



Osaamista  
ja oivallusta  
tulevaisuuden  
tekemiseen

Henri Tuupanen

# Kartanmuokkaustyökalu pulmanrat- kontapeliin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

20.11.2018

|   |  |
|---|--|
| Tekijä<br>Otsikko   | Henri Tuupanen<br>Kartanmuokkaustyökalu pulmanratkontapeliin |
| Sivumäärä<br>Aika   | 33 sivua<br>20.11.2018                                       |
| Tutkinto  | Insinööri (AMK)  |
| Tutkinto-ohjelma  | Tieto- ja viestintätekniikka                                 |
| Ammatillinen pääaine  | Pelisovellukset  |
| Ohjaaja   | Lehtori Miikka Mäki-Uuro                                     |
| <p>Insinööriyön tarkoituksena oli kenttäeditorin kehittäminen olemassa olevaan 2D-pulmanratkontapeliin, joka alkoi opiskeluprojektina ja on tarkoitus julkaista valmistuessaan. Kenttäeditori kehitettiin Unity-pelimootorin laajennokseksi ja samalla perehdyttiin Unityn tarjoamiin laajennusmahdollisuuksiin.</p> <p>Työssä tutustuttiin pelinkehitysprosessiin ja sen sisältämiin rooleihin sekä työkaluihin, joita prosessissa on eri aikakausina käytetty. Pelikenttien rakentamiseen on ollut hyödyllistä luoda oma työkalunsa, ja nykyään löytyy monia työkaluja auttamaan kentänluonnissa. Työssä tarkasteltiin Tiled-työkalun sopivuutta ja Unityn kauppapaikan tarjontaa. Varsinkin tasosuunnittelijan osuuteen ja tasosuunnittelun teoriaan ja käytäntöihin perehdyttiin, jotta kenttäeditorista saataisiin mahdollisimman käytännöllinen suunniteltuun rooliin. Huomattiin, että tasoja suunnitellessa kannattaa taso jakaa loogisiin osiin ja osat suunnitella aluksi erikseen.</p> <p>Kenttäeditorin osien toteutusta käsiteltiin tarkastelemalla Unitya laajentavien luokkien toteutusta ja yksityiskohtia. Työssä luotiin Unitya laajentavia CustomEditor-luokkia ja EditorWindow-luokan laajennos asetusten muuttamiseen. Kenttäeditorin toimintaa arvioitiin ja se todettiin hyväksi. Kehityskohteita ja vaihtoehtoisia toteutustapoja kuitenkin löytyi. Huomattiin, että kameran liikettä ohjaavia liipaisimia tulisi voida asettaa kenttäeditorissa ja että kenttäeditorin tarvitsemat luokat tulisi siivota pois valmiista kentistä.</p> <p>Insinööriyön lopputuloksena luotiin toimiva kenttäeditori, joka täyttää kaikki suunnittelussa asetetut vaatimukset. Kenttäeditoria on tarkoitus käyttää jatkossa projektissa, jolle sitä lähettiin kehittämään.</p> |  |
| Avainsanat  | tasosuunnittelu, pelisuunnittelu, Unity, työkalu             |

|   |   |
|---|---|
| Author<br>Title   | Henri Tuupanen<br>Developing level editor for puzzle game |
| Number of Pages<br>Date   | 33 pages<br>20 November 2018                              |
| Degree  | Bachelor of Engineering                                   |
| Degree Programme  | Information and Communication Technology                  |
| Professional Major  | Game Applications   |
| Instructor  | Miikka Mäki-Uuro, Senior Lecturer                         |
| <p>The goal of this final year project was to create a level editor for a 2D puzzle platformer game that already exists. The game started as a school project and will be published when it is ready. The level editor was developed as a Unity editor extension. During the process the expandability of the Unity editor was investigated in more detail.</p> <p>The thesis investigates the game development process, the roles in the process and the tools that have been used in the process throughout the times. It has been found useful to create a tool for building game levels and nowadays you can find many tools to assist in creating levels. The thesis evaluates how well the Tiled-editor could work with the project and investigates the tools that can be found at the Unity Asset Store. The role of the level designer and the level designing process are especially researched to understand what is required for a good level editor. It was found that it's good to split a level into logical parts and design those parts separately.</p> <p>The development of the different parts of the level editor was presented by looking at the execution and details of the classes that extend the Unity editor. Several CustomEditor-classes were created, and EditorWindow-class was extended for adjusting the settings of the level editor. The functionality of the level editor was evaluated and found acceptable. However, room for improvement and better ways to implement features were found. It was discovered that camera triggers, which control the movement of the camera should be placeable in the level editor. The classes the are needed by the level editor should be removed when the level is finished.</p> <p>In conclusion the result of the final year project was a working level editor, which fulfills all the requirements that were set for it. The level editor will be used in the project that it was developed for.</p> |   |
| Keywords  | level design, game design, Unity, tool                    |

## Sisällys

### Lyhenteet

|       |  |    |
|-------|--|----|
| 1     | Johdanto                                   | 1  |
| 2     | Pelinkehitys ja siinä käytettävät työkalut | 2  |
| 2.1   | Pelinkehitysprosessi                       | 2  |
| 2.2   | Pelinkehitystyökalut                       | 4  |
| 2.2.1 | Unity-pelimoottori                         | 4  |
| 2.2.2 | Työkalujen historiaa                       | 5  |
| 2.3   | Tasosuunnittelun menetelmät                | 8  |
| 2.4   | MushQuad-peli                              | 11 |
| 3     | Kenttäeditorin suunnittelu                 | 14 |
| 3.1   | Halutut ominaisuudet                       | 14 |
| 3.2   | Unityn editorin laajentaminen              | 16 |
| 4     | Kenttäeditorin toteutus                    | 18 |
| 4.1   | Maatasanteiden piirto                      | 19 |
| 4.2   | Pulmaelementtien luominen ja apugrafiikka  | 24 |
| 4.3   | Apuikkuna ja muut laajennokset             | 26 |
| 5     | Kenttäeditorin arviointi                   | 29 |
| 6     | Yhteenveto                                 | 30 |
|       | Lähteet                                    | 32 |

## Lyhenteet

|               |   |
|---------------|---|
| GDD           | Game Design Document. Dokumentti, johon on yksityiskohtaisesti listattu kaikki pelin suunnitellut toiminnot, näkymät ja se, miten peliä pelataan. Tuotantoryhmä toteuttaa pelin GDD:n pohjalta. |
| GUI           | Graphical User Interface. Graafinen käyttöliittymä.   |
| IMGUI         | Immediate Mode GUI. Unityn graafinen käyttöliittymä, joka luodaan ohjelmakoodilla. Tarkoitettu lähinnä ohjelmoijien aputyökaluksi.  |
| Singleton     | Ainokainen. Ohjelmiston suunnittelumalli, joka varmistaa, että yhdestä luokasta on olemassa vain yksi edustaja, johon on globaali pääsy.  |
| Tile          | Ruudukkopohjaisten järjestelmien yksi ruutu. Tekniikka 2D-grafiikan luomiseen pienistä neliön muotoisista ruuduista.  |
| Unity         | Pelimoottori, joka tukee pelien luomista helposti monelle eri alustalle.  |
| Unity-editori | Unityn graafinen käyttöliittymä.  |

## 1 Johdanto

Insinööriyön tarkoituksena on kehittää kartanluontityökalu laajennoksena Unity-pelimoottoriin. Insinööriyöraportissa esitellään 2D-pulmanratkaisupelin kartanluontiprosessia ja pelien tekovaiheessa käytettäviä työkaluja. Insinööriyössä suunnitellaan ja toteutetaan kartanmuokkaustyökalu MushQuad-peliin Unity-pelimoottorin laajennoksena. Kuvassa 1 näkyy MushQuad-pelin mainosjuliste.



Kuva 1. MushQuad-pelin sienihahmot.

Aihe valittiin, koska MushQuad-peliprojekti tarvitsee nopeamman tavan luoda pelikenttiä. Kenttäeditorin luonnin tarkoituksena on helpottaa kentänluontia Unityn editorin toiminnallisuutta laajentamalla. Tarkoituksena on myös oppia tuntemaan Unityn editorin toimintaa syvällisemmin ja tutkia sen laajennusmahdollisuuksien rajoja. Kenttäeditorin luonnin tavoitteena on muuttaa peliprojektin kentänrakennusprosessi hitaasta Unityn editorin sisällä tapahtuvasta käsityöstä nopeaksi kentän maaosien piirtämiseksi ja pulmaelementtien napin painalluksella sijoitteluksi ja yhdistelyksi.

Helppokäyttöiset pelimoottorit ja työkalut ovat aiheuttaneet pienten peliyritysten määrän räjähdysmäisen kasvun. Pelikauppapaikkoihin, esimerkiksi Steam-palveluun, ilmestyy useita uusia pelejä päivässä. Vuonna 2017 Steam-palveluun ilmestyi keskimäärin 21 uutta peliä joka päivä. Erottuakseen massasta uuden pelin on oltava poikkeuksellisen hyvä. Hyvät ja helposti käytettävät työkalut pelin luontivaiheessa auttavat pelikehittäjiä luomaan laadukkaita pelejä. [1.]

Työn luvussa 2 käsitellään pelinkehitysprosessia. Luvussa 3 listataan kenttäeditorilta halutut ominaisuudet ja käydään läpi Unityn editorin laajentamista yleisemmin. Luvussa 4 käydään läpi kenttäeditorin eri osien toteutus. Luvussa 5 arvioidaan lopputulosta ja pohditaan parannuskohteita. Luvussa 6 tehdään yhteenveto koko prosessista.

## **2 Pelinkehitys ja siinä käytettävät työkalut**

Pelinkehityksen roolit ovat monimutkaistuneet prosessin mukana. Luvussa 2.1 käydään läpi pelinkehitysprosessia ja sen rooleja. Myös työkalut ovat ajan myötä kehittyneet. Luvussa 2.2 käydään läpi pelinkehityksessä käytettäviä työkaluja. Tasojen luonnissa käytetyt menetelmät ja ymmärrys siitä, mikä tekee niistä viihdyttäviä, on myös kehittynyt. Luvussa 2.3 käydään läpi taso- ja pulmasuunnittelun menetelmiä. Luvussa 2.4 esitellään MushQuad-peli.

### **2.1 Pelinkehitysprosessi**

Pelinkehitysprosessi on monivaiheinen tapahtuma. Se pitää sisällään kaiken ensimmäisistä ideointitilanteista pelin julkaisuun asti. Täysin vakiintuneita vaiheita sisältöineen ei pelinkehitysprosessissa ole. Prosessi voidaan kuitenkin suurin piirtein jakaa kolmeen vaiheeseen. Vaiheet ovat esituotanto, tuotanto ja jälkituotanto. Esituotantovaiheessa pelisuunnittelijat suunnittelevat pelin toimintaa ja kirjoittavat siitä suunnitteludokumentin, joka tunnetaan nimellä Game Design Document eli GDD. Esituotantovaiheessa artistit luovat konseptitaidetta, jolla pyritään hahmottamaan pelin visuaalista ilmettä.

Pelinkehitysprosessin vaihe, jossa peliä toteutetaan, on tuotantovaihe. Tuotannon aikana käytetään erilaisia työkaluja pelin luomiseen. Pelin tekemiseen osallistuvilla henkilöillä on prosessissa erilaisia rooleja. Keskeisiä rooleja ovat

- pelisuunnittelija
- ohjelmoija
- artisti
- tasosuunnittelija.

Pienemmissä ryhmissä on yleistä, että yhdellä henkilöllä on useampia rooleja. Rooleissa käytetään rooliin sopivia työkaluja pelin tuotannon aikana. Pelisuunnittelijat ylläpitävät tuotantoprosessin aikana GDD:a ja valvovat, että peli noudattaa sovittua suunnitelmaa. Ohjelmoijat kirjoittavat ohjelmakoodin, joka toteuttaa peliin suunnitellun logiikan ja mahdollistaa pelin pelaamisen. Artistit luovat grafiikan peliin ja sen käyttöliittymään.

Jälkituotantovaiheessa julkaistua peliä ylläpidetään ja siihen saatetaan luoda uutta sisältöä. Julkaisun jälkeen havaittuja ohjelmointivirheitä korjataan julkaisemalla ohjelmistopäivityksiä.

Tässä insinööriyössä toteutettu työkalu tulee tasosuunnittelijoiden käyttöön. Tasosuunnittelijat luovat pelin ympäristöt, joissa pelaajat liikkuvat peliä pelatessaan. Tasosuunnittelijoiden työ on tärkeää, koska pelaajat ovat koko pelin ajan suoraan tekemisissä tasosuunnittelijoiden luomien ympäristöjen kanssa.

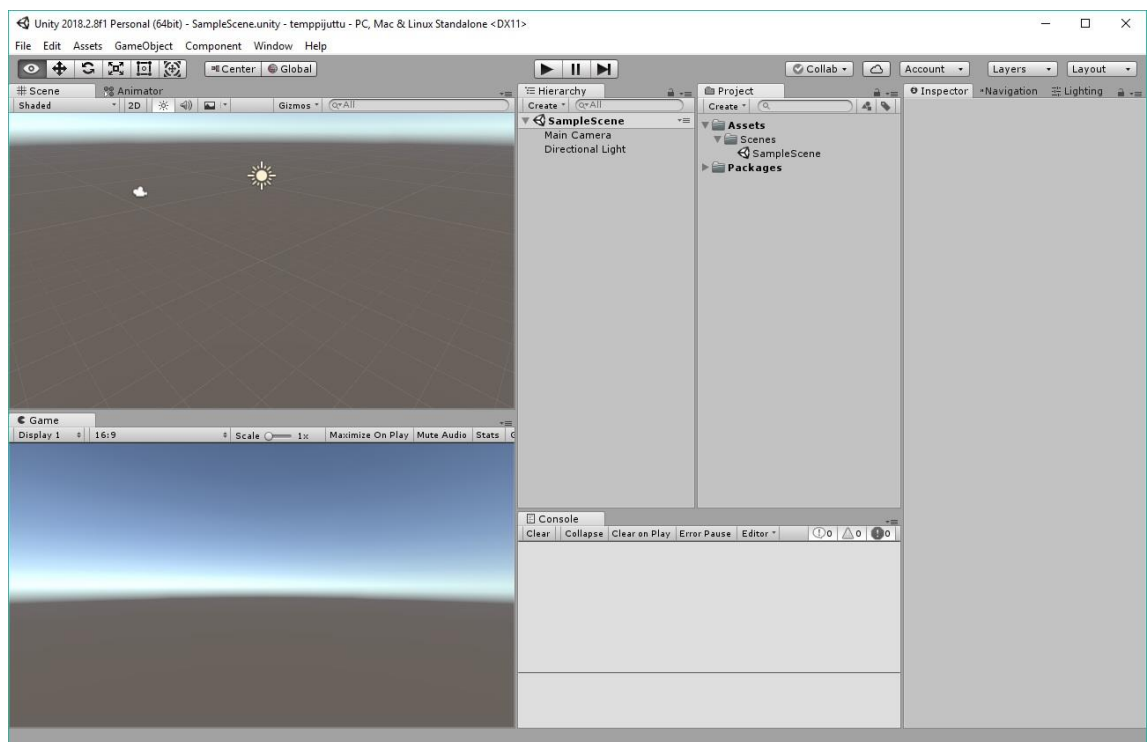
Joskus isommissa ryhmissä on myös erillinen työkaluohjelmoija, jonka tehtävänä on luoda ja ylläpitää muun ryhmän pelituotannossa käyttämiä työkaluja. Erikoistunut työkaluohjelmoija ei yleensä työskentele suoraan pelin parissa, vaan ohjelmoi ryhmän sisäisiä työkaluja. Pienemmissä ryhmissä pelillä saattaa olla vain yksi tai muutamia ohjelmoijia. Luonnollisesti tällaisissa tilanteissa ohjelmoijat luovat peliä ohjelmoidessaan myös muun ryhmän käyttämät työkalut.

## 2.2 Pelinkehitystyökalut

### 2.2.1 Unity-pelimoottori

Unity on Unity Technologiesin kehittämä monialustainen pelimoottori. Se on hyvin suosittu varsinkin pienempien pelinkehittäjien keskuudessa. Sen ensimmäinen versio julkaistiin kesäkuussa 2005, ja vuonna 2018 Unity tukee 27:ää eri alustaa. Yli puolet mobiilipeleistä tehdään Unitylla. [2.]

Unityn lisenssiehdot ovat hyvin houkuttelevia varsinkin pienille pelistudioille. Se on täysin rojaltivapaa. Unity on ilmainen niin kauan, kuin yrityksen tulot ovat alle 100 000 dollaria vuodessa. Sen jälkeen lisenssivaihtoehdot ovat Plus-lisenssi, jos yrityksen tulot ovat alle 200 000 dollaria vuodessa ja Pro-lisenssi, jos tulot ovat suuremmat. Plus-lisenssi maksaa 35 dollaria kuukaudessa yhtä ohjelmoijaa kohden, ja pro-lisenssi maksaa 125 dollaria kuukaudessa ohjelmoijaa kohden. [3.] Kuvassa 2 on Unityn editorinäkymä uudessa projektissa.



Kuva 2. Unity-pelimoottorin editorinäkymä [4].

Unitylla voi tehdä 2- tai 3-ulotteisia pelejä. Sen pääasiallinen skriptauskieli on C#. Itse Unity on kirjoitettu C++-ohjelmointikielellä. Unityn ohjelmointimalli perustuu GameObject-luokkaan. Kaikki pelin sisällä olevat asiat ovat GameObjecteja. Ne eivät itse sisällä mitään toiminnallisuutta, mutta niihin voi liittää komponentteja (engl. component). Unity tarjoaa itse monia keskeisiä komponentteja, kuten valoja, äänilähteitä ja törmäyspintoja (engl. collider). Pelin logiikan ohjelmoiminen vaatii kuitenkin omien komponenttien kirjoittamista. Tämä tehdään luomalla uusi luokka, joka perii MonoBehaviour-luokan. Uusi luokka on perinnän jälkeen uusi komponentti, jonka ohjelmoija voi ohjelmoida tekemään pelissä vaadittuja asioita. Unity-pelissä voi toki käyttää myös tavallisia C#-luokkia, mutta vaikuttaakseen peliin niiden on jollakin tavalla vuorovaikutettava Unityn komponenttien ja GameObjectien kanssa. Myös itse Unityn editoria voi laajentaa luomalla siihen omia ohjelmalaajennoksia. Unityn editorin laajentamista käsitellään lisää luvussa 3.2.

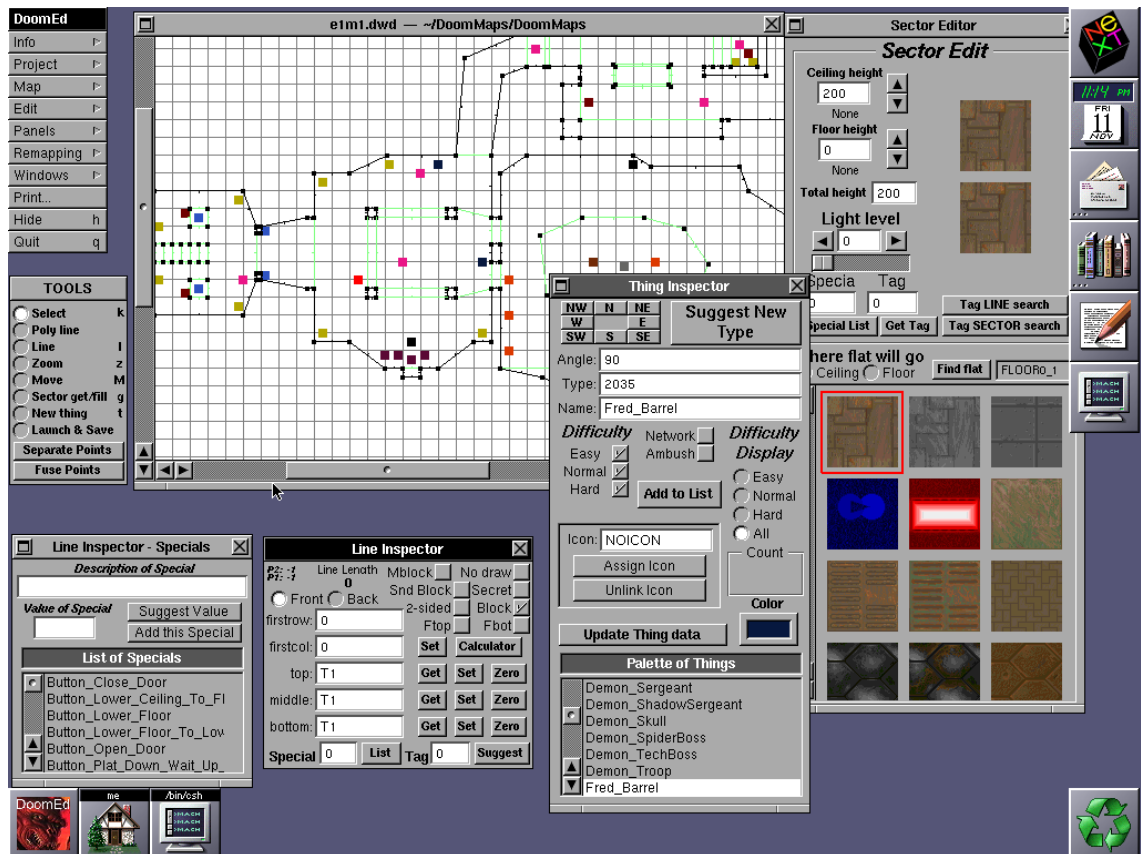
### 2.2.2 Työkalujen historiaa

Pelinkehitystyökalu on ohjelma, joka avustaa tai helpottaa pelinkehitystä. Työkalut voivat hoitaa esimerkiksi tekstuurien, 3D-mallien ja äänitiedostojen tuomista peliin sekä kenttien muokkaamista ja lähdekoodin kääntämistä. Unity-pelimoottori on määritelmältään pelinkehitystyökalu. Useita kaupallisia ohjelmia, kuten 3D Studio Maxia ja Photoshopia käytetään usein pelinkehitysprosessissa, mutta ne eivät kuitenkaan ole pelinkehitystyökaluja, koska ohjelmille on monia muitakin käyttötarkoituksia pelinkehityksen ulkopuolella.

Pelinkehitysprosessin monimutkaistuessa myös prosessissa käytettävät työkalut ovat kehittyneet ja monimutkaistuneet. Aikoina, jolloin pelit olivat yhden henkilön kehittämiä, ei käytetty koodieditoria monimutkaisempia työkaluja pelien luomiseen. Pelien monimutkaistuessa alkoivat kehittäjät kuitenkin luoda omia työkalujaan pelinkehitysprosessin aikana pelin luomisprosessin helpottamiseksi. Pelikonsolivalmistajat luovat usein työkalut, joilla valmistajan konsolille voi kehittää pelejä [5].

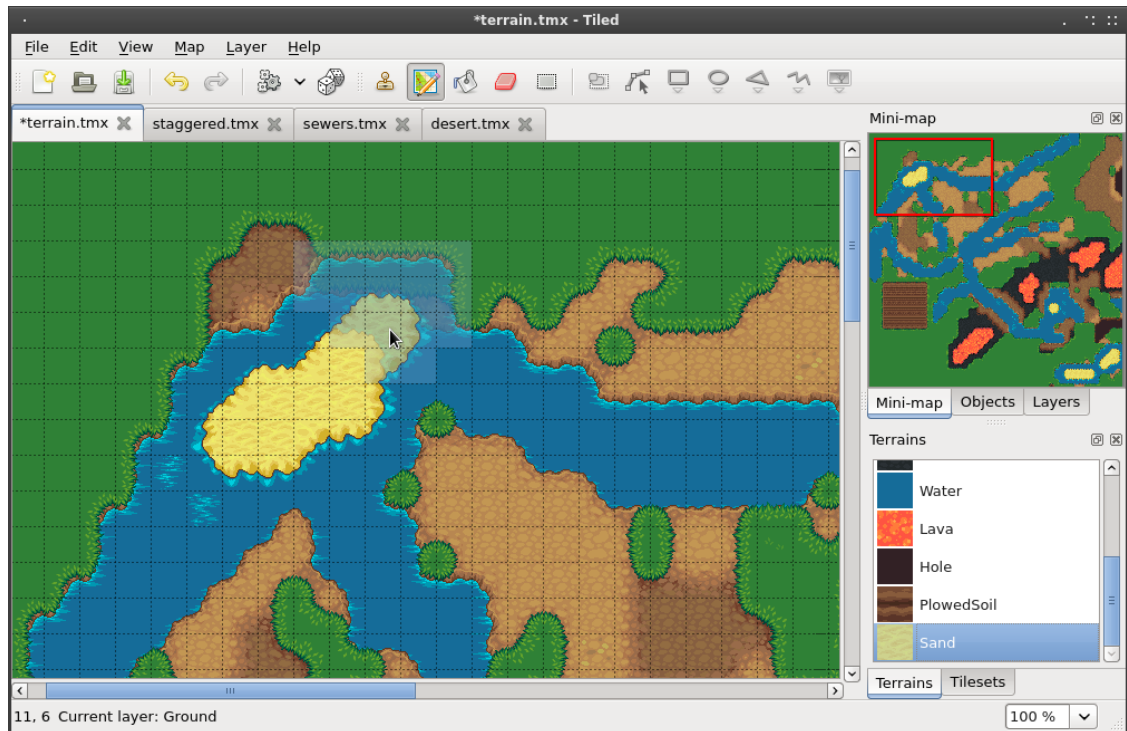
Vuonna 1993 ilmestynyt Doom nosti ensimmäisen persoonan räiskintäpelit suureen suosioon. Doomia pidetään yhtenä merkittävimmistä ja vaikutusvaltaisimmista peleistä. Doomien kehitystiimiin kuului seitsemän henkilöä. [6.]

Ohjelmoija John Carmack loi suurimman osan Doomiin käyttämästä pelimoottorista, ja ohjelmoija John Romero kirjoitti DoomEd-nimisen kenttäeditorin kenttäsuunnittelijoiden käyttöön. Kuvassa 3 näkyy DoomEd-kenttäeditori. Doomin pelimoottorin kentät olivat monimutkaisia rakennelmia. Seinät, katot ja lattiat olivat teksturoituja, ja lattiasektoreilla oli eri syvyyksiä Näiden yksityiskohtien takia kartat olisi ollut lähes mahdotonta kovakoodata ohjelmakoodin sisään. Kenttäeditori mahdollisti kenttien nopean luomisen ja niiden muokkaamisen pelin kehityksen aikana. [7.]



Kuva 3. DoomEd-kenttäeditori NeXTSTEP-käyttöjärjestelmässä [8].

Nykyään on olemassa myös pelimoottorista riippumattomia kenttäeditoreja, joissa luodut kartat tuodaan sisään pelimoottoriin jossakin pelimoottorin ymmärtämässä muodossa sen jälkeen, kun kartta on tehty valmiiksi kenttäeditorilla. Kuvassa 4 näkyy Tiled-editorin päänäkömä.

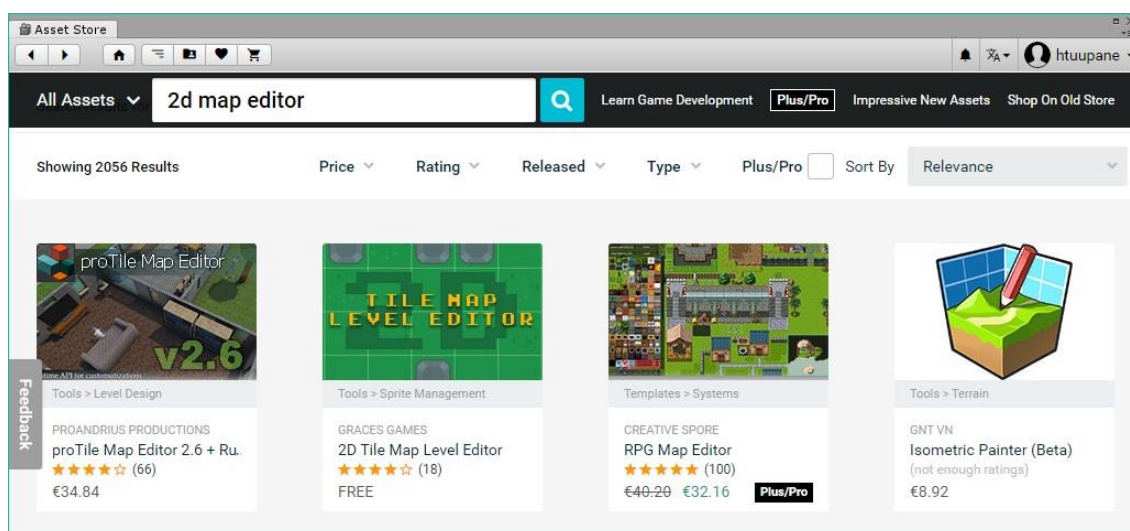


Kuva 4. Tiled-kenttäeditori [9].

Tiled on avoimen lähdekoodin kaksiulotteinen kenttäeditori. Se on tarkoitettu ruudukko-pohjaisten pelikenttien tekemiseen. Se tukee myös isometrisiä ja heksagonisia kenttiä. Tiledissä kentät maalataan maasiveltimillä eri kenttäkerroksille. Kenttäkerrosten lisäksi tiledissä on myös objektikerroksia, joihin voi sijoittaa monenlaisia asioita, kuten esimerkiksi neliöitä, ellipsejä, tekstiä ja kuvia. Objekteilla voi määrittää esimerkiksi alueita ja koordinaatteja, joita pelimoottorissa käytetään pelimekaniikkojen osana. [10.]

Tiled ei kuitenkaan soveltunut MushQuadin kenttien tekoon, koska MushQuadin kentät eivät ole tiukan ruudukkopohjaisia. Kenttien tasanteet ovat mielivaltaisesti määriteltävissä paikoissa ja mielivaltaisen kokoisia.

Unityn editorissa on oma sisäinen kauppapaikka (engl. Asset store), jossa on myynnissä käyttäjien luomia digitaalisia resursseja, esimerkiksi ohjelmia, 3D-malleja, tekstuureja, ääniä ja editorilaajennoksia. Karttaeditorit ovat usein hyvin hyödyllisiä työkaluja, ja syyskuussa 2018 tehdyllä ”2d map editor” -haulla tuleekin 2 056 osumaa, kuten kuvasta 5 näkyy.



Kuva 5. Unityn kauppapaikasta löytyviä kenttäeditoreja [11].

Kauppapaikan tarjonta on monipuolista: kaupasta löytyy kenttäeditoreja niin roolipeleille kuin maailmankartalle. Suuri osa monipuolisemmista työkaluista on kuitenkin maksullisia, eikä MushQuadiin haluttu käyttää rahaa, jos se ei ollut aivan pakollista. Suurin ongelma kuitenkin oli se, että editorit olivat Tiled-editorin tapaan ruudukkopohjaisia. Koska MushQuadissa haluttiin käyttää mielivaltaisesti lomittain sijoiteltavia tasanteita, päädyttiin luomaan oma kenttäeditori, joka soveltuu täysin MushQuadin kenttien luomiseen.

### 2.3 Tasosuunnittelun menetelmät

Tasosuunnittelu on prosessi, jossa suunnitellaan ja toteutetaan pelattava osakokonaisuus peliin. Prosessissa täytyy ymmärtää rajoitteet, joita tasoilla on. Mahdollisia rajoitteita ovat esimerkiksi tason pituus, eli kuinka pitkään sen suorittaminen kestää, ja se, miten pitkällä se on pelin kulussa. Alkupään tasojen tulisi olla helpompia kuin pelin lopun tasojen. Tason sijainti myös määrittää, mitä paloja, esimerkiksi pulmaelementtejä, tasossa on. Muita tärkeitä rajoitteita voivat olla asiat, joita tasolta vaaditaan tarinan, teeman ja juonen kannalta. Kuvassa 6 näkyy tasoluonnos.



Kuva 6. MushQuad-pelin esittelytason alun suunnitelma.

Tasojen luonti on hyvä aloittaa luonnoksella siitä, miltä taso tulee valmiina näyttämään. Kaksiulotteisten pelien tasojen suunnittelun voi aloittaa luonnostelemalla tasoja kynällä ja paperilla tai piirto-ohjelmassa. Taso kannattaa jakaa alueisiin, jotka suunnitellaan erikseen. Pulmanratkaisupelissä yksittäinen alue on yleensä yksi pulma tai pulmakokonaisuus, jonka pelaajat joutuvat ratkaisemaan. [12.]

Kun suunniteltu taso on valmis ja se on luotu suunnitelmien pohjalta peliin, sitä päästään testaamaan pelin sisällä. Usein tässä vaiheessa havaitaan paljon asioita, joita ei suunnitteluvaiheessa huomattu. Myös paljon parannusehdotuksia yleensä kertyy, kun tasoa hetken pelaa. Hyvin suunniteltu taso tarjoaa kuitenkin hyvän pohjan muutoksille ja parannuksille. Voisikin sanoa, että tasojen testaus ja arviointi on puolet koko suunnittelu-prosessista. [12.]

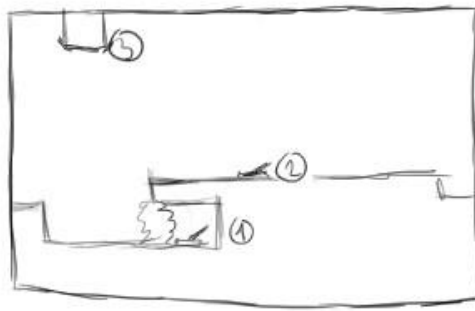
Hyvin suunniteltu pulma tarjoaa pelaajalle sopivan älyllisen haasteen ja onnistumisen tunteen, kun pelaaja ratkaisee sen. On tärkeää kiinnittää huomiota siihen, että pelaaja ymmärtää ratkaisevansa pulmaa. Pahimmassa tapauksessa pelaaja ratkaisee pulman huomaamatta edes pulman olemassaoloa. Tämä saa pelaajan tuntemaan tyytymättömyyttä: olisihan pelaaja osannut ratkaista pulman, jos olisi ollut tietoinen sen olemassaolosta. Toisessa ääripäässä pelaaja yrittää ratkaista pulmaa kokeilemalla asioita satunnaisesti, kunnes pulma ratkeaa. Tämä johtuu useimmiten siitä, että pelaaja ei ymmärrä pulman tavoitetta eikä tiedosta, miten pulman ratkaisuun liittyvät pelielementit vaikuttavat pelitilanteeseen.

Pulmaa suunnitellessa tulee aluksi varmistua siitä, että pelaaja ymmärtää, mitä pulmassa yritetään ratkaista. Esimerkiksi pelaajien etenemisen voi estää kuilu, jonka yli pelaajien on jotenkin päästävä. Tämän jälkeen pelaajien tulee huomata pulman ratkaisuun johtavat asiat, esimerkiksi vivut ja napit. Aluksi pelaajan on siis nähtävä ongelma ja sen jälkeen pulman ratkaisuun johtavat asiat. Tämän jälkeen pelaajan on ymmärrettävä, miten pulman ratkaisevat asiat vaikuttavat toisiinsa. Pelaajan tulee pystyä ratkaisemaan pulma järjeilemällä. Näin vältetään turhilta satunnaisten ratkaisujen kokeiluilta. Lopuksi pelaajan tulee pystyä toteuttamaan ratkaisunsa pelissä ja jatkamaan pelissä etenemistä. [13.]

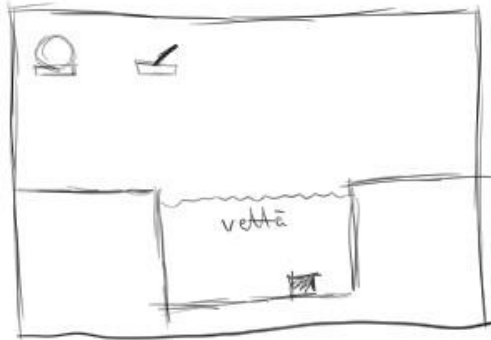
Pulman suunnittelun voi aloittaa monella tapaa. Pulmia, kuten kenttiäkin, voi luonnostella aluksi paperille. Kuvassa 7 näkyy pulmaluonnoksia. Aluksi pulman lähtökohta kannattaa päättää. Pulman lähtökohtana voi olla

- tutkia tietyn mekaniikan käyttöä
- opettaa pelaajalle jotakin uutta
- lukita pelaaja alueelle ja antaa pelaajan löytää tie ulos
- yhdistää pelimekaniikkoja.

Tasojuttuja



- ① Vipu pensaam takana (Pavo)
- ② Vipu ylläältä, vipu 2 (nyppää Pavan päältä) ihmesky
- ③ Hissi laskeutuu



- Viivusta vesi laskee (viivalle pääsee Pavan kautta)
- Veden pohjalla on nappula
- Ylläältä on kivi jota laitetaan nappulan päälle
- Hissillä ylös

Kuva 7. Pulmien luonnoksia.

Pulman luontia voi lähestyä käänteisessä järjestyksessä. Aluksi luodaan pulman lopputilanne ja siihen johtavat askeleet suunnitellaan lopusta alkuun liikkuen. Vaikeusastetta pulmiin voi saada hämärtämällä pulman todellinen ratkaisu. [14.]

Lopulta pulmaa voi testauttaa pelaajilla. Tulee muistaa, että vain se, että pelaaja ratkaisee pulman, ei tee siitä onnistunutta. Nautinto pulmissa tulee niiden ratkaisusta. Jos ratkaisun ymmärtää vasta jälkikäteen, ei pulmaa voi pitää onnistuneena. Olisi hyvä, jos suunnittelija näkisi, miten pelaaja ratkaisee pulman. Vielä parempi olisi, jos suunnittelija voisi istua pelaajan vieressä ja kuulla pelaajalta reaaliajassa, mitä pelaaja ajattelee pulmaa ratkoessaan. [13.]

## 2.4 MushQuad-peli

MushQuad on pulmanratkontatasohyppelypeli, jota voi pelata yhdestä neljään pelaajaa. Pelissä on värikäs grafiikka, ja se on suunnattu lapsille ja nuorille. Peli on suunniteltu

moninpeliksi. Sitä voi pelata paikallisesti usealla ohjaimella tai verkossa. MushQuad alkoi opiskeluprojektina. Pelin tekemiseen on ottanut osaa kuusi opiskelijaa Metropolia-ammattikorkeakoulusta. MushQuad on toteutettu Unity-pelimootorilla. Kuvassa 8 on nähtävissä pelin sienihahmot.



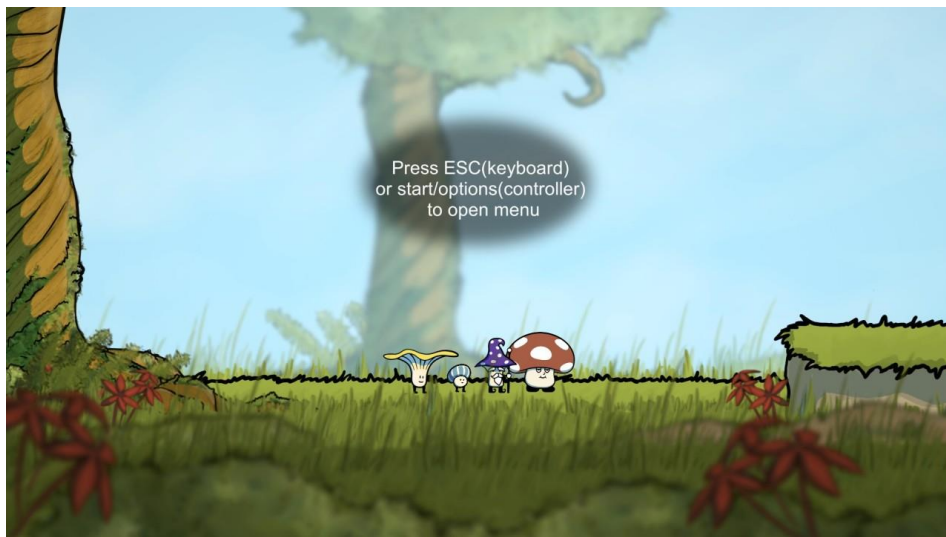
Kuva 8. Pelihahmot Pavo, Abå, Taro ja Sipe MushQuad-pelistä.

Pelissä on neljä pelattavaa sientä, joilla jokaisella on omat erikoiskykynsä. Näitä erikoiskykyjä käyttäen pelaajat ratkovat pelitasojen pulmia. Taro voi liittää ilmassa ja kykenee katkomaan köysiä. Pavo voi toimia trampoliinina ja auttaa muita sieniä hyppäämään korkeammalle. Pavo voi myös tuhota tiettyjä kenttien esteitä hyökkäyksellään. Sipe sylkee vettä ja voi esimerkiksi pyörittää vesipyöriä, kuten kuvan 9 oikeassa puoliskossa on nähtävillä. Abå kykenee nostamaan ja kantamaan esineitä, kuten kuvan 9 vasemmassa puoliskossa. Jokaisella sienellä on myös hyökkäyskyky, jolla pelaajat voivat päihittää pelikentässä olevia vihollisia. Pelissä huomio on pelaajien keskinäisessä yhteistyössä, sillä jokaisen eri pelaajan erikoiskykyjä tarvitaan kentän pulmien ratkaisemiseen.



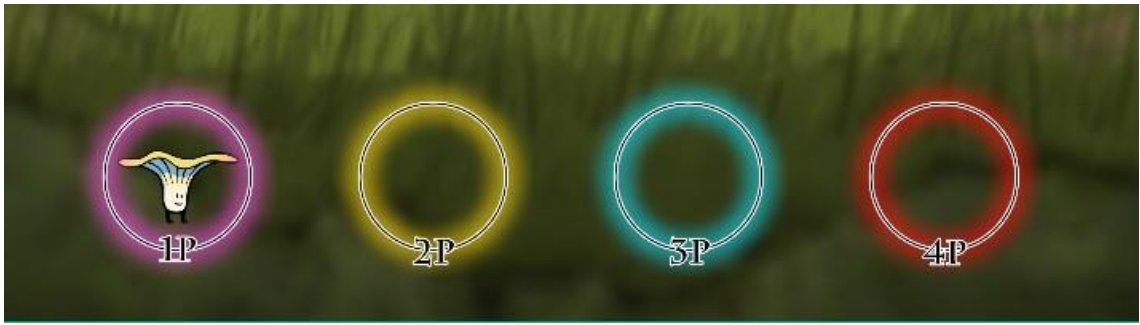
Kuva 9. Pelitilanteita MushQuad-pelistä.

MushQuadin ajatuksena oli luoda kaikille helposti lähestyttävä moninpeli, jota kuka tahansa osaamistasosta riippumatta voi pelata. Teemaksi valittiin värikäs ja piirroselukuvamainen metsäseikkailu ja söpöt pelihahmot. Kuvassa 10 nähdään MushQuad-pelin alunäkymä. Teeman tarkoituksena oli saada pelaajat välittämään metsän ja sienihahmojen kohtalosta ja uppoutumaan pelin fantasiamaailmaan. Peli sopii lasten keskenään pelattavaksi, mutta se sopii myös aikuisille pelaajille. Pelin tarkoitus on olla kokemus, jossa pelaajat eivät kilpaile tai taistele toisiaan vastaan, vaan pelissä etenemiseksi pelaajien tulee toimia yhdessä.



Kuva 10. MushQuad-pelin alunäkymä.

MushQuad-pelin käyttöliittymästä päätettiin tehdä mahdollisimman pelkistetty. Pelin kuussa näytöllä ei näy mitään käyttöliittymän osia, ellei pelitilanne muutu. Kuvassa 11 näkyy, kuinka pelaaja 1 on valinnut sienekseen Taron. Pelaajilla 2, 3 ja 4 ei ole sientä valittuna, eli pelaaja 1 pelaa peliä yksin. Jos muut pelaajat liittyisivät peliin ja valitsisivat sienen, heidän valitsemansa sieni näkyisi valintakäyttöliittymässä. Sienenvaihtokäyttöliittymä ilmestyy näytölle, kun pelaaja vaihtaa sientä, jolla pelaa. Muutaman sekunnin kuluttua valinnasta sienenvaihtokäyttöliittymä katoaa näytöltä.



Kuva 11. MushQuad-pelin graafinen käyttöliittymä.

Pulmien lisäksi pelissä on myös vihollisia, joiden ohitse pelaajien tulee tavalla tai toisella päästä. Myös viholliset on suunniteltu värikkään piirroselokuvamaisiksi ja suhteellisen helpoiksi päihittää. Taistelun ei suunniteltu olevan pääosassa pelissä. Tasokokonaisuuksien loppuun kuitenkin suunniteltiin loppuvastustaisteluita, jotka pelaajien tulisi ratkaista pulmien tapaan.

### 3 Kenttäeditorin suunnittelu

Kenttäeditoria suunnitellessa tulee ottaa huomioon tasosuunnittelijoiden tarpeet sekä pelin erityisominaisuudet, jotka vaikuttavat kenttien toteutukseen. Parhaimmillaan hyvä kenttäeditori nopeuttaa pelikenttien luomista huomattavasti.

#### 3.1 Halutut ominaisuudet

Kenttäeditorin suunnittelu aloitettiin selvittämällä, mitä ominaisuuksia tasosuunnittelijat pitivät tärkeinä ja mitkä ominaisuudet nopeuttaisivat kenttien luomista. Aluksi suunniteltiin myös erillistä valmiin pelin mukana olevaa kenttäeditoria, jota käyttämällä pelaajat voisivat luoda omia kenttiä. Erillisestä kenttäeditorista kuitenkin luovuttiin ajanpuutteen takia. Erillisen editorin luominen olisi vaatinut sen kysymyksen ratkaisua, miten tasot tallennetaan ja ladataan tiedostoon. Myös mahdollisuus luotujen kenttien jakamiseen muiden pelaajien kanssa olisi pitänyt toteuttaa jollakin tavalla. Jo nämä asiat olisivat vaatineet paljon työtä ohjelmakoodin luomisessa ja muokkauksessa.

Keskeinen asia, jota pidettiin tärkeänä, oli pelin maatasanteiden helppo luonti. Se päätettiin toteuttaa Unityn editoria laajentamalla luomalla mahdollisuus piirtää suorakulmioita hiirellä. MushQuadin tasojen geometria perustuu yksinkertaisiin suorakulmaisiin maatasanteisiin, joten suorakulmioita piirtämällä tasosuunnittelijat saavat helposti luotuja haluamansa muotoisia tasanteita.

Toinen tärkeäksi noussut asia on pulmien helppo luonti. Se sisältää pulmaelementtien helpon sijoittelun pelikenttään ja pulmaelementtien yhdistelyn pulmien logiikkaa luotaessa. Tämä päätettiin toteuttaa luomalla pikavalikko, josta tasosuunnittelijat voisivat nopeasti valita haluamansa pulmaelementin ja pudottaa sen kenttään. Kun pulmaelementti valitaan Unityssa, suunniteltiin esiin tulevan apugrafiikkaa, joka näyttäisi, miten pulmaelementti on yhdistetty muihin pelikentän osiin. Pulmaelementin valittuna ollessa suunniteltiin myös olevan mahdollista yhdistää valittu elementti muihin elementteihin yksinkertaisesti nappia painamalla.

Kolmas tärkeä osa kenttäeditoria oli tallennuspisteiden helppo sijoittelu kenttään ja toisaalta sienet tappavien sijaintien helppo piirto kenttään. Tallennuspisteitä haluttiin voida luoda näppäimen painalluksella. Tappavat sijainnit taas haluttiin voida piirtää Unityn editorinäkömään niin kuin maatasanteet. Taulukossa 1 on listattu kenttäeditorilta vaaditut ominaisuudet.

Taulukko 1. Kenttäeditorilta halutut ominaisuudet.

| Ominaisuus                    |
|-------------------------------|
| Maatasanteiden piirto         |
| Pulmaelementtien lisääminen   |
| Pulmaelementtien yhdistely    |
| Tallennuspisteiden sijoittelu |
| Tappavien sijaintien piirto   |

Kenttäeditorin luomisen rajoitteeksi asettui aika. Kenttäeditori päätettiin ohjelmoida kesän aikana, jotta syksyllä sen teknisen toteutuksen pohjalta voisi kirjoittaa opinnäytetyön. Kesän jälkeen myös tasosuunnittelijoilla olisi aikaa käyttää kenttäeditoria, kun kesätyöt päättyvät ja opiskelu jatkuu.

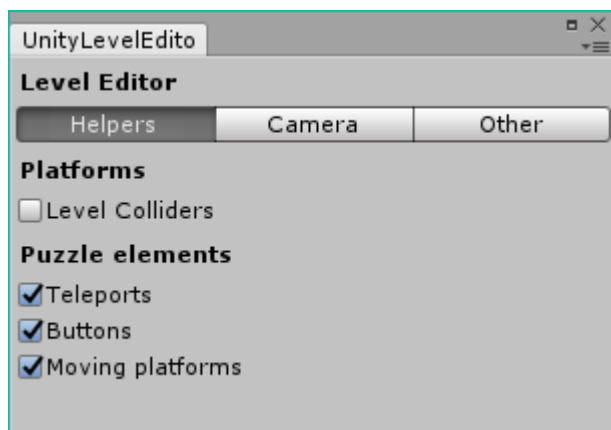
### 3.2 Unityn editorin laajentaminen

Unity tarjoaa editorinsa laajentamiseen monia tapoja. Jotta editoria laajentaviin luokkiin pääsee käsiksi, tulee laajennoksen ohjelmakooditiedostossa ottaa käyttöön UnityEditor-nimiavaruus. Unityn editoriin voi luoda kokonaan omia ikkunoita luomalla luokan, joka perii EditorWindow-luokan. Jotta ikkuna saadaan näkyviin, sille tulee luoda oma valinta Unityn pudotusvalikkoon. Tämä tehdään luomalla staattinen metodi ja lisäämällä MenuItem-attribuutti luodulle metodille, kuten koodiesimerkistä 1 nähdään. Ikkuna on kätevä tapa luoda käyttöliittymä pelin alijärjestelmille. Kuvassa 12 näkyy ikkuna kartanluontityökalulle.

```
[MenuItem ("Window/My Window")]
public static void ShowWindow () {
    EditorWindow.GetWindow (typeof (MyWindow)) ;
}
```

Esimerkkikoodi 1. MyWindow-tyypin ikkunan näyttävän komennon lisääminen pudotusvalikkoon.

Ikkunan käyttöliittymä luodaan kirjoittamalla ohjelmakoodia OnGUI-metodiin. Se ohjelmoidaan samoja luokkia käyttäen kuin muutkin IMGUI-käyttöliittymää käyttävät Unityn osat.



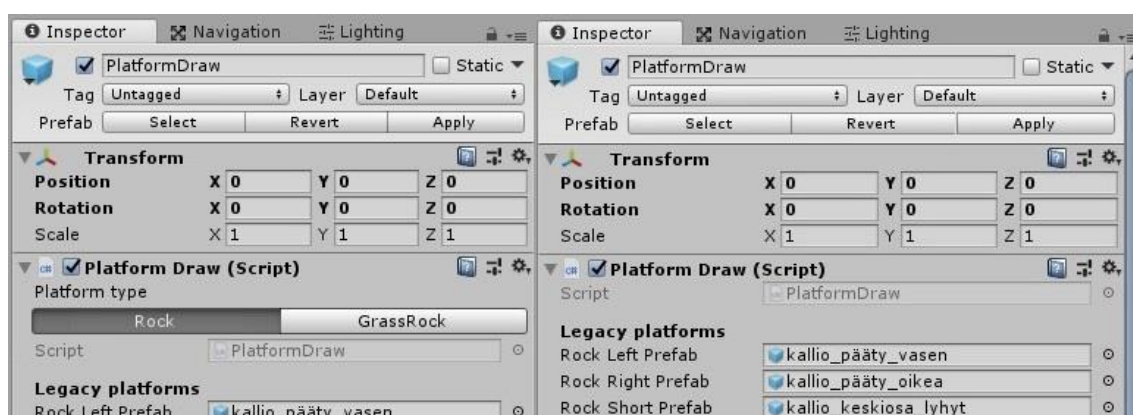
Kuva 12. Kartanluontityökalun apuikkuna.

Toinen tapa muokata Unityn editoria on määrittää itse, miten tarkasteluelementissä näytettävät luokat, jotka täyttävät Unityn serialization-ominaisuuden vaatimukset, näyttävät. Ominaisuuden piirtäjä (engl. Property drawer) liitetään haluttuun luokkaan antamalla

CustomPropertyDrawer-attribuutille luokka, jolle halutaan luoda ominaisuuden piirtäjä. Tämän jälkeen luodaan luokka, joka perii PropertyDrawer-luokan. Uuden luokan OnGUI-metodiin kirjoitetaan ohjelmakoodi, joka piirtää liitetyn luokan tarkasteluelementtiin, niin kuin ohjelmoija haluaa.

Unity tarjoaa myös tavan muokata sitä, miten koko tarkasteluelementti näytetään tietylle komponentille. Tämä tehdään luomalla uusi luokka, joka perii Unityn Editor-luokan. Luokan ylle asetetaan CustomEditor-attribuutti, joka osoittaa, minkä komponentin tarkasteluelementtinä sitä käytetään. Luokan sisältävä ohjelmakooditiedosto on sijoitettava erityiseen Editor-nimiseen kansioon, jotta Unity ymmärtää sen olevan osa editoria. Editor-luokan laajennokset eivät kuitenkaan säilytä muuttujiensa arvoja, vaan aina kun tarkastelijasta siirrytään pois, objekti lakkaa olemasta. Jos joitain muuttujien arvoja halutaan pitää muistissa, ne tulee tallentaa siihen luokkaan, jonka tarkastelija editorilaajennos on.

Kuvassa 13 näkyy, miten editorin valitun kappaleen tarkasteluelementtiin on vasemmanpuoleisessa elementissä lisätty valinta kivisen ja sammaleisen maatasannetyypin välillä. Oikealla elementti sellaisena, miltä se ilman laajennoksia näyttää.



Kuva 13. Unityn editorin tarkasteluelementin laajennos.

Tarkasteluelementin laajentaminen on tehty helpoksi Unityssa. Elementtiin piirrettäviin asioihin voi suoraan vaikuttaa OnInspectorGUI-metodia muuttamalla. Esimerkkikoodissa 2 näkyy yksinkertainen tarkasteluelementin laajennos.

```
// Creates a custom Label on the inspector for all the scripts named Script-
// Name
// Make sure you have a ScriptName script in your
```

```
// project, else this will not work.
[CustomEditor(typeof(ScriptName))]
public class TestOnInspector : Editor
{
    public override void OnInspectorGUI()
    {
        GUILayout.Label ("This is a Label in a Custom Editor");
    }
}
```

Esimerkkikoodi 2. Unityn C#-ohjelmointikielinen editorilaajennos, jossa OnInspectorGUI-metodiin lisätään tekstirivin piirtävä komento [15].

Muu tapa editorin laajentamiseen on grafiikan piirtäminen editorinäkymään. Editorin laajennokset voivat myös lukea näppäimien painalluksia ja hiiren liikettä. Unity tarjoaa myös oman metodin graafisille apugrafiikoille. Jokaisessa Unityn MonoBehaviour-luokan perivässä luokassa voi toteuttaa OnDrawGizmos-metodin, jossa voi piirtää Unityn editorinäkymään apugrafiikkaa. Metodissa voi piirtää esimerkiksi kentän pisteiden väliin viivoja tai halutun kokoisia ympyröitä tai neliöitä.

Toinen tapa piirtää apugrafiikkaa, on lisätä piirtävä ohjelmakoodi Editor-luokan perivien editorilaajennosten OnSceneGUI-metodiin. Tämä metodi piirtää graafista käyttöliittymää suoraan Unityn tasoeditorinäkymään. Tämän metodin sisällä voi käyttää erityistä Handles-luokkaa piirtämään samoja kolmiulotteisessa avaruudessa sijaitsevia kahvoja, joita myös Unity itsessään käyttää GameObjectien käsittelyyn. Handles-luokkaa voi myös käyttää kaksiulotteisten käyttöliittymien piirtämiseen näytölle.

Unityn editoria voi laajentaa todella monilla tavoilla, ja se näkyy myös Unityn kauppapaikan tarjonnassa. Tarjolla on valtavasti työkaluja melkein kaikkeen mahdolliseen, mitä pelejä tehdessä saattaa tarvita. Tämä tarjoaa osaaville työkaluohjelmoijille mahdollisuuden myydä luomiaan työkaluja. Monia käyttäjiltä hyviä arvioita saaneita työkaluja myydään hyvään hintaan Unityn kauppapaikalla.

## 4 Kenttäeditorin toteutus

Tässä luvussa käydään läpi kenttäeditorin toteutusta. Luku 4.1 kertoo, miten maatasanteiden piirto toteutettiin. Luku 4.2 kertoo, miten pulmaelementtien lisäys kenttää toteutettiin. Luku 4.3 käy läpi muita kenttäeditoria varten tehtyjä laajennoksia ja listaa sitä varten tehtyjä kooditiedostoja.

## 4.1 Maatasanteiden piirto

Maatasanteiden piirto toteutettiin editorilaajennoksena, joka lukee käyttäjän näppäinpainalluksia. Piirtoa varten luotiin ohjelmakooditiedostot PlatformDraw, joka on komponentti, ja PlatformDrawEditor, joka on komponentin editorilaajennos. Kun käyttäjä painaa a-näppäintä, ottaa piirtolaajennos muistiin hiiren senhetkisen pisteen editorinäkyvässä ja piirto alkaa. Esimerkkikoodissa 3 näkyy, kuinka näppäimenpainallus luetaan. Kun käyttäjä painaa s-näppäintä piirron ollessa käynnissä, ottaa piirtolaajennos muistiin loppupisteen hiiren senhetkisestä sijainnista editorissa. Nämä pisteet muodostavat suoralinjan vastakkaiset kulmapisteet. Kun pisteet ovat tiedossa, alkaa maatasanteiden luominen pisteiden väliin. Maatasanteiden piirtolaajennos sijoitettiin pikanäppäimen ctrl+F1 taakse.

```
Event e = Event.current;
if (e.type == EventType.KeyDown && e.keyCode == KeyCode.A)
{
    Ray ray = HandleUtility.GUIPointToWorldRay(e.mousePosition);
    dragStartPoint = ray.origin;
    dragEndPoint = Vector3.zero;
    drawing = true;
}
```

**Esimerkkikoodi 3.** Piirtolaajennoksen ohjelmakoodia OnSceneGUI-metodista, missä tarkastellaan viimeisintä tapahtumaa ja aloitetaan piirto, jos tapahtuma on a-näppäimen painallus.

Ohjelmakooditiedostoon PlatformDraw, joka on MonoBehaviour-luokan perivä komponentti, sijoitettiin viittaukset maatasanneosien valmiselementteihin (engl. prefab). Valmiselementeissä on sekä tasanneosan kuva että sen törmäyspinta. Viittaukset oli ensimmäistä kertaa PlatformDraw-komponenttia luodessa asetettava Unityn puolella käsin osoittamaan oikeisiin valmiselementteihin, mutta sen jälkeen itse PlatformDraw-komponentin sisältävästä GameObjectista pystyi luomaan valmiselementin, jonka ctrl+F1-pikanäppäin luo pelikenttään ja näin mahdollistaa piirtolaajennoksen toiminnan. Muut asiat, jotka sijoitettiin PlatformDraw-luokkaan, olivat käyttäjän senhetkiset maatasannetyyppivalinnat ja referenssi uusimpaan luotuun maatasanteeseen. Nämä asiat sijoitettiin PlatformDraw-luokkaan, koska se ja sen sisällä olevat muuttujien arvot pysyvät muistissa Unityn editorin suorituksen aikana. PlatformDrawEditor-luokasta taas luodaan olio joka kerta, kun PlatformDraw-komponentin sisältävä GameObject valitaan Unityn editorissa. Olio myös tuhotaan heti, kun valinta siirtyy pois sen GameObjectista.

PlatformDrawEditor-luokasta luodun olion muuttujien arvoihin ei voi siis tallentaa mitään, minkä haluaa säilyvän pidempään muistissa.

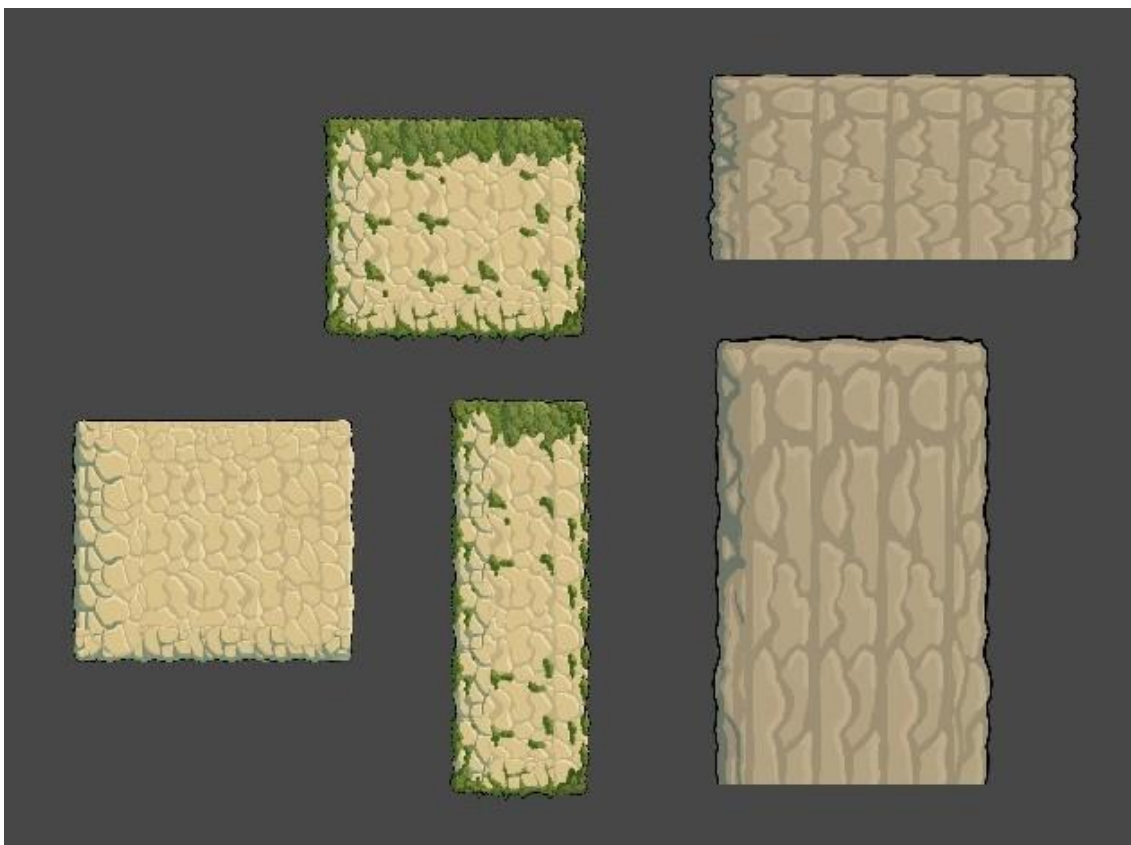
PlatformDrawEditor-luokkaan, joka on PlatformDraw-luokan editorilaajennos, sijoitettiin menet, jotka piirtävät maatasanteet. Piirtotyökalun näppäinten painallusten ja hiiren sijainnin luku tehdään PlatformDrawEditor-luokan OnSceneGUI-metodissa. Esimerkkikoodissa 3 nähdään ohjelmakoodia PlatformDrawEditor-luokan OnSceneGUI-metodista, kuinka piirto aloitetaan. Esimerkkikoodissa 4 nähdään, miten piirto lopetetaan näppäintä painamalla. Tämän jälkeen tarkistetaan käyttäjän valinta uuden ja vanhan maatasannetyypin välillä tarkastelemalla legacyInt-muuttujaa. Jos käyttäjä haluaa piirtää uudentyyppisiä maatasanteita, kutsutaan metodia drawPlatformTiled. Vanhantyyppiset maatasanteet taas piirretään metodilla drawPlatform.

```
if (drawing && e.type == EventType.KeyDown && e.keyCode == KeyCode.S)
{
    Ray ray = HandleUtility.GUIPointToWorldRay(e.mousePosition);
    dragEndPoint = ray.origin;
    if (legacyInt == 0)
        drawPlatformTiled();
    else
        drawPlatform();

    drawing = false;
}
```

**Esimerkkikoodi 4.** Maatasanteen sisältävän suorakulmion piirron lopetus OnSceneGUI-metodissa ja metodikutsut, joilla maatasanne luodaan pelikenttään.

Maatasanteet koostuvat kuvista, jotka kuvaavat tasanteiden päätysivuja ja välisiä. Kuvat on toteutettu niin, että niitä voi asetella peräkkäin ja ne muodostavat katkeamattomalta vaikuttavan kuvan. Vanhemmat tasanteet koostuivat vain oikeasta ja vasemmasta päädyistä ja niiden välissä olevista osista. Tämä kuitenkin loi ongelma, kun tasanteita haluttiin venyttää korkeussuunnassa. Matalat tasanteet näyttivät liiskaantuneilta ja korkeat venyneiltä. Tämän ongelman ratkaisemiseksi artisti loi joka suuntaan aseteltavat tasannekuvat. Kuvassa 14 näkyy vasemmalla kolme vaaleampaa maatasannetta, jotka on toteutettu joka suuntaan aseteltavilla kuvilla. Kaksi kuvan 14 tummemman väristä tasannetta oikealla taas on toteutettu käyttämällä vanhoja, vain sivuttaissuunnassa aseteltavia kuvia. Niissä näkyy selvästi, miten tasanteet ovat pystysuunnassa venyneitä.



Kuva 14. Eri maatasannetyyppejä Unityn editorinäkymässä.

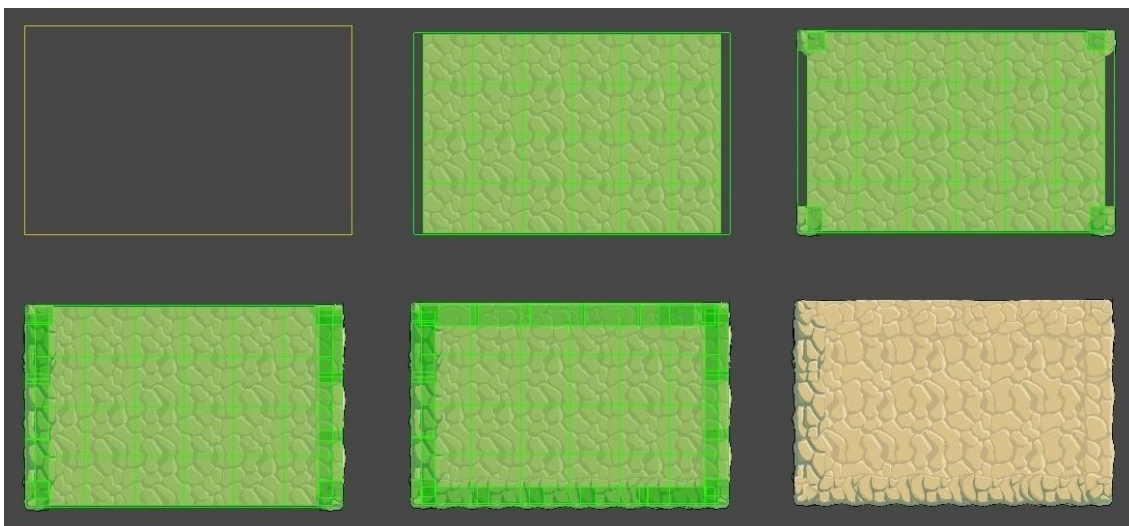
Maatasannekuvien vaihtaminen joka suuntaan aseteltaviin antoi myös tilaisuuden parantaa piirron tarkkuutta. Ensiksi toteutettu sivuttaissuunnassa aseteltavilla paloilla toteutettava tasanteiden luonti asetti päätykuvien kulmat piirrosta saatuihin pisteisiin. Tämä teki tasanteista kuitenkin hieman väärän kokoisia, koska päätykuvissa on aina hieman läpinäkyvää tyhjää tilaa. Piirtoalgoritmia parannettiin, kun uudet kuvat saatiin, ja jatkossa tasanteet piirrettiin käyttämällä osien törmäyspintojen kulmakoordinaatteja täsmälleen oikean kokoisten tasanteiden luomiseksi.

Uusien kuvien sivut, yläpinta ja alapinta toteutettiin niin, että ne voi sijoitella lomittain ilman, että niissä näkyy selviä rajoja. Tämä mahdollistaa hyvinkin vaihtelevien sivupintojen luonnin, kun sivupaloja asetellaan satunnaisia määriä lomittain. Artisti loi myös muutamia kuvia, joita voi asetella oudonnäköisesti lomittuvien kuvien päälle paikkaamaan graafista ilmettä.

Maatasanteen luonnin jälkeen viimeisimmän, eli juuri luodun, tasanteen z-koordinaattia pystyy säätämään tasanteen vasempaan yläkulmaan ilmestyvällä Handles-kahvalla. Tämä säätö määrittää, miten lähellä tai kaukana kamerasta maatasanne on. Koska pelikamera on ortografinen, ei maatasanteiden syvyys vaikuta niiden kokoon pelissä. Syvyydellä säädellään, mikä maatasanne näkyy päällimmäisenä, kun kaksi tai useampia tasanteita on osittain päällekkäin. Jos tätä ei säädettäisi, olisi seurauksena graafinen sekasotku, kun pelimoottori ei osaisi päättää, mikä tasanteiden päällekkäinen graafinen elementti missäkin kohdassa pitäisi piirtää.

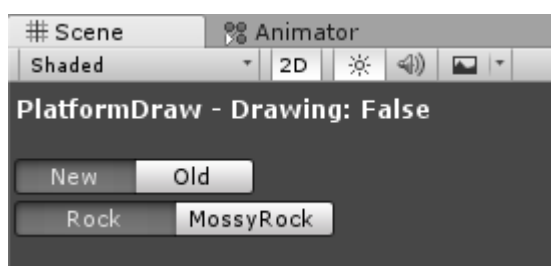
Ohjelmakoodissa uuden tyyppin maatasanne, eli tasanne, joka luodaan asettelemalla tasannepaloja pysty- ja vaakasuunnassa, luodaan drawPlatformTiled-metodissa seuraavilla askeleilla.

1. Lasketaan, kuinka monta neliön muotoista keskipalaa voi asettaa piirrettyyn suorakulmioon pysty- ja vaakasuunnassa. Jos yhtään palaa ei mahdu, keskeytetään piirtäminen varoitusviestiin.
2. Luodaan GameObject, jonka lapsiksi kaikki luotavat tasannepalat asetetaan. Tämä tehdään, jotta pelikentän juuritasolle ei tule valtavaa määrää tasannepaloja ja että koko tasanteen käsittely yhtenä kokonaisuutena olisi helppoa.
3. Luodaan tasanteen keskiosa, ja sijoitetaan luodut maatasanneosat keskelle piirrettyä suorakulmiota. Suorakulmion ylä- ja sivureunoille jää tämän jälkeen jonkin verran tyhjää tilaa, ellei käyttäjä onnistunut piirtämään juuri keskitasanteen monikerran levyistä tai korkuista suorakulmiota. Tämä askel näkyy kuvassa 15 ylhäällä keskellä.
4. Tämän jälkeen luodaan kulmat ja asetetaan ne niin, että kulmien törmäyspisteet ovat täsmälleen samassa paikassa kuin piirretyn suorakulmion kulmat. Kulmien syvyysarvo asetetaan myös sellaiseksi, että ne näkyvät päällimmäisenä maatasanteessa. Maatasanteiden graafiset elementit on suunniteltu niin, että kulmat ovat aina päällimmäisenä. Tämä askel näkyy kuvassa 15 ylhäällä oikealla.
5. Seuraavaksi luodaan sivut. Ne asetellaan niin, että törmäyspintojen sivut ovat täsmälleen samat kuin piirretyssä suorakulmiossa. Sivut luodaan alhaalta ylöspäin, niin että ylemmät sivut menevät alempien päälle. Sivuja luotaessa ne sijoitetaan satunnaisen määrän verran lomittain toistensa päälle, jotta saadaan vaikutelma satunnaisesta kivipinnasta. Sivut ovat myös hieman eri kokoisia kuin keskitiilet, joten niitä täytyy luoda eri määrä. Tämä askel näkyy kuvassa 15 alhaalla vasemmalla.
6. Lopuksi luodaan ylä- ja alapinnat. Ne luodaan samaan tapaan kuin sivut, niin että oikeimmanpuoleinen pala on päällimmäisenä. Pohja kuitenkin sijoitellaan niin, että se on mahdollisten lomittaisten sivuosien alla. Tämä askel näkyy kuvassa 15 alhaalla keskellä. Kuvassa 15 alhaalla oikealla näkyy lopputuloksena luotu maatasanne.



Kuva 15. DrawPlatformTiled-metodin toiminta askel askeleelta.

Piirtolaajennoksen graafinen käyttöliittymä kävi läpi muutaman kehitysaskelen. Kuvassa 13 näkyy, miten valinta kivien tasanteiden ja sammaleisten tasanteiden välillä oli aluksi Unityn tarkasteluelementissä. Lopulta kuitenkin päädyttiin käyttöliittymään, joka sijaitsee Unityn editorin omassa tasonmuokkausnäkyssä (engl. scene view). Näkymän vasemmassa yläkulmassa sijaitseva käyttöliittymä näkyy kuvassa 16. Kuvassa 16 näkyy myös valinta vahojen ja uusien maatasanteiden välillä. Piirron apuna käytetään editorinäkyymään piirrettävää neliötä, josta käyttäjä näkee tulevan tasanteen koon reaaliajassa.

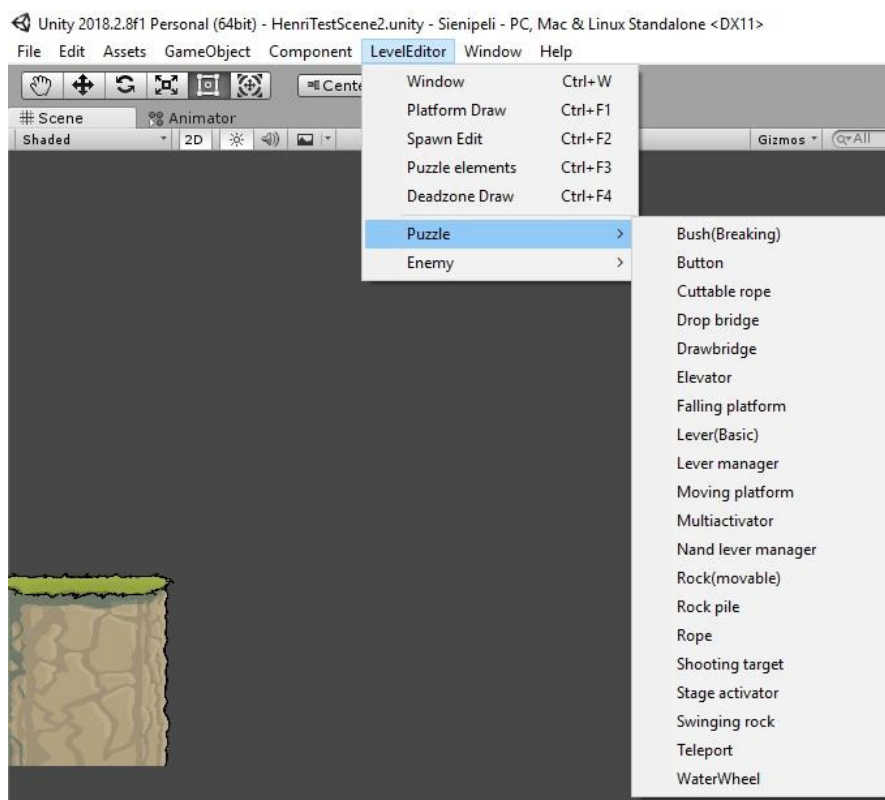


Kuva 16. Tasannepiirtolaajennoksen käyttöliittymä Unityn editorin tasonmuokkausnäkyssä.

Käyttöliittymä sijoitettiin Unityn editorin tasonmuokkausikkunaan, koska näppäinpainalluksia luettaessa tämä ikkunan oli oltava aktiivisena Unityn editorissa. Jos käyttöliittymän napit sijoitettiin tarkastelunäkymän ikkunaan, tuli tarkastelunäkymästä aktiivinen käyttöliittymän nappeja painaessa. Tämä esti uusien tasanteiden piirron niin pitkään, kuin tasonmuokkausikkuna ei ollut aktiivisena.

## 4.2 Pulmaelementtien luominen ja apugrafiikka

Pulmaelementtien luonnin toteutus aloitettiin luomalla valikko, josta pulmaelementtejä voi helposti luoda kenttään. Valikko sijoitettiin ctrl+F3-näppäinyhdistelmän taakse ja kartanmuokkaustyökalun omaan LevelEditor-pudotusvalikkoon, kuten kuvasta 17 näkyy.



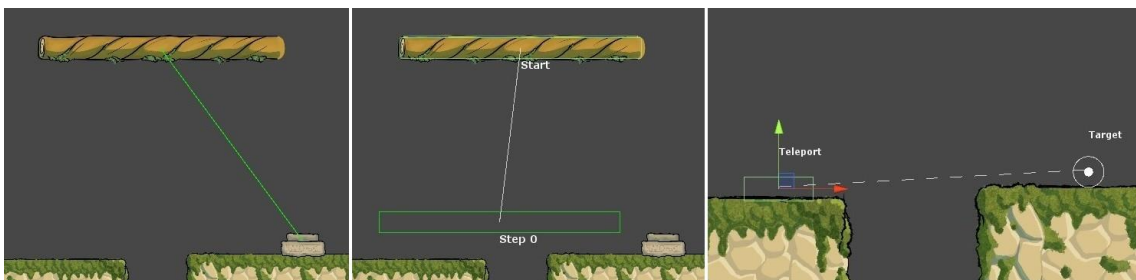
Kuva 17. Pulmaelementtien pudotusvalikko.

Kun pulmaelementti valitaan valikosta, luo laajennos sen keskelle Unityn tasonmuokausnäkömää. Siitä käyttäjä voi sen helposti siirtää hiirellä haluamaansa kohtaan käyttäen Unityn omia siirtotyökaluja. Esimerkkikoodissa 5 näkyy, miten pulmaelementti lisätään pudotusvalikkoon.

```
[MenuItem("LevelEditor/Puzzle/Bush(Breaking)")]
static void BushBreaking()
{
    spawnAsset("BushBreaking");
}
```

Esimerkkikoodi 5. Pulmaelementin luovan komennon lisääminen pudotusvalikkoon. SpawnAsset-metodi lataa pulmaelementin sille annetun nimen perusteella Unityn erityisestä Resources-kansiosta.

Jokaisen pulmaelementin määrittävälle komponentille luotiin myös oma tarkaste-luelementin laajennos. Esimerkiksi Lever-luokalle luotiin sitä laajentava LeverEditor-luokka. Pulmaelementtien omia laajennoksia käytettiin pulmaelementin toiminnan muok-kaamiseen sen ollessa valittuna. Laajennokset esimerkiksi lukivat näppäinpainalluksia, piirsivät apugrafiikkaa näytölle havainnollistamaan pulmaelementin toimintaa ja loivat näytölle kahvoja, joista pulmaelementtien alielementtejä pystyi siirtämään menettämättä pulmaelementin valintaa. Kuvassa 18 vasemmalla näkyy, kuinka nappi on liitetty liikku-vaan tasanteeseen ja sitä havainnollistetaan vihreällä viivalla pulmaelementtien välillä. Kuvan 18 keskiosassa liikkuva tasanne havainnollistaa liikettään piirtämällä katkoviivan liikeradalle ja vihreän neliön paikkaan, johon tasanne liikkuu. Kuvan 18 oikeassa reu-nassa näkyy teleportaatioelementti ja sen kohde. Kohdetta pystyy liikuttamaan hiirellä klikkaamalla valkoisen ympyrän sisällä ja raahaamalla sen haluttuun sijaintiin.



Kuva 18. Pulmaelementtien apugrafiikkaa.

Osan pulmaelementeistä voi aktivoida muilla pulmaelementeillä. Tämä on toteutettu luo-malla peliin IActivate-rajapinta, jossa on metodirungot Activate ja Deactivate. Rajapinta on nähtävissä esimerkkikoodissa 6. Pulmaelementit, jotka voi aktivoida muilla elemen-teillä, toteuttavat tämän rajapinnan.

```
public interface IActivate
{
    void Activate();
    void Deactivate();
}
```

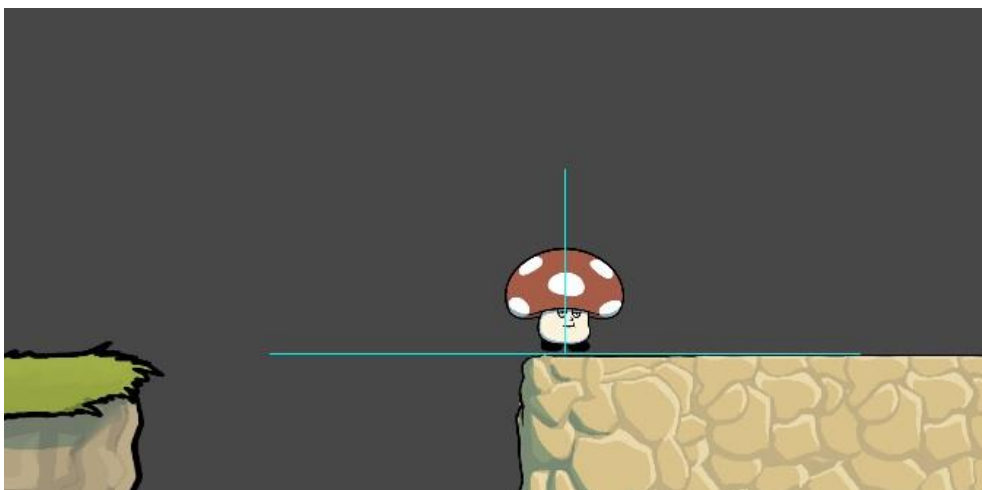
Esimerkkikoodi 6. IActivate-rajapinnan määrittely.

Rajapinnan ansiosta aktivoitavien elementtien lisääminen elementteihin, jotka niitä aktivoivat, on helppoa. Aktivoivassa elementissä, esimerkiksi vivussa, on taulukko IActivate-rajapinnan toteuttavia olioita. Kun vivusta vedetään, kutsuu vivun Lever-komponentti jokaisen taulukossa olevan kohteen Activate-metodia. Jotkin aktivointikomponenttien editorilaajennokset antavat lisätä aktivoitavia asioita taulukkoon siirtämällä hiiri aktivoitavan asian päälle ja painamalla a-näppäintä. Jos elementti löytyy jo aktivointitaulukosta, se poistetaan sieltä. Tämä toteutus tekee eri pulmaelementtien liittämisen toisiinsa helppoksi.

#### 4.3 Apuikkuna ja muut laajennokset

Kenttäeditorille luotiin myös apuikkuna, joka näkyy kuvassa 12. Kenttäeditorin apuikkuna toteutettiin EditorWindow-luokan laajennoksena. Apuikkunasta pystyy säätämään, mitä graafisia apuelementtejä Unityn editoriin piirretään. Graafisten apujen pois kytkeminen osoittautui tärkeäksi ominaisuudeksi, koska pienellä alueella saattaa olla monta pulmaelementtiä, joista jokainen piirtää apugrafiikkaa. Tämä muuttaa näkymän apuviivojen sekamelskaksi, eikä tasosuunnittelija hyödy enää apugrafiikasta mitenkään. Apuikkuna on pikanäppäimen ctrl+w takana.

Apuikkunaan sijoitettiin myös napit, jotka luovat kenttään pelin pelaamisessa tarvittavat oliot, jotka hallinnoivat näppäimien lukua, pelaajien kuolemien käsittelyä ja sitä, kuka pelaaja pelaa milläkin sienellä. Toisesta apuikkunan napista voi kenttään luoda apuruudukon, jota voi käyttää maatasanteita luodessa sen arvioimiseen, voivatko pelaajat hypätä maatasanteelle vai onko se liian korkea. Ruudukosta voi myös arvioida sitä, miten pitkälle pelaajat pystyvät hyppäämään. Hyppypituuksien arviointiin myös itse pelaajahahmoille luotiin apugrafiikka, joka piirtää viivat näyttämään, miten pitkälle ja korkealle pelihahmot voivat hypätä. Pelihahmojen hyppypituuksien apuviivat näkyvät kuvassa 19.



Kuva 19. Pelihahmon hyppypituuksien apuviivat.

Pelihakmojen apuviivojen piirtoa varten ei toteutettu erillistä tarkastelulaajennosta pelihahmon komponentille. Apuviivat piirretään käyttäen komponentin `OnDrawGizmos`-metodia, joka on tarkoitettu juuri tämäntyyppisten graafisten apujen piirtoon. `OnDrawGizmos`-metodi toteutettiin `Mushroom`-luokalle, jonka kaikki neljä eri pelattavaa hahmoa perivät.

Taulukossa 1 mainitaan halutuiksi ominaisuuksiksi myös tallennuspisteiden sijoittelu ja tappavien sijaintien piirto. Tallennuspisteiden sijoittelu toteutettiin luomalla komponentti ja sille tarkastelulaajennos. Tallennuspisteiden muokkaaja lukee käyttäjän näppäinpainalluksia ja lisää hiiren sijaintiin uuden tallennuspisteen, kun s-näppäintä painetaan. Jos sijainnissa on jo tallennuspiste, se poistetaan. Toiminnallisuus on hyvin samankaltainen kuin muissakin näppäimen lukuun perustuvissa laajennoksissa. Kun a-näppäintä painetaan tallennuspistetyökalun ollessa valittuna, siirtää työkalu pelattavat sienet hiiren sijaintiin pelikentällä. Tämä määrittää sen, mistä pelaajat aloittavat pelikentällä.

Tappavien sijaintien piirto toteutettiin samalla tavalla kuin maatasanteiden piirto. Toteutus oli kuitenkin huomattavasti yksinkertaisempi, koska mitään graafisia elementtejä ei liity tappaviin sijainteihin. Piirron aloitus- ja lopetuspisteitä käytettiin luomaan suorakulmio, johon asetettiin törmäyspinta ja tappavan sijainnin aikaansaava `DeadZone`-komponentti.

Kenttäeditoriin toteutettiin myös vihollisten lisäystoiminto. Se toimii samalla tavalla kuin pulmaelementtien lisäystoimintokin. Haluttu vihollinen valitaan pudotusvalikosta, ja laajennos lisää sen keskelle kenttänäkymää. Taulukossa 2 on listaus ja selite kaikille merkittäville kenttäeditoria varten luoduille ohjelmakooditeksteille.

Taulukko 2. Kenttäeditoria varten luotuja ohjelmakooditekstejä.

| Nimi                 | Selite  |
|----------------------|---|
| PlatformDraw         | Komponentti, jota maatasanteiden piirtolaajennos käyttää pohjanaan  |
| PlatformDrawEditor   | PlatformDraw-luokan editorilaajennos, joka vastaa maatasanteiden luonnista  |
| Dead                 | Komponentti, jota tappavien sijaintien piirtolaajennos käyttää pohjanaan  |
| DeadEditor           | Dead-luokan editorilaajennos, joka vastaa maatasanteiden luonnista  |
| Spawn                | Komponentti, jota tallennuspisteiden sijoittelijalaajennos käyttää pohjanaan  |
| SpawnEditor          | Spawn-luokan editorilaajennos, joka vastaa tallennuspisteiden sijoittelusta   |
| MenuShortcuts        | Tiedosto, johon kaikki pudotusvalikkoon lisättävät valinnat kirjoitettiin   |
| UnityLevelEditor     | EditorWindow-laajennos, joka luo työkalun apuikkunan  |
| LevedData            | Singleton-luokka, joka pitää kirjaa ja välittää muuttujien arvoja eri kenttäeditorien osien välillä                                       |
| ButtonEditor         | Button-komponentin, eli pulmaelementtinäpin, editorilaajennos, joka piirtää apugrafiikkaa ja auttaa pulmaelementtien liittämässä          |
| LeverEditor          | Lever-komponentin, eli vivun, editorilaajennos, joka piirtää apugrafiikkaa ja auttaa pulmaelementtien liittämässä                         |
| MovingPlatformEditor | MovingPlatform-komponentin, eli liikkuvan tasanteen, editorilaajennos, joka piirtää apugrafiikkaa ja auttaa reittipisteiden sijoittelussa |
| TeleportEditor       | Teleport-komponentin, eli teleportin, editorilaajennos, joka piirtää apugrafiikkaa ja auttaa kohteen sijoittamisessa                      |
| WaterWheelEditor     | WaterWheel-komponentin, eli vesipyörän, editorilaajennos, joka piirtää apugrafiikkaa ja auttaa pulmaelementtien liittämässä               |

Editorilaajennosluokkia, jotka olivat keskeneräisiä eivätkä tehneet mitään, ei ole listattu taulukkoon 2. Taulukossa 2 mainittu LevedData-luokka toimii singletonina, eli siitä on olemassa vain yksi muistissa pysyvä globaali versio. Se pitää kirjaa apuikkunassa tehdyistä valinnoista, joiden perusteella tiettyjen pulmaelementtien ja maatasanteiden apugrafiikkaa piirretään tai jätetään piirtämättä.

## 5 Kenttäeditorin arviointi

Kenttäeditorista tuli toimiva työkalu pelikenttien luontiin. Sillä pystyy luomaan kenttiä, jotka ovat käytännössä täysin pelattavia alusta loppuun. Varsinkin ehkä keskeisin ominaisuus, maatasanteiden piirto, toimii hyvin ja nopeuttaa pelikenttien luomista jo itsessään huomattavasti. Kenttäeditorilla pystyy luomaan pelikenttään pulmaelementtejä ja yhdistelemään joitakin pulmaelementeistä näppäinpainalluksilla. Kaikille pulmaelementeille ei kuitenkaan ehditty ohjelmoimaan kattavaa tarkastelijalaajennosta. Näiden pulmaelementtien yhdistely jäi Unityn editorissa tehtäväksi käsityöksi.

Kenttäeditorin luonnissa jäi hyvin vähälle huomiolle pelikamera ja sen toiminta. MushQuadin pelikamera toimii niin, että se siirtyy oikealle korkeudelle erityisten kameraliipaisimien avulla. Sivuttaissuunnassa kamera seuraa pelihahmoja. Kameraliipaisimien piirron ja muokkauksen lisääminen kenttäeditoriin on tärkeää kenttien luonnin nopeuttamiseksi lisää. Ne ovat seuraava kenttäeditoriin lisättävä asia.

Tallennuspisteiden asetus toteutettiin luomalla hiiren sijaintiin ennalta määrätyn kokoinen tallennuspiste. Kenttien luomista helpottaisi, jos tallennuspisteet voisi piirtää halutun kokoisiksi. Tästä tosin seuraisi toinen ongelma, koska pelattavat sienet uudelleensyntyvät aina tallennuspisteen keskikohtaan. Korkeissa tallennuspisteissä pelaaja ilmestyisi korkealle ilmaan. Tämän korjaaminen vaatisi tallennuspisteiden käyttäytymisen muuttamista itse pelin puolella.

Muutama laajennos, kuten maatasanteiden piirtotyökalu, toimii pelikenttään lisättävän GameObjectin komponentin laajennoksena. Niitä tarvitaan pelikenttää luotaessa, mutta pelin aikana ne ovat hyödyttömiä ja vievät turhaan muistia. Kenttäeditorin apuikkunaan tulee lisätä nappi, joka poistaa turhat kenttäeditorin objektit, kun pelikenttä on valmis. Toinen, vielä helpompi vaihtoehto ongelman ratkaisuksi olisi asettaa kenttäeditorin tarvitsemille objekteille Unityn EditorOnly-merkki (engl. tag). Tällä merkillä merkityjä objekteja ei oteta mukaan pelin käännöksen.

Kysymys, joka nousi esiin kenttäeditoria ja varsinkin pulmaelementtien laajennoksia luodessa oli se, tarvitaanko joka pulmaelementille omaa tarkastelijalaajennosta. Jos pulmaelementille ei haluta monimutkaisempaa toimintaa, kuten näppäinten lukua tai pulmaelementin aliosien hiirelläliikuttamismahdollisuutta, on laajennoksen luominen kyseiselle

elementille melko turhaa. Mahdolliset graafiset apumerkinnot voi kirjoittaa suoraan pulmaelementin MonoBehaviour-luokan perivän luokan OnDrawGizmos-metodiin.

Pulmaelementtien kanssa työskennellessä tuli esille myös se, miten paljon samankaltaista ohjelmakoodia monen pulmaelementin tarkastelijalaajennokseen tuli. Tältä olisi välttytty, jos pulmaelementit olisi peliä suunniteltaessa toteutettu esimerkiksi luomalla geenierinen Puzzle-luokka, jonka joka pulmaelementti perisi. Tämä olisi johtanut siihen, että suuri osa yleisemmästä laajennoskoodista olisi voitu kirjoittaa Puzzle-luokan laajennokseksi ja vain tietyn pulmaelementin erityisesti vaatima ja muista poikkeava ohjelmakoodi olisi kirjoitettu tämän tietyn pulmaelementin omaan laajennokseen.

## 6 Yhteenveto

Insinööriyössä tutustuttiin pelinkehitykseen, rooleihin siinä ja työkaluihin, joita siinä käytetään. Pelinkehityksessä on monia rooleja, kuten ohjelmoija, artisti ja suunnittelija. Yhdellä henkilöllä voi olla enemmän kuin yksi rooli. Keskeinen rooli insinööriyön näkökulmasta on tasosuunnittelija. Työssä käytiin läpi työkaluja, joita varsinkin tasosuunnittelijat voivat käyttää.

Työssä tutustuttiin taso- ja pulmasuunnittelun menetelmiin. Tasoja suunnitellessa on hyvä aloittaa luonnostelemalla taso. Taso kannattaa jakaa osiin, jotka suunnitellaan erikseen ja lopuksi osista muodostetaan kokonainen taso. Pulmia suunnitellessa tulee kiinnittää huomiota siihen, että pelaaja ymmärtää ratkovansa pulmaa. Jos pelaaja ratkaisee pulman huomaamatta sen olemassaoloa, on seurauksena pettymys. Jos taas pelaaja joutuu yrittämään satunnaisia asioita ratkaistakseen pulman, on pulman suunnittelussa epäonnistuttu. Pelaaja tulee saada ymmärtämään logiikka, joka on pulman ratkaisun edellytys.

Työssä tutustuttiin Unityn laajentamiseen ja todettiin, että Unityn editoria voi laajentaa useilla tavoilla. Siihen voi luoda omia editori-ikkunoita luomalla oman luokan, joka perii EditorWindow-luokan ja Unityn komponenteille voi luoda tarkasteluelementin laajennoksia ja editorinäkömään grafiikkaa ja käyttöliittymän piirtäviä CustomEditor-laajennoksia.

Työssä luotiin kenttäeditori laajentamalla Unityn editoria. Kenttäeditorin luomista käsiteltiin käymällä läpi maatasanteiden piirtolaajennoksen toimintalogiikkaa ja sitä, miten pulmaelementeille toteutettiin apugrafiikka ja niiden yhdistämistä helpottavat toiminnot. Kenttäeditoria varten luodut ohjelmakooditiedostot listattiin toteutuksen käsittelyn lopuksi.

Työssä toteutettua kenttäeditoria arvioitiin ja se todettiin toimivaksi, mutta myös joitakin parannus- ja laajennuskohteita huomattiin. Editoriin voisi lisätä kameraliipaisimien määrittelyn ja tallennuspisteiden muoto tulisi olla vapaasti piirrettävissä. Myös kenttäeditorin käyttämät luokat tulisi siivota pois, kun sillä luotu kenttä on valmis.

Insinööriyön tarkoituksena oli perehtyä Unityn editorin laajennettavuuteen ja kehittää MushQuad-pelille kenttäeditori. Tässä onnistuttiin, ja MushQuad-pelille saatiin luotua toimiva kenttäeditori Unityn editorilaajennoksena. Pelin mukana pelaajille annettavaa kenttäeditoria ei kuitenkaan ehditty tekemään. Unityn editorin laajentamisesta ja sen tarjoamia rajapintoja vasten ohjelmoinnista työn tekeminen opetti paljon. Jatkossa omien työkalujen ohjelmointi muihin projekteihin on paljon helpompaa ja nopeampaa.

Tutkimusvaiheessa luettu teoria pelikenttien suunnittelusta ja pulmien suunnittelusta antoi paljon uusia näkökulmia sisällön luomisesta peleihin. Opittujen uusien suunnittelumenetelmien ja työkaluohjelmoinnin yhdistelmä on tehokas tapa vauhdittaa uuden sisällön luomista nykyisiin ja tuleviin peleihin.

## Lähteet

- 1 Benson, Julian. 2018. Over 7,600 games were released on Steam in 2017, an average of 21 a day. Verkkoaineisto. <<https://www.pcgamesn.com/steam-games-released-2017>>. Luettu 3.10.2018.
- 2 Unity (game engine). 2018. Verkkoaineisto. Wikipedia. <[https://en.wikipedia.org/wiki/Unity\\_\(game\\_engine\)](https://en.wikipedia.org/wiki/Unity_(game_engine))>. Luettu 5.10.2018.
- 3 Compare Plans. 2018. Verkkoaineisto. Unity Technologies. <<https://store.unity.com/compare-plans>>. Luettu 5.10.2018.
- 4 Unity. 2018. Unity Technologies.
- 5 Tools & Middleware for PlayStation 4. 2013. Verkkoaineisto. Sony. <[http://www.tmsstation.scei.co.jp/ps4/index\\_e.html](http://www.tmsstation.scei.co.jp/ps4/index_e.html)>. Luettu 07.11.2018.
- 6 Doom (1993 video game). 2018. Verkkoaineisto. Wikipedia. <[https://en.wikipedia.org/wiki/Doom\\_\(1993\\_video\\_game\)](https://en.wikipedia.org/wiki/Doom_(1993_video_game))>. Luettu 4.10.2018.
- 7 DoomEd. 2018. Verkkoaineisto. Doom Wiki. <<https://doomwiki.org/wiki/DoomEd>>. Luettu 6.10.2018.
- 8 DoomEd running in NeXTSTEP. Verkkoaineisto. Doom Wiki. <<https://doomwiki.org/wiki/File:DoomEd.png>>. Luettu 6.10.2018.
- 9 Tiled Map Editor. Verkkoaineisto. Thorbjørn Lindeijer. <<https://www.mapeditor.org/img/screenshot-terrain.png>>. Luettu 6.10.2018.
- 10 About Tiled. 2018. Verkkoaineisto. Thorbjørn Lindeijer. <<https://doc.mapeditor.org/en/stable/manual/introduction/#about-tiled>>. Luettu 6.10.2018.
- 11 Unity Asset Store. 2018. Unity Technologies.
- 12 Stout, Mike. 2016. A Beginner's Guide to Designing Video Game Levels. Verkkoaineisto. <<https://gamedevelopment.tutsplus.com/tutorials/a-beginners-guide-to-designing-video-game-levels--cms-25662>>. Luettu 6.10.2018.
- 13 Einhorn, Asher. 2015. Four-step puzzle design. Verkkoaineisto. <[https://www.gamasutra.com/blogs/AsherEinhorn/20150528/244577/Fourstep\\_puzzle\\_design.php](https://www.gamasutra.com/blogs/AsherEinhorn/20150528/244577/Fourstep_puzzle_design.php)>. Luettu 6.10.2018.

- 14 Allan, Damien. 2017. Designing Video Game Puzzles. Verkkoaineisto. <[https://www.gamasutra.com/blogs/DamienAllan/20170501/297179/Designing\\_Video\\_Game\\_Puzzles.php](https://www.gamasutra.com/blogs/DamienAllan/20170501/297179/Designing_Video_Game_Puzzles.php)>. Luettu 6.10.2018.
- 15 Editor.OnInspectorGUI. 2018. Verkkoaineisto. Unity Technologies. <<https://docs.unity3d.com/ScriptReference/Editor.OnInspectorGUI.html>>. Luettu 7.10.2018.