OULUN SEUDUN
AMMATTIKORKEAKOULU

Erja Manninen

THE LEGEND OF ZELDA -PELI UNITY3D-YMPÄRISTÖSSÄ

THE LEGEND OF ZELDA -PELI UNITY3D-YMPÄRISTÖSSÄ

Erja Manninen
Opinnäytetyö
Kevät 2013
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistokehitys

Tekijä(t): Erja Manninen
Opinnäytetyön nimi: The Legend of Zelda -peli Unity3D-ympäristössä
Työn ohjaaja(t): Eero Nousiainen
Työn valmistumislukukausi ja -vuosi: Kevät 2013 Sivumäärä: 104 + 8 liitettä

Tämän työn aiheeksi valittiin The Legend of Zelda -pelisarjan ensimmäisen pelin tekeminen uuteen versioon, mahdollisimman tarkasti alkuperäistä pelin logiikkaa jäljitellen. Työ rajattiin pelin ensimmäisen luolaston toteuttamiseen. Tämän lisäksi pelissä on aloitus- ja lopetusruudut.

Tämän pelin tekemisellä ei tavoiteltu minkäänlaista taloudellista hyötyä, vaan tavoitteena oli oppia pelinkehitysprosessin eri osa-alueista mahdollisimman monipuolisesti. Kun työn rungoksi valittiin jo olemassa oleva peli, suunnittelutyön osuus jäi pienemmäksi, ja erilaisten työkalujen käytön opettelu ja pelin varsinainen toteutus olivat päätarkoituksena.

Peli toteutettiin käyttämällä Unity3D-pelinkehitystyökalua. 3D-mallit ja animaatiot tehtiin Blender-mallinnusohjelmalla. Pelin 2D-grafiikka toteutettiin Gimp-kuvankäsittelyohjelmalla.

Kaikki määritetyt tavoitteet täyttyivät ja peli onnistui yli odostusten, vaikka rima oli melko korkealla. Hahmojen animaatioissa, kamerassa, vihollisten tekoälyssä ja monissa muissa asioissa on silti vielä parannettavaa. Koska tässä työssä tehty peli on vain alkuperäisen pelin ensimmäinen luolasto, pelin toteutusta voidaan jatkaa vielä pitkälle. Osa pelin koodiarkkitehtuurin ratkaisuista onkin tehty tämä mahdollisuus huomioon ottaen.

Asiasanat: Unity3D, peliohjelmointi, pelinkehitys, C#, Blender, 3D-mallinnus, videopelit, Nintendo

SISÄLLYS

1 JOHDANTO

Olen pelannut koko ikäni konsolipelejä, ja kun aloitin opiskelemaan ohjelmistokehitystä, aloin kiinnostua myös pelien tekemisestä. Olin käyttänyt hieman Unity3D-pelinkehitystyökalua, ja mielessäni kävi joskus, että olisi hauskaa tehdä uusi versio The Legend of Zelda -pelisarjan ensimmäisestä pelistä. Tämä pelisarja on minulle yksi rakkaimmista. Kun tuli aika valita opinnäytetyön aihetta, kysyin, olisiko tälläinen aihe lainkaan sopiva opinnäytetyöksi, ja onnekseni se hyväksyttiin.

Alkuperäinen peli on melko pitkä, joten siitä täytyi rajata vain pieni osa toteutukseen. Päädyin lopulta pelin ensimmäisen luolaston toteuttamiseen. Tavoitteena oli, että luolaston voi pelata läpi ja sieltä löytyisi mahdollisimman paljon alkuperäisessä pelissä nähtäviä ominaisuuksia uuteen versioon muokattuna.

Työssä tutustutaan pelinkehittämiseen ja peliohjelmointiin Unity3D-pelinkehitystyökalulla sekä 3D-mallintamiseen, animointiin ja teksturoimiseen Blender-mallinnusohjelmalla. Tarvittavan 2D-grafiikan piirtäminen ja kuvien muokkaus tehdään Gimp-kuvankäsittelyohjelmalla.

2 TAUSTAA

2.1 The Legend Of Zelda

The Legend of Zelda on Nintendon tuottama ja kehittämä pelisarja, jonka luojia ovat japanilaiset pelisuunnittelijat Shigeru Miyamoto ja Takashi Tezuka. Zelda-sarja täytti 25 vuotta vuonna 2011 (kuva 1). Sarjaan kuuluu tällä hetkellä 16 peliä eri Nintendon konsoleilla. Mukaan ei laskettu vanhemmista Zelda-peleistä tehtyjä versioita uudemmille konsoleille. (1.)

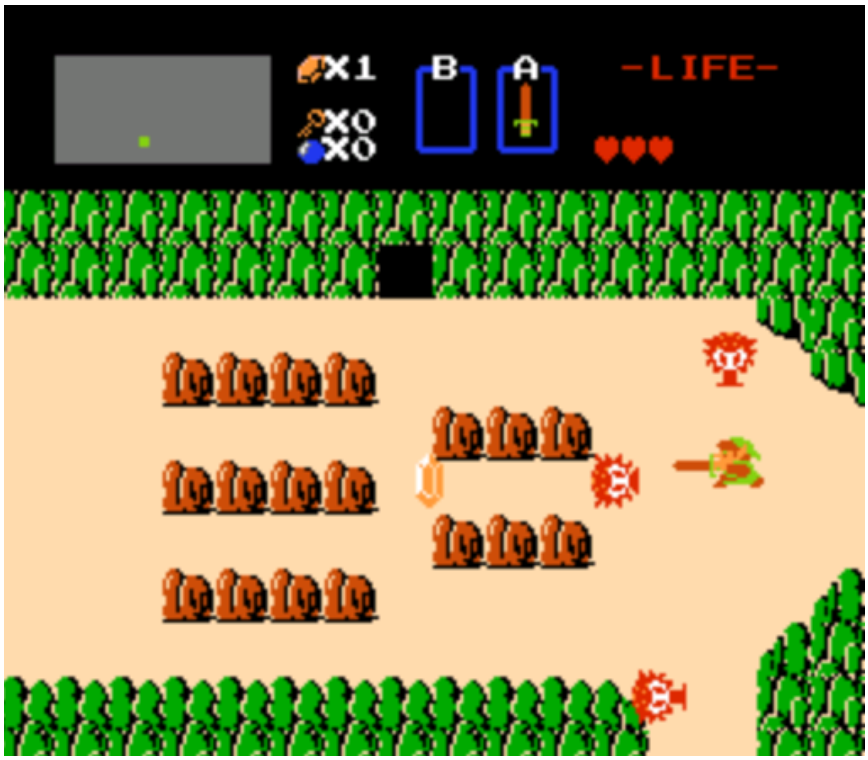
KUVA 1. Pelisarjan 25-vuotisjuhlakuva (2)

The Legend of Zelda on samaa nimeä kantavan pelisarjan ensimmäinen osa. Se julkaistiin vuonna 1986 Nintendo Entertainment Systemille (kuva 2). Peli oli aikanaan mullistava kokemus avoimen pelimaailmansa ansiosta, ja peli on tullut tunnetuksi juuri laajasta maailmastaan, joka oli sen ajan peleille epätyypillistä. Miyamoto halusi tuoda peliin nimenomaan tutkimisen riemua. The Legend of Zelda oli ensimmäinen uuden sukupolven pelikonsolipeli, jota myytiin yli

miljoona kappaletta. Peliä myytiin maailmanlaajuisesti yil 6,5 miljoonaa kappaletta, ja oli Nintendon johdon odotuksien vastaisesti suurmenestys. The Legend of Zelda on lisäksi sijoittunut lukuisille listoille, jotka ovat kartoittaneet parhainta ikinä julkaistua peliä. Peliä varten kehitettiin paristovarmennettu tallennus, joka mahdollistaa pelitilanteen tallentamisen suoraan pelikasetille. (3.)

2.2 Työn esittely ja rajaus

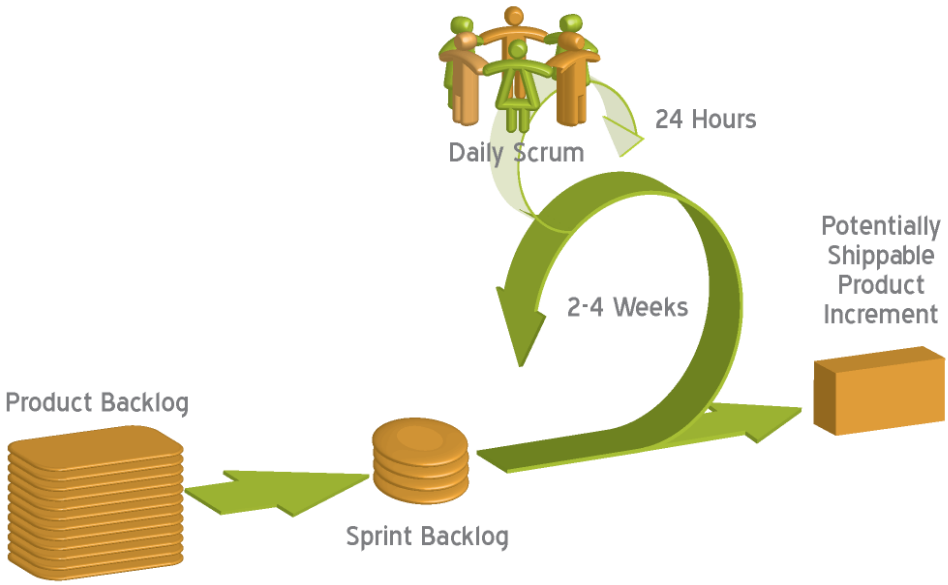
Aihe rajattiin pelin ensimmäiseen luolastoon. Tavoitteena oli, että luolaston voi pelata läpi. Pelihahmolla on alussa miekka ja kilpi ja luolastosta löytyy muutamia lisävarusteita, kuten pommeja, bumerangi ja jousipyssy. Lisäksi luolastosta voi löytää kartan ja kompassin, jotka helpottavat navigointia. Kulkua on rajoitettu lukituilla ovilla. Osa näistä aukeaa avaimilla ja osa päihittämällä kaikki huoneessa olevat viholliset. Tuhotut viholliset saattavat tiputtaa esineitä, kuten rahaa, pommeja ja sydämiä. Luolaston lopussa on suurempi lohikäärmevastus, Aquamentus, jonka päihittämisestä saa yhden lisäsydämen sekä tietenkin ensimmäisen Kolmivoiman palan.



KUVA 2. Kuva ensimmäisestä The Legend of Zelda -pelistä (4)

3 SCRUM-TUOTEKEHITYSMALLI

Scrum on projektinhallinnan viitekehys, jota käytetään yleisesti ketterässä ohjelmistokehityksessä ja projektinhallinnassa. Sen tärkeimpiä ominaisuuksia ovat toimivan ohjelmiston ensisijaisuus, suora viestintä ja nopea muutoksiin reagointi, joka on tärkeämpää kuin suunnitelman noudattaminen. Scrumissa työskennellään toistavasti ja iteratiivisesti riskien kontrolloimiseksi ja seurataan koko ajan työn edistymistä tiimin kesken. Riskejä pyritään minimoimaan jakamalla ohjelmistokehitys lyhyisiin iteraatioihin, jotka kestävät tyypillisesti yhdestä neljään viikkoa (kuva 3). (5.)



KUVA 3. Scrum-menetelmän periaate (6)

3.1 Roolit

Scrum-tiimissä on kolme erilaista roolia: tuotteen omistaja (Product Owner), Scrum-mestari (Scrum Master) ja kehitystiimi (Development Team). Tuotteen omistaja on asiakkaan edustaja, joka vastaa siitä, että lopputuotteen vaatimukset ja niiden tärkeysjärjestys on selvillä koko ajan ja koko tiimille. Hän vastaa tuotteen ominaisuuksista ja toiminnallisuudesta. Scrum-menetelmässä tuotteen vaatimuksia voidaan lisätä, muuttaa tai poistaa jokaisen iteraatiojakson jälkeen. (5.)

Scrum-mestarin tehtävänä on valvoa, että Scrum-menetelmää noudatetaan oikein, mutta hänellä ei varsinaisesti ole käskyvaltaa itseohjautuvaan kehitystiimiin. Hän varmistaa, että tiimillä on työskentelyrauha, ja hän johtaa päivittäiset aamupalaverit. Scrum-mestari ei ole perinteinen projektipäällikkö, vaan tehtävä on lähempänä toiminnan kehittäjän tai laatupäällikön toimenkuvaa. (5.)

Kehitystiimin tehtävä on toteuttaa itsenäisesti ja tasa-arvoisesti projektin tavoitteita. Tiimillä on vastuu työn tuloksista ja laadusta, eli kaikkien täytyy seistä myös toisten tiimiläisten tuotosten takana. (5.)

3.2 Työskentelytavat

Scrumissa, kuten kaikissa muissakin ketterissä malleissa, kehitys rakentuu erimittaisten syklien ympärille. Tavoitteena oleva lopputuote rakentuu useiden kehitysjaksojen aikana pikkuhiljaa täydellisemmäksi ja valmiimmaksi. (7.)

Scrumin tärkeimmät syklit ovat sprintti ja päivä. Sprintti on itsenäinen kehitysjakso, joka kestää yleensä 1–4 viikkoa, maksimissaan 30 päivää. Tämä on sellainen aika, jossa tiimin on ehdittävä toteuttaa jotain merkittävää, joka on julkaistavissa olevaa laatua. (5.)

Ennen kuin koko prosessi voidaan aloittaa, täytyy luoda visio siitä, mikä on koko projektin tarkoitus. Vasta tämän jälkeen luodaan niin sanottu työlista eli listaus niistä ominaisuuksista, joita lopputuotteelta vaaditaan. Jokaisen sprintin sisältö sovitaan sprintin suunnittelupalaverissa ennen sprintin aloitusta, ja toteutettaviksi valitaan sellaisia tuotteen kehitysjonon kohtia, joilla on sillä hetkellä suurin merkitys projektin onnistumiselle. Tuotteen ominaisuuslistan tulee sisältää kaikki ne vaatimukset, jotka kohdistuvat lopputuotteeseen. Ominaisuuslistan työvaiheet täytyy osata priorisoida niin, että ominaisuudet on esitetty tärkeysjärjestyksessä. (5.)

Sprintti alkaa suunnittelupalaverilla, joka kestää maksimissaan kahdeksan tuntia. Palaverissa päätetään yhdessä, mitä kyseisen sprintin aikana tullaan tekemään. Käytännössä tämä tarkoittaa sitä, että tuotteen omistaja valitsee listan kärkipäästä tärkeimpiä tehtäviä ja tiimi itse arvioi, kuinka paljon se ehtii

tulevan sprintin aikana saada valmiiksi. Sprintille asetetaan siis yhdessä tavoite, jonka saavuttamista sprintin loppuarvioinnissa mitataan. Sprintin aikana näitä tavoitteita ei voida muuttaa, mutta tiimillä on täysi vapaus päättää, kuinka ja missä järjestyksessä se annetut tehtävät hoitaa. (5.)

Joka aamu tiimi kokoontuu lyhyeen päiväpalaveriin, jossa jokainen kertoo lyhyesti tilanteensa edellispäivän tuloksista ja tulevan päivän tavoitteistaan. Palaveriin osallistuvat kaikki tiimin jäsenet ja Scrum-mestari. Palaveriin saa tulla myös muita projektista kiinnostuneita. Vain tiimin jäsenet saavat kuitenkin puhua. Tapahtuman tarkoitus on tarjota kaikille tieto siitä, missä projekti menee ja mitä ongelmia sillä on. Näin voidaan tarjota myös apua sitä tarvitseville ja miettiä yhdessä ratkaisuja ongelmiin palaverin jälkeen. Tällä ennaltaehkäistään myös sitä, että useampi tiimiläinen tekee samaa vaihetta toisistaan tietämättä. Jotta päiväpalaveri ei pääsisi venymään, se pidetään aina seisten. Tällä pyritään siihen, että palaveri sujuu joutuisasti ja kukin tiimiläinen keskittyy asiaan. (8.)

Syklin viimeisenä lenkkinä on sprintin arviointipalaveri, jossa Scrum-mestari ja tiimi arvioivat yhdessä kuluneen sprintin onnistumista. Jokaisen sprintin on tuotettava toimivaa ohjelmakoodia. Esittelyvaiheessa ainoastaan täysin valmiit ja toimivat ohjelmaversiot esitellään. Jos joku vaihe on jäänyt kesken tai se ei ole läpäissyt testausta, siirretään se seuraavalle työlistalle. Mikäli työmenetelmissä tai prosessin kulussa on ilmennyt jotain sellaista, jota pitäisi muuttaa, aiheesta keskustellaan ja tehdään mahdolliset muutokset seuraavaan sprinttiin. Sprintin jälkitarkastelun jälkeen kierros alkaa tarvittaessa alusta uuden sprintin suunnittelulla. (5.)

3.3 Dokumentit

Scrum-menetelmän mukaisessa tuotekehityksessä käytettäviä dokumentteja ovat julkaisusuunnitelma (Release Plan), kaikki tuotteen ominaisuudet (All Product Items), tuotteen ominaisuuslista (Product Backlog) ja sprinttien tehtävälista (Sprint BackLog). Julkaisusuunnitelmaan kirjataan ylös projektin käytössä olevat resurssit, kuten kuinka monta jäsentä tiimissä on ja kuinka monta työviikkoa ja työtuntia per viikko on käytettävissä projektin toteutusta

varten yhteensä (liitteet 1–2). Aika jaetaan julkaisusuunnitelmassa pienempiin osiin eli sprintteihin. Suunnitelmaan kirjataan ylös myös projektin tavoite. Tämä dokumentti on siis projektin aikataulu julkaisua varten. (9.)

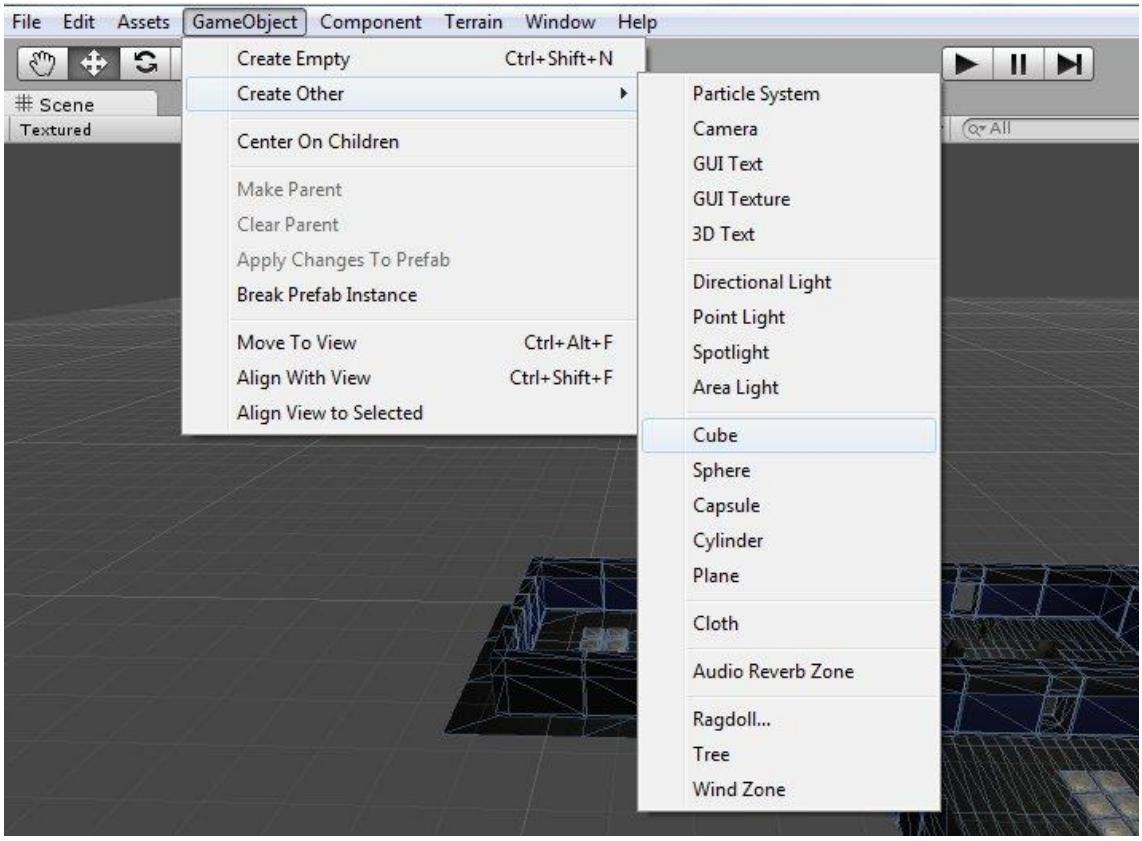
Tämän jälkeen kaikki lopulliseen tuotteeseen tulevat ominaisuudet kirjataan ylös kaikki tuotteen ominaisuudet -dokumenttiin (liite 3). Ominaisuuksille määritetään prioriteetti kirjoittamalla ne valittuun sarakkeeseen. Kun tämä dokumentti on valmis, siitä poimitaan ne tehtävät, jotka tuntuvat järkevältä kokonaisuudelta yhtä sprinttiä varten ja ajatellen, että nämä ehditään toteuttaa määritetyssä ajassa. Nämä valitut ominaisuudet siirretään tuotteen ominaisuuslistaan (liite 4). Ominaisuudet jaotellaan sprinteittäin ja niille määritetään pisteinä (storypoints) arvioitu toteutuksen kesto eli työmäärä, joka kuluu kunkin ominaisuuden toteuttamiseen. Lopuksi nämä samat tehtävät siirretään sprintin tehtävälistaan, jossa jokainen niistä jaetaan vielä pienempiin osasiin (liitteet 5–8). Dokumenttiin merkitään tunteina, kuinka paljon aikaa on käytetty kuhunkin tehtävään ja kuka tehtävää on ollut tekemässä. (9.)

4 TYÖVÄLINEET

4.1 Unity3D-pelinkehitystyökalu

Unity3D on pelimoottori ja pelinkehitystyökalu, jonka kehitys aloitettiin vuonna 2004. Vuonna 2009 Unitysta julkaistiin maksullisen version lisäksi ilmainen versio, joka kasvatti käyttäjien lukumäärää nopeasti. Huhtikuussa 2012 rekisteröityneitä käyttäjiä oli noin 1,3 miljoonaa. Unityn ilmaisversiolla voidaan julkaista pelejä PC:lle, Macille ja webbiin. (10.)

Käytännössä pelinkehitys Unitylla tapahtuu luomalla projekti ja skene. Projektin skenessä täytyy olla ainakin kamera ja valo, koska ilman näitä pelaaja ei näe ruudulla mitään. Skeneen voidaan luoda valikosta valitsemalla erilaisia peliobjekteja (kuva 4) ja viemällä projektiin 3D-malleja, jotka on tehty erillisellä mallinnusohjelmalla. Objekteihin liitetään ohjelmakoodia sisältäviä tiedostoja eli skriptejä (script). (11.)



KUVA 4. Valikosta voidaan luoda erilaisia peliobjekteja skenenäkymään.

4.1.1 Käyttöliittymä

Unityn käyttöliittymä koostuu viidestä osasta: pelinäkymästä (Game view) ja skenenäkymästä (Scene view) sekä kolmesta paneelista (kuva 5). Pelinäkymä ja skenenäkymä ovat pelin ulkoasun rakentamista, tarkastelua ja testaamista varten. Skenenäkymä on 3D-avaruus, johon itse peli rakennetaan. Peliobjekteja, joista peli koostuu, siirrellään, käännetään tai skaalataan erikokoiseksi skenenäkymässä. Pelinäkymästä näkee, mitä pelaaja tulee näkemään eli, mitä pelissä oleva kamera kuvaa. Pelinäkymää varten käyttöliittymän yläosassa on Play-painike, jota painamalla peli siirtyy pelitilaan ja peliä voi pelata, mikäli ohjelmakoodissa ei ole käännösvirheitä. (12.)

Hierarkiapaneeli

Hierarkiapaneelista (Hierarchy panel) näkyvät kaikki aktiivisen skenen objektit eli sen skenen, joka on sillä hetkellä näkyvillä skenenäkymässä. Objekteja voidaan nimetä uudestaan, kopioida tai tuhota sekä liittää niihin materiaaleja, skriptejä tai komponentteja tämän paneelin kautta. Valittuna olevan objektin tarkemmat tiedot näkyvät tarkastelupaneelissa. (13.)

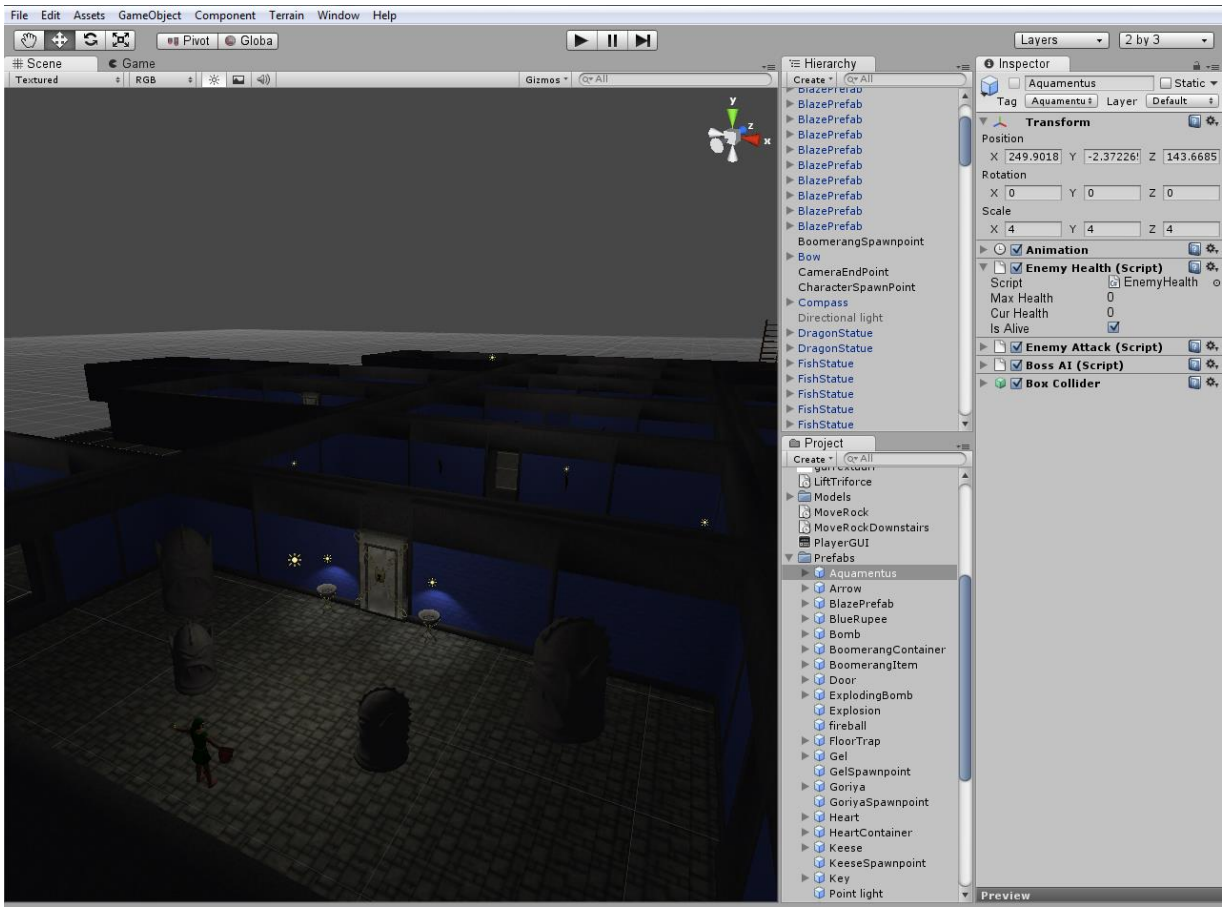
Tarkastelupaneeli

Tarkastelupaneelissa (Inspector panel) näkyy hierarkiapaneelista valitun objektin tarkemmat tiedot, kuten liitetyt komponentit, tagi, materiaalit, tekstuurit ja skriptit. Näitä komponentteja voi myös poistaa objektista tämän paneelin kautta. Mikäli skripteissä on määritetty public-muuttujia, niiden nimet ja asetetut arvot näkyvät tarkastelupaneelissa. Tämä on kätevää jos halutaan nähdä peliä pelatessa, mitä koodissa määriteltyjen muuttujien arvoille tapahtuu. Muuttujille voidaan asettaa arvoja myös suoraan tarkastelupaneelissa. Täytyy muistaa, että tämä ylikirjoittaa kaikki mahdollisesti skriptissä asetetut arvot ja saattaa aiheuttaa hämmennystä ja ei-toivottuja muutoksia. (14.)

Projektipaneeli

Projektipaneelissa (Project panel) näkyvät kaikki projektiin liitetyt osat, kuten skenet, ohjelmakooditiedostot eli skriptit, materiaalit, tekstuurit (2D-grafiikka),

3D-mallit, prefabit ja äänitiedostot. Se on ikään kuin varasto, josta otetaan käyttöön tarvittavia komponentteja siirtämällä niitä hiirellä esimerkiksi hierarkiapaneelissa näkyviin objekteihin. Projektipaneelissa voidaan myös luoda uusia materiaaleja, prefabeja ja skriptejä. (15.)



KUVA 5. Unityn käyttöliittymä. Projektipaneelista valitun prefabin tarkemmat tiedot näkyvät tarkastelupaneelissa oikealla.

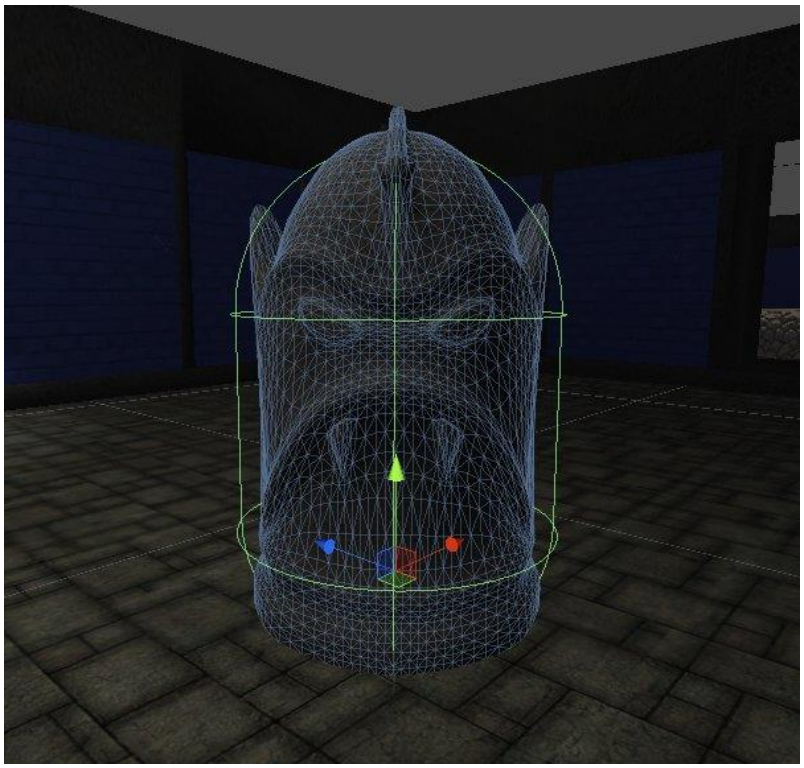
4.1.2 Unityn ominaisuuksia

Peliobjektit ja Mesh

Jokainen objekti pelissä on peliobjekti. Peliobjektit eivät ole yksinään varsinaisesti mitään. Ne tarvitsevat erityisominaisuuksia tullakseen hahmoksi, ympäristöksi tai erikoisefektiksi. Peliobjektit ovat tavallaan säiliöitä, jotka täytetään erilaisilla komponenteilla. Näin niistä muodostuu jokin pelissä oleva asia, kuten vaikka pelihahmo. Komponenteilla voidaan saada aikaan esimerkiksi painovoima tai törmääminen toisiin objekteihin. Tämän lisäksi

objekteihin voidaan liittää skriptejä, jotta niiden välille saadaan aikaan haluttua toiminnallisuutta ja vuorovaikutusta. (16.)

Peliobjektilla on Mesh filter, joka kertoo minkä muotoinen se on. Mesh on tavallaan aine, josta peliobjekti koostuu (kuva 6). Mesh renderer piirtää tämän aineen näkyväksi, jotta pelaaja voi nähdä sen. Jos Mesh poistetaan peliobjektista, objekti on edelleen olemassa, mutta sitä ei voida piirtää. (17.)



KUVA 6. Pelissä sijaitseva peliobjekti. Vaaleansininen verkko on peliobjektin Mesh. Vaaleanvihreä viiva on samaan peliobjektiin liitetty collider-komponentti.

Collider-komponentti

Colliderin avulla tunnistetaan, milloin objekti osuu johonkin. Collider on näkymätön muoto, joka kertoo pelimoottorille, milloin kaksi peliobjektia koskettaa toisiaan eli tapahtuu törmäys. Collidereilla rajataan liikkumista niin, etteivät objektit pysty liikkumaan toistensa läpi tai pelialueen ulkopuolelle. Colliderilla pystytään tekemään myös toiminnallisuutta halutulle alueelle. (18.)

Tämä tapahtuu asettamalla colliderin IsTrigger-ominaisuus arvoon true. Tällöin collider ei enää estä pelihahmon kulkua, vaan se kutsuu OnTriggerEnter-funktiota, kun se havaitsee toisen colliderin ylittävän rajan. (19.)

Näin voidaan tutkia tarkempia tietoja objektista, joka aiheutti funktion kutsun eli esimerkiksi, oliko rajan ylittänyt objekti pelihahmo.

```
void OnTriggerEnter(Collider collider)
if(collider.tag == "Player"){
    //tehdään jotakin
}
```

Audio-komponentit

Äänien käyttöön tarvitaan aina AudioListener, jolla ääni kuullaan, ja AudioSource eli äänilähde. AudioListenereitä voi olla skenessä vain yksi, mutta AudioSourceja voi olla useita. AudioSource voi soittaa vain yhtä ääntä kerrallaan. AudioSourcessa voidaan määrittää, onko ääni 2D- vai 3D-ääni. 3D-äänissä tunnistetaan AudioSourcen ja AudioListenerin välimatka sekä suunta, josta ääni kuuluu. Kaukana olevat äänet siis kuuluvat hiljempaa ja lähempänä olevat kovempaa. 2D-äänet kuuluvat tasaisesti kaikkialta, eikä äänen tulosuuntaa voida havaita. (20.)

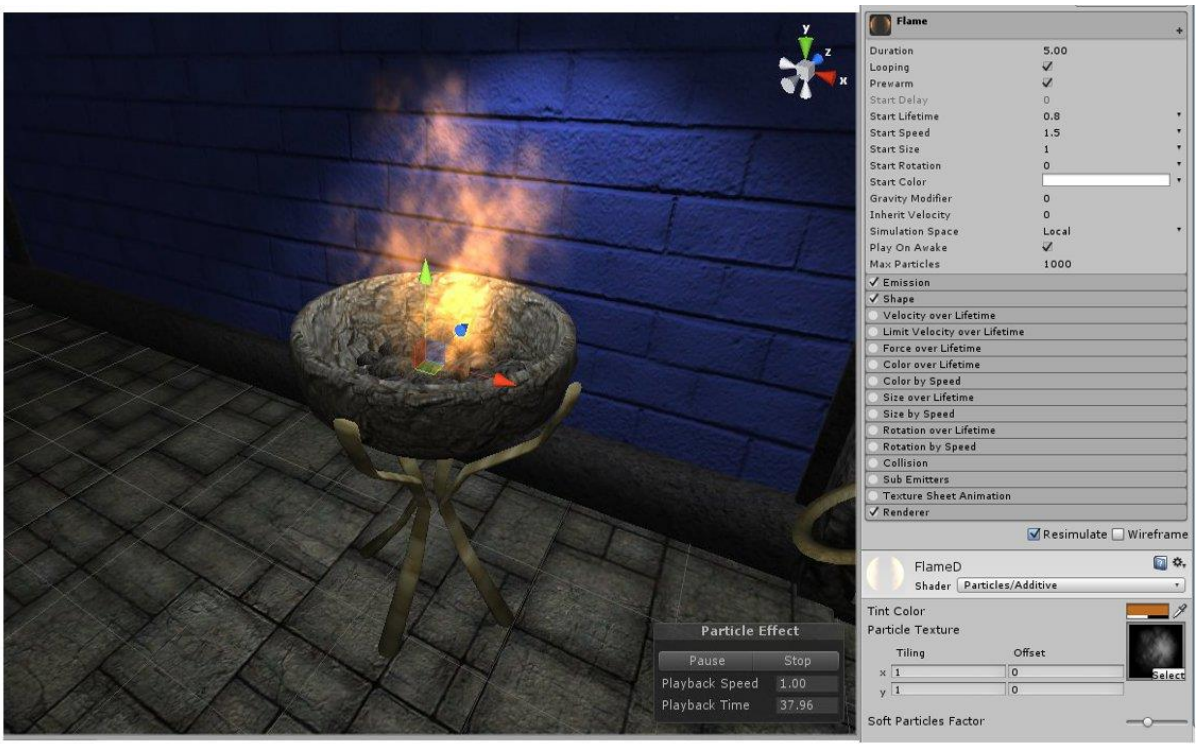
Prefab-objektikokonaisuudet

Erittäin kätevä ominaisuus Unityssa on prefabit. Prefab on kokonaisuus, joka koostuu 3D-mallista, skripteistä ja muista komponenteista. Esimerkkinä kootaan pelin vihollisesta prefab. Vihollisen 3D-malliin lisätään kaikki skriptit, joita se tarvitsee toimiakseen. Siihen lisätään collider eli raja joka kertoo, milloin objekti koskettaa toista colliderıa. Kun vihollinen on toimintavalmis, siitä tehdään prefab. Nyt tästä prefabista voidaan luoda kopioita, jotka ovat täysin identtisiä alkuperäisen kanssa ja suoraan toimintavalmiita. Prefabien avulla säästää siis erittäin paljon aikaa. Jos kokonaisuutta halutaan myöhemmin muuttaa, muutoksen voi tehdä prefabiin ja se muuttaa automaattisesti kaikkia siitä luotuja objekteja.

Tämän lisäksi prefabeista pystytään luomaan ohjelmakoodilla ilmentymiä aina tarvittaessa, joten jos halutaan luoda pelin aikana jotain tyhjästä, prefabien käyttö on välttämätöntä. (21.)

Particle system -komponentti

Particle system on komponentti, joka liitetään objektiin. Se piirtää määriteltyjen asetusten mukaan partikkeleita (particle) määritetylle alueelle. Partikkelit ovat siis 2D-kuvia, joita piirretään 3D-avaruuteen. Niitä käytetään yleensä erilaisten efektien luomiseen, kuten taikuus, kimallus tai savu ja tuli. Particle systemissä on todella paljon erilaisia asetuksia ja näitä muuttamalla saadaan aikaan monenlaisia efektejä. Määritettäviä ominaisuuksia ovat esimerkiksi partikkelin elinikä, nopeus, koko, väri, materiaali ja paljon muuta (kuva 7). (22.)



KUVA 7. Pelissä sijaitseva peliobjekti, johon on liitetty particle system tuliefektiä varten. Oikealla particle systemin asetukset.

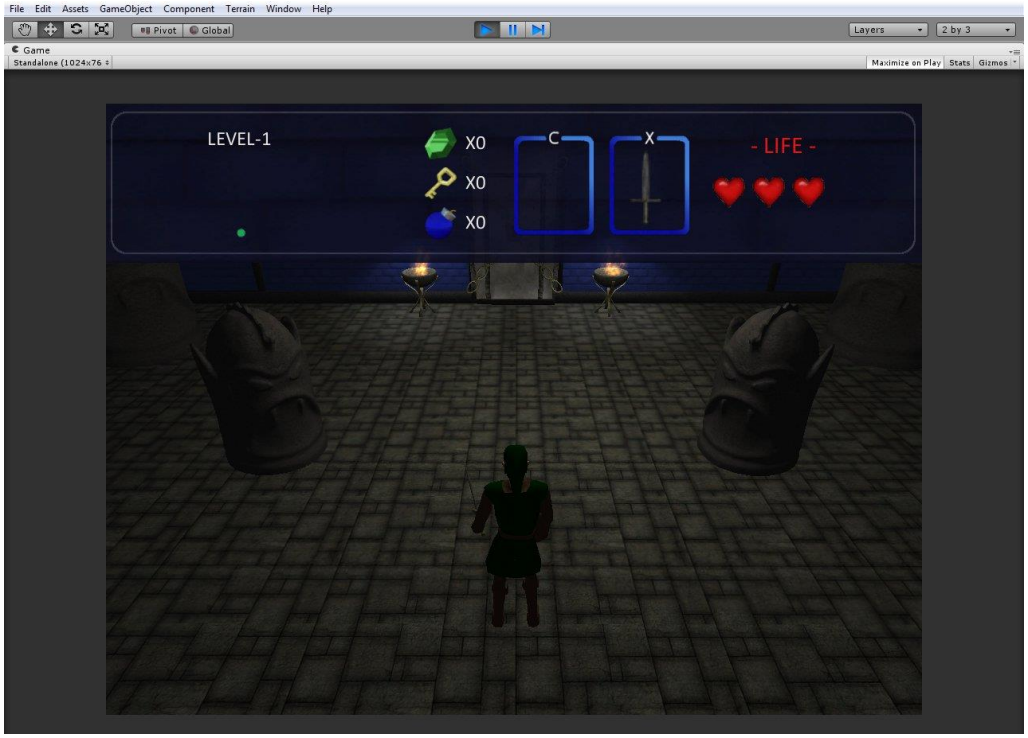
GUI (Graphical User Interface)

GUI tarkoittaa näytöllä näkyvää graafista käyttöliittymää. Se koostuu erilaisista kuvista, ikkunoista ja painikkeista. Yleensä peleissä sitä käytetään valikko-ikkunoiden lisäksi myös antamaan monenlaista informaatiota pelaajalle. (23.)

Unityssa GUI rakennetaan ohjelmakoodin avulla. Elementtiin tuleva tyyli, kuten fontti, tai graafinen ulkoasu määräytyvät oletusasetusten mukaan. Oletusasetukset voidaan ylikirjoittaa käyttämällä GUISkiniä. (24.) GUISkin luodaan Unityn projektipaneelista ja sen voi nimetä itse. Tämän jälkeen tarkastelupaneelissa voidaan asetuksia säätämällä luoda täysin omia kustomoituja tyylejä. GUISkinejä voi luoda useita. Uusi GUISkin otetaan käyttöön tekemällä skriptissä sitä varten public-muuttuja ja raahaamalla projektipaneelista haluttu GUISkin muuttujaan.

```
//luodaan uusi GUISkin
public GUISkin playerSkin;
OnGUI(){
//Otetaan alussa luotu skin käyttöön
    GUI.skin = playerSkin;
//Luodaan normaalisti halutunlainen GUI
    GUI.Label(new Rect(120, 30, GuiMapWidth, GuiMapHeight)," LEVEL-1");
}
```

GUIssa luodut painikkeet ja ikkunat tulevat näkyviin peliruudulle, kun painetaan Play-painiketta Unityn käyttöliittymän ylälaidasta ja siirrytään pelitilaan (kuva 8).

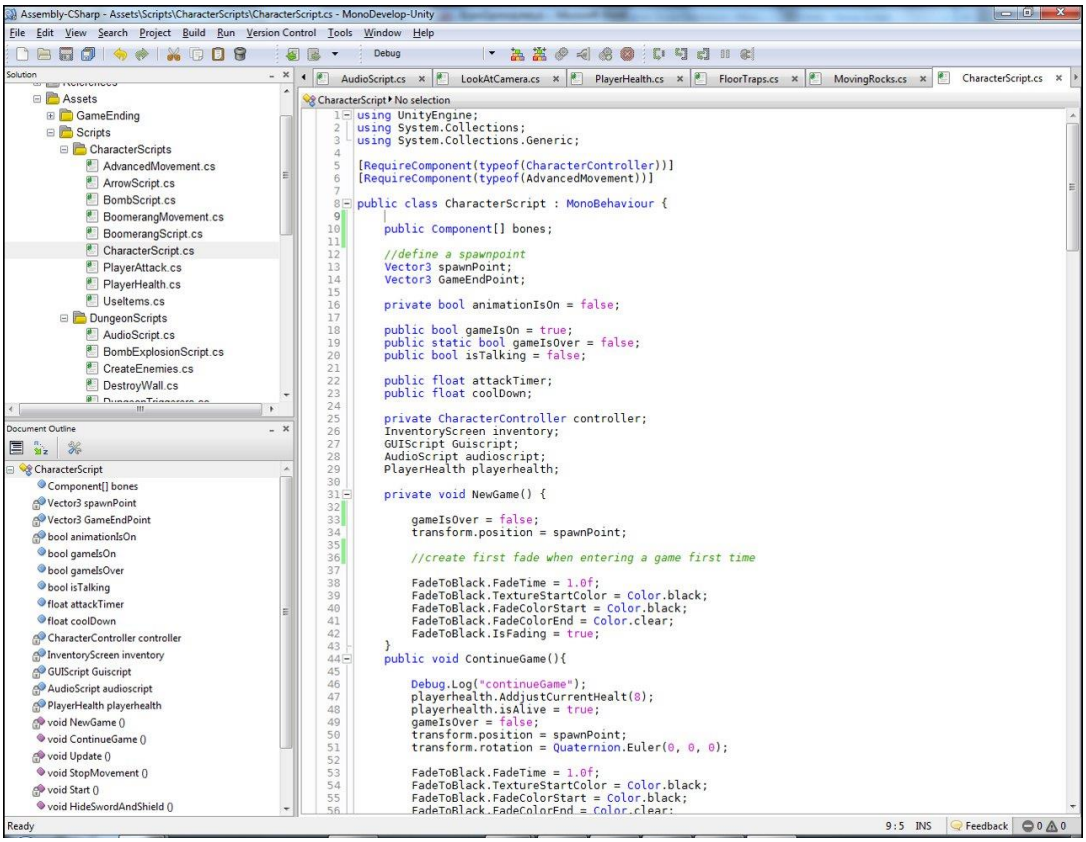


KUVA 8. Pelin GUI piirretään ruudun yläreunaan, kun Play-painike on painettuna. GUIn avulla pelaajalle voidaan välittää monenlaista tietoa pelin kulusta.

4.2 MonoDevelop-ohjelmointiympäristö

Unityn mukana tulee ohjelmointieditori nimeltä Monodevelop (kuva 9). MonoDevelop on alustariippumaton vapaaseen lähdekoodiin perustuva ohjelmointiympäristö, joka on ensisijaisesti suunniteltu C#:lle ja muille Microsoftin .NET-alustan kielille. (25.) MonoDevelopin tilalle voi vaihtaa jonkin muun halutun editorin Unityn asetuksista.

Unity-pelin ohjelmoinnin voi tehdä kolmella eri ohjelmointikielellä: C#, Javascript tai Boo. Eri ohjelmointikieliä ei ole hyvä käyttää ristiin, koska se voi aiheuttaa syntaksiongelmia. Peliobjekteihin päästään käsiksi ohjelmakoodin avulla. Esimerkiksi pelihahmon ohjaaminen näppäimistöllä tai vuorovaikutus vihollisen kanssa tehdään ohjelmakoodilla. Yleensä ohjelmakooditiedosto eli skripti liitetään siihen objektiin, jonka halutaan toimivan koodissa määritellyllä tavalla. (26.) Sanotaan, että Javascript on helpoin oppia uusille ohjelmoijille. Suuri osa tutoriaaleista ja Unityn valmiista ohjelmakooditiedostoista on .js-tyyppisiä. Boo on vähiten käytetty ja muistuttaa syntaksiltaan Pythonia. (27.)



KUVA 9. MonoDevelopin käyttöliittymä ja sillä kirjoitettua ohjelmakoodia.

Perusfunktiot

Tärkeitä perusfunktioita peliohjelmoinnissa ovat Start, Awake, Update, FixedUpdate ja OnGUI, jotka periytyvät MonoBehaviourista (28).

Kuvataajuus (lyhenne FPS eli frames per second) tai kehysnopeus tarkoittaa tietokonepeleissä näytölle sekunnissa piirrettyjen kuvien määrää. Yleissääntönä voidaan pitää, että mitä korkeampi kuvataajuus on, sitä pehmeämmältä liike näyttää kuvassa. (29.)

Start-funktio suoritetaan aina vain kerran instanssin alussa (30).

Awake-funktio suoritetaan kerran ennen Start-funktiota (31).

Update-funktiota aletaan kutsua Start-funktion jälkeen ja sitä kutsutaan kerran framessa (32).

FixedUpdate-funktiota kutsutaan asetuksissa määritellyn aikavälin mukaan. Se ei siis ole riippuvainen kuvataajuudesta, ja kutsuväli pysyy vakiona, vaikka FPS pienenisi. Yleensä sitä käytetään fysiikkaan liittyvien toiminnallisuuksien kanssa. (33.)

OnGUI-funktio on vastaavanlainen kuin Update, eli sitä kutsutaan kerran framea kohden, mutta se on tarkoitettu GUIn piirtoon (34).

Jos mitään yllä olevista funktioista ei tarvita ja halutaan luoda oma luokka, jolla on oma muodostin (constructor), MonoBehaviourista periytyminen kannattaa poistaa. Tämä siitä syystä, että MonoBehaviourin alustusfunktiot saattavat sotkea luokan muodostimessa tapahtuvat alustukset. (35.)

Coroutine-funktio

Yksi tärkeä funktio, joka periytyy MonoBehaviourista, on StartCoroutine. StartCoroutinea tarvitaan IEnumerator-tyyppisen funktion kutsuun. Coroutinea käytetään yleensä viiveen saamiseksi eri tilanteissa. (36.)

Alla olevassa esimerkissä kutsutaan SayHello-nimistä funktiota, joka on IEnumerator-tyyppinen. Kahden sekunnin odotuksen jälkeen tulostuu teksti ”Hello!”.

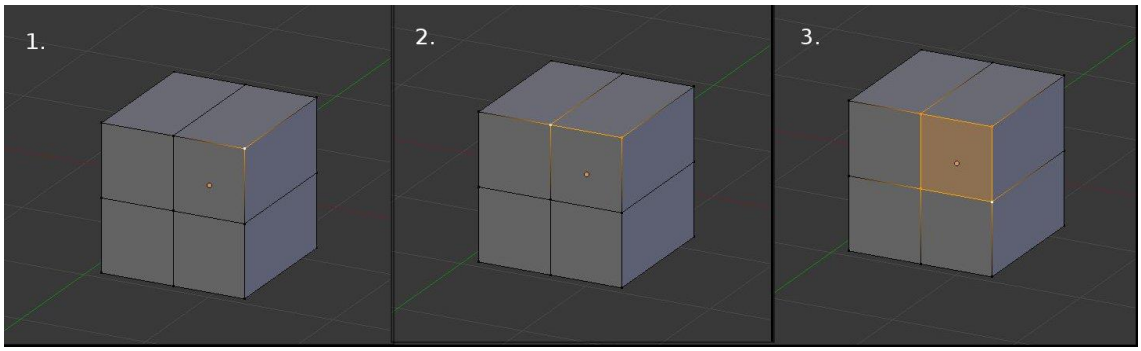
```
void Start() {
        //Kutsutaan SayHello funktiota, odotusajaksi annetaan 2 sekuntia.
          StartCoroutine(SayHello(2.0F));
}
IEnumerator SayHello(float waitTime) {
        //Odotetaan 2 sekuntia
        yield return new WaitForSeconds(waitTime);
        //Tulostetaan teksti Hello!
        print("Hello! "); }
```

4.3 Blender-mallinnusohjelma

Blender on ilmainen ja vapaa 3D-grafiikan mallinnusohjelma. Blenderin oletuskäyttöliittymä koostuu viidestä ikkunasta, joiden paikkaa voi vapaasti vaihtaa, suurentaa tai pienentää. Jokaisella ikkunalla on oma otsikkopaneeli (header), josta voi vaihtaa ikkunan tai editorin tyyppiä esimerkiksi mallintamista, teksturointia tai animointia varten. Ikkunoita voi myös jakaa pienemmäksi. Jos esimerkiksi 3D-editori jaetaan kahteen osaan, voi toisen osan kuvakulman asettaa katsomaan mallia suoraan edestäpäin ja toisen sivultapäin. Näin pystytään näkemään malli useasta eri suunnasta yhtäaikaa, ja helpottamaan hahmottamista 3D-avaruudessa. (37.)

4.3.1 Mallintaminen Blenderillä

Perustilat 3D-editorissa mallintamista varten ovat objektitila (object mode) ja muokkaustila (edit mode), joiden välillä vaihdellaan jatkuvasti. Objektitilassa toiminnot vaikuttavat koko objektiin. Muokkaustilassa pystytään vaikuttamaan ainoastaan geometriaan. (38.)



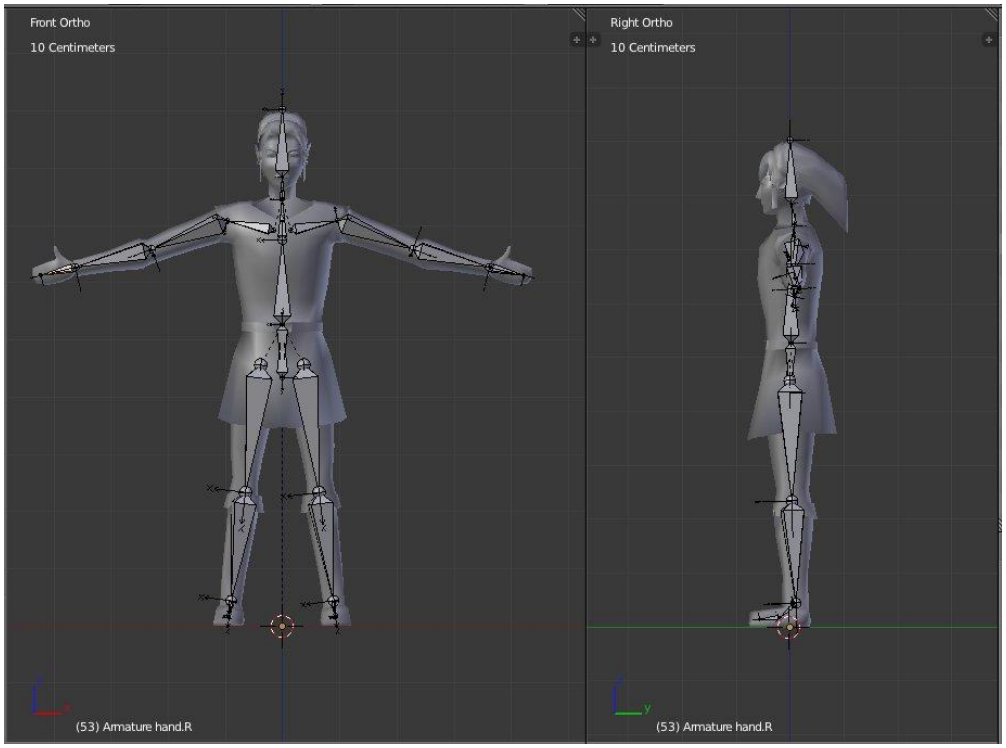
KUVA 10. Kärkipiste, särmä ja tahko valittuna Blenderin 3D-editorissa muokkaustilassa.

Yleensä pohjana käytetään jotain yksinkertaista muotoa, kuten kuutiota, josta lähdetään muokkaamaan haluttua lopputulosta. Kaikki muodot koostuvat kärkipisteistä (vertex), särmistä (edge) ja tahkoista (face). Verteksit yhdistyvät särmillä ja vähintään kolme verteksiä ja niiden välillä olevat särmät muodostavat tahkon (kuva 10). Mallin jokaista pistettä eli verteksiä voidaan liikutella x-, y- ja

z-suunnassa ja mallia voi jakaa pienempiin osiin lisäämällä verteksejä, mikä taas lisää särmiä ja tahkoja. (39.)

4.3.2 Riggaaminen

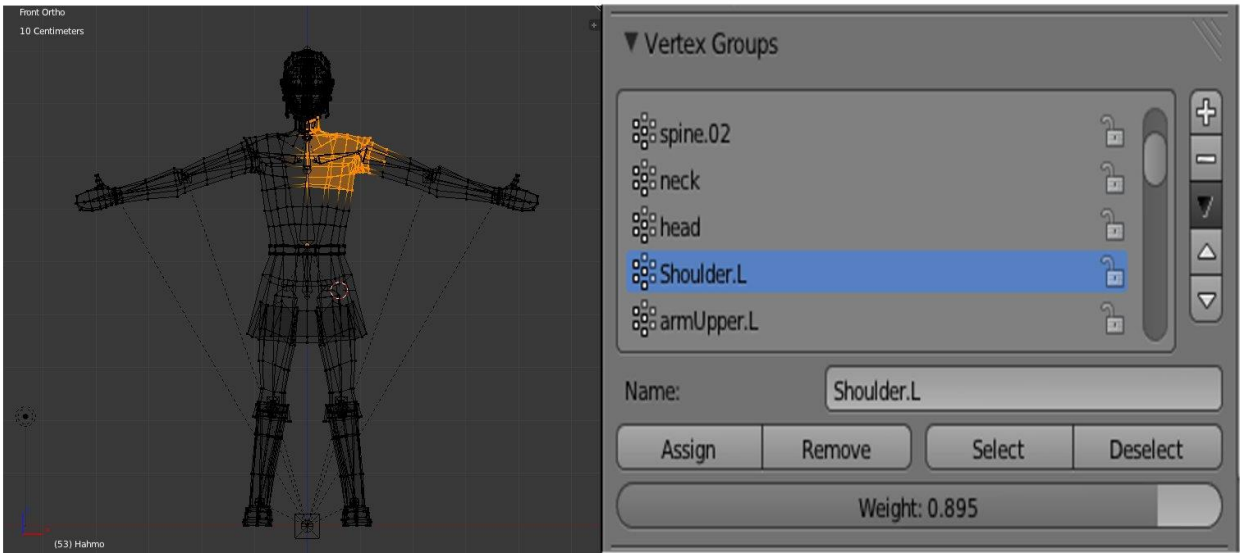
Kun hahmo on saatu valmiiksi ja halutaan saada vaihdettua mallin asentoa, esimerkiksi animointia varten, malli täytyy rigata (rigging). (40.) Riggaaminen tarkoittaa luurangon (armature) asettelua mallinnettuun hahmoon (kuva 11). Luuranko koostuu useista yksittäisistä luista, jotka ovat sidoksissa toisiinsa. Jokainen luu koostuu keskiosasta ja kahdesta päästä. Päät ovat luiden välisiä nivelkohtia ja ne kääntyvät ainoastaan nivelkohdista oikeiden luiden ja nivelten tapaan. Luut kannattaa asetella järkevästi oikeaa luurankoa jäljitellen, jotta liikkeistä saadaan luonnollisia. Ei ole tarpeen lisätä jokaista kehon luuta kuten kylkiluita. (41.)



KUVA 11. Pelihahmolle aseteltu luuranko 3D-editorissa.

Kun luuranko on valmis, se yhdistetään mallinnettuun hahmoon. Tätä prosessia kutsutaan nimellä skinning. Blender voi laskea automaattisesti, mitkä verteksit sidotaan mihinkin luuhun eli mitkä verteksit liikkuvat kunkin luun mukana. Verteksiryhmät voi määrittää myös itse ja niitä voi muutella tarpeen mukaan.

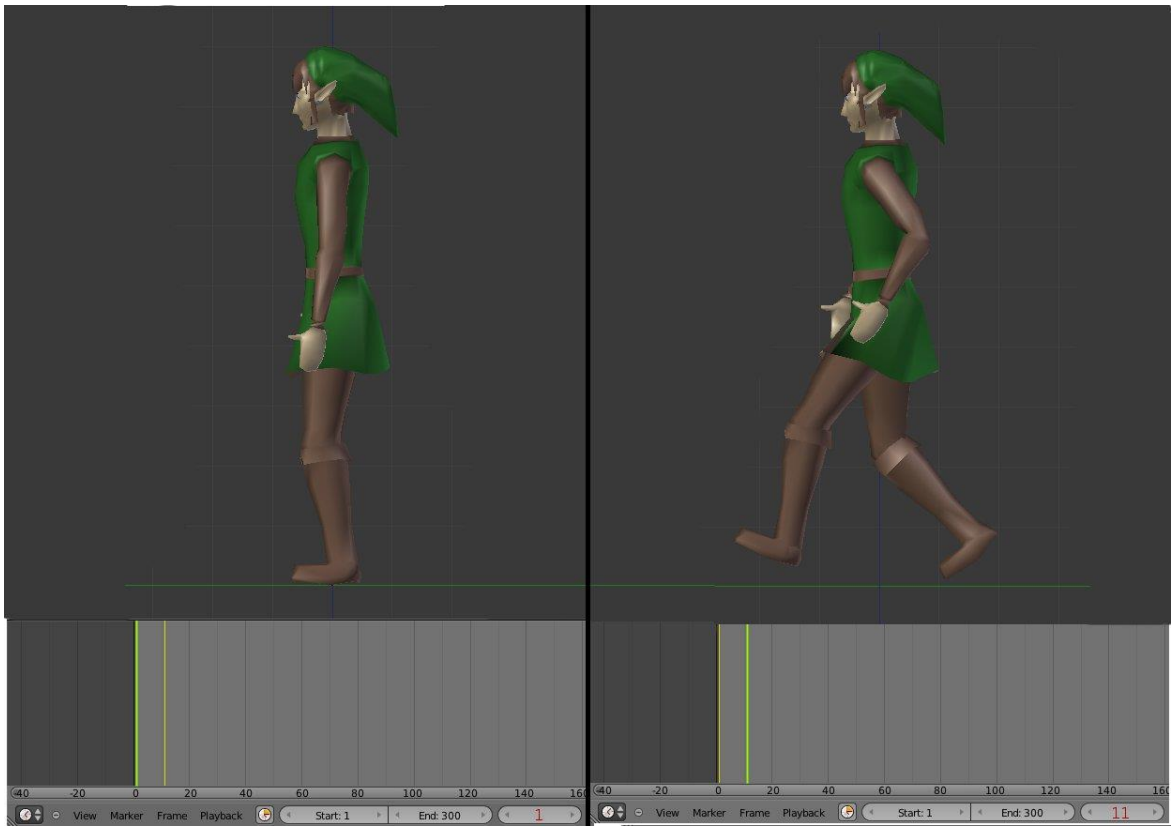
Yleensä automaattinen laskenta toimii ihan hyvin, mutta joitain korjauksia saattaa joutua tekemään, jos malli on kovin erikoisen muotoinen (kuva 12). (42.)



KUVA 12. Verteksiryhmät. Kuvassa oranssina näkyvä alue on liitetty luuhun nimeltä Shoulder.L. Tämän luun liikkuminen vaikuttaa siis näihin vertekseihin.

4.3.3 Animointi

Animaatio on kappaleen liikkumista tai muutosta ajan kuluessa. Animointi blenderillä tapahtuu aikajana-editorissa (timeline) ja koostuu piirtokehyksistä eli frameista. Aikajanalle määritellään sopivin kehysvälein avainkohtia (keyframe), joihin tallennetaan haluttu mallin sijainti ja asento. Avainkohtien väliin jäävät framet generoituvat automaattisesti. Animaatio muodostuu siis mallin siirtymisestä avainkohdasta toiseen (kuva 13). Hahmolle voi tehdä myös useampia animaatioita, kunhan tietää, mistä avainkohdasta tietty animaatio alkaa ja mihin se päättyy. Kun malli on siirretty Unityyn, animaatiot täytyy nimetä ja määrittää, mistä avainkohdasta animaatio alkaa ja mihin se päättyy. Animaation asetuksista voidaan myös asettaa, näytetäänkö animaatio vain kerran vai toistetaanko sitä. (43.)



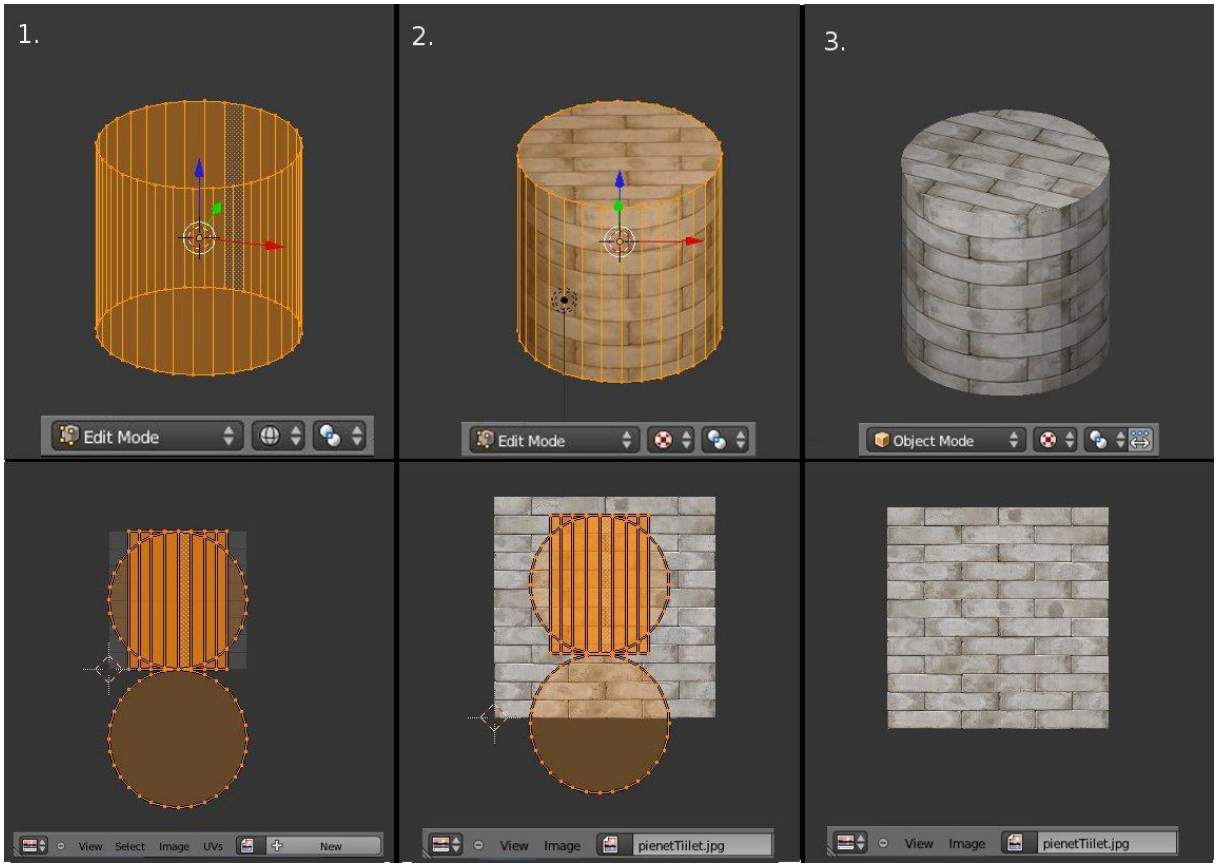
KUVA 13. Animaation tekeminen. Hahmon asento vaihtuu ensimmäisestä kuvasta toiseen 10 framen aikana. Aikajanalle määritellyt avainkohdat näkyvät kuvan alalaidassa keltaisina viivoina ja valittu frame näkyy vaaleanvihreänä viivana.

4.3.4 Materiaalit ja teksturointi

Malliin voidaan liittää materiaaleja ja tekstuureja. Materiaaleja voi olla useita ja ne voidaan lisätä koko malliin tai vain tiettyihin osiin. Materiaalilla voidaan vaikuttaa esimerkiksi mallin väriin, kiiltävyyteen, läpinäkyvyyteen ja moniin muihin ominaisuuksiin. (44.)

Tekstuuri on yleensä 2D-kuva, joka on liitetty materiaaliin (45.). Myös tekstuurilla on paljon erilaisia ominaisuuksia kuten läpinäkyvyys, tekstuurin kuvion toistuminen, värit ja tärkeimpänä kartoitus siitä, kuinka se asettuu objektin pintaan (mapping). Paras tapa asettaa 2D-tekstuuri 3D-mallin pinnalle on UV-mappaus (UV-mapping). UV-mappauksessa kolmiulotteisen mallin pinnat kääritään auki kaksiulotteisesti UV-editoriin. Kirjainlyhenteellä UV viitataan kaksiulotteisuuteen ja yleensä korkeutta ja leveyttä kuvaavia kirjaimia

X ja Y ei käytetä, jottei se sekoittuisi kolmiulotteisuuteen. Kun pinnat on kääritty auki, on mahdollista asetella tekstuuri mallin pinnalle halutulla tavalla (kuva 14). UV-editorista pystyy myös tallentamaan auki käärityn 3D-mallin pintojen ääriviivat erilliseen png-tiedostoon. Ääriviivojen avulla pystyy piirtämään erillisellä kuvankäsittelyohjelmalla, kuten Gimpillä, tiettyjä alueita varten täysin omaa 2D-grafiikkaa. (46.)

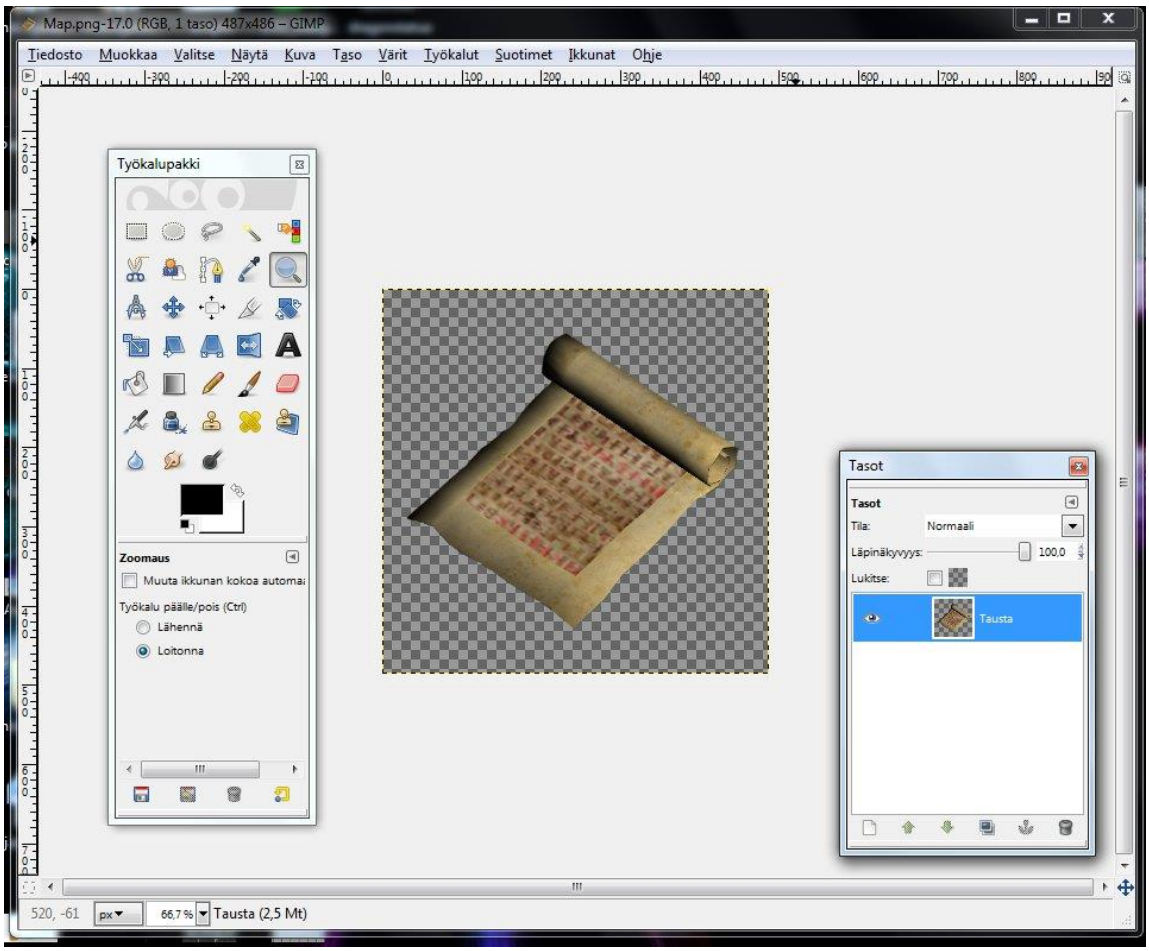


KUVA 14. Sylinterin teksturointi. Ylemmät kuvat ovat 3D-editorista ja alemmat UV-editorista. Sylinterin pinnat avataan UV-editoriin ja tekstuuri asetellaan objektin pinnoille.

4.4 Gimp-kuvankäsittelyohjelma

Gimp (GNU Image Manipulation Program) on avoimeen lähdekoodiin perustuva monipuolinen kuvankäsittelyohjelma. Gimpillä onnistuu helposti grafiikan luominen, kuvakoon muuttaminen, värien vaihtaminen, kuvien yhdistäminen tasojen avulla, kuvanparannus sekä kuvaformaattien väliset muunnokset. (47.)

Gimpillä voi avata yhtä aikaa monta kuvaa ja jokainen niistä aukeaa omaan ikkunaansa, joten kuvien yhdistely on näppärää. Työkalupalkki ja muut apuikkunat ovat myöskin vapaasti liikuteltavissa (kuva 15). Piirroksen eri osat voidaan laittaa eri tasoihin ja kuva voidaan tallentaa .xcf-muotoon, jolloin tasot säilyvät. Näin voidaan säästyä paljolta vaivalta, jos kuvaa halutaan vielä muokata jälkikäteen. Gimpillä voi myös muutamalla napin painalluksella muuttaa tekstuurin saumattomasti jatkuvaksi tai taustaltaan läpinäkyväksi. (47.)



KUVA 15. Kuva Gimpin käyttöliittymästä.

Gimpillä voidaan piirtää myös piirtopöytää käyttäen. Tarkkuutta vaativien kuvien piirtäminen kuvankäsittelyohjelmalla voikin olla hankalaa pelkästään hiirtä käyttämällä. Piirtopöydän avulla voidaan jäljitellä oikeiden piirustus -ja maalausvälineiden ominaisuuksia. Nykyiset piirtopöydät tunnistavat paineen ja jopa piirtopöydän kanssa käytettävän kynän eli styluksen kallistuskulman. (48.) Tässä opinnäytetyössä käytettiin Wacom Bamboo Pen -piirtopöytää (kuva 16.).

KUVA 16. Piirtopöytä (49)

5 PELIN TOTEUTUS

Peli toteutettiin Unity3D:n ilmaisversiolla, jonka käytöstä oli hieman kokemusta aikaisemman työharjoittelun kautta. Ohjelmointikieleksi valittiin C#. 3D-mallit ja animaatiot tehtiin Blenderillä. Tarvittavat 2D-grafiikat peliin tehtiin Gimpillä. Projektin hallintaan käytettiin Scrum-menetelmästä tuttuja dokumenttipohjia.

Pelin karkea suunnitelma tehtiin listaamalla paperille, mitä kaikkea pelin ensimmäisestä luolastosta löytyy, ja samalla työn sisältö rajattiin. Tämän pohjalta mietittiin tulevan pelin arkkitehtuuria ja suunniteltiin, millaisiin kokonaisuuksiin pelin eri osat, kuten viholliset, esineet, luolasto ja pelihahmon ominaisuudet, jaettaisiin ja miten ne kommunikoivat keskenään. Samalla täytyi pohtia, mitä muutoksia alkuperäiseen peliin tehtäisiin uutta versiota varten. Kun projektin sisältöä oli hahmoteltu tarpeeksi, otettiin käyttöön Scrum-dokumentit.

Alkuperäisen pelin pelaamisesta oli kulunut myös sen verran aikaa, että sitä täytyi pelata työn lomassa uudestaan muistin virkistykseksi.

5.1 Scrum-suunnitelmadokumentit

Projektin toteutus jaettiin neljään sprinttiin (Liitteet 3–4). Jokaiseen sprinttiin jaettiin sopivasti sekä mallinnustyötä että varsinaista koodausta ja projektin rakentelua Unityn puolella (kuva 17). Tilannetta, jossa peli toimisi hyvin, mutta olisi puutteellinen grafiikan suhteen tai päinvastoin, haluttiin välttää.

Ominaisuuden ID				
Prioriteetti korkea	Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus ...	Sprintin numero	Toteutuksen kesto (=Story Point)	Huomioitavaa
1	Luo pohja pelimaailmalle (projekti ja terrain).	1	1	
2	Tee hahmon ohjaus, (Third Person Character Controller).	1	1	
3	Luo kamera, joka kuvaa hahmoa ja siirtyy pelaajan mukana kun huone vaihtuu (Camera Controller).	1	1	
4	Mallinna luolasto, jonka sisällä pelaaja pelaa (pelin kenttä).	1	2	
5	Mallinna pelihahmo ja tee hahmolle kävelyanimaatio.	1	3	
6	Tee inventory-ruutu (GUI), josta pelaaja voi tarkastella omistamiaan esineitä ja vaihtaa käytössä olevan esineen.	2	1	
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot	2	1,5	
8	Pelaajan ja hahmon välinen vuorovaikutus, pelaaja ottaa vahinkoa vihollisesta ja pystyy tuhoamaan ne.	2	0.5	
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)	2	1	
10	Pelin tallennus, talteen jää kerätyt rahat, kerätyt esineet, tuhotut seinät ja avatut ovet. Viholliset spawnaa uudestaan kun luolastoon astutaan.			
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.	3	3	
12	Viholliset tiputtaa satunnaisesti rahaa, sydämiä ja pommeja	2	0.5	
13	Vihollisten "tekoäly", ts liikkuminen ympäriinsä.	2	1,0	
14	Äänet ja musiikit peliin	4	2,0	

KUVA 17. Tuotteen kaikki ominaisuudet -dokumentti.

Ensimmäisessä sprintissä (liite 5.) tehtiin pelikentän, eli luolaston ja pelihahmon mallintaminen ja animoiminen sekä kameran toimintaan saaminen. Aikaa meni myös Blenderin käytön opetteluun.

Toisessa sprintissä (liite 6.) tehtiin pelin GUI ja pelissä olevia esineitä. Esineitä pystyy poimimaan ja niiden lukumäärä sekä pelihahmon terveydentila näkyy GUIssa. Lisäksi mallinnettiin ensimmäinen vihollinen ja tehtiin vuorovaikutusta niiden välille. Pelihahmo ottaa vahinkoa vihollisesta ja pystyy vahingoittamaan sitä. Tuhoutuessaan vihollinen voi pudottaa esineen. Pelihahmo pääsee alakertaan tikkaita pitkin.

Kolmannessa sprintissä (liite 7.) peliin tehtiin lisää erilaisia vihollisia ja loppuvastus. Yksittäisen vihollisen tekeminen sisältää sen mallintamisen, riggaamisen, skinnaamisen, animoimisen ja teksturoimisen. Tämän lisäksi se täytyy saada toimintaan Unityn puolella. Vihollisten luominen muutettiin sellaiseksi, että jokaisen huoneen viholliset luodaan vasta huoneeseen astuttaessa. Peliin lisättiin pommin toiminnallisuus, jolla voi räjäyttää pari seinää ja tuhota myös vihollisia. Peliin lisättiin erilaisia efektejä, kuten savua, tulta ja seinänkappaleita. Luolastoon lisättiin myös erilaisia tapahtumia. Kun huoneesta tuhotaan viholliset, voi ilmestyä avain tai ovi aukeaa

Neljännessä sprintissä (liite 8.) paranneltiin ja tehtiin yleisesti kaikki, mitä ei ollut aiemmin ehditty tehdä. Jousipyssy ja bumerangi mallinnettiin ja saatiin toimintaan. Peliäänet ja musiikki lisättiin. Peliin täytyi mallintaa myös kartta ja kompassi ja saada ne toimimaan. Ansat aseteltiin paikoilleen ja niille tehtiin toiminnallisuus. Peliin täytyi piirtää aloitusruutu, sekä tehdä pelin loppumista varten loppudemo. Pelihahmo pystyy kuolemaan ja pelin voi aloittaa alusta tai sitä voi jatkaa samasta tilanteesta. Peliin mallinnettiin vanha mies ja hänelle tehtiin kauppa. Yksi lisäviholinen mallinnettiin ja laitettiin toimintaan. Pelin grafiikkaa ja monia malleja paranneltiin. Kameran toimintaa korjattiin. Peliin mallinnettiin koristeita, kuten patsaita, soihtuja ja roihuja, ja niitä aseltetiin pitkin luolastoa. Peliä täytyi myös optimoida valojen suhteen, koska se alkoi olla aika raskas. Valot syttyvät, kun pelihahmo astuu huoneeseen, ja sammuvat kun hän poistuu sieltä. Peliä testattiin ja löytyneitä virheitä korjattiin.

5.2 Pelin toteutus Unityssa

Työnteko aloitettiin luomalla Unityn projekti. Aluksi käytettiin vain yhtä skeneä, joka on varsinainen peliskene. Ensimmäinen tärkeä päätös oli, millainen pelin kamera tulee olemaan. Ensin aikeissa oli käyttää First Person -kuvakulmaa, mutta tästä luovuttiin, koska yksikään Zelda-peli ei ole kuvattu tästä kuvakulmasta ja olisi mukavaa jos pelihahmon pystyisi näkemään. Googlettamalla löytyi hyvä tutoriaali erilaisista kameratyypeistä:

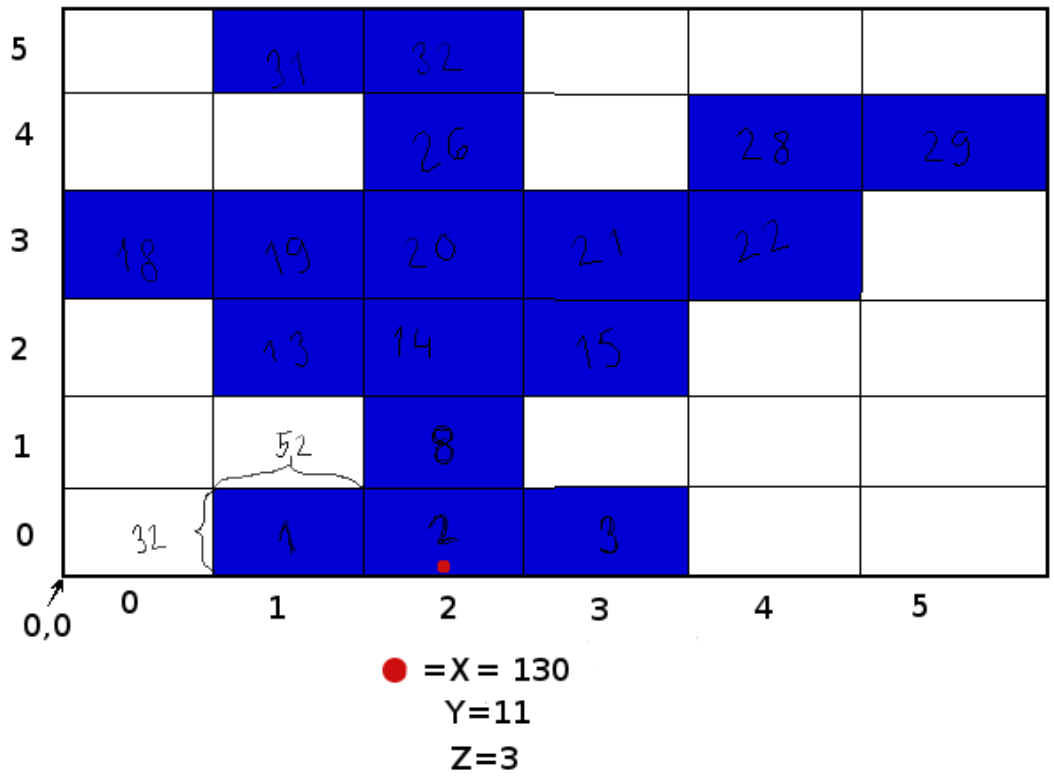
http://mobile.tutsplus.com/tutorials/game-engine/unity3d-third-person-cameras/.

Erilaisia kameravaihtoehtoja kokeiltiin luomalla pelihahmon korvikkeeksi kuutio, johon lisättiin yksinkertainen ohjattavuus. Lopulta suunniteltiin tutoriaalista löytyvien kameraskriptien pohjalta oma versio, joka on kahden eri kameratyypin sekoitus. Kamera kuvaa huonetta yhdestä pisteestä ja tähtää koko ajan kohti pelihahmoa. Kun huone vaihtuu kamera rullaa seuraavan huoneen pisteeseen ja jatkaa sieltä pelihahmon katsomista.

5.2.1 Luolasto ja kamera

Kun luolasto oli mallinnettu Blenderilla ja siirretty Unityyn, kameraa pystyttiin testaamaan käytännössä. Luolaston malli asetettiin skenenäkymään niin, että sen koordinaatit alkoivat pisteestä 0,0,0 (x, y, z). Jokaiseen seinään, lattiaan, kynnykseen ja kiveen asetettiin colliderit, jotta pelialue on rajattu oikeanlaiseksi. Joka huoneeseen määritettiin oma paikka kameraa varten, ja täytyisi olla tiedossa, missä huoneessa pelihahmo milloinkin on, jotta kamera on aina samassa huoneessa. Tätä varten täytyi laskea indeksi jokaiselle huoneelle ja laskea pelihahmon koordinaattien perusteella, mikä huone on kyseessä.

Jokainen huone luolastossa on samankokoinen (52 x 32). Indeksin laskemista varten luotiin ruudukko. Pelikentän pituus ja leveys huoneissa on maksimissaan 6 x 6 (kuva 18). Pelissä ei kumminkaan ole 36:ta huonetta, mutta ylimääräisillä huoneilla ja niiden indekseillä ei ole mitään väliä, koska pelikenttä rajoittaa pelaajan kulkua.



KUVA 18. Pelin luolaston pohjapiirros. Piirroksessa näkyvät rivit ja kolumnit sekä huoneiden indeksit.

Huoneen indeksi voidaan laskea rivin ja kolumnin avulla, jotka taas lasketaan pelihahmon koordinaattien perusteella.

```
//Huoneen mitat
int maxColumn = 6;
int maxRow = 6;
public float roomWidth = 52.0f;
public float roomHeight = 32.0f;
//Lasketaan kolumni ja rivi koordinaattien perusteella
column = System.Math.Floor(target.transform.position.x / roomWidth);
row = System.Math.Floor(target.transform.position.z / roomHeight);
```

Kun kolumni ja rivi on tiedossa, lasketaan huoneen indeksi.

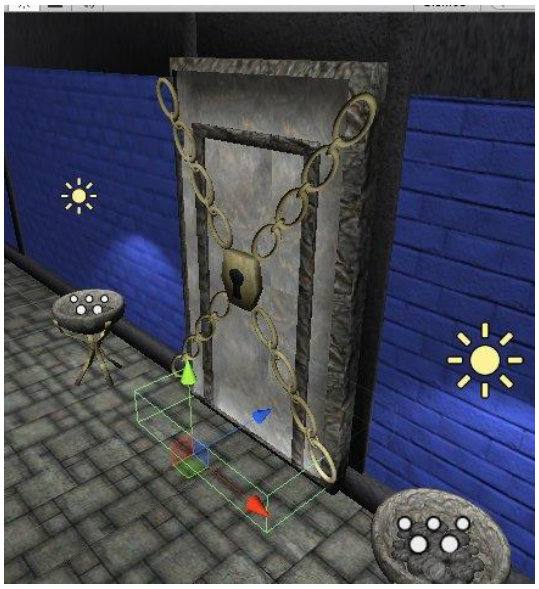
```
Indeksi = column + (row * maxColumn);
```

Aluksi kameralle laskettiin sijainti joka huoneeseen ruudukon avulla, ja ne laitettiin Vector3-tyyppiseen listaan. Vector3 sisältää x-, y- ja z-koordinaatit. Listassa olevan sijainnin indeksi on aina vastaava kuin huoneen indeksi. Kun pelihahmo siirtyy vaikkapa huoneeseen, jonka indeksi on 2, kamera siirtyy listassa indeksiin 2 olevaan pisteeseen ja niin edelleen.

Ongelmia kamerassa tuotti kameraan päin kulkeminen eli kun pitäisi kulkea ovesta, joka sijaitsee suoraan kameran alapuolella. Tätä varten tehtiin omat erikoispisteet kameraa varten. Jos kameran kulma on yli 70 astetta pelihahmoon nähden, pelihahmo on oviaukon luona ja kamera siirretään ”alapuolella” olevan huoneen keskipisteeseen. Talteen otetaan pelihahmon senhetkinen paikka. Kameran ja pelihahmon tallennetun paikan välimatkaa tutkitaan. Jos välimatka kasvaa yli määritetyn rajan, kamera vapautetaan taas liikkumaan normaalisti siihen huoneen indeksiin, jossa pelihahmo on. Jos astutaan uuteen huoneeseen, indeksin vaihtuessa kamera siirtyy normaalisti omaan indeksiinsä.

Ovet

Kun kamera toimi tarpeeksi hyvin, luolastoon lisättiin ovet, jotka estävät hahmon kulkemista. Oven ohi pääsee kulkemaan vain, jos hahmolla on mukanaan avain. Ovien eteen lisättiin toiset colliderit, joita käytettiin triggereinä (kuva 19).

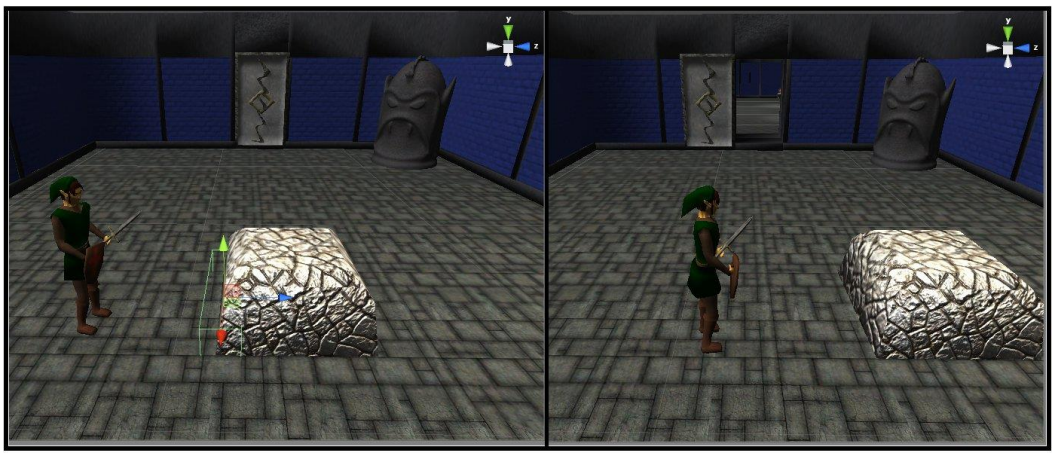


KUVA 19. Lukitun oven edessä oleva triggeri.

Jos triggeri havaitsee pelihahmon saapuvan oven luo ja mukana on avain, ovi tuhotaan ja avainten määrää vähennetään yhdellä. Koska pelissä on useita ovia, myös tässä tarvittiin tageja yksilöimään mikä ovi täytyy tuhota. Tavallisten ovien lisäksi pelissä on ovia, jotka avautuvat ja sulkeutuvat erilaisten tapahtumien mukaan. Tätä varten tarvitaan tieto, missä huoneessa kulloinkin ollaan. Kameraa varten huoneen indeksi onkin jo jatkuvasti tiedossa ja sitä hyödynnettiin myös tässä. Oveen on tehty Blenderissä yksinkertaiset animaatiot avaamista ja sulkemista varten. Kun huoneeseen astutaan, käynnistetään oven animaatio, joka saa sen siirtymään oviaukon eteen. Tähän tarvittiin vähän viivettä (Coroutine), jottei ovi pamahda kiinni ennen hahmon saapumista huoneeseen. Ovi avataan, kun huoneesta on tuhottu kaikki viholliset. Toinen vastaavanlainen ovi avautuu, kun huoneessa siirretään erästä kiveä, ja kolmas ovi sen jälkeen, kun loppuvastus on voitettu.

Siirrettävät kivet

Luolastossa on muutama kivi, joita pystyy siirtämään. Näihin kiviin tehtiin animaatio Unityn puolella, koska ne ovat osa isompaa luolastomallia (50). Kiven yhdelle sivulle on asetettu collider triggeröintiä varten, samalla tavalla kuin ovissakin. Kun pelihahmon havaitaan tulevan kiven luo, käynnistetään animaatio, joka siirtää kiveä. Ensimmäisen kiven liikauttaminen käynnistää samalla myös oven animaation, joka saa oven aukeamaan (kuva 20). Toinen liikuteltava kivi taas on vain tien tukkeena ja se täytyy siirtää pois, jotta päästään etenemään alakertaan.



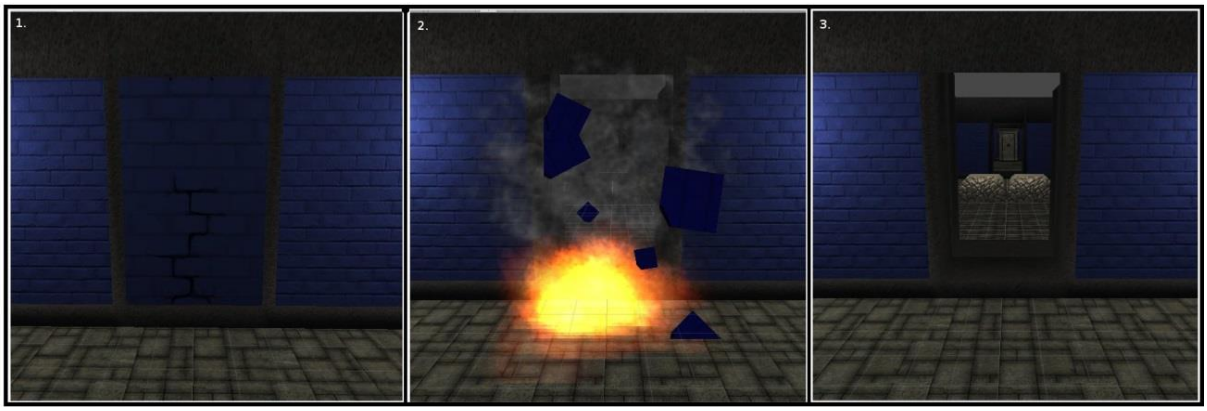
KUVA 20. Kiven siirtäminen avaa huoneessa olevan oven.

Tikkaat

Eräästä huoneesta päästään toiseen huoneeseen tikkaita pitkin. Tikkaiden luona on samanlainen triggeri kuin ovissa ja kivissä, mikä saa aikaan siirtymisen toiseen huoneeseen. Samalla kameran kuvaa sumennetaan hetkeksi. Sekä kellarihuoneessa että yläkerrassa tikkaiden luona on tyhjät peliobjektit, jotka osoittavat paikan, johon hahmo siirretään. Tälläisiä pisteitä kutsutaan spawnpointeiksi. Kun pelihahmo siirtyy alakertaan, samalla kamera siirretään kellarihuoneen indeksiin. Kellari ei oikeasti sijaitse tikashuoneen alapuolella, vaan on samassa tasossa muiden huoneiden kanssa, jotta sen sai helposti indeksoitua. Alakerrassa on toiset tikkaat, joiden luona on myös triggeri, joka siirtää pelihahmon takaisin yläkerrassa sijaitsevaan spawnpointtiin.

Räjähtävät seinät

Luolastossa on pari seinää, jotka pystyy räjäyttämään pommilla. Luolastomalliin on jätetty oviaukot näitä kohtia varten. Aukot on peitetty Unityn puolella asettamalla erillinen seinäpala aukon peitoksi. Näin seinä näyttää aivan kiinteältä, eikä sen läpi pääse kulkemaan. Jotta pelaaja hoksaisi, että seinän voisi ehkä räjäyttää, siitä tehtiin tarkoituksella hieman erinäköinen. Seinän tapauksessa ei käytetty samanlaista triggeriä kuin ovissa ja siirtyvissä kivissä. Varsinainen komento, tuhotaanko seinä vai ei, tulee pommissa olevasta skriptistä. Seinässä on kiinni kaksi particle systemiä, toinen savua ja toinen seinänkappaleita varten.



KUVA 21. Räjäytettävä seinä on hieman erinäköinen kuin tavallinen seinä. Kun pommi räjähtää seinän lähellä, tarvitaan useita efektejä ja seinä tuhotaan.

Kun seinälle tulee komento tuhoutua, seinän MeshRenderer ja collider asetetaan falseksi ja käynnistetään particle systemit, jotka alkavat luoda savua ja seinänpalasia (kuva 21). Seinä häviää näkyvistä ja sen läpi pääsee kulkemaan. Pelaajan näkökulmasta seinää ei enää ole olemassakaan. Pienen viivytyksen jälkeen pysäytetään particle systemien emittointi ja lopuksi seinäobjekti tuhotaan oikeasti ohjelmakoodilla. Tämä kaikki tapahtuu hyvin nopeasti.

Seinän tuhoaminen viiveellä tehdään efektien vuoksi. Jos seinäobjektin tuhoaisi suoraan, efektiä ei ehtisi huomata ollenkaan. Particle systemit katoaisivat samaan aikaan kuin seinäkin, koska ne ovat sen lapsiobjekteja. Niiden ei olisi pakko olla, mutta niiden tuhoutumisesta pitäisi muutoin huolehtia erikseen ja näin on helposti tiedossa, minkä seinän particle systemit käynnistetään tai suljetaan.

Vihollisten luonti

Kun peli aloitetaan, luolasto on tyhjä. Viholliset luodaan sitä mukaa, kun luolastossa edetään. Ei olisi kovin järkevää luoda heti pelin alussa luolastoa täyteen vihollisia, koska pelaaja ei pysty näkemään niitä ja pelistä tulisi raskas. Monissa tapauksissa tarvitaan tieto siitä, ovatko huoneen viholliset elossa vai onko ne tuhottu. Tämän tiedon pohjalta käynnistetään erilaisia tapahtumia, kuten oven avaaminen tai avaimen ilmestyminen.

Vihollisten luomista varten tarvittaviin huoneisiin on aseteltu useita tyhjiä peliobjekteja, joita käytetään spawnpointteina. Spawnpoint on näkymätön ja sen tehtävänä on määrittää paikka, johon vihollinen luodaan. Jokaiselle vihollistyypille on oma spawnpointinsa. Heti pelin alussa haetaan kaikki pelin spawnpointit listaan nimeltä AllSpawnPoints.

```
public void AddAllEnemieSpawnpoints(){
//Etsitään pelin kaikki peliobjektit  joiden tagi on Spawnpoint
GameObject[] Spawnpoint = GameObject.FindGameObjectsWithTag("Spawnpoint")
//Lisätään jokaisen löydetyn objektin transform listaan
foreach(GameObject sp in Spawnpoint){
        AddSpawnpoint(sp.transform);
        }}
```

Jälleen kerran hyödynnetään kameraa varten tehtyä indeksointia, eli tietoa siitä, missä huoneessa pelihahmo on. Joka kerta kun astutaan uuteen huoneeseen, tutkitaan onko huoneessa spawnpointteja. Tämä tapahtuu laskemalla jokaiselle AllSpawnPoints-listassa olevalle spawnpointille indeksi sen koordinaattien perusteella ja vertaamalla sitä huoneen indeksiin, jossa pelihahmo on.

```
public void CheckRoomSpawnpoints(int indeksi){
//Jos huoneessa ei ole vielä käyty luodaan sinne viholliset
if(camera.visitedRooms[indeksi] == 0){
//Käydään läpi lista, jossa on kaikki pelin spawnpointit
    for(int i=0; i<=AllSpawnpoints.Count-1; i++){
  //Jokaiselle pisteelle lasketaan kolumni ja rivi koordinaattien perusteella
    column = System.Math.Floor(AllSpawnpoints[i].transform.position.x /
                                                LookAtCam era.roomWidth);
    row = System.Math.Floor(AllSpawnpoints[i].transform.position.z /
                                            LookAtCamera.roomHeight);
     //Lasketaan indeksi
    Index = (int) camera.CalculateIndex(column,row);

//Verrataan indeksiä pelaajan senhetkisen huoneen indeksiin
    if(Index == indeksi)
    {
        //Jos huone on sama, kuin missä pelaaja on, luodaan näistä pisteistä viholliset
          CreateThisRoomEnemies(AllSpawnpoints[i]);
    }
}
//Lisätään äsken luotujen vihollisten määrä toiseen listaan
EnemyCount[indeksi] = enemyCount;
enemyCount = 0;}
```

Toisin sanoen listasta etsitään ne spawnpointit, joilla on sama indeksi kuin pelihahmolla. Kun on tiedossa, mitkä spawnpointit ovat samassa huoneessa, tutkitaan spawnpointin nimi, joka kertoo vihollisen tyypin ja sen mukaan luodaan näihin pisteisiin sen tyyppinen vihollinen.

```
public void CreateThisRoomEnemies(Transform sp){
//Kasvatetaan luotujen vihollisten lukumäärää yhdellä
enemyCount++;
//Otetaan spawnpointin sijainti talteen
```

```
Vector3 spawnpoint = sp.transform.position;
//Tutkitaan spawnpointin tyyppi
switch(sp.name){
    //Luodaan Stalfos
    case "StalfosSpawnpoint":
        Instantiate(stalfos, spawnpoint, Quaternion.identity);
        break;
  //Luodaan Keese
  case "KeeseSpawnpoint":
      Instantiate(keese, spawnpoint, Quaternion.identity);
      break;
   ...
```

Samalla otetaan talteen, kuinka monta vihollista luotiin. Lukumäärä tallennetaan toiseen listaan huonetta vastaavaan indeksiin. Tämän avulla voidaan pitää kirjaa siitä, kuinka monta vihollista missäkin huoneessa on luotuna. Aina kun vihollinen kuolee, vähennetään lukumäärää yhdellä taulukon siitä indeksistä, jossa vihollinen on. Tietenkin koko ajan täytyy pitää kirjaa myös siitä missä luolaston huoneissa on käyty, jottei vihollisia luotaisi joka kerta uudestaan, kun huoneeseen astutaan.

Valot

Luolasto sijaitsee maan alla, joten se on pimeä paikka. Olikin hyvin tärkeää saada pelaajalle riittävä valaistus, mutta kumminkin luolamaisen hämärä vaikutelma. Jokaiseen huoneeseen asetettiin Point Light -tyyppinen valo ja sen asetuksista valon väriä muutettiin kellertäväksi. (51.) Tämän lisäksi huoneisiin aseteltiin soihtuja ja roihuja, joista kukin sisältää myös valon.

Valot alkavat käyttäytyä hieman omituisesti, jos niitä on hyvin paljon lähekkäin. Unityn asetuksista pystytään säätämään pelin grafiikkaa ja laatua (QualitySettings). Suodatuksen (Rendering) asetuksissa on arvo Pixel Light Count, joka on oletuksena 10. Tätä arvoa nostamalla saadaan valojen toimintaa säädettyä. (52.) Tämä tietysti vaikuttaa suorituskykyyn, joten kannattaa testata mikä arvo on sopiva, ellei asiaan ole perehtynyt.

Valojen määrä tässä skenessä alkoi kuitenkin olla sen verran suuri, että niitä täytyi lopulta optimoida. Eihän ollut tarpeellista, että pelin yli 60 erilaista valoa olisivat päällä yhtä aikaa, puhumattakaan soihduissa ja roihuissa olevista tuliefekteistä. Huoneiden perusvalot jätettiin päälle, mutta kaikki soihduissa olevat valot ja particle systemit pistettiin kiinni. On tarpeen sytyttää kerrallaan vain ne valot, jotka sijaitsevat pelihahmon siinä huoneessa, jossa pelihahmo on. Tämä tehtiin ottamalla ensin talteen kaikki pelin soihdut listaan.

```
List<GameObject> Torches;
public void AddAllTorches(){
//Etsitään kaikki peliobjektit joilla on tagi Torch
GameObject[] torch = GameObject.FindGameObjectsWithTag("Torch");
   //Lisätään kaikki löydetyt objektit listaan
    foreach(GameObject t in torch){
                Torches.Add(t);
                }
 }
```

Aina kun astutaan huoneeseen, etsitään listasta siinä huoneessa sijaitsevat valot. Tämä tapahtuu samalla tavalla kuin vihollisten spawnpointtien etsiminen, eli koordinaattien perusteella lasketun indeksin vertailulla pelihahmon senhetkisen huoneen indeksiin. Kun oikeat valot on löydetty, kutsutaan toista funktiota, joka sytyttää valon ja saa tuliefektin päälle. Sytytettyjen soihtujen peliobjektit otetaan talteen toiseen listaan, jotta ne voidaan helposti sammuttaa kun huoneesta poistutaan.

```
 void LitThisRoomFires(GameObject fire){
//Lisätään sytytettävä valo listaan
CurrentTorches.Add(fire);
//Sytytetään valo
foreach(Light gameObj in fire.GetComponentsInChildren<Light>()){
    gameObj.enabled = true;
}
// Käynnistetään tuliefekti
foreach(ParticleSystem flame in fire.GetComponentsInChildren<ParticleSystem>()){
        flame.Play();
 }
}
```

Kun huoneesta poistutaan ja indeksi vaihtuu, tutkitaan jälleen seuraavan huoneen valot ja sytytetään ne. Edellisen huoneen valot löytyvät aina CurrentTorches-listasta ja ne sammutetaan. Kun valot on sammutettu, lista tyhjennetään.

```
//Jos listassa on valoja, ne sammutetaan
 if(CurrentTorches.Count > 0){
        TurnOfLights();
  }
void TurnOfLights(){
//sammutetaan jokainen listasta löytyvä valo ja efekti
    for(int i= 0; i < CurrentTorches.Count; i++){
        CurrentTorches[i].GetComponentInChildren<Light>().enabled = false;
        CurrentTorches[i].GetComponentInChildren< ParticleSystem >().Stop();
    }
     //Tyhjennetään lista
       CurrentTorches.Clear();
}
```

5.2.2 Pelihahmo Link

Kun pelihahmon 3D-malli ja animaatiot oli tehty, oli aika luoda hahmolle parempi kontrolleri. Tähän löytyi onneksi hyvä ohje BurgZerg Arcade Games -sivustolta, jonka pohjalta luotiin oma kontrolleri. (53.)

Ohjauksen lisäksi pelihahmolle tarvittiin skriptit terveydentilaa, hyökkäystä ja vaihtoehtoisten aseiden käyttöä varten.

Hahmon ohjaus

Ohjausta varten pelihahmoon täytyy liittää Character Controller -komponentti, jota käytetään oman kontrollerin tekemiseen (54). Komponentti sisältää esimerkiksi törmäyksen tutkimisen, eli colliderin ja muita asetuksia. Hahmon ohjaaminen on jaettu kahteen skriptiin.

Movement-skriptissä on määritetty eri tiloja, kuten juoksu, hyökkäys tai idle. Jokaiseen tilaan liittyy animaatio, jota näytetään tilan ollessa aktiivisena. Tämän lisäksi skriptissä on varsinainen pelihahmon liikuttaminen ja kääntäminen.

Character-skriptissä on pelaajan tekemien napinpainallusten tutkiminen. Liikkuminen tapahtuu nuolinäppäimiä tai näppäimistön WASD-painikkeita painelemalla. Miekkaa käytetään painamalla X-painiketta. Inventory-ruudulta voidaan valita toinen ase, jota käytetään C-painikkeella. Kun nappia painetaan, lähetetään oikeanlainen komento Movement-skriptiin, joka käynnistää animaation ja liikuttamisen.

Terveydentila

Player Health -skripti sisältää pelihahmon terveydentilan tiedot. Terveydentila esitetään sydäminä. Skriptissä on määritetty maksimimäärä sydämille sekä senhetkinen sydänten määrä. Tämä skripti pitää ajan tasalla terveydentilan muutokset. Sydänten määrää lisätään, jos luolastosta kerätään sydän, tai vähennetään, jos pelihahmo saa osuman vihollisesta. Sydänten määrä ei voi kasvaa yli määritetyn maksimiarvon, eli jos pelihahmolla on jo täydet sydämet, kerätyllä sydämellä ei ole vaikutusta.

Hyökkäys

Hyökkäystä varten olevassa skriptissä aletaan tutkia, osuuko lyönti viholliseen, kun lyömisnappia painetaan. Hyökkäykseen liittyy ajastin, joka hyväksyy painalluksen vain 2 sekunnin välein, jotta saadaan aikaa animaation suoritukselle. Osuman tutkiminen tapahtuu täysin ohjelmallisesti. Siihen käytettiin Physics-luokan funktiota OverlapSphere. (55.)

Physics.OverlapSphere(transform.position, radius);

Yllä oleva funktio toimii siten, että se havaitsee määritetyn säteen sisällä olevat tai sitä koskettavat colliderit, ja palauttaa ne taulukossa. Ensimmäinen parametri transform.position on pallon keskipiste ja radius on pallon säde. Koska tämä skripti on kiinni pelihahmossa, transform.position on siis yhtä kuin pelihahmon senhetkinen sijainti nappia painettaessa. Funktion palauttamaa taulukkoa tarkastelemalla voidaan tutkia, mitä kaikkea lyömisetäisyydellä on. Tageja käyttämällä saadaan selville, oliko lyömisetäisyydellä vihollisia.

Koska Physics.OverlapSphere on pallon muotoinen, se saa tiedon myös pelihahmon takana olevista vihollisista. Koska ei ole järkevää, että myös pelihahmon selän takana olevat viholliset saavat osuman, täytyy tutkia, onko vihollinen pelihahmon edessä. Tämä onnistuu vertailemalla pelihahmon ja lyödyn vihollisen vektoreita. Hit.transform.position sisältää lyödyn kohteen sijainnin ja transform.position pelihahmon sijainnin.

Vector3 dir = (hit.transform.position - transform.position).normalized;
float direction = Vector3.Dot(dir, transform.forward);

Osuma hyväksytään, jos direction-muuttujan arvo on suurempi kuin 0, mikä tarkoittaa sitä, että vihollinen on pelihahmon edessä. Kun vihollinen saa osuman, väläytetään sitä hetkeksi punaisena, jotta pelaaja tietäisi osuiko lyönti viholliseen (kuva 22). Tämä tehdään ottamalla talteen lyödyn vihollisen Meshin materiaalin alkuperäinen väri.

color =
hit.gameObject.GetComponentInChildren<SkinnedMeshRenderer>().renderer.material.color;

Tämän jälkeen IEnumerator-tyyppistä funktiota käyttämällä vaihdellaan väriä punaisen ja alkuperäisen välillä muutaman kerran.

KUVA 22. Kun vihollista lyödään, se välähtää punaisena merkkinä osumasta.

Vaihtoehtoisten aseiden käyttö

Pelaajan valittavissa olevia aseita varten on oma skripti. Sen jälkeen kun ne on löydetty luolastosta, valittavissa ovat pommi, jousipyssy ja bumerangi. Aseet ilmestyvät Inventory-ruutuun, josta voi valita yhden kerrallaan aktiiviseksi. Valittuna oleva ase näkyy GUIssa olevassa laatikossa. Kun C-nappia painetaan, tutkitaan mikä ase on valittuna ja kutsutaan sen mukaan oikeaa funktiota. Jokaista asetta varten käytetään prefabia.

Pommi

Kun valittuna on pommi, luodaan nappia painettaessa prefabista ilmentymä seuraavaan tapaan (56).

```
//prefab
public GameObject bomb;
public void CreateBomb(){
//luodaan pommi
Instantiate(bomb, transform.position, Quaternion.identity);
}
```

Ensimmäisellä parametrillä kerrotaan, mistä prefabista ilmentymä halutaan luoda. Koska peliobjekti bomb on public-tyyppinen, sen pystyy näkemään tarkastelupaneelissa. Tarkastelupaneelin bomb-muuttujaan täytyy raahata haluttu prefab projektipaneelista. Toisella parametrillä kerrotaan, mihin ilmentymä halutaan luoda. transform.position tarkoittaa pelihahmon sijaintia, eli pommi luodaan siihen kohtaan, jossa pelihahmo sillä hetkellä on. Quaternion tarkoittaa ilmentymän kiertokulmaa eli sitä, miten päin se asettuu pelimaailmaan. Quaternion.identity tarkoittaa, ettei sen kiertokulmaa muuteta, vaan se on sama kuin prefabissa, josta se luodaan.

Jousipyssy

Kun valittuna on jousipyssy, nappia painettaessa luodaan nuoli, joka sinkoutuu pelihahmosta suoraan eteenpäin. Luominen tapahtuu lähes samalla tavalla kuin pommissakin, mutta siihen täytyy lisäksi käyttää AddForce-funktiota, joka saa aikaan liikkeen. (57.) Jotta AddForce-funktiota voidaan käyttää, nuoleen täytyy

liittää Rigidbody-komponentti. Rigidbodyn avulla saadaan aikaan painovoima, ja muita fysiikan ominaisuuksia. (58.)

```
//prefab
public Transform arrow;
public void CreateArrow(){
//luodaan nuoli
Transform flyingArrow = Instantiate(arrow, RangedSpawnpoint.position,
                                    Quaternion.Euler(0, 90, 0)) as Transform;
```

Nuolen prefab raahataan samalla tavalla tarkastelupaneelin arrow-muuttujaan kuin pommissa. Nuolen luomista varten tehtiin tyhjä peliobjekti, joka nimettiin RangedSpawnpointiksi. Se asetettiin pelihahmon eteen sopivalle korkeudelle. Spawnpointti on pelihahmon lapsiobjekti, joten se pysyy aina samassa kohdassa suhteessa hahmoon. Tätä tyhjää objektia käytetään nuolen luontipisteenä, eli spawnpointina. Jotta nuoli saadaan lentämään oikein päin, sen Y-akselin kiertokulmaa muutetaan 90 astetta. Jos kiertokulmaa ei muuteta, nuoli ei lennä kärki edellä, vaan poikittain. Nuolen 90 asteen kiertokulman voi määrittää myös suoraan prefabissa olevan 3D-mallin Transform-asetuksista. Silloin sitä ei tarvitse muuttaa koodissa. 3D-mallien kiertokulmat määräytyvät sen mukaan miten päin ne on tehty mallinnusohjelmalla, joten ne ovat aina tapauskohtaisia. (59.)

Jotta nuoli saadaan lentämään aina pelihahmosta katsottuna 90 astetta, riippumatta siitä mihin suuntaan katsotaan, täytyy sen kiertokulmaan lisätä 90 asteen lisäksi pelihahmon senhetkinen kiertokulma.

```
//nuolen kiertokulma                    //Pelihahmon kiertokulma
flyingArrow.transform.eulerAngles += transform.rotation.eulerAngles;
```

Lopuksi nuoleen lisätään voima, joka saa sen liikkumaan suoraan eteenpäin. Rigidbodyn aikaansaama painovoima saa nuolen laskeutumaan pikkuhiljaa maata kohti sen lentäessä.

```
flyingArrow.rigidbody.AddForce(transform.forward * 1300);
```

Bumerangi

Kun valittuna on bumerangi, nappia painettaessa lähetetään lentoon bumerangi, joka kaartaa lyhyen lennon jälkeen takaisin. Myös bumerangi luodaan erikseen joka heittokerta prefabista. Pelaajan näkökulmasta bumerangeja on olemassa vain yksi, toisin kuin nuolia ja pommeja. Vain yksi bumerangi voi olla olemassa yhtä aikaa, joten pelaaja saa vaikutelman, että kyseessä on koko ajan sama esine.

```
//prefab
public GameObject boomerang;
public void CreateBoomerang(){
//Luodaan bumerangi
Instantiate(boomerang, RangedSpawnpoint.position, transform.rotation);
}
```

Bumerangin prefab raahataan tarkastelupaneelissa boomerang-muuttujaan ja luontipisteenä käytetään samaa RangedSpawnpointtia, jota käytetään nuoli-prefabille. Kiertokulmaksi annetaan pelihahmon kiertokulma, jotta lentokulma on oikea ja lähtee pelihahmosta eteenpäin. Bumerangille ei lisätty Rigidbody-komponenttia, joka olisi lisännyt painovoiman, koska haluttu lentorata ei ole realistinen.

5.2.3 Viholliset

Pelissä on useita erilaisia vihollisia, jotka liikkuvat luolaston huoneissa. Vihollisia varten tarvittiin aika lailla samanlaisia ominaisuuksia kuin pelihahmollekin, kuten terveydentila, hyökkäys ja liikkuminen. Vihollisille pystyttiin käyttämään samaa Movement-skriptiä, jota käytettiin pelihahmolla. Erotuksena oli vain se, että vihollisen liikkumisen komennot eivät tule pelaajalta, vaan sillä on jonkinlaista tekoälyä. Viholliset voivat myös jättää kuoltuaan jälkeensä aarteita, joita voi kerätä.

Terveydentila

Eri vihollisilla on keskenään hieman eri ominaisuuksia, kuten sydänten määrä. Jotta jokaista erilaista vihollista varten ei tarvitsisi tehdä vain pienten

eroavaisuuksien vuoksi kokonaan eri skriptiä, käytettiin tageja. Kun vihollinen luodaan, tutkitaan, mikä kunkin vihollisen tagi on, ja tämän mukaan sille määräytyy sydänten määrä seuraavaan tapaan.

```
//Vihollisten sydäntenmäärät tyypeittäin
if( gameObject.tag == "Keese" || gameObject.tag == "Gel"){
                maxHealth = 1;
                curHealth = 1;
 }
else if( gameObject.tag == "Stalfos"){
                maxHealth = 2;
                curHealth = 2;
}...
```

Skriptissä pidetään ajan tasalla vihollisten sydänten väheneminen ja tutkitaan, milloin vihollinen kuolee. Vihollisen kuollessa ilmaan leviää pieni savun pöllähdys. Savuefekti on tyhjä peliobjekti, johon on liitetty particle system, ja siitä on tehty prefab.

```
//luodaan savu
Instantiate(smokePuff, DiePosition, Quaternion.identity);
```

smokePuff-prefabissa on kiinni myös skripti, joka saa sen lopettamaan savun luomisen lyhyen hetken kuluttua luomisesta ja joka tuhoaa lopuksi itse itsensä.

Hyökkäys

Viholliset eivät varsinaisesti käy pelihahmon kimppuun. Ne vain liikkuvat ympäriinsä sattumanvaraisesti, kuten alkuperäisessäkin pelissä. Viholliset aiheuttavat pelihahmolle vahinkoa, mikäli mennään liian lähellä niitä. Tämä tehdään tutkimalla pelihahmon ja vihollisen välistä etäisyyttä. Myös vihollisilla on 2 sekunnin ajastin, ettei pelihahmo kuole saman tien astuessaan liian lähelle vihollista

```
//pelihahmon ja vihollisen välinen etäisyys
float distance = Vector3.Distance(target.transform.position, this.gameObject.transform.position);
if(distance <= 4.0f){
    if(attackTimer == 0) {
        attackTimer = coolDown;
```

```
Attack();    }}
```

Mikäli pelihahmon ja vihollisen välimatka on määriteltyä lyhyempi ja viime hyökkäyksestä on kulunut vähintään 2 sekuntia, hyökätään. Varsinaisessa Attack-funktiossa tutkitaan vielä, oliko pelihahmo vihollisen edessä, eli osuuko lyönti. Jos pelihahmo saa osuman, se vilkkuu hetken ajan läpinäkyvästä näkyvään. Tämä tehdään asettamalla pelaajan MeshRenderer falseksi ja trueksi vuoronperään.

Tekoäly

Vihollisia varten määriteltiin 5 eri tilaa: liikkuminen, kääntyminen vasemmalle, kääntyminen oikealle, paikallaan pysyminen ja kuoleminen.

```
//vihollisten tilat
public enum State {

 Move,
 TurnLeft,
TurnRight,
Idle,
Die,
 }
```

Skriptissä on ajastin, joka kutsuu tietyn väliajoin funktiota, joka arpoo neljästä eri vaihtoehdosta tilan ja vaihtaa sen. Tilan vaihto saa vihollisen pysähtymään, kävelemään tai kääntymään johonkin suuntaan. Funktiossa on kuitenkin rajattu, että jos vihollisen tila on viimeksi ollut kääntyminen, uuden tilan täytyy olla joko idle tai käveleminen. Tilan vaihdon yhteydessä määritetään myös uusi aika ajastinta varten, joka kutsuu funktiota seuraavan kerran. Kävely tai Idle kestää pari sekuntia, kunnes tilaa taas vaihdetaan. Kääntymiseen varattu aika on reilusti lyhyempi, vain 0,3 sekuntia, jottei vihollinen ehdi pyöriä itsensä ympäri useampaa kertaa.

Vihollisen tila vaihdetaan kuolevaksi, kun se on menettänyt kaikki sydämensä. Tätä varten vihollisen terveydentila-skriptissä on boolean-muuttuja isAlive. Kun vihollinen kuolee, arvotaan tiputtaako se aarteita vai ei. Kuoleminen tapahtuu pienellä viiveellä, koska ennen sitä täytyy tehdä asioita tietyssä järjestyksessä.

Esimerkiksi luodaan esine, jonka se mahdollisesti tiputtaa. Lisäksi joissain huoneissa vihollisten kuolema aiheuttaa jonkin tapahtuman, ja tämä täytyy ehtiä tutkia ennenkuin vihollinen tuhotaan. Lopuksi savuntuprahdus-efekti täytyy luoda, ennen kuin vihollisen objekti tuhotaan.

Aarteet

On mahdollista, että vihollinen pudottaa esineen kuoltuaan. Kun vihollinen kuolee, käytetään yksinkertaista arvontaa kahden luvun väliltä määrittämään, luodaanko esinettä vai ei. Jos tulokseksi saadaan, että se tiputtaa jotain, arvotaan lisäksi, mikä vihollisen jättämä esine on. Myös esineen valintaan käytetään arvontaa. Mahdolliset aarteet on asetettu taulukkoon ja arvonnalla valitaan taulukon indeksi.

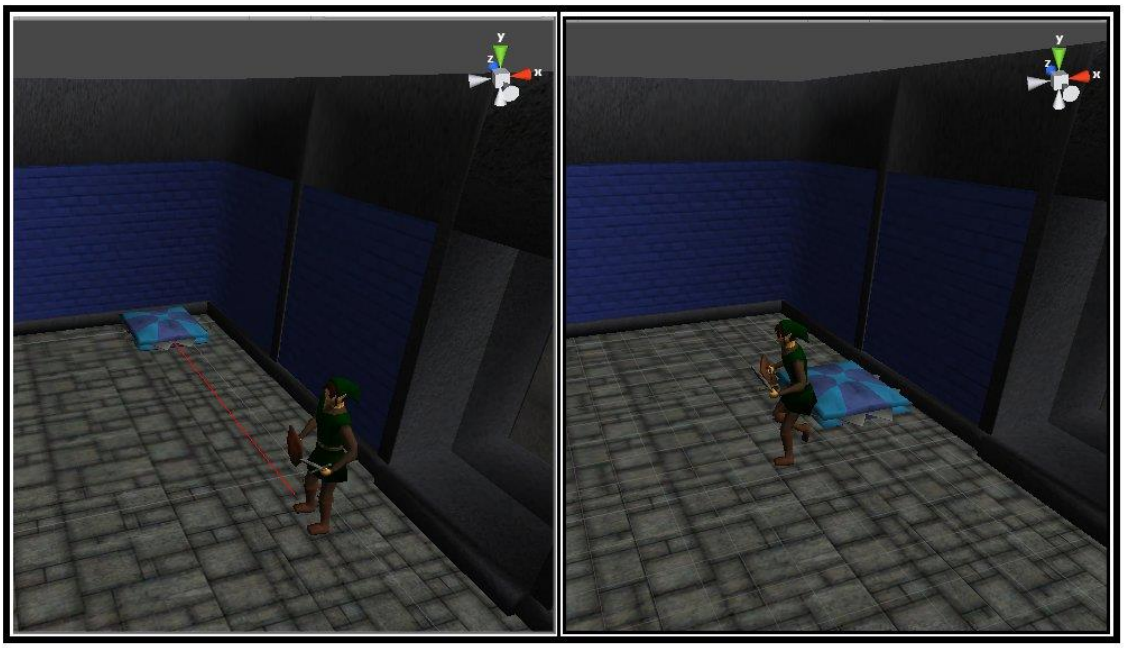
```
//Kaikki mahdolliset  esineet
items[0] = rupee;
items[1] = bluerupee;
items[2] = bomb;
items[3] = heart;
items[4] = key;
//Arvotaan ja luodaan esine
IEnumerator CreateItem(){
int number = Random.Range(0, 4);
Instantiate(items[number], LootSpawnPosition, Quaternion.identity);
}
```

Valitussa indeksissä oleva esine luodaan siihen kohtaan, johon vihollinen kuolee. Vaihtoehtoina on rahaa, pommi ja sydän. Osa vihollisista on sillä tavalla erikoisia, että ne tiputtavat aina avaimen. Myös tämä ominaisuus tutkitaan tagin avulla. Jos vihollisen tagi on ”EnemyWithKey”, sen kuollessa luodaan aina avain.

Ansat

Pelissä on tavallisten kävelevien vihollisten lisäksi ansoja (Blade trap). Ansoilla ei ole terveydentilaa, joten niitä ei voi tuhota ja niillä on täysin omanlaisensa toiminta. Ansat ovat paikoillaan huoneen nurkissa, ja kun ne havaitsevat pelihahmon kulkevan tietyn kohdan yli, ne lähtevät liikkeelle (kuva 23). Ne

käyvät määritetyssä pisteessä asti ja palaavat sen jälkeen takaisin paikoilleen. Ansat pystyvät kulkemaan vain kahteen suuntaan, suoraan eteenpäin ja sivulle.



KUVA 23. Ansa havaitsee pelihahmon kulkevan tietyn linjan yli ja lähtee hyökkäykseen. Kuva on skenenäkymästä, joten siinä näkyy punainen viiva, joka on debuggausta varten. Oikeassa pelissä linja on näkymätön.

Liikkeen havainnointia varten käytettiin Physics-luokan Raycastia. (60.) Se luo määritellystä pisteestä näkymättömän säteen määriteltyyn suuntaan. Jos tällä linjalla havaitaan toinen objekti eli sen collider, tästä saadaan tieto. Tarkempia tietoja havaitusta objektista voidaan saada out-parametria käyttämällä.

```
//Suuntavektori
Vector3 fwd = transform.TransformDirection(Vector3.forward);
if (Physics.Raycast(transform.position, fwd, out hit, 14){
        //Jos havaittu objekti oli pelihahmo eli Link, lähdetään liikkeelle
        if(hit.collider.name == "Link"){
                direction = "forward";
                attack = true;
        }}
```

Pelissä on vain neljä ansaa, ja ne sijaitsevat yhden huoneen jokaisessa nurkassa. Ansoja varten on oma skripti, joka saa ne tarkkailemaan tiettyä linjaa ja liikkumaan, jos ne havaitsevat pelihahmon. Skriptin Start-metodissa tutkitaan

peliobjektin nimen perusteella, mistä ansasta on kyse, jotta saadaan määriteltyä sen liikkumisen suunnat. Jokaisen ansan alkusijainti otetaan talteen ja sille määritellään loppusijainti eli lähtöpiste, mistä aletaan liikkua, ja loppupiste, mihin liikutaan. Jotta ansa osaa liikkua oikeaan suuntaan, otetaan talteen millä suunnalla se havaitsi pelaajan. Sen takia on määritelty direction-muuttuja, joka voi olla joko eteenpäin, oikealle tai vasemmalle. Kun hyökkäys käynnistyy, ansaa liikutetaan kohti laskettua määränpäätä vertailemalla ansan ja lasketun määränpään välistä etäisyyttä. Seuraavat koodinpätkät ovat skriptin Update-metodista, eli sitä kutsutaan kerran framessa.

```
//verrataan ansan ja lasketun loppupisteen välistä matkaa
if(Vector3.Distance(trapTransform.position,targetPositionForward) > 0.5){
        //liikutetaan ansaa eteenpäin
trapTransform.position += trapTransform.forward * 10.0f * Time.deltaTime;
}
```

Kun ansa on saavuttanut määränpäänsä, se liikkuu takaisin paikalleen vähän hitaammin.

```
//verrataan ansan ja alkuperäisen sijainnin välistä matkaa
if(Vector3.Distance(trapTransform.position, originalPosition) > 0.5){
    //liikutetaan ansaa taaksepäin
trapTransform.position -= trapTransform.forward * 7.0f * Time.deltaTime;
}
```

Ansa ei lähde uuteen hyökkäykseen ennen kuin se on käynyt määränpäässään ja palannut takaisin entiselle paikalleen. Tämän toiminnan aikaansaamiseksi tarvittiin boolean-muuttuja, jolla nähdään onko ansa hyökkäämässä ja toinen boolean-muuttuja tutkimaan, milloin ansa on palaamassa paikalleen. Jokaiselle kolmelle suunnalle on omat ehtonsa. Tämän lisäksi skriptissä käytetään jälleen OnTriggerEnter-funktiota, joka havaitsee osuuko ansa pelihahmoon. Jos ansa osuu, aiheutuu vahinkoa.

5.2.4 Loppuvastus

Loppuvastuksella on sama terveydentilaa tutkiva skripti kuin muillakin vihollisilla. Myös hyökkäystä varten oleva skripti on sama, mutta siinä on oma ehtonsa loppuvastusta varten, jossa verrataan pelihahmon etäisyyttä

lohikäärmeen päähän. Lisäksi skriptissä on lohikäärmettä varten oma funktionsa, joka saa sen ampumaan tulipalloja (kuva 24). Tulipallon luominen tapahtuu samaan tapaan, kuin jousipyssyllä ammuttava nuoli. Se luodaan prefabista ja sille on oma luontipiste suoraan lohikäärmeen pään edessä. Tulipalloon käytetään myös RigidBody-komponenttia ja Addforce-funktiota, joka saa sen sinkoutumaan eteenpäin. Tulipallo on vain pallo, johon on laitettu materiaali, joka saa sen pintaan tulitekstuurin. Lisäksi tulipallossa on particle system, jolla tehdään tuliefekti. Tulipallossa on skripti, joka tutkii sen osumista pelihahmoon OnTriggerEnter-funktiolla. Jos tulipallo osuu, pelihahmolta vähennetään sydän. Skriptissä on myös ajastin, joka saa tulipallon tuhoamaan itsensä tietyn ajan kuluessa.

Loppuvastustuksen tekoälyä varten on oma skripti ja se ei käytä Movement-skriptiä, kuten muut viholliset. Loppuvastus ei kävele, joten sillä on vain kolme tilaa: paikallaan pysyminen, hyökkäys ja kuoleminen. Tilojen vaihtelu tehdään samaan tapaan, kuin tavallisilla vihollisillakin. Erona tietysti se, että siinä arvotaan vain, milloin lohikäärme hyökkää.

KUVA 24. Loppuvastuksena oleva lohikäärme syöksee tulipalloja.

Loppuvastuksen kuollessa luodaan lisäsydän, eli HeartContainer, jonka kerättyään pelaajan maksimisydänten määrä kasvaa yhdellä.

5.2.5 Esineet

Pelissä on monenlaisia esineitä, kuten erilaisia aseita ja luolastossa etenemiseen tarvittavia esineitä. Avaimet mahdollistavat lukituista ovista kulkemisen, rahalla voi ostaa luolastosta löytyvältä vanhalta mieheltä pommeja ja sydämiä. Kartta ja kompassi auttavat luolastossa navigointia. Lisäksi ovat erikoisesineet, lisäsydän ja kolmivoima. Esineisiin liittyy kaksi muuta skriptiä, GUI-skripti ja inventoryScreen-skripti.

Esineiden kerääminen

Jokaisella pelin esineellä on sama skripti, joka saa aikaan sen, että pelaaja voi kerätä sen. Kun pelaaja koskettaa esineen collideria, tutkitaan OnTriggerEnter-funktiolla, mikä on esineen tagi ja toimitaan sen mukaan.

```
void OnTriggerEnter(Collider character)
{       //Jos esineeseen osui pelihahmo
        if (character.tag == "Player"){
            //Jos esine on avain
             if (itemTag == "Key"){
                    //Tuhotaan avaimen peliobjekti
                    Destroy(gameObject);
                    //Lisätään avainten määrää yhdellä
                    guiscript.key++;
                    //Jos esine on sydän
            } else if (itemTag == "Heart"){
                    //Lisätään pelihahmon terveyttä kokonaisella sydämellä
                    playerHealth.AddjustCurrentHealt(2);
                    //Tuhotaan sydämen peliobjekti
                    Destroy(gameObject);
            }...
```

Kun pelaaja osuu esineeseen, esine tuhotaan pelikentältä. Näin saadaan vaikutelma, että pelaaja poimi esineen. Eri esineiden kohdalla toimitaan hieman eri tavoin. Rahan, pommien ja avainten kohdalla niiden määrää lisäksi

kasvatetaan. Pommi, jousipyssy ja bumerangi ovat käyttöesineitä, jotka ilmestyvät pelaajan inventoryyn, kun ne on kerätty. Niiden lisäämistä varten on oma funktio inventoryScreen- skriptissä. Samalla tavalla lisätään myös kartta ja kompassi, vaikka ne eivät ole varsinaisia käyttöesineitä vaan ne toimivat automaattisesti.

```
//Jos esine on jousipyssy
else if(itemTag == "Bow"){
    //Tuhotaan jousipyssyn peliobjekti  ja lisätään se pelaajan Inventoryyn
            Destroy(gameObject);
            inventory.AddItem(itemTag);}
```

Sydänsäiliö on erikoispalkinto, joka saadaan loppuvastuksen päihittämisestä. Se kasvattaa pelaajan maksimisydänten määrää yhdellä. Peli loppuu siihen, kun pelaaja saa haltuunsa kolmivoiman. Kolmivoiman kerääminen lopettaa pelin ja käynnistää pidemmän tapahtumasarjan.

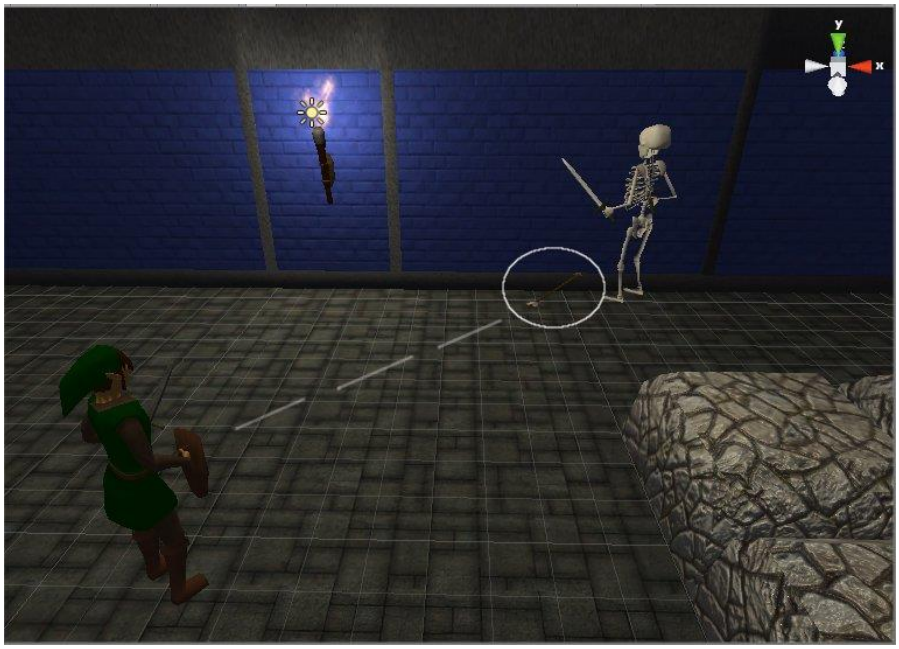
Pommin toiminta

Pommiin liittyy kaksi eri prefabia, varsinainen pommi ja räjähdys. Kun pommi luodaan, se luo samaan kohtaan, jossa se itse on, toisesta prefabista räjähdyksen, joka on ajastettu näyttämään tuliefekti oikeaan aikaan (kuva 25). Varsinaisessa pommiskriptissä tutkitaan Physics.OverlapSphereä käyttämällä, mitä pommin vaikutusalueella on. Jokainen vaikutusalueella oleva vihollinen saa vahinkoa. Jos pommin vaikutusalueella on seinä, jonka voi tuhota, lähetetään myös sille komento tuhoutua. Pommien määrää on rajattu, ja niitä voi luoda vain niin monta, kuin pelaajalla on niitä hallussaan. Pommin luominen vähentää niiden määrää yhdellä.

KUVA 25. Pommi räjähtää muutaman sekunnin kuluttua sen asettamisesta.

Jousipyssyn toiminta

Kun jousipyssyä käytetään, se luo nuolen joka sinkoutuu suoraan pelaajasta eteenpäin (kuva 26). Nuolessa on oma skripti, joka tutkii onTriggerEnter-funktiota käyttämällä, osuuko se vihollisiin. Jos osuu, vihollinen ottaa vahinkoa. Nuolia voi olla ilmassa monta yhtä aikaa ja niiden määrää ei ole rajattu, alkuperäisestä pelistä poiketen.



KUVA 26. Pelaaja ampuu kohti Stalfosta jousipyssyllä.

Bumerangin toiminta

Bumerangin toimintaan liittyy kaksi eri skriptiä. Toinen skripti hoitaa bumerangin lentoradan laskemisen ja liikkumisen, toinen taas tutkimisen, osuuko se vihollisiin. Bumerangi asetettiin tyhjän peliobjektin sisään, koska tarvittiin animaatio, joka pyörittää sitä itsensä ympäri. Pyöriminen sotkisi bumerangin lentoradan, joten skripti, joka hoitaa liikkumisen on laitettu kiinni tähän tyhjään peliobjektiin. Animaatio taas on tyhjän peliobjektin sisällä olevassa varsinaisessa esineessä. Näin animaatio ja lento eivät vaikuta toisiinsa millään tavalla. Lentoradan laskemiseen käytetään Ray-luokan GetPoint-funktiota. (61.)

```
public Ray ray;
Vector3 targetpoint;


void Start () {
    //Luodaan Ray-luokasta olio, ray. Parametreina lähtöpiste ja suuntavektori.
      ray = new Ray(transform.position, player.forward);
      // Otetaan talteen pelaajan edessä määritetyn matkan päässä oleva piste.
      targetpoint = ray.GetPoint(10);
```

Bumerangi luodaan joka kerta nappia painettaessa prefabista. Luontipiste on suoraan pelaajan edessä. Heti kun bumerangi luodaan, lasketaan piste, johon sen halutaan lentävän. Ensimmäinen parametri on lähtöpiste, eli tässä tapauksessa bumerangin sijainti luontihetkellä. Toinen parametri on suuntavektori. Suunnan halutaan olevan aina suoraan pelihahmosta eteenpäin, riippumatta siitä mihin suuntaan katsotaan. GetPoint-funktiolla otetaan talteen Vector3-tyyppiseen muuttujaan piste, joka sijaitsee pelaajasta suoraan eteenpäin katsottuna. Parametrina oleva numero tarkoittaa, kuinka kaukaa lähtöpisteestä tämä uusi piste otetaan talteen.

```
//Tutkitaan bumerangin ja lasketun määränpään välistä etäisyyttä
if(Vector3.Distance(gameObject.transform.position,targetpoint) > 1.0){
//Liikutetaan bumerangia kohta määränpäät
gameObject.transform.position += gameObject.transform.forward * 12.0f * Time.deltaTime;
}
```

Bumerangia liikutetaan kohti laskettua määränpäätä niin kauan, kunnes välimatka on tarpeeksi pieni. Tämän jälkeen bumerangin täytyy palata takaisin pelaajan luo (kuva 27). Bumerangia aletaan kääntää kohti pelaajaa.

```
//Käännetään bumerangia,  parametrina mistä kulmasta mihin käännetään
transform.rotation = Quaternion.Slerp(transform.rotation,
Quaternion.LookRotation(player.position - transform.position), 4.0f * Time.deltaTime);
```

Samalla sitä liikutetaan kohti pelaajaa.

```
//Liikutetaan bumerangia kohti pelaajaa.
if(Vector3.Distance(gameObject.transform.position, player.position) > 2.0){
gameObject.transform.position += gameObject.transform.forward * 12.0f * Time.deltaTime;
}
```

KUVA 27. Bumerangin lentorata.

Pelihahmolla pystyy liikkumaan bumerangin ollessa lennossa, ja bumerangi seuraa hahmoa, kunnes se saavuttaa hänet. Tämän jälkeen bumerangi tuhotaan. Uutta bumerangia ei voi heittää ennen kuin edellinen on palannut takaisin. Toinen skripti, joka on kiinni varsinaisessa bumerangissa, tutkii

OnTriggerEnter-funktiolla osuuko bumerangi vihollisiin. Jos osuu, vihollinen vahingoittuu.

5.2.6 GUI ja Inventory

Peliruudun yläosassa näkyy koko ajan pelaajan sydänten määrä, hallussa olevien pommien, rahan ja avainten määrä, pieni kartta, mikäli sen on löytänyt luolastosta, ja sillä hetkellä aktiivisena oleva ase (kuva 28). Välilyöntiä painamalla avautuu myös inventory-ikkuna, josta näkee, missä luolaston huoneissa on käyty sekä missä huoneessa ollaan sillä hetkellä. Ikkunassa näkyy myös, mitä esineitä luolastosta on löytänyt. Vaihtoehtoisen aseen voi vaihtaa aktiiviseksi nuolinäppäimiä painamalla.

GUI

Pelin GUIta varten on oma skriptinsä, joka on kiinni pelin kamerassa. Lisäksi tehtiin oma GuiSkin, joka ylikirjoittaa oletuksena olevan. Oman GUISkinin avulla voidaan luoda omia tyylejä tarpeen mukaan. (62.)

KUVA 28. Pelaajan keräämien esineiden määrä sekä terveydentila näkyvät koko ajan pelin GUIssa. Kartta ja loppuvastuksen sijainti tulevat näkyviin, kun kartta ja kompassi on löydetty luolastosta.

Pelin GUI on 200 pikseliä korkea alue peliruudun ylälaidassa, jonka sisään on aseteltu tiedot, jotka halutaan kertoa pelaajalle. GUI.BeginGroupia käytetään helpottamaan asettelua. Groupin sisällä olevat koordinaatit ovat suhteessa siinä määriteltyyn alueeseen. Ilman Groupia ne olisivat suhteessa näyttöön. Parametreinä ovat x, y, leveys, korkeus, teksti ja lopuksi tyyli. Tämän alueen taustana käytetään GUITausta-nimistä tyyliä, joka on määritelty omassa GUISkinissä. Se sisältää GUI:n taustakuvan.

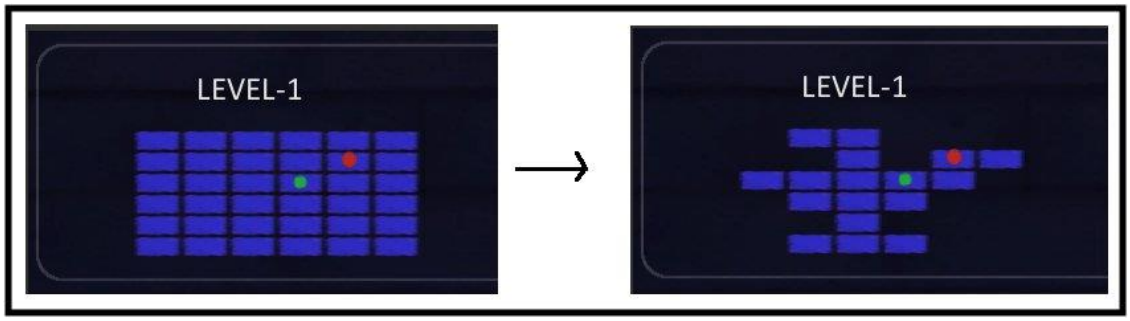
```
//Luodaan group, jossa käytetään omaa tyyliä.
GUI.BeginGroup (new Rect (0, 0, Screen.width, 200),"","GUITausta");
```

Rahaa, pommeja ja avaimia varten on omat labelinsa, joissa näytetään esineiden määrä, sekä ikoni, joka kertoo, mistä esineestä on kyse.

```
GUI.Label(new Rect(GuiMapWidth , 30, 40, 40),"" ,"RupeeIcon");
GUI.Label(new Rect(GuiMapWidth + 50, 35, 40, 40),"X" + money);
```

Aseita varten piirretään kaksi laatikkoa, joiden ylälaidassa lukee mistä napista sen sisällä olevaa esinettä käytetään. Toisen laatikon sisään piirretään aina miekka. Toisen laatikon sisään vaihdetaan kuva sen mukaan, mikä on valittuna Inventory-ruudussa.

GUIssa olevaa pientä karttaa varten täytyi tehdä oma 6 x 6 ruudukko, joka on vastaavanlainen kuin kameraa varten tehty. Pelaajan liikkumista varten on aina tiedossa, mikä on sen huoneen indeksi, jossa hän sijaitsee. Tätä tietoa käytetään hyväksi myös kartassa. Pelihahmon paikan kertova vihreä piste piirretään indeksin osoittamaan paikkaan, joka on vastaava kuin pelihahmon todellinen sijaintinsa luolastossa. Aluksi piirretään siis pelkkä vihreä piste, ei varsinaisia huoneita. Jotta saadaan piirrettyä huoneet oikeisiin kohtiin, tarvitaan tyhjä 36-paikkainen taulukko. Kun pelaaja on löytänyt kartan, lisätään taulukon huoneita vastaaviin indekseihin numero 1. Nämä ovat samat indeksit, jotka nähdään kuvassa 18. Ehdossa tutkitaan, löytyykö taulukon paikasta numero 1, Jos se löytyy, piirretään siihen kohtaan huone (kuva 29). Muussa tapauksessa ei piirretä.



KUVA 29. Kartan piirtämisen logiikka. Ensimmäisessa kuvassa 36-paikkainen ruudukko. Kun lisätään ehto, että piirretään vain ne huoneet, jotka on indeksoitu, saadaan jälkimmäinen kartta, joka vastaa todellisen luolaston pohjapiirrosta. Kuvassa näkyvät myös pelaajan ja loppuvastuksen sijainnit.

```
// Piirretään 6 x 6 ruudukko
for(int j = 0 ; j < Karttarivilaskuri; j++){
   for(int i=0; i<Karttakolumnilaskuri; i++){
      // Piirretään huone, jos huonetaulukosta löytyy arvo 1
       if(GUIMapRooms[Karttaindeksilaskuri] == 1) {
      GUI.Label(new Rect(x + (36 * i), y - (16 * j), 36, 16), "" , "Room");
      //Piirretään vihreä pallo siihen indeksiin kartassa, jossa pelaaja on
      if(gameObject.GetComponent<LookAtCamera>().CurrentRoom == Karttaindeksilaskuri){
      GUI.Label(new Rect(x + 10 + (36 * i), y - (16 * j), 15, 15), "" , "PlayerSpot");
       }
      }
     else { // Ei piirretä huoneita
```

Pelaajan terveydentilaa indikoivat sydämet piirretään myös GUIssa ja ne pysyvät ajan tasalla, koska funktiota kutsutaan jatkuvasti. Sydämiä varten on kaksi eri for-looppia. Ensimmäinen looppi piirtää tyhjien sydänten ääriviivat. Toinen looppi piirtää samoihin kohtiin kokonaiset sydämet. Jos pelaaja menettää sydämen, kokonaista sydäntä ei enää piirretä ja jäljellä jää vain ääriviivat.

```
for(int i=0; i< drawmaxHearts ; i++ )
{
//piirretään sydänten ääriviivat
GUI.Button(new Rect(x + (50 * i), y ,60,60), (Texture2D)Resources.Load("EmptyHeart"));
}
```

```
//piirretään kokonaiset sydamet
for(int j=0; j < drawCurrentHearts; j++ ){
GUI.Button(new Rect(x + (50 * j), y ,60,60), (Texture2D)Resources.Load("heart"));
}
```

Pelaaja voi menettää myös puolikkaan sydämen ja puolikasta sydäntä varten on oma looppinsa. Maksimisydänten määrä esitetään tätä varten puolikkaina, joten maksimimäärä on 6 puolikasta. Jos nykyinen sydänten määrä jaetaan kahdella ja sen jakojäännös on jotain muuta kuin 0, piirretään puolikas sydän oikeaan kohtaan.

```
//Piirretään puolikkaat sydämet
if(CurrentHearts %2 != 0){
GUI.Button(new Rect(x + (drawCurrentHearts * 50), y ,60,60),    (Tex-
ture2D)Resources.Load("halfOfheart"));
}
```

Inventory

Inventory-ikkuna avautuu kun painetaan välilyöntiä, ja se toimii myös pelin keskeyttäjänä. Kun inventory-ikkuna on auki, viholliset eivät liiku. Inventory luodaan windowRectiä käyttämällä. Ruudun ylälaidassa on valikko, jossa näkyy kerätyt aseet. Niistä voidaan valita aktiivisena oleva liikuttamalla kursoria nuolinäppäimillä (kuva 30).

KUVA 30. Inventory-ikkuna. Punainen kursori määrittää mikä esine on valittuna. Valittu esine piirretään erikseen myös vasemmalla näkyvään ruutuun. Karttaan on piirretty ne huoneet, joissa pelaaja on vieraillut.

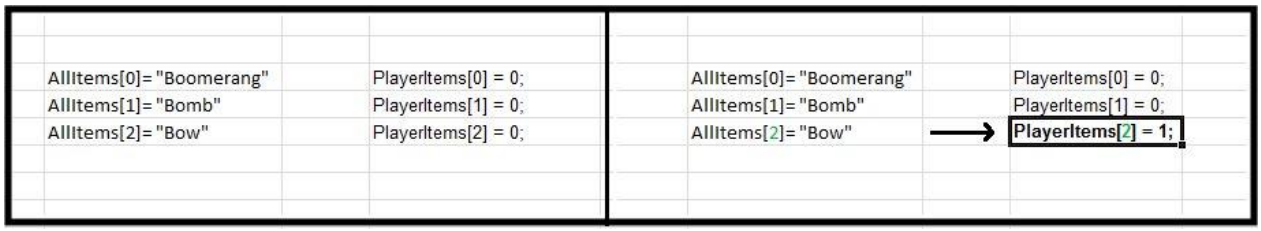
Kursorin liikuttelua varten täytyy laskea, missä kohti kursori on milläkin hetkellä ja mikä on sen ruudun indeksi. Kun inventory on auki ja painetaan nuolinäppäintä oikealle, liikutetaan kursorin kuvaketta 100 pikseliä oikealle. Tämän jälkeen lasketaan koordinaattien perusteella, mikä on tämän kohdan indeksi. Kursorin liikuttelu on estetty, mikäli sen arvo on menossa yli 100:n. Samantyyppinen ehto on myös vasempaan päin mentäessä. CurrentIndex-muuttujassa on tallessa, mikä indeksi on valittuna valikosta.

```
float cursorHeight = 100;
Update(){
//Jos inventory-ikkuna on auki
if(showInventory){
//Jos  painetaan nuolinäppäintä oikealle
if(Input.GetKeyUp(KeyCode.RightArrow)){
 //Jos indeksi ei ole jo viimeisessä paikassa
    if(indexX <= 100){
      //Siirretään kursoria oikealle
         indexX += 100;
```

```
        cursorX += 100;
    //Lasketaan koordinaattien perusteella, mikä on paikan indeksi
CurrentIndex = (int)(CalculateIndex(System.Math.Floor(indexX / curso-
rHeight),System.Math.Floor(indexY / cursorHeight)));
}...
```

InventoryScreen-skriptissä pidetään kirjaa siitä, mitä esineitä pelaaja on saanut haltuunsa. Tätä varten on luotu kaksi taulukkoa. Toinen on string-tyyppinen ja sisältää kaikki pelissä olevat aseet. Toinen on int-tyyppinen, joka sisältää aseet, jotka pelaaja on kerännyt. Näitä kahta taulua vertaillaan niin, että jos pelaajan esineet-taulusta löytyy arvo 1 vastaavasta indeksistä, kuin pelin aseet-taulusta, hän omistaa esineen (kuva 31).

Esineen lisäämistä varten skriptissä on AddItem-funktio, jota kutsutaan kun pelaaja kerää esineen. Funktiolle lähetetään parametrina tieto, minkä esineen pelaaja on kerännyt.



KUVA 31. Esimerkki pelaajan saamasta esineestä. PlayerItems taulukon paikasta 2 löytyy arvo 1. Se tarkoittaa, että hänellä on hallussaan AllItems-taulukon paikasta 2 löytyvä esine eli tässä tapauksessa jousipyssy.

AddItem-funktio etsii AllItems-taulukosta saadun esineen indeksin. Tämän jälkeen pelaajan esineet-taulukkoon asetetaan arvo 1 tähän samaan indeksiin.

```
public void AddItem(string item){
//Etsitään kaikista pelin esineistä saadun esineen indeksi
for(int i = 0; i< AllItems.Length; i++){

        if(AllItems[i] == item){
            //Merkataan saatu esine pelaajalle.
            PlayerItems[i] = 1;
        }
}
```

AddItem-funktiossa on omat ehtonsa kartalle ja kompassille. Niitä ei lisätä taulukkoon, vaan niitä varten on vain boolean-muuttuja, joka asetetaan trueksi, jos pelaaja on löytänyt ne luolastosta. Syy taulukon käyttöön tietyillä esineillä on se, että pelaaja voi valita, mitä asetta hän haluaa käyttää, ja se täytyy piirtää erikseen toiseen ruutuun. Täytyy siis olla tiedossa, mikä esine on valittuna, jotta sen voi piirtää, ja tämä tieto saadaan helposti indeksejä tutkimalla.

```
//Jos pelaajalla on tämä esine hallussaan ja se on valittuna
if(PlayerItems[CurrentIndex] == 1){
//Piirretään kyseisen esineen kuva valittu-ruutuun
GUI.Label(new Rect(neljasOsaruudusta + 75, 45, 80, 80), "", AllItems[CurrentIndex]);
//Otetaan talteen valittuna oleva esine
EquippedItem = AllItems[CurrentIndex];
}
Jos esinettä ei ole hallussa, ei piirretä mitään
else{
GUI.Label(new Rect(neljasOsaruudusta+ 70, 45, 60, 60), "");
EquippedItem = "";
}
```

Valikon alla on paikat kartalle ja kompassille, ja kun esineet on löydetty luolastosta, ne piirretään omille paikoilleen inventory-ikkunaan. Inventory-ikkunassa näkyy myös kartta, johon piirtyy huoneita sitä mukaa, kun pelaaja on vieraillut niissä. Lisäksi piirretään vihreä piste kertomaan senhetkinen sijainti. GUIn ja inventoryn välisen kartan ero on siinä, että toinen näyttää koko luolaston huoneet ja toinen vain ne huoneet, joissa pelaaja on käynyt. Näin pelaaja voi tarkastella, missä huoneissa hän ei ole vielä käynyt.

Kompassin löytymisen jälkeen kartalle piirtyy punainen piste siihen kohtaan, josta voidaan löytää pelin loppuvastus.

5.2.7 Vanha mies

Yhdessä pelin huoneessa on vanha kaapuun pukeutunut mies, joka antaa vinkin kolmivoiman sijainnista luolastossa. Tähän huoneeseen johtavan oven saa auki työntämällä kiveä ja se on ikään kuin salainen huone. Vanhalla miehellä ei sinänsä ole mitään isoa roolia, mutta alkuperäisessä pelissä

pelihahmo saa heti pelin alussa miekan kaapuun pukeutuneelta mieheltä, ennen kuin luolastoon tullaan. Tämän lisäksi alkuperäisessä pelissä on useita kauppoja, joista pystyy ostamaan tavaraa. Koska tämä peli on rajattu ainoastaan ensimmäiseen luolastoon, pelaaja ei pääse näkemään näitä asioita. Lisäksi raha olisi ilman kauppaa tässä pelissä tarpeetonta. Tämän vuoksi haluttiin tässä pelissä vanhalle miehelle antaa vähän isompi rooli sekä rahalle käyttötarkoitus, joten häneltä voi vinkin saamisen lisäksi ostaa sydämiä tai pommeja (kuva 32).

KUVA 32. Vanhan miehen kauppa.

Vanhan miehen lähelle on asetettu triggeri, jolla havaitaan milloin pelihahmo tulee hänen luokseen. Triggeri luo pelin alalaitaan ikkunan, jossa pystytään nuolinäppäimillä valitsemaan jokin kolmesta vaihtoehdosta. Ikkuna luotiin GUI-luokan Window-funktiolla (63).

```
void OnTriggerEnter(Collider character)
{
    //Pelaaja saapuu vanhan miehen luo
    if(character.tag == "Player"){
```

```
        //Pysäytetään hahmon normaalit toiminnallisuudet ja piirretään valikko-ikkuna
        Link.GetComponent<CharacterScript>().isTalking = true;
        Link.GetComponent<CharacterScript>().StopMovement();
        StartCoroutine("activateMenu");
      }
}
```

Pelihahmon liikkuminen täytyy pysäyttää, koska juoksuanimaatio voi jäädä päälle ikkunan avautuessa. IsTalking-muuttujan ollessa true estetään kaikki pelihahmon normaalitoiminnot, kuten esineiden käyttö ja liikkuminen.Tällöin nuolinäppäimet sallivat ainoastaan valikossa liikkumisen ja painikkeilla C ja X, jotka ovat normaalisti aseiden käyttöön varattuja, vahvistetaan haluttu vaihtoehto. Nuolinäppäimiä painettaessa siirrytään valikossa alas ja ylöspäin.

```
if(Input.GetKeyUp(KeyCode.DownArrow)){
                //Kun painetaan alas-nuolinäppäintä, indeksiä muutetaan
                  if(CurrentChoice == 2){
                 CurrentChoice = 0;
                }
                else
                CurrentChoice++;
```

Vaihtoehtoina olevat tekstit lukevat string-tyyppisessä Choises-taulukossa. GUI-funktiossa piirretään valitun vaihtoehdon teksti tyylillä ShopText, joka saa aikaan punaisen tekstin.

```
if(CurrentChoice == 0){
Valittuna olevan vaihtoehdon fontti on punainen, muuten valkoinen
GUI.Label(new Rect(110, 60, 200, 30), Choices[0], "ShopText");
GUI.Label(new Rect(110, 90, 200, 30), Choices[1]);
GUI.Label(new Rect(110, 120, 200, 30), Choices[2]);
}
else if(CurrentChoice == 1){
GUI.Label(new Rect(110, 60, 200, 30), Choices[0]);
GUI.Label(new Rect(110, 90, 200, 30), Choices[1], "ShopText");
GUI.Label(new Rect(110, 120, 200, 30), Choices[2]);
}…
```

Valinta vahvistetaan C- tai X-painikkeella valitun vaihtoehdon kohdalla. Pommeja ostettassa täytyy tarkastaa, onko pelaajalla ennestään pommeja hallussaan. Jos ei ole ja hän saa ensimmäisen pomminsa ostamalla, täytyy pommi lisätä myös inventoryyn, jotta niitä voi käyttää.

```
if(Input.GetKeyUp(KeyCode.X) || Input.GetKeyUp(KeyCode.C)){
//Osta sydän
if(CurrentChoice == 0){
if(guiscript.money > 2){
guiscript.money -= 2;
Link.GetComponent<PlayerHealth>().AddjustCurrentHealt(2);
audioscript.PlaySound("GetHeart");
}}
//Osta pommi
else if(CurrentChoice == 1){
        if(guiscript.money > 3){
                if(guiscript.bomb == 0){
                inventory.AddItem("Bomb");
                }
         guiscript.bomb++;
         guiscript.money -= 3;
         audioscript.PlaySound("GetItem");
…
```

Kolmas vaihtoehto Leave sulkee kauppa-ikkunan ja vapauttaa hahmon taas liikkumaan ja käyttämään esineitä normaalisti.

5.2.8 Äänet ja musiikki

Netistä löytyy useita sivustoja, jotka sisältävät vanhojen pelien ääniefektejä ja musiikkia. Googlettamalla löytyi myös pelkästään Zelda-peleille omistettu sivusto, josta ladattiin ensimmäisen Zelda-pelin ääniefektit:

http://noproblo.dayjo.org/ZeldaSounds/.

Pelin varsinainen soundtrack löytyi toiselta Zelda-sivustolta.

http://www.zeldauniverse.net/music/zelda-soundtracks/the-legend-of-zelda-original-soundtrack/

Kun äänitiedostot oli ladattu, ne siirrettiin Unity-projektiin. Pelin ääniä ja musiikkia varten tehtiin oma Audio-skripti, joka on kiinni pelin kamerassa. Jokaista pelin ääntä varten täytyi tehdä oma AudioSource, jotka asetettiin tyhjiin peliobjekteihin. Näin useita ääniä voidaan soittaa yhtä aikaa. Nämä tyhjät peliobjektit asetettiin kameran lapsiobjekteiksi, jotta ne pysyvät aina kameran mukana. Kaikki AudioSourcet haetaan listaan pelin alussa.

```
public List<GameObject> Audiosources;
GameObject[] audio = GameObject.FindGameObjectsWithTag("Audiosource");
                //Lisätään kaikki löydetyt peliobjektit, joilla on tagi Audiosource listaan.
                foreach(GameObject a in audio){
                Audiosources.Add(a);
```

Kun halutaan soittaa tiettyä ääntä, kutsutaan Audio-skriptissä olevaa funktiota, jolle lähetetään parametrina ääni, joka halutaan soittaa. Funktiossa etsitään listasta oikea ääni ja soitetaan sitä.

```
public void PlaySound(string selectedAudioObject){

    for (int i=0;  i<Audiosources.Count; i++){
      //Etsitään listasta oikea ääni ja soitetaan sitä
      if( Audiosources[i].name == selectedAudioObject) {
     Audiosources[i].GetComponent<AudioSource>().audio.Play();
     }
 }
}
```

Yleensä ääntä soitetaan vain kerran, mutta muutamassa tapauksessa ääni on toistuva ja jää soimaan. Tätä varten Audio-skriptissä on myös äänen pysäyttämistä varten oma funktio.

```
Audiosources[i].GetComponent<AudioSource>().audio.Play();
```

5.2.9 Pelin alkaminen ja päättyminen

Pelin alkua varten on oma aloitusruutu, jossa nähdään pelin logo. Aloitusruutu on oma skenensä, jossa sijaitsee vain kamera ja GUI-kuva, joka peittää koko ruudun. Välilyöntiä painamalla skene vaihtuu varsinaiseksi peliskeneksi, ja peli voi alkaa.

Peli voi päättyä kahdella tavalla. Joko pelihahmo kuolee ja pelaaja ei enää jatka pelaamista tai hän läpäisee pelin tuhoamalla loppuvastuksen ja keräämällä kolmivoiman palan.

Pelihahmon kuolema

Pelihahmo kuolee, kun hän menettää kaikki sydämensä. Tällöin käynnistetään animaatio, joka lyyhistää pelihahmon maahan. Pelihahmon liikuttamiseen liittyvät napit eivät enää aiheuta toiminnallisuutta. Tämän jälkeen piirretään Game Over –ikkuna (kuva 33). Kun pelihahmo kuolee, asetetaan boolean ShowGameOverRect trueksi. Tällöin piirretään ruudun alalaitaan ikkuna, josta voidaan valita nuolinäppäimillä jatketaanko peliä vai aloitetaanko alusta.

```
Rect GameOverRect = new Rect(200, Screen.height-200, Screen.width-400,  200);
OnGUI(){
        if(ShowGameOverRect){
                GameOverRect = GUI.Window(3, GameOverRect, createGameOverWindow, "");
        }
}
public void createGameOverWindow(int windowID){
   //Ikkunan valinnat...
}
```

KUVA 33. Game Over -ruutu.

Jos valitaan Continue, samaa peliä jatketaan. Talteen jäävät kaikki pelaajan tekemät asiat, kuten kerätyt esineet, tapetut viholliset, avatut ovet ja niin edelleen. Ainoa vaiva, jonka pelaaja saa, on pelihahmon siirtyminen takaisin luolaston alkuun. Jotta peliä voidaan jatkaa kuoleman jälkeen, täytyy pelihahmon sydämet täyttää ja asettaa useita boolean-muuttujia takaisin alkuperäisiin arvoihinsa. Tämän lisäksi AdvancedMovement-skriptissä olevat asetukset täytyy resetoida ja musiikki käynnistää uudelleen.

```
public void ContinueGame(){
        //Täytetään pelaajan sydämet
        playerhealth.AddjustCurrentHealt(8);
        //Asetetaan Boolean-arvot oikein
        playerhealth.isAlive = true;
        gameIsOver = false;
        //Siirretään pelihahmo pelin alkuruutuun ja oikein päin
        transform.position = spawnPoint;
        transform.rotation = Quaternion.Euler(0, 0, 0);

        //Sumennetaan kameran kuva
        FadeToBlack.FadeTime = 1.0f;
        FadeToBlack.TextureStartColor = Color.black;
        FadeToBlack.FadeColorStart = Color.black;
        FadeToBlack.FadeColorEnd = Color.clear;
        FadeToBlack.IsFading = true;
        //Tehdään uudestaan AdvancedMovement-skriptin alustukset
        SendMessage ("StartNewGame");
        //Käynnistetään taustamusiikki
        GameObject.Find("MainCamera").audio.Play();
        }
```

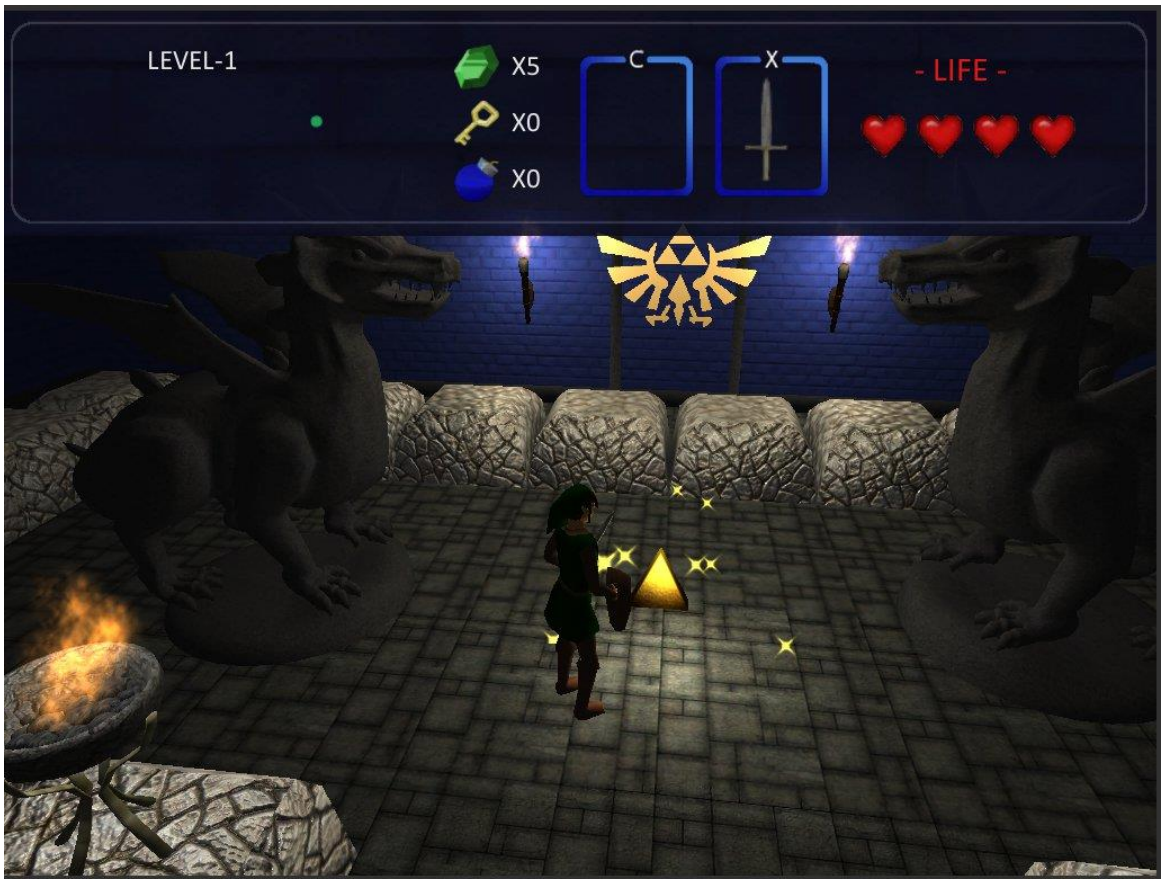
Jos valitaan New Game, peli siirtyy takaisin alkuruutuun, josta voi halutessaan aloittaa pelin täysin alusta.

Pelin voittaminen

Peli voitetaan keräämällä luolaston lopussa sijaitse kolmivoiman pala. Päästäkseen huoneeseen, jossa kolmivoima sijaitsee, pelaajan täytyy päihittää luolaston loppuvastus, Aquamentus. Myös kolmivoimassa on samanlainen triggeri, kuin muissakin esineissä ja sen luokse kävéleminen käynnistää

pidemmän tapahtumasarjan (kuva 34). Pelihahmoa ei voi enää ohjata ja kaikki tapahtuu tästä eteenpäin automaattisesti. Kamera siirtyy loppua varten tehtyyn erikoispisteeseen, ja pelihahmo käännetään kohti kameraa.

Kolmivoima nousee ilmaan pyörien itsensä ympäri ja taustalla soi voitonsävelmä. Kilpi ja miekka täytyy piilottaa ennen kuin käynnistetään animaatio, jossa hahmo nostaa kädet ilmaan. Miekka ja kilpi piilotetaan asettamalla niiden MeshRenderer falseksi. Kaiken tämän toteutukseen tarvittiin useita ajastettuja funktioita, joissa käynnistetään animaatioita, käännetään hahmoa ja siirretään kameraa oikeaan aikaan. Hetken päästä kolmivoiman keräämisestä skene vaihdetaan viimeiseen skeneen, jossa näytetään pelin loppudemo.



KUVA 34. Pelin viimeisessä kammiossa sijaitsee kolmivoima. Kun pelaaja poimii sen, peli päättyy ja varsinainen loppudemo alkaa.

Loppudemo

Loppudemossa ruudulla näytetään pelin logo ja ruudun halki kävelee vuorollaan joku pelin vihollisista. Jokaisen vihollisen jälkeen ruudulle ilmestyy tekstinä tietoa pelin alkuperästä ja toteutuksesta, eli niin kutsutut lopputekstit (kuva 35). Taustamusiikiksi valittiin uusimman Zelda-pelin, Skyward Swordin tunnusmusiikki, koska tuntui mukavalta ajatukselta yhdistää tämän upean pelisarjan ensimmäinen osa jollain tapaa uusimpaan Zelda-peliin. (64)

Loppudemoa varten tehtiin kolme erilaista skriptiä. Yksi tehtiin GUIta varten, joka piirtää peliruudulle tekstejä ja kuvia. Toinen tehtiin luomaan vihollisia ajastetusti ja kolmas vihollisten yksinkertaistettua tekoälyä varten. Kamerassa olevista asetuksista säädettiin taustaväri mustaksi. Skenenäkymään luotiin plane, jonka materiaalin väri asetettiin myös mustaksi, joten se ei erotu mustaa taustaa vasten ollenkaan. Planen kummallekin laidalle asetettiin tyhjät peliobjektit, jotka toimivat jälleen spawnpointeina. Näistä spawnpointeista luodaan vuoronperään tietynlainen vihollinen, joka kävelee suoraan eteenpäin. Oikeaan aikaan ruudulle piirretään haluttu teksti. Kun vihollinen on kävellyt ulos kameran näköpiiristä, se tuhotaan. Koko loppudemo toimii ajastetuilla funktiolla jotka kutsuvat toisiaan tietyin aikavälein.

KUVA 35 . Loppudemo.

5.3 Mallit ja animaatiot Blenderissä

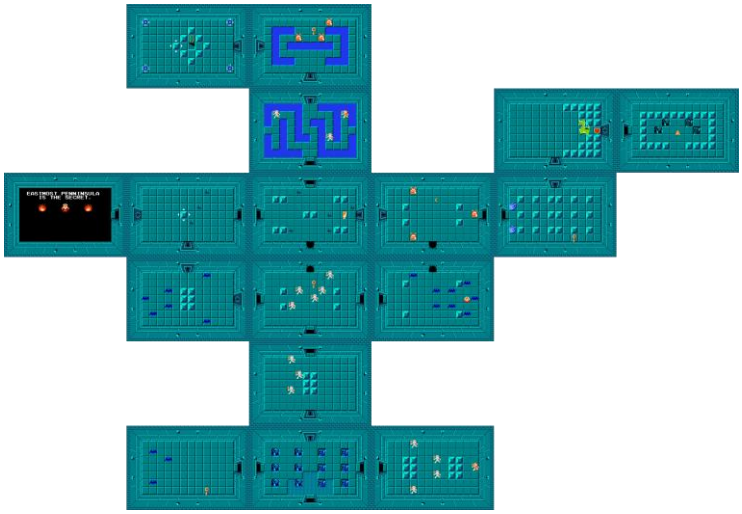
Mallintamiseen, animointiin ja teksturointiin meni lähes puolet kokonaisajasta, ja malleja syntyi kaikkiaan noin kolmekymmentä. Blenderiä oli käytetty hyvin vähän ennen tätä työtä ja siitäkin oli kulunut sen verran aikaa, että moni asia oli unohtunut. Alussa mallintaminen olikin hyvin hidasta, koska se oli samalla työkalun opettelua. Blenderissä tehokas mallintaminen vaatii suuren määrän pikanäppäimien ulkoa opettelua. Ennen kuin aloitettiin mallintamaan ensimmäistä mallia, katsottiin useita tutoriaaleja perusteista lähtien muistin virkistykseksi ja uusien keinojen oppimiseksi. Melko kattava sivusto, alkaen aivan perusteista, löytyy osoitteesta

http://gryllus.net/Blender/VideoTutorials/AllVideoTutorials.html.

Ensimmäiseksi mallinnettavaksi valittiin pelin luolasto, koska se oli yksi projektin alkuvaiheessa tarvittavista malleista ja vaikutti helpommalta, kuin esimerkiksi pelihahmon mallintaminen.

5.3.1 Luolasto

Ennen kuin mallintaminen aloitettiin, täytyi miettiä kentän mittasuhteita. Alkuperäisessä pelissä huone on kuvattu ylhäältä päin. Jokainen huone on 12 ruutua pitkä ja 7 ruutua leveä. Seinän paksuus on puolet ruudun leveydestä (kuva 36).



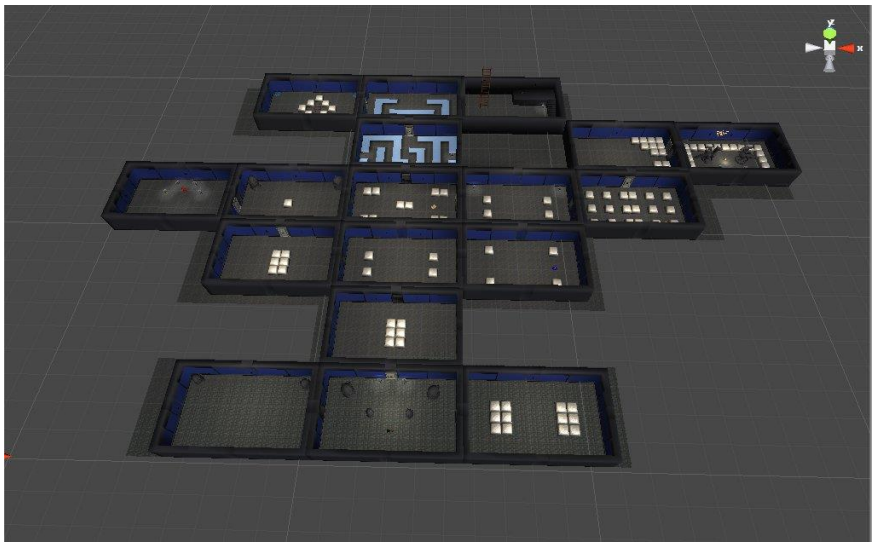
KUVA 36. Alkuperäisen pelin luolasto (65)

Mallintaminen aloitettiin luomalla plane lattiaksi, ja seiniä alettiin tehdä kuutiosta muokkaamalla. Ensin mallinnettiin kokonainen huone ja kokeiltiin, mikä näyttäisi hyvältä huoneen ja oven korkeudelta. Kun huone oli sopiva, siitä tehtiin kopio ja se siirrettiin oikealla paikalleen. Huoneiden ovet ovat eri seinillä, joten ne piti leikata tai peittää erikseen tarpeen mukaan. Tämän jälkeen lisättiin huoneissa näkyvät kivet paikoilleen. Luolasto näytti melko karulta, joten tämän jälkeen lisättiin vielä oven karmit ja vähän yksityiskohtia seiniin. Muutamassa huoneessa on vettä ja lattiaan piti tehdä urat sitä varten. Kun malli oli muuten valmis, se piti teksturoida. Hyviä tekstuureja löytyi netistä googlettamalla ja eräällä sivustolla onkin paljon erilaisia ilmaisia tekstuureja.

http://www.cgtextures.com.

Tekstuureista muokattiin sopivia ja niistä tehtiin tarvittaessa saumattomasti jatkuvia Gimpillä. Luolaston seinät UV-mapattiin ja tekstuurit aseteltiin eri pinnoille halutulla tavalla. Lattiaan oli oma tekstuurinsa ja seiniin käytettiin kahta erilaista kivi -ja tiilitekstuuria.

Kun malli oli tarpeeksi hyvä, se siirrettiin Unityyn ja katsottiin miltä se näyttää (kuva 37). Mallissa olikin aika monta reikää, ja ne piti korjata Blenderin puolella. Reiät johtuvat siitä, että osa mallin tahkoista on väärin päin. Tahkon pystyy näkemään vain toiselta puolelta katsottuna. Reiät pystyy korjaamaan valitsemalla tahkon, josta näkyy läpi, ja painamalla Flip Direction -painiketta. Tämä kääntää tahkon normaalit toisin päin. Normaali tarkoittaa matematiikassa pintaa tai janaa vastaan kohtisuorassa olevaa suoraa tai vektoria (66). Muutosten jälkeen malli täytyy siirtää uudestaan Unityyn, jotta tehdyt muutokset siirtyvät myös sinne.



KUVA 37. Pelin luolasto valmiina Unityn skenenäkymässä.

Ovet ja tikkaat

Pelissä on kahdenlaisia ovia, jotka mallinnettiin erikseen ja aseteltiin oikeille paikoilleen Unityn puolella. Luolastoon tarvittiin lisäksi portaat kellariin pääsyä varten. Tähän versioon portaat muutettiin tikkaiksi. Tikkaita pitkin ei oikeasti pääse kiipeämään, joten ne ovat vain rekvisiittaa. Pelihahmo siirretään tikkaiden luona olevasta triggeristä alakertaan ja päinvastoin. Tikkaita täytyi tehdä myös kahdet, yläkerrassa näkyvä lyhyt pätkä, joka näkyy lattiassa olevasta aukosta, ja alakerrassa pitkät tikkaat, jotka ovat johtavinaan yläkertaan. (Kuvat 38 ja 39.)

KUVA 38. Alkuperäisen pelin lukittu ovi, sulkeutuva ovi ja portaat alakertaan.

KUVA 39. Uudet versiot ovista ja tikkaat.

5.3.2 Pelihahmo Link

Pelihahmon mallintaminen oli aluksi todella vaikeaa. Netistä löytyi onneksi vinkkejä kuinka päästä alkuun. Mallin teossa käytettiin mirror modifieria. Mirror modifier on apuväline, joka luo mallinnettavasta kappaleesta peilikuvan. Tällöin tarvitsee mallintaa vain puolet hahmosta, ja toinen puoli muodostuu automaattisesti peilikuvana. Mallintaessa täytyy olla tarkkana, jotta puolikkaat liittyvät toisiinsa saumattomasti. (67.)

Hahmon rungon mallintamisessa käytettiin apuna seuraavaa ohjetta:

http://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Modeling_a_Human_Character_-_Modeling.

Kun hahmon runko oli valmis, siirryttiin tekemään yksityiskohtia, kuten kengät, vaatteet, vyö ja lopuksi pään yksityiskohdat. Tämän jälkeen oli aika tutustua riggaamiseen. Mallille täytyi saada luuranko, jotta sen pystyisi animoimaan. Pelihahmo on rigattu seuraavan videotutoriaalin mukaan.

http://cgcookie.com/blender/2011/12/12/blender-introduction-to-character-rigging/

Video kestää 45 minuuttia, koska siinä lisättiin perusluurangon lisäksi niin sanottuja IK-luita (Inverse Kinematics). IK:lla pystytään helpottamaan ja

nopeuttamaan animaation tekoa ja niiden avulla saadaan aikaan luonnollisen näköistä liikettä. (68.)

Pelihahmon perusvarustukseen kuuluu miekka ja kilpi. Miekan liittämistä ohjelmakoodin avulla hahmon käteen kokeiltiin Unityn puolella. Miekasta tehtiin hahmon käden lapsiobjekti. Miekka asettuikin käteen ja pysyi sen mukana liikkuessa, mutta se meni käden läpi ranteesta. Tarvittiin siis jonkinlainen yhtymäpiste hahmon ja aseen välille, jotta se asettuisi järkevästi. Yhtymäpisteeksi tarvittiin tyhjä objekti, joka asetettiin luurangon lapsiobjektiksi Blenderissä. Kun luuta liikutetaan, sen lapsiobjekti liikkuu aina mukana. Miekka tulisi asettaa tämän tyhjän objektin lapseksi käden sijaan.

Loppujen lopuksi miekka ja kilpi liitettiin pelihahmon käsiin jo Blenderissä, koska ne vaikuttavat olennaisesti hahmon tapaan juosta puhumattakaan lyömisestä. Järkevän animaation tekeminen näkemättä varusteita olisi todella vaikeaa. Alkuperäisessä pelissä miekka ja kilpi ovat koko ajan pelihahmon käsissä ja esimerkiksi jousipyssyä käytettäessä näkyy vain nuoli, joka lentää hahmosta. Asian annettiin toistaiseksi olla näin myös tässä versiossa. Kumpaankin käteen lisättiin varmuuden varalta tyhjät objektit, joiden avulla saisi vaihdettua tarvittaessa kädessä olevan aseen Unityn puolella.

Pelihahmolle tarvittiin useita animaatioita. Aluksi tehtiin juoksu-, lyönti- ja idle-animaatiot. Lyöntianimaation tekeminen tuntui vaikealta saada oikean näköiseksi, koska pelihahmo on vasenkätinen ja oli vaikeaa kuvitella liikerataa omasta näkökulmasta peilikuvana. Idleanimaatio on niitä hetkiä varten, kun hahmolla ei liikuta ja se seisoskelee paikallaan. Olisi luonnottoman näköistä, jos hahmo seisoisi paikallaan aivan kuin patsas. Idle animaatio sisältää pientä liikettä paikallaan, joka saa hahmon vaikuttamaan elävältä. Myöhemmin hahmolle lisättiin vielä animaatiot kuolemaa ja loppua varten. Lopetusanimaatiossa pelihahmo nostaa kummatkin kädet ilmaan, kun kolmivoima saadaan haltuun. Kuolema-animaatiossa pelihahmo lyyhistyy maahan.

Lopuksi hahmo täytyi vielä teksturoida. Hahmoon lisättiin kangastekstuurit hattua, tunikaa ja paitaa varten, nahkatekstuuri saappaita ja vyötä varten,

metallitekstuuri vyönsolkiin ja kengänsolkiin, ihotekstuuri, hiustekstuuri ja lopuksi vielä silmätekstuuri. Silmien teksturointia varten tallennettiin mallin silmien ääriviivat UV-editorista erilliseen png-tiedostoon ja piirrettiin Gimpillä oma tekstuuri. Huulia ja korvarenkaita varten lisättiin materiaali, joilla niihin saatiin väriä.

Kilpeä varten tallennettiin UV-editorista ääriviivat yksityiskohtia varten ja siihen lisättiin puutekstuuri. Miekka sai terää ja kahvaa varten omat metallinsävyiset tekstuurit. Tämän jälkeen hahmon sai vihdoinkin siirrettyä Unityyn, jossa siihen liitettiin tarvittavat komponentit ja ohjelmakoodit. Lopputulos näkyy kuvassa 40.

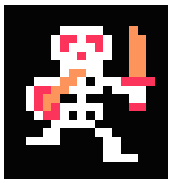
KUVA 40. Pelihahmon alkuperäiset mallit (67). Oikealla näiden pohjalta tehty oma pelihahmo.

5.3.3 Viholliset

Luolastossa on viisi erilaista perusvastusta. Jokainen näistä vihollisista voidaan nähdä myös uudemmissa Zelda-peleissä.

Stalfos

Stalfos on luuranko, jolla on kummassakin kädessä miekka (kuva 41). Stalfos on yleinen vihollinen pelissä, eikä se ole mitenkään erityisen voimakkaita tai vaikeita voittaa. (70.)



KUVA 41. Alkuperäinen Stalfos (70)

Mallinnus aloitettiin muovaamalla ensin jalkaterä ja jatkamalla siitä ylöspäin kohti polvea ja lantiota. Kun jalka oli valmis, siitä tehtiin kopio ja se käännettiin peilikuvaksi, jotta mallille saatiin helposti toinen jalka. Kun lantio oli saatu muokattua suurin piirtein sopivaksi, jatkettiin selkärangan muovaamisella. Luurangossa on niin paljon ohuita kohtia, että osat oli helpompi mallintaa erikseen, jottei mene sekaisin, mitä on tekemässä. Tämän jälkeen muovattiin rintakehä ja olkapäät, sekä kädet. Selkäranka liitettiin kiinni rintakehään ja kallo selkärangan jatkoksi. Lopuksi luurangolle tehtiin sormet.

Kun malli oli muodoltaan valmis (kuva 42), koko luurankoon asetettiin vaalea luutekstuuri. Seuraavaksi luuranko täytyi rigata animointia varten. Blenderissä pystyy hakemaan eri blend-tiedostoista objektien dataa ja yhdistämään niitä toisiin blend-tiedostoihin. Pelihahmon blend-tiedostosta haettiin hahmon luuranko ja lisättiin se samaan tiedostoon Stalfoksen blend-tiedoston kanssa. Pelihahmon malli oli hieman pienempi kuin Stalfoksen malli, joten luuranko piti skaalata ja asetella uudestaan sopivaksi Stalfokselle.

KUVA 42. Uusi Stalfos

Tässä säästyi paljon aikaa, koska pelihahmon luuranko oli tehty hyvin huolellisesti, ja se saatiin IK-luineen käyttöön pienellä vaivalla myös Stalfokselle. Kun luut ja nivelet oli aseteltu sopivaksi Stalfosta varten, malli ja luuranko yhdistettiin, eli mallin luihin yhdistettiin eri verteksiryhmät käyttämällä automaattista laskentaa. Verteksiryhmiä piti kumminkin säätää manuaalisesti, koska vaikutusalueet olivat hieman liian suuret. Tämä tarkoittaa käytännössä sitä, että niveltä taivutettaessa liian suuri alue kohdealueesta liikkuu mukana. Stalfos on siis itsekin luuranko, ja luut eivät tietenkään veny liikkeiden mukana samalla tavalla kuin lihakset ja iho. Lopuksi Stalfoksen kumpaankin käteen asetettiin miekat. Miekka on sama, joka löytyy myös pelihahmolta ja niitä on vain skaalattu isommaksi Stalfosta varten. Stalfosta varten tehtiin lopuksi kaksi animaatiota, idle ja kävely.

Keese

Keese on lepakko, joita lentelee luolaston huoneissa (kuva 43). Lepakot jäävät välillä paikalleen ja jatkavat hetken kuluttua lentoaan ympäri huonetta. Ne ovat heikkoja ja tuhoutuvat yhdestä osumasta. (71.)

KUVA 43. Alkuperäinen Keese (71)

Keesen mallintaminen aloitettiin kuutiosta ja siihen käytettiin mirror modifieriä, niin kuin pelihahmoonkin. Siipien tekeminen tuntui aluksi melko haastavalta, joten mallia katsottiin oikeasta lepakon kuvasta.

KUVA 44. Uusi Keese

Kun malli oli valmis, se teksturoitiin käyttämällä ruskeaa turkkitekstuuria sekä kahta erilaista nahkatekstuuria siipien sisäpinnoille, korviin ja suuhun. Hampaisiin lisättiin luutekstuuri ja silmiä varten piirrettiin oma tekstuurinsa Gimpillä (kuva 44). Myös Keeseä varten käytettiin samaa luurankoa kuin pelihahmolle ja se lisättiin Keesen blend-tiedostoon pelihahmon blend-tiedostosta. Luurankoa täytyi pienentää ja muutella melko paljon, sekä lisätä yksi sormiluu, jotta se saatiin sopivaksi keeseä varten. Lepakon sormet alkavat noin puolivälistä siipeä ja ovat todella pitkät. Siipien nahka levittyy näiden sormien väliin. Siipien luiden asettelu oli melko helppoa, koska malli oli tehty oikeaa lepakkoa jäljitellen. Keeseä varten tehtiin kaksi animaatiota, idle ja lentäminen. Idle animaatiossa lepakko on piiloutunut siipiensä suojiin.

Gel

Gel on jonkinlainen hyytelömäinen pisara, joka liikkuu ympäri luolaston lattioita (kuva 45). Pisarat ovat heikkoja vastuksia ja kuolevat yhdestä osumasta. (72.)

KUVA 45. Alkuperäinen Gel (72)

Gel tehtiin luomalla ensin pallo, josta muovattiin pisaran muotoinen. Geliin ei lisätty tekstuuria vaan väri saatiin aikaan materiaalilla (kuva 46). Unityn puolella muutettiin pisaran materiaalin läpinäkyvyyttä, niin että lopullinen malli on hieman läpikuultava. Koska muoto on näin yksinkertainen, luurankoa ei tarvita ollenkaan ja tarvittavat animaatiot, idle ja liikkuminen, tehtiin skaalaamalla pisaraa erikokoiseksi tietyin väliajoin.

KUVA 46. Uusi Gel

Goriya

Goriyat ovat ihmisen ja suden sekoitukselta näyttäviä kahdella jalalla liikkuvia olentoja (47). Ne käyttävät aseenaan normaalisti bumerangia ja ovat hieman kestävämpiä, kuin edellä esitellyt viholliset. (73.)

KUVA 47. Alkuperäinen Goriya (73)

Goriya oli pelin viimeinen mallinnettu vihollinen ja sen pohjana käytettiin pelihahmon ihmisvartalon runkoa. Ensimmäisenä hahmon jalkoja lyhennettiin reippaasti ja kantapäätä nostettiin ylöspäin. Mallia katsottiin hieman oikean suden takajaloista. Hahmon ylävartaloa ja käsivarsia paksunnettiin reilusti ja niskasta tehtiin leveä ja voimakkaasti eteenpäin kaartuva. Hahmon päätä siirrettiin eteenpäin niin, että se jatkuu suoraan kehosta ilman varsinaista kaulaa. Päähän muovattiin kuono hampaineen ja korvat. Käsiin lisättiin sormet ja pitkät kynnet samoin kuin jalkoihin. Käsiä ja jalkateriä skaalattiin reilusti isommaksi.

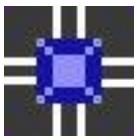
KUVA 48. Uusi Goriya

Goriyan teksturointiin käytettiin tummanharmaata turkkitekstuuria. Korviin, nenään ja suuhun asetettiin omat nahkatekstuurinsa. Kynsiin ja hampaisiin käytettiin luutekstuuria, ja lopuksi sudelle piirrettiin Gimpillä tekstuuri silmiä varten (kuva 48). Jälleen kerran käytettiin samaa luurankoa, kuin pelihahmolla ja se piti muokata ja skaalata sopivaksi Goriyalle. Mittasuhteita piti hieman muutella, erityisesti jaloissa. Kun luuranko yhdistettiin mallin vartaloon, täytyi verteksiryhmiä jälleen muokata manuaalisesti. Suden pää on kuonon vuoksi melko pitkä, joten se jäi kokonaan pois laskuista ja piti erikseen lisätä päätä

liikuttavaan luuhun. Goriyalle tehtiin kaksi animaatiota, jotka ovat idle ja kävely. Tässä pelin versiossa jätettiin Goriyan bumerangin käyttö pois.

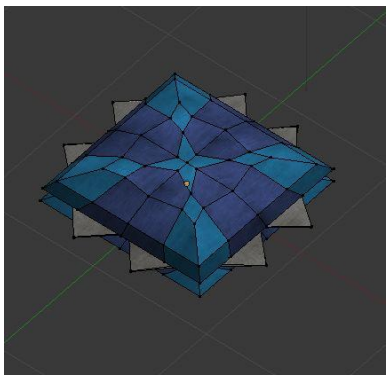
Trap

Ansat ovat usein huoneiden nurkkiin sijoitettuja laatikonmuotoisia esineitä, joiden jokaiselta sivulta tulee esiin piikit (kuva 49). Ansoja ei voi tuhota, ja ne lähtevät liikkeelle havaitessaan liikettä. (74.)



KUVA 49. Alkuperäinen trap (74)

Ansaa lähdettiin muokkaamaan kuutiosta ja sen mallintaminen oli nopeaa ja helppoa. Ansan pintaan on käytetty kahta eri metallitekstuuria ja piikkeihin omaansa (kuva 50). Ansaa ei tarvinnut animoida, koska sen liikkuminen toteutettiin Unityssa.



KUVA 50. Uusi trap

5.3.4 Loppuvastus

Pelin ensimmäinen loppuvastus on tultasyöksevä lohikäärme, Aquamentus. Loppuvastus tuhotaan alkuperäisessä pelissä lyömällä sen päässä olevaa sarvea (kuva 51). Se kestää enemmän iskuja kuin pelin tavalliset viholliset, mutta on melko helppo tuhota käyttämällä pommeja tai jousipyssyä. (75.)

KUVA 51. Alkuperäinen Aquamentus (75)

Aquamentuksen mallintamiseen käytettiin enemmän aikaa, kuin mihinkään muuhun malliin, koska siinä on niin paljon yksityiskohtia. Mallintaminen alkoi jälleen kuutiosta ja siihen käytettiin mirror modifieriä. Kuutiosta muotoiltiin ensin karkea luonnos lohikäärmeen vartalosta. Tämän jälkeen neliskanttista muotoa alettiin pyöristellä ja hioa kohti lopullista muotoa. Hahmon päästä venytettiin kuono kahdessa osassa, jotta sille saatiin aukeavat leuat. Kuonon muoto, silmäkuopat ja sarvet muotoiltiin esiin. Suu täytettiin hampailla.

Tämän jälkeen siirryttiin mallintamaan lohikäärmeen kouria. Kun etukäpälän yksi varvas oli saatu valmiiksi, siitä kopioitiin ja muokattiin toiset kaksi varvasta sekä peukalo. Kun etukäpälä oli tarpeeksi hyvä, otettiin varpaista peukaloa lukuunottamatta kopio ja ne liitettiin takajalkaan. Takajalkaan liitettiin neljäs varvas ja käpälän kokoa kasvatettiin hieman. Siipiä varten hyödynnettiin Keese-mallia. Keesen siivistä otettiin kopio ja niitä suurennettiin reilusti ja käännettiin oikeaan kulmaan lohikäärmettä varten. Tämän jälkeen siipi kiinnitettiin malliin. Lopuksi lohikäärmeen häntä muovattiin suipentumaan kohti hännänpäätä ja lohikäärmeen selkään kiinnitettiin tietyin välimatkoin koristeellisia suomuja.

Teksturointiin käytettiin kahta erilaista suomutekstuuria lohikäärmeen rintaan ja vartaloon. Vihreällä nahkatektuurilla päällystettiin lohikäärmeen siipien sisäpinnat sekä selässä olevien suomujen osasia. Suun sisäpuoleen käytettiin punaista nahkatekstuuria. Hampaisiin, sarviin, kynsiin sekä siivissä oleviin piikkeihin käytettiin luutekstuuria. Lohikäärmeen silmiä varten piirrettiin tekstuuri Gimpillä (kuva 52).

KUVA 52. Uusi Aquamentus

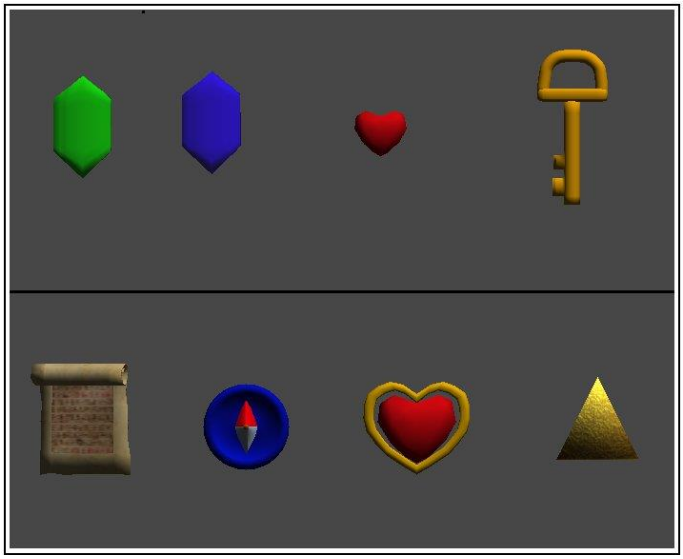
Myös lohikäärmettä varten käytettiin samaa luurankoa kuin muillekin hahmoille. Luurankoa täytyi skaalata paljon suuremmaksi ja muokata luiden paikat kokonaan uusiksi. Häntää varten lisättiin kolme luuta. Aquamentusta varten tehtiin idle ja hyökkäys -animaatiot.

5.3.5 Esineet ja aseet

Peliin mallinnettavia esineitä (76.) oli melko monta ja kaikki olivat melko nopeita ja helppoja mallintaa ja teksturoida (kuva 53).

KUVA 53. Alkuperäisen pelin esineitä.

Vihreä ja sininen raha ovat sama malli, joissa on vaihdettu vain materiaalin väriä. Sydämen pohjaksi otettiin pallo, josta muovattiin oikean mallinen ja lopuksi se värjättiin punaisella materiaalilla. Sydänsäililö on sama sydän, mutta siihen mallinnettiin ympärille rengasmainen kultainen koriste. Avain on mallinnettu useista osista, ja siinä on pohjana särmiö. Väri saatiin aikaan kullankeltaisella materiaalilla. Kartta tehtiin kuutiosta litistämällä ja pyörittämällä sitä hieman päästä rullalle. Tekstuuriksi valittiin vanha paperi ja keskelle lisättiin toinen tekstuuri, jossa on riimukuvioita. Kompassi tehtiin särmiöstä litistämällä ja muokkaamalla ja siihen lisättiin värit materiaaleilla. Kolmivoima on muokattu kuutiosta ja se on teksturoitu kullalla (kuva 54).

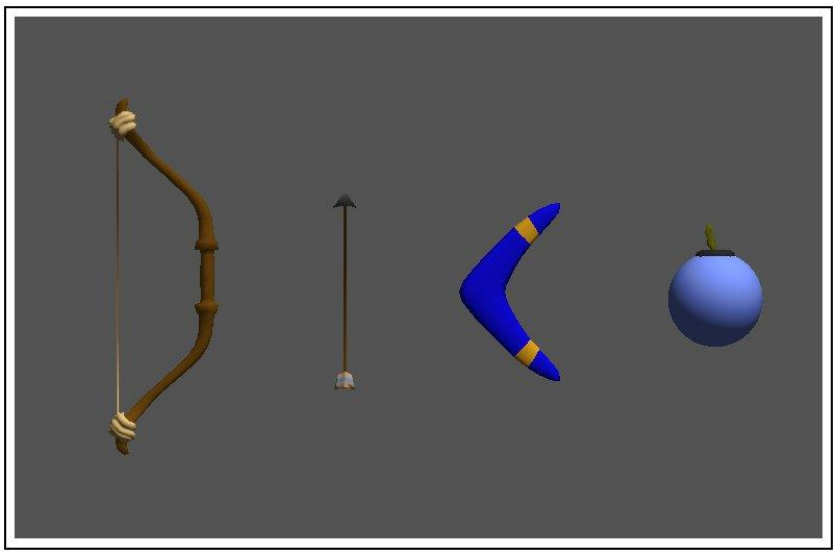


KUVA 54. Uusia pelin esineitä.

Aseiden (kuva 55) mallintaminen oli hieman työläämpää, kuin esineiden ja näistä aikaa vievin oli jousipyssy. Sekä jousipyssyyn että bumerangiin käytettiin mirror modifieria. Jousipyssy teksturoitiin puulla ja jänne sai värin materiaalista. Myös nuoli on teksturoitu puulla ja kärki sai metallitekstuurin. Pommi on

muovattu pallosta, johon on lisätty sytytyslanka ja pääliosa. Pommi sai värinsä materiaalista (kuva 56).

KUVA 55. Alkuperäisiä pelin aseita.



KUVA 56. Uusia pelin aseita.

5.3.6 Muut mallit

Pelin muita malleja ovat vanha mies sekä luolaston koristeluun tarkoitetut patsaat ja soihdut.

Vanha mies

Pelin salaiseen huoneeseen tarvittiin vanha mies opastamaan pelaajaa, sekä myymään tavaraa (kuva 57).

KUVA 57. Alkuperäinen vanha mies (77)

Vanhaa miestä alettiin mallintaa kuutiosta, johon oli liitetty mirror modifier. Ensin siitä muovattiin sopivan korkea ja leveä möykky, josta venytettiin esiin kädet. Tämän jälkeen alettiin muovata hupun ja kasvojen yksityiskohtia. Tästä uudesta vanhasta miehestä tuli itseasiassa vahingossa huppupäinen, koska unohdettiin, ettei alkuperäisellä hahmolla ole huppua. Mies oli kumminkin hupullisena ihan hieno, joten ei nähty järkeväksi lähteä enää muuttamaan sitä.

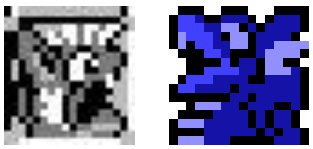
KUVA 58. Uusi vanha mies

Miehen kaapu teksturoitiin punaisella sametilla. Ihoon käytettiin samaa nahkatekstuuria kuin pelihahmollakin ja partaan harmaata turkkitekstuuria (kuva 58). Vanhalle miehelle piti lisätä myös luuranko animointia varten. ja tässä käytettiin jälleen samaa luurankoa kuin kaikille muillekin animoiduille hahmoille.

Luuranko oli sellaisenaan melko sopiva ja vähällä vaivalla siitä saatiin toimiva. Vanha mies tarvitsi vain yhden animaation, joka oli idle.

Patsaat

Luolaston sisäänkäynnillä ja viimeisessä kammiossa on koristeena kala- ja lohikäärmepatsaita (kuva 59). Patsaita lisättiin loppujen lopuksi myös muualle luolastoon, alkuperäisestä pelistä poiketen.



KUVA 59. Alkuperäiset kala- ja lohikäärmepatsas

Kalapatsasta alettiin muokkaamaan sylinteristä ja siihen käytettiin mirror modifieria. Patsaan naamapuolelle painettiin syvänteinä suun ja silmien alueet ja sivulta vedettiin esiin evät. Pään keskeltä niskaan kulkee myös evien jono. Lopuksi patsas teksturoitiin tummalla kivitekstuurilla (kuva 60).

KUVA 60. Uusi kalapatsas

Lohikäärmepatsaan tekemiseen käytettiin hyödyksi jo valmista Aquamentus-lohikäärmeen mallia. Lohikäärmeen häntää taivutettiin sivulle päin ja sen

jalkojen juureen lisättiin koroke. Vanhat tekstuurit poistettiin lohikäärmeen pinnalta ja ne korvattiin samalla tummalla kivitekstuurilla, kuin kalapatsaallakin. Patsas jätettiin tarkoituksella karkean ja veistetyn näköiseksi (kuva 61).

KUVA 61. Uusi lohikäärmepatsas

Soihdut ja roihut

Luolastoon mallinnettiin sekä koristeeksi että valonlähteeksi kaksi erilaista soihtua. Toinen on kivinen astia, jossa on hiiliä (kuva 62) ja toinen pidikkeessä oleva soihtu (kuva 63). Astia tehtiin pallosta muovaamalla ja hiilet tehtiin pienemmistä palloista kopioimalla ja muuttamalla niiden kokoa erilaisiksi. Telineen jalka muovattiin sylinteristä taivuttelemalla, ja kun se oli hyvä, siitä tehtiin kolme kopiota. Lopuksi jalat aseteltiin sopivalla tavalla tukemaan astiaa. Astia teksturoitiin kivitekstuurilla ja jalkoihin käytettiin metallitekstuuria. Hiiliä varten käytettiin hiilitekstuuria.

KUVA 62. Roihu

Soihtu muovattiin sylinteristä ja sille tehtiin erikseen pidike, jolla se kiinnittyy seinään. Soihdun kärkeen käytettiin samaa hiilitekstuuria kuin ylempänä olevan mallin hiiliin. Soihdun varteen laitettiin puutekstuuri ja pidikkeeseen kuparitekstuuri.

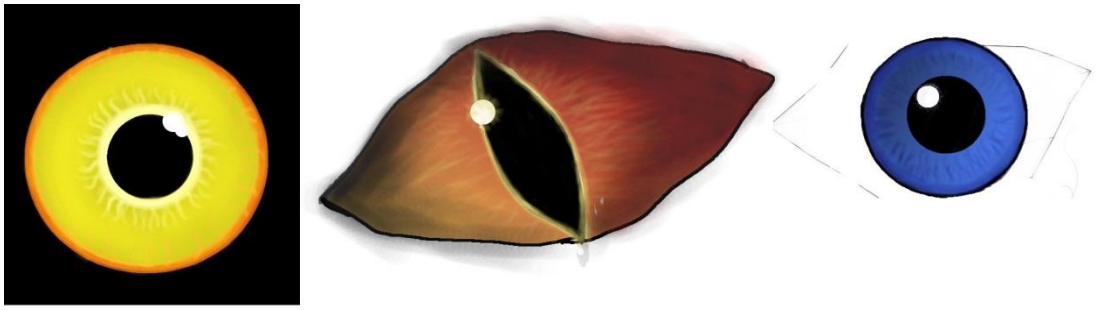
KUVA 63. Soihtu

5.4 2D-grafiikka Gimpillä

Lähes kaikki pelissä käytetyt tekstuurit käsiteltiin Gimpillä ennen niiden käyttämistä Blenderissä (kuva 63). Usein kyseessä oli tekstuurin saumattomaksi muuttaminen tai värimuunnos. Peliä varten olevia tekstuureja tuli kaikenkaikkiaan yli viisikymmentä.

KUVA 63. Erilaisia tekstuureita. Kultaa, kiveä ja vettä.

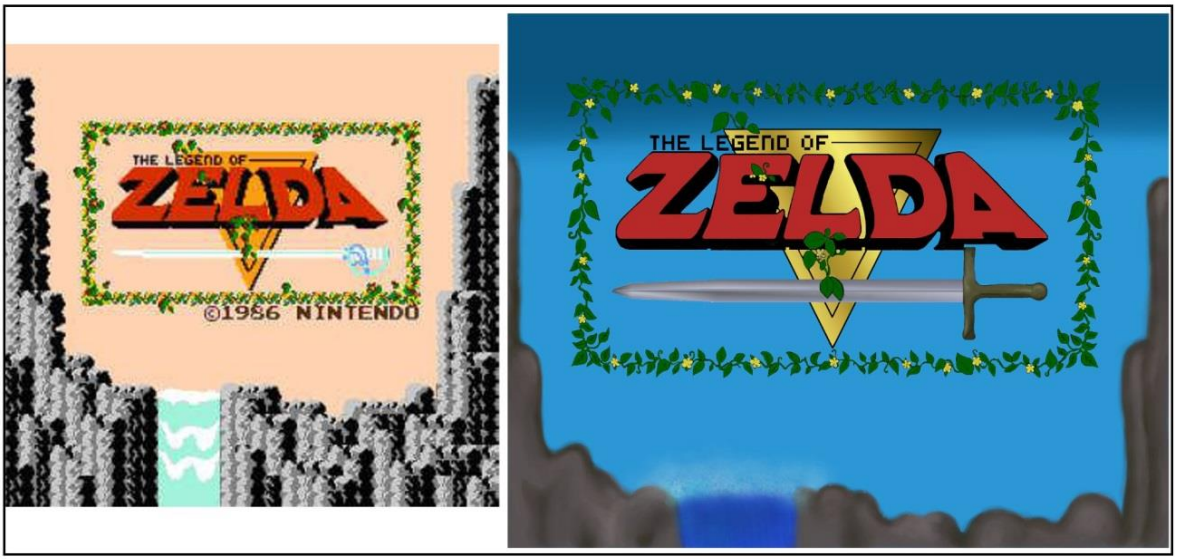
Useita malleja varten täytyi myös piirtää omia tekstuureja, esimerkiksi useita hahmojen silmiä (kuva 64). Silmät piirrettiin piirtopöytää käyttämällä.



KUVA 64. Erilaisia silmiä. Suden, lohikäärmeen ja pelihahmon silmät.

Pelin GUIta ja inventorya varten täytyi piirtää melko paljon erilaisia taustakuvia, sekä kuvat pelin esineistä. Pelin esineet ja aseet kuvattiin Blenderissä tehdyistä malleista ja ne muokattiin Gimpillä siistiksi, sopivan kokoiseksi ja taustaltaan läpinäkyväksi.

Eniten aikaa vienyt yksittäinen piirretty kuva oli pelin alkuruutu, joka piirrettiin myös piirtopöytää käyttäen (kuva 65). Aloituskuva piirrettiin useissa eri tasoissa niin, että teksti oli omassaan, tekstin takana näkyvä kolmivoiman pala, miekka ja köynnös omassaan ja kolmannessa sininen tausta vuoristoineen.



KUVA 65. Pelin alkuperäinen aloitusruutu ja sen pohjalta piirretty uusi.

Gimpiä käytettiin erittäin paljon myös tämän opinnäytetyön kuvituksen tekemiseen. Peli sisältää paljon pieniä yksittäisiä esineitä ja näistä tehtyjä malleja, ja nämä kuvat haluttiin yhdistää isommiksi kokonaisuuksiksi.

6 YHTEENVETO

Pelin tekeminen oli todella mukavaa ja mielenkiintoista. Lopputulos ylitti kaikki odotukseni. En odottanut saavani valmiiksi läheskään kaikkea, mitä olin listannut pelin ominaisuuslistaan. Olin miettinyt jo etukäteen, mitä jätän pois, jos aika loppuu kesken. Loppujen lopuksi sain tehtyä kaiken, mitä halusinkin, ja ehdin jopa testata ja korjailla peliäni. Opin tämän työn aikana niin paljon uusia asioita kaikesta pelinkehitykseen liittyvästä, että sitä on vaikea kuvailla, ja kehitys oli huimaa. Ensimmäisen parin viikon aikana en saanut aikaan juuri mitään, kun taas projektin viimeisinä viikkoina tein peliin valtavan määrän uusia ominaisuuksia, optimointia, grafiikkaa ja malleja.

Varsinaisen opinnäytetyön tekeminen oli jaettu kahteen osaan. Työn tekemiseen oli varattu Scrum-dokumenttien mukaan noin 250 tuntia. Tämä aika oli jaettu 9 viikolle. Todellisuudessa arvioisin, että työn tekemiseen käytetty tuntimäärä on lähempänä 350:tä tuntia tai jopa enemmän, koska työskentelin lisäksi myös viikonloppuina ja usein pidempää päivää.

Scrum-menetelmä ei oikein pääse oikeuksiinsa silloin, kun kyseessä on yhden henkilön tiimi. Ei ole suurta tarvetta esittää tehtäviä ja jakaa ominaisuuksia pienemmäksi niin tarkasti vain itseä varten, koska tiedän jo, mitä teen. Tämän lisäksi varsinkin projektin alussa työtehtäviin menevää aikaa oli hyvin vaikea arvioida, koska käytetyistä työkaluista ei ollut paljoa kokemusta. Kirjasin Scrum-dokumenttien lisäksi pelin eri ominaisuuksia paperille miettiessäni koodiarkkitehtuuria, ja minun kohdallani paperi toimii siinä paremmin kuin dokumenttipohjat. Minusta on helpompi hahmottaa asioita piirtämällä ja suunnittelen mielelläni ominaisuuksia kirjoittamalla pseudokoodia. Jos tiimissä olisi ollut useampi henkilö, Scrum-menetelmästä olisi varmasti ollut hyötyä.

Koska olen pelannut paljon, minulla oli rima aika korkealla myös oman pelini suhteen. Hahmojen animaatioissa, kamerassa, vihollisten tekoälyssä ja monissa muissa asioissa on vielä paljon parannettavaa. Jos aloittaisin pelin tekemisen nyt alusta, muuttaisin kamerasysteemin kokonaan toisenlaiseksi, koska jotkut asiat eivät vain toimi 3D:nä riittävän hyvin. Peliä tehdessä joutuikin

usein jatkamaan eteenpäin, vaikka jäi tunne, että olisin pystynyt parempaan. Koko ajan oppi uutta ja oivalsi uusia parempia ratkaisuja jo aiemmin tehtyihin ominaisuuksiin. Aika oli kumminkin hyvin rajallinen, joten ei ollut mahdollista palata muuttamaan jo toimivia ominaisuuksia paremman koodiarkkitehtuurin vuoksi.

Jonkin olemassa olevan pelin pohjalta uuden tekeminen on mielestäni todella hyvä tapa oppia, jos peliala kiinnostaa. Tällöin voidaan keskittyä enemmän tekemiseen, oppia työkalujen käyttö ja nähdä, millaiset ratkaisut toimivat tai eivät toimi. Tämän jälkeen on varmasti helpompi suunnitella ja tehdä omaa peliä, kun on käsitystä, miten erilaisia ominaisuuksia voidaan toteuttaa parhaiten.

Vaikka olen pääosin tyytyväinen työhöni, en voisi kuvitellakaan tekemäni version kilpailevan alkuperäisen klassikon kanssa, eikä näin ole tarkoituskaan. Työn ehdottomasti vaikein osuus oli tämän opinnäytetyön kirjoittaminen, koska oli vaikea tiivistää kaikki oppimani ja tekemäni järkeväksi kokonaisuudeksi. Toivon kuitenkin, että siitä on jotain iloa tai jopa hyötyä myös muille, joita peliala kiehtoo.

Arvostukseni oikeita pelialan ammattilaisia kohtaan kasvoi entisestään. Hyvän ja näyttävän pelin tekeminen vaatii todellista osaamista. Myös kiinnostukseni pelialaa kohtaan voimistui ja toivon, että voisin joskus itsekin olla osa oikeaa pelinkehitystiimiä.

LÄHTEET

1. The Legend of Zelda. 2013. Saatavissa: http://en.wikipedia.org/wiki/The_Legend_of_Zelda. Hakupäivä 11.3.2013.

2. The Legend of Zelda: 25th Anniversary-25 Years of Wisdom. 2011. Saatavissa: http://www.zeldadungeon.net/2011/11/the-legend-of-zelda-25th-anniversary-25-years-of-wisdom/. Hakupäivä. 11.3.2013.

3. The Legend of Zelda (video game). 2013. Saatavissa: http://en.wikipedia.org/wiki/The_Legend_of_Zelda_(video_game). Hakupäivä 11.3.2013.

4. Zelda sharing tunes with the rest of the Nintendo family. 2011. Saatavissa: http://shockingvideogamesecrets.wordpress.com/tag/legend-of-zelda/. Hakupäivä 11.3.2013.

5. Scrum. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Scrum. Hakupäivä 11.3.2013.

6. Scrummaster.com.au. 2012. Saatavissa: http://scrummaster.com.au/Home.mvc. Hakupäivä 11.3.2013.

7. Ketterä ohjelmistokehitys. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Ketterä_ohjelmistokehitys. Hakupäivä 11.3.2013.

8. Stand-up meeting. 2013. Saatavissa: http://en.wikipedia.org/wiki/Stand-up_meeting. Hakupäivä 11.3.2013.

9. Scrum Documentation. 2009. Saatavissa: http://www.chrismcspiritt.com/scrum-documentation/. Hakupäivä 11.3.2013.

10. Unity (game engine). 2013. Saatavissa: http://en.wikipedia.org/wiki/Unity_Technologies. Hakupäivä 11.3.2013.

11. Creating Scenes. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Manual/CreatingScenes.html. Hakupäivä 11.3.2013.

12. Learning the Interface. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/LearningtheInterface.html. Hakupäivä 13.3.2013.

13. Hierarchy. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Hierarchy.html. Hakupäivä 13.3.2013.

14. Inspector. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Inspector.html. Hakupäivä 13.3.2013.

15. Project Browser. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/ProjectView40.html. Hakupäivä 13.3.2013.

16. GameObjects. 2010. Saatavissa: http://docs.unity3d.com/Documentation/Manual/GameObjects.html. Hakupäivä 11.3.2013.

17. Mesh Filter. 2010. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-MeshFilter.html. Hakupäivä 11.3.2013.

18. Box Collider 2009. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-BoxCollider.html. Hakupäivä 11.3.2013.

19. Collider.OnTriggerEnter. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Collider.OnTriggerEnter.html. Hakupäivä 11.3.2013.

20. Audio Components. 2007. Saatavissa: http://docs.unity3d.com/Documentation/Components/comp-AudioGroup.html. Hakupäivä 11.3.2013.

21. Prefabs. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Prefabs.html. Hakupäivä 11.3.2013.

22. Particle Systems. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Manual/ParticleSystems.html. Hakupäivä 12.3.2013.

23. GUI Scripting Guide. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Components/GUIScriptingGuide.html. Hakupäivä 12.3.2013.

24. GUI Skin. 2007. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-GUISkin.html. Hakupäivä 12.3.2013.

25. Monodevelop. Saatavissa: http://monodevelop.com/. Hakupäivä 13.3.2013.

26. Using Scripts. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Manual/Scripting.html. Hakupäivä 13.3.2013.

27. Boo, C# and JavaScript in Unity –Experiences and Opinions. 2009. Saatavissa: http://forum.unity3d.com/threads/18507-Boo-C-and-JavaScript-in-Unity-Experiences-and-Opinions. Hakupäivä 13.3.2013.

28. Monobehaviour. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.html. Hakupäivä 13.3.2013.

29. Kuvataajuus. 2013. Saatavissa: http://fi.wikipedia.org/wiki/Kuvataajuus. Hakupäivä 15.3.2013.

30. MonoBehaviour.Start. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Start.html. Hakupäivä 13.3.2013.

31. MonoBehaviour.Awake. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Awake.html. Hakupäivä 13.3.2013.

32. MonoBehaviour.Update. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.Update.html. Hakupäivä 13.3.2013.

33. MonoBehaviour.FixedUpdate.Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.FixedUpdate.html. Hakupäivä 13.3.2013.

34. MonoBehaviour.OnGUI. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.OnGUI.html. Hakupäivä 13.3.2013.

35. Overview: Writing Scripts in C# and Boo. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/index.Writing_Scripts_in_Csharp_26_Boo.html. Hakupäivä 13.3.2013.

36. MonoBehaviour.StartCoroutine.2013.Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/MonoBehaviour.StartCoroutine.html. Hakupäivä 13.3.2013.

37. The Window System. Saatavissa: http://wiki.blender.org/index.php/Doc:2.6/Manual/Interface/Window_system. Hakupäivä 13.3.2013.

38. Object Mode and Edit Mode. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Tutorials/Animation/BSoD/Character_Animation/Object_and_Edit_Modes. Hakupäivä 13.3.2013.

39. Selecting Mesh Components. Saatavissa: http://wiki.blender.org/index.php/Doc:2.6/Manual/Modeling/Meshes/Selecting . Hakupäivä 13.3.2013.

40. Introduction. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Rigging. Hakupäivä 13.3.2013.

41. Editing Bones. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Rigging/Armatures/Editing/Bones. Hakupäivä 13.3.2013.

42. Skinning to Shapes. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Rigging/Skinning/ObData#Vertex_Groups. Hakupäivä 13.3.2013.

43. Timeline Window.Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Animation/Timeline. Hakupäivä 13.3.2013.

44. Creating a new Material. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Materials/Assigning_a_material. Hakupäivä 13.3.2013.

45. Introduction. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Textures. Hakupäivä 13.3.2013.

46. UV Explained. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Textures/Mapping/UV. Hakupäivä 13.3.2013.

47. Gimp. 2013. Saatavissa: http://www.gimp.org/. Hakupäivä 14.3.2013.

48. Piirustuslauta (digitointilaite). 2013. Saatavissa: http://fi.wikipedia.org/wiki/Piirustuslauta_(digitointilaite) .Hakupäivä 20.3.2013.

49. The Bamboo Pen tablet. 2013. Saatavissa: http://www.wacom.eu/index2.asp?pid=294&spid=3. Hakupäivä 20.3.2013.

50. Animation View Guide. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Components/AnimationEditorGuide.html. Hakupäivä 15.3.2013.

51. Light. 2013.Saatavissa: http://docs.unity3d.com/Documentation/Components/class-Light.html. Hakupäivä 15.3.2013.

52. Quality Settings. 2013. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-QualitySettings.html. Hakupäivä 14.3.2013.

53. 103. Unity3d Tutorial – Advanced Movement FSM. 2010. Saatavissa: http://www.burgzergarcade.com/tutorials/game-engines/unity3d/103-unity3d-tutorial-advanced-movement-fsm. Hakupäivä 15.3.2013.

54. Character Controller. 2012. Saatavissa: http://docs.unity3d.com/Documentation/Components/class-CharacterController.html. Hakupäivä 15.3.2013.

55. Physics.OverlapSphere. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Physics.OverlapSphere.html. Hakupäivä 15.3.2013.

56. Object.Instantiate. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Object.Instantiate.html. Hakupäivä 12.3.2013.

57. Rigidbody.AddForce. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody.AddForce.html. Hakupäivä 14.3.2013.

58. RigidBody. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Rigidbody.html. Hakupäivä 14.3.2013.

59. Quaternion.EulerAngles.Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Quaternion-eulerAngles.html. Hakupäivä 13.3.2013.

60. Physics.Raycast. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Physics.Raycast.html. Hakupäivä 13.3.2013.

61. Unity Manual/ Ray. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/Ray.html. Hakupäivä 13.3.2013

62. GUI. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/GUI.html. Hakupäivä 13.3.2013.

63. GUI.Window. Saatavissa: http://docs.unity3d.com/Documentation/ScriptReference/GUI.Window.html Hakupäivä 17.3.2013

64. The Legend of Zelda: Skyward Sword. 2013. Saatavissa: http://en.wikipedia.org/wiki/The_Legend_of_Zelda:_Skyward_Sword. Hakupäivä 17.3.2013.

65. Legend of Zelda Maps. Saatavissa: http://ian-albert.com/games/legend_of_zelda_maps/ . Hakupäivä 17.3.2013.

66. Normaali(matematiikka). 2013. Saatavissa: http://fi.wikipedia.org/wiki/Normaali_(matematiikka) Hakupäivä 11.3.2013

67. Mirror Modifier. Saatavissa: http://wiki.blender.org/index.php/Doc:2.4/Manual/Modifiers/Generate/Mirror. Hakupäivä 13.3.2013.

68. Inverse Kinematics. Saatavissa: http://wiki.blender.org/index.php/Doc:2.6/Manual/Rigging/Posing/Inverse_Kinematics. Hakupäivä 13.3.2013.

69. Link. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Link. Hakupäivä 13.3.2013.

70. Stalfos. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Stalfos. Hakupäivä 13.3.2013.

71. Keese. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Keese. Hakupäivä 13.3.2013.

72. Gel. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Gel. Hakupäivä 13.3.2013.

73. Goriya. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Goriya. Hakupäivä 13.3.2013.

74. Blade Trap. 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Blade_Trap. Hakupäivä 13.3.2013.

75. Aquamentus (The Legend of Zelda). 2013. Saatavissa: http://www.zeldadungeon.net/wiki/Aquamentus_%28The_Legend_of_Zelda%29. Hakupäivä 13.3.2013.

76. The Legend of Zelda Items. 2013. Saatavissa: http://www.zeldadungeon.net/Zelda01-the-legend-of-zelda-items.php. Hakupäivä 13.3.2013.

77. Old Man. 2013. Saatavissa: http://www.zeldadungeon.net/Zelda01-the-legend-of-zelda-items.php. Hakupäivä 13.3.2013.

LIITTEET

Liite 1 Resurssien käytettävyys

Liite 2 Julkaisusuunnitelma

Liite 3 Kaikki tuotteen ominaisuudet

Liite 4 Tuotteen ominaisuuslista

Liite 5 Sprintin 1 tehtävälista

Liite 6 Sprintin 2 tehtävälista

Liite 7 Sprintin 3 tehtävälista

Liite 8 Sprintin 4 tehtävälista

OAMK
Resurssien käytettävyys
Projekti: The Legend of Zelda-peli Unity3D-ympäristössä
Scrum Master: -
Aikataulu: 7.1.2013 – 14.3.2013

Henkilö	ARKIPÄIVIÄ PROJEKTIN AIKANA	MUUT PROJEKTIT	LOMAT	MUUT	PROJEKTITYÖN TEKEMISEEN PÄIVIÄ	PROJEKTIYÖN TEKEMISEEN TUNTEJA
Erja Manninen	49	8	5	0	36	252
					0	0
					0	0
Yhteensä	**49**	**8**	**5**	**0**	**36**	**252**

YRITYKSEN NIMI	OAMK
PROJEKTIN NIMI	The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU	7.1.2013 – 14.3.2013

PROJEKTISSA JÄSENIÄ	1
SPRINTTEJÄ PROJEKTISSA	4
TUNTEJA PER PÄIVÄ	7

PROJEKTIN RESURSSIT

HENKILÖT	KÄYTETTÄVISSÄ VIIKKOJA	KÄYTETTÄVISSÄ PÄIVIÄ	KÄYTETTÄVISSÄ TUNTEJA
Erja Manninen	9	36	252
			0
			0
YHTEENSÄ		36	252

SPRINTIN 1 AIKATAULU

HENKILÖ	VIIKKOJA SPRINTISSÄ	PÄIVIÄ SPRINTISSÄ	TUNTEJA SPRINTISSÄ	OMINAISUUKSIA SPRINTISSÄ
Erja Manninen	2	8	56	
			0	
			0	
YHTEENSÄ		8	56	0
SPRINTIN 1 MAALI:				

SPRINTIN 2 AIKATAULU

HENKILÖ	VIIKKOJA SPRINTISSÄ	PÄIVIÄ SPRINTISSÄ	TUNTEJA SPRINTISSÄ	OMINAISUUKSIA SPRINTISSÄ
Erja Manninen	2	8	56	
			0	
			0	
YHTEENSÄ		8	56	0
SPRINTIN 2 MAALI:				

SPRINTIN 3 AIKATAULU

HENKILÖ	VIIKKOJA SPRINTISSÄ	PÄIVIÄ SPRINTISSÄ	TUNTEJA SPRINTISSÄ	OMINAISUUKSIA SPRINTISSÄ
Erja Manninen	2	8	56	
			0	
			0	
YHTEENSÄ		8	56	0
SPRINTIN 3 MAALI:				

SPRINTIN 4 AIKATAULU

HENKILÖ	VIIKKOJA SPRINTISSÄ	PÄIVIÄ SPRINTISSÄ	TUNTEJA SPRINTISSÄ	OMINAISUUKSIA SPRINTISSÄ
Erja Manninen	3	12	84	
			0	
			0	
YHTEENSÄ		12	84	0
SPRINTIN 4 MAALI:				

HENKILÖ	VIIKKOJA YHTEENSÄ	PÄIVIÄ YHTEENSÄ	TUNTEJA YHTEENSÄ	OMINAISUUKSIA TUOTTEESSA
Erja Manninen	9	36	252	
			0	
			0	
YHTEENSÄ		36	252	0
JULKAISUN MAALI	**Pelaaja voi pelata läpi Zelda 1:n ensimmäisen luolaston.**			

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013

Ominaisuuden ID

Prioriteetti korkea	Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus ...	Sprintin numero	Toteutuksen kesto (=Story Point)	Huomioitavaa
1	Luo pohja pelimaailmalle (projekti ja terrain).	1	1	
2	Tee hahmon ohjaus, (Third Person Character Controller).	1	1	
3	Luo kamera, joka kuvaa hahmoa ja siirtyy pelaajan mukana kun huone vaihtuu (Camera Controller).	1	1	
4	Mallinna luolasto, jonka sisällä pelaaja pelaa (pelin kenttä).	1	2	
5	Mallinna pelihahmo ja tee hahmolle kävelyanimaatio.	1	3	
6	Tee inventory-ruutu (GUI), josta pelaaja voi tarkastella omistamiaan esineitä ja vaihtaa käytössä olevan esineen.	2	1	
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot	2,3	1.5,1	Jatkuu seuraavaan sprinttiin
8	Pelaajan ja hahmon välinen vuorovaikutus, pelaaja ottaa vahinkoa vihollisesta ja pystyy tuhoamaan ne.	2	0.5	
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)	2,3	1,2	Jatkuu seuraavaan sprinttiin
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.	3,4	1,3	Jatkuu seuraavaan sprinttiin
12	Viholliset tiputtaa satunnaisesti rahaa, sydämiä ja pommeja	2	0.5	
13	Vihollisten "tekoäly", ts liikkuminen ympäriinsä.	2	1,0	
14	Äänet ja musiikit peliin	4	2,0	
15	Mallinna pelissä olevia itemeitä ja esineitä	2,4	0,5,3	Jatkuu seuraavaan sprinttiin
16	Piirrä grafiikkaa ja teksturoi pelissä olevia itemeitä ja asioita.	2	0.5	
17	Aloitusruutu josta peli alkaa ja lopetusruutu, grafiikka niihin	4	2	
18	Pelin GUI jossa näkyy pelaajan health, käytössä olevat itemit, kerätyt rahat ym.	2	0.5	
19	Mallinna luolaston loppuvastus ja tee animaatiot, vihollinen ampuu tulipalloja.	3	2	
20	Kartan ja kompassin toimintojen lisääminen	4	1	
21	Efektien tekeminen peliin	3	2	

Prioriteetti matala	Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus ...	Sprintin numero	Toteutuksen kesto (=Story Point)	Huomioitavaa
21	Luolastossa näkyy kartta joka kertoo missä huoneessa pelaaja on	4	1	

Toivottava	Ominaisuuden kuvaus, käyttäjätarina, toiminnallisuus ...	Sprintin numero	Toteutuksen kesto (=Story Point)	Huomioitavaa

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013

Tilakoodit
Valmis
Kesken
Ei aloitettu
Hylätty

Ominaisuuden ID	Ominaisuuden kuvaus	Sprintin numero	1	2	3	4		
		Sprintissä päiviä	8	8	8	12	Huomioitavaa	Tila
1	Luo pohja pelimaailmalle (projekti ja terrain).		1					Valmis
2	Tee hahmon ohjaus, (Third Person Character Controller).		1					Valmis
3	Luo kamera, joka kuvaa hahmoa ja siirtyy pelaajan mukana kun huone vaihtuu (Camera Controller).		1					Valmis
4	Mallinna luolasto, jonka sisällä pelaaja pelaa (pelin kenttä).		2					Valmis
5	Mallinna pelihahmo ja tee hahmolle kävelyanimaatio.		3					Valmis
								Valmis
	SPRINTIN 1 MAALI:Pelihahmon liikkumiseen liittyvät asiat kuten kamera, animaatiot ja controllerit toimii							
6	Tee inventory-ruutu (GUI), josta pelaaja voi tarkastella omistamiaan esineitä ja vaihtaa käytössä olevan esineen.			1,0				Valmis
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot			1,5				Valmis
8	Pelaajan ja hahmon välinen vuorovaikutus, pelaaja ottaa vahinkoa vihollisesta ja pystyy tuhoamaan ne.			0,5				Valmis
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)			1,0				Valmis
12	Viholliset tiputtaa satunnaisesti rahaa, sydämiä ja pommeja			0.5				Valmis
13	Vihollisten "tekoäly", ts liikkuminen ympäriinsä.			1.0				Valmis
15	Mallinna pelissä olevia itemeitä.			1,5				Valmis
16	Piirrä grafiikkaa ja teksturoi pelissä olevia itemeitä ja asioita.			0.5				Valmis
18	Pelin GUI jossa näkyy pelaajan health, käytössä olevat itemit, kerätyt rahat ym.			0,5				Valmis
	SPRINTIN 2 MAALI: Pelaajan ja vihollisten, sekä tiettyjen alueiden ja esineiden välillä on vuorovaikutusta							
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot				1			Valmis
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)				2			Valmis
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.				1			Valmis
19	Mallinna luolaston loppuvastus ja tee animaatiot, vihollinen ampuu tulipalloja.				2			Valmis
21	Efektien tekeminen peliin				2			Valmis
	SPRINTIN 3 MAALI: Pelaaja voi taistella vihollisia vastaan ja kerätä esineitä, sekä edetä luolastossa							
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.					3		Valmis
14	Äänet ja musiikit peliin					2		Valmis
15	Mallinna pelissä olevia itemeitä ja esineitä					3		Valmis
17	Aloitusruutu josta peli alkaa ja lopetusruutu, grafiikka niihin					2		Valmis
20	Kartan ja kompassin toimintojen lisääminen					1		Valmis
21	Luolastossa näkyy kartta joka kertoo missä huoneessa pelaaja on					1		Valmis
	SPRINTIN 4 MAALI: Pelissä on toimiva kartta, lisää käytettäviä aseita ja aloitus ja lopetusruutu							
	JULKAISUN MAALI: Pelaaja voi pelata läpi Zelda 1:n ensimmäisen luolaston.							

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013
SPRINTIN AIKATAULU: 7.1.2013 – 18.1.2013

SPRINTIN 1 MAALI: SPRINTIN 1 MAALI:Pelihahmon liikkumiseen liittyvät asiat kuten kamera, animaatiot ja controllerit toimii

				ma	ke	to	pe	ma	ke	to	pe			
			Kalenteripäivä	7.1	9.1	10.1	11.1	14.1	16.1	17.9	18.1			
			Sprintin päivä	1	2	3	4	5	6	7	8			YHTEENSÄ
			Päiväkohtainen resurssi tunneissa	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0			56,0
			Päivän aikana tehty tuntimäärä	0,0	6,0	6,0	7,0	7,0	12,0	10,0	10,0	0,0	7,5	65,5
Ominaisuuden/Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**													**Tehtävään käytetty tuntimäärä**
1	Luo pohja pelimaailmalle (projekti ja terrain).	Tekijä(t)		Tehdyt tunnit sprintin aikana										
	Luo projekti ja pohjaa johon peli alkaa rakentumaan	EM			1,0									1,0
														0,0
														0,0
2	Tee hahmon ohjaus, (Third Person Character Controller).													
	Yksinkertainen Controlleri jolla hahmoa voi liikuttaa nuolinäppäimillä	EM			1,0									1,0
	Advanced characterController												1,0	1,0
														0,0
														0,0
														0,0
3	Luo kamera, joka kuvaa hahmoa ja siirtyy pelaajan mukana kun huone vaihtuu (Camera Controller).													
	Kamera scripti, kamera seuraa pelaajaa tietyn välimatkan päästä	EM			2,0									2,0
	Kamera scripti, kamera on paikallaan ja se kääntyy pelaajan liikkuessa.	EM			2,0									2,0
	Tutki pelaajan paikka ja laske näiden arvojen perusteella huoneelle indeksi	EM						2,5						2,5
	Lisää jokaista huonetta varten taulukkoon paikka kameraa varten. Kameran ja huoneen indeksit täsmäävät	EM						2,0						2,0
	Siirrä kamera uuteen pisteeseen kun pelaaja kävelee eri huoneeseen	EM						1,0						1,0
	Pysäytä kamera tietyn kulman jälkeen	EM						1,0						1,0
														0,0
														0,0
4	Mallinna luolasto, jonka sisällä pelaaja pelaa (pelin kenttä).													
	Luolaston mallintaminen ja teksturointia					4,0	7,0							11,0
	Lisää colliderit Unityn puolella							0,5					0,5	1,0
														0,0
														0,0
														0,0
5	Mallinna pelihahmo ja tee hahmolle kävelyanimaatio.													0,0
	Blender tutoriaalien katsominen.					2,0								2,0
	Hahmon mallintaminen								12,0	7,0				19,0
	Hahmon riggaaminen eli luurangon laittaminen									3,0	5,0		3,0	11,0
	Hahmon animointi										5,0		3,0	8,0
														0,0

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013
SPRINTIN AIKATAULU: 21.1.2013 - 1.2.2013

SPRINTIN 2 MAALI: Pelaajan ja vihollisten, sekä tiettyjen alueiden ja esineiden välillä on vuorovaikutusta

				ma	ke	to	pe	ma	ke	to	pe			
			Kalenteripäivä	**21.1**	**23.1**	**24.1**	**25.1**	**28.1**	**29.1**	**30.1**	**1.2**			
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**			**YHTEENSÄ**
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**			**56,0**
			Päivän aikana tehty tuntimäärä	**8,0**	**7,0**	**9,0**	**8,0**	**9,0**	**0,0**	**8,5**	**9,0**	**0,0**	**0,0**	**58,5**
Ominaisuuden/Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**													**Tehtävään kägtetty tuntimäärä**
6	Tee inventory-ruutu (GUI), josta pelaaja voi tarkastella omistamiaan esineitä ja vaihtaa käytössä olevan esineen.	(t)		**Tehdgt tunnit sprintin aikana**										
	Tee inventory window										1,0			1,0
	lisää eri elementit										1,0			1,0
	Nuolilla pystyy liikkumaan inventoryssa jos siellä on tavaraa										1,0			1,0
	valittuna oleva item näkyy erillisessä laatikossa ja ilmestyy myös GUI:n										1,0			1,0
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot													
	Luurangon (Stalfos) mallintaminen					6,0	8,0	3,0						17,0
	Riggaa luuranko							3,0						3,0
	Animoi luuranko							3,0						3,0
8	Pelaajan ja hahmon välinen vuorovaikutus, pelaaja ottaa vahinkoa vihollisesta ja pystyy tuhoamaan ne.													
	Pelaajan lyömisanimaation tekeminen ja sen lisääminen Unityn puolella kun painetaan nappia			2,0										2,0
	Kun vihollista lyödään, siltä lähtee healthia				1,0					1,0				2,0
	Linkiltä voi lähteä myös puolikas sydän				1,5									1,5
	vihollisessa olevassa scriptissä on eri tyyppisille vihollisille eri verran healttia				1,5									1,5
	Kun viholliseen törmää, lähtee Linkiltä healttia									1,0				1,0
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)													
	Lukitusta ovesta ei pääse kulkemaan ellei ole avainta			0.5										0,0
	Ovi häviää jos sinulla on mukanasi avain									1,0				1,0
12	Viholliset tiputtaa satunnaisesti rahaa, sydämiä ja pommeja													
	Eri esineistä prefabit, toistaiseksi nappia painamalla voi luoda rahaa tai muita esineitä			1,0										1,0
	Lisää LootEnemy-Scripti viholliseen niin että se tiputtaa esineen, random-generaattori joka arpoo minkä esineen/ei mitään vihollinen tiputtaa										1,0			1,0
	Joku tietty viholinen tiputtaa avaimen jota tarvitsee edetäkseen luolastossa										1,0			1,0
13	Vihollisten "tekoäly", ts liikkuminen ympäriinsä.													
	Luurangon tekoäly									4,0				4,0
15	Mallinna pelissä olevia itemeitä.													
	Luolastoon lisätty kivet, vesi ,salahuoneeseen korkeampi taso ja näihin tekstuurit				3,0	3,0								6,0
	Tee tikkaat joilla pääsee salahuoneeseen													0,0
	Tee ovi ja lukko									1,0				1,0
	Colliderit veteen ja kiviin										1,0			1,0
16	Piirrä grafiikkaa ja teksturoi pelissä olevia itemeitä ja asioita.													
	Piirrä grafiikkaa GUI:a varten			2,0										2,0
	Piirrä grafiikkaa Inventoryruutua varten													0,0
	Tee tekstuurit Linkin vaatteita, ihoa ja hiuksia varten. Piirrä yksityiskohtia kuten silmät ja kilpi .										2,0			2,0
18	Pelin GUI jossa näkyy pelaajan health, käytössä olevat itemit, kerätyt rahat ym.													
	Tee scripti GUI:ta varten, sijoittele kuvat ruutuun			2,0										2,0
	Kun pelaaaja kävelee itemin kohdalle, item katoaa ja itemin lukumäärää kasvatetaan GUI:ssa			1,0										1,0
	Kun itemi käytetään lukumäärää vähennettään									0,5				0,5

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013
SPRINTIN AIKATAULU: 4.2.2013-15.2.2013

SPRINTIN 3 MAALI: Pelaaja voi taistella vihollisia vastaan ja kerätä esineitä, sekä edetä luolastossa

				ma	ke	to	pe	ma	ke	to	pe			
			Kalenteripäivä											
			Sprintin päivä	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**			**YHTEENSÄ**
			Päiväkohtainen resurssi tunneissa	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**	**7,0**			**56,0**
			Päivän aikana tehty tuntimäärä	**7,0**	**8,0**	**11,5**	**8,0**	**8,0**	**8,0**	**7,0**	**7,0**	**0,0**	**0,0**	**64,5**
Ominaisuuden/Tehtävän ID	**Ominaisuuden/tehtävän kuvaus**													**Tehtävään käytetty tuntimäärä**
7	Mallinna luolastossa olevia vihollisia ja tee niille liikkumisanimaatiot	**Tekijä(t)**		**Tehdyt tunnit sprintin aikana**										
	Mallinna Keese, lepakko	EM		5,0										5,0
	Teksturoi Keese	EM		2,0										2,0
	Laita luuranko	EM			1,0									1,0
	Animoi	EM			1,0									1,0
	Siirrä Unityyn ja laita toimimaan	EM			1,0									1,0
	Mallinna trap, teksturoi ja tee animaatio	EM									2,0			2,0
	Mallinna Gel, teksturoi ja tee animaatio	EM				1,5			2,0					3,5
9	Luolaston logiikka (ovet jotka aukeavat avaimella, ovet jotka aukeavat kiveä työntämällä, seinä joka räjähtää pommilla.)													
	Tee kivi jota voi työntää, ja ovi aukeaa	EM							1,0					1,0
	Tee spawnpointit vihollisille ja luo näistä spawnpointeista viholliset kun astutaan huoneeseen	EM						7,0						7,0
	Lisää Slamdoorit peliin, kun astutaan tiettyyn huoneeseen, ovi avautuu vasta kun kaikki huoneen viholliset on tapettu	EM							2,0					2,0
	Lisää pikkuinen kartta inventoryyn ja GUI:n (Toiminnallisuus jos kerkee)	EM									1,0			1,0
	Yleistä luolaston logiikkaa, itemeitä, ovia, vihollisia ym.	EM							2,0	2,0				4,0
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.													
	Pommi tuhoaa viholliset jos ne ovat tarpeeksi lähellä	EM									1,0			1,0
	Pommia ei voi tehdä jos niitä ei ole	EM												0,0
19	Mallinna luolaston loppuvastus ja tee animaatiot, vihollinen ampuu tulipalloja.													
	Mallinna Aquamentus, lohikäärme	EM			5,0	9,0								14,0
	Teksturoi Aquamentus	EM				1,0	3,0							4,0
	Laita luuranko	EM					2,0							2,0
	Animoi	EM					1,0							1,0
	Siirrä Unityyn, tee Scriptit jossa lohikäärme liikkuu ja hyökkää	EM					1,0							1,0
	Lohikäärme ampuu tulipalloja	EM					1,0							1,0
21	Efektien tekeminen peliin													
	Pommi räjähtää, tulta ja savua ja sen jälkeen pommi katoaa	EM								2,0	1,0			3,0
	Seinä räjähtää pirstaleiksi	EM								2,0	1,0			3,0
	Kun vihollista lyö se välähtää	EM						1,0						1,0
	Vihollisen kuoleminen	EM									1,0			1,0
	Oven katoaminen kun käytetään avain	EM												0,0
	Oven pamahtaminen kiinni	EM							1,0	1,0				2,0
														0,0

YRITYKSEN NIMI: OAMK
PROJEKTIN NIMI: The Legend of Zelda-peli Unity3D-ympäristössä
PROJEKTIN AIKATAULU: 7.1.2013 – 14.3.2013
SPRINTIN AIKATAULU: 18.2.2013-8.3.2013

SPRINTIN 4 MAALI: Pelissä on toimiva kartta, lisää käytettäviä aseita ja aloitus ja lopetusruutu

			ma	ke	to	pe	ma	ti	to	pe	ma	ti	ke	to	pe	
		Kalenteripäivä														
		Sprintin	1	2	3	4	5	6	7	8	9	10	11	12		YHTEENSÄ
		ohtaien resurs	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	7,0	91,0
		aikana tehty tuntim	6,0	7,5	7,0	13,0	7,0	8,0	7,0	8,0	9,5	6,0	5,0	7,0	2,0	93,0
Ominai-suuden/Teh-täävän ID	Ominaisuuden/tehtävän kuvaus	Tekijä(t)		Tehdyt tunnit sprintin aikana												Tehtävään käytetty tuntimäärä
11	Itemeiden käyttö, bumerangi, miekka, jousi ja pommi.															
	Jousipyssyn toiminnallisuuden lisääminen peliin eli sillä voi ampua			2,0												2,0
	Bumerangin toiminnallisuuden lisääminen peliin									4,0						4,0
	Vanha mies antaa vihjeen dungeonista, häneltä voi myös ostaa pommeja tai sydämiä rahaa vastaan									2,0						2,0
14	Äänet ja musiikit peliin															
	Tee audioscripti ja tyhjät peliobjektit kaikille äänille				2,0											2,0
	Lisää oikeat äänet oikeisiin paikkoihin ja testaa				3,0											3,0
	Lisää taustamusiikki titlescreeniin										0,5					0,5
15	Mallinna pelissä olevia itemeitä ja esineitä															
	Mallinna ja teksturoi kartta ja vie unityyn			1,0												1,0
	Mallinna ja teksturoi jousipyssy ja vie unityyn			1,5												1,5
	Mallinna ja teksturoi nuoli ja vie unityyn			0,5												0,0
	Mallinna ja teksturoi bumerangi ja vie unityyn						2,0									2,0
	Mallinna,teksturoi ja animoi vanha mies ja vie unityyn						5,0	3,0								8,0
	Mallinna ja teksturoi patsas ja vie unityyn							3,0								3,0
	Muokkaa Aquamentuksesta patsas ja vie unityyn							2,0								2,0
	Mallinna roihu, teksturoi ja vie unityyn								3,0							3,0
	Mallinna Goriya									2,0	5,0					7,0
	Animoi ja teksturoi Goriya										3,0					3,0
	Siirrä ja laita kuntoon Goriyaan liittyvät jutut Unityssa											2,0				2,0
17	Aloitusruutu josta peli alkaa ja lopetusruutu, grafiikka niihin															
	Piirrä TitleScreen				2,0	10,0										12,0
	Lisää uusi Scene aloitusruutua varten.					1,0										1,0
20	Kartan ja kompassin toimintojen lisääminen															
	Pelissä on 2 karttaa, toinen näkyy inventoryssa ja toinen GUI:ssa		2,0													2,0
	Kun kartan on löytänyt, kaikki luolaston huoneet näkyvät GUI:ssa		1,0													1,0
	Kun kompassin on löytänyt näkyy missä huoneessa loppubossi on		1,0													1,0
21	Luolastossa näkyy kartta joka kertoo missä huoneessa pelaaja on															
	GUI:n kartassa näkyy missä huoneessa pelaaja on		1,0													1,0
	Inventoryssa olevassa kartassa näkyy missä huoneessa pelaaja on ja missä huoneissa on käyty		1,0													1,0
	muuta															
	Floortrappien toiminta kuntoon, ne lähtevät kohti pelaajaa kun kävellään tietystä kohdasta ja palaavat takaisin			3,0												3,0
	yleistä fiksausta					2,0			4,0			4,0	3,0			13,0
	Korjaa pelihahmon animaatiot													1,0		1,0
	Korjaa kameran toiminnallisuutta															0,0
	Korjaa valaistusta													2,0		2,0
	Vaihda itemien ikonit														2,0	2,0
	Lisää soihtuja huoneisiin.													1,0		1,0
	Lisää feidaukset kun scene vaihtuu ja kun mennään kellariin.										1,0					1,0
	Nosta huoneiden seinien korkeutta												2,0			2,0
	Pelihahmon kuoleminen(spawnaa takaisin alkuun, Itemit,rahat,pommit säilyy													3,0		3,0
	lisää 5-rupee item												0.5			0,0