PELIEDITORIN LUONTI UNITY 3D:SSÄ

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Mediatekniikka
Tekninen visualisointi
Opinnäytetyö
Kevät 2014
Petteri Paju

Lahden ammattikorkeakoulu
Mediatekniikka

PAJU, PETTERI: Pelieditorin luonti Unity3d:ssä

Mediatekniikan opinnäytetyö, 62 sivua

Kevät 2014

TIIVISTELMÄ

Tämän opinnäytetyön tarkoitus on havainnollistaa, mitä pelieditorit ovat ja näiden tietojen pohjalta kehittää samanlainen alusta Unity 3d:llä. Työssä perehdytään yleisesti pelieditoreihin ja pelieditoreille tärkeisiin toimintoihin. Yleisen kartoituksen jälkeen työssä keskitytään tarkemmin Unity3d:hen ja sen toimintoihin. Unityä läpikäydessä keskitytään alustan eri toimintoihin ja miten niitä voidaan hyödyntää pelieditoreissa. Opinnäytetyössä käydään läpi Unityn tiedonkäsittelytoiminnot sekä käyttöliittymän elementit.

Opinnäytetyön lopussa toteutetaan case, jossa luodaan pelieditori visual novel -peleille. Editorin tarkoituksena on luoda pelinkehitysympäristö, jonka avulla voi luoda visual novel pelejä, ilman pelikoodin kirjoittamista. Case-osuudessa käsitellään, miten Unityn ominaisuuksia käytettiin hyväksi pelieditorin luomisessa ja miten editorin kehittäminen suunniteltiin. Luvussa käydään läpi erilaisia alustaa varten kehitettyjä työkaluja. Luvussa selitetään kunkin työkalun toimintaperiaate ja rooli osana isompaa pelieditoria.

Asiasanat: unity3d, c#, games, video games, xml

Lahti University of Applied Sciences
Degree Programme in Media Technology

PAJU, PETTERI: Developing Game Editor in Unity3d

Bachelor's Thesis in Media Technology, 62 pages

Spring 2014

ABSTRACT

The goal of this thesis was to study what game editors are and to use this information to develop a similar tool for Unity3d. The thesis deals with game editors in generaland functions which are crucial for game editors. After the general analysis, the thesis focuses on Unity3d, its functions and how they can be used in game editors. The thesis reviews Unitys different file handling functions and also different GUI -elements.

The thesis concludes in a case, which is about a creating game editor for visual novel games. The purpose of the game editor is to create a game developing environment, which can be used to develop games, without code writing. The Case chapter illustrates how inbuilt the functions of Unty3d were used to create the editor and how the development was planned. The chapter also reviews different tools that were developed for the platform. The chapter explains the operating principle of each tool and the role as a part of a larger game editor.

Key words: unity3d, c#, games, video games, xml

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön tarkoituksena on havainnollistaa, miten pelieditorin voi luoda Unity3d-alustan sisälle. Pelieditorit ovat hyvin tärkeä työkalu pitkäkestoisessa pelin kehityksessä. Hyvä pelieditori nopeuttaa pelinkehitysprosessia ja helpottaa pelinkehittäjien työntekoa. Monimutkaisen pelin kehittäminen ilman jonkinlaista kehitystyökalua on usein tehotonta ja vaativaa. Pelieditorit antavat käyttäjälle mahdollisuuden muokata pelien sisältöä helposti ilman vaativaa koodaamista. (Wikipedia 2014a.)

Työn alussa määritellään pelieditorin käsite ja esitellään muutama tunnettu pelieditori. Tältä pohjalta kartoitetaan pelieditoreille tyypilliset ominaisuudet ja vaatimukset. Tämän jälkeen työssä keskitytään tarkemmin Unity3d:hen ja kuinka pelieditorin rakentaminen ohjelmiston sisään voidaan toteuttaa. Tämä kattaa datan käsittelyn ja käyttöliittymän luomisen.

Työn käytännön osuudessa luodaan konkreettinen pelieditori visual novel -peleille. Työssä käydään läpi, miten pelieditorin kehitys tapahtuu ja miten pelieditorin kehittämisessä pitää ottaa huomioon.

2 PELIEDITORIT

Pelieditori on ohjelmisto tai alusta, jota käytetään videopelien luomiseen ja muokkaamiseen. Pelieditori on kokoelma työkaluja, joita voi käyttää erityyppisten pelien luomiseen. Hyvä esimerkki pelieditorista on Unity 3D. Pelieditorien tarkoitus on helpottaa pelinkehitystä sekä vähentää tai virtaviivaistaa kehittäjän koodaamisen tarvetta. Pelieditori on ohjelmisto, jossa koodaaminen onnistuu mahdollisimman automaattisesti käyttöliittymän avulla. (Harbour 2011a, 259.)

Pelieditori voi olla yhteen pelityyppiin keskittyvä alusta, kuten RPG-Maker, jota käytetään yksinomaan roolipelien luomiseen. Suosituimman pelieditorit ovat kuitenkin niin sanottuja yleishyödyllisiä editoreja, joilla voi teoriassa luoda minkä tyyppisiä pelejä tahansa. Tällaisia yleishyödyllisiä alustoja ovat esimerkiksi Unity3d ja Unreal Editor. Etuna yksittäisiin genreihin keskittyneillä editoreilla on, että niiden työkalut kattava valmiiksi genrelle tyypilliset toiminnot. Yleishyödylliset editorit puolestaan tarjoavat enemmän vapautta kehittäjille, mutta vaativat enemmän koodaustaitoa, koska kaikkia työkaluja ei ole valmiina. Jos siis roolipelin kehittämiseen valittaisiin Unity, täytyisi iso osa pelin kehitystä helpottavista työkaluista luoda itse. Valitsemalla RPG Makerin nämä työkalut olisivat valmiina, mutta ne eivät välttämättä olisi käyttäjän tarpeiden mukaisia.

Pelieditoreja ei kannata sekoittaa pelimoottoreihin eli game engineihin. Pelimoottorit ovat kuitenkin tavalla tai toisella kaikkien pelieditorien pohjana ja niiden toiminnot kulkevat aina käsi kädessä. Pelimoottori on järjestelmä, joka hoitaa esimerkiksi grafiikan piirtämisen näytölle, äänen toiston, fysiikat ja muut vastaavat taustatyöt. Pelieditoreissa käyttäjät puolestaan määrittelevät millaista grafiikkaa näytölle piirretään, millaista ääntä toistetaan ja miten fysiikoita käytetään (McShaffry & Graham 2012, 17). Pelieditori on siis pelimoottorin ympärille rakennettu visuaalinen käyttöympäristö. Tyypillisesti pelimoottorit tulevat aina editorien mukana samassa paketissa, joko sisäänrakennettuna tai erillisenä ohjelmistona.

2.1 Unity 3D

Unity3d on Unity Technologiesin kehittämä ohjelmointiympäristön ja pelimoottorin kokonaisuus. Unityn pelimoottori on hyvin monipuolinen ja kattaa suure määrän eri ominaisuuksia ja efektejä. Unity kattaa kutakuinkin kaikki pelinkehitykselle tärkeät toiminnot. Alusta on lähtökohtaisesti suunniteltu 3d-pelien tuottamiseen, mutta 4.3 versiosta eteenpäin Unity on alkanut laajentua myös 2d-pelien kehittämiseen.

Unityn suosio on noussut huomattavasti viimeisten vuosien aikana. Ohjelmistoa ovat käyttäneet niin amatöörit kuin ammattilaisetkin. Alusta on kerännyt paljon kehuja ja palkintoja (Unity Technologies 2013a). Unityn helppokäyttöinen ohjelmointiympäristö on antanut kaikille mahdollisuuden ohjelmoida tyylikkäitä pelejä, ilman isoja kustannuksia ja monimutkaisia käyttöympäristöjä.

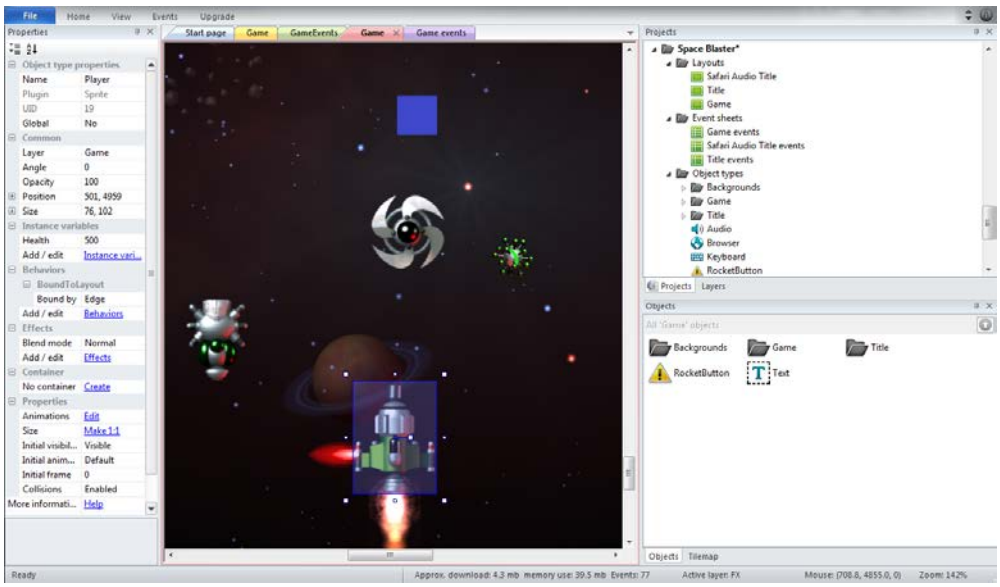
KUVA 1. Unity3d (Ruutukaappaus)

Unity on niin sanottu yleishyödyllinen pelieditori. Unityn sisältää suuren määrän pelinkehitystä helpottavia työkaluja, kuten peliobjektien ominaisuuksien visuaalinen editointi, Windows-tyylinen tiedostonhallinta ja loistavat

julkaisutyökalut. Osa näistä työkaluista näkyy kuvassa 1. Unity on hyvä esimerkki siitä, miksi yleishyödylliset pelieditorit vaativat käyttäjältä enemmän kokemusta. Ilman erikseen ladattavia lisäosia Unityn käyttöliittymä ei ole erikoistunut erityisesti mihinkään peligenreen. Tällaiset työkalut on joko koodattava itse tai ladattava ulkopuolisista lähteistä.

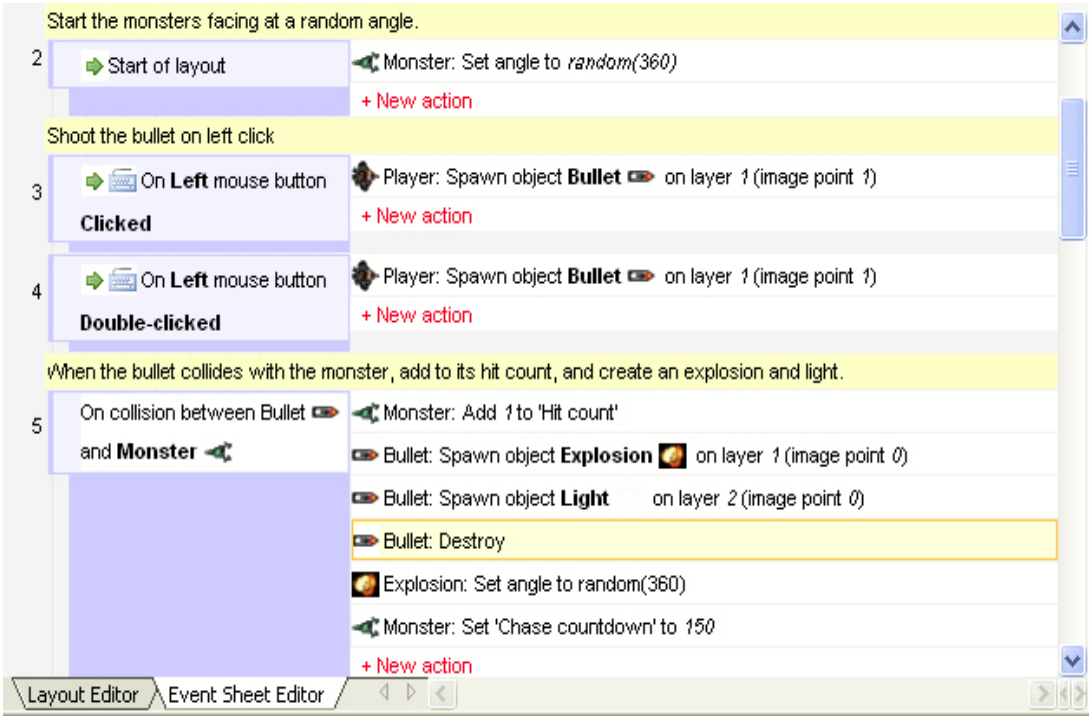
2.2 Construct 2

Construct 2 (kuva 2) on 2d-peleille suunnattu kehitystyökalu HTML5-ympäristöön. HTML5 on HTML-merkistökoodikielen uusin versio, joka on uhannut korvata Adoben Flashin verkon multimediapalvelujen kehitysalustana. HTML5 ja sen taustalla pyörivät Javascript-ja Jquery-koodikielet ovat hyvin laajasti tuettuja. Construct 2:lla voi kehittää pelejä myös mobiililaiteille, työpöytäkoneille sekä Nintendon WiiU-pelikonsolille. (Scirra LTD 2014.)



KUVA 2. Construct 2 (Ruutukaappaus)

Construct 2 on tullut tunnetuksi muun muassa ohjelmiston helpon käytettävyyden ansiosta. Construct 2:n on sisäänrakennettu erinomainen visuaalinen koodaus käyttöliittymä, josta esimerkki kuvassa 3. Visuaalisen koodauksen avulla koodaaminen onnistuu ilman laajaa ymmärrystä ohjelmoinnista. Pelieditorissa on myös sisäänrakennettu fysiikkamoottorin ja WiFi-testaustyökalut, joilla pelejä voi testata mobiililaitteilla langattoman verkon välityksellä. (Scirra LTD 2014.)



KUVA 3. C2 Koodauskäyttöliittymä (Ruutukaappaus)

Construct 2:ta voivat käyttää sekä ammattilaiset että aloittelijat. Helppokäyttöisyytensä ansiosta alustalla on helppo ja nopea kehittää pelejä kokemuksesta riippumatta. Construct 2:sta on saatavilla sekä rajoitettu ilmaisversio että toiminnoillaan laajempi maksullinen versio.

2.3 RPG Maker VX Ace

RPG Maker on ehkä tunnetuin yhteen peligenreen keskittyvä pelieditori. RPG Maker on suunnattu yksinomaan japanilaistyylisten roolipelien kehittämiseen PC-alustalle. RPG Makerillä on pitkä historia. Ohjelmiston ensimmäinen versio julkaistiin jo vuonna 1995. (Wikipedia 2014c.)

KUVA 4. RPG Maker (Enterbrain Inc 2014)

Ohjelmistosarjan viimeisin jäsen RPG Maker VX Ace on malliesimerkki helppokäyttöisestä pelieditorista. RPG Maker tarjoaa käyttäjälle kaiken tarpeellisen roolipelien luomiseen visuaalisessa käyttöliittymässä. RPG Makerin Map-editorissa on helppo luoda kuvan 4 mukainen tiilipohjainen maailma yksinkertaisilla piirtotyökaluilla. (Duggan 2011, 123–214.)

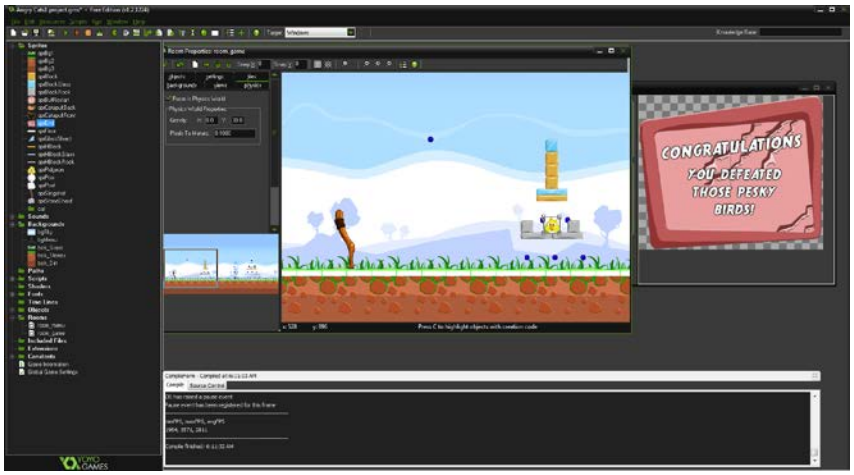
Editoriin on rakennettu valmis tietokanta pelihahmoille, esineille ja vihollisille. Tietokannan muokkaus onnistuu suoraan visuaalisessa käyttöliittymässä. Ilman pelieditoria tällaisen toiminnollisuuden toteuttaminen voi olla hyvinkin aikaa vievää. (Duggan 2011, 177–178.)

RPG Maker on omalla tavallaan myös varoittava esimerkki pelieditorien huonoista puolista. RPG Makerillä kehitetyt pelit muistuttavat usein pelattavuudelta hyvin paljon toisiaan. Tämä johtuu siitä, että RPG Makerin perustyökalut eivät anna kehittäjille tarpeeksi vapauksia vaikuttaa pelimoottorin toimintoihin. Seurauksena on iso määrä pelattavuudeltaan identtisiä pelejä, joiden ainoa erottava tekijä on visuaalinen ilme. (Quijano 2013.)

RPG Makerissä on kuitenkin perinteiset koodaustyökalut, jotka mahdollistavat ohjelman laajentamisen Ruby-koodikielellä. Useille kehittäjille voi kuitenkin olla hyvin vaikeaa siirtyä RPG Makerin selkeästä käyttöliittymästä perinteiseen koodaamiseen. Tämä johtuu erityisesti virallisten teknisten dokumentaatioiden puutteesta.

2.4 GameMaker: Studio

Samoin kuin Construct 2, GameMaker: Studio (kuva 5) on 2d-peleille suunnattu pelieditori. GameMaker on keskittynyt helppokäyttöisyyden, järjestelmällisyyteen ja workflown nopeuttamiseen. Grafiikkamoottori tukee useita shadereitä, jotka muuttavat grafiikan ulkonäköä ilman että kuvatiedostoja muutetaan. Shaderit tarjoavat rajattomat mahdollisuudet erilaisille visuaalisille efekteille. Game Makerissa ei ole sisäänrakennettua visuaalista koodauskäyttöliittymää. Tämän puutteen vuoksi Game Maker on astetta vaikeakäyttöisempi kuin Construct 2. (YoYo Games Ltd 2014.)



KUVA 5. GameMaker Studio (Ruutukaappaus)

GameMaker Studiossa on automatisoidut julkaisutyökalut, joilla sama peli on helppo julkaista useille erityyppisille alustoille. Julkaisutyökalujen ansiosta pelin sisältöä ei tarvitse muuttaa eri alustoja varten. GameMaker Studio tukee kaikkia suosittuja alustoja sekä mobiili-että työpöytä-ympäristöissä. Pelit voi julkaista suoraan pelieditorista verkkokauppoihin, kuten Google Playhin, Apple App Storeen ja Windows Phone Storeen. (YoYo Games Ltd 2014.)

Kaupallisten pelien kehittäjille GameMaker on erinomainen valinta. GameMaker sisältää työkalut sosiaalisen median integroinnille, pelin sisäisiin mainoksiin ja käyttäjien kiinnostusten analysointiin. Isoja kehitystiimejä varten GameMaker tukee projektinhallintatyökaluja kuten, GitHubia. GameMaker on selvästi suunnattu ammattilaisille ja sillä on kehitetty lukuisa myyntihitti pelejä, kuten

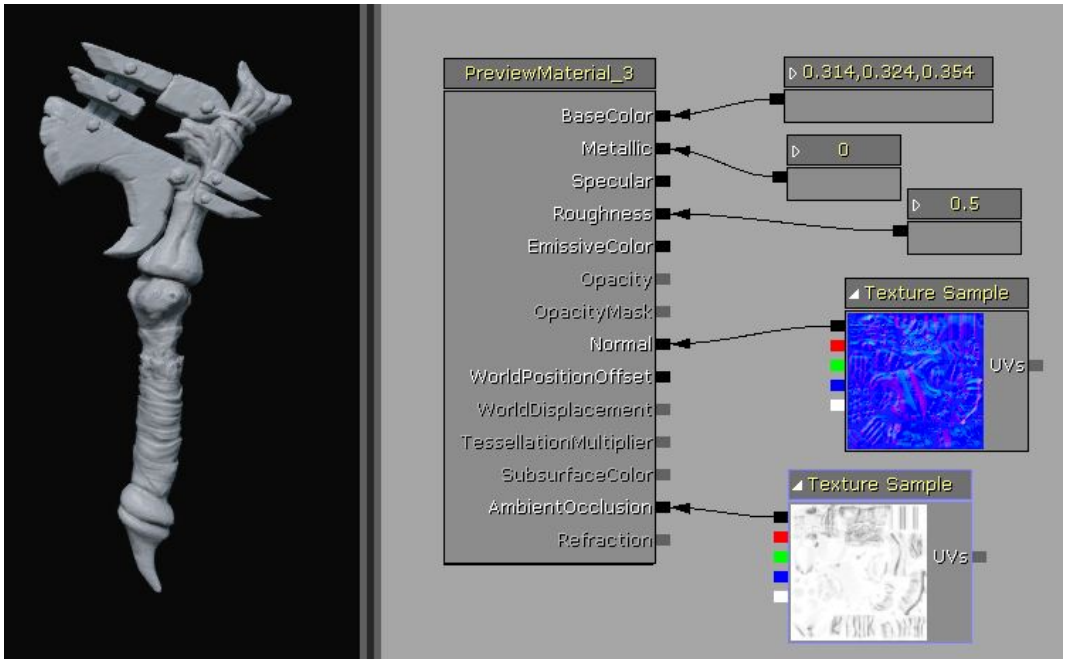
Spelunky, Samurai Gunn ja Game Of The Year -palkinnon voittanut Sacan Ascent. (Elliott 2013, 335–337.)

2.5 Unreal Development Kit

Unreal Develepoment Kit eli UDK on Epic Games -peliyhtiön julkaisema pelinkehitysalusta. UDK on Unity3d:n tapaan 3d-peleille suunnattu pelieditori. UDK oli alun perin suunnattu ensimmäisen persoonan toimintapeleille, ja suurin osa kaupallisista tuotteista on juuri tästä peligenrestä. UDK:lla on kuitenkin mahdollista kehittää myös muiden peligenrejen pelejä, kuten roolipelejä ja hiiviskelypelejä. (Voyles 2013, 1.)

KUVA 6. UDK (Epic Games Inc)

Pelieditorina UDK on hyvin käyttäjäystävällinen. Toisin kuin Unityn käyttöliittymän UDK:n käyttöliittymä on täynnä nappeja ja valikoita (kuva 6). Yleisellä tasolla editorin visuaalinen käyttöliittymä on paljon monipuolisempi kuin Unityn. Esimerkiksi 3d-tekstuurien luonti ja muokkaus -käyttöliittymä on aivan omaa luokkaansa Unityyn verrattuna. Kuten kuvasta 7 voi nähdä, UDK:n tekstuurieditori muistuttaa paljon 3d Studio Maxista tuttua materiaalieditoria. Editorissa erityyppisten materiaalien asettaminen objekteihin onnistuu visuaalisilla linkeillä. (Ahearn 2001, 67.)



KUVA 7. UDK Tekstuuri Editori (Epic Games Inc)

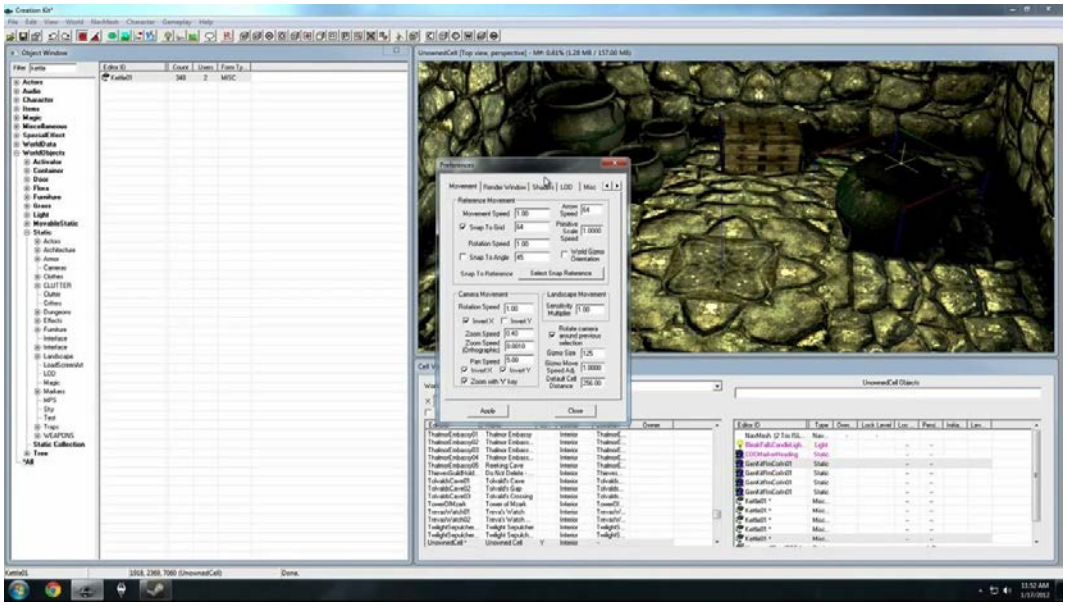
Unreal Editorin taustalla pyörii pelialalla hyvin tunnettu Unreal Engine -pelimoottori (Doran 2013, 1). Pelimoottoria on käytetty lukuisissa myyntihitti-peleissä, kuten Deus Exissä ja Gears of Warissa. Viimeisin julkinen versio Unreal Editorista käyttää Unreal Engine 3 -versiota, mutta on odotettavissa. että lähivuosina editori tulee käyttämään vuonna 2014 julkaistua 4:tä versiota. (Wikipedia 2014c.)

2.6 Skyrim Creation Kit

Skyrim Creation Kit (SCK) on Bethesda Softworks;in julkaisema pelieditori Elder Scrolls IV: Skyrim -videopelille. Toisin kuin muut yllä listatut pelieditorit, SCK:ta ei ole tarkoitettu uusien pelien luontiin, vaan jo olemassa olevan pelin kehittämiseen. SCK:lla kehittäjät voivat muokata lähes kaikkia pelin ominaisuuksia tai luoda kokonaan uusia. SCK:lla pystyy luomaan esimerkiksi uusia tehtäviä, sijainteja sekä esineitä. (Bethesda Softworks LLC 2013.)

Skyrim Creation Kit julkaistiin ilmaiseksi yleisölle 2012. Tämän jälkeen pelille on julkaistu satoja erilaisia modauksia, jotka tuovat peliin ilmaista lisäsisältöä.

SCK:n kaltaiset työkalut ovat muodostuneet perinteeksi Bethesdan kehittämille peleille ja hyvästä syystä. Modit parantavat peliä ja tuovat siihen uutta sisältöä, ilman lisäkustannuksia Bethesdan puolelta.



KUVA 8. Skyrim Creation Kit (IGN 2013)

Pelieditorina SCK sisältää laajat työkalut erilaisten modien kehittämiseen. Tehtävien luomiseen, teksturointiin ja kenttäsuunnitteluun on kehitetty omat työkalut. SCK:n käyttöliittymä ei kuitenkaan ole hyvin käyttäjäystävällinen. Suurin osa työkaluista rakentuu pitkistä listoita ja harmaista valikoista (kuva 8). Tekstuurien esikatselu ja objektien siirtely 3d-näkymässä ovat niitä harvoja visuaalisia elementtejä, joita SCK tarjoaa. (Bethesda Softworks LLC 2013.)

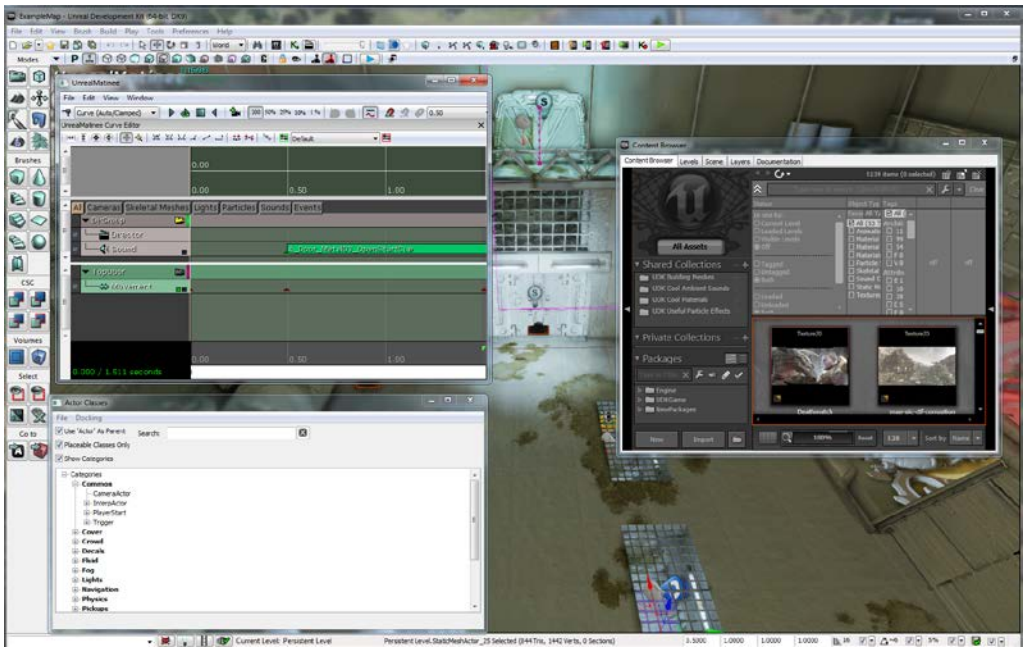
3 PELIEDITORIEN TOIMINNOT

Pelieditoreita on monenlaisia. Jotkin editorin on suunnattu puhtaasti 3d-peleille, jotkin taas esimerkiksi 2d-toimintapeleille. Jotkin editorit voivat olla hyvin vaikeakäyttöisiä, kun jotkut taas ovat yksinkertaisia käyttää sekä aloittelijoille että ammattilaisille. Eroista huolimatta, lyhyessäkin vertailussa alustojen välillä löytyy tiettyjä yhtenevyyksiä ja perustoimintoja, jotka määrittävät pelieditorien perustan.

3.1 Käyttöliittymä

Pelieditorien ideana on yksinkertaistaa pelikehittäjien työskentelyä. On siis tärkeää, että käyttöliittymän käytettävyys on kohdillaan. Käyttöliittymään kuuluvat kaikki napit, ikkunat ja valikot, joita editorien käyttäjät käyttävät pelin luomiseen. Käyttöliittymäsuunnittelu on hyvin tärkeä osa pelieditorin kehitystä, koska sekava käyttöliittymä johtaa suoraan heikompaan työtehokkuuteen. (Wihlida 2006, 68.)

Ideaalisti koko pelinkehitysprosessi, koodausta lukuun ottamatta, voidaan hoitaa suoraan pelieditorin käyttöliittymässä. Käyttöliittymässä kehittäjä pystyy näkemään tarkan esikatselun siitä, miltä peli tulee näyttämään. Esikatseluikkunan kautta voidaan myös valita ja siirrellä peliobjekteja hiirellä. Esikatseluikkunassa käyttäjä voi esimerkiksi nähdä, onko pelin valotus kohdillaan ja miltä peli näyttää pelaajan näkökulmasta. Tällä tavoin pelikehittäjä saa paremman yleiskuvan pelistä. (Pierce 2012, 28.)



KUVA 9. UDK käyttölliittymä (Ruutukaappaus)

Käyttöliittymää kehittäessä pitää ottaa huomioon tarvittavien toimintojen määrä ja niiden käyttötarkoitus. Tärkeiden työkalujen pitäisi olla näkyvissä aina kun mahdollista. Vähemmän tärkeät työkalut on hyvä sijoittaa välilehdille, työkalupalkkeihin tai suljettaviin ikkunoihin, jotta ne ovat pois tieltä, kun niitä ei tarvita. (Chu 2013, 35.)

Suljettavat ikkunat ja erilaiset välilehdet tarkoittavat muokattavaa käyttöliittymää. Pelinkehityksessä on useita eri vaiheita, jotka kaikki vaativat hyvin erilaisia työkaluja. Antamalla käyttäjälle vapaus muokata käyttöliittymän työkalut haluamaansa järjestykseen parannetaan huomattavasti editorin käyttökokemusta. Sama periaate pätee kaikkiin tietokoneohjelmistoihin. (Chu 2013, 37.)

3.2 Grafiikka

Tekstiseikkailuja lukuun ottamatta, jokainen peli on audiovisuaalinen kokonaisuus. Pelieditorista täytyy siis löytyä toiminnot sekä visuaalisen sisällön että äänisisällön lisäämiseen ja hallintaan. Visuaalisella sisällöllä tarkoitetaan esimerkiksi 3d-malleja, tekstuureja, 2d-kuvia ja joitain käyttöliittymän elementtejä. Äänisisällöksi voidaan laskea erilaiset ääniefektit ja taustamusiikit. (Menard 2011, 111.)

On hyvin harvinaista, että grafiikkaa luodaan suoraan pelieditorissa. Yleensä grafiikka luodaan jossain ulkoisessa ohjelmassa, kuten 3d Studio Maxissa tai Adobe Photoshopissa. Jotta 3d-mallit saataisiin sisään pelieditoriin, täytyy ohjelmiston luonnollisesti kyetä lukemaan 3d-tiedostomuotoja. Tyypillisesti pelieditoreihin ohjelmoidaan käännöstyökalut, jotka muuntavat 3d-tiedostot automaattisesti pelimoottorille sopivaan muotoon. Vaihtoehtoisesti pelimoottori voidaan rakentaa niin että se tukee suoraan näitä tiedostomuotoja. (Harbour 2011b, 107.)

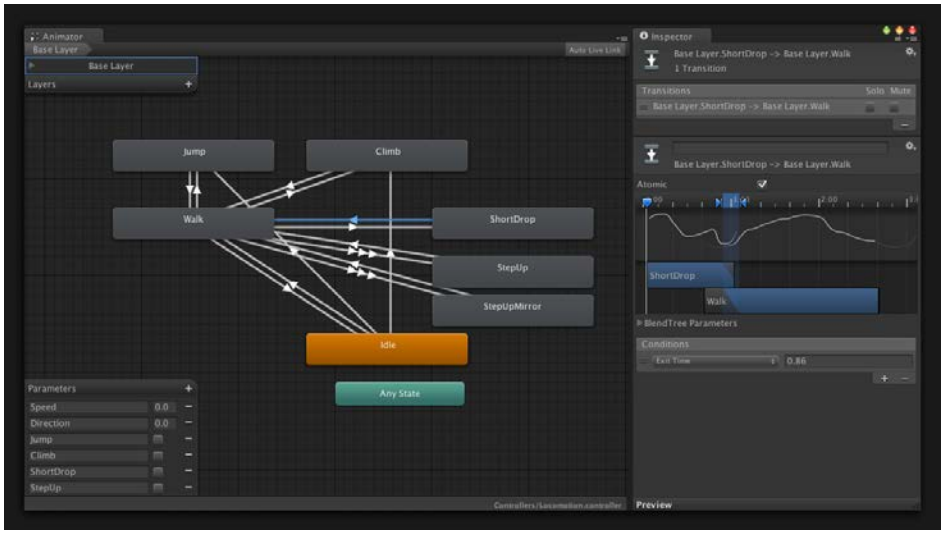
Oli sisääntuontimetodi mikä tahansa, grafiikka ei välttämättä ole aina käyttövalmis heti sisään tuonnin jälkeen. Esimerkiksi 2d-tekstuurien asettelu paikalleen 3d-objekteihin ja näiden tekstuurien asetusten hienosäätö hoidetaan usein myös pelieditorin puolella. Nykyään on kuitenkin hyvin yleistä, että tekstuurit tuodaan sisään samassa tiedostossa 3d-mallin kanssa. (Blackman 2011, 247.)

3.3 Animointityökalut

Animaatiot tuovat peleihin eloa ja liikettä. Animaatiolla voidaan tarkoittaa yksinkertaista liikettä oikealta vasemmalle tai monimutkaista 3d-hahmon juoksuanimaatiota. Yksinkertainen liike oikealta vasemmalla on mahdollista toteuttaa myös koodaamalla, ilman erityisiä työkaluja. Peleissä monimutkaiset juoksu- ja raaja-animaatiot luodaan aina etukäteen ja toistetaan pelin aikana. Animaatiotyökaluja voidaan käyttää näiden animaatioiden luomiseen, muokkaukseen ja hallintaan. Animaatiotyökalun laajuus vaihtelee hyvin paljon eri editorien välillä. (Menard 2011, 234.)

Sekä 2d- että 3d-animoinnissa käytetään paljon samoja työkaluja. Yleisimmät perusanimaatio-tyypit ovat siirto-, skaalaus- ja rotaatioanimaatiot. Nämä perustyökalut ovat yhteisiä kaikille animaatiosovelluksille. Tämän takia kaikki perusanimaatiotyökalut näyttävät jokseenkin samanlaisilta. Pelieditoreissa luodut animaatiot ovat varmaakin poikkeuksetta keyframe-pohjaisia animaatioita, eli objektit animoituvat automaattisesti kahden tai useamman arvon välillä (McShaffry 2009, 201.)

Tämänkaltaiset työkalut ovat hyvin yleisiä kaiken tyyppisissä multimedia-ohjelmissa. Yksinkertaiset animaatiot eivät kuitenkaan välttämättä riitä videopelien tarpeisiin. Varsinkin 3d-pelit ovat täynnä erilaisia animaatioita. Yksittäinen pelihahmo voi sisältää jopa kymmeniä erilaisia animaatioita, joita toistetaan pelitilanteen mukaan. Pelitilanteen nopeat muutokset vaikeuttavat animaatioiden toteuttamista. Esimerkiksi jos vihollispelaaja kaatuu maahan kesken juoksuanimaation, on juoksuanimaatio keskeytettävä ja kaatumisanimaation toistaminen aloitettava. Äkillinen juoksuanimaation katkaiseminen näyttää kuitenkin luonnottomalta. Varsinkin 3d-pelieditorit tarvitsevat kehittyneet animaatiotyökalut, joilla voi dynaamisesti sekoittaa animaatioita toisiinsa. (Pardew 2006, 481.)



KUVA 10. Unity Animaatio-työkalu (Unity Technologies 2013)

Sekoitus-, eli blendaustyökalut löytyvät esimerkiksi Unity3d:stä ja UDK Editorista. Unityssä animaatioiden blendausta kontrolloidaan kuvan 10 mukaisella työkalulla. Toistettavat animaatiot riippuvat pelihahmon sen hetkisestä tilanteesta. Esimerkiksi jos pelikoodi ilmoittaa pelimoottorille että hahmo juoksee, pelimoottori toistaa juoksuanimaatiota. Kun pelihahmo pysähtyy paikalleen, ilmoitetaan tästä tieto pelimoottorille. Pelimoottori häivyttää hiljakseen pois juoksuanimaation ja tuo samalla esiin pysähtymisanimaation. Unityn blendaus -työkaluilla saadaan siis pelaajan hahmojen animaatiot vastaamaan dynaamisesti pelitilanteeseen. (Blackman 2011, 593.)

Usein animaatiot luodaan ulkoisessa 3d-ohjelmassa, mutta erityisen kehittyneet pelieditorit tarjoavat nämä työkalut myös editorin sisällä (Blackman 2011, 223). 3d-hahmojen animoinnissa käytetään yleensä niin sanottua luustoa. Luusto-työkalulla luodaan hahmolle ihmisen luustoa muistuttava rakenne, jonka avulla hahmon raajoja voidaan animoida. Luusto voidaan joko luoda pelieditorissa tai sitten tuoda 3d-tiedoston mukana 3d-mallinnusohjelmasta. (Blackman 2011, 581.)

2d-peleissä animaatioeditorit ovat varsin yksinkertaisia. 2d-peleissä hahmojen animaatiot toteutetaan spritejen avulla. Spritet ovat 2d-peleissä käytettyjä kuvia, jotka sisältävät useampia pieniä kuvia. Pienet kuvat yhdessä muodostavat animaation. Animaatiot tuodaan siis editoriin sisään kuvina. Yksittäisten sprite-kuvien muokkaamista varten ei ole järkevää kehittää monimutkaista muokkaustyökaluja pelieditorin sisälle. (Elliott 2013, 12.)

KUVA 11. Sprite Sheet (Roberts J. 2010)

Koska sprite-animaatiot ovat vain kuvia, ei pelieditorin puolella tarvita monimutkaisia animaatiotyökaluja. Esimerkiksi Construct 2 ei sisällä varsinaista animaatioeditoria, vaan tukeutuu yksinomaan sprite-animaatioihin ja koodauspohjaisiin animaatioihin. Tässä tapauksessa editoria kehitettäessä huomio täytyy keskittää sprite-animaatioiden sujuvaan sisääntuontiin. Hyvän Sprite-työkalun täytyy kyetä jakamaan sprite-kuvista yksittäiset animaatioframet. Kuva

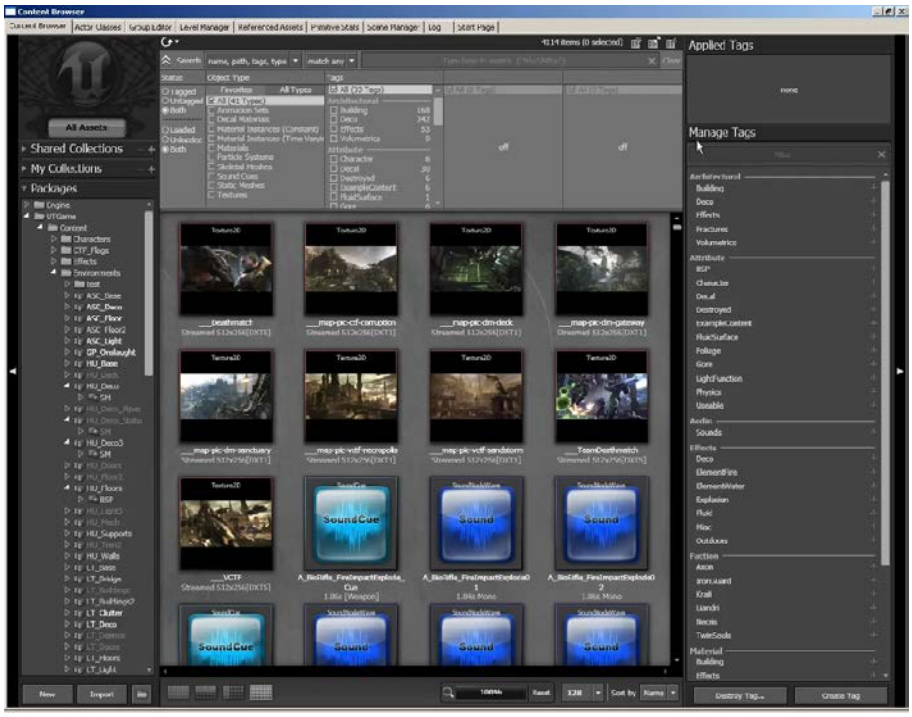
11 sisältää 22 eri animaatioframea, ohjelman täytyy siis tarjota työkalut kuvan hajottamiseksi pienempiin osiin. Tällä ei varsinaisesti tarkoiteta kuvan jakamista erillisiin kuvatiedostoihin, vaan kykyä lukea yksittäisiä osia isommasta kuvatiedostosta. (Harbour & Smith, 217.)

GameMaker ja Construct 2 eivät kuitenkaan sisällä automaattista sprite-kuvien jakajaa. Käyttäjien käyttäjän täytyy itse kertoa ohjelmalle tarkat tiedot siitä, kuinka monta kuvaa kuvatiedostossa on ja kuinka monta kuvaa yhdellä rivillä on. Unityn Sprite Editor kykenee kuitenkin tunnistamaan automaattisesti kuvien määrän ja jakamaan ne osiin. Vaihtoehtoisesti kuvat voi tietysti tuoda sisään yksittäisinä kuvatiedostoina, jos kuvien valtava määrä ei häiritse. (Unity Technologies 2013b.)

3.4 Tiedostohallinta

Pelit rakentuvat yleensä jopa sadoista erityyppisistä tiedostoista, kuten 3d-malleista, bittikartoista ja tekstitiedostoista. Pelieditorien tiedostohallinnassa ei kannata tukeutua pelkästään Windowsin tai Macin tiedostonhallintajärjestelmään. Tähän on useita syitä. Ilman omaa tiedostonhallinta-käyttöliittymää pakotetaan käyttäjä jatkuvasti vaihtelemaan ikkunoita Windowsin ja pelieditorin välillä. (Smith & Queiroz 2013, 1.)

Pelit ovat rakenteeltaan hyvin hajanaisia. Peliobjektit rakentuvat useista eri tiedostoista tavalla tai toisella. Esimerkiksi animoitu 3d-malli voi koostua animaatio- ja 3d-tiedostoista sekä useista tekstuuritiedostoista. Tiedostojen linkit tukeutuvat usein pelkästään niiden sijaintiin pelikansiossa. Tiedostojen huolimaton siirtäminen aiheuttaa usein ongelmia, kuten kadonneita tekstuureja ja rikkinäisiä malleja. Hyvin koodatun tiedostonhallinnan kautta näitä tiedostoja voidaan siirtää paljon hallitummin, hajottamatta tiedostorakennetta. (Menard 2011, 17.)



KUVA 12. UDK Browser (Ruutukaappaus)

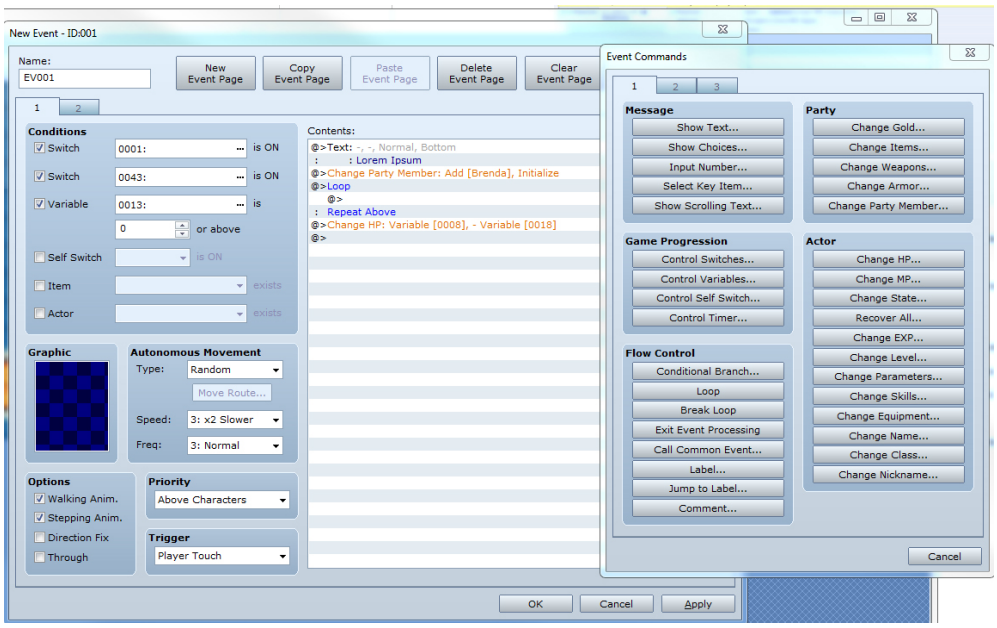
Pelieditorien kansiorakenteet sisältävät aina useita alakansioita. Pelieditorin oma tiedostonhallinta voi esittää peliprojektin tiedostot paljon havainnollisemmalla tavalla kuin Windows. Nopeat hakutoiminnot, grafiikoiden esikatselut (kuva 12), kategorisointi ja sisääntuontiasetukset ovat tyypillisiä pelieditorien tiedostonhallintaominaisuuksia. (Menard 2011, 17.)

3.5 Ohjelmointityökalut

Editoreja luodessa kehittäjien on muistettava, että editori on kuitenkin vain työkalu eikä se saa olla esteenä pelinkehitykselle. On hyvin vaikea, ellei mahdotonta, suunnitella alustaa, joka sopii kaikkien kehittäjien tarpeisiin. Peli editoreissa täytyy aina löytyä mahdollisuus laajentaa peliä ilman editorin asettamia rajoituksia. Vaikka editorien tarkoitus on minimoida käyttäjän tarve koodata, se ei kuitenkaan tarkoita, että tarve koodata poistettaisiin kokonaan. Oman koodin luonti on lähes välttämättömyys pelieditoreille. Mahdollistamalla koodaaminen annetaan käyttäjille suurempia vapauksia pelin luomiseen. (Guldbrandsen & Storstein 2010 , 86.)

Pelieditorit voivat kuitenkin pyrkiä virtaviivaistamaan koodin kirjoittamista. Esimerkiksi Construct 2 ja RPG Maker sisältävät semivisuaalisen koodauskäyttöliittymän. Visuaalisen koodauksen avulla koodia on helppo luoda ja muuttaa ilman, että se edes tuntuu koodaamiselta. (Voyles 2013, 130.)

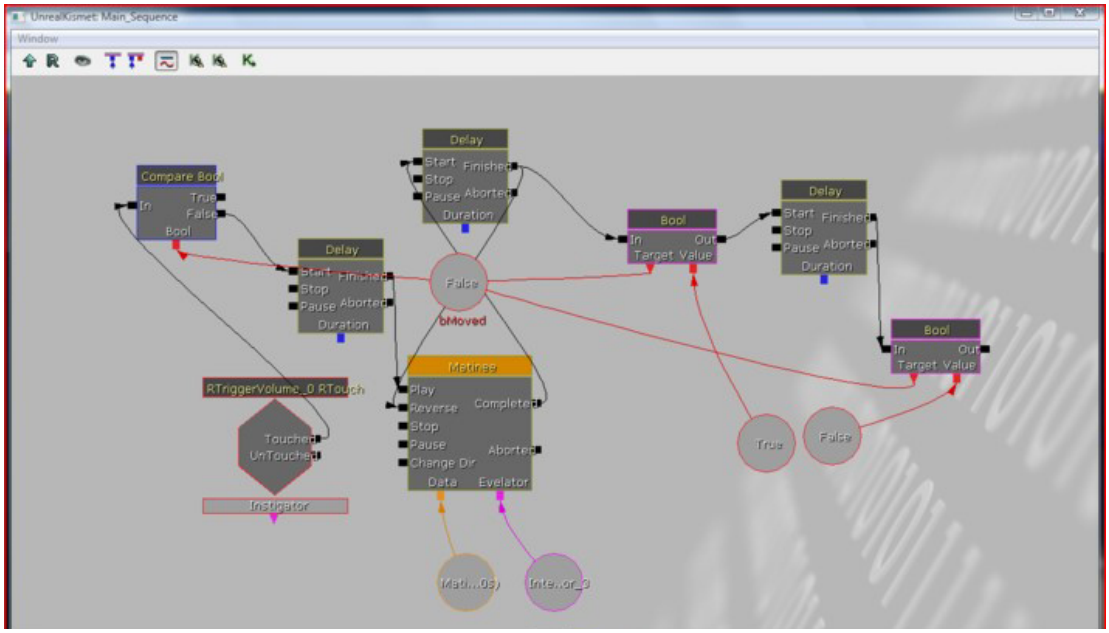
Tärkein ero visuaalisten koodauskäyttöliittymien ja perinteisen tekstikoodauksen välillä on helppokäyttöisyys ja monipuolisuus. Hyvin tehty visuaalinen koodaus on ikään kuin aputyökalu normaalille koodaukselle. Perinteinen koodaus vaatisi käyttäjältä koodin kirjoittamista omin käsin. Visuaalinen käyttöliittymä korvaa koodaamisen napeilla ja valikoilla, jotka luovat koodin automaattisesti ennalta määritetyllä tavalla. RPG Makerin käyttöliittymä (kuva 13) onnistuu tässä erinomaisesti. (Guldbrandsen & Storstein 2010, 130.)



KUVA 13. RPG Maker Koodaus (Ruutukaappaus)

Toinen hyvä esimerkki visuaalisesta koodaustyökalusta on Unreal Editorin Kismet-työkalu (Kuva 14). Toisin kuin RPG Makerin koodauskäyttöliittymä, Kismet on lähempänä konkreettista visuaalista koodausta. Kismet-työkalu perustuu ajatuskarttamaiseen rakenteeseen, jossa käyttäjä voi rakentaa yksinkertaisia ehtolauseita ja käynnistää funktioita esimerkiksi pelin alkaessa. Kismet on kuitenkin tarkoitettu vain suhteellisen yksinkertaisten toimintojen

koodaamiseen. Monipuolisempaan koodaamiseen Unreal Editor tarjoaa c++- ja Java-koodauskieliä muistuttavan UnrealScriptin. (Doran 2013,10.)



KUVA 14. UDK Kismet (Wikipedia 2014)

Visuaalinen scriptaus ei kuitenkaan ole aina hyvä asia. Käyttämällä pelkästään visuaalista käyttöliittymää pelinkehittäjä rajoittaa itsensä vain käyttöliittymän tarjoamiin mahdollisuuksiin. Jos käyttöliittymä ei täytä pelinkehittäjän vaatimuksia eikä muuta koodausmahdollisuutta ole, pelinkehittäminen voi jämähtää paikalleen. Parhaallakin visuaalisella editorilla on rajansa. Tämän takia pelieditorin täytyy aina mahdollistaa myös perinteinen manuaalinen koodaaminen. Unity3d alusta esimerkiksi tarjoaa mahdollisuuden lisätä projektiin omia C#- tai javascript-scritptejä. Unityn scripteillä on mahdollista vaikuttaa myös Unityn käyttöliittymään sekä fysiikkamoottoriin. Koodit kirjoitetaan kuitenkin ulkoisissa sovelluksissa. (Tracy & Reindell 2012, 354.)

Visuaalinen koodaus-käyttöliittymä on paljon helpompi luoda yhteen genreen suuntautuneisiin editoreihin. Yksittäiset peligenret ovat jo vuosia vanhoja, joten niiden perustoiminnot on helppo kartoittaa ja automatisoida pelieditoriin. Tämä ei kuitenkaan ole mahdollista Unity 3D:n kaltaisissa editoreissa. Kaikkien peligenrejen kaikkien mahdollisten toimintojen ahtaminen yhteen alustaan, johtaisi todennäköisesti sekavaan lopputulokseen.

3.6 Debug-työkalut

Debuggauksella tarkoitetaan ohjelmistossa tapahtuvien virheiden tunnistusta ja korjausta. Pelit ovat kehitysvaiheessa usein hyvin epävakaita ja täynnä ohjelmointivirheitä. Ohjelmointivirheet estävät usein pelin toimimisen kokonaan, ja ongelmakohtien kartoitus ilman työkaluja on lähes mahdotonta. (Elliott 2013, 41.)

Koska pelin virheet liittyvä teoriassa aina editorin pelimoottoriin, pelimoottoriin tulisi aina koodata virheentarkistustoimintoja niin paljon kuin mahdollista (Matloff & Salzman 2008, 3). Virheentarkistuksessa pelimoottori ilmoittaa pelieditorille, missä kohdassa koodia ohjelmointivirhe on ja mahdollisesti sen, miten sen voi korjata.

Pelieditorin puolella debuggaus-työkalut sisältävät aina virhekonsolin, joka viestii käyttäjälle pelimoottorin lähettämät virheviestit. Debuggaus helpottaa käyttäjän työurakkaa, vähentämällä virheen etsimistä silmämääräisesti. Yksinkertaisessa pelieditorissa virhekonsoli voi olla jo tarpeeksi, mutta monimutkaisimmissa projekteissa monipuolisemmat työkalut tulevat tarpeeseen. Erityisesti 3d-pelit vaativat kehittyneemmät virheentarkistustyökalut, joilla voidaan seurata koodissa tapahtuvia muutoksia pelin ollessa käynnissä. (Pierce 2012, 232.)

3.7 Ohjeistus ja dokumentaatio

Pelieditorit ovat usein monimutkaisia ohjelmistoja, täynnä erilaisia työkaluja ja valikoita. Jotta aloittelijankin olisi helppo käyttää editoria, on ohjelmiston käyttöön luotava kirjallinen ohjeistus. Ohjeen tulee kertoa mahdollisimman kattavasti miten alustaa ja sen työkaluja käytetään.

Ohjeistuksia voi olla monenlaisia. Yksi vaihtoehto on luoda kaksi eri ohjetta: aloittelijaohje ja edistynytohje. Aloittelijaohjeessa selitetään ohjelman toiminnot yleisesti. Aloittelijaohjeen tarkoitus on auttaa uutta käyttäjää ohjelman oppimisessa, menemättä liian syvälle. Edistynytohje käy läpi loput monimutkaisemmat toiminnot ja toimintamenetelmät. Yhdessä ohjeet antavat uudelle käyttäjälle mahdollisimman monipuolisen tietotaidon ohjelman

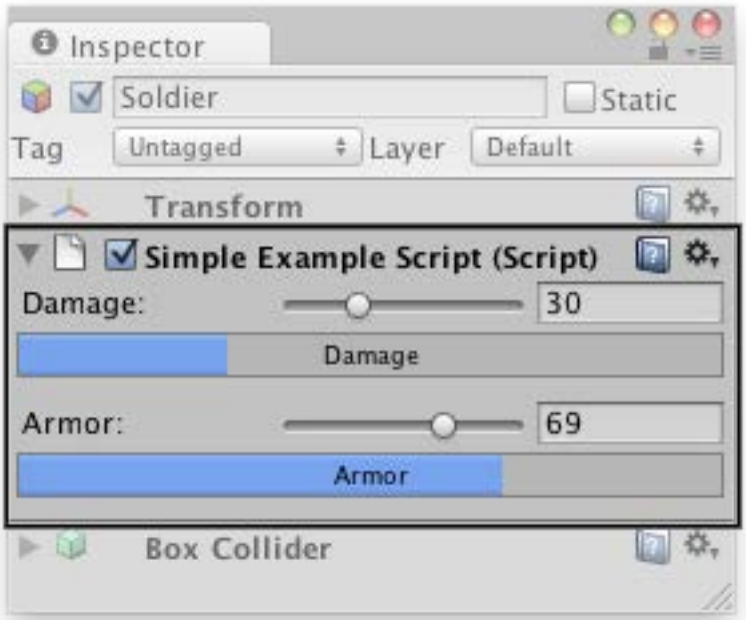
käyttämiseen. Ohjeistuksen tulee olla aina kirjallisessa muodossa, joko internetissä tai painettuna. (Watters 2010.)

Lisäksi internet-aika on tuonut mukanaan erilaiset videotutoriaalit, jotka ovat usein paljon havainnollisempia kuin painetut ohjeet. Videotutoriaalit ovat kuitenkin hyvin pitkiä, eivätkä ne sovellu nopeaan tiedonhakuun yhtä hyvin kuin kirjalliset vaihtoehdot.

Toinen tärkeä ohjeistuksen muoto on tekninen dokumentaatio. Tekniset dokumentaatiot sukeltavat vielä syvemmälle ohjelmiston toimintoihin. Teknisestä dokumentaatiosta löytyy tietoa pelimoottorista, koodaamisesta ja pelieditorin rakenteista. Toisin kuin muut ohjeet, tekniset dokumentaatiot eivät sisällä suurta määrää tekstiä, vaan ne ovat enemmänkin lista työkaluista ja niiden ominaisuuksista. Tekninen dokumentaatio voi esimerkiksi kertoa, mitä erilaisia muuttujia pelimoottori käyttää. Tekniset dokumentaatiot ovat tärkeitä erityisesti koodaajille, jotka luovat uutta koodia pelimoottorin päälle. (Menard 2011, 190.)

4 KÄYTTÖLIITTYMÄN LUOMINEN UNITYYN

Unityä ei ole erityisesti suunnattu millekään peligenrelle. Ohjelmisto sisältää ison määrän universaalisti hyödyllisiä työkaluja, joita voi käyttää kaikentyyppisissä peleissä. Tämä on johtanut siihen, että työkalut eivät erityisesti täytä minkään peligenren vaatimuksia, vaan kehittäjien täytyy luoda nämä työkalut itse.

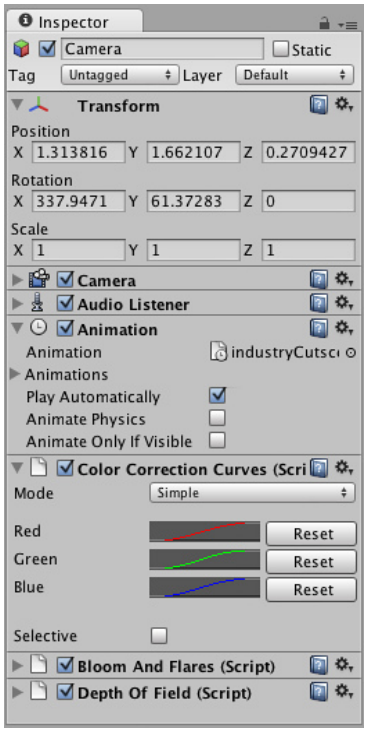


KUVA 15. Unityn Custom Käyttölittymä (Unity Technologies 2013)

Esimerkiksi voidaan ottaa kuvan 15 mukainen toimintapelin viholliseditori. Editorin kautta voidaan vaikuttaa siihen, kuinka voimakas vihollinen on muuttamalla vihollisen haarniskan kestävyyttä ja aseiden tehoa. Käyttöliittymän kautta tämä voi olla muutaman hiirenklikkauksen päässä, mutta ilman visuaalisia työkaluja tieto pitäisi kirjoittaa käsin koodiin tai tekstitiedostoon. Unity onneksi tarjoaa mahdollisuuden laajentaa ohjelman GUI-elementtejä. Tämä mahdollistaa täysin uudenlaisten työkalujen kehittämisen.

4.1 Inspector

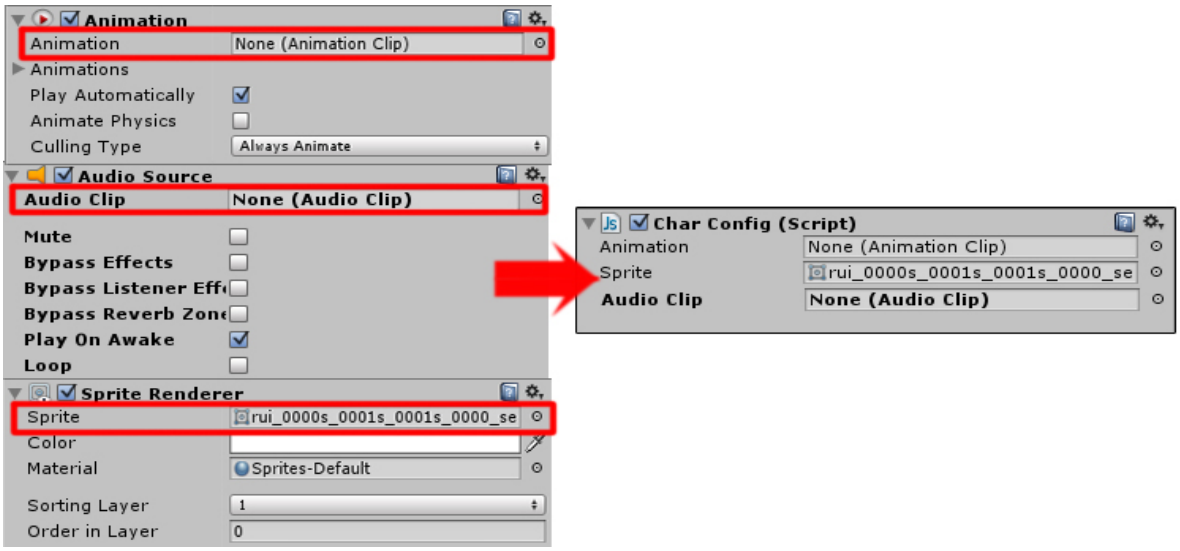
Inspector-ikkuna aukeaa, kun jokin peliobjekti valitaan Unityn käyttöliittymän kautta. Inspector-ikkunan kautta voidaan muokata valitun peliobjektin sisältäviä komponentteja. Jos objektiin on kiinnitetty esimerkiksi äänilähde, voidaan Inspectorin kautta valita toistettava äänitiedosto ja hallita äänenvoimakkuutta. Jokaiselle komponentille on luotu oma Inspector-valikko, jonka rakenteeseen ei voida vaikuttaa. Näiden valikoiden toimintaa on mahdollista laajentaa UnityGUI:lla luomalla täysin uusi Inspector-valikko. (Smith & Queiroz 2013, 11.)



KUVA 16. Inspector (Unity Technologies 2013)

Itse luodulla Inspector-valikolla voi vaikuttaa kaikkiin peliobjektin komponentteihin samalla tavalla kuin Unityn omilla valikoilla. Itseluotuihin ikkunoihin voidaan kuitenkin lisätä toimintoja ja muokattavuutta, joita Unityn omissa valikoissa ei ole. Unityn omat Inspector-valikot voivat sisältää ison määrän eri elementtejä, joista osa voi olla kehitettävän pelin kannalta tarpeettomia. Poimimalla tärkeimmät ominaisuudet omaan Inspector-valikkoon

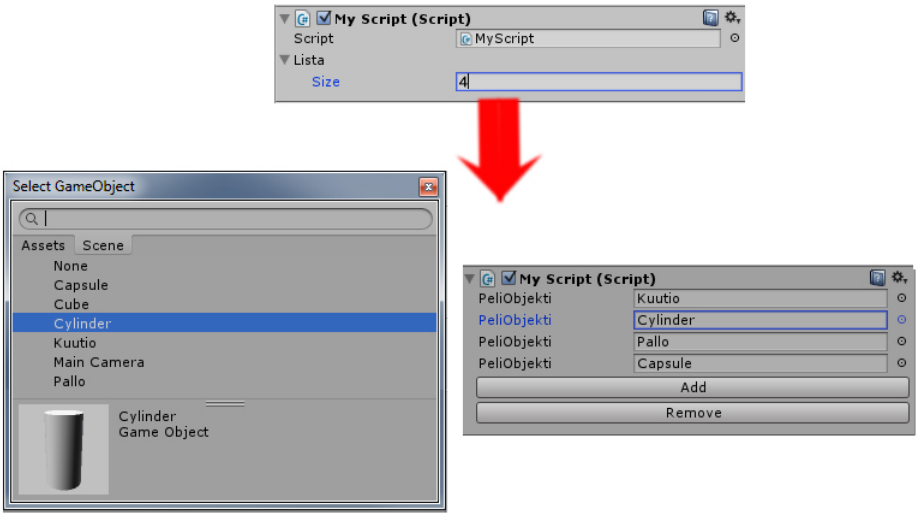
saadaan luotua keskitetty, helposti selattava käyttöliittymä (kuva 17). Käyttöliittymän elementit kytkeytyvät siis kunkin komponentin pelikoodin muuttujiin ja muuttavat niitä. (Unity Technologies 2013b.)



KUVA 17. Inspector Esimerkki 1 (Ruutukaappaus)

Omalla Inspector-valikolla voi myös vaikuttaa peliobjektiin kiinnitettyyn pelikoodiin. Normaalisti Inspectorin kautta on mahdollista muokata vain pelikoodin muuttujien arvoja. Esimerkiksi Color-värimuuttujaa varten Unity tarjoaa kuvankäsittelyohjelmista tutun värinvalintatyökalun. Useissa tapauksissa tämä voi riittää, mutta koodin muokkaamiseksi voidaan luoda myös paljon monipuolisempi käyttöliittymä. (Unity Technologies 2013b.)

Kuvan 18 esimerkissä on luotu peliobjekti, joka syöttää ulos muita peliobjekteja. Inspector-ikkunan kautta voidaan määritellä, mitä peliobjekteja syötetään ulos. Syötettäviä peliobjekteja varten on luotu C#-lista, josta valitaan sattumanvaraisesti yksi ulossyötettävä peliobjekti. Peliobjektilistan muokkaaminen Unityn oman Inspectorin kautta ei ole mahdollista, koska Inspector näyttää vain listan pituuden eikä sen sisältöä. Itse luotuun Inspectoriin voidaan kuitenkin luoda toiminnollisuus, jossa generoidaan listan sisältö Inspector-ikkunaan muokattavaksi.



KUVA 18. Inspector Esimerkki 2 (Ruutukaappaus)

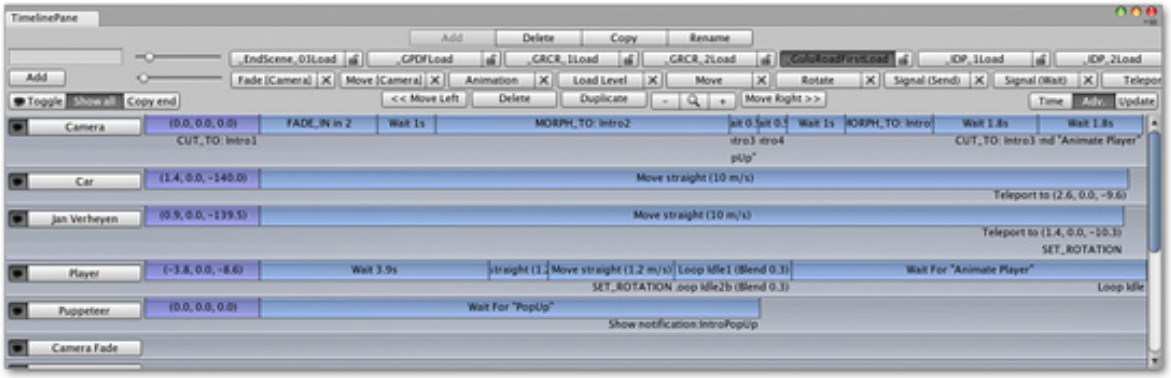
Inspector GUI:ta voidaan siis käyttää yksittäisten objektien pelikoodin visuaaliseen muokkaamiseen. Laajentamalla Inspector GUI:ta voidaan ennestään virtaviivaistaa ja monipuolistaa pelikoodausta ja vähentää tarvetta luoda koodia tekstieditoreilla. Itsekoodatun Inspectorin kautta tehdyt muutokset tallentuvat samalla tavalla kuin normaalin Inspector-ikkunan kautta.

4.2 EditorWindow

Toisin kuin Inspector, EditorWindow ei kiinnity yksittäisiin peliobjekteihin vaan se on yksi itsenäinen työkaluikkuna. Tämä tarkoittaa sitä, että Editor GUI -ikkunan sisältö voi pysyä staattisena valitusta peliobjektista riippumatta. EditorGUI:lla on kuitenkin mahdollista käsitellä valittua peliobjektia Selection.activeGameobject-funktion kautta. Tällä menetelmällä päästään hyvin samanlaiseen lopputulokseen Inspectorin kanssa. Kuvan 18 esimerkki on teoriassa mahdollista toteuttaa Selection-luokan avulla. (Unity Technologies 2011.)

EditorWindowia on parempi käyttää silloin kun halutaan käsitellä laajempia kokonaisuuksia. EditorWindowin ja peliobjektin välille voidaan luoda staattinen linkki, joka ei katoa vaikka peliobjekteja ei olisi valittu. Tämä antaa EditorWindowille edun Inspector-ikkunaan, joka käytännössä katoaa kokonaan, kun objektista poistetaan valinta. Toinen etu on että EditorWindowit ovat itsenäisiä ikkunoita, mikä antaa käyttäjälle mahdollisuuden hallita niiden sijaintia tai sulkea ne kun niitä ei tarvita. EditorWindow-työkalulla voi toteuttaa hyvinkin

monipuolisia työkaluja, kuten Serious Games Interactiven luoma TimelinePanel (kuva 19), jonka kautta voi luoda ja hallita pelien cutscenejä, eli animoituja välikohtauksia. (Unity Technologies 2013b.)



KUVA 19. Time Panel (Unity Technologies 2013)

4.3 Game GUI

Game GUI-luokka ei sinänsä liity pelieditorin käyttöliittymän luomiseen. Game GUI:ta käytetään pelin sisäisen käyttöliittymän luomisessa, eli sen joka näytetään pelaajalle. Toisin kuin muut GUI-elementit, Game GUI:ta ei piirretä Unityn käyttöliittymään vaan pelikameran ”linssiin”, kun peli on käynnissä. (Unity Technologies 2011.)

Game GUI on myös ainoa Unityn käyttöliittymän elementti, jonka kanssa pelaaja pystyy kommunikoimaan peliohjaimella tai hiirellä. Game GUI on ainoa ratkaisu silloin, kun halutaan luoda nappeja joita pelaaja voi klikata, tai tekstejä, joita käyttäjä voi lukea. (Unity Technologies 2011.)

Game GUI-luokka sisältää valmiiksi luotuja tekstilaatikkoja, nappeja ja muita elementtejä, joita voi odottaa löytyvän kaikista peleistä. Unity ei kuitenkaan tarjoa minkäänlaista valmista asettelupohjaa, joten uuden käyttöliittymän koodaaminen voi olla aloittelevalle kehittäjälle haastavaa. Tämä taistelee pelieditorin tarkoitusta vastaan helpottaa pelinkehitystä ja vähentää koodaamista. Unityn sisälle olisi siis hyvä rakentaa valmis käyttöliittymäpohja ja työkalut sen muokkaamiseen. (Unity Technologies 2011.)

4.4 GUI:n ulkoasun muokkaus

Game GUI:n ulkoasun muokkaaminen muistuttaa hyvin paljon HTML- ja CSS-verkkokoodaamista. GUI-objektiin voidaan kiinnittää GUI-Style-pelikoodi, joka generoi automaattisesi uuden Inspector-valikon. Inspector-valikko sisältää ison määrän erilaisia valikoita, joilla GUI:n ulkoasua voi muokata. (Unity Technologies 2011.)

Inspector-valikon kautta objektille voidaan asettaa halutut koordinaatit ja skaalan. Ikkuna sisältää hyvät työkalut erityisesti yksinkertaisen nappien luomiseen. Käyttöliittymän takana toimii koodi, joka havaitsee napinpainallukset ja vaihtaa painettaessa napin tekstuuria. Tekstilaatikoita varten GUI:hin voidaan asettaa CSS:stä tuttuja padding- ja margin-arvoja joiden avulla tekstielementit voidaan asettaa paikalleen. (Unity Technologies 2011.)

GameGUI:n luonnissa on otettava huomioon erilaiset resoluutiot. Unityn GUI ei ota asetetuissa koordinaateissa huomioon eri resoluutioiden aiheuttavia koordinaattien muutoksia. Tämän kiertämiseksi on joko luotava usea eri GUI-Style joka resoluutiolle tai luoda koodi, joka laskee koordinaatit uudestaan näyttökoon mukaan. (Unity Technologies 2011.)

5 TIEDONKÄSITTELY UNITYSSÄ

Pelieditorit ovat pohjimmiltaan automaattisia koodin generoijia. Jotta käyttöliittymään tehdyt muutokset saadaan pysyväksi osaksi peliä, täytyy tieto muutoksista tallentaa ulkoisiin tiedostoihin. Unityn täytyy myös kyetä avaamaan ja muuntamaan nämä tiedostot sellaiseen muotoon, että sekä pelimoottori että pelieditori voivat lukea niitä.

Unity tarjoaa suuren määrän erilaisia tapoja ladata sisältöä peliin. Näistä useimmin käytetty on Unityn oma käyttöliittymä. Käyttäjä voi vetää kuvatiedostoja Unityyn suoraan työpöydältä, Windowsista tutulla tavalla. Ladatut kuvat on sisääntuonnin jälkeen helppo kiinnittää kappaleisiin suoraan käyttöliittymässä. Tämä ei kuitenkaan ole aina optimaalinen ratkaisu. (Smith & Queiroz 2013, 16.)

Kaiken datan lataaminen peliin kerralla nopeuttaa kyllä pelin latausaikoja, mutta täyttää nopeasti pelinmuistin ja aiheuttaa suorituskykyongelmia. Tallentamalla tiedostot Unityn tiedostorakenteen ulkopuolelle ne voidaan ladata peliin vain kun niitä tarvitaan. Tiedostojen tallentaminen ulkoiseen lähteeseen nopeuttaa myös pelin käynnistyksessä ladattavaa datan määrää. Dynaamisesti ladattava sisältö voi olla esimerkiksi kuvagalleria, randomisti ladattavat tekstuurit tai erilaiset tekstisisällöt. (McShaffry 2009, 195.)

5.1 WWW-luokka

WWW-luokkaa voidaan käyttää tiedostojen lataamiseen internetistä tai kovalevyltä tiedoston WWW-osoitteen avulla. Internetistä ladattava sisältö voi olla ihan mitä tahansa dataa, jota Unity pystyy käsittelemään. Data voi olla esimerkiksi piste-ennätyslista, musiikki tai tekstuuri. Dataa voi ladata myös SQL-tietokannoista, joissa pistetilastot tyypillisesti sijaitsevat. Ladattua dataa voi lataamisen jälkeen käsitellä Unityn koodissa normaalisti. WWW-luokka pystyy kommunikoimaan serverin kanssa verkkokoodauksesta tutuilla GET- ja POST-viesteillä. (Unity Technologies 2014.)

WWW-luokkaa on myös mahdollista käyttää datan lokaaliin lataamiseen tietokoneen kovalevyltä. Tiedostojen lataamiseksi tarvitaan kuitenkin tiedoston tarkka tiedostonsijainti. Unityn pelikansion sisällä olevan datan lataaminen onnistuu helposti Application.dataPath-funktion avulla. Funktio palauttaa ohjelmalle pelikansion tiedostosijainnin, alustasta riippumatta. Pelikansion ulkopuolisen datan lataamista kannattaa käyttää harkiten, koska eri alustoissa on aina erilaiset tiedostorakenteet. (Unity Technologies 2014.)

WWW-luokalla on myös mahdollista lähettää dataa tekstidataa serverille perinteisen form-tyyppisen datan avulla. Form-dataa voidaan käyttää esimerkiksi pelinsisäisissä kirjautumisissa tai top-lista tallennuksissa. (Unity Technologies 2014.)

5.2 Resources-kansio

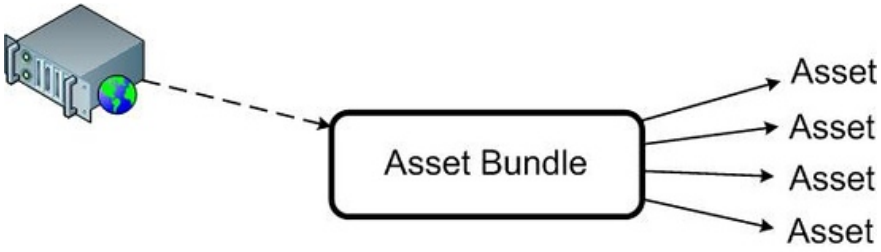
Resources-kansio on Unityn käyttämä erityinen kansio, joka sijaitsee pelin Assets-kansion sisällä. Kaikki data, joka on tallennettu Recourse-kansioon, voidaan ladata Unityn Resources.Load-funktiolla. Resources.Load-funktio on erinomainen silloin kun halutaan ladata dynaamisesti dataa levyltä pelin aikana. Resources-kansioon voidaan tallentaa esimerkiksi iso määrä kuvia, joita voidaan ladata vain, kun peli tarvitsee niitä. (Unity Technologies 2014.)

Resources.Load-funktiolla voidaan ladata samaa dataa kuin käyttölliittymän kautta. Kansiolla on kuitenkin muutamia rajoituksia. Isoin rajoitus on, että Resources-kansio ei varsinaisesti tue alakansioita. Jos alakansioita halutaan käyttää, Resources.Load-funktiolle täytyy antaa myös alakansion nimi, esimerkiksi Resources.Load(”/Alakansio/kuvanimi). Tämä ei ehkä tunnu suurelta ongelmalta, mutta alikansioiden puute vaikeuttaa huomattavasti tiedostojen organisointia. Jos peli sisältää ison määrän dynaamisesta ladattavia kuvia, on niiden oltava samassa kansiossa, mikä puolestaan vaikeuttaa tiedostojen hallintaa. (Unity Technologies 2014.)

Toisekseen Resources-kansiota voi käyttää vain staattisten, muuttumattomien tiedostojen hallintaan. Kun peli kehittämisvaiheessa pakataan julkaisuformaattiin, Resources-kansio ja kaikki sen sisältö pakataan assets-tiedostoon. Assets-

tiedostotyyppi on suojattu tiedosto, eli sitä ei pysty aukaisemaan normaalilla menetelmillä. Normaaleissa olosuhteissa vain pelisovellus kykenee aukaisemaan tiedoston, mutta ei muokkaamaan sen sisältöä. Resources-kansiota ei siis voida käyttää tapauksissa, joissa pelin pitää luoda, muokata tai tallentaa sisältöä (Smith & Queiroz 2013, 212.)

5.3 Assets Bundle



KUVA 20. Assets Bundle (Unity Technologies 2013)

Assets Bundlella on Unity Pro -version ominaisuus, jolla pystyy pakkaamaan peliobjekteja erillisiin pakkaustiedostoon ja tallentamaan ne verkkoserverille kuvan 18 havainnollistamalla tavalla. Normaalisti peliobjektit ovat kiinni siinä peliprojektissa johon ne on luotu. AssetsBundlejen avulla voidaan saavuttaa hajautettu tiedostorakenne, jonka avulla peliobjekteja voi jakaa jopa eri pelien välillä. (Unity Technologies 2013b.)

AssetsBundlejen luomiseksi on kuitenkin luotava manuaalisesti oma Editori-luokka, jonka kautta peliobjektit pakataan AssetsBundleksi. Unityn verkkosivut tarjoavat myös hyvät koodiesimerkit, jotka tallentavat automaattisesti kaikki peliobjekteihin linkitetyt tiedostot samaan AssetsBundleen. AssetsBundleihin voidaan siis pakata esimerkiksi 3d-objekti ja siihen linkitetyt animaatiotiedostot. Tiedostojen lataaminen onnistuu normaalisti WWW-luokan avulla. (Unity Technologies 2013b.)

5.4 Streaming Assets

Normaalisesti kun peli on valmis, kaikki pelitiedostot, kuten kuvat ja tekstitiedostot, pakataan yhteen muun pelidatan kanssa. Jos tiedostot halutaan säilyttää sellaisenaan pakkaamattomassa muodossa, voidaan käyttää

StreamingAssets-funktiota. Samalla tavoin kuin Resources-kansio, StreamingAssetsilla on myös oma määrätty kansionsa peliprojektissa. Toisin kuin Resources-kansio, joka pakataan pelin julkaisussa, StreamingAssets-kansio on avoin myös muille ohjelmille. StreamingAsseteja käytetään siis silloin, kun pelidata halutaan säilyttää yleisesti luettavassa muodossa. (Unity Technologies 2013b.)

StreamingAssets ei kuitenkaan toimi täydellisesti kaikilla alustoilla. StreamingAssetsien lataamiseksi tarvitaan tarkka tiedostoreitti, mikä johtaa ongelmiin alustoilla, joissa on poikkeava tietorakenne. StreamingAssets-kansion löytäminen kaikilta alustoilta vaatii kovakoodatut tiedostoreitit jokaiselle alustalle. Unity kykenee kuitenkin hakemaan pelikansion sijainnin Application.datapath-metodilla, mutta StreamingAssets-kansion sijainti eroaa eri alustojen välillä. Kansioiden tiedostoreitit ovat yleisimmillä alustoilla seuraavanlaiset:

```
Windosw & Mac:
path = Application.dataPath + "/StreamingAssets";
IOS:
path = Application.dataPath + "/Raw";
Android:
path = "jar:file://" + Application.dataPath + "!/assets/";
```

(Unity Technologies 2014.)

Androidissa kaikki pelidata tallennetaan jar-tiedoston sisään. Jar-tiedostot vastaavat Windowsin zip-tiedostoja. Tiedostot saa tarvittaessa helposti purettua, mutta ei voida olettaa, että jokainen käyttäjä kävisi itse purkamassa tiedostot. Ainoa mahdollisuus lukea dataa StreamingAsseteista Androidilla on käyttää Unityn WWW-luokkaa tiedoston lataamiseen ja ulkoista purkuohjelmaa jar-tiedoston purkamiseen. WebPlayerissä StreamingAssetseja ei voi käyttää käytännössä ollenkaan, koska internetissä ei ole tiedostorakennetta samalla tavalla kuin esimerkiksi pöytäkoneissa. (Unity Technologies 2013b.)

5.5 PlayerPrefs

PlayerPrefs on Unityn ainoa oma työkalu, jolla voi tallentaa dataa levylle. PlayerPrefsiä käytetään yleensä tallentamaan tietoa pelaajan peliasetuksista, jotka ladataan kun pelaaja seuraavan kerran käynnistää pelin. Peliasetuksista voidaan säätää esimerkiksi, kuinka kovalla pelaaja pitää musiikkia tai minkä vaikeustason hän valisti viimeksi. (Unity Technologies 2013b)

Toiminnollisuudeltaan PlayerPrefs on varsin rajoitettu. Funktiolla voi tallentaa vain yhden tyyppistä dataa: mikä datan nimi on ja mikä sen arvo on. Datan arvo voi olla joko tekstiä, integraaleja tai float-numeroita. Tämäntyyppistä dataa käytetään peleissä asetusdatan ja pelaajan tallennusdatan käsittelyyn. Muihin tarkoituksiin parempi ratkaisu on luoda oma datantallennus-funktio koodaamalla. (Unity Technologies 2013b.)

5.6 Tekstisisältö

Unityssä tekstitiedostoja käsitellään TextAsset-luokkana. TextAssetsit ovat normaaleja tekstitiedostoja, kuten txt-, html-, xml- ja json-tiedostotyypit. Kun tekstitiedostot on ladattu TextAssetteina, tekstitiedostot menettävät tiedostotyypin erikoispiirteensä ja niitä käsitellään normaaleina tekstitiedostoina. Tämä rajoitus on mahdollista kiertää koodaamalla. Testitiedostoihin voidaan tallentaa esimerkiksi save-dataa, eli tallennusdataa pelaajan etenemisestä. TextAsset-luokka ei kuitenkaan sisällä toimintoja tiedostojen tallentamiseen. Tallennusta ja lataamista varten tukeudutaan joko C#- tai Javascript-koodikieleen. (Unity Technologies 2013b.)

Koska Unity tukee C#-koodauskieltä, tukee se samalla C#:n lukuisia merkkijonojen käsittelyfunktioita sekä System.IO-nimiavaruutta, joka sisältää paljon tiedoston käsittelyyn liittyviä toimintoja. Näillä toiminnoilla voidaan ladata, tallentaa ja muokata erilaista tekstidataa. (Smith & Queiroz 2013, 247.)

Kun tekstiä halutaan tallentaa tai lukea levyltä, voidaan ottaa käyttöön C#:n System.IO-luokka. System.IO tarjoaa File-, Writer- sekä Reader-luokat. File-luokalla voi tehdä yksikertaisia toimintoja, kuten luoda, siirtää, kopioida ja poistaa

tiedostoja. File-luokalla voidaan myös lisätä tekstiä tiedoston loppuun. File-luokkaa on siis hyvä käyttää tiedostojen luonnissa, mutta muut luokat hoitavat muokkaamisen paljon monipuolisemmin. FileInfo on toiminnoiltaan hyvin samanlainen, mutta se kykenee suorittamaan samalle tiedostolle useamman toiminnon kerrallaan. (Moghadampour 2009, 324–327.)

FileStreamit ovat .NET-pohjaisia funktiota, joita voidaan käyttää sekä tiedostojen lukemiseen että muokkaamiseen. FileStreamit toimivat kuitenkin vain Windows- tai Mac-pohjaisissa standalone-sovelluksissa. Ne toimivat vain siis työpöytä-sovelluksissa eikä esimerkiksi Unityn Web Playerissä. Streamien sijasta on parempi käyttää instanssipohjaista ratkaisua, jossa tekstitiedosto tallennetaan Unityssä muuttujaan. Tekstitiedosto kopioidaan väliaikaisesti Unityn välimuistiin Recourse.Load-metodilla ja sitä käsitellään TextAssettina. Kun tiedoston muokkaus on valmis, tallennetaan se vanhan tekstitiedoston päälle. (Smith & Queiroz 2013, 245.)

Writer- ja Reader-luokkia käytetään usein yhdessä tekstitiedostojen avaamiseen ja kirjoittamiseen. Nämä luokat tukeutuvat kuitenkin FileStream-luokkaan, joten ne eivät kykene käsittelemään kovalevyllä olevaa dataa muualla kuin työpöytäympäristössä. Tämän takia Reader-luokan sijasta, tiedostojen lukemiseen kannattaa käyttää File-luokkaa, WWW-luokkaa tai Recouces.Loadia. (Smith & Queiroz 2013, 245.)

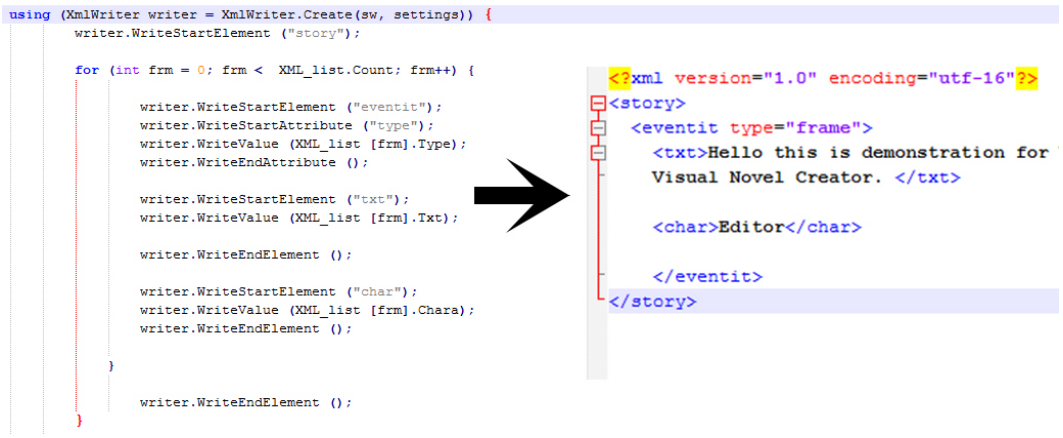
System.IO sisältää erilaisia Writer-luokkia. Näistä luokista tekstinkäsittelyyn sopivat StringWriter- ja TextWriter-luokat. StringWriter tallentaa saadun tekstin string-tyyppiseen muuttujaan, kun taas Writer tallentaa sen tekstitiedostoon. Writer-luokalla voidaan luoda sekä tallentaa esimerkiksi binääritiedostoja sekä normaaleja tekstitiedostoja. Writer-luokan etuna on sen kyky luoda ja tallentaa tiedostoja samalla metodilla. Normaalien tekstitiedostojen käsittelyyn se ei kuitenkaan tarjoa kovinkaan paljoa apua. (Smith & Queiroz 2013, 353–355.)

Tiedostojen avaamisessa on siis järkevintä käyttää joko WWW-luokkaa, Recourse.Loadia tai C#:n File-luokkaa. FileSreamin ja Reader-luokan käyttöä kannattaa harkita vain jos pelieditori on suunnattu vain työpöytä-peleille.

5.7 XML

XML-tiedostoja käytetään hyvin usein erilaisissa peleissä. Niiden looginen tiedostorakenne täyttää hyvin videopelien vaatimukset. Unityssä ei ole oletuksena luokkaa XML-dokumenteille vaan ne luokitellaan TextAsseteiksi (Unity Technologies 2013b). Koska Unity käsittelee XML-tiedostoja normaaleina tekstitiedostoina, ohjelman omilla työkaluilla ei voida lukea XML-tietorakennetta. XML-dokumentteja lukiessa pitää aina tukeutua C# tai Javascript koodaamiseen. C#:n XML-nimiavaruus tarjoaa kuitenkin kaikki tarpeelliset työkalut XML-tiedostojen lukemiseen ja luomiseen. Muuntamalla TextAssetin merkkijono XMLDocument-luokkaan palautetaan tekstille XML-tietorakenne. XMLDocument-luokka on kuitenkin vielä kuin avaamaton XML-tiedosto, jota ei vielä suoraan pysty lukemaan. (Smith & Queiroz 2013, 251.)

XML-tietorakenne rakentuu yksittäisistä nodeista, jotka voivat sisältää lapsi-nodeja tai tekstidataa. Nodejen lukemiseen XML-nimiavaruudessa on oma XMLNode-luokka. Jos XML-tiedostorakenteessa on esimerkiksi node, joka sisältää lapsi-noden, koko tiedostorakenteen lukemiseksi tarvitaan kaksi XMLNode-muuttujaa. XMLNode-luokka sisältää XML-tiedoston lukemiselle oleellisia metodeja, kuten lapsi-nodejen valitsemisen ja noden tekstisisällön käsittelyn. Jotta XML-dataa voitaisiin lukea XMLNode-luokan avulla, on teksti ensin muunnettava XMLDocument-luokkaan. (Smith & Queiroz 2013, 250–252.)



Kuva 21. XML Esimerkki (Ruutukaappaus)

Samoin kuin System.IO-nimiavaruus, XML-luokka sisältää myös Writer-luokan. XMLWriter on kuitenkin hyvin monipuolisempi kuin TextWriter. Toisin kuin

TextWriter, jolla kirjoitettiin yksinkertaista tekstidataa, XML-writerilla voidaan kirjoittaa monimutkaisia XML-tiedostorakenteita suoraan koodissa. XML Writer pystyy automaattisesti generoimaan XML-nodeja ja muokkaamaan niiden sisältöä ja attribuutteja (kuva 21). (Smith & Queiroz 2013, 252–254.)

5.8 Bittikartat

Unityssä voi ladata bittikarttoja moneen eri tarkoitukseen. Yleisimmät ovat erilaiset 3d-kappaleiden tekstuurit ja käyttöliittymän sisältö. Unity tukee yleisimpiä bittikarttamuotoja, kuten jpeg-, png- ja bmp-muotoja. Kuvia voi lisätä joko Unityn käyttöliittymän kautta tai dynaamisesti yllämainituilla menetelmillä. Unity tukee automaattisesti myös kuvien läpinäkyvyyttä. (Unity Technologies 2013b.)

Unityssä bittikarttoja käsitellään joko Texture2d -tai Sprite-luokalla. Texture2d sisältää ison määrän esiasetuksia eri tarkoituksiin. Ennen Sprite-luokkaa, Texture2d oli ainoa luokka bittikarttoja varten. Texture2d-bittikarttoja käytetään tämän takia edelleen kaikissa tekstuureja vaativissa tehtävissä. Tärkeimpinä käyttökohteina ovat 3d-mallien tekstuurit ja käyttöliittymän grafiikat. Texture2d-kuvia voi myös käyttää normal-mappeina, light-mappeina ja muina 3d-ohjelmista tuttuina tekstuurikarttoina. (Unity Technologies 2013b.)

Unity 4.3 -versiosta eteenpäin, Unity on tukenut 2d-spritekuvia. Aikaisemmin spritejen käyttämiseen tarvittiin kolmannen osapuolen lisäosia, jotka edelleen tukeutuivat Texture2d-luokkaan. Unityn 4.3 -versio esitteli Sprite-luokan ja Spritejen käsittelyä helpottavat Sprite Editorin ja Sprite Packerin. (Unity Technologies 2013c.)

Unityn import -työkalu tarjoaa kehittäjille mahdollisuuden määritellä, kuinka isona kuvat ladataan peliin. Esimerkiksi 1000 x 1000 pikselin kuva voidaan ladata peliin 500 x 500 tarkkuudella. Tämä ei kuitenkaan kutista levyllä olevaa kuvatiedostoa, vaan Unity lataa kuvan pienemmällä tarkkuudella. Isojen kuvien lataaminen voi olla raskas tehtävä mobiililaitteiden näyttösuorittimelle, minkä takia Unity tarjoaa mahdollisuuden pakata isot kuvatiedostot pienempään kokoon pelin julkaisun yhteydessä. (Unity Technologies 2013b.)

6 CASE

6.1 Casen esittely

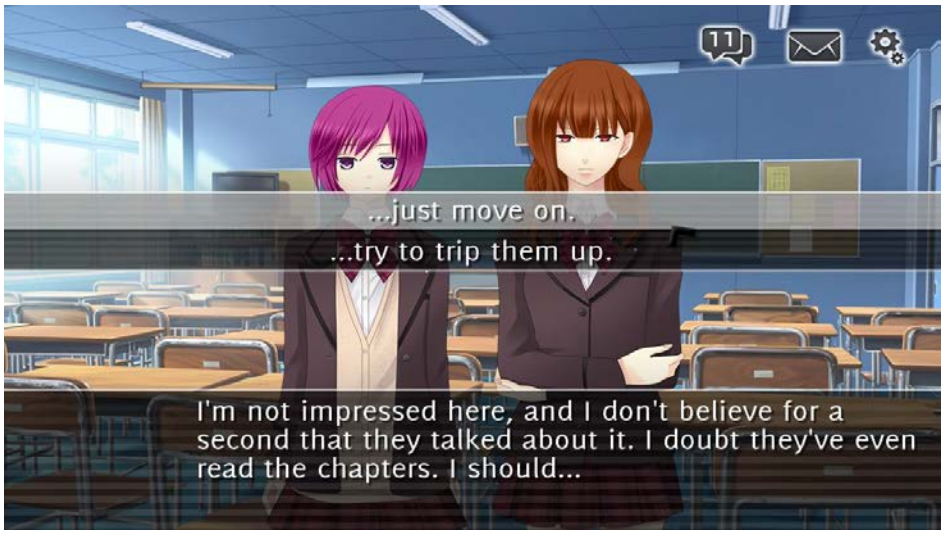
Casen tavoitteena on luoda Unity 3D:n päälle rakennettu visual novel -pelimoottori ja editori. Ideana on luoda loppukäyttäjälle monipuolinen työkalu visual novellien luomiseen. Työnimellä Visu10 luodun alustan on tarkoitus kattaa visual novelleille tyypilliset ominaisuudet ja toiminnot.

Visu10:n käyttöliittymä koodataan suoraan Unityn omaan käyttöliittymään. Alusta suunnitellaan niin, että loppukäyttäjä kykenee muokkaamaan pelin kaikkia ominaisuuksia, ilman tarvetta koskea pelin koodiin. Alusta suunnitellaan mahdollisimman monipuoliseksi ja helppokäyttöiseksi. Tavoitteena on luoda sujuva kokonaisuus, jolla voi tehdä sekä isoja että pieniä pelejä, ilman minkäänlaista peliohjelmointikokemusta. Unityn lisäksi alustan kanssa tullaan tarvitsemaan myös jonkinlaista kuvankäsittelyohjelmaa. Tämän työn esimerkeissä käytetään Adobe Photoshop -ohjelmistoa.

Pelieditorin pohjaksi valittiin Unity3d, koska sen kautta on helppo julkaista pelejä useille eri alustoille. Visual novel -pelit soveltuvat hyvin sekä pöytäkoneille että mobiililaitteille. Unityn monipuoliset julkaisutyökalut tulevat varmasti helpottamaan uusien pelien julkaisemista. Lisäksi Unityn 4.3-version uudet 2d-toiminnot soveltuvat hyvin pelityypin luomiseen.

6.2 Visual novellin määritys

Visual novel on Japanista lähtöisin oleva peligenre. Visual novellit ovat nimensä mukaisesti enemmän novelleja kuin videopelejä. Yksinkertaisimmillaan visual novellit ovat kuvakertomuksia, joissa tarina kerrotaan erilaisten hahmokuvien, tekstien ja äänten avulla. Harva visual novel on kuitenkaan niin yksinkertainen. Peligenre luokitellaan interaktiiviseksi fiktioksi, koska pelaaja voi itse vaikuttaa pelin tarinaan tekemällä erilaisia valintoja pelin edetessä (kuva 22). Tämä johtaa hyvinkin monihaaraisiin tarinoihin ja useisiin erilaisiin lopetuksiin. (Lebowitz & Klug 2011, 193–194.)



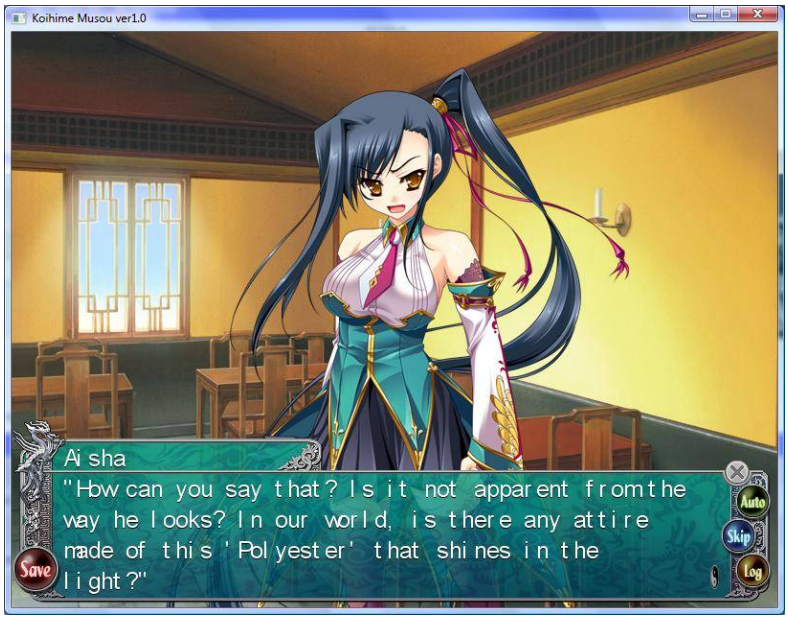
KUVA 22. Visual Novel (Vestal A. 2013)

Nykyään visual novellit on usein sekoitettu yhteen useiden erilaisten peligenrejen kanssa. Esimerkiksi 2013 julkaistussa Fire Emblem: Awakening -pelissä tarina kerrotaan pääosin pelaajalle visual novel -muodossa, mutta itse pelattava osuus on 3d-vuoropohjastrategiaa. Visual novel on suosittu formaatti, koska se on pohjimmiltaan hyvin yksinkertainen toteuttaa. Kuten kuvassa 23 näkyy, Fire Emblemin 3d-hahmomallit ovat epätarkkoja lowpoly-malleja. Hahmojen esittäminen käsin piirrettyinä 2d-kuvina on visuaalisesti miellyttävämpää kuin lowpoly-mallien käyttäminen. Tällä tavoin on myös mahdollista säästää hahmomallinnuksesta aiheutuvia kustannuksia. (Wikipedia 2014d.)

KUVA 23. Fire Emblem Awakening (Tøvik J. E. 2013)

6.3 Suunnittelu

Suunnittelun ensimmäinen vaihe oli luonnollisesti tarvittavien toimintojen kartoittaminen. Tätä varten tutkittiin monia suosittuja genren pelejä. Visual novelleissa on aina suhteellisen samat perustoiminnot. Kuvassa 24 on kuva tyypillisestä visual novellista. Analysoimalla kuvaa saadaan jo hyvä kuva siitä, mitä peliin vaaditaan.



KUVA 24. Visual Novel esimerkki (CB Interavtive Inc. 2007)

Analysoinnilla ei pyritty luomaan kopiota tietystä pelistä, vaan kartoittamaan visual novelleille välttämättömät toiminnot. Toimintojen kartoituksen jälkeen ne kategorisoitiin aihealueen mukaan. Erityisen tärkeät kategoriat olivat tekstitoiminnot, kuvatoiminnot, animaatiotoiminnot ja UI-toiminnot.

Tekstitoiminnot ovat yhdessä kuvatoimintojen kanssa visual novellien perusta. Kuten kuvassa 24 näkyy, tekstilaatikko peittää huomattavan osan pelin ruudusta. Tekstisisällöksi luetaan myös mahdollisen puhujan nimi, joka näkyy ison testilaatikon yläpuolella. Laatikon teksti vaihtuu jokaisen framen välillä. Framet ovat kuin kirjan sivuja, jokaisella omat tekstinsä ja kuvansa. Tarinat etenevät siirtymällä framesta toiseen. Alustan täytyy siis kyetä lukemaan tekstiä ja tulostamaan se ruudulle halutussa framessa. Framepohjaisuus toi mukanaan idean XML-pohjaiselle tietorakenteelle.

Kuvatoiminnot, eli kuvien lataus, asettelu ja hallinta, ovat epäilemättä tärkein osa visual novelleja. Visuaalisuus on yksi niistä harvoista tekijöistä, jotka määrittelevät peligenren. Pelin täytyy kyetä lukemaan iso määrä kuvia pelin aikana. Käyttäjän täytyy siis kyetä hallitsemaan, mitä kuvia hän haluaa ladata tietyssä framessa. Ison budjetin visual novellit voivat sisältää satoja, tai jopa tuhansia yksittäisiä hahmo-, tausta- ja event-kuvia. Alustaa suunnitellessa piti siis ottaa huomioon, miten tehdä kuvien lataamisen peliin mahdollisimman vaivatonta.

Vaikka tätä ei stillkuvasta näekään, visual novellit sisältävät aina myös jonkin tyyppisiä yksinkertaisia animaatioita. Yleensä kun hahmo ilmestyy ruutuun, se tapahtuu sisäänhäivytyksellä tai siirtämällä hahmo sisään ruudun ulkopuolelta. Rajoitetun 2d-ikkunan takia monimutkaisia animaatiotyökaluja ei onneksi tarvita. Visual novellit sisältävät usein tuhansia frameja, mikä johtaisi isoon määrään yksittäisiä animaatiotiedostoja. Tämän rajoituksen kiertämisestä kehkeytyi suuri haaste kehittämisvaiheessa.

Viimeisenä ison kategoriana määriteltiin UI-toiminnot. Tällä tarkoitetaan pelin sisäisiä Game GUI-menuja ja valikoita eikä itse pelieditorin käyttöliittymää. UI:n rakentamisessa pitää ottaa huomioon interaktiivisuus, visuaalinen ulkoasu ja muokattavuus. Nappien ja valikoiden toiminnollisuus pitää olla valmiina pelieditorissa, lisäksi käyttäjän kokemuksen helpottamiseksi valikoita varten tulisi luoda myös oma editori.

6.4 Hahmojen luominen

Visual novellit voivat sisältää satoja eri kuvia (Lebowitz & Klug 2011, 193–194) . Tästä syystä kuvien manuaalinen lisääminen peliin Unityn käyttöliittymän kautta ei ollut järkevä vaihtoehto. Tämän takia kuvien lataamiseen päätettiin käyttää Resources-kansiota.

Visu10:ssä on hahmoja varten luotu pohja eli template-peliobjekti, johon on valmiiksi asetettu kuvien esittämiseen tarvittavat komponentit ja lapsi-peliobjektit. Peliobjektiin on kiinnitetty myös hahmo animaatioita ja käyttöliittymää varten

luodut pelikoodit. Tällä hetkellä pelihahmo on rajoitettu kahteen layeriin, mutta teoriassa tämä määrä on rajaton.

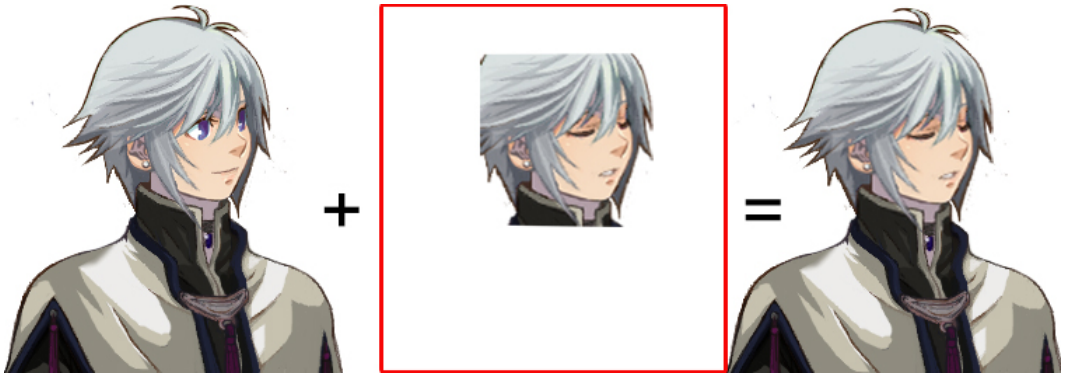
Visual novelleissa hahmot rakentuvat aina kutakuinkin samalla tavalla. Hahmot rakentuvat tyypillisesti 2 tasosta: vartalosta ja kasvoista. Ison budjetin peleissä hahmoilla voi olla useita erilaisia asentoja, kullekin omat vaatetukset ja kasvot. Jotta vältyttäisiin luomasta satoja isokokoisia kuvia jokaiselle vartalo-kasvo yhdistelmälle, on tullut standardiksi luoda yksi iso vartalokuva ja useita pieniä kasvokuvia. Kuvassa 25 esitetään tyypillinen hahmorakenne. Kasvokuvat ovat todellisuudessa erillisiä kuvatiedostoja.

KUVA 25. Hahmorakenne 1 (Soramani Kagome . 2013)

Pienikokoisissa peleissä ei välttämättä haittaa, vaikka kasvot ja vartalo olisivatkin samoissa kuvissa, mutta ison budjetin pelien vartalokuvat ovat yleensä useiden megabittien kokoisia, kun taas kasvokuvissa puhutaan vain kilotavuista. Erottamalla kuvat toisistaan säästetään siis huomattavasti levytilaa. Tämä tuo kuitenkin haasteen alustan koodaajalla. Iso kysymys on, miten kasvot saadaan vartalon suhteen oikealle kohdalle.

Vaikka Visu10-alusta tukee useita eri tapoja toteuttaa hahmorakenne, päädyttiin yksinkertaisuuden vuoksi seuraavaan rakenteeseen. Yksinkertaisin ratkaisu on tallentaa vartalo ja kasvot täsmälleen samaan kokoon. Tällä tavalla kuvat asettuvat automaattisesti samaan kokoon ja koordinaatteihin. Ainoa huono puoli ratkaisussa ovat aavistuksen verran isommat tiedostokoot. Erossa on kuitenkin kyse vain

muutamasta kymmenestä kilotavusta kuvaa kohden. Hahmojen asettelusta esimerkki kuvassa 26.



KUVA 26. Hahmorakenne 2 (Soramani Kagome . 2013)

Toinen jokseenkin yleinen tapa olisi luoda koodi, joka kirjoittaisi jokaisen kasvon halutun sijaintitiedon tekstitiedostoon. Tämä tapa mahdollistaisi sen. että kasvot voisivat olla minkä kokoiset tahansa. Tällä hetkellä käytetään kuitenkin yksinkertaisempaa vaihtoehtoa.

6.5 Animaatiot

Visual novellien animaatiot voivat yksinkertaisimmillaan alle sekunnin pituisia siirtymiä oikealta vasemmalle. Monimutkaisimmillaankin ne ovat vain yksinkertaisia tärinäanimaatioita tai heilumisanimaatioita.

Unityssä on sisäänrakennettu animaatioeditori, jolla luodaan ja muokataan animaatioita. Normaali ratkaisu olisi käyttää tätä työkalua animaatioiden luomiseen. Ongelmaksi kehkeytyy kuitenkin se, että Unityn animaatio-editorin kautta luodut animaatiot vaativat aina oman tiedoston. Toisin sanoen jokainen yksinkertainen siirtymä- ja häivitysanimaatio vaatisi oman animaatiotiedoston. Tämä johtaisi vaikeasti ylläpidettävään animaatiotiedosto viidakkoon, jota halutaan välttää.

Tämän takia Visu10:n on luotu omat tiedostovapaat animaatiotyökalut. Uudet animaatiotyypit soveltuvat tarvittavien yksinkertaisten animaatioiden luomiseen.

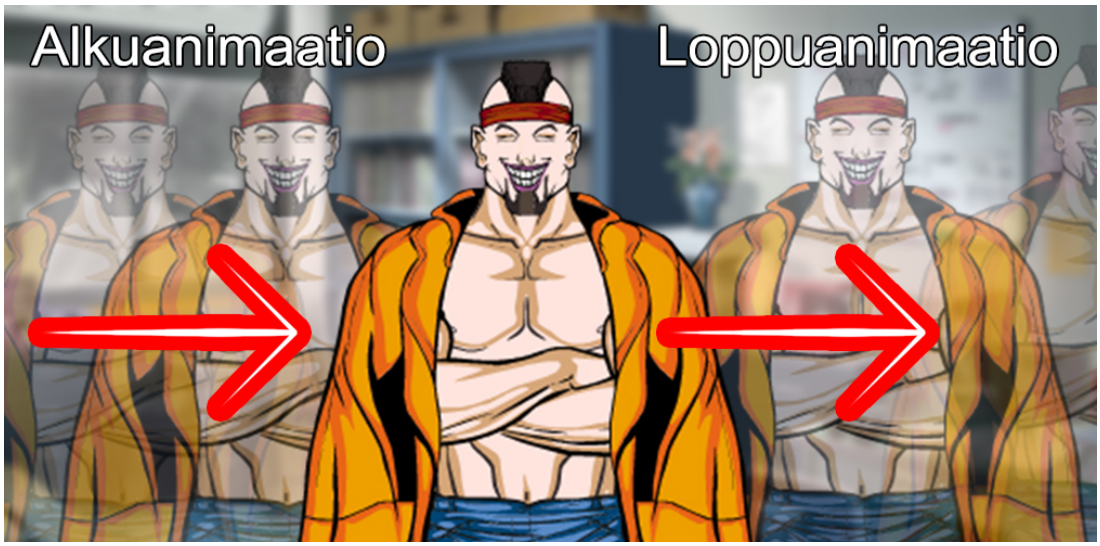
Monimutkaisia animaatioita varten tarjotaan myös vaihtoehto käyttää Unityn omia työkaluja. Käytössä olevat animaatiotyypit ovat seuraavat:

- Scripti-animaatiot
- Tiedosto-animaatiot
- Crossfade-animaatiot
- Camera-animaatiot.

6.5.1 Scripti-animaatiot

Scripti-animaatiot ovat dynaamisesti luotavia animaatioita, jotka ladataan suoraan XML:stä. Varastoimalla animaatiokomennot XML:ään vältytään isolta määrältä yksittäisiä animaatiotiedostoja. Scripti-animaatioiden tarkoitus on luoda yksinkertaisia animaatioita. Scripti-animaatioita on yhteensä 4 eri tyyppiä: Move, Scale, Fade ja File. Animaatioita voidaan toistaa myös samaan aikaan.

Kaikilla scripti-animaatiolla on kaksi eri variaatiota, joka määrää animaation suunnan. Animaation suunta riippuu siitä, missä kohtaa framea animaatiot toistetaan. Scripti-animaatioden parametrit ovat aina suhteessa koordinaatteihin, mihin hahmo on luotu.



KUVA 27. Visu10 Animaatiot (Ruutukaappaus)

Kuvan 27 hahmolle on luotu sekä alku- että loppuanimaatio, kullakin kaksi animaatiota: Move ja Fade.

Alkuanimaation komento on seuraavanlainen:

*Move(**0**,50,2.5,linear);*

Fade(0,2.5,linear);

Loppuanimaatio on vastaavasti:

*Move(**500**,50,2.5,linear);*

Fade(0,2.5,linear);

Kahden komennon ero on siis vain Move-komennon x-arvo. Alku animaatiossa hahmo siirtyy annetuista koordinaateista hahmon pääkoordinaatteihin. Tätä vastoin loppuanimaatiossa hahmo siirtyy pääkoordinaateista loppukoordinaatteihin.

Jaottelemalla scripti-animaatiot alku- ja loppuanimaatioihin, saadaan luotua dynaamisesti muokkautuva animaatioympäristö, jossa animaatiot muokkautuvat aina suhteessa hahmon pääkoordinaatteihin. Kehityksen alkuvaiheessa logiikka oli täysin erilainen. Aikaisemmin animaatio toistettiin aina samalla tavalla, riippumatta missä kohtaa framea animaatio toistettiin. Animaation suunta oli siis riippumaton pääkoordinaateista, minkä takia hahmon sijaintia oli vaikea muokata käyttöliittymän kautta.

Move- ja Scale-animaatiot ovat nimensä mukaisesti yksinkertaisia liikkumis- ja skaalaus animaatioita. Animaatiolle annetaan parametreiksi halutut koordinaatit, josta animaatio lähtee liikkeelle, tai koordinaatit joihin se päättyy. Fade-animaatio on todennäköisesti kaikkein käytetyin animaatiotyyppi, koska se on yksinkertaisin tapa tuoda tai poistaa hahmo näkyvistä.

6.5.2 Muut animaatiot

Tiedosto-animaatioissa käytettään Unityn omia animaatiotyökaluja. Visu10-animaatiotyökaluilla ei saada aikaan monimutkaisia tai looppaavia animaatioita. Tätä varten alustalle luotiin myös tuki Unityn omalle animointityökalulle, jolla saa aikaan paljon monimutkaisempia animaatioita. Unityn omat animaatiot vaativat kuitenkin animaatiotiedoston. XML sisältää ladattavan animaatiotiedoston nimen.

Crossfade-animaatioita käytetään kahden bittikartan välisissä ristihäivytyksissä. Crossfadeja käytetään esimerkiksi kun halutaan muuttaa ruudulla olevan hahmon ilmettä tai asentoa. Toisin kuin scripti-animaatioiden Fade-animaatio, Crossfade ei vaikuta koko hahmopeliobjektiin, vaan pelkästään hahmon haluttuun layer-tasoon. Crossfadella voidaan siis muuttaa hahmon kasvojen ilmettä, mutta pitää vartalon asento samana tai päinvastoin. Crossfadeissa häivytettävä kuva asetetaan pohjakuvan päälle. Häivyttämällä kuva hitaasti pois saadaan aikaan pehmeä siirtymä kuvien välillä. Crossfade kuvan voi valita Visu10-käyttöliittymän kautta.

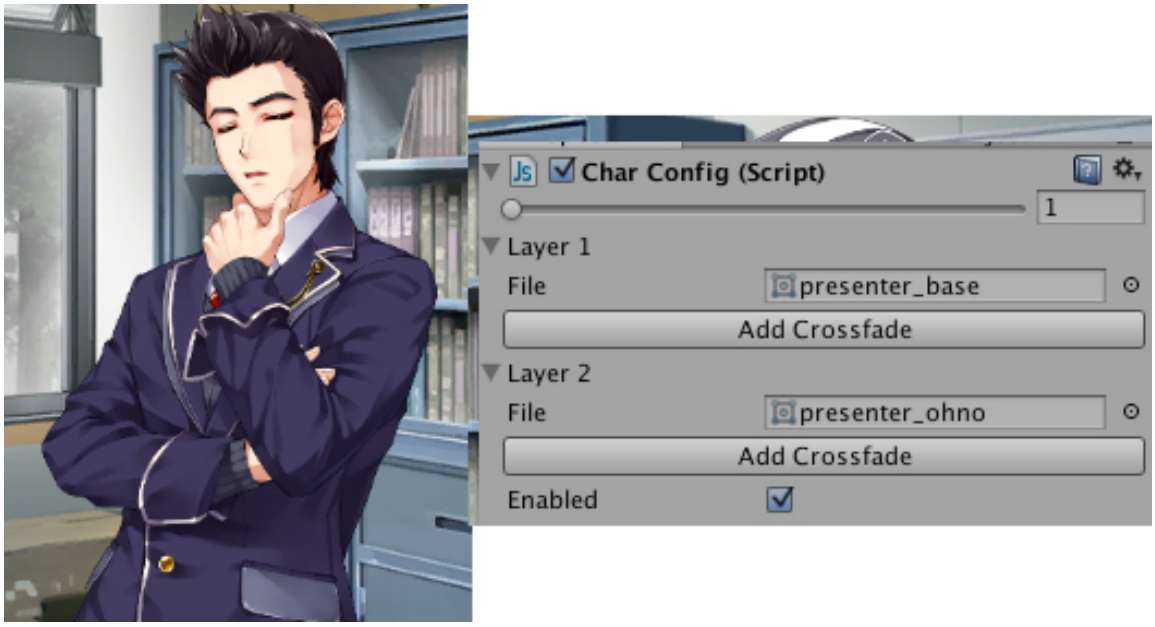
Kamera-animaatiota käytetään silloin kun halutaan tummentaa koko ruutu johonkin väriin tai tekstuuriin. Yleisimmät käyttötarkoitukset kamera-animaatioille ovat erilaiset siirtymät kohtausten välillä.

6.6 Editorin käyttöliittymä

Visu10-editorin käyttöliittymä sisältää työkalut, joilla voi muokata yksittäisiä frameja ja niiden sisältöä. Käyttäjä voi navigoida eri framejen ja XML-tiedostojen välillä yksinkertaisilla navigointityökaluilla. Toistaiseksi Visu10 käyttää Unityn omaa käyttöliittymää hyväkseen GameGUI:n muokkaamisessa. Muut tärkeät muokkaustyökalut on koodattu joko Inspectoriin tai Editor Window:hin.

6.6.1 Hahmoeditori

Hahmoeditorin kautta muokataan hahmojen kuvatiedostoja ja kontrolloidaan niiden syvyysjärjestystä peliruudulla. Kuvassa 28 näkyvien File-laatikoiden kautta voidaan helposti valita haluttu bittikartta-tiedosto ja kiinnittää se halutulle layerille. Toistaiseksi käyttöliittymä tukee vain kahta kuvatasoa. Teoriassa editoria on mahdollista laajentaa niin, että käyttäjä voi itse luoda uusia layeritä.

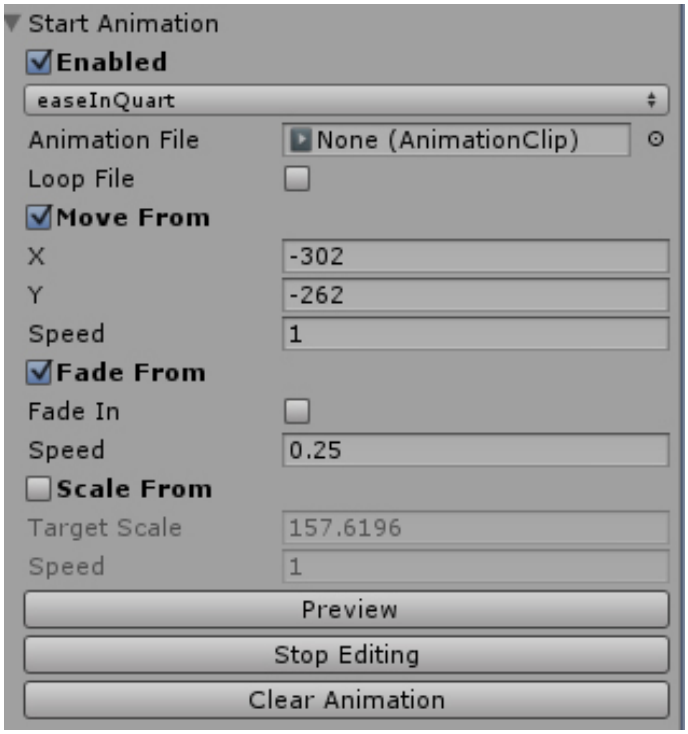


KUVA 28. Hahmo Editori (Ruutukaappaus)

Hahmoeditorin kautta voidaan myös ottaa käyttöön Crossfade-animaatiot. Crossfadeissä käytettävien kuvien valinta onnistuu samalla tavalla kuin normaalien kuvien valinta. Crossfade-kuva ja sen layeri tuhotaan automaattisesti animaation lopussa.

6.6.2 Animaatioeditori

Animaatioeditorilla (kuva 29) luodaan ja muokataan XML:stä luettuja animaatioita. Animaatioeditori on koko alustan monimutkaisin toiminto. Editori on luotu Unityn Inspector GUI-luokalla, joten se muokkaa aina valitun hahmon animaatioita. Editorissa voi säätää script-animaatioiden kaikkia ominaisuuksia sekä määrittää vaihtoehtoisen animaation tiedoston. Editorin arvot ovat yhteydessä hahmon C#-animaatio-luokan arvoihin.

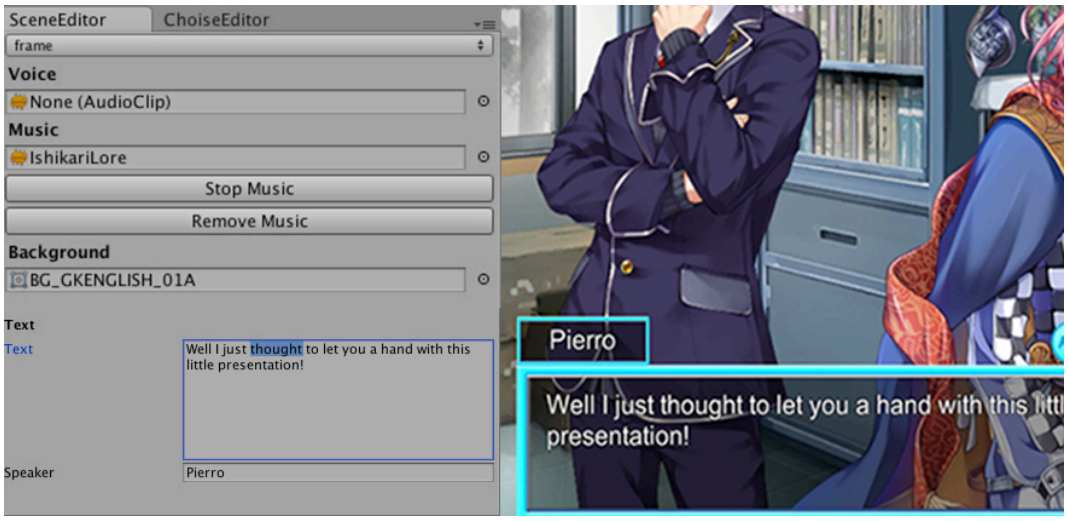


KUVA 29. Animaatio Editori (Ruutukaappaus)

Editointi tapahtuu joko manuaalisesti säätelemällä tekstilaatikoiden arvoja tai käyttämällä Visu10:n Edit-moodia. Edit-moodi käynnistetään painamalla editorin Start Editing -näppäintä. Edit-moodissa käyttäjä pystyy liikuttamaan hahmoa hiirellä. Hahmon sijainti Edit-moodissa on animaation tyypistä riippuen animaation alku -tai loppupiste. Edit-moodin tarkoitus on tehdä animaatioiden luonnista helppoa ja käytännöllistä. Käyttäjä pystyy tarkistamaan miltä animaatio näyttää preview-nappia. Hahmo palaa takaisin pääkoordinaatteihin, kun Edit-mode lopetetaan.

6.6.3 Scene-editori

Scene-editorin kautta voidaan vaikuttaa yksittäisen framen sisältöön. Editorin kautta voidaan muokata tekstilaatikoiden tekstiä, vaihtaa taustakuvaa ja hallita äänimaailmaa. Kuten kuvasta 30 näkyy, Scene-editorin testilaatikon sisältö muokkaa automaattisesti Game GUI:n tekstilaatikon sisältöä.



KUVA 30 Scene Editori (Ruutukaappaus)

Scene Editorin kautta voi lisäksi vaikuttaa, minkä tyyppinen frame on kyseessä. Frametyypi vaikuttaa siihen, miten frame toistetaan. Tämänhetkiset frametyypit ovat seuraavat:

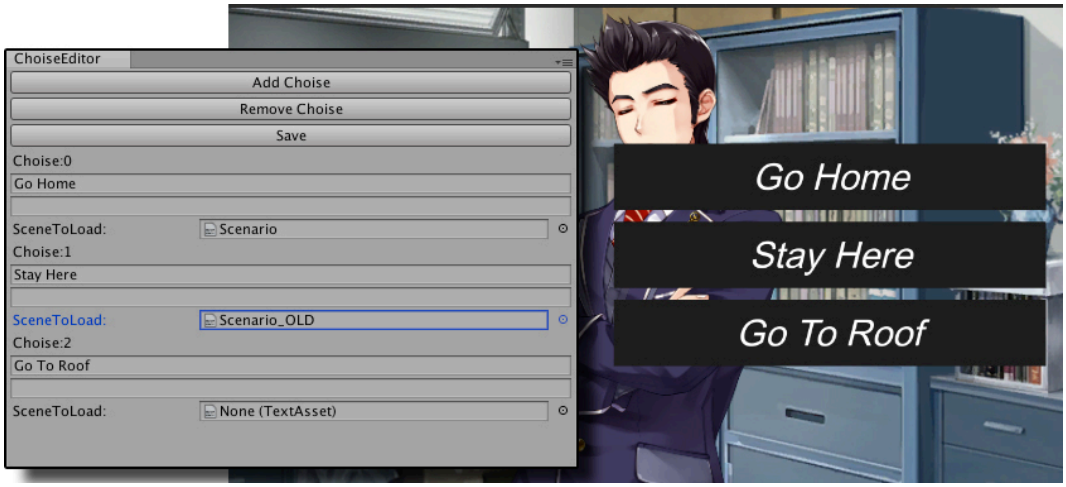
- Normal Frame
- Plus Frame
- End Frame
- If Frame.

Normal Frame on peruskäyttöön tarkoitettu frame. Frame toistuu automaattisesti, kun käyttäjä siirtyy frameen ja siirtyy seuraavaan käyttäjän napinpainalluksella. Plus Framet siirtyvät automaattisesti seuraavaan frameen, käyttäjästä riippumatta. Plus Framet tuovat variaatiota tarinankerrontaan. Plusframejen avulla voidaan esimerkiksi luoda kohtaus, jossa hahmon kasvot muuttuvat kesken lauseen. EndFrame käytetään pelin lopussa. Endframe ohjaa pelaajan automaattisesti takaisin päämenuun. If Framessa voidaan valita seuraavaksi avattava XML-tiedosto, riippuen pelaajan aikaisemmista valinnoista.

6.6.4 Valintaeditori

Valinnat ovat visual novellien ainoa varsinainen pelillinen ominaisuus. Valinta editorin kautta kehittäjä voi luoda frameen valintalaatikoita ja muokata niiden sisältöä. Eri valinnat johtavat erilaiseen tarinaan, mahdollisesti myös erilaiseen lopetukseen.

Kuvassa 31 voi nähdä, mitä valintalaatikot sisältävät. Ylin laatikko on valintalaatikon teksti. Toiseksi ylimpänä oleviin laatikkoihin kirjoitetaan arvo, joka tallennetaan pelin save-dataan, josta peli voi myöhemmin tarkistaa, mitä valintoja pelaaja on tehnyt aikaisemmin. Tällä tavoin mahdollistetaan monihaaraisten tarinoiden luontia. Kolmas laatikko on XML-tiedosto, joka avataan kun pelaaja painaa kyseistä laatikkoa. Jos XML-tiedostoa ei anneta, siirrytään automaattisesti nykyisen XML tiedoston seuraavaan frameen.



KUVA 31. Valintaeditori (Ruutukaappaus)

Luotavien valintalaatikoiden määrä on teoriassa rajaton, mutta näytön koko rajoittaa mahdollisten laatikoiden määrän 4–5 laatikkoon. Valintalaatikoita varten on luotu myös erilaisia siirtymäanimaatioita, jotka toistuvat framen alussa ja lopussa.

6.7 Tiedonkäsittely

Pelieditorit perustuvat käyttäjän itse luoman datan käsittelyyn. Pelieditori on suunniteltu niin, että käyttäjän on helppo lisätä sisältöä peliin, pelkästään siirtämällä tiedostot pelikansioon ja muokkaamalla niitä käyttöliittymässä. Visu10:n käyttäliittymän lukee ja tallentaa tehdyt muutokset XML-tiedostoihin.

6.7.1 XML

XML-data on Visu10:n tärkein datan tallennusmuoto. XML-data sisältää käyttäjän luoman pelin rakenteen. XML:ssä on kunkin framen tiedot, eli minkälaista sisältöä peliin ladataan. Kuva 32 sisältää yhden framen ja kaikki mahdolliset XML-tagit, jotka se voi sisältää. Nodejen käyttötarkoitus on selitetty taulukossa 1.

```
<eventit type="frame">
  <txt>Framen teksti</txt>
  <char>Puhuja</char>
  <voice>Ääni</voice>
  <musa>Taustamusiikki</musa>
  <bgcg>BG_GKOKUJYO_01A_L</bgcg>
  <camera>
    <animationStart>$(CFade(1,2););</animationStart>
  </camera>
  <chdata></chdata>
  <choise>
    <flag>
      <txt>Valinta 1</txt>
      <data>0</data>
      <file>Scenario</file>
    </flag>
    <flag>
      <txt>Valinta 2</txt>
      <data>1</data>
      <file>Scenario2</file>
    </flag>
  </choise>
</eventit>
```

KUVA 32. Framejen XML-rakenne (Ruutukaappaus)

TAULUKKO 1. Framejen XML-nodet

Tagi	**Tarkoitus**
txt	Tekstilaatikon sisältö
char	Puhuja laatikon sisältö
voice	Toistettava äänitiedosto

Tagi	Tarkoitus
musa	Toistettava taustamusiikki, looppaava.
bgcg	Taustakuvan bittikartta
chdata (Character Data)	Sisältää hahmojen data. Tarkemmin alla
camera	Kameran animaatio
choise	Sisältää valintalaatikoiden datan

XML:n chdata-node sisältää tiedot kaikista kyseisen framen hahmoista. Tärkeimmät tiedot ovat ladattavat kuvat, hahmoanimaatiot ja paikkakoordinaatit. Kuvassa 33 on riisuttu versio yhden hahmon XML-tiedostoista. Jokaisen noden käyttötarkoitus selitetään taulukossa 2. Hahmolle on yksinkertaisuuden vuoksi asetettu vain yksi layeri.

```
<chdata>
  <actor>
    <layer>
      <file>presenter</file>
      <crossfade>presenter2</crossfade>
      <x>-234.917755</x>
      <y>-152.486389</y>
      <order>0</order>
      <sx>0.3575651</sx>
      <sy>0.357565</sy>
    </layer>
    <animationStart>$(Move(-992.9091,-151.5086,1);Fade(1,1););</animationStart>
    <animationEnd />
  </actor>
</chdata>
```

KUVA 33. Hahmojen XML-rakenne (Ruutukaappaus)

TAULUKKO 2. Framejen XML-nodet

Node	Tarkoitus
File	Ladattavan kuvan nimi.
Crossfade	Ristihäivytyksessä käytettävä kuva
x,y,sx,sy	Koordinaatit ja skaala, johon hahmo
Order	Hahmon z-syvyys ruudulla.
AnimationStart & AnimationEnd	Scripti-animaatioiden komennot

XML:stä ladattu data käsitellään ja muunnetaan C#-luokaksi. Visu10 generoi pelin aikana hahmot ja muun sisällön luokan tietojen pohjalta. C#-luokan avulla XML-data muunnetaan helposti käsiteltävään muotoon. Esimerkiksi koordinaatit muunnetaan Vector3-luokaksi, jota Unity käyttää koordinaattien määrittämiseen. Valmis C#-luokka tallennetaan C#-listaan, joka pidetään Unityn välimuistissa XML:stä ladatun kohtauksen ajan. Kaikki tarvittavat kuvatiedostot ladataan välimuistiin yhtä aikaa XML-datan kanssa. Tällä tavoin taataan, ettei kuvien lataaminen hidasta pelin toimintaa kesken kohtauksen.

6.7.2 Recourse.Load

Resources.Loadia käytetään lähes kaiken datan lataamiseen. Tähän ainoana poikkeuksena ovat GUI-elementit, joita ei tarvitse ladata dynaamisesti. XML, kuvat, animaatiot ja äänitiedostot ovat sisältöä, jotka ladataan dynaamisesti pelin aikana. Joitain tiedostoja voi pahimmillaan tarvita vain kerran pelin aikana ja kaiken datan lataaminen kerralla olisi hyvin epäkäytännöllistä. Ladattavien tiedostojen nimet haetaan C#-luokasta, joka sisältää ladatun XML:n sisällön.

Recource-kansioon on luotu muutama alikansio. XML-ja Animaatio-kansiot luotiin tiedostorakenteen selventämiseksi. Kansioiden tiedostoreitit on kovakoodattu pelikoodiin, joten XML-tiedostoja ei voi ladata XML-kansion ulkopuolelta.

6.8 Ongelmat

Lähtökohtana visual novellit ovat varsin yksinkertainen peligenre. Kaikki toiminnot ovat yksinkertaisia muihin peligenreihin verrattuna. Ongelmakohdat johtuvat joko epäloogisesta pelikoodista tai Unity-alustan pakottamista rajoitteista.

Yksi iso Unityn asettama rajoite on epälooginen muokkauskäyttöliittymä. Visu10-tekstilaatikot on pakko piirtää Unityn Game GUI:hin, eli pelin kameraan. Ongelmana on, että peliobjektien siirtäminen Game GUI-ikkunan kautta ei ole mahdollista, eli käyttäjän on siirreltävä peliobjekteja Scene-ikkunassa, joka ei

puolestaan näytä GUI-elementtejä. Optimaalinen ratkaisu olisi yksi esikatseluikkuna, jonka kautta peliä voitaisiin editoida. Unityn tämänhetkisen GUI-järjestelmän takia tämä ei kuitenkaan ole mahdollista.

Animaatiotyökalujen toiminta on vielä kyseenalainen. Animaatioiden toteuttamiseen käytetty ITween-plugin toimii joskus hyvin epävakaasti ja jumittaa animaatiot. Tämä voi johtua itse pluginista tai epäloogisesta koodista. Animaatiot jämähtävät paikalleen varsinkin isojen kuvatiedostojen kanssa, joten ongelma voi liittyä myös alustan suorituskykyyn.

Käytettävyyden puolesta pelieditori ei ole vielä julkaisukelpoinen. Pelieditoria luotaessa keskityttiin enemmän käyttöliittymän toiminnollisuuteen kuin käytettävyyteen. Käyttöliittymä ei mahdu kunnolla kaikille näytöille, minkä takia jotkin työkalut jäävät piiloon Mac-kannettavilla. Ongelma on kuitenkin korjattavissa jatkokehityksellä.

7 YHTEENVETO

Opinnäytetyön tarkoituksena oli kartoittaa pelieditorien vaatimuksia ja tutkia miten pelieditorin voi kehittää Unity3d:n sisään. Vertailemalla eri pelieditoreja saatiin hyvä kuva siitä, mistä pelieditoreissa on kyse ja mitä niiden kehittämisessä pitää ottaa huomioon. Tärkeintä on luoda mahdollisimman laaja määrä työkaluja, jotka ottavat huomioon pelinkehitykselle tärkeät toiminnot. Kehityksessä pitää ottaa huomioon, mitä erityyppiset käyttäjät vaativat editorilta ja miten nämä tarpeet saadaan täytettyä. Ei ole tarpeeksi, että editorissa on animaatiotyökalut, vaan niiden pitää olla myös helppokäyttöisiä.

Pelieditorin ovat hyvin monimutkaisia työkaluja. Pelieditorien kehittämisessä pitää ottaa huomioon käyttäjien tarpeet ja kohdealustojen vaatimukset. Unity3D tarjosi editorin kehittämiselle vahvan pohjan. Valmiin kehitysympäristön ja pelimoottorin ansioista pelieditorin kehityksessä voitiin keskittyä pelieditoreille oleellisiin toimintoihin.

Pelieditorin rakentaminen Unityn sisään oli sopiva haaste. Valmiit 2d- ja GUI-työkalut olivat isoin apu pelieditorin kehittämiseen. Isoin osa työstä kului juuri Unityn käyttöliittymän tutkimiseen ja rakentamiseen. Datan lukeminen ja muokkaamien käyttöliittymän kautta vaati perehtymistä sekä C#-koodaukseen, Javascript-koodaukseen että UnityGUI-koodaukseen. Lopputuloksena saatiin toimiva kokonaisuus, jonka kautta erilaisen datan muokkaaminen onnistuu vaivattomasti.

Datarakenteen suunnittelu vaati perehtymistä sekä XML-rakenteeseen että C#-olio-ohjelmointiin. Alusta kehittyi ajan kanssa sekavasta koodirykelmästä monipuoleisempaan tietorakenteeseen, jossa käytettiin hyväksi esikoodattuja C#-luokkia datan jäsentelyyn. Datan jäsentelyä käytetään hyväksi kaiken tyyppisissä peleissä. Pelieditorin kehittäminen antoi siis erinomaisen kuvan siitä, miten pelit toimivat ja mitä niiden kehittämisen taustalla tapahtuu.

LÄHTEET

Painetut lähteet:

Ahearn L. 2001. Game Art Elements F/X & Design. Kanada: Coriolis Group Books.

Blackman S. 2011. Beginning 3d Game Development. USA:Apress.

Chu P. 2013. Learn Unity 4 for IOS. USA:Apress.

Doran J. P. 2013. Mastering UDK Game Development. Iso-Britannia: Packt Publishing Ltd.

Duggan M. 2011 .RPG Maker for Teens. Iso- Britannia: Cengage Learning.

Elliott J. 2013. HTML5 Game Development with GameMaker. Iso-Britannia: Packt Publishing Ltd.

Harbour J. & Smith J. 2006. DarkBasic Pro Game Progamming (2nd Edition). Iso- Britannia: Cengage Learning.

Harbour J. 2011a. Visual C# Game Programming for Teens. Iso- Britannia: Cengage Learning.

Harbour J. 2011b. XNA® Game Studio 4.0 for Xbox 360® Developers. USA: Course Technology.

Matloff N. & Salzman P. 2008. Art Of Debugging. USA: No Starch Press.

McShaffry M. & Graham D. 2012 Game Coding Complete, Fourth Edition. Iso-Britannia: Cengage Learning.

McShaffry M. 2009. Game Coding Complete. Iso- Britannia: Cengage Learning.

Menard M. 2011. Game Development with Unity. USA: Course Technology.

Moghadampour G. 2009. C#-ohjelmointi. Jyväskylä: WSOYPro Oy

Pardew L. 2006. Game Character Animation All in One. Iso- Britannia: Cengage Learning.

Pierce G. 2012. Unity iOS Game Development. Iso-Britannia: Packt Publishing Ltd.

Smith M. & Queiroz C. 2013. Unity 4.x Cookbook. Iso-Britannia: Packt Publishing Ltd.

Tracy S. & Reindell P. 2012. CryENGINE 3 Game Development : Beginner's Guide. Iso-Britannia: Packt Publishing Ltd.

Voyles D. 2013. UnrealScript Game Programming Cookbook. Iso-Britannia: Packt Publishing Ltd.

Wihlidal G. 2006. Game Engine Toolset Development. USA: Course

Sähköiset lähteet:

Bethesda Softworks LLC. 2013. Creation Wiki [viitattu 26.3.2014]. Saatavissa: http://www.creationkit.com/Creation_Kit

Guldbrandsen K. & Storstein K. 2010. Evolutionary Game Prototyping using the Unreal Development Kit [viitattu 26.3.2013] Yliopisto-Opinnäytetyö. Saatavissa: http://www.diva-portal.org/smash/get/diva2:356726/FULLTEXT01.pdf

Quijano J. 2013. On RPG Maker and Game Engines [viitattu 26.3.2013]. Saatavissa: http://johansenquijano.wordpress.com/2013/09/21/on-rpg-maker-and-game-engines/

Scirra LTD. 2014. Construct 2 [viitattu 26.3.2013]. Saatavissa: https://www.scirra.com/construct2

Unity Technologies. 2011. Unity Reference Manual. [viitattu 26.3.2014]. Saatavissa: http://docs.unity3d.com/Documentation/Components/index.html

Unity Technologies 2013a. Unity – Fun Facts [viitattu 28.3.2014] Saatavissa: http://unity3d.com/company/public-relations/

Unity Technologies. 2013b. Unity Manual [viitattu 26.3.2014]. Saatavissa: http://docs.unity3d.com/Documentation/printable.html

Unity Technologies. 2013c. What New In Unity 4.3 [viitattu 26.3.2014]. Saatavissa: http://unity3d.com/unity/whats-new/unity-4.3

Unity Technologies. 2014. Unity Scripting Reference. [viitattu 26.3.2014]. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/index.html

Watters A. 2010. Tips for Writing Good Documentation [viitattu 26.3.2014]. Saatavissa: http://readwrite.com/2010/08/14/tips-for-writing-good-document#awesm=~ozBlrEJYMs0ksv

Wikipedia. 2014a. Game creation system [viitattu 26.3.2014]. Saatavissa: http://en.wikipedia.org/wiki/Game_creation_system

Wikipedia. 2014b. RPG Maker [viitattu 26.3.2014]. Saatavissa: http://en.wikipedia.org/wiki/RPG_Maker

Wikipedia. 2014c. List of Unreal Engine games [viitattu 26.3.2014]. Saatavissa: http://en.wikipedia.org/wiki/List_of_Unreal_Engine_games

Wikipedia. 2014d. Visual Novel [viitattu 26.3.2014]. Saatavissa: http://en.wikipedia.org/wiki/Visual_novel

YoYo Games Ltd . 2014. GameMaker: Studio [viitattu 26.3.2014]. Saatavissa: https://www.yoyogames.com/studio/

Kuvalähteet:

Kuva 1 Ruutukaappaus, Unity-ohjelmasta

Kuva 2-3 Ruutukaappaus, Construct 2-ohjelmasta

Kuva 4. Enterbrain Inc. 2014. RPG-Maker. Saatavissa osoitteesta http://www.rpgmakerweb.com/

Kuva 5 Ruutukaappaus, GameMaker Studio -ohjelmasta

Kuvat 6-7. Epic Games Inc. Unreal Editor Manual. Saatavissa osoitteesta: https://docs.unrealengine.com/latest/INT/Engine/index.html

Kuva 8. IGN. 2013. How the Modders Conquered Skyrim. Saatavissa osoitteesta: http://ap.ign.com/en/feature/8157/how-the-modders-conquered-skyrim

Kuva 9 Ruutukaappaus, Unreal Editor – ohjelmasta

Kuva 10. Unity Technologies. 2013. Animation. Saatavissa osoitteesta: http://download-cdn.unity3d.com/unity/animation/

Kuva 11. Roberts J. 2010. Saatavissa osoitteesta: http://www.flickr.com/photos/goosemouse/4998615328/

Kuva 12 Ruutukaappaus, Unreal Editor – ohjelmasta

Kuva 13 Ruutukaappaus, RPG-Maker – ohjelmasta

Kuva 14. Wikipedia. 2014. UnrealEd. Saatavissa osoitteesta: http://en.wikipedia.org/wiki/UnrealEd

Kuva 15. Unity Technologies. 2013. Editor. Saatavissa osoitteesta: https://docs.unity3d.com/Documentation/ScriptReference/Editor.html

Kuva 16. Unity Technologies. 2013. Inspector. Saatavissa osoitteesta: https://docs.unity3d.com/Documentation/Manual/Inspector.html

Kuvat 17-18 Photoshopissa muokattu ruutukaappaus, Unity-ohjelmasta

Kuva 19. Unity Technologies. 2013. EditorWindows. Saatavissa osoitteesta: http://docs.unity3d.com/Documentation/Components/editor-EditorWindows.html

Kuva 20. Unity Technologies. 2013. AssetBundlesIntro. Saatavissa osoitteesta: https://docs.unity3d.com/Documentation/Manual/AssetBundlesIntro.html

Kuva 21 Ruutukaappaus, Notepad++-ohjelmasta

Kuva 22. Vestal A. 2013. Love Week: Don’t Take it Personally. Saatavissa osoitteesta: http://www.thegia.com/2013/09/10/love-week-dont-take-it-personally/

Kuva 23. Tøvik J. E. 2013. Fire Emblem: Awakening – Sorta Review. Saatavissa osoitteesta: http://varewulf.wordpress.com/2013/08/06/fire-emblem-awakening-sorta-review/

Kuva 24. CB Interavtive Inc. 2007. Koihime + Musou. Saatavissa osoitteesta: http://www.gamefaqs.com/pc/935937-koihime-musou/images/screen-2

Kuvat 25-26. Soramani Kagome -blogi. 2013. Remurusu. Muokattu, alkuperäinen saatavissa: http://soramani.kagome-kagome.com

Kuva 27 Ruutukaappaus, Unity-ohjelmasta

Kuvat 27–32 Photoshopissa muokattu ruutukaappaus, Unity-ohjelmasta

Kuva 33 Ruutukaappaus, Notepad++-ohjelmasta