Lauri Hankilanoja

3D-kenttäeditori Unity-peleille

Insinööri (AMK)

Tieto- ja viestintätekniikka

Kevät 2018

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Tiivistelmä

Tekijä: Hankilanoja Lauri

Työn nimi: 3D-kenttäeditori Unity-peleille

Tutkintonimike: Insinööri (AMK), tieto- ja viestintätekniikka

Asiasanat: kenttäeditori, 3D-malli, Unity, ohjelmistokehitys, videopeli

Työn tilaajana toimi kajaanilainen peliyritys TJR Games Oy. TJR:n pelinkehitys erikoistuu Unity-pelimoottorilla tehtyihin peleihin. Yrityksen menestynein peli on vuonna 2015 ilmestynyt vuoropohjainen strategiapeli Interplanetary.

Työn tavoitteena oli kehittää 3D-kenttäeditori yrityksen käyttöön. Kenttäeditorin tehtävänä oli ohittaa Unity-pelimoottorin asettama rajoitus, joka estää 3D-mallien lataamisen ajon aikana. Editorin toteutukseen käytettiin Unitya ja sen tukemaa C#-ohjelmointikieltä.

Raportti on jaettu kahteen osaan: teoriaan ja toteutukseen. Teoriaosuudessa syvennytään kenttäeditorin toteutusta edellyttäviin käsitteisiin: kenttäeditoreihin, Unityyn ja 3D-malleihin. Ohjelmiston toteutus puolestaan jaetaan perinteisen ohjelmistonkehitysprosessin neljään vaiheeseen: vaatimusmäärittelyyn, suunnitteluun, toteutukseen ja testaukseen.

Opinnäytetyön lopputuloksena on käyttövalmis kenttäeditori. Valmis editori sisältää tarvittavat työkalut kentän tekemiseen ja tallentamiseen, mutta muutama suunniteltu ohjelman käyttöä helpottava työkalu jäi implementoimatta.

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Abstract

Author: Hankilanoja Lauri

Title of the Publication: 3D Level Editor for Unity Games

Degree Title: Bachelor of Engineering, Information Technology Engineering

Keywords: level editor, 3D model, Unity, software development, video game

This thesis was ordered by TJR Games Oy, a video game company located in Kajaani. TJR specialises in video games made with Unity game engine. Their most famous game is a turn based strategy game called Interplanetary which was released in 2015.

The goal of this thesis was to develop a 3D level editor for the previously mentioned company. The level editor was to bypass a restriction set by Unity that prevents 3D model loading at runtime. The level editor was made with Unity and C# programming language.

The report is divided into two parts: theory and execution. The theory consists of concepts that are essential to know before moving onto development. These concepts are as follows: level editors, Unity and 3D models. The execution on the other hand is divided according to traditional software development process which consists of 4 chapters: specification, design process, implementation and testing.

The end product is a working level editor. The level editor has the necessary tools to create and save levels but few of the planned features were not implemented. These tools were designed to make the software easier to use but were not essential for using the product.

Sisällys

1 Johdanto

Kenttien tekeminen käsin tekstitiedostoon on työlästä. Tästä syystä kenttien tekemiseen käytetään nykyään useimmiten siihen erikoistunutta työkalua: kenttäeditoria. Kenttäeditori tuo kentän tekemiseen mukaan visuaalisuuden, estää monia huolimattomuusvirheitä ja nopeuttaa pelinkehitystä.

Työn tilaajana toimii TJR Games Oy. TJR (Team Jolly Roger) on vuonna 2013 perustettu kajaanilainen peliyritys. TJR on erikoistunut Unity-pelimoottorilla toteutettuihin peleihin ja tekee omien videopelien lisäksi myös alihankintatöitä. Yritys on julkaissut pelejä älypuhelimille ja tietokoneille. [1.]

Peleistä menestynein on vuonna 2015 ilmestynyt strategiapeli Interplanetary, josta julkaistiin vuonna 2017 paranneltu versio nimellä Interplanetary: Enhanced Edition. Interplanetary on avaruuteen sijoittuva vuoropohjainen strategiapeli. Peli on saatavilla Windows, Mac ja Linux -käyttöjärjestelmille.

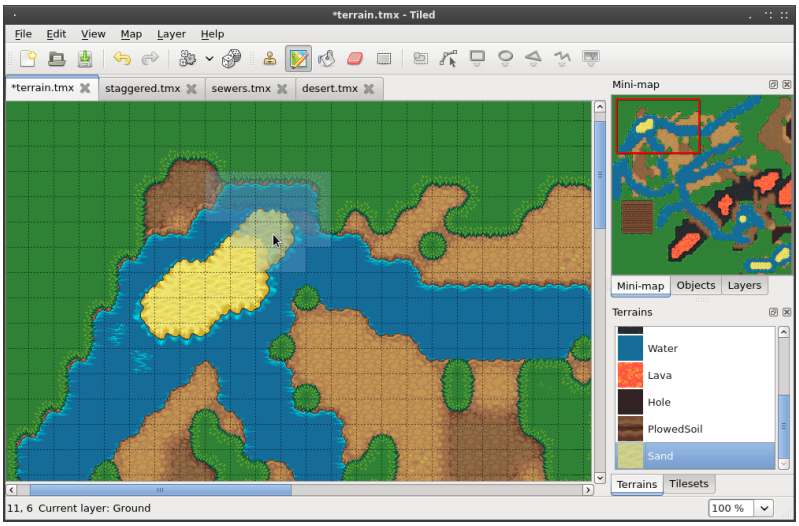
Yritys kehittää tällä välin Worbital nimistä uutta peliä. Peli sijoittuu Interplanetaryn tapaan ulkoavaruuteen, mutta strategiaelementit on vaihdettu nopeatempoiseen vihollisplaneettojen räjäyttämiseen.

Työn tavoitteena on ohjelmoida Unity-pelimoottorilla tehtyjen pelien kanssa yhteensopiva 3D-kenttäeditori. Unity tarjoaa oman ratkaisunsa kenttien rakentamiseen, mutta sillä on rajoituksensa. Tavoitteena on ohjelmoida kenttäeditori, joka kiertää Unityn rajoitteen, joka estää 3D-mallien lataamisen ajon aikana. Rajoitteen ohittamisen tarkoituksena on välttää tarve tuoda 3D-mallit Unity-projektiin ja mahdollistaa tuki pelaajien tekemille kentille.

2 Kenttäeditori

Kenttäeditori on pelinkehitystä helpottava työkalu, jolla voidaan nopeasti ja helposti rakentaa peliympäristöjä peliä varten. Kenttäeditoreja on monenlaisia eri käyttötarkoituksiin. Editori voi olla pelkästään pelinkehitystiimin käyttämä työkalu, mutta vaihtoehtoisesti kenttäeditorin voi antaa käyttöön myös peliä pelaaville, jolloin he voivat lisätä peliin omaa sisältöä. On myös tapauksia, joissa pelaajille ei ole tarjottu työkaluja kenttien tekemiseen, joten pelaajat ohjelmoivat omat työkalunsa siihen tarkoitukseen [2].

Kenttäeditori voi olla joko oma erillinen ohjelmansa tai pelin sisäinen ominaisuus. Monet pelimoottorit sisältävät tarvittavat työkalut kenttien tekemiseen. Nämä kenttäeditorit on suunniteltu tukemaan mahdollisimman montaa pelityyppiä. Yksittäiselle pelille on kuitenkin mahdollista ohjelmoida oma kenttäeditori, jos kentän tekemiseen tarvitaan lisäominaisuuksia. Kuvassa 1 on 2D-peleille suunniteltu kenttäeditori Tiled.



Kuva 1. Tiled on monikäyttöinen kenttäeditori 2D-peleille

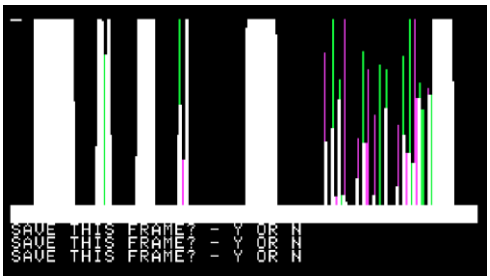
2.1 Historia

Ensimmäiset kenttäeditorit tulivat peleihin 80-luvun alkupuolella. Vuonna 1981 julkaistut pelit Pegasus 2 [3, s. 7] ja Munchkin [4] sekä vuonna 1983 julkaistu Lode Runner [5] olivat ensimmäisten pelien joukossa, jotka sisälsivät kenttäeditorin.

On-Line Systemsin Apple II:lle julkaistussa Pegasus 2:ssa pelaajan tehtävänä on ohjata lentokonetta ja kerryttää pisteitä tuhoamalla vihollisia ja heidän rakennuksiaan. Tuhoaminen tapahtuu ampumalla suoraan eteenpäin tai pudottamalla pommeja kaaressa alaspäin. Pelaajan täytyy samalla väistellä erilaisia esteitä ja pitää silmällä polttoainemittaria, jotta lentokone ei tipu maahan. [6.] Kuvassa 2 näkyy pelin ensimmäinen taso.

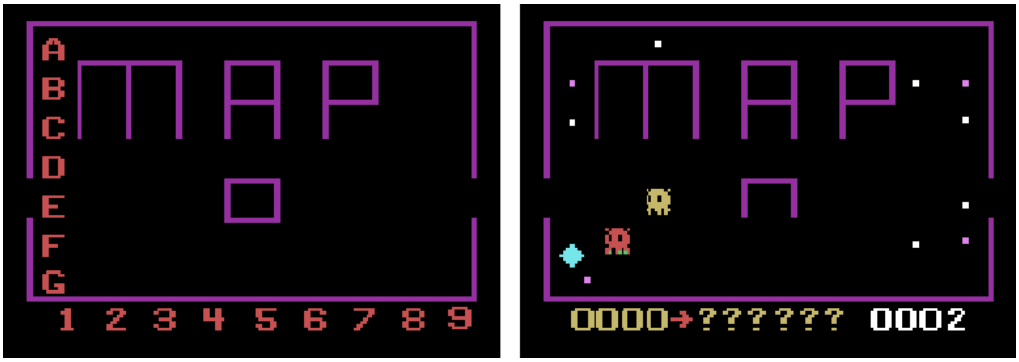
Kuva 2. Pegasus 2

Pelin kenttäeditorissa pelaaja pystyy luomaan erilaisia kentän osia säätämällä maaston korkeutta. Korkeuden säätäminen tapahtuu liikuttamalla sauvaohjainta ylös ja alas sitä mukaa, kun editori muodostaa maastoa vaakasuunnassa vasemmalta oikealle. Yksittäinen taso koostuu 21:stä osasta, jotka kenttäeditori sijoittelee satunnaisessa järjestyksessä ketjuun, muodostaen pelattavan tason. [6.] Kentän tekoprosessi on kuvattuna kuvassa 3.



Kuva 3. Pegasus 2:n primitiivinen kenttäeditori

Munchkin on Ed Averettin suunnittelema ja ohjelmoima Pac-Manin kaltainen sokkelopeli Philips Videopacille. Pelissä kerätään tasoon sijoiteltuja pisteitä ja vältetään kolmea pelaajaa jahtaavaa vihollista. Osa kentän pisteistä välkkyy ja välkkyvän pisteen keräämällä pelin päähahmo, Munchkin, saa hetkellisesti kyvyn syödä häntä jahtaavat viholliset. Pelin kenttäeditorilla pelaaja voi tehdä oman tasonsa asettamalla ja poistamalla seiniä. [4.] Kuvassa 3 on pelin kenttäeditori ja sillä tehdyn kentän pelaaminen.

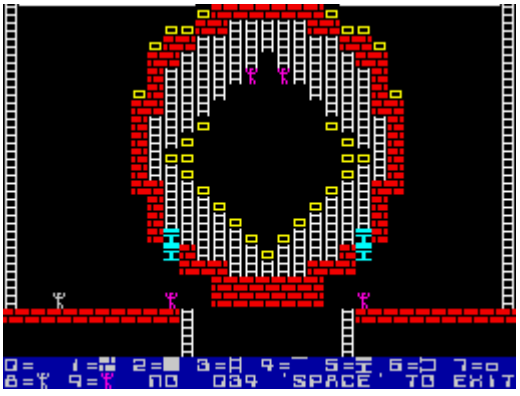


Kuva 4. Muchkin-pelin taso, suunnittelu- ja pelausvaiheessa

Monille kotikoneille julkaistu Lode Runner on pulmapelin ja tasohyppelyn risteytys, jossa pelaajan tehtävänä on kerätä kaikki kulta tasosta ja paeta tikkaita pitkin seuraavaan tasoon. Tehtävää vaikeuttavat vartijat, jotka yrittävät napata pelaajan. [7, s. 1.]

Pelin ohjelmoija, Douglas E. Smith, aloitti pelin tekemisen kesällä vuonna 1982 [8]. Id Software -peliyrityksen perustajan, John Romeron, mukaan Lode Runnerin kenttäeditori sai alkunsa pelin ohjelmoineen Smithin naapurissa asuvista lapsista. Naapuruston lapset tapasivat testata Smithin ohjelmoimia pelejä, ja Lode Runneria pelattuaan lapset halusivat tehdä peliin omia kenttiä. Peliin lisättiin sen seurauksena kenttäeditori ja osa pelin 150:stä tasosta onkin Romeron mukaan näiden lasten suunnittelemia. [3, s. 7.]

Smith opiskeli Washingtonin yliopistossa, jossa hän jakoi pelin kaikkien yliopiston opiskelijoiden käyttöön. Peli oli hyvin suosittu opiskelijoiden keskuudessa, osittain pelin kenttäeditorin takia. Smith lisäsi ja poisti kesän aikana monia ominaisuuksia peliä pelanneiden antaman palautteen perusteella. [8.] Pelin kenttäeditori on esitetty kuvassa 5. Editorissa jokaista näppäimistön numeroa vastaa erilainen rakennuspalikkatyyppi. [7, s. 7.]



Kuva 5. Lode Runnerin kenttäeditori

Peli oli myyntimenestys tietokoneilla, mutta suurimmat tulot tulivat Japanista pelin ilmestyttyä Nintendon Famicomille. Pelin kehittäjän mukaan peliä myytiin n. 1,5 miljoonaa kappaletta Famicomille ja hän arvioi kaikkien alustojen kesken myydyksi määräksi yli 3 miljoonaa kappaletta. [8.]

3D-peleihin siirryttäessä kenttäeditoreista on tullut lähestulkoon pakollisia työkaluja 3D-kenttien kompleksisuuden takia. 2D-maailma on suhteellisen helppo luoda ja muokata tekstitiedostona, mutta kolmas ulottuvuus nostaa datan määrää ja vaikeuttaa datan rakenteen hahmottamista. Nykyään pelikohtaisia kenttäeditoreja tehdään pelin suunnittelua varten vähemmän, koska yleisimmin käytössä olevat pelimoottorit tarjoavat käyttäjille tasojen rakentamiseen tarvittavat työkalut.

2.2 Ominaisuudet

Editorin ominaisuudet riippuvat sitä käyttävän pelin tarpeista. Kenttäeditorit, jotka pyrkivät tukemaan monenlaisia pelejä, joutuvatkin tästä syystä pitämään ominaisuudet geneerisempinä. Universaaliin kenttäeditoriin on turha ohjelmoida ominaisuuksia, jotka ovat hyvin pelikohtaisia ja tulevat harvoin tarpeen. Tällaisista ominaisuuksista esimerkkeinä ovat kuvassa 6 esiintyvän Worms Armageddonin työkalut siltojen lisäämiselle ja veden pinnan korkeuden muokkaamiselle. Ominaisuudet ovat tärkeä osa Worms Armageddon peliä, mutta kaikki pelit eivät tarvitse kyseisiä työkaluja.

Kuva 6. Worms Armageddonin kentissä on mahdollista muokata veden pinnan korkeutta

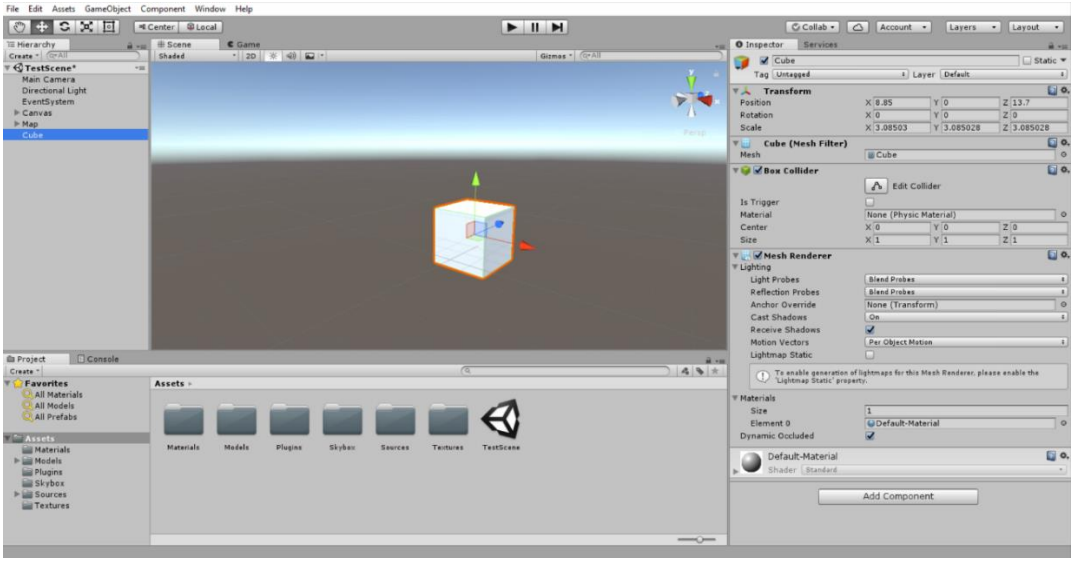
Kenttien tekemiseen käytettävät työkalut vaihtelevat editorin mukaan. Useimmiten nykyisissä editoreissa on kuitenkin mahdollisuus muuttaa pelialueen kokoa, muokata maastoa, asettaa kenttään objekteja ja tallentaa valmis kenttä myöhempää käyttöä varten.

2.3 Hyödyt

Kenttäeditorin tekemiseen kuluva aika vähentää huomattavasti itse kenttien tekemiseen kuluvaa aikaa. Peleissä kenttiä on yleensä useita ja niiden muokkaaminen on työlästä ilman niille suunniteltua työkalua. Kenttäeditori tuo mukaan visuaalisuuden, joka tekee työprosessista selkeämpää. Myös mahdollisten virheiden todennäköisyys vähentyy, kun editoriohjelma pitää kirjaa siitä, mitä on mahdollista tehdä ja mitä ei. Kenttäeditorin rakenteesta riippuen samaa kenttäeditoria, tai osaa sen sisältämästä koodista, voi hyödyntää myös tulevissa projekteissa. [9.]

3 Unity

Unity on Unity Technologiesin tuottama pelimoottori, jonka ensimmäinen versio tuli markkinoille kesällä 2005 [10, s. 8]. Pelimoottorit ovat hyvin samankaltaisia kenttäeditoreihin verrattuna, sillä myös ne on suunniteltu nopeuttamaan ja helpottamaan pelien kehitystä, vaikkakin laajemmassa skaalassa. Pelimoottoreilla on usein visuaalinen näkymä, jossa käyttäjä voi muokata peliänsä. Kuvassa 7 on kuvattuna Unityn käyttöliittymä.



Kuva 7. Unity

Pelimoottori koostuu komponenteista, jotka suorittavat useita toimintoja käyttäjän puolesta. Käyttäjän ei esim. tarvitse Unitya käyttäessään kirjoittaa pelimaailman renderöinnin suorittavaa pohjakoodia, vaan Unity sisältää renderöintikoodin ja suorittaa sen automaattisesti. Pelimoottorin yleisimpiin komponentteihin kuuluvat mm. renderöinti, grafiikat, äänet, fysiikat ja skriptaus. [11, s. 3.]

Unityn perusversio on ilmainen, mutta käyttäjän tulee päivittää kuukausimaksulliseen lisenssiin, jos vuosittainen bruttotulo ylittää sopimusehdoissa annetun tulorajan [12]. Unityn versiot ja niiden hinnoittelu on esitetty taulukossa 1.

Versio	Hinta	Tuloraja
Personal	Ilmainen	< $100000/v
Plus	$35/kk	< $200000/v
Pro	$125/kk	Ei rajoitusta
Enterprise	Sovittava erikseen	Ei rajoitusta

Taulukko 1. Unityn hintaporrastus [12]

3.1 Ominaisuudet

Unity on monipuolinen moottori, joka soveltuu niin pienempiin kuin suurempiinkin peliprojekteihin. Unity tarjoaa laajan määrän erilaisia työkaluja, jotka pyrkivät antamaan käyttäjille mahdollisuuden tehdä, mitä he ikinä vain haluavatkaan. Unitylla voi tehdä 2D- tai 3D-pelejä pelin genrestä riippumatta monille eri alustoille.

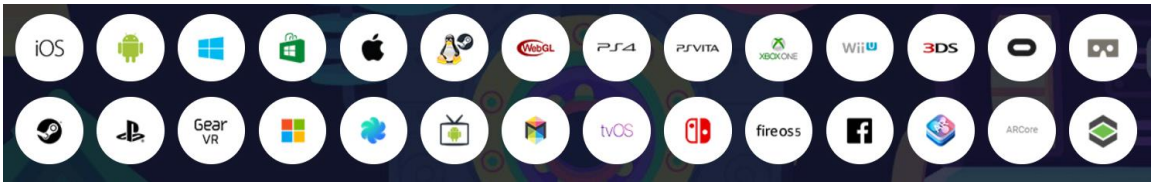
Unitylla tehdyt pelit koostuvat peliobjekteista, joita käyttäjä voi sijoittaa pelimaailmaan. Jokaisella peliobjektilla on ”transform”-komponentti, joka sisältää objektin position, rotaation ja skaalan. Peliobjekteille voi lisätä myös muita komponentteja, kuten valoja, ääniä, 3D-malleja, fysiikkakomponentteja ja ohjelmakoodia.

Unity tukee pääsääntöisesti C# -ohjelmointikieltä, mutta projekteihin on mahdollista lisätä myös mm. C ja C++ -liitännäisiä. [13] Ohjelmointi tapahtuu Unityn ulkopuolella joko Unityn mukana tulevalla MonoDevelop-ohjelmalla tai jollain tuetuista ohjelmista, kuten Visual Studiolla. Myös muita resursseja, kuten 3D-malleja ja ääniä täytyy tuottaa Unityn ulkopuolella.

3.2 Vahvuudet

Unity on suosittu pelimoottori etenkin pienten peliyritysten ja harrastelijoiden keskuudessa, koska sen perusversio on ilmainen ja yksinkertainen käyttää [12]. Unitylla on tarjolla Asset Store -niminen verkkokauppa, josta pystyy lataamaan 3D-malleja, musiikkia, valmista koodia ja muita pelin tekemistä helpottavia asioita niin ilmaiseksi kuin myös maksua vastaan.

Unity on tunnettu laajasta alustatuestaan. Unity tukee useampaa alustaa kuin kilpailevat pelimoottorit. Tuettavia alustoja on yhteensä yli 25, ja niihin kuuluvat kaikki yleisimmät käytössä olevat tietokone-, mobiili-, konsoli- ja VR-käyttöjärjestelmät. [14.] Tuetut alustat ovat esiteltynä kuvassa 8.

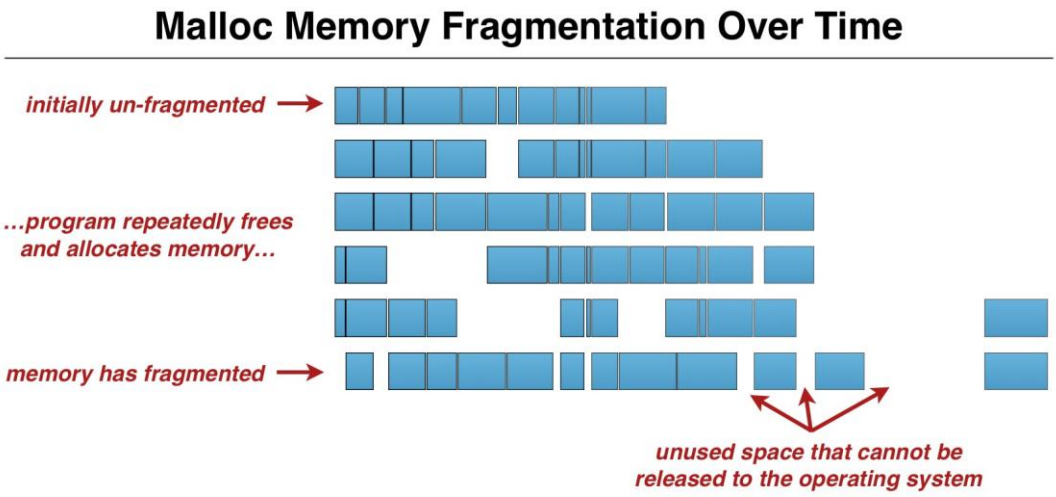


Kuva 8. Unityn tukemat alustat

3.3 Heikkoudet

Työn kannalta tärkeässä osassa on Unityn kykenemättömyys tuoda peliin 3D-malleja ajon aikana. Tämä este on kuitenkin onnistuttu kiertämään mm. suomalaisessa Cities: Skylines -pelissä, jossa 3D-malli ladataan manuaalisesti mallitiedostosta [15].

Unity käyttää paljon muistia, mikä saattaa aiheuttaa ongelmia laitteilla, joissa muistin määrä on pieni. Muistin vapautuksesta huolehtii ”automaattinen roskienkeräys”, joka vapauttaa muistia automaattisesti, kun muistissa olevaan dataan ei enää viitata. Roskienkeräys saattaa viedä paljon aikaa ja johtaa suorituskykyongelmiin, jolloin peli hidastuu hetkellisesti. Muisti voi myös pirstoutua kuvan 9 tavalla. Allokoitaessa muistista käytetään ensimmäinen muistin määrää vastaava, tai sitä isompi, muistipaikka. Vapauttaessa aiempia muistivarauksia muistiin jää aukkoja, jotka täytetään myöhemmin uudelleen. Varattavan ja vapautettavan muistin määrä voi kuitenkin vaihdella ja se johtaa tilanteisiin, joissa vapaan muistin kokonaismäärä voi olla allokoinnille tarpeeksi, mutta allokointiin tarvittavaa tarpeeksi suurta yhtenäistä muistipaikkaa ei löydy. [16.]



Kuva 9. Muistin pirstoutuminen [17]

Muistinhallintaan on mahdollista vaikuttaa esim. hyödyntämällä välimuistia, välttämällä muistiallokointia silmukoissa ja ohjelmoimalla ns. object pool -järjestelmän, jonka periaatteena on varata ja vapauttaa suuria määriä muistia kerralla, jolloin vältytään jatkuvilta allokointikutsuilta. [18.]

Unityn lähdekoodi ei ole yleisesti saatavilla, joten mahdollisia moottorin sisäisiä ohjelmointivirheitä ei voi korjata itse, vaan ongelmasta täytyy ilmoittaa eteenpäin Unity-tiimille. Tämän rajoituksen pystyy kiertämään vaihtamalla Unity Enterpriseen, mutta se on tarjolla vain yli 20 hengen sovelluskehitystiimeille yritysten keskenään sopimaan hintaan. [19.]

3.4 Unitylla tehdyt sovellukset

Unity on käytössä monilla pienillä peliyrityksillä, mutta myös isot yritykset, kuten Blizzard, Electronic Arts, Square Enix ja Ubisoft, ovat tehneet pelejä Unitylla. Unity on suunniteltu ensisijaisesti pelien kehitykseen, mutta sillä on mahdollista tehdä myös muita sovelluksia. Tällaisia ohjelmia ovat mm. Googlen VR-piirustusohjelma Tilt Brush ja MedStar SiTELin valmistama Virtual Zoll Defibrillator Trainer -opetusohjelma sairaanhoitajille.

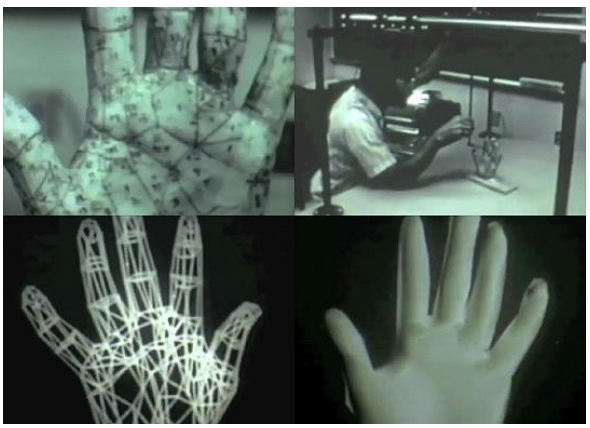
4 3D-mallit

3D-mallit ovat tietokoneiden tapa käsitellä kolmiulotteisia objekteja. 3D-malleja voidaan mallintaa niihin erikoistuneilla mallinnusohjelmilla, kuten esim. ilmaisella Blenderillä tai AutoDeskin maksullisilla sovelluksilla: Mayalla ja 3DS Maxilla. Valmista 3D-mallia voidaan käyttää tietokonesovelluksissa tai tulostaa 3D-tulostimella. Kuvassa 10 on perinteisesti koekäyttöön tarkoitettu 3D-malli Utah'n teekannu.

Kuva 10. Utah'n teekannu

4.1 Historia

Yksi ensimmäisistä 3D-malleista esiintyi vuonna 1972 tietokoneanimoidussa lyhytelokuvassa A Computer Animated Hand [20]. Kyseinen malli vastasi elokuvatuottaja Edwin Catmullin vasenta kättä. Malli luotiin fyysisen mallikappaleen pohjalta, johon piirrettiin käden muodostavat monikulmiot. Fyysisen mallin valmistuttua malli digitalisoitiin mittaamalla monikulmioiden muodostamien kärkipisteiden koordinaatit ja syöttämällä arvot tietokoneelle. [21.] Mallin työprosessin vaiheet on esitetty kuvassa 11. Samainen 3D-malli nähtiin uudestaan neljä vuotta myöhemmin elokuvassa Futureworld [20].



Kuva 11. Edwin Catmullin 3D-mallin työprosessin vaiheet

Videopeliteollisuudessa kolmiulotteisuutta jäljiteltiin monilla erilaisilla tekniikoilla jo 1980-luvun alkupuolella, mutta varsinaiset täytettyjä polygoneja käyttävät pelit tulivat saman vuosikymmenen loppupuolella. Näistä ensimmäisten joukossa oli Namcon vuonna 1988 Japanissa julkaisema kolikkopeli Winning Run, joka on esitetty kuvassa 12. Peli hyödyntää Namco System 21 -piirilevyä, joka oli ensimmäinen 3D-mallien prosessointiin suunniteltu piirilevy kolikkopeleille. [22].

Kuva 12. Winning Run

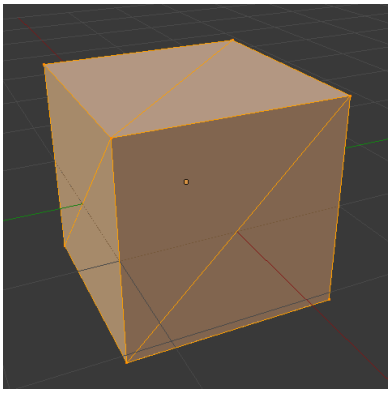
Varsinaisen läpimurron teki Sega, aloittaen ensin 3D-kolikkopeleillä ja siirtymällä niistä 3D-peleille suunniteltuun pelikonsoliin, Sega Saturniin. Sega sai pian peräänsä myös kilpailevan Sonyn, jonka pelikonsoli PlayStation tuli markkinoille samana vuotena. [23.]

4.2 3D-mallin rakenne

3D-mallit koostuvat geometrisesta datasta. Tässä alaluvussa on esitetty tärkeimpiä käsitteitä 3D-mallin rakenteeseen liittyen.

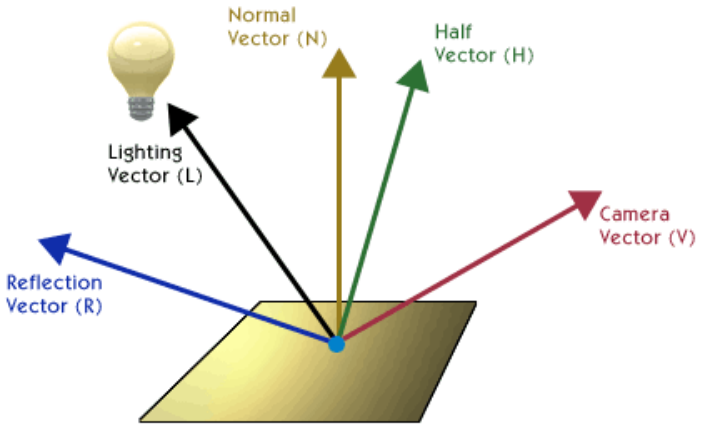
Kärki: Pistemäinen kappale kolmiulotteisessa X, Y, Z -koordinaatistossa. Tunnetaan myös nimellä verteksi.

Tahko: Muodostavat kappaleen pinnan. Määritellään kappaleen kärkien indekseillä. Yhdellä tahkolla voi olla viittaus kolmeen tai useampaan indeksiin, jotka yhdistämällä saadaan aikaiseksi monikulmio. Yleisimmin tahkot ovat kolmioita tai nelikulmioita. Kuvan 13 kuutio muodostuu 8:sta kärjestä ja 12:sta kolmion muotoisesta tahkosta.



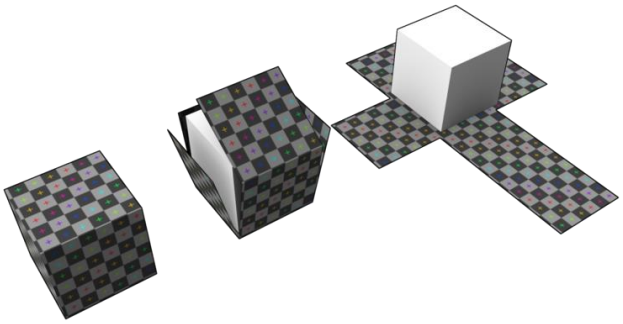
Kuva 13. Kuvan kuutiossa on 8 kärkeä ja 12 tahkoa

Normaali: Kappaleen pinnasta kohtisuoraan poispäin osoittava vektori. Määrittävät, kuinka valo osuu kappaleeseen. Kuvassa 14 on esitetty, kuinka normaalivektori vaikuttaa kappaleen valaistukseen.



Kuva 14. Normaalivektorin vaikutus kappaleen valaistukseen

UV-koordinaatit: Tunnetaan myös nimellä tekstuurikoordinaatti. Määrittelevät kaksiulotteisessa U, V -koordinaatistossa, kuinka tekstuurit asettuvat kappaleen pinnalle. Kuvassa 15 on esimerkki tekstuurin asettumisesta kuution ympärille UV-koordinaattien arvojen mukaan.



Kuva 15. Tekstuurin asettuminen kuution ympärille

Materiaali: Sisältää mm. kappaleen väriarvot ja tekstuurit. Kuvassa 16 on sama malli kahteen kertaan, joista vasemmanpuolinen sisältää oikeat tekstuurit ja väriarvot.

Kuva 16. Teksturoitu ja teksturoimaton 3D-malli

4.3 OBJ-tiedostomuoto

OBJ-tiedostomuoto ja sen parina toimiva MTL-tiedostomuoto ovat Wavefront Technologiesin kehittämiä dataformaatteja. Niiden yksinkertaisuus on yhtaikaa sekä niiden vahvuus että myös heikkous. OBJ-muoto sisältää vain tärkeimmät tiedot 3D-mallista, joten se asettaa ymmärrettävästi samalla rajoituksia. Objektin materiaalidata tallennetaan erilliseen MTL-tiedostoon, johon OBJ-tiedostolla on sisällään viittaus. [24.]

OBJ-tiedostossa informaatio jakautuu riveille. Jokainen rivi alkaa tunnuksella, joka kertoo datatyypin, ja sitä seuraa itse data, joka jaetaan välilyönneillä omiin lohkoihinsa. Objektin

koordinaatit sisältävät datatyypin mukaan kaksi tai kolme lukua, jotka muodostavat yhdessä vektorin. Mallin tahkot muodostuvat kolmesta tai useammasta indeksiryhmästä. Yhdessä ryhmässä on korkeintaan kolme indeksiä, riippuen siitä, onko tiedostolle määritelty UV-koordinaatteja. Indeksit erotellaan vinoviivoilla. Ensimmäinen indeksi viittaa kappaleen kärkiin, toinen UV-koordinaatteihin ja kolmas normaaleihin. Kuvassa 17 on avattuna teksturoidun kuution mallitiedosto ja taulukossa 2 on listattuna kuvassa näkyvien datatunnusten merkityksiä. [24.]

```
# Blender v2.79 (sub 0) OBJ File: ''
# www.blender.org
mtllib TexturedCube.mtl
o Cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -0.999999
v 0.999999 1.000000 1.000001
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 0.000000 0.000000
vt 0.000000 0.000000
vt 1.000000 1.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 1.000000
vt 1.000000 1.000000
vt 1.000000 1.000000
vt 1.000000 0.000000
vt 1.000000 1.000000
vn 0.0000 -1.0000 0.0000
vn 0.0000 1.0000 0.0000
vn 1.0000 -0.0000 0.0000
vn 0.0000 -0.0000 1.0000
vn -1.0000 -0.0000 -0.0000
vn 0.0000 0.0000 -1.0000
usemtl Material
s off
f 2/1/1 4/2/1 1/3/1
f 8/4/2 6/5/2 5/6/2
f 5/7/3 2/8/3 1/3/3
f 6/9/4 3/10/4 2/11/4
f 3/12/5 8/13/5 4/2/5
f 1/14/6 8/15/6 5/6/6
f 2/1/1 3/16/1 4/2/1
f 8/4/2 7/17/2 6/5/2
f 5/7/3 6/18/3 2/8/3
f 6/9/4 7/17/4 3/10/4
f 3/12/5 7/19/5 8/13/5
f 1/14/6 4/20/6 8/15/6
```

Kuva 17. Teksturoitu kuutio tallennettuna OBJ-muotoon

Tunnus	Tyyppi
mtllib	Materiaalitiedosto
usemtl	Käytettävä materiaali
o	Objektin nimi
v	Kärki
vt	UV-koordinaatti
vn	Normaali
f	Tahko
s	Graafinen tasoitus

Taulukko 2. OBJ-tiedoston tunnuksia

OBJ-tiedoston parina toimiva materiaalitiedosto on hyvin samantapainen rakenteeltaan. MTL-tiedosto voi sisältää useamman materiaalin, jotka erotellaan ”newmtl”-tunnuksella. Materiaali sisältää pääasiassa objektin väriarvot, läpinäkyvyyden, valaistusmallin ja tekstuuripolut. Kuvassa 18 on avattuna teksturoidun kuution materiaalitiedosto ja taulukossa 3 on listattuna kuvassa näkyvien datatunnusten merkityksiä. [25.]

```
1  # Blender MTL File: 'None'
2  # Material Count: 1
3
4  newmtl Material
5  Ns 96.078431
6  Ka 1.000000 1.000000 1.000000
7  Kd 0.640000 0.640000 0.640000
8  Ks 0.500000 0.500000 0.500000
9  Ke 0.000000 0.000000 0.000000
10 Ni 1.000000
11 d 1.000000
12 illum 2
13 map_Kd Assets\Models\textures\ColorPixelArt (x2).png
```

Kuva 18. MTL-tiedoston rakenne

Tunnus	Tyyppi
newmtl	Materiaalin nimi
Ns - Ni	Erilaisia väriarvoja
d	Läpinäkyvyys
illum	Valaistusmalli
map	Tekstuuripolku

Taulukko 3. MTL-tiedoston tunnuksia

5 Kenttäeditorin toteutus

Seuraavissa alaluvuissa käydään läpi ohjelmiston kehitys perinteisen ohjelmistonkehitysprosessin neljän eri vaiheen mukaan. Vaiheet jakautuvat vaatimusmäärittelyyn, suunnitteluun, toteutukseen ja testaukseen.

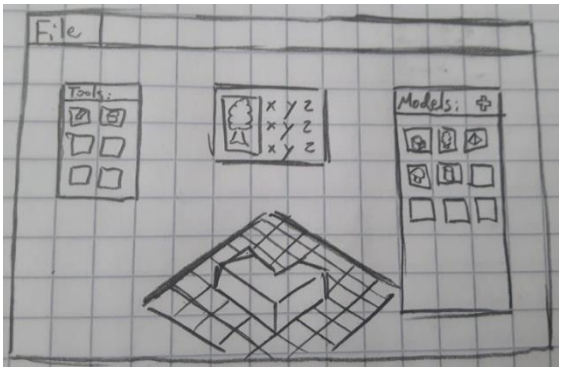
5.1 Vaatimusmäärittely

Tehtävänä on luoda TJR Games Oy:n kaupalliseen pelinkehityskäyttöön soveltuva 3D-kenttäeditori. Kenttäeditorin tulee toimia Windows-ympäristössä. Kenttiin tulee pystyä lisäämään 3D-malleja sekä ruudukolle että vapaasti koordinaatistoon. Valmis kenttä tallennetaan kenttätiedostoksi, jonka tulee olla yhteensopiva Unity-pelien kanssa, pelien alustasta riippumatta.

5.2 Suunnittelu

Koska kenttäeditorilla tehtyjen kenttien tulisi olla yhteensopiva Unity-pelien kanssa, itse editorin tekemiseen käytettiin myös Unitya ja sen tukemaa C#-ohjelmointikieltä. Tällä tavoin editorin käyttämää mallin- ja kentänlatauskoodia voi samalla käyttää myös kenttiä käyttävässä pelissä.

Ohjelman suunnittelu alkoi käyttöliittymän suunnittelusta. Tarkoituksena oli tehdä mahdollisimman sulava ja helppokäyttöinen käyttöliittymä. Kuvassa 19 on ensimmäinen hahmotelma ohjelman käyttöliittymästä. Keskellä tulisi olemaan näkyvissä muokattava kenttä, ja näkymää ympäröisivät erilaiset vapaasti liikuteltavat valikot. Kenttäeditorin käyttöliittymä tarjoaa tarvittavat työkalut kentän rakentamiseen. Ohjelman käyttäminen edellyttää hiirtä ja näppäimistöä.



Kuva 19. Luonnos kenttäeditorin tulevasta käyttöliittymästä

Käyttöliittymäsuunnittelun jälkeen vuorossa oli tarvittavien komponenttien listaus. Komponentit järjestettiin tärkeysjärjestykseen ja niille asetettiin aikataulutavoitteet. Tärkeimmäksi komponentiksi valikoitui 3D-mallien lataaminen, joka tulisi oletettavasti olemaan eniten aikaa vievä ominaisuus. Siitä syystä sen ohjelmoiminen tulisi aloittaa hyvissä ajoin. Tätä ominaisuutta seurasivat mm. mallien sijoittelu kenttään ja siihen tarvittava ruudukko, tarvittavat valikot sekä kenttien tallentaminen ja lataaminen.

Suunnitteluvaiheessa valittiin myös ohjelman tukema tiedostomuoto. OBJ-tiedostomuoto osoittautui työn kannalta hyväksi tiedostomuodoksi, koska se on vapaa tiedostomuoto ja sen rakenne on hyvin yksinkertainen. OBJ-tiedosto on helppo avata ja lukea, joten tuen toteuttaminen tulisi olemaan nopeampaa verrattuna muihin tiedostomuotoihin ja mallien latauksessa pääsisi keskittymään enemmänkin ladattujen arvojen asettamiseen Unityn peliobjektille.

Unity vaatii, että 3D-mallin tahkot ovat kolmioita, mutta mallitiedostossa tahkot voivat koostua monenlaisista monikulmioista. Tällöin 3D-malli tulee kolmioida. Kolmioinnilla tarkoitetaan polygonin jakamista kolmioihin. 3D-mallin kolmiointiin tutustuminen ja toteuttaminen on kuitenkin liian laaja kokonaisuus sisällyttää aikatauluun, joten se joudutaan suorittamaan mallinnusohjelman puolella ennen mallin tuomista kenttäeditoriin.

5.3 Toteutus

Suunnittelun jälkeen vuorossa on toteutus. Toteutuksen alaluvut perustuvat kenttäeditorin pääkomponentteihin. Aluksi käydään läpi ohjelman käyttöliittymän toteutus, tästä jatketaan 3D-malleihin, ja viimeiseksi siirrytään kenttien tallentamiseen ja lataamiseen.

5.3.1 Käyttöliittymä

Käyttöliittymän tiedostonselausvalikko ohjelmoitiin toimimaan sitä kutsuvasta peliobjektista riippumatta. Valikko näyttää aina kansiorakenteen, mutta sitä kutsuva objekti voi määrittää tiedostopäätteet, joiden perusteella tiedostoja listataan. Kansiota painaessa nykyiseen polkuun lisätään kansion nimi ja siirrytään kyseiseen kansioon. Kansiosta poistuessa polusta puolestaan poistetaan viimeisimmän kansion nimi. Tiedostopäätteeseen loppuvaa valintaa painaessa tiedostopolku palautetaan tiedostonselausvalikon avanneelle objektille, joka määrittelee, mitä kyseiselle tiedostolle tehdään. Tällä tavalla samaa valikkoa on mahdollista käyttää niin mallitiedostojen etsimisen lisäksi myös muiden tiedostojen, kuten kenttien, tallentamiseen ja lataamiseen.

Kenttä koostuu ruudukosta, johon voidaan asettaa 3D-malleja painamalla ruudukon solua. Ruudukko toteutettiin ensin asettamalla koodin kautta yksittäisiä ruutuja vierekkäin silmukoiden avulla. Kyseessä on kuitenkin raskas toiminto, joka vie sitä enemmän aikaa, mitä enemmän ruutuja on. Tästä syystä paremmaksi keinoksi osoittautui vaihtaa koko ruudukko yhdeksi objektiksi. Objektia skaalataan ruutujen määrän ja koon mukaan ja objektin sisältämä tekstuuri jaetaan oikean kokoisiksi ruuduiksi. Käyttäjän valitseman ruudun sijainti lasketaan hiiren painalluksen törmäyspisteestä. Malleja voi asettaa ruudukon lisäksi myös vapaasti peliympäristöön, jolloin törmäyspistettä ei tarvitse erikseen muuttaa ruudukon koordinaatteihin.

5.3.2 3D-mallien käsittely

Kenttäeditori avaa mallitiedostot ja ottaa niistä talteen editorin tarvitseman datan rivi riviltä datan tunnisteen perusteella. Sama operaatio suoritetaan myös mallia vastaavalle materiaalitiedostolle. Tiedostoista luettu data asetetaan tämän jälkeen Unityn peliobjektille, jotta ladattu malli saadaan näkymään ruudulla. Unityssa 3D-mallin sisältävä peliobjekti tarvitsee MeshFilter ja MeshRenderer -komponentit. MeshFilter sisältää kappaleen geometrian, kun taas MeshRenderer sisältää materiaalit ja piirtää kappaleen näytölle.

Unity vaatii, että mallilla on sama määrä kärkiä, UV-koordinaatteja ja normaaleja. Mallista luettujen arvojen määrä ei kuitenkaan ole välttämättä sama jokaisella komponentilla. Osa koordinaateista täytyy siis monistaa tarpeen mukaan. Koordinaattien määrät vaihtelevat, koska samaa koordinaattia voidaan käyttää useampaan kertaan. Indeksit viittaavat käy-

tettävään koordinaattiin, ja niiden määrä indeksityypistä riippumatta on aina sama, olettaen, että mallilla on UV-koordinaatit. Indeksit läpi käymällä saadaan täten mallille tarvittavat koordinaatit jokaiselle pisteelle. Kuvassa 20 käydään läpi koordinaattien ja kolmioiden asettaminen mallille pseudokoodina.

```
Lista tahkoista;
Hakemisto kombinaatioista;
Malli;

Käy läpi kaikki tahkot
{
    Kärki = Hae kärki-indeksiä vastaava kärkikoordinaatti();
    UV = Hae UV-indeksiä vastaava UV-koordinaatti();
    Normaali = Hae normaali-indeksiä vastaava normaalikoordinaatti();

    Jos kyseinen kombinaatio on jo olemassa
        Lisää tahkolistaan aiemmin löytyneen kombinaation indeksi;
    Jos kyseinen kombinaatio ei ole olemassa
    {
        Lisää kombinaatio hakemistoon;
        Lisää samaisen kombinaation indeksi tahkolistaan;
    }
}

Mallin kärjet = kombinaatioiden kärjet;
Mallin UV-koordinaatit = kombinaatioiden UV-koordinaatit;
Mallin normaalit = kombinaatioiden normaalit;
Mallin kolmiot = tahkot;
```

Kuva 20. Datan geometrian asettaminen pseudokoodina

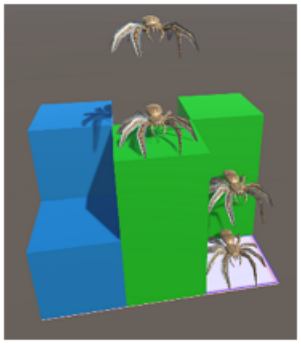
Mallille annetaan latausprosessin jälkeen editorin käyttämät lisäarvot, kuten nimi, ikoni ja koko. Koko annetaan ruutujen määrinä ja malli skaalataan mahtumaan annettuun ruutumäärään. Mallin uusi koko lasketaan etsimällä mallin leveydestä, korkeudesta ja syvyydestä suurin arvo, jolla mallin koko ensiksi jaetaan ja sen perään kerrotaan ruudun koolla ja käytettävien ruutujen määrällä.

Ladattu 3D-malli kopioidaan operaation päätteeksi kansioon, josta ohjelma tietää ladata aikaisemmin tuotuja malleja ohjelman käynnistyessä tai tasotiedostoa aukaistessa. Tiedoston kopioiminen takaa sen, että malli ei häviä kenttäeditorista, vaikka alkuperäinen tiedosto hävitettäisiin tai siirrettäisiin toiseen kansioon. Mallikansioon luodaan jokaiselle ladatulle mallille oma kansio, joka sisältää mallitiedoston lisäksi mallin ikonin, materiaalitiedoston, mallin tekstuurit, sekä kenttäeditorin käyttämät lisäparametrit. Samalla vaihdetaan myös malli- ja materiaalitiedoston sisältämät tiedostopolut vastaamaan tekstuurien ja materiaalitiedoston nykyistä sijaintia.

5.3.3 Kenttätiedostot

Kenttää tallennettaessa ohjelma kirjoittaa kenttätiedostoon tarvittavat tiedot kentän uudelleenrakentamiseksi. Tiedostoon tallennetaan ensimmäisenä kentän koko ja kentän muodostavien ruutujen koko. Tämän jälkeen tallennetaan mallien määrä tasossa ja erilaisten mallien määrä, jotka kertovat, montako riviä mitäkin datatyyppiä on. Seuraavana kirjoitetaan mallitiedostojen polut. Viimeisenä tallennetaan kaikkien kentässä olevien 3D-mallien tiedot. Mallista otetaan ylös niiden mallitiedostoon viittaava indeksi, positio, rotaatio ja skaala. Kuvassa 21 esiintyy osa tallennetusta kenttätiedostosta kommentoituna, sekä dataa vastaava kenttä kenttäeditorissa.

```
1  (3.0, 4.0, 3.0)                    // Kentän koko
2  (1.0, 1.0, 1.0)                    // Ruudun koko
3  12                                 // Mallien määrä tasossa
4  3                                  // Erilaisten mallien määrä
5  LevelEditor/Models/BlueBlock.obj   // Mallitiedoston polku
6  LevelEditor/Models/GreenBlock.obj
7  LevelEditor/Models/Spider.obj
8  0                                  // Mallin id
9  (0.0, 0.0, 1.0)                    // Mallin sijainti
10 (0.0, 0.0, 0.0)                    // Mallin rotaatio
11 (1.0, 1.0, 1.0)                    // Mallin skaala
12 0
13 (0.0, 1.0, 1.0)
14 (0.0, 0.0, 0.0)
15 (1.0, 1.0, 1.0)
16 0
17 (0.0, 0.0, 0.0)
18 (0.0, 0.0, 0.0)
19 (1.0, 1.0, 1.0)
```



Kuva 21. Lyhennetty kenttätiedosto ja sitä vastaava kenttä

Kenttää ladatessa tallennettu data otetaan talteen ja kenttäeditori etsii oikean mallin mallivalikosta tiedostopolun perusteella. Jos mallitiedostoa ei löydy mallivalikosta, tiedosto ladataan ja lisätään valikkoon. Varsinaisen kentän uudelleenluonti aloitetaan, kun kaikki mallit on ladattu. Ohjelma luo mittojen mukaisen tyhjän kentän, johon mallit sijoitetaan datan mukaan.

5.4 Testaus

Ohjelman testausta suoritettiin säännöllisin väliajoin. Testauksessa pyrittiin etsimään ohjelmasta ongelmakohtia käymällä editorin ominaisuudet yksi kerralla läpi, testaten mahdollisimman montaa rajatapausta, joissa ongelmia voisi ilmetä. Ohjelman performanssia seurattiin samalla myös Unityn tarjoamilla työkaluilla. Tällä tavalla huomattiin pikaisesti, jos ohjelman suorituskyvyssä tai toiminnassa oli ongelmia.

Ohjelmaa testatessa huomattiin toteutusprosessin alkupäässä ohjelmoidun ruudukon performanssiongelmat, joista päästiin eroon muuttamalla ruudukko yhdeksi teksturoiduksi objektiksi. Pieniä performanssiongelmia esiintyi myös ohjelmaa avatessa. Ohjelman auetessa kenttäeditori lataa aiemmin käytetyt mallit mallikansiosta ja lisää ne mallilistaan. Tämä prosessi vie sitä enemmän aikaa ja muistia, mitä enemmän ladattavia malleja on. Alla olevaa koodia optimoitiin ja ohjelmaan lisättiin latausruutu.

Ohjelman testauksen kannalta tärkeimmässä asemassa oli 3D-mallien lataaminen. Mallien lataukseen kuluvaa aikaa mitattiin kolmella eri mallilla. Mallien koostumuksesta otettiin ylös kärkien, tahkojen ja tekstuurien määrät. Jokainen malli sisältää UV-koordinaatit. Jokaisesta mallista mitattiin, kuinka kauan lataamisessa kuluu, kun samaa mallia ladataan erikseen 1, 10 ja 100 kappaletta. Ajan ottaminen aloitetaan ensimmäisen mallin avaamisesta ja lopetetaan, kun viimeinen malli on lisätty ohjelman malliluetteloon. Ajat esitetään sekunneissa kolmen desimaalin tarkkuudella.

Mallien latauksen tulokset on esitetty taulukossa 4. Tuloksista nähdään, että mallin lataus suoriutuu tyydyttävässä ajassa. Saman mallin lataukseen menee joka kerta suunnilleen saman verran aikaa, eikä virheitä tapahdu. Eniten aikaa kului mallissa 3 ja vähiten mallissa 2. Syyksi voidaan olettaa mallien sisältämien tahkojen määrä, sillä tahkojen asettaminen on latauksen eniten aikaa vievä prosessi. Ohjelma lataa suuria määriä malleja ainoastaan ohjelman auetessa tai kenttää ladatessa. Mallien latauksen ei näin ollen pitäisi tuottaa ongelmia.

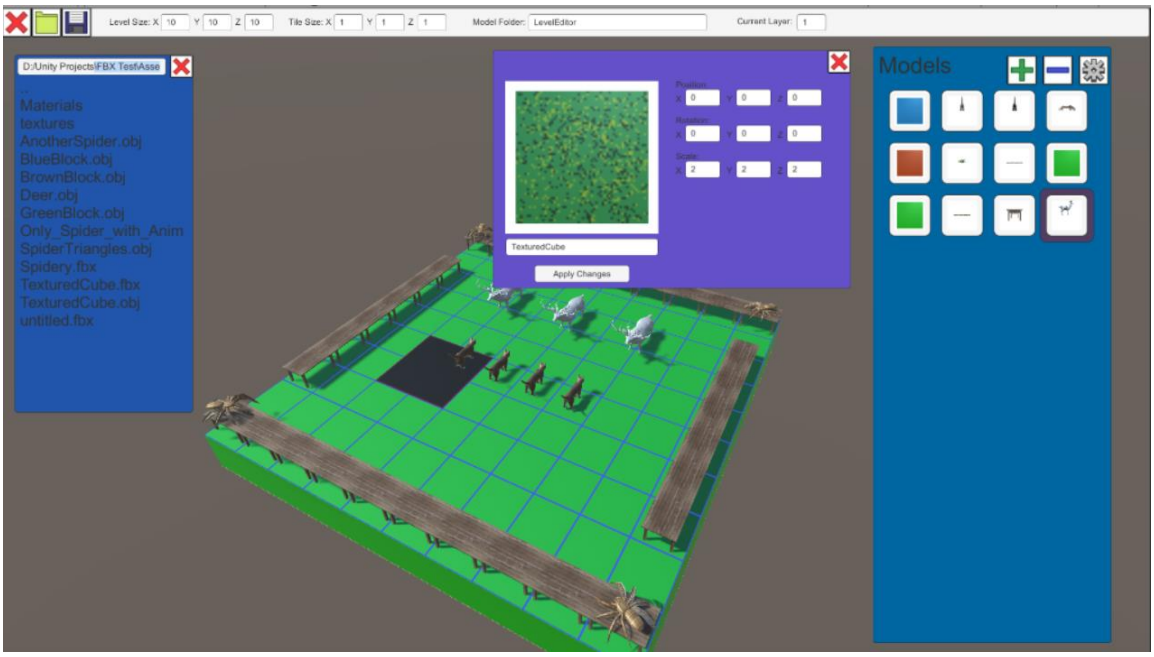
Malli 1:				Malli 2:				Malli 3:			
Kärjet:	2311			Kärjet:	4483			Kärjet:	2787		
Tahkot:	8676			Tahkot:	4509			Tahkot:	11862		
Tekstuurit:	5			Tekstuurit:	0			Tekstuurit:	1		
Määrä:	Aika 1 (s):	Aika 2 (s):	Aika 3 (s):	Määrä:	Aika 1 (s):	Aika 2 (s):	Aika 3 (s):	Määrä:	Aika 1 (s):	Aika 2 (s):	Aika 3 (s):
1	0,189	0,185	0,188	1	0,164	0,163	0,162	1	0,215	0,213	0,208
10	1,479	1,473	1,471	10	1,173	1,135	1,159	10	1,640	1,609	1,616
100	10,718	10,747	10,736	100	8,338	8,830	8,453	100	13,484	13,471	13,336

Taulukko 4. OBJ-tiedostojen lataukseen kuluneita aikoja

Lopputestauksessa ohjelmalla rakennettiin muutama kenttä kaikkia ohjelman ominaisuuksia hyödyntäen. Valmiit kentät tallennettiin ja avattiin uudelleen ohjelman uudelleenkäynnistyksen jälkeen. Kenttätiedostot pysyivät muuttumattomana, joten voitiin todeta, että ohjelma toimii halutulla tavalla.

6 Tulokset

Luotu kenttäeditori täyttää vaatimusmäärittelyn ehdot ja on valmis käyttöön otettavaksi. Editoriin on mahdollista tuoda mallinnusohjelmalla tehtyjä OBJ-muotoisia 3D-malleja, ja ladattuja malleja voidaan käyttää kentän rakentamiseen. Valmis kenttä voidaan tallentaa kenttätiedostoksi myöhempää käyttöä varten. Valmis kenttäeditori on esiteltynä kuvassa 22.



Kuva 22. Valmis kenttäeditori

6.1 Onnistumiset

Kenttäeditorin ohjelmointi oli mielenkiintoista ja opin uusia asioita etenkin 3D-malleista. Ohjelmoinnissa ei tullut suurempia esteitä vastaan. Aihe oli mielestäni hyvin mielenkiintoinen ja olin motivoitunut koko ohjelmistonkehitysprosessin ajan. Työn toimeksiantaja on tyytyväinen valmistuneeseen sovellukseen.

Kuten kuvasta 23 voi nähdä, kenttäeditorista tuli käyttöliittymältään ja toiminnallisuudeltaan suunnitelman mukainen. Käyttöliittymä on yksinkertainen ja helppokäyttöinen. Editori näyttää käyttäjälle, mahtuuko malli ruudukkoon ja estää käyttäjää asettamasta malleja virheellisesti.

6.2 Vastoinkäymiset

En pysynyt suunnittelemassani aikataulussa. Joidenkin ominaisuuksien ohjelmoimisessa meni odotettua kauemmin. Syynä tähän oli oletettavasti omien taitojeni yliarvioimisen ja tehtävien kompleksisuuden aliarvioimisen kombinaatio. Aikataulussa pysyminen oli helppoa projektin alkupuolella, mutta se muuttui joka kerta vaikeammaksi, kun jokin tehtävistä ei valmistunutkaan ajoissa. Jouduin jättämään osan suunnittelemistani ominaisuuksista pois ajan puutteen vuoksi. Kyseiset ominaisuudet eivät kuitenkaan olleet ohjelman toiminnan kannalta elintärkeitä, vaan kyseessä oli pienempiä kentän rakentamista helpottavia työkaluja.

Ohjelman rungon suunnitteluun olisi pitänyt käyttää enemmän aikaa. Kenttäeditoria suorittavan koodin hierarkia on hieman sekava ja koodin laatu vaihtelee. Kokonaisuuden hahmottaminen oli suunnitellessa haastavaa, mutta uskon, että kyseessä on taito, joka kehittyy kokemuksen myötä.

6.3 Kehityskohteet

3D-mallien latauksen yhteyteen olisi mahdollista lisätä mallin kolmiointi, joka poistaisi tarpeen kolmioida malli etukäteen mallinnusohjelman puolella. Olisin mielelläni lisännyt kolmiointimahdollisuuden, mutta jouduin jättämään sen pois aihepiirin laajuuden ja kompleksisuuden takia. OBJ- ja MTL-tiedostoissa on vielä parametreja, joita kenttäeditori ei hyödynnä. Lisäksi tukea voisi laajentaa myös muihin tiedostomuotoihin.

Mallien asettaminen maailmaan on työlästä suuremmissa tasoissa, koska malleja voi nykyisessä versiossa sijoitella vain yksi kerrallaan. Ohjelmaan voisi lisätä työkaluja, jotka tekisivät kentän tekemisestä helpompaa. Käyttöliittymään voisi myös lisätä käyttäjää varten ohjeistusta esim. näyttämällä, mitä kukin työkalu tekee, kun hiirtä pitää hetken työkalun päällä.

Ohjelmaa olisi vielä mahdollista optimoida. Mallien lataus suoriutuu suhteellisen nopeasti, mutta sen saisi varmasti toteutettua vielä nopeammin. Ohjelman muistinkäyttöä olisi myös mahdollista vähentää.

7 Yhteenveto

Työn tavoitteena oli luoda kenttäeditori TJR Games Oy:n pelinkehitystä varten. Kenttäeditorin vaatimuksiin kuuluivat kolmiulotteisten mallien lataus, niiden asettaminen kenttään ja kentän tallentaminen. Valmiin kentän tuli toimia Unity-pelimoottorilla tehdyissä peleissä, pelin alustasta riippumatta.

Kenttäeditorin toteutus jakautui perinteisen ohjelmistonkehitysprosessin neljään vaiheeseen: vaatimusmäärittelyyn, suunnitteluun, toteutukseen ja testaukseen.

Suunnitteluvaiheessa listattiin kaikki ohjelman tarvitsemat komponentit ja niille asetettiin tavoiteaikataulut. Ohjelman toteutuksessa eniten aikaa vei mallien lataamisen toteuttaminen. Ohjelman kehityksessä hyödynnettiin opinnäytetyön teoriapohjaa. Ohjelman ominaisuuksien toimivuutta testattiin säännöllisin väliajoin ja lopussa suoritettiin lopputestaus, jonka tulosten pohjalta korjattiin viimeiset ohjelmointivirheet.

Ohjelmasta tuli toiminnaltaan ja käyttöliittymältään suurimmaksi osaksi suunnitelmien mukainen. Suunnitteluvaiheessa laadittu aikataulutus ei kuitenkaan pitänyt projektin loppuun asti, joten muutama ohjelman käyttöä helpottava ominaisuus jouduttiin karsimaan pois.

Kenttäeditori pystyy joka tapauksessa suorittamaan jokaisen vaatimusmäärittelyssä asetetuista tehtävistä. Kenttäeditori tukee OBJ-tiedostomuotoa, ladattuja malleja on mahdollista asettaa joko kentän ruudukkoon tai vapaasti ympäristöön, ja tallennetut kentät ovat alustariippumattomia. Kenttäeditori on valmis käyttöönottoa varten.

Lähteet

1 Team Jolly Roger - home. Saatavilla: http://www.teamjollyroger.com/. Viitattu 4/4/2018, 2018.

2 Hexcells Fan-Made Editor Will Feed Your Hexual Appetite. 2014; Saatavilla: https://www.rockpapershotgun.com/2014/09/29/hexcells-fan-made-editor-will-feed-your-hexual-appetite/. Viitattu 20/4/2018, 2018.

3 Barton M. Honoring the Code: Conversations with Great Game Designers. : CRC Press; 2013.

4 Bunch K. Bizarre Game Mash-Up: A K.C.'s Krazy Chase Retrospective. 2010; Saatavilla: https://kotaku.com/5705553/bizarre-game-mash-up-a-kcs-krazy-chase-retrospective. Viitattu 13/4/2018, 2018.

5 Brudvig E. LODE RUNNER'S LEVEL EDITOR. 2009; Saatavilla: http://www.ign.com/articles/2009/04/07/lode-runners-level-editor. Viitattu 13/4/2018, 2018.

6 Pegasus II (Apple II). Saatavilla: http://www.mobygames.com/game/apple2/pegasus-ii. Viitattu 13/4/2018, 2018.

7 Smith DE, Bigham D. lode Runner. Brøderbund.Multiple Platforms 1983. Saatavilla: http://www.retrogames.cz/manualy/DOS/Lode_Runner_-_Manual_-_PC.pdf. Viitattu 19/4/2018, 2018.

8 LOCK'N'LODE. 1999; Saatavilla: http://www.ign.com/articles/1999/02/18/locknlode. Viitattu 15/4/2018, 2018.

9 Pemmaraju V. Make Your Life Easier: Build a Level Editor. 2012; Saatavilla: https://gamedevelopment.tutsplus.com/articles/make-your-life-easier-build-a-level-editor--gamedev-356. Viitattu 10/4/2018, 2018.

10 Haas J. A History of the Unity Game Engine. Worcester, UK: Worcester Polytechnic Institute 2014. Saatavilla: http://web.wpi.edu/Pubs/E-project/Available/E-project-030614-143124/unrestricted/Haas_IQP_Final.pdf. Viitattu 9/4/2018, 2018.

11 Gregory J. Game engine architecture. 1st ed.: AK Peters/CRC Press; 2009.

12 Unity - Store. Saatavilla: https://store.unity.com/. Viitattu: 22/4/2018, 2018.

13 Unity - Manual: Native Plugins. Saatavilla: https://docs.unity3d.com/Manual/NativePlugins.html. Viitattu: 9/4/2018, 2018.

14 Unity - Multiplatform - Publish your game to over 25 platforms. Saatavilla: https://unity3d.com/unity/features/multiplatform. Viitattu: 22/4/2018, 2018.

15 Cities: Skylines | Paradox Interactive. Saatavilla: https://www.paradoxplaza.com/cities-skylines/CSCS00GSK-MASTER.html. Viitattu 22/4/2018, 2018.

16 Unity - Manual: Understanding Automatic Memory Management. Saatavilla: https://docs.unity3d.com/Manual/UnderstandingAutomaticMemoryManagement.html. Viitattu 9/4/2018, 2018.

17 Hempel B. Optimizing Rails for Memory Usage Part 2: Tuning the GC. 2015; Saatavilla: https://collectiveidea.com/blog/archives/2015/02/19/optimizing-rails-for-memory-usage-part-2-tuning-the-gc. Viitattu 4/5/2018, 2018.

18 Unity - Optimizing garbage collection in Unity games. Saatavilla: https://unity3d.com/learn/tutorials/topics/performance-optimization/optimizing-garbage-collection-unity-games. Viitattu 9/4/2018, 2018.

19 Unity - Unity Enterprise. Saatavilla: https://store.unity.com/products/unity-enterprise. Viitattu 9/4/2018, 2018.

20 2011 National Film Registry More Than a Box of Chocolates. 2011; Saatavilla: https://www.loc.gov/item/prn-11-240/. Viitattu 10/4/2018, 2018.

21 Rizvi S. Ed Catmull's 'Computer Animated Hand' Added To National Film Registry. 2011; Saatavilla: http://pixartimes.com/2011/12/28/ed-catmulls-computer-animated-hand-added-to-national-film-registry/. Viitattu 10/4/2018, 2018.

22 Winning Run (Game) - Giant Bomb. Saatavila: https://www.giantbomb.com/winning-run/3030-40043/releases/. Viitattu 10/4/2018, 2018.

23 15 Most Influential Games of All Time. 2001; Saatavilla: https://web.archive.org/web/20100412225953/http://www.gamespot.com/gamespot/features/video/15influential/p13_01.html. Viitattu 10/4/2018, 2018.

24 Bourke P. Object Files (.obj) - Paul Bourke. Saatavilla: http://paulbourke.net/data-formats/obj/. Viitattu 9/4/2018, 2018.

25 Bourke P. MTL OBJ materials file - Paul Bourke. Saatavilla: http://paul-bourke.net/dataformats/mtl/. Viitattu 9/4/2018, 2018