OULUN AMMATTIKORKEAKOULU

Aleksi Statsevich

Proseduraalinen generointi 2D-videopelissä

Proseduraalinen generointi 2D-videopelissä

Aleksi Statsevich
Opinnäytetyö
Syksy 2017
Tietojenkäsittely
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietojenkäsittely, Internet-palvelut ja sosiaalinen media

Tekijä: Aleksi Statsevich
Opinnäytetyön nimi: Proseduraalinen generointi 2D-videopelissä
Työn ohjaaja: Matti Viitala
Työn valmistumislukukausi ja -vuosi: Syksy 2017 Sivumäärä: 35

Tämä opinnäytetyö perustuu toimeksiantoon Super God Oy:ltä. Super God on oululainen pelialan yritys, joka tekee 2D-pelejä pc:lle, konsoleille ja mobiilialustoille. Yritys on käyttänyt Game Maker Studiota pelikehityksessä. Opinnäytetyön lopputuloksena oli pelidemo, joka toteutettiin uudella Game Maker 2:lla. Tämän opinnäytetyön lopputuloksena tulleesta demosta Super God Oy voi jatkokehittää lopullisen tuotteen. Opinnäytetyön pääpaino oli vihollisten ja pelikenttien proseduraalisessa generaatiossa. Kenttien generointi perustuu konseptiin, mitä käytetään videopelissä nimeltä Spelunky. Vihollisten proseduraalinen generointi on kokonaan itse kehitetty.

Opinnäytetyön lähdemateriaaleina on käytetty lähes yksinomaan internet-lähteitä, koska Game Maker 2:sta ei ole olemassa paljoa kirjallisuutta. YouTubessa on taas paljon opetusvideota Game Maker 2:lle ja niitä olen käyttänyt työssä apuna.

Opin proseduraalisesta generoinnista ja sen konsepteista paljon tämän opinnäytetyön tekemisen aikana. Rakennettu systeemi on hyvin muokattavissa uusia peliprojekteja varten ja palvelee näin toimeksiantajan tarpeita hyvin.

Asiasanat: Pelit, Pelikehitys, Pelisuunnittelu, Peliohjelmointi

ABSTRACT

Oulu University of Applied Sciences
Information technology, Internet services and social media

Author: Aleksi Statsevich
Title of thesis: Procedural generation in 2D video game
Supervisor: Matti Viitala
Term and year when the thesis was submitted: Autumn 2017 Number of pages: 35

This thesis is an assignment from Super God ltd. Super God is a game company from Oulu which makes 2D games for pc, consoles and mobile. The company uses Game Maker Studio for development. Product of this thesis was a game demo developed with Game Maker 2 which is newer version of the Game Maker Studio. Super God ltd can develop demo made in this thesis to a full product. Main focus in this thesis is on procedural generation of levels and enemies. Generation of levels is based on concept used in game called Spelunky. Procedural generation of the enemies is completely developed by me.

Information gathered for this thesis is completely from web because there is not much hard cover literature about Game Maker 2. On the contrary YouTube has a lot of tutorials and information about the topic and I have used them a lot.

I learned a lot about procedural generation and its concepts while doing this thesis. System that was built is modifiable for future game projects.

Keywords: Games, Game Development, Game Design, Game Programming

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheena on proseduraalinen generointi 2D-videopelissä. Proseduraalisella generoinnilla tarkoittaa säännönmukaista satunnaisuutta. Näitä sääntöjä luodaan algoritmeilla. Eli toisin sanottuna tietokoneelle annetaan raamit joiden sisällä satunnaisuuden pitää tapahtua. Minua kiinnosti opetella ja tutkia proseduraalisen generonnoin menetelmiä ja toteutusta. Opinnäytetyö on toimeksianto Super God Oy:lle, joka on oululainen peliyhtiö. Super God on julkaissut yhden pelin Steamissa. Pelin nimi on Riptale.

Työssä keskityttiin vihollisten ja pelikenttien proseduraaliseen generointiin. Tavoitteena oli saada pelidemo, jossa on edellä mainitut ominaisuudet. Työ toteutetaan Game Maker 2:lla. Toimeksiantaja käyttää vielä vanhaa Game Maker Studiota. Tässä opinnäytetyössä on myös tarkoitus selvittää, onko vaihto Game Maker Studiosta Game Maker 2:een vielä tarpeen.

Koulussa on käyty pelikehittämisen liittyviä opintokokonaisuuksia. Sen lisäksi opiskelin pelikehitystä vuoden verran Oulun Ammattikorkeakoulun Game Labissa. Olin itse omassa projektissa pelisuunnittelijana. Tässä opinnäytetyössä halusin oppia pelikehittämisen lisäksi myös paremman ohjelmointirutiinin, mikä auttaa työelämässä. Lähteenä on käytetty internet-lähteitä, koska Game Maker 2:sta ei ole paljoa kirjallisuutta. YouTubessa on taas paljon opetus- ja teoriavideoita liittyen Game Makerin molempiin versioihin ja pelikehittämiseen.

2 GAME MAKER

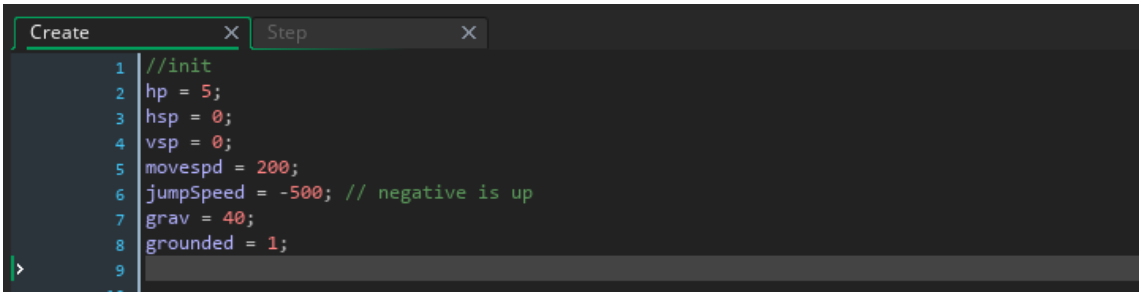
Game Makerin ensimmäinen versio julkaistiin vuonna 1999. Alkuperäisen Game Makerin viimeinen versio on 1.4 ja se tunnetaan myös nimellä Game Maker Studio. (Wikidot 2017, Viitattu 25.9.2017.) Game Maker 2 on julkaistu vuonna 2017 (Yoyo Games 2017b, Viitattu 25.9.2017). Game Makerin suosio on kasvanut indie-kehittäjien keskuudessa ja osa sillä tehdyistä peleistä on menestynyt kaupallisesti. Hyper Light Drifter julkaistiin 31. maaliskuuta 2016 ja se on myynyt n. 570 000 kappaletta Steamissä. Muita Game Makerilla tehtyjä menestyneitä pelejä ovat muun muassa Stealth Bastard Deluxe, Downwell ja Risk of Rain. (Yoyo Games 2017, viitattu 25.9.2017 ja Steamspy 2017, viitattu 25.9.2017.) Super God kehitti ensimmäisen tuotteensa Riptalen Game Maker Studiolla. Tässä osiossa avataan Game Makerin perusteita ja termejä mitä tullaan käyttämään myöhemmin opinnäytetyössä.

2.1 Tapahtumat

Game Makerissa ohjelma pyörii tapahtumissa. Tässä kappaleessa kerrotaan yleisimpiä Game Makerissa käytettyjä tapahtumia ja kerrotaan niiden toiminnasta.

Create-tapahtuma

Create eventissä oleva koodi ajetaan vain silloin kun instanssi luodaan ensimmäisen kerran. Create tapahtumassa yleensä alustetaan kyseisen objektin tarvitsemat muuttujat. (Yoyo Games 2017a, viitattu 26.9.2017.) Kuvio 1 on esimerkki create tapahtumasta



KUVIO 1. Create-tapahtuma

Step-tapahtuma

Game Makerissa aika jaetaan Steppeihin. Pelin room speed määrittää kuinka monta steppiä sekunnissa tapahtuu. Step-tapahtuman sisällä oleva koodi ajetaan yhden stepin aikana. Step-

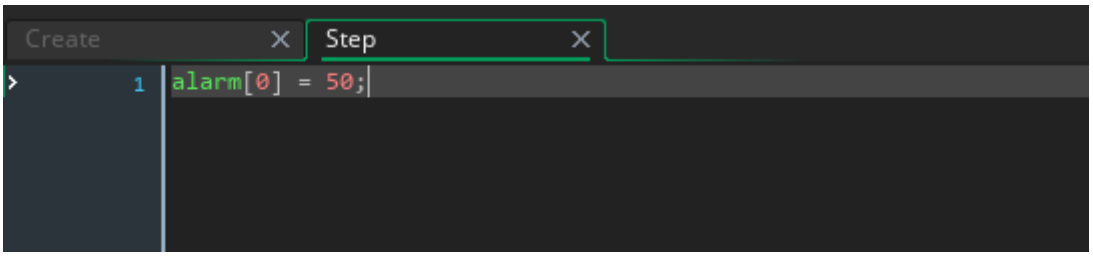
tapahtumia on kolme erilaista: on pelkkä step, jossa toiminta tapahtuu edellä mainitulla tavalla, begin step, joka ajetaan ennen muita step-tapahtumia ja end step, joka ajetaan muiden step-tapahtumien jälkeen. (Yoyo Games 2017j, viitattu 28.9.2017.)

Destroy-tapahtuma

Destroy-tapahtuma ajetaan silloin kuin instanssi tuhotaan. Sitä käytetään monesti efektien luomisessa silloin, kun pelihahmo on tuhottu. (Yoyo Games 2017k, viitattu 28.9.2017.)

Alarm-tapahtuma

Alarmit toimivat kuin sekuntikellot. Niiden sisällä oleva koodi ajetaan vasta sitten, kun niihin asetettu aika (steppeinä) on mennyt loppuun. Alarmin kutsuminen tapahtuu toisesta tapahtumasta säätämällä kyseisen alarmin laukaisuajan. (Yoyo Games 2017l, viitattu 28.9.2017.) Kuvio 2 on esimerkki step-tapahtumasta.



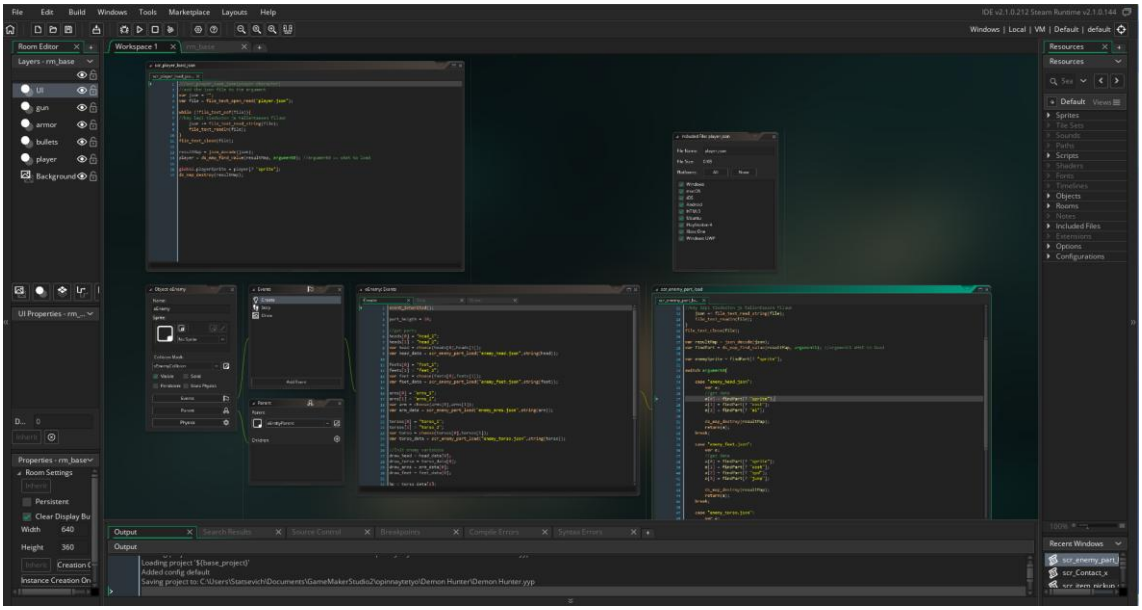
KUVIO 2. Step-tapahtuma

Draw-tapahtuma

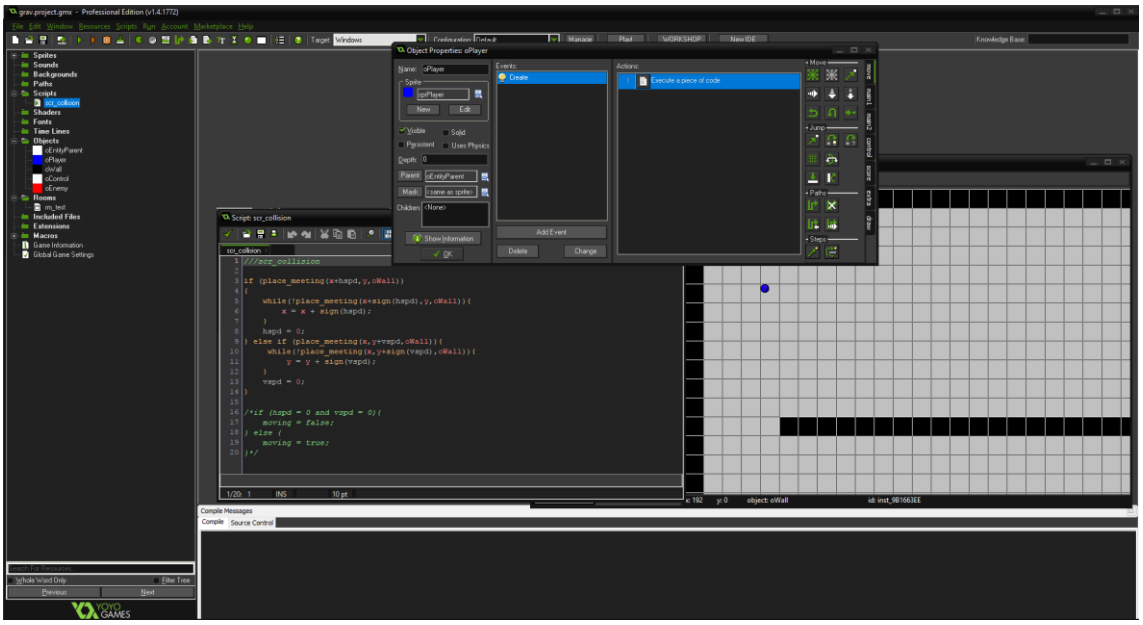
Draw-tapahtumassa hallinnoidaan asioiden piirtoa näytölle. Draw tapahtumassa voi ajaa muutakin koodia, mutta se ei ole selkeyden takia kannattavaa. (Yoyo Games 2017m, viitattu 28.9.2017.)

2.2 Game Maker 2:n uudet omaisuudet

Game Maker 2:een on lisätty uusia ominaisuuksia. Isoin muutos on käyttöliittymän päivitys. Uusi työtila on helposti luettavissa, kun koodi-ikkunat linkittyvät toisiinsa ja asioiden suhteet on helppo nähdä. Käyttöliittymä on muokattavissa, joten käyttäjä voi tehdä siitä mielensä mukaisen. (Yoyo Games, 2017. Viitattu 28.9.2017) Näkymä on huomattavasti selkeämpi kuin Game Maker Studiossa, jossa kaikki asiat avautuivat uuteen ikkunaan ja asioiden suhteita ei voinut nähdä muuta kuin koodista katsomalla. Game Maker 2:n käyttöliittymä näkyy kuviossa 3 ja Game Maker Studion työtila näkyy kuviossa 4.



KUVIO 3. Game Maker 2:n Työtila ja chain view

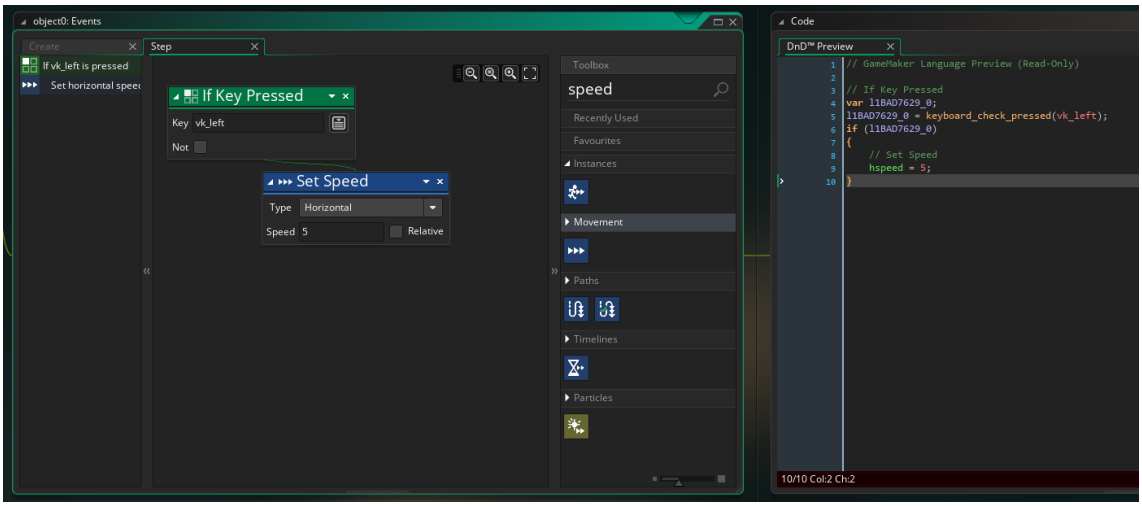


KUVIO 4. Game Maker Studion Työtila ja työikkunat

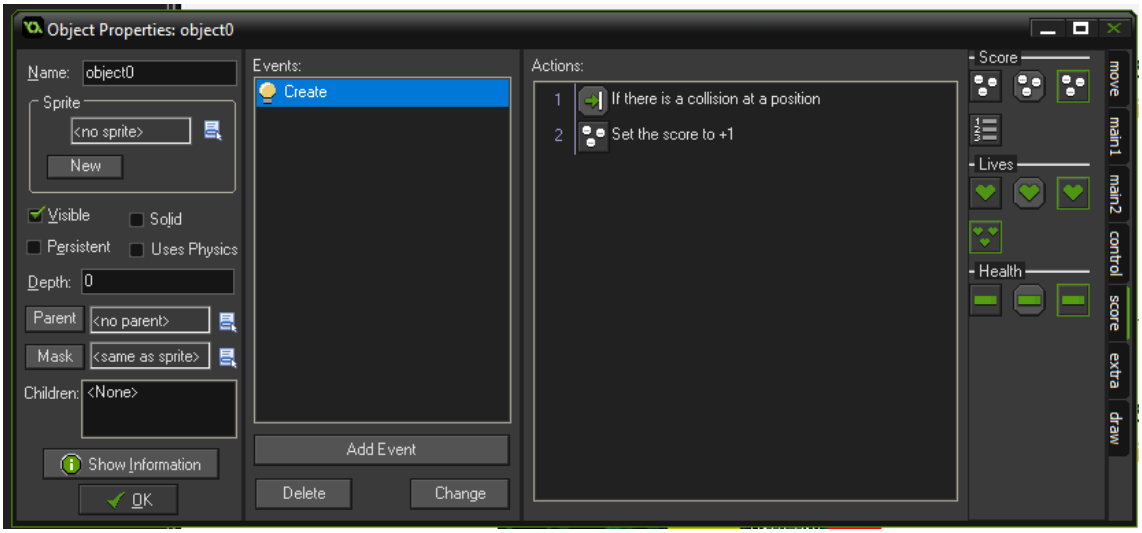
Game Maker 2:ssa on uusi room editor. Editointi perustuu uudessa Game Makerissa tasoihin, kun taas alkuperäisessä Game Makerissa se perustui pelkästään syvyyteen. (Yoyo Games, 2017e, Viitattu 25.9.2017.) Uusi tileset editori mahdollistaa omien brushien eli harjojen tekemisen. Autotile-ominaisuus on myös lisätty. Autotile mahdollistaa tilejen piirtämisen siten, että tietokone tunnistaa automaattisesti kulmat, eikä käyttäjän tarvitse itse vaihtaa aina piirrettävää tileä kulmien kohdalla. Game Maker 2 tileset editorissa voi myös piirtää animoituja tilejä. (Yoyo Games 2017f, Viitattu 25.9.2017.) Image editor on päivitetty Game Maker 2:ssa. Uutena ominaisuutena on tasoihin perustuva editointi. Image editorissa animaatioita voi käsitellä reaaliajassa. (Yoyo Games 2017g, Viitattu 25.9.2017.)

2.3 Drag and Drop -ominaisuus

Drag and drop -ominaisuus on muutettu toimimaan node pohjalta, jolloin loogisia rakenteita on helpompi seurata. Nodeilla tarkoitetaan tietorakennetta, joka linkittyy muihin nodeihin muodostaen tietoverkon. Käyttäjä voi myös katsoa minkälaista koodia nodejen sisällä live preview -tilassa (kuvio 5). Drag and Drop projektin voi muuttaa myöhemmin suoraan Game Maker Language projektiksi. (Yoyo Games 2017h, viitattu 25.9.2017). Game Maker Studion drag and drop –ominaisuus on kuvattuna kuviossa 6.



KUVIO 5. Game Maker 2 nodeihin perustuva drag and drop ja live preview -tila



KUVIO 6.Game Maker Studio drag and drop

Game Makerin eri versioden välillä suurin muutos on tapahtunut visuaalisessa ilmeessä ja uusien funktioiden muodossa. Perusasiat ovat pysyneet samana, ja kuilu näiden kahden pelimoottorin välillä on pieni. (Spalding 2017, viitattu 25.9.2017.) Alkuperäisen Game Makerin käyttöliittymä on nykystandardeilla hyvinkin kankea ja epäkäytännöllinen, kun taas Game Maker 2 onnistuu täyttämään nykypäivän käytettävyyden ja visuaalisuuden vaatimukset.

2.4 Game Maker Studio projektin siirtäminen Game Maker 2:een

Game Maker Studio projektin pystyy siirtämään suoraan Game Maker 2:lle. Projekti vaatii kuitenkin jotain muutoksia, koska Game Maker 2:ssa on muutettu joitain funktioita. Game Maker 2 on

myös uusia funktioita. Game Maker 2 luo automaattisesti yhteensopivuusskriptejä, jotka hoitavat muutosten käsittelyn. (Yoyo Games, 2017o. Viitattu 12.10.2017.) Projektin siirtäminen versioiden välillä ei ole välttämättä järkevää, koska Yoyo Games on ilmoittanut, että yhteensopivuusongelmia voi olla vaikka yhteensopivuusskriptejä on tehty. On turvallisempaa tehdä jo olemassa oleva projekti valmiiksi Game Maker Studiolla, kuin siirtää se Game Maker 2:een.

2.5 Game Maker Language

Game Maker käyttää omaa ohjelmointikieltä, jota kutsutaan Game Maker Languageksi. Game Maker Language perustuu C-ohjelmointikieleen. (Yoyo Games 2017d, viitattu 25.9.2017.) Game Maker Language on pysynyt lähes samanlaisena kuin edellisessä versiossa. Huomattavin muutos on instance_create funktion muuttuminen instance_create_layer-funktioksi. (Spalding 2017, viitattu 25.9.2017.) Edellä mainittu ero on suuri siitä syystä, että se on yksi käytetyimmistä funktioista Game Makerissa. Suurta muutosta Game Maker Languagessa ei ole tapahtunut versioiden välillä. Tämä madaltaa kehittäjien kynnystä vaihtaa uuteen versioon.

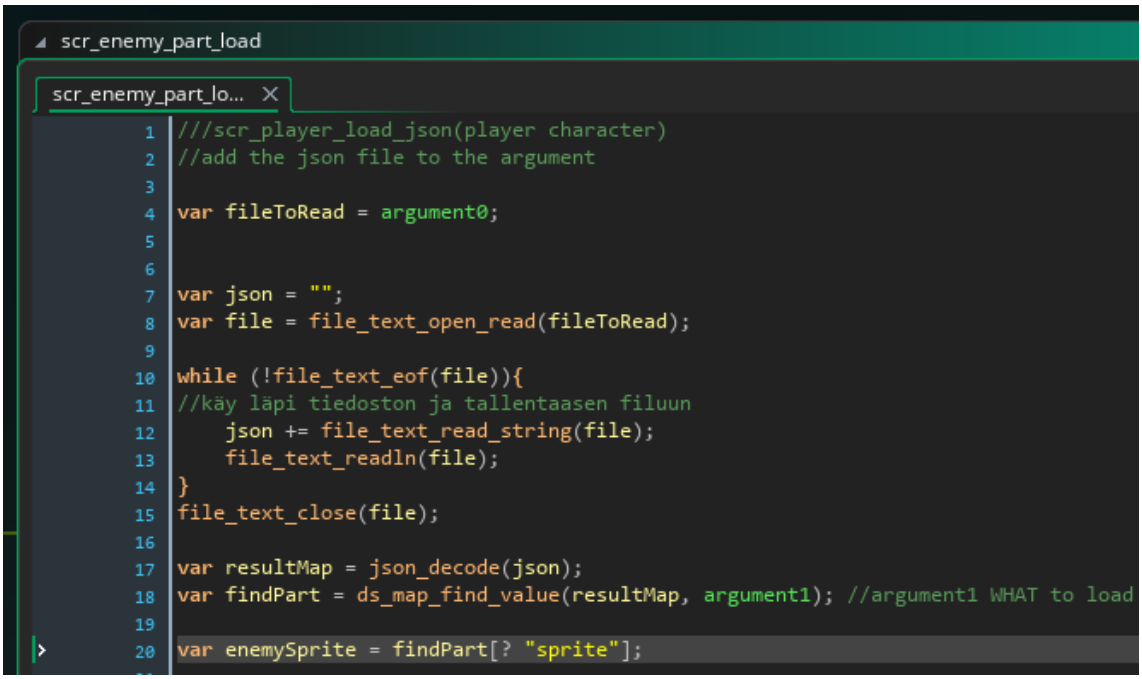
3 JSON JA GAME MAKERIN TIETORAKENTEET

Vihollisten ja pelaajan tiedot haetaan JSON-tiedostosta. JSON tulee sanoista JavaScript Object Notation, ja siihen pystyy tallentamaan tietoa hyvin organisoidusti. JSON on ihmisille ja tietokoneelle helposti luettavissa olevassa muodossa. (Copterlabs 2017, Viitattu 28.9.2017.) Esimerkki pelissä käytettävästä JSON-tiedostosta kuviossa 7.

```
{
        "torso_1":{
                "sprite": "sTorso1",
                "hp" : 10,
                "resistance":"",
                "cost": 10
                },
        "torso_2":{
                "sprite": "sTorso2",
                "hp" : 15,
                "resistance":"",
                "cost": 20
                },
        "humanoid":{
                "sprite": "sTorso_humanoid",
                "hp" : "15",
                "resistance":"",
                "cost": 20
                },
        "metal_hull":{
                "sprite": "sTorso_metal_hull",
                "hp" : "15",
                "resistance":""
                },
        "elephant":{
                "sprite": "sTorso_elephant",
                "hp" : "15",
                "resistance":"",
                "cost": 20
                },
        "deer":{
                "sprite": "sTorso_deer",
                "hp" : "15",
                "resistance":"",
                "cost": 20
                },
```

KUVIO 7. Pelin JSON-tiedosto.

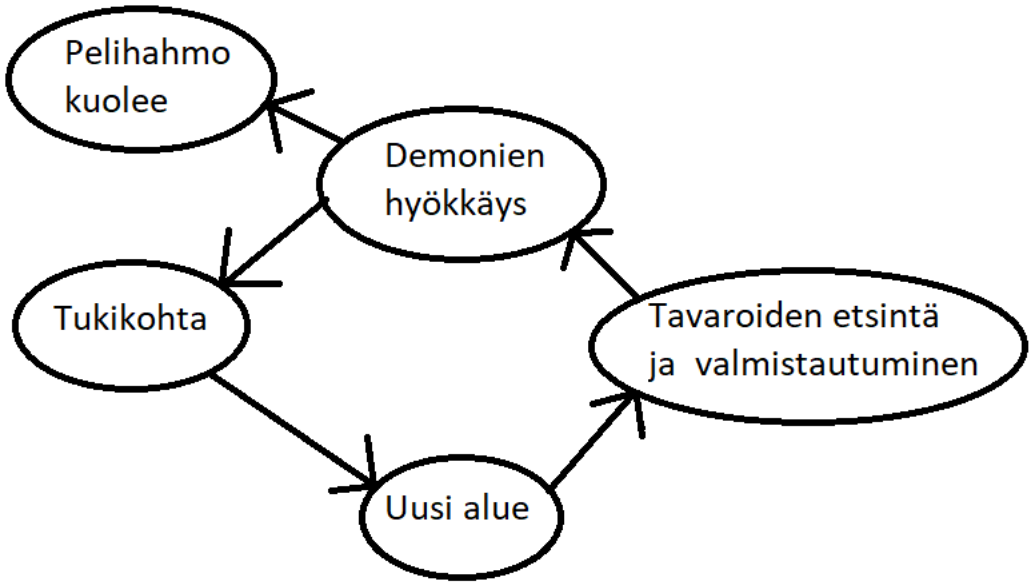
Game Makerissa JSON-tiedostot avataan json_decode()-funktiolla, joka muuttaa JSON-tiedoston DS mapiksi (Yoyo Games 2017n, viitattu 28.9.2017). Game Maker ei siis suoraan osaa lukea JSON-tiedostoa sellaisenaan vaan se pitää muuttaa ennen käyttöönottoa. DS map on tietorakenne, jossa data on pareittain. Tietoa haetaan avaimella, joka palauttaa avainparia vastaavan arvon. (Yoyo Games 2017i, viitattu 28.9.2017.) JSON-tiedostoon on helppo lisätä uusia asioita. Systeemi on rakennettu dynaamiseksi, mikä palvelee kehittäjää ja nopeuttaa työn tekemistä. Kuviossa 8 on skripti, jossa ladataan vihollishahmon tietoja JSON-tiedostosta.



KUVIO 8. Vihollisen osien lataaminen

4 PELIN KULKU JA MEKANIIKAT

Pelin työnimenä toimii Demon Survivor. Pelin genre on roguelike. Roguelike on peligenre, jossa pelaajalla on vain yksi elämä. Pelaajan kuollessa alkaa peli alusta. Pelin teema sijoittuu maailmanlopun jälkeiseen maailmaan, joka kuhisee demoneja. Pelaaja aloittaa pelin päivät aina tukikohdasta. Pelaaja valitsee varusteensa ja hahmonsa ennen uudelle alueelle siirtymistä. Alueella pelaajalla on aikaa kerätä esineitä ja asentaa suojia ja ansoja. Odotusajan loputtua alkaa alueelle tulla demoneja, jotka pelaajan täytyy tuhota jatkaakseen seuraavaan päivään. Peli ydinsilmukka on kuviossa 9. Alueiden vaikeustaso määritellään vaikeuspisteiden perusteella, eli difficulty pool -pisteillä. Pisteet vaikuttavat vihollisten ominaisuuksiin, määrään ja siihen, millaisia rakennelmia alue sisältää. Joka päivä on edellistä vaikeampi, ja uusia vihollisia tulee pelin edetessä. Pelaaja saa kokemuspisteitä, joilla pystyy hankkimaan lisäetuja pelihahmolle.



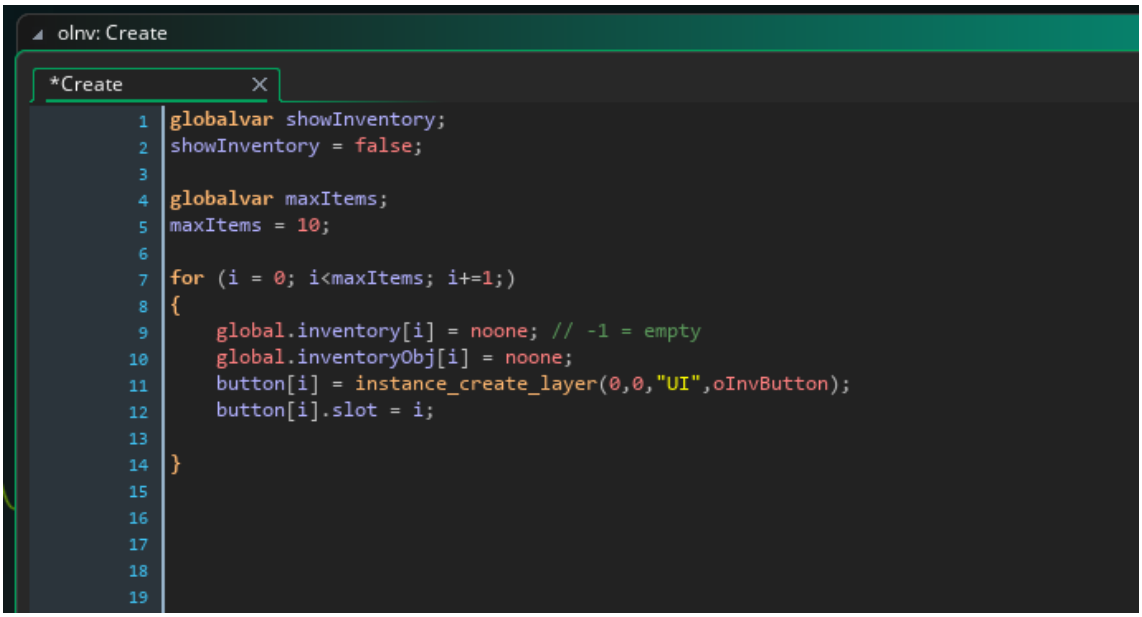
KUVIO 9. Pelin ydinsilmukka

Pelaajalla voi olla erilaisia suojavarusteita, aseita ja apuvälineitä. Suojavarusteet vähentävät vastaanotettua vahinkoa demoneilta. Pelissä on erilaisia aseita. Aseet koostuvat osista jotka vaikuttavat aseen käyttäytymiseen. Esimerkiksi piippu vaikuttaa tarkkuuteen ja aseen perä vaikuttaa tähtäämiseen.

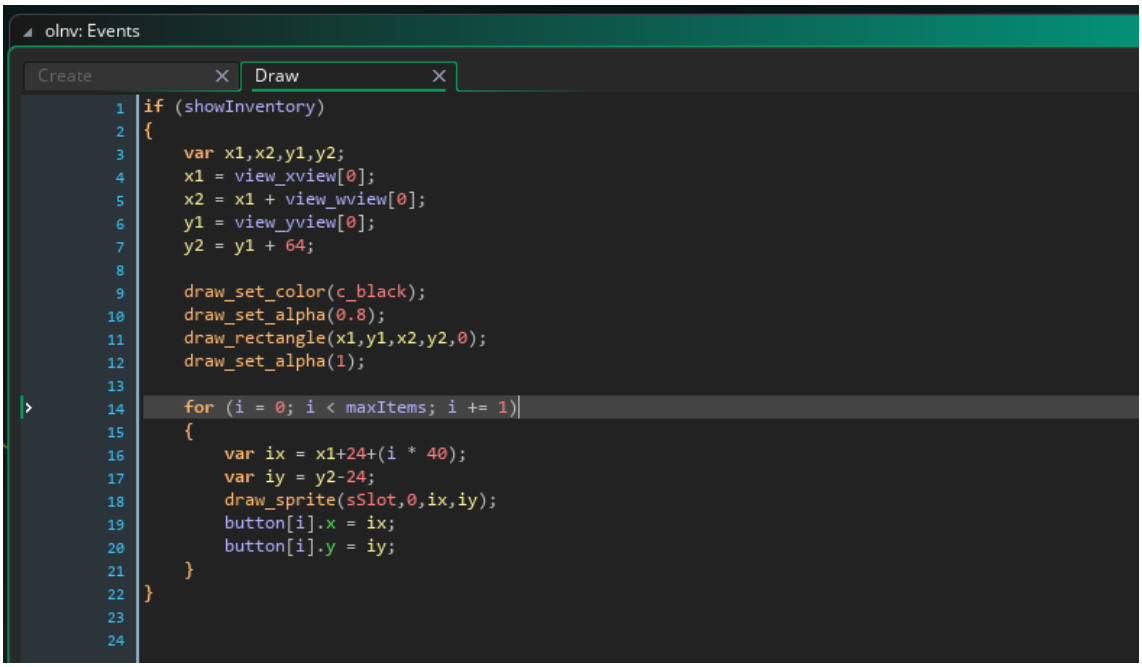
5 INVENTORY- JA EQUIPMENT-SYSTEEMIT

Peliin tehdyt Inventory-, craft- ja equipment-systeemit perustuvat YouTube-käyttäjä Shaun Spaldingin tekemiin videoihin:”Game Maker Studio: Inventory tutorial” ja “Game Maker Studio: Inventory with Mouse Control”. Systeemeissä on otettua mallia Spaldingin tekemistä videoista, mutta ne ovat myös muokattu pelin tarkoitukseen sopivaksi.

Inventory-systeemillä tarkoitetaan pelaajan ”reppua”, minne tallennetaan pelimaailmasta poimittuja objekteja. Inventory-systeemi rakennetaan listan pohjalle, ja se on helposti mukautettavissa ja laajennettavissa muihinkin tarkoituksiin (Spalding, 2014). Listan paikkoihin tallennetaan objektin tiedot, ja pelaajalle näytetään objektin kuva pelin käyttöliittymässä tai lyhyemmin UI:ssa. Inventory-systeemiä pelissä hallinnoi objekti nimeltä oInv. Kuvio 10 esitellään oInv-objektin create-tapahtumaa.

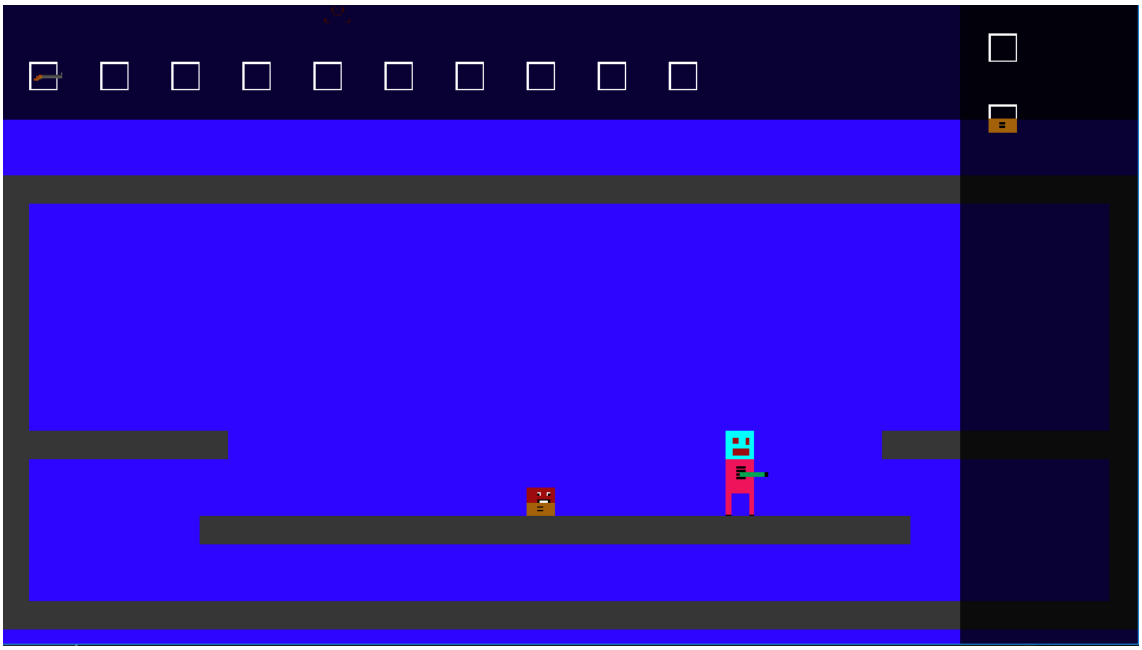


KUVIO 10. oInv create-tapahtuma



KUVIO 11. oInv draw-tapahtuma

Equipment-systeemin toimintaperiaate on samanlainen kuin inventory, mutta siinä slotit toimivat pelaajan varustepaikkoina. Pelaajalla voi olla hallussa suojavaruste ja ase. Suoja vaikuttaa pelaajan vastaanottamaan vahinkoon.



KUVIO 12. Inventory- ja Equipment-systeemi pelissä. Pelaajalla on päällä suojavaruste

6 PROSEDURAALINEN GENEROINTI

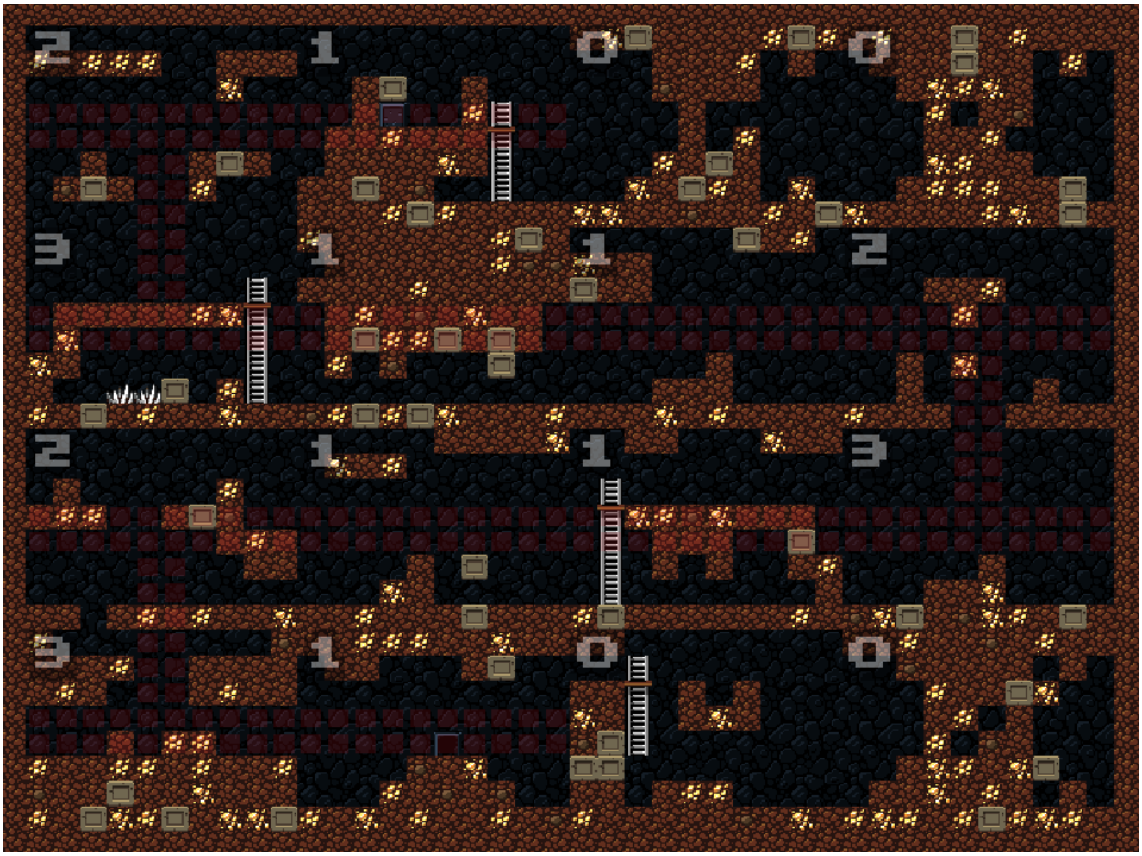
6.1 Proseduraalinen generointi peleissä

Tarkastellessa Wikipedian listaa peleistä, missä on proseduraalinen generointi, voi huomata, että sellaisten pelien määrä on noussut huimasti viime vuosina (Wikipedia, 2017. Viitattu 18.10.2017). Proseduraalinen generointi toimii algoritmien pohjalta, joilla sisältöä luodaan peliin. Toisin sanottuna satunnaisuus toimii ohjeiden perusteella. Satunnaisgeneraatio lisää peleihin vaihtelevuutta ja se estää sen, että pelin pystyisi opettelemaan ulkoa, koska jokainen pelikerta on erilainen. (Muropaketti, 2016. Viitattu 18.10.2017.) Tämä nostaa pelin uudelleenpeluuarvoa.

No Man's Sky on videopeli, jossa on yli 18 kvintiljoonaa planeettaa. Kehittäjät käyttivät proseduraalista generaatiota kaikkien planeettojen luomiseen. (Co.Design, 2015. Viitattu 18.10.2017.) Pelissä on myös proseduraalisesti generoitu eläimiä ja kasveja (PBS Game/Show, 2015a. Viitattu 18.10.2017). Minecraft on peli, jossa pelaaja kerää resursseja pelimaailmasta ja voi rakentaa asioita mielikuvituksen rajoissa. Minecraft hyödyntää myös proseduraalista generointia pelimaailman luomisessa. Pelimaailman luonti perustuu Perlin Noise algoritmiin. (PBS Game/Show, 2015a. Viitattu 18.10.2017.) Minecraftin proseduraalisesti luotu maailma näkyy kuviossa 13.

KUVIO 13. Minecraftin proseduraalisesti generoitu pelimaailma.

Spelunky on Game Maker Studiolla tehty peli, joka hyödyntää myös proseduraalista generointia. (Mark Brown, 2016. Viitattu 30.10.2017.) Esimerkki Spelunkyn pelimaailmasta on kuviossa 14.

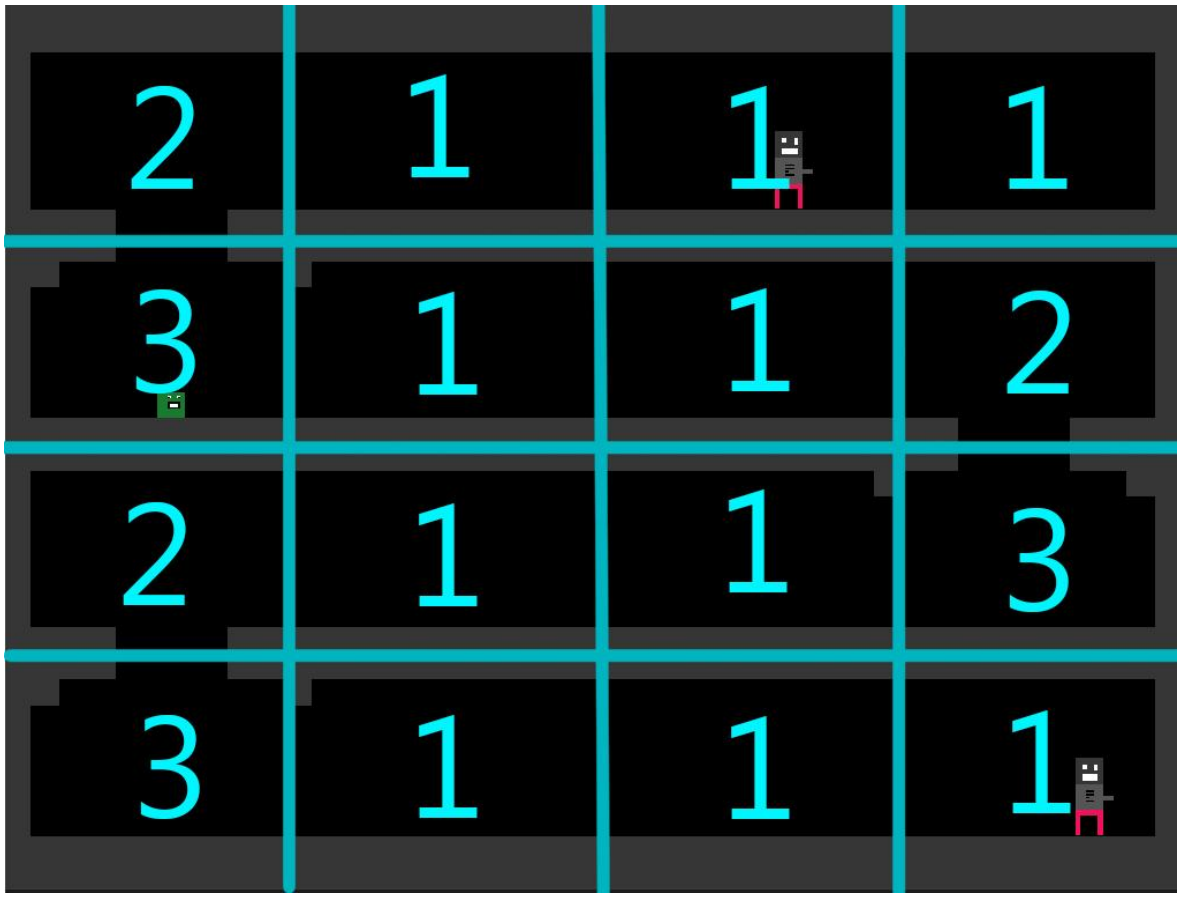


KUVIO 14. Spelunkyn proseduraalisen generaation luoma kartta.

6.2 Kenttien proseduraalinen generointi

Kenttien proseduraalinen generointi on tehty samankaltaiseksi kuin Spelunky-pelissä. Se on sekoitus itse tehtyä ja satunnaisuutta. Spelunkyssa yksi kenttä on tehty 16 huoneesta, jotka luodaan 4x4-ruudukolle. Kentät alkavat aina ylhäältä ja loppuvat alimmalle tasolle. Algoritmi luo satunnaisen suoran reitin alku- ja loppupisteen välille ja täyttää loput tyhjät ruudut muilla huoneilla. (Mark Brown, 2016. Viitattu 30.10.2017.) Tällainen malli valittiin, koska se sopii pelin ideaan paremmin kuin enemmän satunnaiseen luottava algoritmi ja se on suhteellisen helppo toteuttaa. Kenttien muodostumistapaa on myös helppo muuttaa vain muuttamalla reitinluonti algoritmia ja tekemällä erilaisia kenttiä. Myös huoneiden määrää ja kokoa voi muuttaa helposti. Toimeksiantaja hyötyy tällaisesta kenttägeneraattorista huomattavasti, koska sitä voidaan käyttää tulevaisuudessa muissakin peliprojekteissa.

Kentän luomista hallinnoi oLevelCreator-objekti. Siinä asetetaan ruudukon huoneiden määrä korkeus- ja leveyssuunnassa. Huoneen koko määritetään laskemalla huoneiden määrät suunnassaan ja kertomalla se ruudukon koolla. (Gloomy Toad Studios, 2017a. Viitattu 30.10.2017.) Huoneita hallinnoidaan 2D-listassa, johon tallennetaan huonetyypit. Huonetyypit määräytyvät numeroilla. Tyypin 1 huoneeseen on pääsy oikealta ja vasemmalta. Tyypin 2 huoneeseen on pääsy oikealta, vasemmalta ja alas. Tyypin 3 huoneeseen on pääsy ylhäältä, oikealta ja vasemmalta. (Gloomy Toad Studios, 2017b. Viitattu 30.10.2017.)



KUVIO 15. Generoitu kenttä ja sitä vastaavat arvot listassa.

Huonegeneroinnissa käytettävä algoritmi arpoo ensimmäisen huoneen satunnaisesti ja alkaa sen jälkeen luoda reittiä ylhäältä alas. Se arpoo suunnan oikealle, vasemmalle tai alas. Aina kun algoritmi huomaa, että se on kentän oikealla tai vasemmalla reunalla, se luo 2 tyypin huoneen. Algoritmi tarkistaa, että huonetyyppi 2:n alle luodaan aina huonetyyppi 3. Näin algoritmi varmistaa, että pelaajalla on pääsy kaikille tasoille. Jos huoneeseen jää tyhjiä paikkoja, niihin asetetaan satunnainen huonetyyppi. (Mark Brown, 2016. Viitattu 30.10.2017; Gloomy Toad Studios, 2017a. Viitattu 30.10.2017.)

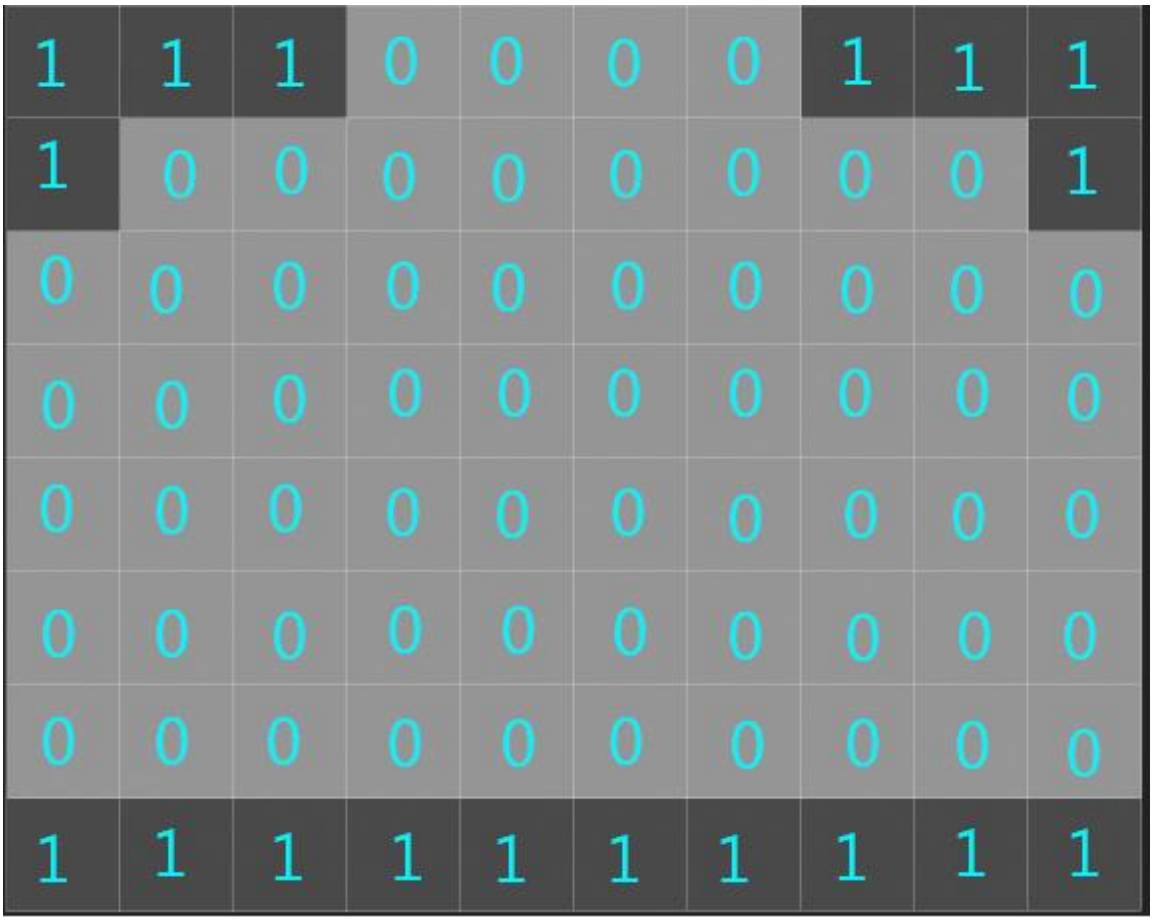
Huoneiden data on talletettu tietorakennelistaan. Generate level -skripti luo ensin reunat kentälle, jonka jälkeen se tarkistaa huonetyypin sections-listan jokaisesta kohdasta. Kun huonetyyppi tiedetään, valitaan satunnainen huone joka vastaa kyseistä huonetyyppiä. Lista käydään läpi kahdella sisäkkäisellä silmukalla ja jokaiseen kohtaan luodaan huonetyyppiä vastaava huone. (Gloody Toad Studios 2017e. Viitattu 31.10.2017.)

```
///Generate Level

//generate border for your room
for (_i = 0; _i < room_width; _i += gridSize)
{
    instance_create_layer(_i,0,"Instances",oWall);
    instance_create_layer(_i,room_height - gridSize,"Instances",oWall);
}

for (_i = gridSize; _i< room_height - gridSize; _i += gridSize)
{
    instance_create_layer(0, _i, "Instances", oWall);
    instance_create_layer(room_width - gridSize, _i, "Instances", oWall);
}


var sectionStringData = "";

for (_y = 0; _y < ySections; _y++)
{
    for (_x = 0; _x < xSections; _x++)
    {
        sectionStringData = loadRandomSection(sections[_x, _y]);
        createSectionFromString(_x, _y, sectionStringData);
    }
}
```

KUVIO 16. Huoneiden lataaminen ja asettaminen pelimaailmaan

6.2.1 Huoneiden luonti

Jokainen huone peliin on käsin tehty. Huoneet luodaan Game Maker 2 omassa room editorissa. Kun haluttu huone on tehty, sinne asetetaan lopuksi objekti oTemplateController, joka ajetaan huoneessa. Objekti oTemplateController tarkistaa huoneen jokaisen ruudukon kohdan sisältämän objektin. oTemplateController luo huoneista string tiedostoja, jotka sisältävät kentän datan. Data tallennetaan data structureen, josta se haetaan kentän generoinnin aikana. (Gloomy Toad Studios, 2017c. Viitattu 30.10.2017.) Huoneen voi myös halutessaan tehdä tekstitiedostona. Silloin täytyy olla tarkkana, että merkkijono on oikean pituinen ja sisältää ainoastaan oikeita merkkejä. Kuviossa 17 esitetään, miten tieto on siirretty kentän osaan. 1 Tarkoittaa seinää ja 0 tyhjää. Kuviossa 18 esitetään encode-skripti, jossa luodaan käsin tehdystä huoneesta tekstitiedosto. Kuvio 19 taas esittää sen, miten tekstitiedostosta luodaan kentän alue pelimaailmaan.



KUVIO 17. Huoneen data

```
///EncodeTemplateToString
//Save level data to string

gridSize = 16;
xTiles = room_width/gridSize;
yTiles = room_height/gridSize;

var levelData = "";

//Cycle through grid
for (var _y = 0; _y<yTiles;_y++)
{
    for(var _x = 0; _x<xTiles; _x++)
    {
        //check if instance is in that position
        var inst = instance_position(_x*gridSize,_y*gridSize,all);

        if (inst != noone)
        {
            inst = inst.object_index;
        }

        //level to string
        switch (inst)
        {
            case noone:
                levelData += "0";
            break;
            case oWall:
                levelData += "1";
            break;
            case oLadder:
                levelData += "L";
            break;
            case oRandomBlock:
                levelData += "R";
            break;
            case oChunk:
                levelData += "C";
            break;
        }
    }
}
//show_message(levelData);
saveDir = get_save_filename("*.txt", room_get_name(room) + ".txt");
file = file_text_open_write(saveDir);
file_text_write_string(file, levelData);
file_text_close(file);
```

KUVIO 18. Huoneen encode-skripti

```
///create sections from string

var _currentXSection = argument0;
var _currentYSection = argument1;
var _sectionStringData = argument2;
var typeOfObject;

for (var c = 1; c < string_length(_sectionStringData) + 1; c++)
{
    switch (string_char_at(_sectionStringData, c))//in this function 1st index is number 1 not 0
    {
        case "0"://do nothing
        continue;

        case "1":
            typeOfObject = oWall;
        break;

        case "L":
            typeOfObject = oLadder;
        break;

        case "R":
        {
            if (choose(true,false))
            {
                typeOfObject = oRandomBlock;
                break;
            } else continue;
        }
        case "C":
            typeOfObject = oChunk;
        break;
    }
            instance_create_layer((_currentXSection * sectionWidth) + (((c - 1) mod sectionTilesX) * gridSize) + gridSize,
                                  ((_currentYSection * sectionHeigth) + (floor((c-1) / sectionTilesX) * gridSize) + gridSize),
                                   "Instances", typeOfObject);
}
```

KUVIO 19. Skripti createSectionFromString.

6.3 Vihollisten proseduraalinen generointi

Vihollisten satunnaisgenerointia hallinoidaan oEnemySpawner-objektissa. Objektissa on alarm-tapahtuma, jossa luodaan vihollisten ominaisuudet ja synnytetään viholliset pelimaailmaan. Alarm-tapahtumalla hallinnoidaan vihollisten hyökkäysaaltoja. Aaltosysteemiä kontrolloidaan oEnemySpawnerissa. Siinä tarkistetaan, onko vihollishahmoja huoneessa, ja jos huoneessa ei ole vihollishahmoja, niin asetetaan ajastimeen haluttu aika. Ajan loputtua syntyy uusi vihollisaalto. Aaltojen hallinta näkyy kuviossa 20. Kellon loputtua oEnemySpawnerin-objektin alarm-tapahtuma laukeaa ja hoitaa vihollisten spawnaamisen. Spawnaamisella tarkoitetaan vihollisten syntymistä pelimaailmaan. Peli arpoo mitä osia käytätetään ja lataa ne JSON-tiedostosta. Arvottu osa lähetetään skriptiin scr_enemy_part_load() (kuvio 22), jossa luetaan mistä ja mitä pitää hakea. Tiedot tallennetaan listaan ja skriptin tiedot palautetaan muuttujaan (kuvio 21).

```
//wave spawner
if (!instance_exists(oEnemy) and alarm_triggered = true)
{
    alarm[0] = room_speed * 3;
    alarm_triggered = false;
}
```

KUVIO 20. Vihollisten tarkistaminen

```
//get parts
heads[0] = "head_1";
heads[1] = "head_2";
var head = choose(heads[0],heads[1]);
var head_data = scr_enemy_part_load("enemy_head.json",string(head));

feets[0] = "feet_1";
feets[1] = "feet_2";
var feet = choose(feets[0],feets[1]);
var feet_data = scr_enemy_part_load("enemy_feet.json",string(feet));

arms[0] = "arms_1";
arms[1] = "arms_2";
var arm = choose(arms[0],arms[1]);
var arm_data = scr_enemy_part_load("enemy_arms.json",string(arm));

torsos[0] = "torso_1";
torsos[1] = "torso_2";
var torso = choose(torsos[0],torsos[1]);
var torso_data = scr_enemy_part_load("enemy_torso.json",string(torso));
```

KUVIO 21. Vihollishahmon luonti

```
var resultMap = json_decode(json);
var findPart = ds_map_find_value(resultMap, argument1); //argument1 WHAT to load

var enemySprite = findPart[? "sprite"];

switch argument0{

    case "enemy_head.json":
        var a;
        //get data
        a[0] = findPart[? "sprite"];
        a[1] = findPart[? "cost"];
        a[2] = findPart[? "ai"];

        ds_map_destroy(resultMap);
        return(a);
    break;

    case "enemy_feet.json":
        var a;
        //get data
        a[0] = findPart[? "sprite"];
        a[1] = findPart[? "cost"];
        a[2] = findPart[? "spd"];
        a[3] = findPart[? "jump"];

        ds_map_destroy(resultMap);
        return(a);
    break;

    case "enemy_torso.json":
        var a;
        //get data
        a[0] = findPart[? "sprite"];
        a[1] = findPart[? "cost"];
        a[2] = findPart[? "hp"];
        ds_map_destroy(resultMap);
        return(a);
    break;

    case "enemy_arms.json":
        var a;
        //get data
        a[0] = findPart[? "sprite"];
        a[1] = findPart[? "cost"];
        a[2] = findPart[? "attack"];
        ds_map_destroy(resultMap);
        return(a);
    break;
}
```

KUVIO 22. Funktio scr_enemy_part_load().

Jokaisella osalla on vaikeusarvo, jotka lasketaan yhteen. Vihollisten syntymismäärä lasketaan käytettävien pisteiden määrä jaettuna vihollishahmon vaikeusarvolla. Saatu tulos pyöristetään alaspäin lähimpään kokonaislukuun floor()-funktiolla. Vaikeuspisteiden laskeminen näkyy kuviossa 23.

```
//count the enemy cost
var head_cost = real(head_data[1]);
var torso_cost = real(torso_data[1]);
var arm_cost = real(arm_data[1]);
var feet_cost = real(feet_data[1]);

//count amount of enemies
total_cost = head_cost + torso_cost + arm_cost + feet_cost;

var n = floor(difficultyPool/total_cost)
show_debug_message("Spawn count: " + string(n));
show_debug_message("Enemy cost is: " + string(total_cost));
```

KUVIO 23. Vaikeuspisteiden laskeminen

6.3.1 Vihollisten asettaminen pelimaailmaan

Osien arpomisen jälkeen ja vaikeustason määrityksen jälkeen vihollishahmot luodaan oEnemySpawner-objektissa. Kaikki tieto asetetaan vihollishahmon tietoihin, jota sitten käytetään sen muussa toiminnallisuudessa. Vihollisten luonti näkyy kuviossa 24.

```
//spawn n amount of enemies determined by the difficulty cost
repeat (n)
{
    //get free position
    var xxx,yyy;
    xxx = random(room_width);
    yyy = random(room_height);
    //create new enemy and apply all the data to it
    var new_enemy = instance_create_layer(xxx,yyy,"enemy",oEnemy);
    new_enemy.draw_head = head_data[0];
    new_enemy.draw_torso = torso_data[0];
    new_enemy.draw_arms = arm_data[0];
    new_enemy.draw_feet = feet_data[0];
    new_enemy.hp = real(torso_data[2]);
    new_enemy.ai = head_data[2];
    new_enemy.movespd = real(feet_data[2]);
    new_enemy.jump_spd = real(feet_data[3]);
    new_enemy.melee_attack = arm_data[2];
}
```

KUVIO 24. Vihollisten luonti.

Vihollishahmo asetetaan satunnaiseen paikkaan oEnemySpawnerissa, mutta ongelma on se, että vihollishahmo voi syntyä seinän (oWall) sisään. Tällöin vihollishahmo ei pääse liikkumaan. Ongelma on ratkaistu siten, että vihollishahmon create tapahtumassa ajetaan skripti, joka siirtää vihollishahmoa niin kauan, kunnes se on sellaisessa paikassa, missä on vapaata. Lisäksi skripti vielä tarkistaa, että jos vihollishahmo syntyy ilmaan se siirrettään maahan. Skripti on kokonaisuudessaan kuviossa 25.

*scr_checkSpawnPl...

```
1  ///checks that spawn place is free and places enemy on ground
2  while (place_meeting(x,y,oWall))
3  {
4   x = random(room_width);
5   y = random(room_height);
6  }
7
8  do {
9      y += 1;
10 } until(place_meeting(x,y,oWall));
11 //makes sure that obj is on the wall not in it
12 y -= 1;
13
```

KUVIO 25. Vihollishahmon siirtäminen vapaaseen paikkaan ja oWall päälle.

Vihollishahmon piirtämistä hallinnoidaan vihollishahmon omassa draw-tapahtumassa. Vihollishahmolle asetetut osat ja niiden spritet haetaan muuttujasta. Piirtämisen ohjelma näkyy kuviossa 26.

```
3  var scale_x;
4  if (dir != 0)
5  {
6      scale_x = dir;
7  } else
8  {
9      scale_x = 1;
10 }
11
12 //draw head
13 draw_sprite_ext(asset_get_index(draw_head),0,x,y-(part_height),scale_x,1,0,c_white,1);
14 //draw torso
15 draw_sprite_ext(asset_get_index(draw_torso),0,x,y,scale_x,1,0,c_white,1);
16 //draw_legs
17 draw_sprite_ext(asset_get_index(draw_feet),0,x,y+(part_height),scale_x,1,0,c_white,1);
18 //draw hands
19 draw_sprite_ext(asset_get_index(draw_arms),0,x,y,scale_x,1,0,c_white,1);
```

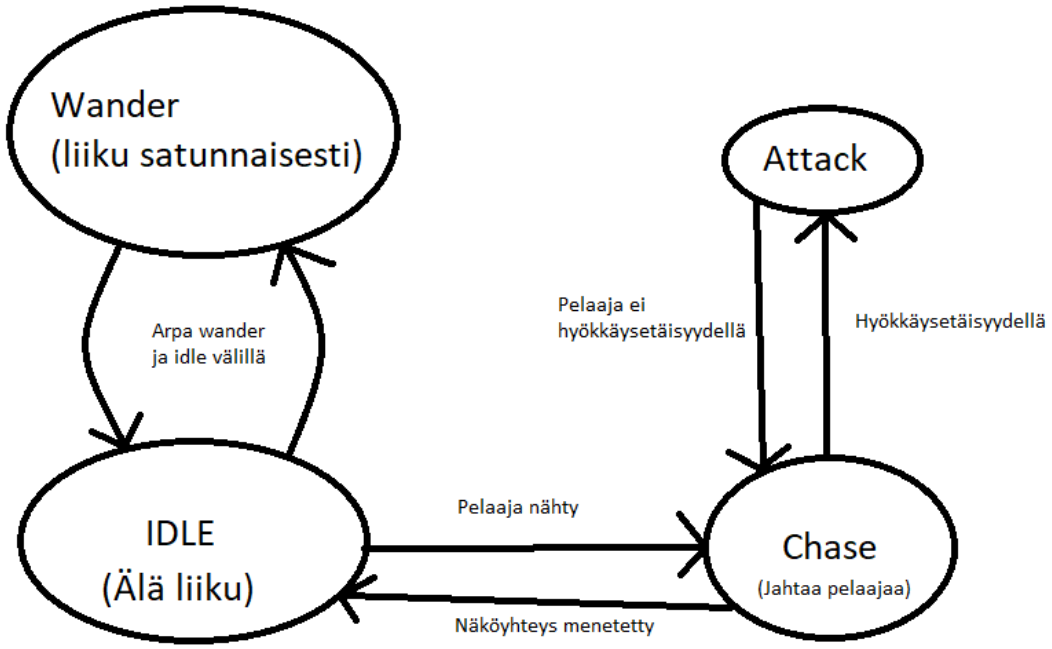
KUVIO 26. oEnemy draw-tapahtuma.

7 VIHOLLISTEN TEKOÄLY

Videopeleissä tekoäly on yksi suurimmista haasteista. Sen pitää pystyä toimimaan uskottavasti pelaajan silmissä tai muuten koko pelikokemus voi tuntua typerältä. Aloittaessani tekemään tekoälyä, aloin tutkimaan muutamia erilaisia algoritmeja joiden avulla reitinlaskentaa voisi suorittaa. Huomasin kuitenkin nopeasti, että kokonaisen algoritmisysteemin luominen olisi vienyt liikaa aikaa ja päätin luopua kyseisestä ideasta sen jälkeen, kun keskustelin asiasta toimeksiantajan kanssa. Ehdin ottaa selvää A*-algoritmista ja Dijkstran algoritmista ennen kuin päätin ajanpuutteen takia tekemään yksinkertaisemman tekoälyn. Isoimmaksi ongelmaksi edellä mainittujen algoritmien kanssa olisi pelissä oleva painovoima. Molemmat algoritmit toimivat helposti silloin, kun kaksiulotteisessa tilassa ei ole painovoimaa. Game Makerin omat polunetsintäfunktiot perustuvat A* algoritmiin (Yoyo Games 2017p, viitattu 15.10.2017).

7.1 Äärellinen automaatti

Vihollisten tekoäly luotiin äärellisellä automaatilla. Äärellisessä automaatissa tekoälyllä on erilaisia tiloja, jotka voivat olla vain yksi kerrallaan päällä (Tutsplus 2017, Viitattu 15.10.2017). Peleissä äärellisessä automaatissa tilat ovat toimintoja, jota vihollishahmo tekee. Kuviossa 27 havainnollistetaan vihollishahmon äärellisen automaatin tilat.



KUVIO 27. Vihollishahmon äärellisen automaatin tilat

8 POHDINTA

Opin ymmärtämään paremmin, mitä kaikkea suunnitteluvaiheessa pitää ottaa huomioon. Opinnäytetyön tekeminen herätti myös kiinnostusta ohjelmistojen suunnittelua kohtaan näiden havaintojen takia. En toiminut niin suunnitelmallisesti, kuin olisi pitänyt. Se näkyi siinä, että jouduin tekemään asioita välillä useamman kerran, koska en ollut ajatellut niitä loppuun asti. Kuitenkin mikään haaste, mikä työn aikana tuli eteen, ei osoittautunut ylitsepääsemättömäksi, vaan sain aina tehtyä jonkun ratkaisun. Tekoälyn luominen osoittautui vaikeimmaksi asiaksi, mitä tuli vastaan työtä tehdessä. Käytin paljon aikaa pelkkään tutkimustyöhön ja huomasin nopeasti, että hyvän tekoälyn tekemiseen aika ei riitä, ja sovimme toimeksiantajan kanssa, että teen vain yksinkertaisen tekoälyn demoon.

Indie-kehittäjillä ei ole yleensä paljoa resursseja käyttää pelin tekemiseen, ja hyvin tehty satunnaisgenerointi on hyvä tapa saada sisältöä ja vaihtelevuutta peliin. Oli mielenkiintoista tehdä satunnaisgeneraattori, joka mahdollistaa lukuisten erilaisten vihollisten ja kenttien luomisen. Otin paljon selvää tehdessäni kenttägeneraattoria, joka perustuu Spelunky pelin proseduraaliseen generointiin. Opin hyvin siinä käytetyt konseptit ja tulevaisuudessa pystyn hyödyntämään niitä myös omissa projekteissani. Vihollisten proseduraalisen generaation tein kokonaan itse ja olen siihen hyvin tyytyväinen. Se toki perustui pelisuunnitelmassa esitettyyn malliin, mutta sen toimimaan saaminen pelissä oli mukavaa ja sopivan haastavaa.

Selvitystyönä ollut Game Maker Studion ja Game Maker 2:n vertailussa päädyin siihen, että vielä ei ole mitään tarvetta vaihtaa uuteen versioon. Työn aikana sain paljon harmaita hiuksia, kun Game Maker 2 kaatuili. Pienen tutkimustyön jälkeen selvisi, että vika saattaa olla keväällä tulleessa Windows 10 -päivityksessä. Vastaavia ongelmia en havainnut, kun testasin Game Maker Studiota.

Olen tyytyväinen omiin oppimistuloksiini opinnäytetyössä. Sain hyvää rutiinia ohjelmointiin ja opin mitä ei kannata tehdä. Henkilökohtaiset oppimistavoitteet siis täyttyivät. Demo näyttää harmittavasti rumalta, koska minulla ei ollut graafikon tekemää grafiikkaa työssä. Tärkeintä kuitenkin on se, että demo toimii ja tuotokset helpottavat toimeksiantajaa nykyisessä ja tulevissa projekteissa.

LÄHTEET

Co. Design 2015. How 4 Designers Build A Game With 18.4 Quintillion Unique Planets. Viitattu 18.10.2017
https://www.fastcodesign.com/3048667/how-4-designers-built-a-game-with-184-quintillion-unique-planets

Copterlabs, 2017. JSON: What It Is, How It Works, & How to Use It. Viitattu 28.9.2017
https://www.copterlabs.com/json-what-it-is-how-it-works-how-to-use-it/

Darius Kazemi, 2017a. Spelunky Generator Lessons Part 1: Generating the Solution Path. Viitattu 18.10.2017
http://tinysubversions.com/spelunkyGen/

Darius Kazemi, 2017b. Spelunky Generator Lessons Part 2: Generating the Rooms. Viitattu 18.10.2017
http://tinysubversions.com/spelunkyGen2/

Extra Credit, 2015. Procedural Generation – How Games Create Infinite Worlds.
https://www.youtube.com/watch?v=TgbuWfGeG2o

Gloomy Toad Studios, 2017a. Build Spelunky Level Generator in Gamemaker part 1. Viitattu 30.10.2017
https://www.youtube.com/watch?v=ouh7EZ5Qh9g&t

Gloomy Toad Studios, 2017b. Build Spelunky Level Generator in Gamemaker part 2. Viitattu 30.10.2017
https://www.youtube.com/watch?v=1QRuQgKIJxM

Gloomy Toad Studios, 2017c. Build Spelunky Level Generator in Gamemaker part 3. Viitattu 30.10.2017
https://www.youtube.com/watch?v=gUv7KQgrbFI

Gloomy Toad Studios, 2017d. Build Spelunky Level Generator in Gamemaker part 4. Viitattu 30.10.2017
https://www.youtube.com/watch?v=d5jmix0wcIw&t

Gloomy Toad Studios, 2017e. Build Spelunky Level Generator in Gamemaker part 5. Viitattu 30.10.2017
https://www.youtube.com/watch?v=CmiP6amWLw0&t

Gloomy Toad Studios, 2017f. Build Spelunky Level Generator in Gamemaker part 6. Viitattu 30.10.2017
https://www.youtube.com/watch?v=J7T_LTNof4Y

Mark Brown, 2016. How (and why) Spelunky Makes its own Levels | Game Maker's Toolkit. Viitattu 30.10.2017
https://www.youtube.com/watch?v=Uqk5Zf0tw3o

Muropaketti, 2016. Proseduraalinen generointi on No Man's Skyn salaisuus – mutta mitä se on.
https://muropaketti.com/pelit/peliartikkelit/proseduraalinen-generointi-no-mans-skyn-salaisuus-mita/

PBS Game/Show, 2015a. How No Man's Sky Creates A Universe! Viitattu 18.10.2017
https://www.youtube.com/watch?v=2KlmY7zxAp0

PBS Game/Show, 2015b. How Minecraft Generates Such HUGE Worlds. Viitattu 18.10.2017
https://www.youtube.com/watch?v=8Ex_eHJ29iQ

Shaun Spalding, 2014. Game Maker Studio: Inventory Tutorial
https://www.youtube.com/watch?v=LwBC6kyTa0M&t

Shaun Spalding, 2015. Game Maker Studio: Inventory with Mouse Control
https://www.youtube.com/watch?v=AmTSaHqt5Xw&t

Shaun Spalding, 2017. GMS1 vs GMS2 – The 10 Biggest changes in GameMaker Studio 2
https://www.youtube.com/watch?v=7H5oCgA_JVQ

Steam Spy, 2017. Hyper Light Drifter. Viitattu 25.9.2017
https://steamspy.com/app/257850

Tutsplus, 2017. Finite-State Machines: Theory and Implementation. Viitattu 15.10.2017
https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867

Wikidot 2017. GameMaker Versions. Viitattu 25.9.2017
http://game-maker.wikidot.com/game-maker-versions

Wikipedia, 2017. List of games using procedural generation.
https://en.wikipedia.org/wiki/List_of_games_using_procedural_generation

Yoyo Games 2017a. The Create Event. Viitattu 19.9.2017,
https://docs.yoyogames.com/source/dadiospice/000_using%20gamemaker/events/create%20event.html

Yoyo Games 2017b. RoadMap. Viitattu 25.9.2017
https://help.yoyogames.com/hc/en-us/articles/231719448-RoadMap

Yoyo Games, 2017c. Showcase. Viitattu 25.9.2017
https://www.yoyogames.com/showcase

Yoyo Games 2017d. GameMaker studio 2. Viitattu 25.9.2017
https://www.yoyogames.com/gamemaker/features

Yoyo Games, 2017e. GameMaker Studio 2 – Room editor
https://www.youtube.com/watch?time_continue=102&v=GVb5py7QL3Q

Yoyo Games, 2017f. GameMaker Studio 2 – Tileset Editor
https://www.youtube.com/watch?v=R0t9j82-438

Yoyo Games, 2017g. GameMaker Studio 2 – Image editor
https://www.youtube.com/watch?time_continue=12&v=VRPln502FO4

Yoyo Games, 2017h. GameMaker Studio 2 – Drag and Drop
https://www.youtube.com/watch?v=ADlB0XFjsSk

Yoyo Games, 2017i. DS Maps. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/002_reference/data%20structures/ds%20maps/index.html

Yoyo Games, 2017j. The Step Event. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/000_using%20gamemaker/events/step%20event.html

Yoyo Games, 2017k. The Destroy Event. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/000_using%20gamemaker/events/destroy%20event.html

Yoyo Games, 2017l. The Alarm events. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/000_using%20gamemaker/events/alarm%20event.html

Yoyo Games, 2017m. The Draw Event. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/000_using%20gamemaker/events/draw%20event.html

Yoyo Games, 2017n. json_decode. Viitattu 28.8.2017
https://docs.yoyogames.com/source/dadiospice/002_reference/file%20handling/json_decode.html

Yoyo Games, 2017o. Porting A GMS 1.4 Game To GameMaker Studio 2. Viitattu 12.10.2017
https://help.yoyogames.com/hc/en-us/articles/231719468-Porting-A-GMS-1-4-Game-To-GameMaker-Studio-2

Yoyo Games, 2017p. Motion Planning. Viitattu 15.10.2017
https://docs.yoyogames.com/source/dadiospice/002_reference/movement%20and%20collisions/motion%20planning/index.html