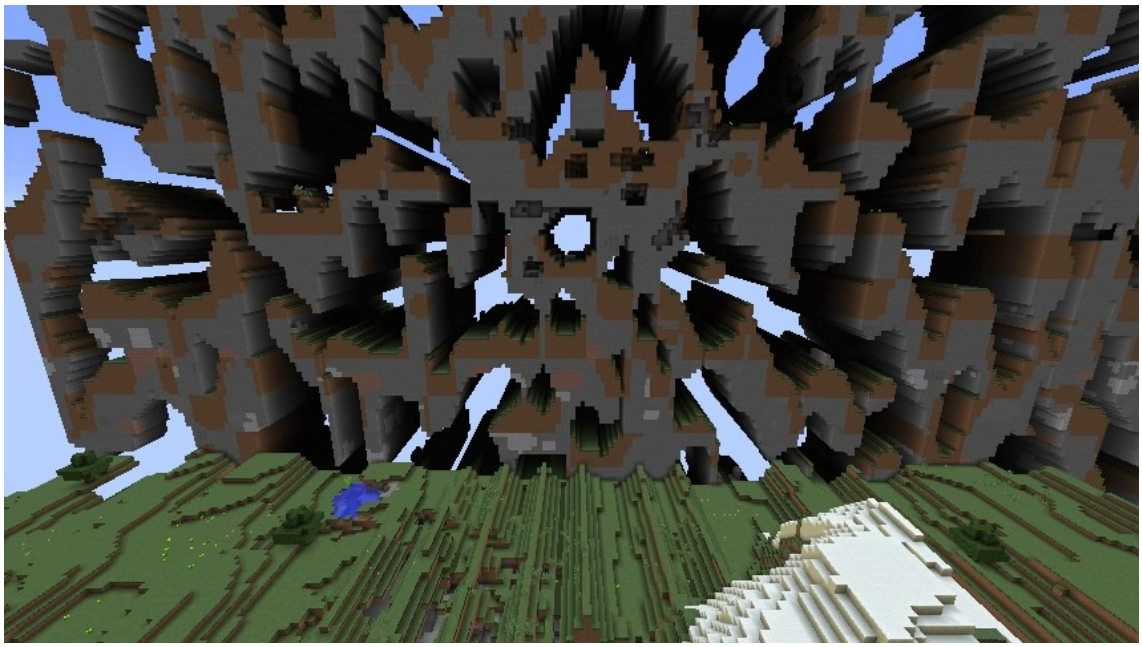
Yksi tapa käyttää proseduraalista generointia on, että sitä käytetään vain pelin kehitysvaiheessa esimerkiksi erittäin laajan maaston topologian luomiseen. Kun sattumanvaraisesti luoduista maastoista löytyy tarpeeksi mieluinen, jatketaan pelin kehittämistä sen pohjalta hienosäätäen yksityiskohtia niin pitkälle kuin on tarpeellista. The Elder Scrolls: Skyrim (2011) on yksi esimerkki proseduraalisen generoinnin käytöstä vain pelin kehitysvaiheessa. Proseduraalista generointia voi käyttää myös pulmapeleissä, jos pelin kehittäjä löytää algoritmin, jolla ratkaistavissa olevia esimerkkitasoja voidaan luoda proseduraalisesti. Näin pelin kehittäjä voi valita parhaat ja hauskimmat esimerkkitasot varsinaisiksi pelin tasoiksi. (1, s. 4.)

Joskus proseduraalista generointia käytetään vain yhdessä pelin sivupelimuodossa, mutta pääpeli on kuitenkin käsin suunniteltu. Tällaiset pelimuodot julkaistaan usein DLC:nä, (engl. downloadable content) eli pelin varsinaisen julkaisun jälkeisenä lisäosana. Näin tehtiin esimerkiksi Warhammer: Vermintide 2:ssa (2018). Toisinaan proseduraalisesti generoituja alueita voidaan yhdistää muuten käsintehtyyn tasosuunnitteluun pelin pääpelimuodossakin, kuten esimerkiksi jokin minipeli tai vaikkapa esimerkiksi jonkinlainen sattumanvarainen labyrintti vain yhdessä osiossa peliä. Jos pelin proseduraalisessa osuudessa huomataan ongelmia, voidaan se vaikka kokonaan korvata käsintehdyllä versiolla, mikäli koetaan, ettei proseduraalista algoritmia kannata lähteä enää korjaamaan vaikkapa rajallisen kehitysajan takia. (1, s. 5.)

Proseduraalisen maailmanluonnin käyttämisessä on kuitenkin riskejä, sillä algoritmit saattavat tuottaa ei-toivottuja tuloksia. Pelistä voi esimerkiksi tulla mahdoton läpäistä, jos sattumanvaraisesti asetellut elementit vaikkapa estävät pääsyn pelissä etenemisen kannalta tärkeään paikkaan. Pelin vaikeustasoa voi myös olla vaikea saada halutun tasoiseksi, mikäli mahdollisia variaatioita maailman eri elementeissä on liikaa. Proseduraalisesta generoinnista tuleekin helposti erittäin monimutkaista, varsinkin jos yritetään saada peli tietylle vaikeustasolle. Syy, miksi proseduraalista generointia ei usein käytetä AAA-tason eli suuren budjetin isojen studioiden peleissä, on se, ettei proseduraalisesti generoidun sisällön laadusta voida olla täysin varmoja. (1, s. 6.)

Proseduraalisesti generoitujen pelimaailmojen luontialgoritmien testaamiseen käytetään usein automatisoituja testejä, joilla tarkastetaan, että taso on varmasti aina läpäistävissä. Indietiimeillä ei usein ole tarpeeksi resursseja kehittää sellaisia. Indieyrityksen yksi etu on usein tosin se, ettei siltä odoteta niin paljon kuin AAA-pelinkehitysyrityksiltä. Indiepeleissä jonkinlainen keskeneräisyys on usein jopa odotettavissa, sillä pelit usein julkaistaan ennakkojulkaisuna (engl. early access). Early Access -pelit ovat pelejä, jotka on julkaistu jo keskeneräisenä, jotta pelin kehittäjä saa ansaittua tarpeeksi rahaa pelin valmiiksi saamiseen. (1, s. 6.)

Toisenlaisia ongelmia proseduraalisessa generoinnissa voivat olla vaikeasti löydettävät ohjelmointivirheet. Mojang Studiosin julkaisemassa Minecraftissa (2011) on kuuluisa Far Lands -ohjelmointivirhe, jossa n. 12 550 821 palikan päässä maailman origosta maaston generointi menee sekaisin (Kuva 1). Maaston pinta on yhtäkkiä eri korkeudella kuin lähempänä origoa, ja siinä näkyy sivusta katsoessa pesusienimäisiä reikiä. (2.)



Kuva 1. Minecraftin maailmangenerointialgoritmi menee sekaisin kaukana maailman origosta. Monta generointialgoritmin käyttämää kohinageneraattoria (engl. noise generator) lakkaa toimimasta äärimmäisen kaukana origosta. (2.)

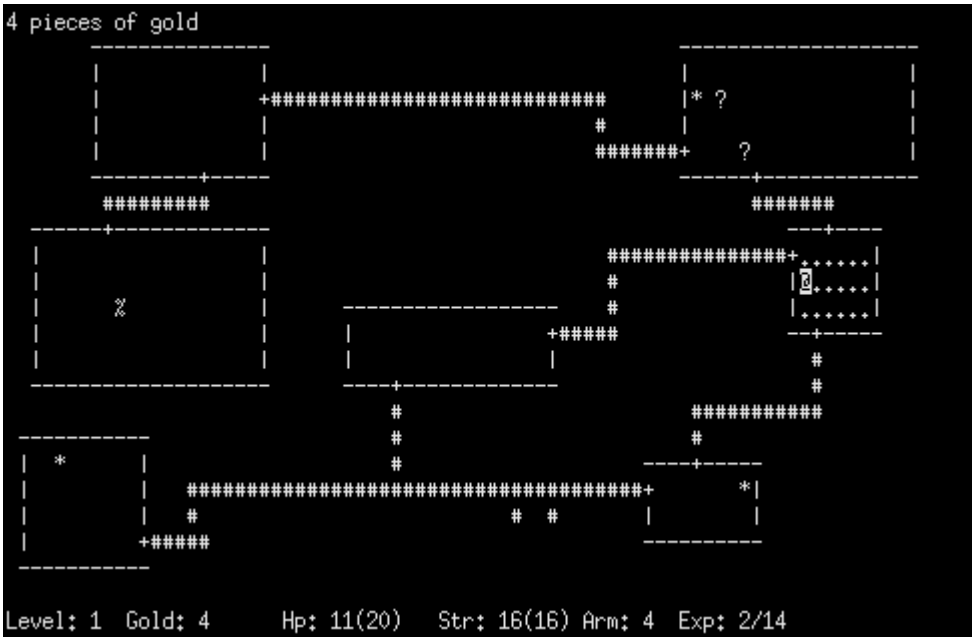
Riippuen pelin tyylistä kentät voivat olla enemmän tai vähemmän proseduraalisia. Joissain peleissä on esimerkiksi valmiiksi käsintehtyjä huoneita, jotka sitten yhdistetään jonkinlaisella algoritmilla niin, että joka pelikerralla huoneisiin saavutaan eri järjestyksessä. Eräs indiepeleissä suosittu tapa on, että lautapelimaisella kartalla on valmiiksi käsin tehtyjä kohtaamisia vihollisten kanssa ja esimerkiksi kauppoja ja muita tilanteita joihin törmätä. Ongelmana valmiiden palasten yhdistelyssä on ainakin se, että jos valmiita palasia on liian vähän, pelaaja törmää samoihin tilanteisiin usein.

On monenlaisia pelityyppejä, joihin proseduraalinen generointi ei sovi helposti. Erittäin tarinapohjaisissa peleissä sattumanvaraisuus voi olla enemmän uhka kuin mahdollisuus. Jos pelin keskeinen idea on ”speedrunnaaminen”, eli pelin mahdollisimman nopea läpäisy kilpaillen muita pelin pelaajia vastaan, proseduraaliset elementit ja sattumanvaraisuus vähentävät pelin läpäisyajan riippuvuutta pelaajan taidoista. Kilpailullisissa moninpeleissä proseduraalinen

generointi voi usein aiheuttaa pelaajien välistä voima-epätasapainoa, joka on harvoin toivottua. (1, s. 7.)

2.2 Proseduraalisen generoinnin historiaa

Proseduraalista generointia käytettiin jo lautapeleissä, esimerkiksi Advanced Dungeons & Dragonsissa (julkaistu välillä 1977-1979), jossa maastoa voitiin luoda nopanheittojen avulla. Niin kuin voikin kuvitella, tämä oli varsin aikaavievää verrattuna tietokoneiden prosessointinopeuteen. Tietokonepeleissä ensimmäisiä, jotka käyttivät proseduraalista maailmanluontia, olivat ”roguelike”-pelit, eli pelit, jotka saivat vaikutteita Rogue-nimisestä ”dungeon crawler”-lajityypin pelistä. (Kuva 2.) Rogue oli vuoropohjainen ylhäältä kuvattu roolipeli, jossa pelihahmon kuolema tarkoitti pelin kokonaan uudelleen alusta aloittamista. Roguen kentät olivat tyrmän kerroksia, jotka koostuivat käytävistä ja huoneista. Roguen kaltaisia suosittuja pelejä olivat esimerkiksi Hack (1982), NetHack (1987), The Dungeons of Moria (1983) ja Angband (1990).



Kuva 2. Roguen pelialue koostuu huoneista ja niitä yhdistävistä käytävistä (3).

Bethesda Softworksin vuonna 1996 julkaisema The Elder’s Scrolls II: Daggerfall on toimintaroolipeli, jossa on todella suuri avoin maailma. Daggerfallin noin Brittein saarten kokoinen maailma generoitiin etukäteen jo pelin kehitysvaiheessa, ja siinä oli noin 15 000 uniikkia sijaintia ja yli 4 000 tyrmää ja luolaa. Alkuinnostuksen jälkeen peliä alettiin kritisoimaan sijaintien samankaltaisuudesta, ja joissain tapauksissa jotkin tyrmät ja luolat olivat mahdottomia läpäistä, sillä generointialgoritmissa oli virheitä. (4.)

Blizzard Northin kehittämä Diablo (1997) oli erittäin suosittu toimintaroolipeli, jossa tyrmät generoitiin proseduraalisesti tiettyjä tasopohjaisia kriteerejä noudattaen. Diablon seuraajassa, Diablo II:ssa (2000) tyrmien lisäksi myös kylien ja kaupunkien ulkopuoliset vihollisia täynnä olevat erämaa-alueet on myös sattumanvaraisesti järjestelty. Jokaisen luvun erämaa-alueelta löytyy sisäänkäynti tyrmään, joka on sattumanvaraisessa paikassa. Pelaajan täytyy usein tutkia suuri osa maanpäällisestä alueesta, ennen kuin tyrmä vihdoin löytyy.

2.3 Proseduraalisen generoinnin menetelmiä

Proseduraalista maailmanluontia voidaan käyttää eri määriä erilaisissa peleissä. Joskus koko pelin kenttä generoidaan proseduraalisesti ja vielä niin, että pelin maailma on käytännössä loputtoman suuri. Tällaisiin peleihin lukeutuu esimerkiksi erittäin suosittu Mojangin julkaisema Minecraft (2011), jossa maailmaan luodaan monimutkaisella algoritmilla erittäin monimuotoisia ympäristöjä, lähes ainoana rajoituksenaan pelin ”laatikkoisuus”. Minecraftin maailmaan on ripoteltu esimerkiksi erilaisia biomeja ja maan alle erilaisia luolia ja malmeja, joista jotkut ovat harvinaisempia kuin toiset. Minecraft käyttää myös valmiiksi käsintehtyjä palasia, esimerkiksi puut, kylien rakennukset ja maanalaiset kaivokset. Myös malmiesiintymiä tarkastellessa kokenut pelaaja huomaa muutamaa erilaista valmiiksi määriteltyä muotoa. Viholliset voivat pääasiallisesti syntyä pimeisiin alueisiin, kun taas eläimiä voi syntyä mihin vain.

System Era Softworksin kehittämä ja julkaisema Astroneer (2019) seuraa proseduraalisessa generoinnissaan Minecraftin jalanjäljissä, mutta maaston tuhoutuminen on joillain tavoin vapaampaa, sillä peli ei perustu laatikkomaisiin tuhoutuviin palikoihin. Pelaaja voi kaivaa, lisätä tai tasoittaa maata maastotyökalulla, mutta rajoitusten puute vaikeuttaa varsinaisten rakennusten rakentamista, sillä seiniä on vaikea suoristaa ja erilaisia rakennusmateriaaleja ei ole. Pelissä on luotu kokonaisia planeettoja proseduraalisesti, ja pelaaja voi kulkea planeetan ympäri ja kaivaa todella syvälle. Astroneerin proseduraalinen generointi tuottaa uskottavamman näköisiä ympäristöjä kuin Minecraftissa, mutta Minecraft sallii mielenkiintoisempaa rakentamista.

Erittäin tunnettu ja Minecraftia vanhempi ja monimutkaisempi proseduraalisesti luotu pelimaailma on Bay 12 Gamesin julkaisemassa pelissä Dwarf Fortress (2006) (Kuva 3.). Ennen pelin alkua pelaaja voi maailmaa luodessaan valita sen koon ja kuinka paljon historiaa sille luodaan. Maailmalle generoidaan yleensä satojen vuosien historia, johon lukeutuu historiallisia tapahtumia, paikkoja ja henkilöitä, joille generoidaan myös nimet ja taustatarinat. Pelaaja valitsee maailman luonnin jälkeen alueen, jolla varsinainen pelaaminen lopulta tapahtuu. Koko maailma on silti simuloitu, ja peliajan kuluessa asioita tapahtuu myös muualla maailmassa. (5.)

Kuva 3. Dwarf Fortress on ulkoasultaan ja käyttöliittymältään sekava aloittelijalle. Kuvassa on monta kääpiötä maan pinnalla pelin alussa niiden saavuttua pelaajan maailmankartalta valitsemalle varsinaiselle pelialueelle. (5.)

Kaikki Dwarf Fortressin hahmot on luotu lähes täysin proseduraalisesti, niiden nimiä myöten. Hahmoilla on omat ominaiset taitonsa, oma persoonallisuutensa ja mieltymyksensä, ja ne vaikuttavat merkittävästi hahmojen toimintaan. Esimerkiksi pelissä pelattavat kääpiöt voivat tulla hulluksi kateudesta, suuttua toisille hahmoille, rakastua tai vaikkapa inspiroitua tekemään taidetta, riippuen hahmon luonteesta. Yksinkertaisista, mutta aloittelijalle hankalalukuisista ASCII-grafiikoistaan huolimatta Dwarf Fortress on laajimpia fantasiamaailmasimulaatioita, joita on koskaan tehty.

Eräs Dwarf Fortressista paljon vaikutteita saanut Ludeon Studiosin julkaisema koloniasimulaattori Rimworld (2018) tuo samankaltaista pelikokemusta, mutta paljon käyttäjäystävällisemmässä muodossa. Peli on hieman yksinkertaisempi, sillä maailmalle ei esimerkiksi luoda yhtä syvällistä historiaa.

Monissa avaruuspeleissä, joissa planeettoja ja galakseja on lukemattoman paljon, ne on usein generoitu algoritmein, sillä kokonaisten planeettojen ja tähtijärjestelmien käsin rakentaminen olisi täysin mahdotonta. Pelissä No Man’s Sky (2016) planeetoille on proseduraalisesti luotu myös uniikkeja eläinlajeja ja kasveja, tosin käyttäen valmiiksi käsin luotuja elementtejä. (Kuva 4.) Koska

pelin maailmangenerointilogiikka on täysin determinististä, universumi generoituu jokaiselle pelaajalle samalla tavalla.

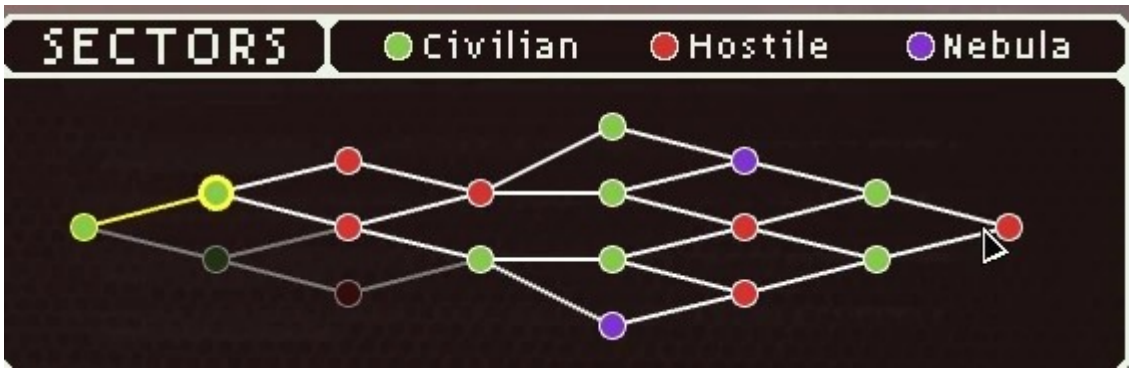
Peliin on kovakoodattu kaikille yhteinen siemenluku, jonka pohjalta monimutkaisella algoritmilla planeettoja luodaan sitä mukaan, kuin pelaaja on niitä tarpeeksi lähellä, kaikille samalla tavalla. Tämän menettelyn heikkoutena on se, että pelaajien tekemät muutokset planeettoihin eivät säily pelaajan lähdettyä kauas planeetan luota ja uudelleen lähestyttäessä samainen planeetta generoidaan jälleen uudelleen, samoin kuin ensimmäisellä kerralla.

Kuva 4. No Man’s Skyn planeetat ja niillä elävät eliöt ovat proseduraalisesti generoituja (6).

Toisinaan proseduraalisesti luotuja elementtejä käytetään käsintehtyjen kokonaisuuksien yhdistelyyn. Tätä tekniikkaa on käytetty hyväksi esimerkiksi Faster than Lightissa (2012) ja Peglinissä (2021). Molemmissa peli sijoittuu lautapelimäiselle kartalle ja pelaaja voi toimillaan vaikuttaa kulkuunsa monihaaraisella kulkureitillä. Faster than Lightissa on kaksi pelikarttaa.

Sektorikartassa pelaaja voi valita, millaiseen sektoriin kulkee seuraavaksi. Jokaisen sektorin sisäisessä majakkakartassa (engl. beacon map), pelaaja valitsee reittinsä majakoita pitkin sektorin ulospääsylle ja törmää jokaisella majakalla jonkinlaiseen tapahtumaan.

Pelaajan tarkoitus on yrittää kehittää pelihahmoa Peglinissä tai avaruusalusta Faster than Lightissa mahdollisimman tehokkaasti selvitäkseen pikkuhiljaa vaikeutuvista vihollisten kohtaamisista. Haastavammat kohtaamiset yleensä palkitsevat pelaajaa paremmin, jolloin pelaaja joutuu tekemään usein vaikeita valintoja riskin ottamisessa. Faster than Lightissa pelaaja valitsee seuraavan sektorin klikkaamalla sitä (kuva 5), mutta Peglinissä seuraavan tason valinta perustuu siihen, osuuko pelaajan pallo alareunan vasempaan vai oikeaan reunaan. Peglin-pelikentällä on esteitä, jotka voivat kentästä riippuen välillä tehdä reitin valinnasta lähes sattumanvaraista.



Kuva 5. Faster than Light -pelin sattumanvaraisesti pelin alussa generoitu sektorikartta. Pelin alun ensimmäinen sektori on aina tavallista helpompi siviilisektori. (7.)

Yksi tapa käyttää proseduraalista generointia on lisäpelimuotona muuten käsintehdyssä pelissä. Hyvä esimerkki löytyy Fatsharkin julkaisemasta Warhammer: Vermintide 2:sta (2018). Neljännessä lisäosassaan peliin lisättiin pelimuoto nimeltä ”The Chaos Wastes”, jossa tavallisten tasojen alueita yhdistellään roguelikemaisessa pelimuodossa. Pelihahmon varusteiden

kehittäminen on erillistä verrattuna muihin pelimuotoihin, ja kehityspisteitä sitä varten voi saada vain pelaamalla kyseistä pelimuotoa.

Joissain peleissä itse pelin säännöissä on käytetty proseduraalista generointia. Kanadalaisen Steel Crate Gamesin julkaisemassa Keep Talking and Nobody Explodes-pelissä (2015) kaksi pelaajaa pyrkivät purkamaan pommin. Pelin juju on se, että vain toinen pelaajista näkee pommin ja toinen näkee purkuohjeet kaikkiin mahdollisiin pommin osien variaatioihin. Pommin osat ovat jokainen omia pulmiaan, joista pommin purkajan on selitettävä ohjeiden lukijalle, jotta tämä osaa katsoa ohjeista oikeita neuvoja. Pommi koostuu useasta osasta, jotka on proseduraalisesti generoitu. Peliin on kovakoodattu siemenarvo, jonka perusteella itse pulmat, mutta myös niiden ratkaisuohjeet on generoitu.

2.4 Yleisiä proseduraalisen generoinnin algoritmeja

Soluautomaatti (engl. cellular automata) on tyypillisesti kaksiulotteisella ruudukolla toimiva proseduraalisen generoinnin algoritmi, jossa jokaisella aika-askeleella ruudukon kohdat vaikuttavat toisiinsa jollain tavoin. Näin voidaan simuloida populaatioiden kasvua ja leviämistä, nesteen käyttäytymistä tai vaikkapa liikenteen sujuvuutta. Tunnettu tähän perustuva peli on John Conwayn vuonna 1970 kehittämä Game of Life. (1, s. 289.)

Luonnollisen kaltaisten maastojen luontiin voidaan käyttää fractal landscape-menetelmää, jossa maasto-meshiin lisätään iteratiivisesti pisteitä aiempien väliin päätetynlaisella painotetulla satunnaisuudella (8). Suuria pelimaailmoja kehittäessä maastosta voidaan generoida proseduraarisesti ehdokkaita tämänkaltaisin menetelmin, joista sitten valitaan mieluisin pohjaksi jatkokehitykselle.

Esimerkiksi luonnollisten näköisten kasvien generointiin soveltuu L-systeemialgoritmi. Siinä kasvatetaan esimerkiksi kasvia lisäämällä siihen rekursiivisesti osia, kunnes todetaan että se on tarpeeksi monimutkainen.

Algoritmin avulla voidaan luoda myös esimerkiksi kaupunkeja, haarautuvia jokia tai haarautuvia teitä. (1, s. 283.)

2.5 Linux pelikehityksessä

Pelien kehitys on vielä nykyään hieman vaikeampaa Linux-käyttöjärjestelmää käyttäen verrattuna Windowsiin, jos tarkoituksena on käyttää jotain suosituimmista pelimoottoreista. Jos peliä alettaisiin tekemään aivan alusta pelimoottoria myöten, esimerkiksi OpenGL:llä (9) tai Vulkanilla (10), Linuxilla se ei olisi oikeastaan yhtään sen vaikeampaa kuin Windowsilla. Lähes ainoa rajoitus esimerkiksi C++:aa tai C#:a käyttäessä on, ettei Microsoftin kehittämä Visual Studio (11) ole saatavilla Linuxille. Suosituimmat pelien kehitysympäristöt ja pelimoottorit on suunniteltu ensisijaisesti Windowsille, jolloin jokaisen uuden ohjelmistoversion toimivuus Linuxilla on yleensä vähemmän testattua. Tämä johtuu esimerkiksi siitä, että valtaosa pelien kehittäjistä ja niiden pelaajista käyttää Windowsia, minkä takia pelimoottorien kehittäjät eivät näe niin suurta syytä panostaa Linux-kehityksen toimivuuden ja sulavuuden varmistamiseen.

Linux-pelaamista on viime vuosina huomattavasti edistänyt Valven kehittämä Wine-pohjainen (12) Proton-yhteensopivuuskerros (13), jota käyttämällä suurin osa vain Windowsille suunnitelluista peleistä toimii myös Linuxilla, yleensä suunnilleen yhtä hyvin kuin alkuperäisellä alustallaan Windowsilla. Wine on Windows-rajapinnan avoin toteutus, jonka avulla Windows-rajapinnan komennot käännetään Linux-käyttöjärjestelmärajapinnan kutsuiksi. Suurin osa Windows-sovelluksista siten toimii Linuxilla lähes saumattomasti. Valve on paikkaillut Winen Proton-versiossaan Winen ongelmakohtia, jotka ovat estäneet lukemattomia pelejä toimimasta hyvin Linuxilla.

Suurin rajoittava tekijä nykypäivänä pelien yhteensopivuudessa Linuxin kanssa ovat huijauksenesto-ohjelmat tai niiden väärä konfigurointi. Esimerkiksi paljon käytetty BattlEye on nykypäivänä yhteensopiva Linuxin kanssa, ja esimerkiksi Squad (2020) ja Dayz (2018) toimivat Linuxilla erittäin hyvin. Vaikka Escape from Tarkov (2017) käyttää BattlEyeta huijauksenestonaan, se ei silti toimi

Linuxilla, sillä pelin kehittäjät eivät ole halunneet pyytää BattlEyeta sallimaan pelinsä pelaamista Linux-käyttöjärjestelmillä. (14.) Jotkut pelien kehittäjät käytävät erittäin tunkeutuvia huijauksenesto-ohjelmia, kuten esimerkiksi Riot Gamesin julkaisema Valorant (2020). Huijauksenestonmuokkaaminen Linuxin kanssa yhtensopivaksi voi olla todella iso urakka, joka ei ole kaiken työn arvoista vain suhteellisen pienen potentiaalisen pelaajakunnan saavuttamiseksi.

Linuxin käyttö on viime aikoina lisääntynyt myös pelaajien keskuudessa. Protondb, Linux-pelaajien luoma pelien yhteensopivuusraportointisivusto, kertoo, että suosituimman pelialustan Steamin sadasta pelatuimmasta pelistä enää vain 10 ei toimi Linuxilla. Valven julkaisema Steam Deck -käsikonsoli käyttää käyttöjärjestelmänään Valven kehittämää Arch Linux -pohjaista SteamOS:aa, mikä on saanut monia pelinkehittäjiä ottamaan Linuxin paremmin huomioon pelejä tehdessään.

3 Luolaseikkailupeli

Luvussa käydään läpi proseduraaliseen generointiin pohjautuvan pelin suunnittelua ja kehitystä ja vedetään kehitysprosessista lopuksi johtopäätöksiä. Aluksi käydään läpi pelin kehittykseen käytetyt ohjelmistot.

3.1 Luolaseikkailupelin kehityksessä käytetyt ohjelmistot

Unity-pelimoottori valittiin insinöörityöprojektiin siksi, että tekijällä oli sen käyttämisestä eniten kokemusta aiempien opiskeluprojektien myötä. Unreal Engine 4 on ollut viime aikoina paljon enemmän käytössä töiden takia, mutta koska peli-idea oli selkeästi kaksiulotteinen, olisi Unreal Engine ollut epäideaali. Godot on myös suosittu pelimoottori, nimenomaan kaksiulotteisia pelejä varten, mutta koska siitä ei ollut lainkaan kokemusta, sitä ei kannattanut tähän projektiin valita.

Linux-jakeluversiona eli distribuutiona käytettiin Fedora Linux -käyttöjärjestelmää (15), joka on suhteellisen suosittu yritysten keskuudessa, vaikkakin harvinaisempi kuin esimerkiksi Ubuntu Linux. Fedora valittiin suurimmaksi osaksi siksi, että se oli valmiiksi asennettuna. Nykypäivänä Linux-jakeluversiolla ei ole kovin paljon väliä lähes missään yleisessä käyttötarkoituksessa, ja projektia varten olisi kelvannut lähes mikä tahansa aktiivisessa kehityksessä oleva jakeluversio.

Fedora on yleisesti hyvä siitä, että vaikka se tarjoaa todella uusia versioita ohjelmista, ne on silti yleensä testattu hyvin ennen julkaisua. Kontrastina voidaan tarkastella esimerkiksi Arch Linuxia, joka tarjoaa niin uusia versioita paketeista, ettei järjestelmän vakauteen voi luottaa tarpeeksi, jos kyseisellä tietokoneella on tarkoitus tehdä tärkeitä asioita. Toisessa ääripäässä on sitten Debian, jonka monet ohjelmat ovat usein vuosia vanhoja. Tämä tekee Debianista erittäin stabiilin, mutta uudempien ohjelmien asennus voi olla hankalampaa.

Fedoralla Unity-kehitysympäristön asentaminen on hieman joitain muita Linux-käyttöjärjestelmiä vaikeampaa, sillä Unityllä ei ole virallista pakettia Fedoran käyttämässä RPM-pakettihallintajärjestelmässä. Onneksi Unity tarjoaa RPM-pakettiarkiston, joka sisältää paketin Unity Hub -sovellukselle. Unity mainostaa kyseisen arkiston sopivan Red Hat Enterprise Linuxille tai CentOS:lle, mikä voi aloittelijalle olla hämäävää. Arkisto on kuitenkin ladattavissa Fedora Linuxille ja muille RPM-pohjaisille Linux-jakeluille, kuten OpenSUSE:lle. Unity tarjoaa myös Debianille ja Ubuntulle pakettikirjaston, joka ei ole yhteensopiva RPM-pohjaisten pakettihallintajärjestelmien kanssa, sillä Debian ja Ubuntu käyttävät paketinhallintajärjestelmänään APT:ia. Unity tarjoaa sivullaan beetaversioita Unity-hub sovelluksesta, mutta tätä projektia varten käytettiin tavallista versiota. Esimerkkikoodi 1 sisältää komennon, jolla Unityn pakettikirjasto saadaan lisättyä Fedoralle.

```
sudo sh -c 'echo -e "[unityhub]\nname=Unity
Hub\nbaseurl=https://hub.unity3d.com/linux/repos/rpm/stable\
nenabled=1\ngpgcheck=1\ngpgkey=https://hub.unity3d.com/linux/repos/
```

```
rpm/stable/repodata/repomd.xml.key\nrepo_gpgcheck=1" >
/etc/yum.repos.d/unityhub.repo'
```

Esimerkkikoodi 1: Komentoriville kirjoitettuna RPM-pakettihallintaa käyttävällä Linuxin jakeluversiolla yllä oleva komento kirjoittaa ”unityhub.repo”-tiedostoon RPM-yhteensopivalle pakettihallintaohjelmalle tarpeelliset tiedot Unityn tarjoamien pakettien asentamiseen ja ylläpitoon (16).

Esimerkkikoodin 1 komennon jälkeen unityhub-paketti voidaan asentaa komennolla ”sudo dnf update && sudo dnf install unityhub”. Paketinhallintajärjestelmällä koko järjestelmän paketit voidaan päivittää kerralla ”sudo dnf update” -komennolla, eikä esimerkiksi Unity Hubin päivittämisestä tarvitse enää huolehtia erikseen.

Insinöörityöprojektin kehitykseen käytettiin Unity-pelimoottorin versiota 2021.3.22f1, joka on niin sanotun ”pitkän ajan tuen” versio (engl. long term support, LTS). Suurin ongelma kyseisessä Unityn versiossa Linuxia käyttäessä oli, että kehitysympäristön ”Play In Editor”-tila aiheutti lähes joka toinen kerta sitä käynnistäessä koko Unity-ohjelmiston kaatumisen, jolloin ylimääräistä aikaa kului sen uudelleenkäynnistelyyn.

Pelikehitykseen Unity vaatii rinnalleen C#-koodin muokkaukseen soveltuvan tekstinmuokkaussovelluksen. Teoriassa lähes mikä tahansa tekstinkäsittelyohjelma toimii, mutta koodaamista helpottaa huomattavasti, jos tekstinkäsittelyohjelma osaa antaa vinkkejä koodin oikeinkirjoitukseen ja kielen syntaksin tarkistukseen. Yleensä helpoin koodinmuokkausohjelma Unityä käyttäessä on Microsoftin Visual Studio, mutta Linuxille sitä ei ole saatavilla. Ilmaisista vaihtoehdoista kätevimmäksi projektia varten todettiin Microsoftin Visual Studio Code, joka sisältää lisäosan C#-ohjelmointikielelle, jota Unreal tukee. Maksullinen Jetbrains Rider -koodinmuokkaussovellus on myös erittäin hyvä, mutta Visual Studio Code valittiin tällä kertaa sen maksuttomuuden ja tarpeeksi hyvän C# tuen takia.

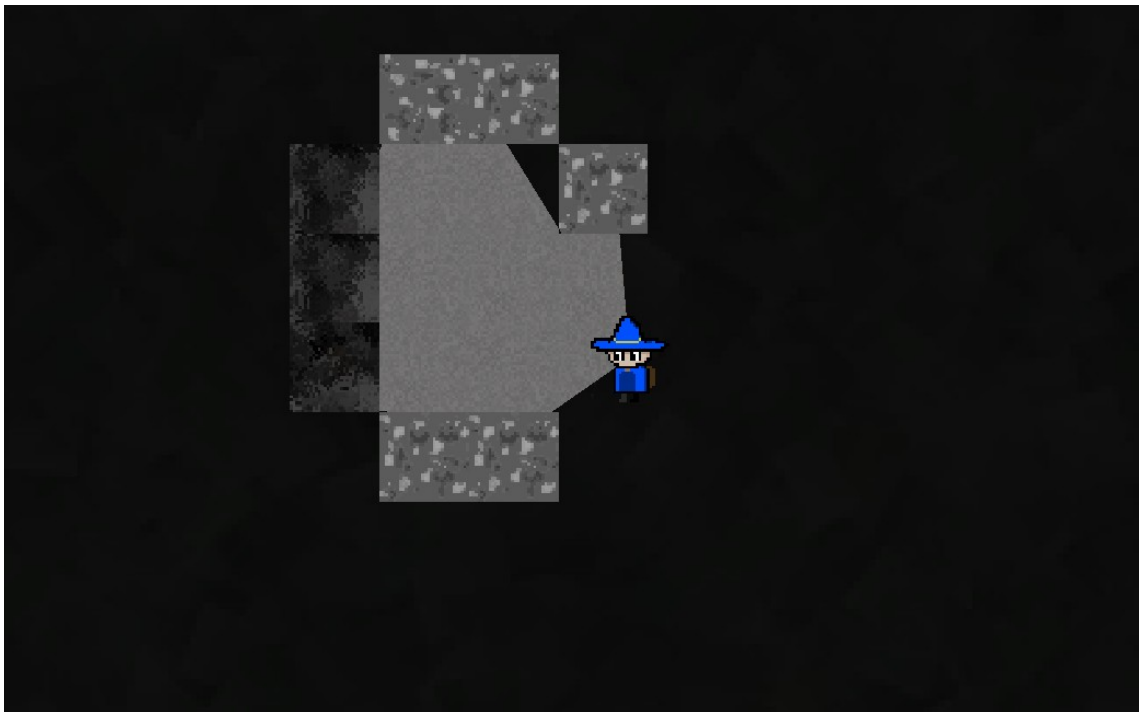
Peli tarvitsee tekstuureja, ja niiden piirtämistä varten käytettiin Gimp-kuvankäsittelyohjelmaa, koko nimeltään GNU Image Manipulation Program. Pikselitaiteen piirtämiseen kävisi lähes mikä kuvankäsittelyohjelma, mutta

esimerkiksi Gimpin tarjoama alfakanava on erittäin tarpeellinen tekstuurien läpinäkyvyyden mahdollistamiseksi. Gimpin tarjoamat suodattimet olivat erittäin käteviä maastotekstuurien luontiin, sillä niihin on hyvä luoda vähän sattumanvaraisuutta. Linuxille ei ole saatavilla esimerkiksi Adobe Photoshopia, ja se on muutenkin maksullinen. Gimpin ensimmäinen versio 0.54 julkaistiin helmikuussa vuonna 1996.

3.2 Luolaseikkailupelin idea

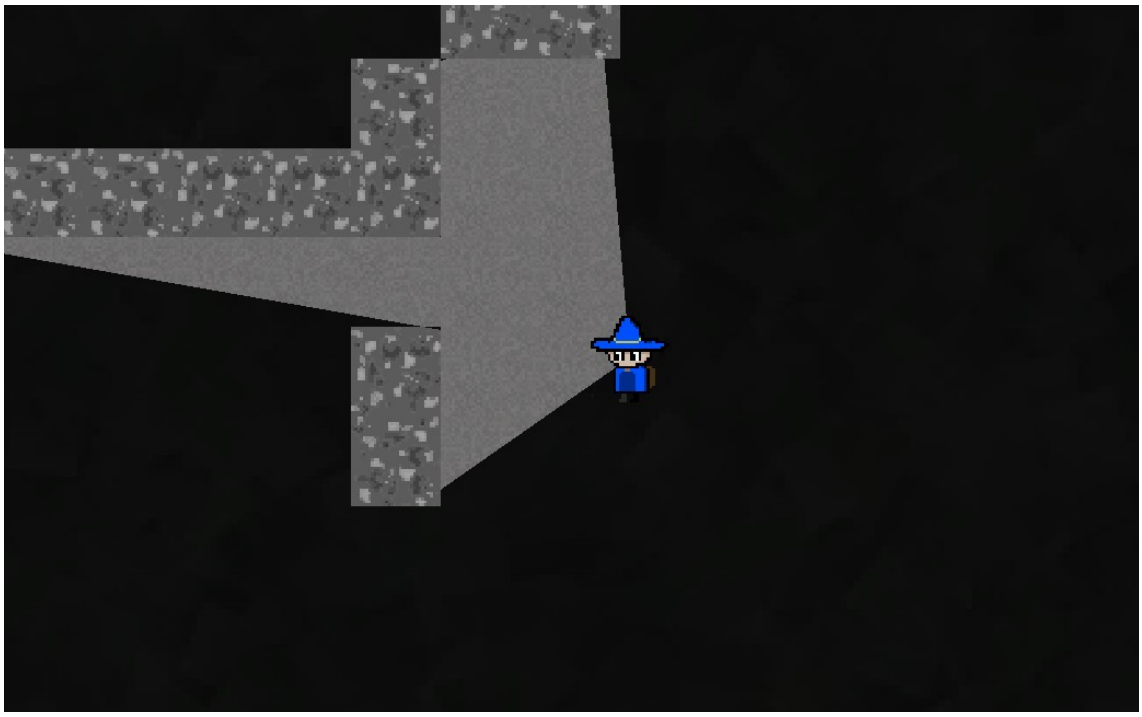
Luolaseikkaikupelin tarkoituksena oli kokeilla pelimaailman luontia proseduraalisin keinoin, ja se toimi pelin keskeisenä mekaniikkana jo ihan kehityksen alusta alkaen. Pelin alkuperäinen idea oli rakentaa peli resurssien kaivelun ja niistä asioiden rakentelun pohjalle. Tarkoituksena oli myös lisätä peliin muutamanlainen taika-ase ja yksi tai kaksi eri vihollistyyppiä.

Pelihahmona toimii sinikaapuinen velho, joka kykenee kävelemään, juoksemaan ja tuhoamaan neliön muotoisia seinäpalasia. Pelin idea on edetä proseduraalisesti luoduissa maanalaisissa kerroksissa alaspäin niin, että jokainen matalampi kerros on edellistä isompi ja vaikeampi. Pelaajan pitää löytää kerrokseen piilotettu reikä alempaan kerrokseen. Pelaaja aloittaa maanalaisen alueen keskeltä ja pystyy kaivelemaan laatikkomaisia seinäpalasia, kunnes törmää kentän tuhoutumattomiin reunoihin (kuva 6), joita pelaaja ei voi enää kaivaa. Uusia kenttiä tulee loputtomasti pelin edetessä, eikä sitä voi varsinaisesti voittaa.



Kuva 6. Pelialueen tuhoutumattomat sivuseinät ovat muita tummempia laattoja, joita klikkaamalla mitään ei tapahdu.

Pelaajalla on näkökenttä (kuva 7), jonka ulkopuolisia seiniä pelaaja ei näe. Seinät myös estävät näkyvyyttä, eli pelaaja näkee vain sen, mitä itse pelihahmokin voisi nähdä. Pelialueella on tavallisia seiniä ja erilaisia malmeja, joita on ripoteltu seinien sekaan niin sanottuihin ryppäisiin. Pelissä on tällä hetkellä hiiltä, rautaa ja kuparia tavallisten kiviseinien lisäksi. Seinien tuhoamiseen tarvitaan vain klikkaus näkyvään seinäpalaseen, jolloin palanen katoaa ja paljastaa lattiapalasen altaan. Näkökenttä paljastaa myös kentän maalin, reiän lattiassa, joka johtaa seuraavaan tasoon. Pelaajan osuessa maaliin pelaaja siirretään heti uuteen tasoon, sen origoon.



Kuva 7. Pelaajan näkökenttä, joka seuraa hiiren kohdistinta. Pelaajan näkökenttä on 120 astetta leveä.

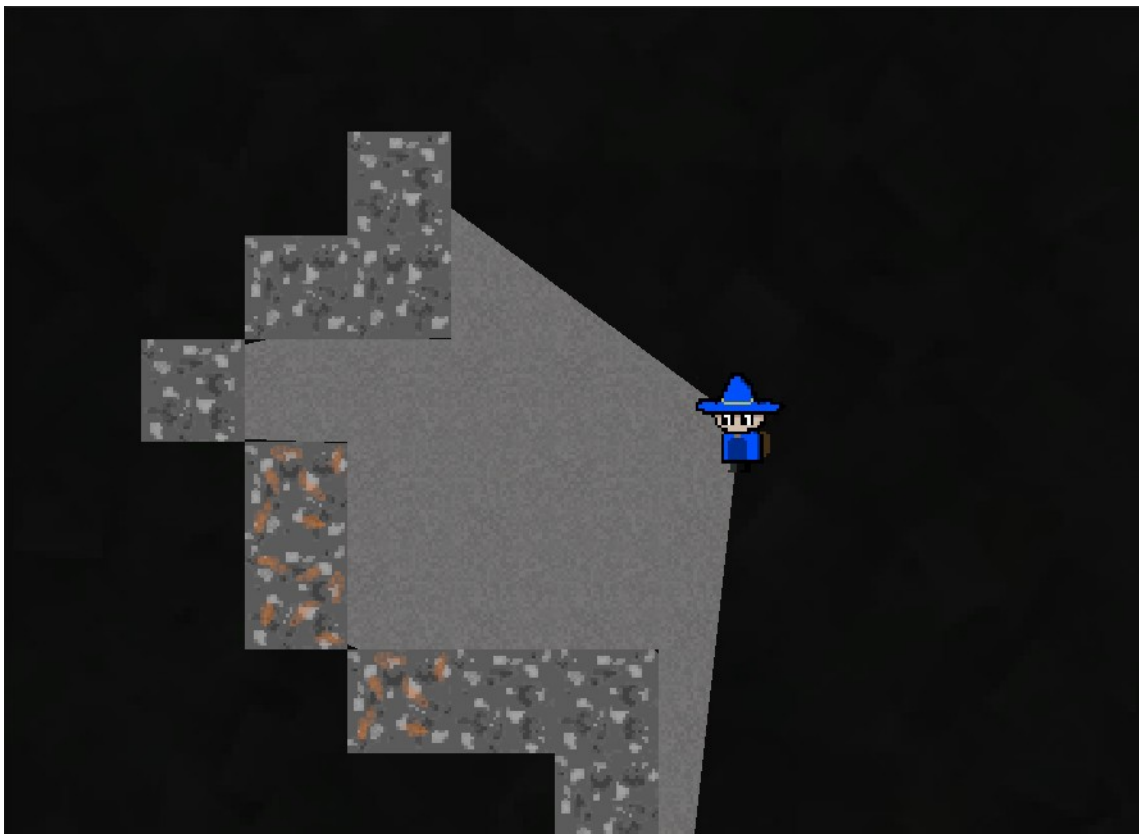
3.3 Luolaseikkailupelin toteutus

Luolaseikkailupelin kehityksen keskeisin elementti oli proseduraalisesti generoidut luolat ja seinät. Peli rakennettiin yhteen Unityn ”sceneen” eli ympäristöön, ja seuraavat tasot ladataan aiemman poistamisen jälkeen samaan paikkaan. Generointi koodattiin siten, että ensin luotiin kaksiulotteinen lista, jonka luolagenerointialgoritmi täyttää eri laattatyypeillä. Listan täyttämisen jälkeen itse varsinaiset seinät ja lattiat luodaan ja pelaaja siirretään kentän keskelle. Kaikki satunnaisuus perustuu yhteen siemenmerkkijonoon, ja samalla siemenarvolla saadaan aina samanlainen kenttä, jos kentän koko pysyy samana. Pelialueen keskelle algoritmi varmistaa tarpeeksi tyhjää tilaa, ettei pelaaja keskeltä aloittaessaan aloita seinien sisältä.

Pelin ympäristöön luotiin kätevästi variaatiota siten, että jokaisen laatan orientaatio on satunnaistettu, jolloin maasto ei näytä toistavan itseään

läheskään niin pahasti kuin voisi. Tietyntyyppiset laatat valitaan myös listasta sen variaatioita, joten pelissä oleva järjestelmä sallii jo sen, että jo esimerkiksi kolmella kiviseinävariaatiolla saadaan 12 erinäköistä seinää ja viidellä jo 20.

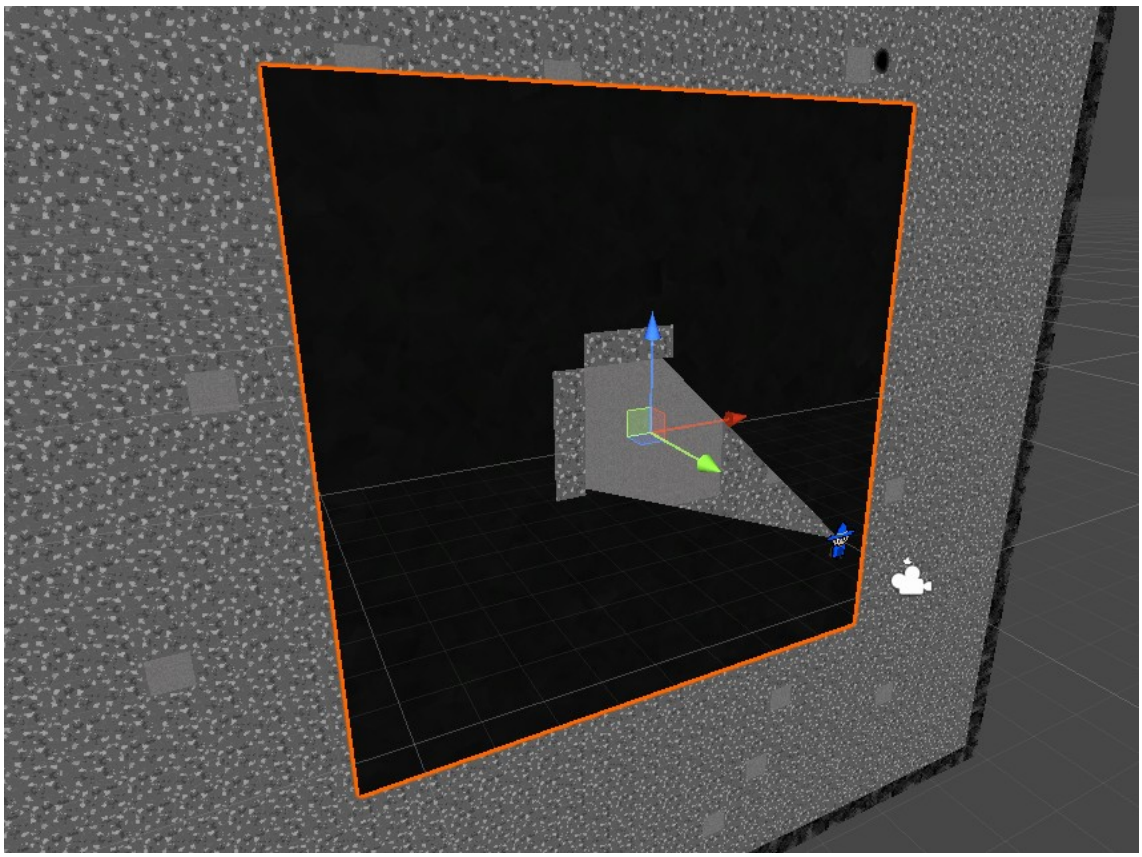
Malmiryppäät (kuva 8) toteutettiin niin, että maaston generoinnin yhteydessä osa kiviseinälaatoista vaihdettiin malmilaatoiksi, ja varsinaisen generoinnin jälkeen jo paikalla olevien malmilaattojen ympärille laitettiin uusia, muutaman kerran kyseistä prosessia iteroiden. Jokaisella iteraatiolla naapurilaatan lisäämistodennäköisyys laskee lineaarisesti.



Kuva 8. Kuparimalmia seinässä. Malmin ympäriltä on kaiveltu pois seiniä, jotta enemmän malmilaattoja on näkyvissä.

Vaikka pelissä kaikki näyttää kaksiulotteiselta, peli sijoittuu oikeasti kolmiulotteiseen maailmaan. Pelihahmon ja maastolaattojen välissä on suuri, kameran kuvaa suurempi musta levy (kuva 9), joka kuvaa aluetta, jota ei nähdä. Näkökenttä sallii näkemisen mustan levyn läpi, jolloin sen alta paljastuu lattiaa ja seiniä. Näkökentässä olevia seinälaattoja siirretään hetkellisesti lähemmäs

kameraa, ja ne palaavat takaisin mustan levyn taakse piiloon, kun niitä päin ei enää katsota. Ilman tätä näkökentässä olevat laatat estäisivät näkyvyyttä ja niiden omat tekstuurit olisivat mustan peitossa. Näkökenttä-meshiä päivitetään pelin jokaisella logiikkapäivityksellä (engl. tick) käyttäen kaksiulotteisia säteensuuntauksia (engl. raycast).



Kuva 9. Pelaajan näkökenttä on pelaajahahmon ja pelimaailman välissä oleva mesh eli kappale, jonka varjostin (engl. shader) antaa nähdä tumman näköesteen läpi.

3.4 Toteutuksen arviointi

Luolaseikkailupeli tehtiin lyhyessä ajassa, ja siinä on monia ongelmia. Pelin rajallisen kehitysajan sallima laajuus yliarvioitiin pahasti, ja moneen tavoitteeseen ei päästy. Peliin ei ehditty lisätä yhtään käyttöliittymän elementtejä, mutta monien mekaniikkojen puuttuessa se ei lopulta haitannut

kovasti. Peli ei ole kovin mielenkiintoinen pelata, sillä siitä uupuu paljon mahdollisia mielenkiintoa luovia elementtejä, kuten viholliset, taiat ja crafting-järjestelmä, joka olisi perustunut malmien keräämiseen ja niistä resurssien irrottamiseen. Pelin maailman generointi jäi varsin suppeaksi, ja kentissä olisi voinut olla enemmän luolia ja käytäviä, lähes täysin täytetyn alueen sijaan. Malmien kaivamiseen ei ehditty lisätä mitään resurssienkeräämismekaniikkaa, eikä inventory-järjestelmää. Tavaratila (engl. inventory) olisi ollut myös kriittinen crafting-järjestelmää varten. Malmien keräämisestä ei tällä hetkellä saa edes pisteitä, joten niiden lisääminen jäi lähinnä esteettiseksi valinnaksi.

Malmiryppäät toteutettiin projektissa iteratiivisesti lisäämällä malmilaattoja jo kentällä olevien malmilaattojen viereen, jokaisella iteraatiolla pienemmällä todennäköisyydellä. Jälkikäteen ajateltuna järkevämpää olisi ollut käyttää Minecraftin käyttämän menetelmän kaltaista tapaa, eli jokaiselle malmityypille olisi tehty valmiiksi vaikkapa viisi eri malmirypäsvariaatiota, joista sitten sattumanvaraisesti malmiksi valittujen laattojen ympärille asetettaisiin ryppään muut malmilaatat. Näin säästyttäisiin turhalta iteroinnilta, sillä malmiryppään muodon satunnaisuus ei ole kovin tärkeää.

Monien mekaniikkojen puuttuminen johti siihen, että pelin ainoa idea on vain etsiä reitti alempaan kerrokseen. Ongelmana on, että seinälaattojen määrän takia reitti voi olla täysin piilossa yhden laatan kokoisella lattialaatalla, täysin seinälaattojen ympäröimänä. Tämän myötä reiän etsiminen johtaa herkästi siihen, että pelaaja alkaa vain järjestelmällisesti kaivelemaan seinälaattoja pois, kunnes löytää kuopan. Tällainen pelin rakenne ei ole kovin mielenkiintoinen pelaajalle, joten peliin nopeasti kyllästyminen on erittäin todennäköistä.

4 Yhteenveto

Insinöörityössä pyrittiin tarkastelemaan proseduraalisen generoinnin eri menetelmiä tekemällä peli sen pohjalta ja tutkimalla, miten sitä on käytetty muissa peleissä. Proseduraalisella sisällön generoinnilla voidaan luoda paljon

sisältöä peliin pienemmällä vaivalla, kuin mitä vaadittaisiin sen itse taiteilemiseen. Proseduraalisen generoinnin käytön haittapuolena ovat esimerkiksi epähalutut generointitulokset, kuten kenttä, joka on mahdoton läpäistä. Proseduraalisen generoinnin käytön keskeisyys vaihtelee pelien välillä. Joskus se on pelin keskeisin mekaniikka, kun taas joissain peleissä sitä saatetaan käyttää apuna vain kehitysvaiheessa.

Työssä oli tavoitteena saada aikaan lähes kokonainen toimiva peli, jonka olisi proseduraalisen kenttänsä lisäksi tarkoitus olla hauska pelattava. Peliprojekti jäi suppeaksi, mutta siihen saatiin silti proseduraalinen maailmangenerointi pohjalle, joten sen osalta projekti onnistui. Pelin mielekkyyteen jäi kuitenkin paljon parantamisen varaa, ja pelaaja todennäköisesti kyllästyy nopeasti pelin sisällön ja vaihtelun puutteeseen. Peliä voitaisiin käyttää pohjana paljon monimutkaisempaan peliprojektiin, jolloin sen pelattavuus ja hauskuus voitaisiin säätää kohdalleen.

Lähteet

1. Short, Tanya & Adams, Tarn. 2017. Procedural Generation in Game Design. Taylor & Francis Group.

2 Burkett, Jacob. What caused the Minecraft Far Lands to happen? Verkkoaineisto. <https://www.sportskeeda.com/minecraft/what-caused-minecraft-s-far-lands-generate>. 11.5.2022. Luettu 17.5.2023.

3 Rogue. 1980. Mastertronic.

4 McMullen, Chris. 2022. What The Elder's Scrolls VI Needs to Learn from Daggerfall. Verkkoaineisto. <https://www.escapistmagazine.com/the-elder-scrolls-ii-daggerfall-procedural-generation-lessons-tes-vi-bethesda/>. 12.4.2022. Luettu 27.4.2023.

5 Dwarf Fortress. 2006. Kitfox Games.

6 Parrish, Ash. 2021 Verkkoaineisto. The Verge. <https://www.theverge.com/2021/10/20/22736219/no-mans-sky-new-expedition-emergence-hello-games>. 20.8.2021. Luettu 17.5.2023.

7 Sector. Verkkoaineisto. Faster Than Light Wiki. <https://ftl.fandom.com/wiki/Sector>. Luettu 28.4.2023.

8 Hatzis, Alex & Lin, Nicole & Bradford, Katie. 2020. Fractal Landscapes. Verkkoaineisto. Cornell University. <https://people.ece.cornell.edu/land/courses/ece5760/FinalProjects/f2020/keb278_ajh322_nl392/keb278_ajh322_nl392/keb278_ajh322_nl392.html>. Luettu 17.5.2023.

9 OpenGL. Verkkoaineisto. <https://www.opengl.org/>. Luettu 28.4.2023.

10 Vulkan. Verkkoaineisto. <https://www.vulkan.org/>. Luettu 28.4.2023.

11 Visual Studio. Verkkoaineisto. <https://visualstudio.microsoft.com/>. Luettu 28.4.2023.

12 WineHQ. Verkkoaineisto. <https://www.winehq.org/>. Luettu 28.4.2023.

13 Proton. Verkkoaineisto. <https://github.com/ValveSoftware/Proton>. Luettu 28.4.2023.

14 Desaulniers, Simon. 2022. Escape from Tarkov: Yes! GNU/Linux runs it! Verkkoaineisto. <https://sim590.github.io/en/gaming-on-linux/tarkov/>. 12.2.2022. Luettu 25.4.2023.

15 Fedora. Verkkoaineisto. <https://fedoraproject.org/>. Luettu 29.4.2023.

16 Install the Unity Hub. Verkkoaineisto. Unity Technologies. <https://docs.unity3d.com/hub/manual/InstallHub.html#install-hub-linux>. Luettu 29.4.2023.