TAMPEREEN
AMMATTIKORKEAKOULU

PROSEDURAALINEN KYBERPUNK-PELIMAAILMA

Henri Reunanen

Opinnäytetyö
Joulukuu 2017
Tietojenkäsittely
Pelituotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Pelituotanto

REUNANEN, HENRI:
Proseduraalinen kyberpunk-pelimaailma

Opinnäytetyö 30 sivua
Joulukuu 2017

Proseduraalinen sisällöntuotanto on suosittu tapa luoda enemmän sisältöä vähemmällä vaivalla varsinkin pienemmissä indie-peleissä. Proseduraalisella sisällöntuotannolla tarkoitetaan algoritmipohjaista lähestymistapaa, jossa ohjelma valmistaa peliin sisältöä, yleensä kenttiä, ihmisen puolesta. Usein haasteeksi nousee sisällön laadunhallinta, sillä käsin tehty on luotettavammin laadukasta.

Opinnäytetyössä selvitettiin, miten proseduraalisuutta on käytetty erilaisissa peliprojekteissa. Lisäksi kuvattiin, miten eri tavoin proseduraalinen kenttägenerointi suunniteltiin ja toteutettiin Neon Chrome -peliin ja mitä muuta proseduraalista sisältöä pelissä on.

Työssä keskityttiin Neon Chromen kenttäsuunnittelun tekniseen puoleen, eikä siinnä käsitelty kenttien visuaalista puolta, kuten valaistusta, lattiatekstuurien valintaa tai esineiden sijoittelua.

Ensimmäisen pelattavan version kehitys kesti noin kaksi kuukautta, mutta sitä paranneltiin ja optimoitiin projektin edetessä. Neon Chrome julkaistiin noin vuosi tämän opinnäytetyön toteutuksen jälkeen keväällä 2016.

Asiasanat: kenttäsuunnittelu, videopelit, proseduraalisuus

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree Programme in Business Information Systems
Game Development

REUNANEN, HENRI:
Procedural Cyberpunk Game World

Bachelor's thesis 30 pages
December 2017

Procedural content generation is an easy way to create more content with less effort, used especially in indie games. Most common challenge in procedural content generation is quality control, as handmade content is usually more reliably of high quality.

In this thesis, I will first give some example of how procedurality is used in other games. Then I'll demonstrate different ways of how we implemented procedural level generation in the game Neon Chrome and what other procedural content the game has.

The text focuses on the technical details of level design in Neon Chrome and I won't be discussing the visual aspects like lighting or the use of different textures and props.

My main part in the project was creating a grid based level generator. First playable version took about two months to make but further development persisted throughout the project.

Key words: level design, video games, procedural

SISÄLLYS

ERITYISSANASTO

Kenttä	Pelin alue, jolla on alku ja loppu.
Kenttäsuunnittelu	Pelin kenttien luonti
Proseduraalinen generointi	Automaation käyttö sisällön luonnissa.
Kyberpunk	Scifin alalaji, jonka pääteemana on futuristisen teknologian yhdistäminen likaiseen alamaailmaan.
Skripti	Itsenäinen lyhyt ohjelma. Peliohjelmoinnissa näitä voidaan ajaa pelimoottorin päätoimintojen ohessa.
Kenttägeneraattori	Peliin kentän luova skripti.
Squirrel	Tässä projektissa käytetty skriptikieli.
Virittävä puu	Verkko, joka yhdistää kaavion jokaisen solun ilman silmukoita.
Pienin virittävä puu	Virittävä puu, jossa on mahdollisimman vähän solujen välisiä viivoja.

1 JOHDANTO

Opinnäytetyöni toimeksianto oli kehittää kenttägeneraattori kyberpunk-teemaiseen peliin. Pelissä oli jo yksi kenttägeneraattori, mutta toimeksiantajani toivoi lisää variaatiota pelin kenttiin tuomalla uuden näkökannan projektiin.

Työni toimeksiantaja oli 10tons Oy, tamperelainen kymmenen hengen pelialan yritys vuodesta 2003. Toimenkuvani oli Neon Chrome-pelin kenttäsuunnittelu.

Proseduraalinen kenttägeneraatio on helppo tapa luoda peliin sisältöä, mutta laadunvarmistus on välillä haastavaa. Parhaassa tapauksessa pelaaja ei edes tajua pelaavansa algoritmin luomaa kenttää, mutta usein kentät tuntuvat kaikki hyvin samanlaisilta.

Tarkastelen työssäni muutamaa esimerkkiä, miten proseduraalisen kenttägeneraation voi toteuttaa. Havainnollistamiseen käytän Neon Chromen lisäksi Derek Yun kehittämää Spelunkya, sekä Phi Dinhin TinyKeepia, jotka molemmat toimivat esikuvina Neon Chromen kenttägeneroinnissa.

2 Neon Chrome

2.1. Juoni

Neon Chrome on PC:lle, Xbox Onelle ja Playstation 4:lle julkaistu ylhäältä kuvattu kyberpunk-teemainen ammuntapeli, jossa pelaajan tavoitteena on selvitä massiivisen rakennelman huipulle, ja syrjäyttää sen nykyinen johtaja. Rakennuksen nimi on Neon Chrome ja johtajaa kutsutaan Valvojaksi (engl. Overseer).

Neon Chrome -rakennus on arkologia, eli massiivinen omavarainen rakennus. Arkologia huolehtii asukkaidensa perustarpeista, kuten sähkön- ja ruoantuotannosta, itsenäisesti.

Rakennuksella on johtokunta, joka valitsee seuraavan Valvojan äänestyksellä neljän vuoden välein. Valvoja hallitsee rakennusta immersiotuolista, joka yhdistää hänet rakennuksen jokaiseen sensoriin ja järjestelmään. Pelin sivuilla Valvojan toimenkuvaa verrataan toimitusjohtajan virkaan. Pelin tapahtumat alkavat, kun senhetkinen Valvoja tulee hulluksi, julistautuu itsevaltiaaksi ja ottaa vallan täysin itselleen.

2.2. Mekaniikat

Peli on jaettu viiteen eriteemaiseen kappaleeseen, joiden lopussa on aina loppuvastus. Kappaleiden teemat ovat laboratorio, toimisto, varasto, asuinalue ja sairaala. Jokaisessa kappaleessa on 3─7 kenttää, ja ne luodaan proseduraalisesti, eli sisältöä on käytännössä loputtomasti ja jokaisen läpipeluukerran pitäisi olla erilainen.

Kentistä löytyy uusia, tehokkaampia aseita, sekä kyberpunkista tuttuja augmentaatioita, kuten hahmon lyöntivoimaa lisäävä metallinen luuimplantti. Peli pitää tallessa tiedon pelaajan parhaan löytämän aseen tasosta, jota mitataan asteikolla 1—25. Aselaatikon avaaminen antaa aina vähintään yhdellä tasolla korkeamman aseen. Asepäivityksen

avaaminen nostaa pelaajan senhetkisen aseen tasoa yhdellä. Aseen taso ei säily kuoleman jälkeen.

Pelin ympäristö on myös enimmäkseen tuhottavissa, mutta kenttien ulkoseinät ovat tuhoutumattomia. Kentän tuhoutuvuus mahdollistaa vaihtoehtoisten lähestymistapojen käyttämistä. Esimerkiksi huoneen ovea vahtivan vihollisen voi yllättää seinän läpi.

Uusia vihollistyyppejä ilmestyy pelin edetessä keskimäärin kolmen kentän välein. Tyypit voidaan jaotella ihmisiin tai ihmisen kaltaisiin ja robotteihin. Eri tyypeillä on erilaisia tekoälyjä ja käyttäytymismalleja. Ihmiset saattavat esimerkiksi kuulla pelaajan ammuskelun ja lähtevät tutkimaan mekkalaa, kun taas robotit eivät koskaan ”unohda” pelaajaa ja lopeta pelaajan jahtaamista.

Kuoleminen vie pelaajan takaisin alkuun, mutta yritysten välissä voi käyttää pelin valuuttaa parantaakseen pysyvästi hahmon kykyjä, esimerkiksi aseiden tehoa ja hahmon osumapisteitä. Loppuvastuksen voittaminen avaa pelaajalle mahdollisuuden jatkaa kyseisen vastuksen jälkeisestä kentästä, joka vähentää kenttien määrää. Tämä kuitenkin vaikeuttaa peliä lievästi, sillä myöhemmät kentät ovat vaikeampia, eikä pelaajalla ole aikaisemmista kentistä saatuja aseita tai augmentaatioita.

Pelin läpi päästyään pelaaja avaa aina seuraavan vaikeusasteen, joita on käytännössä loputtomasti. Vaikeusasteen kasvu nostaa vihollisten määrää, nopeutta, vahingontuottoa ja osumapisteitä sekä aarteiden määrää.

2.3. Teknologia

Peli käyttää 10tons Oy:n omaa pelimoottoria, joka on kirjoitettu C++:lla. Squirrel-ohjelmointikielellä luodut skriptit hoitavat kenttägeneraation. Lisäksi peli käyttää Lua-ohjelmointikieltä käyttöliittymäelementtien sijoitteluun, sekä xml-tiedostoja esimerkiksi kenttien, vihollistyyppien ja aseiden tilastojen tallentamiseen.

Kentät koostuvat 16x16 pikselin tiilistä (engl. Tile). Peli on tiilipohjainen (engl. Tile based), eli yhdessä tiilessä voi olla vain yhtä lattiatekstuuria, ja seinät ovat aina tiilien

välissä. Pelimoottorissa on kattava tiilipohjainen kenttäeditori, joka käyttää xml-tiedostotyyppiä kenttien tallennukseen. Kenttäeditorista voi myös ajaa luotuja skriptejä, mikä helpottaa testausta ja mahdollistaa automaation käytön myös kenttien käsin luonnissa. Kirjoitin esimerkiksi skriptin mikä täyttää tyhjät hyllyt laatikoilla, mikä oli käsin tehtynä tuskallisen hidasta.

KUVA 1. Kuvankaappaus kenttäeditorista, avattuna on yksi pelin erikoiskentistä.

Pelistä löytyy myös tuki yhteisön luomille modifikaatioille, joita pelaajat voivat jakaa Steam-palvelussa PC:llä. Käyttäjät voivat esimerkiksi muokata tai lisätä kenttiä ja aseita, sekä vaikka muuttaa pelin mekaniikkoja luoden uusia pelimuotoja.

2.4. Kenttäsuunnittelu

Suurin osa pelin kentistä luodaan proseduraalisesti, eli kenttägeneraattoriskripti ottaa kentälle määritellyt vaatimukset (kentän numeron, vaikeusasteen, koon, teeman, erikoismuuttujat ja aarteet) ja luo niiden mukaisen kentän. Kentän satunnaisuus tulee ns.

siemennumerosta (engl. Seed), josta satunnaisalgoritmi johtaa tarvittavat muuttujat, kuten huoneiden sisällöt. Kenttägeneraattorin tulisi aina luoda identtisen kentän samalla siemennumerolla.

Kentät koostuvat enimmäkseen valmiista käsintehdyistä huoneista. Jos generaattori ei löydä sopivan kokoista ja muotoista huonetta, se luo tyhjän huoneen ja lisää sinne muutaman esineen ja vihollisen.

KUVA 2. Yksi pelikerta muodostuu useammasta kentästä.

Huoneiden lisäksi myös jotkin kentät ovat kokonaan käsin tehtyjä. Niitä löytyy pelin alussa tutoriaalikenttinä, kappaleiden aluista tarinaa avaavina kenttinä, kappaleen lopputaisteluista sekä erikoiskentistä, joihin pelaaja yleensä törmää muutaman kerran yhden elämän aikana.

Tutoriaaleissa pelaajalle opetetaan pelin kontrollit ja mekaniikat. Kentät ovat melko lyhyitä, eikä niissä ole suurempia vaaroja. Valvoja puhuu myös pelaajalle ensimmäistä kertaa näissä kentissä.

Tarinakentissä avataan pelin juonta. Kentät alkavat melko rauhallisesti, mutta tarinaosuuden loputtua pelaajalle annetaan jokin suurempi haaste, kuten laseransa, josta pelaajan täytyy selvitä edetäkseen pelissä. Yleisenä teemana oli, että kentän alku ja loppu

olivat lähellä toisiaan, mutta pelaajan täytyi kulkea kentän toiseen laitaan ja aktivoida haaste.

Lopputaistelut ovat aina jotain erikoisempaa vihollista vastaan, jolla on hieman omaperäisempiä kykyjä. Näihin taisteluihin luotiin omat käsintehdyt kentät, jotta ne olisivat varmasti sopivia haasteeseen nähden. Esimerkiksi tuhatjalkaista muistuttavan loppupomon haluttiin kiemurtelevan pilarien välejä. Kentässä piti siis olla pilareita, mutta sen tuli myös olla melko avoin, jotta vihollinen mahtui liikkumaan, ja jotta pelaajalla olisi tilaa väistellä hakeutuvia ohjuksia.

Erikoiskenttien tarkoitus on antaa pelaajalle lisää aarteita, jos he selviävät haasteesta. Nämä kentät ovat lyhyitä, mutta usein haastavampia kuin normaalit kentät. Pelaaja ei voi paeta takaisin edelliseen kenttään, joten pelaajan täytyy pohtia, onko palkinto riskin arvoinen.

2.5. Pelin testaus

Koska proseduraalinen sisällöntuotanto on matemaattista ja automatisoitua, pitää lopullinen tuote tarkistaa tarkkaan, jotta peli on vielä läpäistävissä, eikä kenttägeneraattori vaikka jätä kentän loppua luomatta. Projektin pääohjelmoija muokkasi vihollisen tekoälystä pelaajalle oman tekoälyn tätä tarkoitusta varten. Näin pelin testaus saatiin automatisoitua. Jos tämä tekoäly ei saavuta kentän loppua tietyssä aikamääreessä, se tallentaa kentän senhetkisen tilan, josta näkyy pelaajan sijainti, sekä tarvittavat tiedot kentän uudelleenluomiseksi. Kun kenttägeneraatioon tehtiin suurempia muutoksia, automaattitestaaja jätettiin pelaamaan kenttiä läpi yön yli ja aamulla tarkistettiin ongelmatilanteet.

Automaattipelaaja ei kuitenkaan pysty arvioimaan kenttien ulkonäköä tai pelattavutta, joten peliä piti testata paljon myös manuaalisesti. Tähän kului melkei yhtä paljon työajastani kuin kenttäsuunnitteluun.

3 Työskentely 10tons Oy:ssa

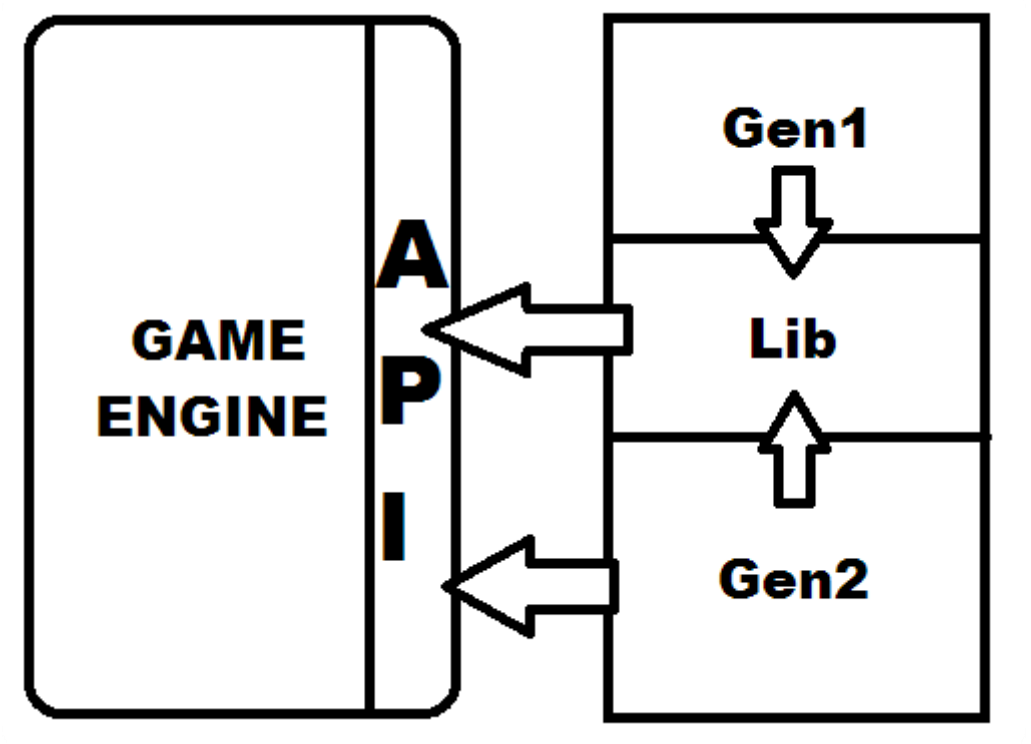
Työsopimuksen alkaessa sovimme, että aloitan tekemään huoneita ensimmäiseen kappaleeseen, jonka teemana on laboratorio. Näille kentille oli jo kenttägeneraattori, mutta sovimme että toiseen kappaleeseen tekisin erilaisen ratkaisun. Huoneiden suunnittelu oli hyvä tapa tutustua pelin kenttägeneraattoriin sekä ensimmäisen kenttägeneraattorin toimintaan.

Reilun kuukauden päästä aloitin oman kenttägeneraattorin suunnittelun toimistoteemaiseen kappaleeseen. Olin jo hieman lukenut artikkeleita aiheesta ja suunnitellut, minkä näköisiä kentistä haluaisin. Ensimmäinen ajatukseni kyberpunk-tyylisestä toimistoympäristöstä oli pitkä rivi työpisteitä ja halusin kenttien olevan säännöllisemmän näköisiä. Lisäksi halusin käyttää pelissä vielä käyttämättömäksi jääneitä lasisiltoja.

Ensimmäisen kenttägeneraattorin toiminnot olivat enimmäkseen erillisissä tiedostoissa, joita sain käyttää omassa skriptissäni vapaasti. Osa näistä toimi suoraan omassa toteutuksessani, mutta osissa oli riippuvuuksia toisiin kirjastoihin, joiden koin olevan tarpeettomia tässä versiossa, ja jotka olisivat monimutkaistanut toteutustani turhaan.

Esimerkkinä tästä on Decoman-kirjasto, joka muun muassa sijoittaa huoneet paikoilleen, mutta käyttää Floormapper-kirjastoa, joka taas vaatii omat kirjastonsa. Päätin kopioida kyseisen funktion omaan toteutukseeni ja muokata sen toimimaan ilman riippuvuuksia. Tämä toimi tarpeeksi hyvin tarpeisiini, mutta jouduin vähän väliä tekemään muutoksia omaan skriptiini, jotka olisin välttänyt jos olisin käyttänyt enemmän kirjastoja.

Kenttägeneraattorien ja kirjastojen toiminnot oli ohjelmoitu pelimoottoriin, niin sanuttuun ohjelmointirajapintaan (engl. Application programming interface, API). Suurin osa tarvittavista toiminnoista oli jo valmiina ensimmäisen kenttägeneraattorin vuoksi, mutta välillä tarvitsin jotai erikoisempaa toimintoa. Usein nämä vaativat vain pieniä muutoksia valmiisiin toimintoihin, mutta välillä ne olivat joko tarpeettoman vaativia tai jopa mahdottomia toteuttaa.



KUVA 3. Pelin rakenne. Ensimmäisen generaattorin toiminnot, kuten vihollisten korvaaminen, olivat enimmäkseen erillisissä tiedostoissa, joita kutsuttiin kirjastoiksi (engl. Library). Käytin toisessa kenttägeneraattorissa vain osittain näitä valmiita kirjastoja.

Tämän työn aikana muut työntekijät olivat enimmäkseen kesälomalla, mutta sain tarvittaessa yhteyden pääsuunnittelijaan, joka oli tehnyt pelin ensimmäisen kenttägeneraattorin, melko lyhyellä varoitusajalla. Sain myös usein hyödyllistä palautetta varsinkin suunnittelun alkuvaiheessa.

4 Proseduraalisuus

4.1. Yleisesti

Proseduraalisuuden idea on luoda käytännössä ääretön määrä pelattavaa ja näin pelille saadaan korkea uudelleenpeluuarvo. Proseduraalisuudella saadaan myös tarvittaessa pelikokemus jopa mukautumaan pelaajan tarpeisiin pelikokemuksen parantamiseksi. Jos pelaajalla menee huonosti ja osumapisteet ovat vähissä, saattaa kenttägeneraattori laittaa seuraavaan kenttään enemmän ensiapuvälineitä tai vähemmän vihollisia. (Extra Credits, 2015)

Yleisiä sudenkuoppia proseduraalisessa sisällöntuotannossa ovat laadunvalvonta ja kustannustehokkuus. Jos peli on lyhyt, ei välttämättä ole järkevää luoda sisältöä proseduraalisesti. Laadunvalvontaan saattaa helposti mennä enemmän aikaa, kuin käsin tekemiseen.

4.2. Erilaisia lähestymistapoja kenttägeneraatiossa

4.2.1 TinyKeep

Tutkiessani eri tapoja toteuttaa kenttägeneraattori, törmäsin useampaankin otteeseen TinyKeepin tekijän, Phi Dinhin, blogikirjoituksiin aiheesta. TinyKeep on ylhäältä kuvattu luolaseikkailu, jossa pelaajan tavoitteena on selvitä luolaston loppuun.

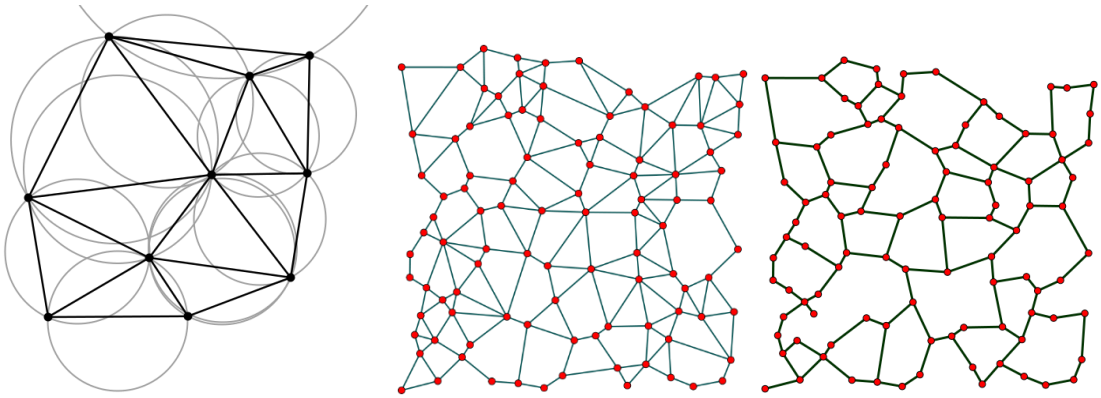
TinyKeepin kentät ovat luolastoja, joissa on selkeä alku ja loppu. Viimeinen ovi on usein suljettu, ja pelaajan täytyy löytää avain loppupomolta tai aktivoida vihollisinvaasio päästäkseen seuraavaan kenttään.

Aluksi luodaan suorakulmion muotoisia soluja (engl. cell) satunnaisesti alueen keskelle. Määrällä ei ole niin väliä, kunhan niitä on reilusti enemmän kuin tavoiteltu huoneiden määrä. Monet soluista ovat tässä vaiheessa päällekkäin. (Adonaac, A. 2015)

Seuraavaksi soluja liikutetaan poispäin toisistaan, kunnes yksikään solu ei leikkaa toisiaan.

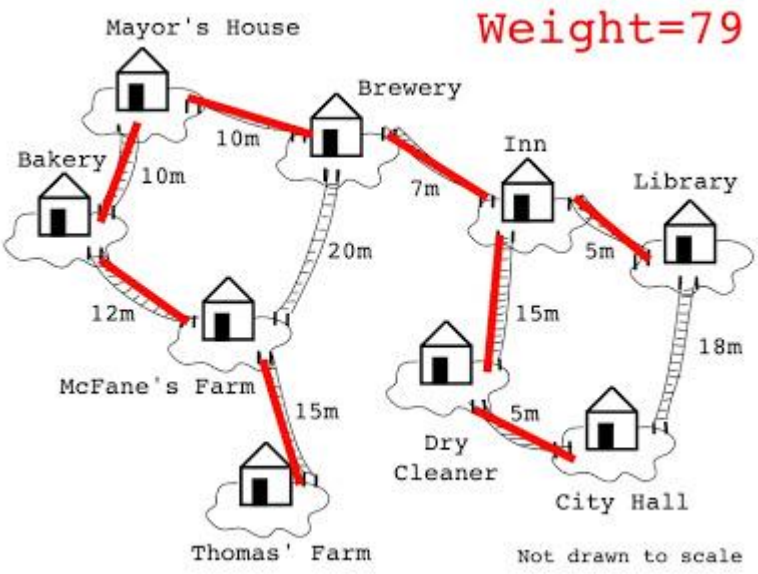
Sitten skripti päättää, mitkä soluista valitaan lopulliseen kenttään huoneiksi. TinyKeepissä huoneet valitaan määrittelemällä pienin sallittu leveys ja korkeus, mutta muissa projekteissa voisi esimerkiksi määritellä miten paljon lattiapinta-alaa kentässä pitää olla.

Huoneiden yhteydet määritellään algoritmilla. TinyKeepissa käytetään Delaunay triangulaatiota, mutta on myös olemassa Gabriel graafi (engl. Gabriel graph), sekä relatiiviympäristögraafi (engl. Relative Neighborhood Graph, RNG). Kaikki nämä tekevät käytännössä saman asian, eli yhdistävät huoneet viivoilla, jotka eivät risteä keskenään.



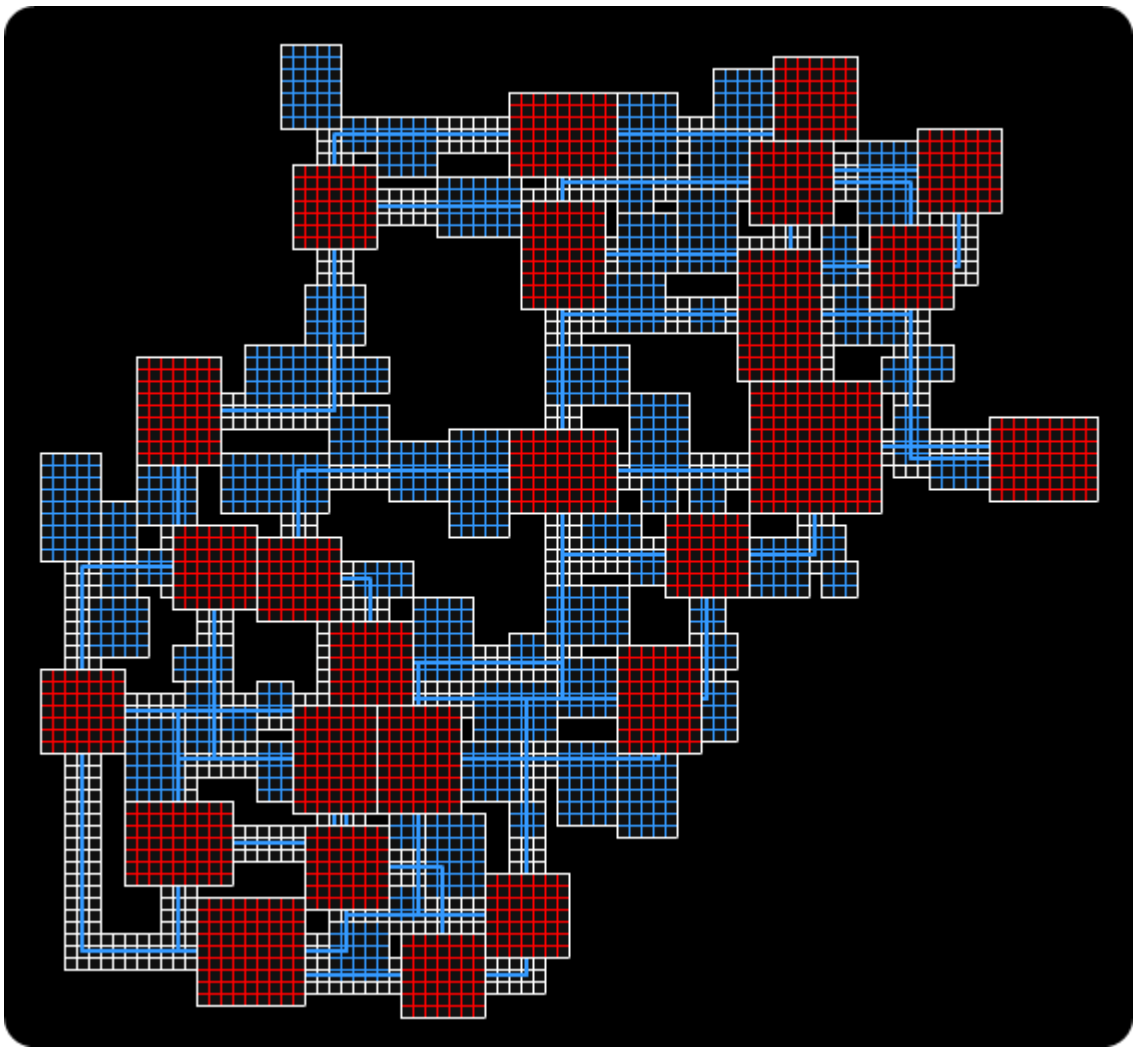
KUVA 4. Delaunay triangulaatio, Gabriel graafi sekä Relatiiviympäristögraafi. (Lähde: Wikipedia)

Tästä johdetaan polku, ns. pienin virittävä puu (engl. Minimum spanning tree), joka yhdistää jokaisen huoneen minimimäärällä viivoja (Kivinen, J., 2008). Tällä metodilla ei synny silmukoita, joten pelissä on yksi polku, joka vie kentän alusta loppuun, sekä polulle kuulumattomia huoneita, jotka muodostavat umpikujia.



KUVA 5. Esimerkki pienimmästä virittävästä puusta (punaiset viivat). Kuvitteellinen tarina kaupunkisuunnittelijasta, jonka työnä oli päivittää kaupungin sillat mahdollisimman halvalla niin, että jokaisesta saaresta pääsisi vielä kulkemaan, vaikka vanhat sillat hajoaisivat. (Computational Fairy Tales, 2011)

Huoneiden yhteydet on nyt määritelty, mutta ne ovat vielä todennäköisesti fyysisesti erillään toisistaan. Luodaan käytäviä, jotka yhdistävät huoneet toisiinsa muodostaen yhtenäisen ja pelaajalle kuljettavan pelimaailman.



KUVA 6. Esimerkki TinyKeepin interaktiivisen esittelyn lopputuloksesta. (Dinh, P. 2013)

4.2.2 Spelunky

Toinen hyvin havainnollistettu ja dokumentoitu esimerkki proseduraalisesta kenttägeneroinnista on Derek Yun Spelunky. Spelunky on sivulta kuvattu tasohyppely, jossa pelaajan on tarkoitus kerätä mahdollisimman paljon aarteita yhden elämän aikana. Pelaajan käytössä on rajattu määrä köysiä ja pommeja, joita löytyy lisää kentistä. Spelunkyn kenttägeneraattorin perusperiaate muistuttaa paljon luomaani kenttägeneraattoria, joten käsittelen sen toimintaa hieman tarkemmin kuin edellistä.

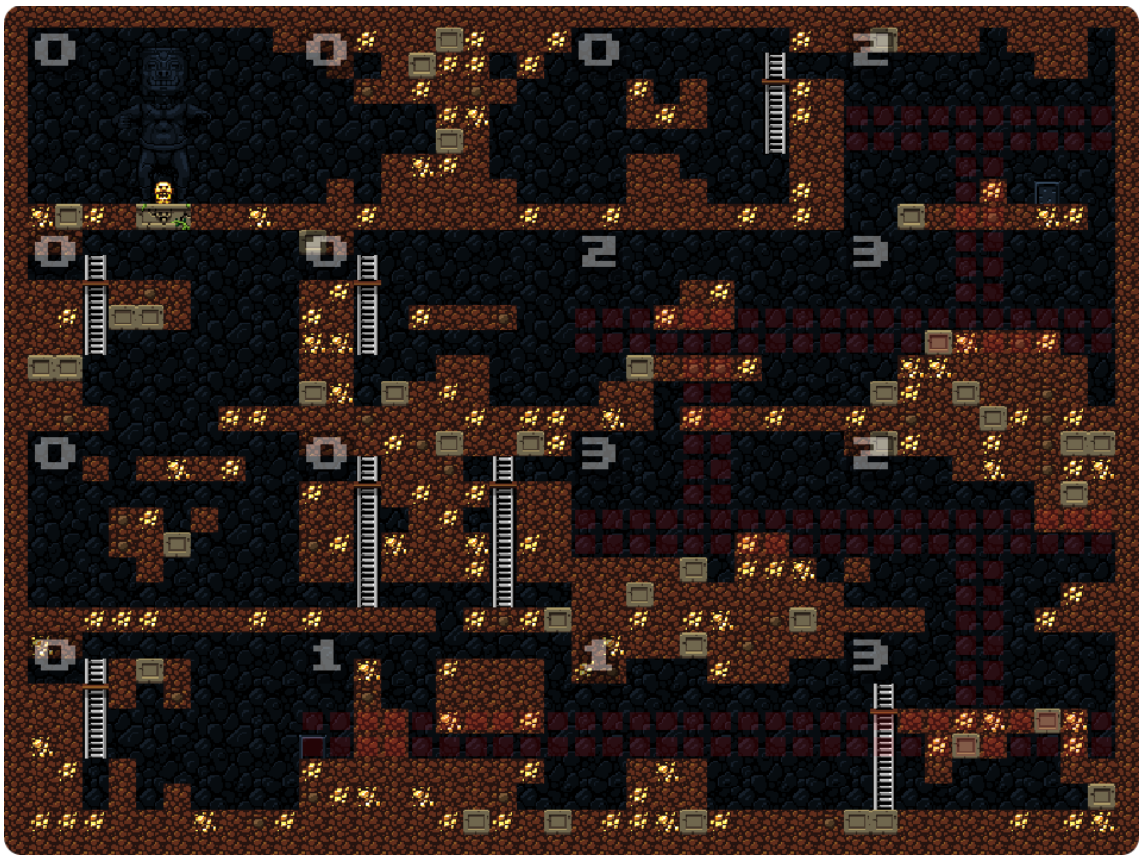
Spelunkyn pelialue on jaettu kuuteentoista huoneeseen, jotka ovat 4x4 ruudukossa. Kenttä alkaa aina ylärivistä ja päättyy alariviin. Satunnaisesti valitusta alkuhuoneesta kenttägeneraattori alkaa kulkemaan huoneiden läpi, kunnes se päätyy alimmalle riville, jossa jokaisella huoneella on todennäköisyys muuttua loppuhuoneeksi. Tästä muodostuu

niin sanottu kriittinen polku, jota pitkin pelaaja aina pääsee kentän loppuun. Polulle kuulumattomat huoneet täytetään satunnaisilla huonetyypeillä.

Huoneita on neljää eri tyyppiä: yhdessä on vain suora reitti, toisessa on pudotus alas ja kolmanteen voi pudota. Viimeinen tyyppi ei kuulu polulle, joten sen muoto voi olla lähes mitä vain. Nämä ovat numeroitu kuvassa 7: suora reitti on 1, pudotus alas on 2, pudottautumispala on 3 ja polulle kuulumaton pala on 0.

Jokaiselle huonetyypille on useita erilaisia sapluunoita (engl. Template), joista kenttägeneraattori satunnaisesti valitsee joka huoneen kohdalle. (Kazemi, D., 2013) Sapluunoissa on määriteltyjen avonaisten reunojen välillä avoin reitti, jonka pelaaja pystyy kulkemaan käyttämättä pommeja tai köysiä.

Huoneille luodaan lisää variaatiota todennäköisyyksillä. Tarkoitan tällä, että on tietty todennäköisyys, että ansa luodaan sille annettuun paikkaan. Joissain huoneissa on myös paikka 5x3 kokoisille esteille, joilla on omat todennäköisyytensä esiintyä. (Yu, D. 2016)



KUVA 7. Esimerkki Spelunkyn kentästä. (Kazemi, D., 2013)

4.3. Proseduraalisuus Left4Dead-pelissä

Left4Dead on Turtle Rock Studiosin tuottama, Valve Corporationin julkaisema, pelaajilta yhteistoimintaa vaativa ammuntapeli. Pelaajien on tarkoitus yhdessä selvitä kentän loppuun taistellen erilaisia zombeja vastaan.

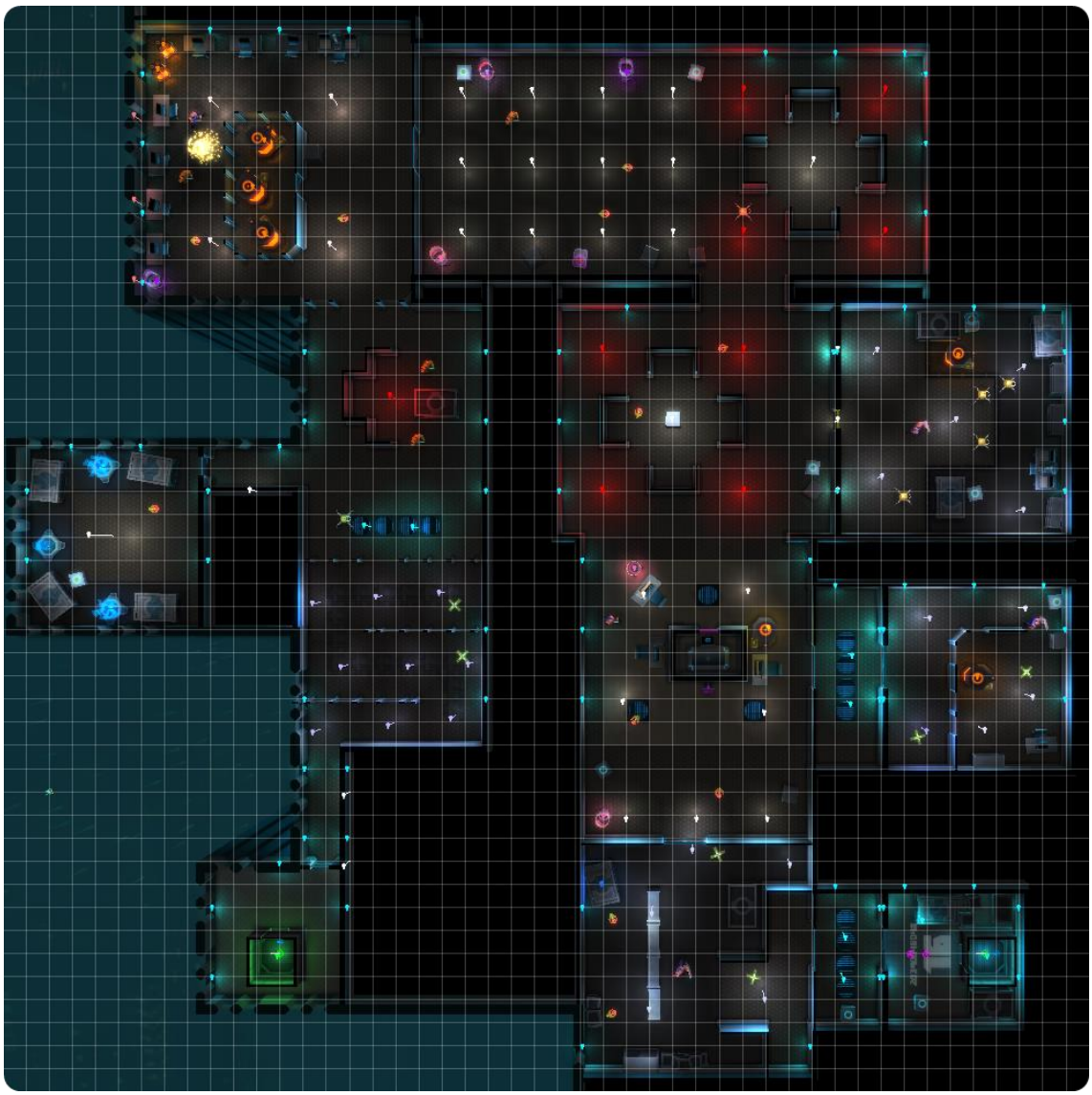
Itse kentät ovat Left4Deadissa kokonaan käsintehtyjä, mutta pelin vaikeusaste mukautuu vahvasti pelaajien taitojen mukaan. Valven toimitusjohtaja Gabe Newell kutsuu Edgelle kirjoittamassaan blogikirjoituksessa tätä dynaamista vaikeusastetta proseduraaliseksi kerronnaksi. (Newell, G., 2008)

Peli tutkailee, miten pelaajat käyttäytyvät ja selviävät haasteista, ja säätää seuraavia kohtaamisia sen mukaisesti. Mikäli pelaajat rauhallisesti ja järjestelmällisesti tyhjentävät huoneet hetkessä, saattaa seuraavan nurkan takaa hyökätä aalto zombeja. Jos jokin tietty zombityyppi on tuottanut pelaajille erityisesti ongelmia, peli ottaa sen huomioon tasapainotuksessa.

5 Proseduraalisuus Neon Chromessa

5.1. Algoritmipohjainen kenttägeneraattori

Neon Chromen ensimmäisen kenttägeneraattorin ohjelmoi pelin pääsuunnittelija. Se on ottanut vaikutteita TinyKeepin toteutuksesta, mutta huoneiden välisiä käytäviä on huomattavasti vähemmän.



KUVA 8. Esimerkki valmiista laboratorioteemaisesta kentästä.

Käytävien luomisen sijaan kenttägeneraattori pyrkii saamaan kaikki huoneet kiinni toisiinsa niin, että huoneiden väliseen seinään mahtuu ovi. Liiallinen käytävien käyttö ei sopinut Neon Chromen pelityyliin, koska pelaajan sivuttaisliike on tärkeä osa pelin vetovoimaa. Lisäksi viholliset hyökkäisivät pelaajan kimppuun jonossa, mikä ei ole haluttu lopputulos sekä visuaalisesti että pelattavuuden kannalta.

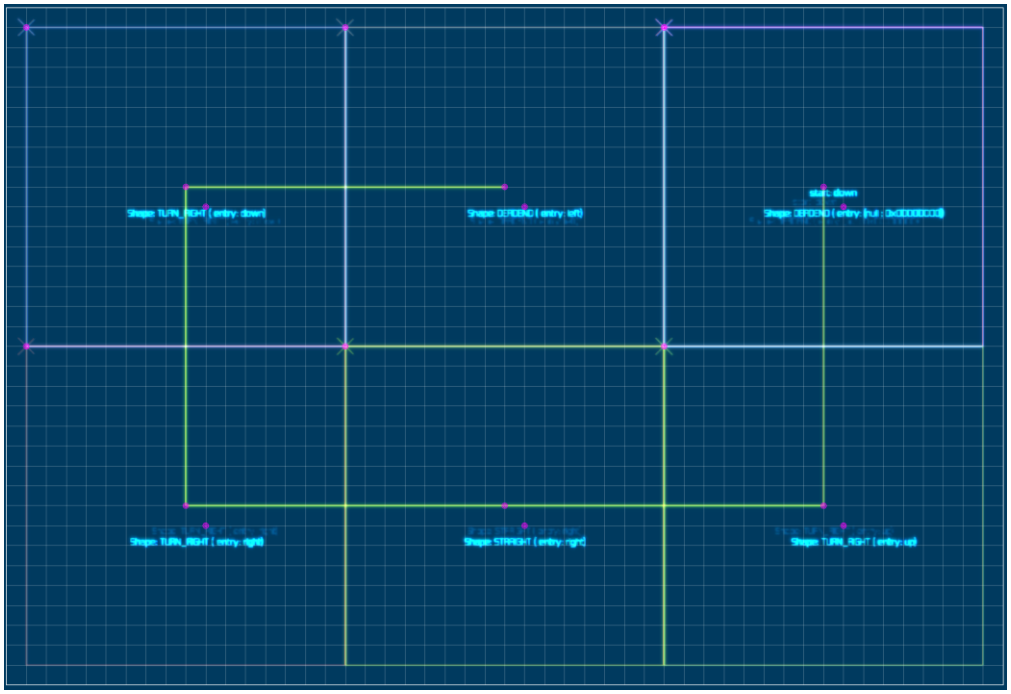
Huoneet on jaoteltu eri muotojen mukaan. Vasemmalle ja oikealle mutkalle on eri huonetyypit, koska pelaajan tulosuunta vaikuttaa huoneen pohjapiirustukseen. Viholliset voidaan laittaa katsomaan pelaajan tulosuuntaan tai olla selkä pelaajaa päin. Huoneissa voi myös olla suljettuja ovia tai ansoja, jolloin pelaajan täytyy painaa nappia tai tuhota energianlähde (kuva 9) päästäkseen läpi. Näissä huoneissa on tärkeää tietää pelaajan tulosuunta.

KUVA 9. Esimerkki vasemmasta mutkasta, jossa pelaajan täytyy tuhota energianlähde (korostettu punaisella laatikolla) päästäkseen eteenpäin (vihreä laatikko).

5.2. Ruudukkopohjainen kenttägeneraattori

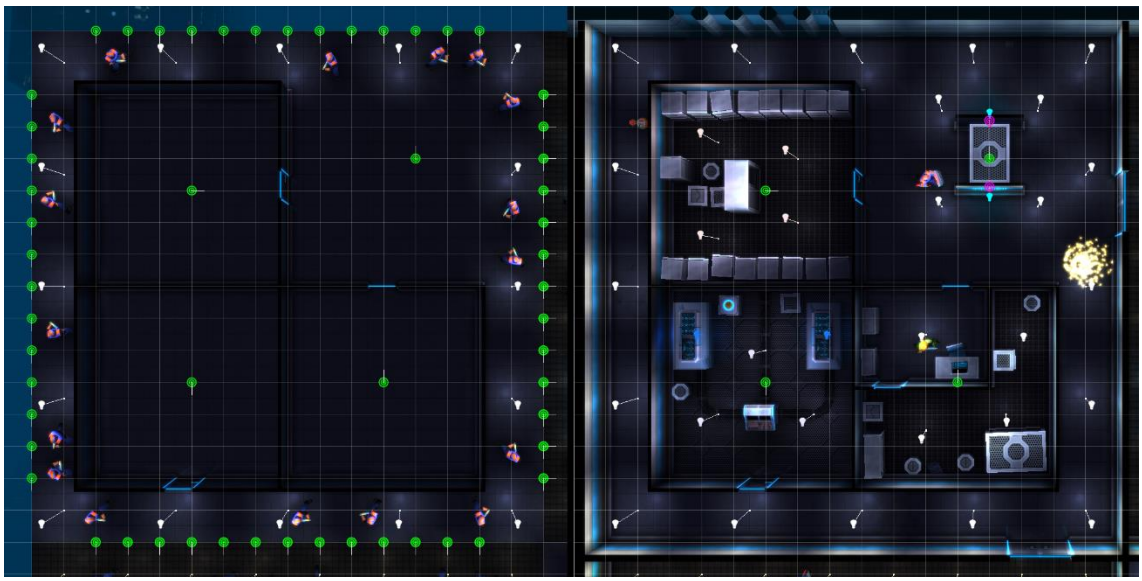
Tämän kenttägeneraattorin haluttiin luovan vaihtelua kenttiin verrattuna edelliseen ratkaisuun. Tästä syystä päädyin seuraamaan Derek Yun ideaa Spelunkysta, jossa pelialue jaettiin ruudukoksi. Tämä idea myös tuki ajatustani pitkästä rivistä työpisteitä, jonka koin kuvastavan pelin teemaa ja toimistoympäristöä, johon kenttäni tulisivat sijoittua.

Skripti pyöristää pelialueen koon kuudellatoista jaolliseksi pysty- ja vaakasuunnassa ja jakaa pelialueen 16x16 neliöihin. Maksimikoko kentälle on 48x48, mutta yleensä kentät ovat enintään 48x32, koska isoimpien kenttien läpipeluuaika koettiin kohtuuttomaksi. Kenttä alkaa aina pelialueen kulmasta ja kentän loppuun vievä reitti kattaa jokaisen 16x16 neliön.



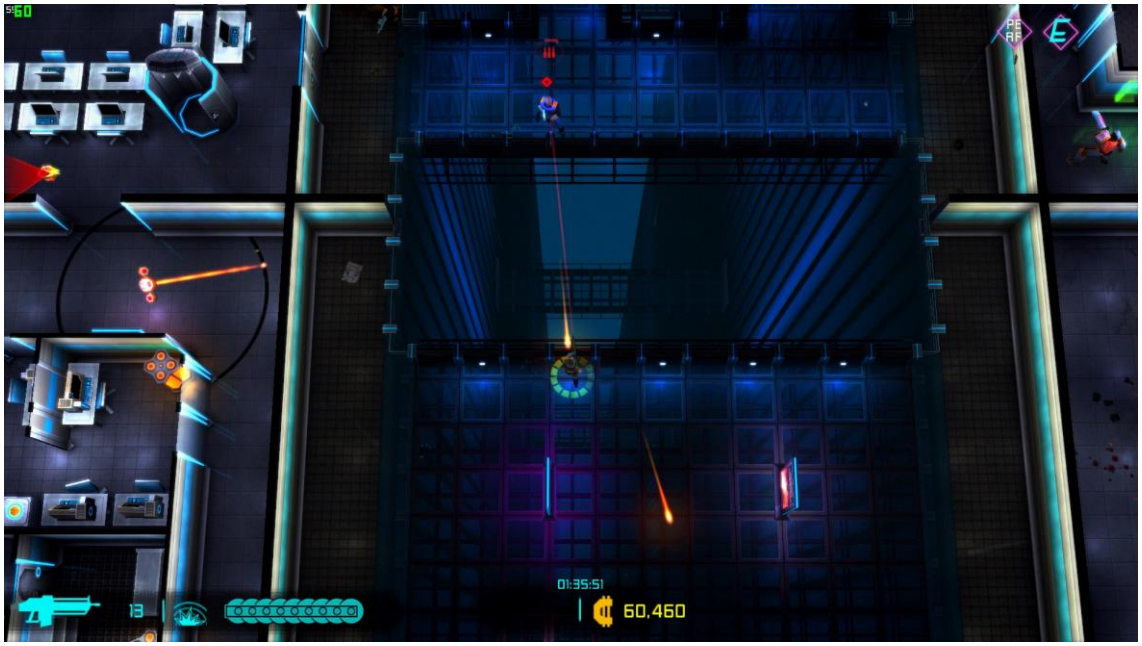
KUVA 10. Esimerkki reitistä. Tämä kenttä tulee alkamaan oikeasta ylänurkasta.

Nämä 16x16 neliöt korvataan editorissa luoduilla huoneilla, joita kutsun kokoumiksi (engl. Aggregate), koska niissä on paikkoja pienemmille huoneille. Tarkoituksena on, että yhdistelemällä pienempiä huoneita luodaan enemmän variaatiota vähemmällä vaivalla. Osassa kokoumista ei ole paikkoja pienemmille huoneille. Näissä on usein jokin suurempi kokonaisuus, jota ei pystynyt helposti jakamaan pienempiin osiin, esimerkiksi ruokala.



KUVA 11. Vasemmalla kokouman alkutilanne, oikealla sama kokouma valmiissa kentässä.

Suorat palaset ovat aina avoimia yhdyskäytäviä, jotka vievät pelaajan välillä ulkoilmaan. Nämä tilat koostuvat kahdesta parvekkeesta, joiden välillä on lasisilta. Yleensä nämä tilat ovat hyvin avoimia, eikä pelaajalla ole mitään suojaa. Joskus lasisiltoja saattaa olla kahdessa rinnakkaisessa palasessa, jolloin pelaajan ja vihollisten välille voi syntyä tulitaistelu, jollaista ei muissa kentissä voi sattua.

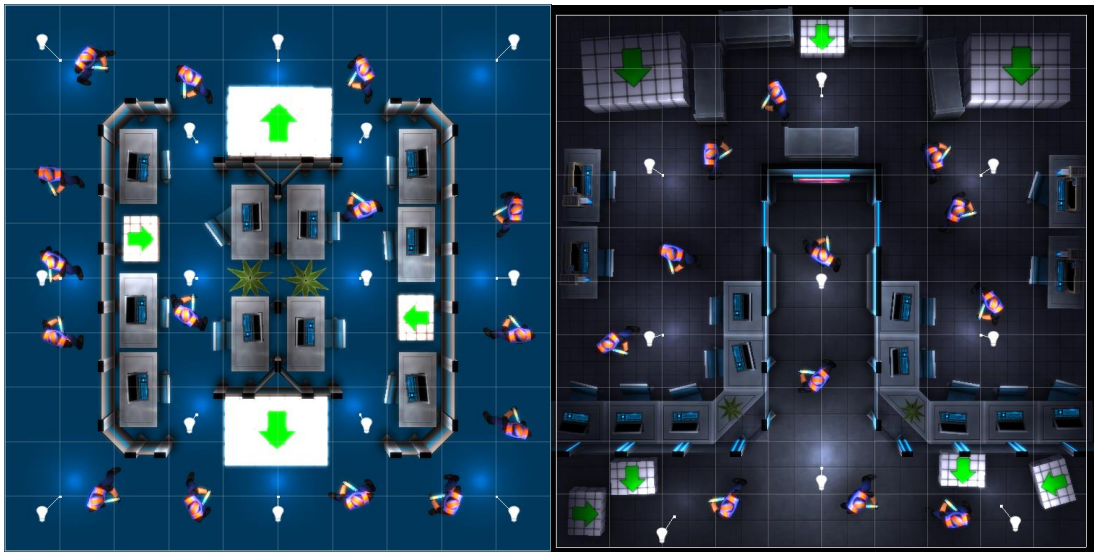


KUVA 12. Esimerkki rinnakkaisista lasisilloista.

Ovien sijoitus oli alun perin tarkoitus tehdä määrittelemällä kokoumalle mahdolliset ovikohdat ja tarkistamalla että yhdistettävistä kokoumista löytyy yhteinen kohta, johon oven voi laittaa. Ongelmaksi nousi ovikohtien merkintä, joka tehtiin väliaikaisilla merkeillä (vihreät ympyrät kuvassa 13). Kaikki huoneet, mukaan lukien kokoumat, tallennetaan xml-tiedostoihin, joiden nimet ladataan pelin alussa muistiin. Pelimoottori ei kuitenkaan tukenut näiden tiedostojen lukemista suorituksen aikana, koska se pidentäisi latausaikoja kenttien välissä. Päädyin tästä syystä käyttämään samantyylistä muotoihin jaottelua, kuin muissa pelin generaattoreissa. Tämä karsii hieman kokoumien muotojen variaatiota, koska jokaisessa kokoumassa pitää jättää pitkä pätkä seinää avoimeksi mahdollisen oven sijoittelulle.

KUVA 13. Vasemmalla avoin kokouma, oikealla mutka.

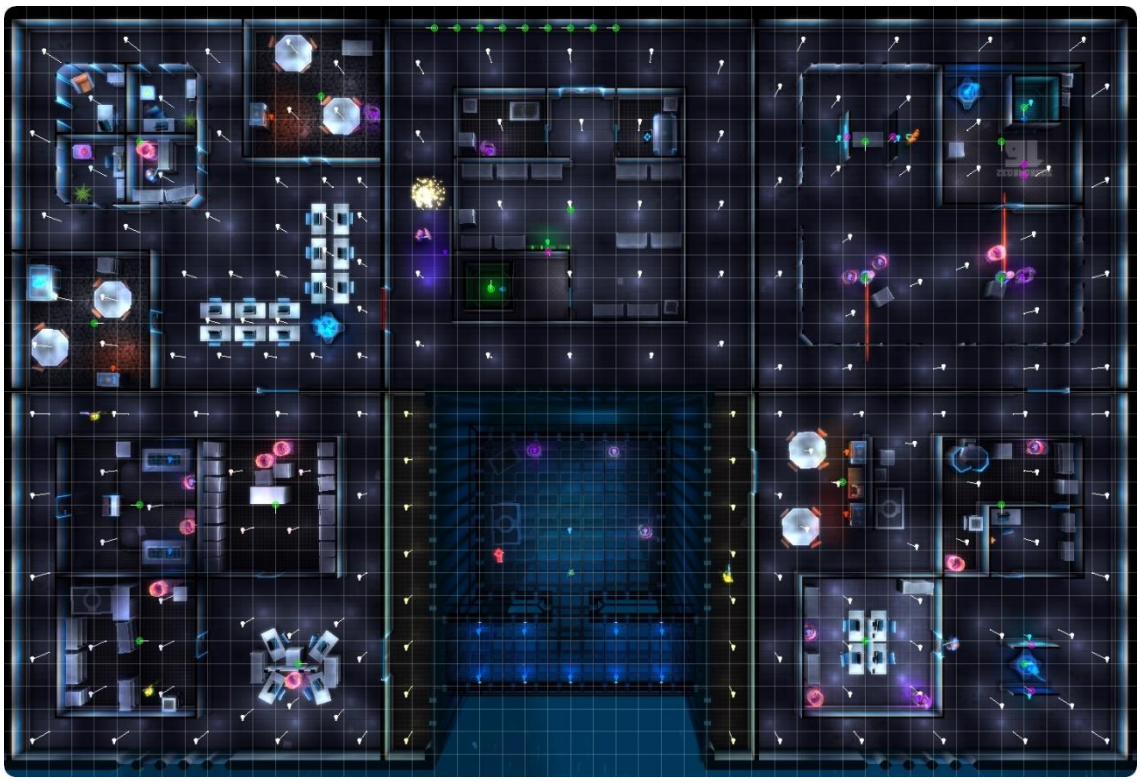
Pienemmät huoneet ovat jaoteltu avoimiin ja suljettuihin muotoihin. Avoimia huoneita voi käännellä 90 asteen kulmissa miten päin vain, mutta suljettujen huoneiden suunta on aina määritelty kokoumassa niihin johtavan oven sijoittelun helpottamiseksi.



KUVA 14. Vasemmalla avoin huone, oikealla suljettu. Avoimissa huoneissa ei ole määritelty lattiaa, koska se rikkoi lattiatekstuurin jatkuvuuden. Suljetut huoneet on ympäröity seinillä, joten samaa ongelmaa ei ilmaannu.

Vaikka kentät ovat käytännössä hyvin lineaarisia, ja reitti kentän alusta loppuun johtaa joka kokouman läpi ilman umpikujia, pelaaja voi joskus jättää osan vihollisista rauhaan. Seinien tuhoamisen mahdollisuus myös rikkoo lineaarisuuden tunnetta.

Viimeiseen kokoumaan vievä ovi on aina suljettu ja vaatii punaisen avainkortin. Kenttägeneraattori sijoittelee sen satunnaisesti johonkin ovea edeltävistä kokoumista.

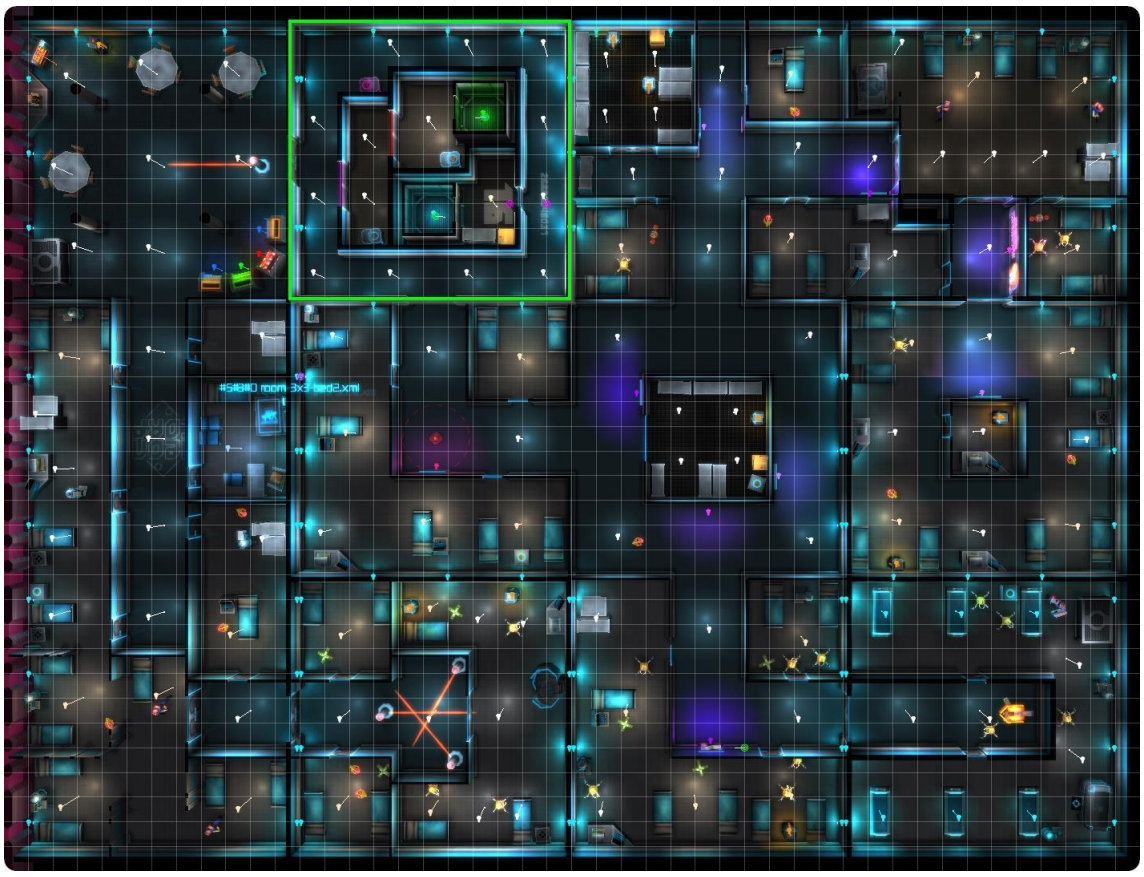


KUVA 15. Esimerkki valmiista toimisto-teeman kentästä editorinäkymässä.

Verrattuna ensimmäiseen generaattoriin, kentät ovat hyvin säännöllisen näköisiä. Tämä toimii sen puolesta ja vastaan, sillä vaikka kentät näyttävät realistisemmilta, varsinkin rakennuksen ulkoreunojen kannalta, saattaa monet kentät vaikuttaa hyvin samanlaisilta. Onneksi yhdyskäytävien siniset lasisillat (kuva 15, alarivin keskellä) tuovat vähän variaatiota kenttien harmauteen ja rikkovat ulkoseinien suoran linjan.

5.3. Hybridikenttägeneraattori

Ruudukkoideasta inspiroituneena pääsuunnittelija päätti kirjoittaa pelin viimeiseen, sairaalateemaiseen kappaleeseen vielä uuden kenttägeneraattorin, jossa on yhdistelty molempien edellä mainittujen skriptien ominaisuuksia. Tämä jakaa kentän 11x11 neliöihin, ja kentän alku ja loppu ovat samassa neliössä. Pelaajan tavoitteena on löytää avainkortti ja tuhota kentästä löytyvät satunnaisesti sijoitellut energianlähteet päästäkseen seuraavaan kenttään.



KUVA 16. Esimerkki sairaala-teemaisesta kentästä. Kentän alku ja loppu on korostettu vihreällä laatikolla.

Ruudukkogeneraattorin tapaan hybridigeneraattorissa on paikkoja pienemmille huoneille, mutta lisäksi samalla metodilla kenttiin on laitettu pienempiä yksityiskohtia. Joillakin käytävillä saattaa olla ansoja, kuten lasersäteitä tai miinoja. Koska kyseessä on sairaalaympäristö, kentissä on paljon vuoteita. Näille on varattu 3x3 ja 4x4 tiloja, joihin generaattori etsii sopivan kokoisen ”huoneen”.

Jokainen polku päätyy umpikujaan, joten kentissä joutuu kulkemaan edestakaisin edellisiä kenttiä enemmän. Näitä kenttiä on kuitenkin vasta pelin loppupuolella, joten vihollisia on huomattavasti enemmän. Tämä tuo kenttiin myös eräänlaista rytmitystä hengähdystauoilla suurempien taistelujen välissä.

5.4. Muu proseduraalisuus

Kenttien lisäksi peli muuttuu ja mukautuu ajan myötä. Osa muutoksista riippuu pienissä määrin pelaajan taidoista. Nämä muutokset ovat kuitenkin tarkoituksella melko pieniä, jotta pelaajalle ei tulisi tunne, että häntä pidellään kädestä. Muutosten tarkoitus on vähentää pelaajan turhautumista ja taata miellyttävämpi kokemus.

Jos pelaaja on ottanut paljon osumia, pelin yleistä vaikeusastetta vähennetään pienellä määrällä. Tämä ei tarkista pelaajan senhetkistä tilannetta, vaan katsoo yleiskuvaa koko peliajalta. Käytännössä tämä siis vähentää vihollisten määrää, jos pelaajan tuottama vahinko ja ottama vahinko ovat tiettyjen raja-arvojen ulkopuolella.

Jos pelaajan osumapisteet alkavat olla vähissä, ensiapupakkauksen löytymisen todennäköisyys kasvaa. Tämä antaa pelaajalle lisää toivoa edetä pelissä, vaikkakin ensiapupakkaukset parantavat vain 10%. Suurin osa kuolemista tulisi tapahtua lopputaisteluissa, jotta ne tuntuisivat suuremmilta haasteilta kuin perusviholliset.

Kaavallisuuden rikkomiseksi pelissä on myös muuntimia (engl. Modifier), jotka muuttavat kenttää tai vihollista. Kenttämuuntimet voivat esimerkiksi lisätä kenttään räjähtäviä tynnyreitä ja miinoja tai poistaa kaikki ovet. Vihollismuuntimet luovat haastavampia erikoisversioita vihollisista, jotka ovat esimerkiksi nopeampia tai parantavat muita vihollisia ympärillään. Vihollismuuntimia on vähintään yksi per kenttä, mutta kenttämuuntimet ovat melko harvinaisia.

6 POHDINTA

Lopputulos vastaa sille annettuja vaatimuksia. Nämä vaatimukset muuttuivat projektin myötä, esimerkiksi kenttämuuntimet tulivat vasta myöhemmin. Koska en ollut käyttänyt monia ensimmäisen kenttägeneraattorin funktioita, piti uudet ominaisuudet usein lisätä käsin.

Huomasin vasta jälkikäteen tätä opinnäytetyötä kirjoittaessani, että Derek Yu oli käyttänyt Spelunkyssa samankaltaista pienempien yksityiskohtien lisäämistä huoneisiin, kuin käytin toteuttamassani ruudukkopohjaisessa kenttägeneraattorissa. Olin tutkinut Spelunkya vain pintapohjaisesti aloittaessani työn tekemistä, eikä esimerkiksi Spelunky-kirjaa ollut vielä silloin saatavilla.

Tämä projekti oli ensimmäinen myyntiin päätynyt peli, missä olin mukana. Ohjelmointiosaamiseni auttoi, vaikka en ollut squirrel-kielestä aikaisemmin kuullutkaan. Kenttäsuunnittelun lisäksi opin paljon ryhmätyöskentelystä ja pelisuunnittelusta. Koen myös, että pääsin helpolla, sillä projektilla ei ollut ulkopuolista julkaisijaa, joten deadlinet eivät olleet tiukkoja.

Oletan että tein jotain oikein, koska minut palkattiin kolmen kuukauden työharjoittelun jälkeen jatkamaan projektin loppuun asti.

LÄHTEET

10tons Oy. 2016. About Neon Chrome. Luettu 7.11.2016. http://neonchromegame.com/about/

Extra Credits. 22.7.2015. Procedural Generation - How Games Create Infinite World. Katsottu 7.11.2016. https://www.youtube.com/watch?v=TgbuWfGeG2o

Adonaac, A. 3.9.2015. Procedural Dungeon Generation Algorithm. Luettu 7.11.2016. http://www.gamasutra.com/blogs/AAdonaac/20150903/252889/Procedural_Dungeon_Generation_Algorithm.php

Dinh, P. 2013. TinyKeep Dungeon Generation Demo. Katsottu 11.7.2016. http://tinykeep.com/dungen/

Kivinen, J. Helsingin Yliopisto. Kevät 2008. Tietorakenteet-kurssin luentomateriaali, luentoviikko 11. Luettu 7.11.2016. https://www.cs.helsinki.fi/u/jkivinen/opetus/tira/k08/viikko11.pdf

Kubica, J. Computational Fairy Tales. 16.8.2011. Minimum Spanning Trees, Prim's Algorithm, and Bridge Upgrades: Part 8 of Ann's Visit to G'Raph. Luettu 14.11.2016. http://computationaltales.blogspot.fi/2011/08/minimum-spanning-trees-prims-algorithm.html

Yu, D. 2016. Spelunky. Boss Fight Books, Los Angeles, CA. Kirja myynnissä vain sivustolla https://bossfightbooks.com/

Kazemi, D. 19.8.2013. Spelunky Generator Lessons. Katsottu 7.11.2016. http://tinysubversions.com/2013/10/spelunky-generator-lessons/

Brown, M. 12.4.2016. Game Maker's Toolkit: How (and Why) Spelunky Makes its Own Levels. Katsottu 7.11.2016. https://www.youtube.com/watch?v=Uqk5Zf0tw3o

Newell, G. Valve Software. 2008. Gabe Newell Writes for Edge. Luettu 7.11.2016, arkistoidulta sivulta vuodelta 2013. https://web.archive.org/web/20130403031720/http:/www.edge-online.com/features/gabe-newell-writes-edge/

Töyssy, S. 10tons Oy. 3.3.2016. About Procedural Level Generation in Neon Chrome. Luettu 7.11.2016. http://neonchromegame.com/2016/03/03/neon-chrome-level-generation/