Markus Hänninen

PELIPROJEKTIN TOTEUTUS UNITYLLÄ

Opinnäytetyö
Tietojenkäsittely

Toukokuu 2013

MIKKELIN AMMATTIKORKEAKOULU
Mikkeli University of Applied Sciences

KUVAILULEHTI

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Opinnäytetyön päivämäärä** 31.5.2013
Tekijä(t) Markus Hänninen	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittely
Nimeke Peliprojektin toteutus Unityllä	

Tiivistelmä

Pelienkehitys on muuttunut 2000-luvulla paljon helpommin lähestyttäväksi. Enää ei tarvitse olla laiteläheisen ohjelmoinnin osaaja, vaan pelikoodin kirjoittaminen onnistuu myös esimerkiksi JavaScriptillä. Myös peliprojektien koot ovat jakautuneet aina yhden henkilön kehittämistä omakustannepeleistä jopa sadan miljoonan euron budjetin peleihin.

Kävin läpi tässä opinnäytetyössä erilaisia pelinkehitysympäristöjä kuten XNA, Unity, Source SDK ja Unreal Developement Kit. Näistä paneuduin tarkemmin Unityyn esittelemällä sen käyttöliittymää ja sen tarjoamaa pelimoottoria. Keskityin myös sen tarjoamiin työkaluihin ja siihen, mitä peliprojektin tekemiseen Unityllä tarvitsee tietää.

Kerroin opinnäytetyössä myös peliprojektin eri tuotantovaiheista yleisesti. Näihin kuuluvat pelin suunnittelu, käsikirjoittaminen, ohjelmointi, peliobjektien tekeminen sekä markkinointi ja levitys. Lisäksi kävin läpi, miten osa näistä asioista näkyy Unityllä töitä tehdessä.

Käytännöntoteutuksena tein pelin Unityllä JavaScriptiä käyttäen. Peli on ensimmäisestä persoonasta kuvattu seikkailupeli, jossa pelaajan tehtävänä on ratkaista pulmia. Pelin ohjelmoinnin lisäksi kävin läpi hieman pelin 3D-mallien tekoa ja pelisuunnittelua. Päätännössä tiivistin ajatuksiani projektin onnistumisesta ja pienistä peliprojekteista yleisesti.

Asiasanat (avainsanat)

JavaScript, peliohjelmointi, peliteollisuus

Sivumäärä	**Kieli**	**URN**
34	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi Jukka Selin	**Opinnäytetyön toimeksiantaja** Mikkelin ammattikorkeakoulu

DESCRIPTION

MIKKELIN AMMATTIKORKEAKOULU Mikkeli University of Applied Sciences	**Date of the bachelor's thesis** 31st of May 2013
Author(s) Markus Hänninen	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

Completion of a game project with Unity

Abstract

Game development is now more accessible than it ever was. Nowadays good looking games can be made with almost any kind of basic programming languages like JavaScript with no need to even know about low-level programming. The size of a game project can also vary from a small indie game with no budget to a big AAA game with a budget of 100 million euros.

The first part in this bachelor's thesis was about some of the most used game development environments. The environments included were XNA, Unreal Development Kit, Source SDK and Unity. From these environments Unity was the one that I studied more thoroughly. I explained how its user interface worked and was required when working with Unity.

I also explained how game production worked step by step. How games proceed from a design document to a product that could be sold and what kind of marketing games need. I also introduced how these steps show up when making a game with Unity.

The practical part was this thesis is a game made with Unity. I used JavaScript and Unity to make a first-person point-and-click adventure game. This part introduced how I planned the game and how I turned the plans into a game. The last part of this bachelor's thesis is a summary of how the game project succeeded and some of my personal thoughts about indie game development.

Subject headings, (keywords)

JavaScript, game programming, game industry

Pages	**Language**	**URN**
34	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	Mikkeli University of Applied Sciences

SISÄLTÖ

1 JOHDANTO 1

2 TYÖVÄLINEET 2

2.1 XNA (C#) 2

2.2 Unreal Development Kit 3

2.3 Source Development Kit 4

2.4 Unity 5

3 UNITY 6

4 PELIPROJEKTI 10

4.1 Suunnittelu 10

4.2 Käsikirjoittaminen 11

4.3 Peliobjektit 12

4.4 Ohjelmointi 13

4.5 Markkinointi ja levitys 14

5 CASE: PELI 16

5.1 Suunnittelma 16

5.1.1 Pelin idea 17

5.1.2 Pelimekaniikka 18

5.1.3 Pelin pulmat 19

5.2 Käsikirjoitus 20

5.3 Peliobjektit 21

5.4 Ohjelmointi 24

5.4.1 Interaktio 25

5.4.2 Inventory 28

5.5 Markkinointi ja levitys 30

6 PÄÄTÄNTÖ 32

LÄHTEET 35

1 JOHDANTO

Pelien kehittäminen on viimeisen muutaman vuoden aikana muuttunut radikaalisti. Siinä missä aikaisemmin alalla oli mahdollisuudet vain ammattilaisille isolla budjetilla, nyt kenen tahansa on mahdollista tehdä näyttävän näköinen peli. Erilaiset pelinkehitysympäristöt kuten Unity ja XNA ovat helpottaneet pelin tekemiseen tarvittavaa ohjelmointia niin paljon, että asiaan aikaisemmin tutustumatonkin voi oppia pelien tekemisen pääpiirteet melko vaivattomasti.

Tämän opinnäytetyön aikana kehitänkin pelin ja tutkimusongelmani on yhden miehen peliprojektin loppuun vieminen. Aloitan pelin tekemisen paperille suunnittelusta ja jatkan aina alun ja lopun sisältävään pelattavaan tuotokseen asti. Työvälineenä tässä käytän Unity Technologiesin Unity-pelinteko-ohjelmaa. Opinnäytetyön toimeksiantaja on Mikkelin ammattikorkeakoulu.

Opinnäytetyön toisessa luvussa kerron erilaisista suosituista työvälineistä pelien tekemiseen. Näihin kuuluvat Unityn lisäksi XNA, Unreal Development Kit ja Source Development Kit. Kerron lyhyesti mitä ne ovat, mikä niiden historia on ja missä peleissä niitä on käytetty.

Kolmannessa luvussa keskityn edellisessä luvussa mainituista pelintekovälineistä Unityyn hieman syvällisemmin. Kerron sen käyttöliittymästä, erilaisista toiminnoita ja siitä miten sitä käytetään. Mainitsen myös mihin kaikkeen sen pelimoottori pystyy ja miten paljon yksinkertaisempaa sen käyttäminen on kuin perinteisen peliohjelmointiympäristön.

Neljäs luku keskittyy pelien tekemiseen yleisesti. Kerron tässä pelin kehitysvaiheet aina suunnitelmasta markkinointiin asti ja mitä ne vaiheet tarkoittavat. Mainitsen myös miten jotkin asiat näkyvät Unityllä pelejä tehdessä.

Viides luku eli Case keskittyykin omaan peliprojektiini. Luku koostuu samoista alaluvuista kuin neljäskin luku, mutta kerron tässä ne asiat tietenkin peliprojektini näkökulmasta. Aloitan pelin suunnittelulla ja käsikirjoittamisella ja sitten siirryn peliobjektien tekemiseen ja koodin kirjoittamiseen. Lopuksi käsittelen mitä mahdollisuuksia minulla olisi pelini markkinoimisen suhteen.

Päätännössä tiivistän projektini kulun ja sen mitä sain aikaiseksi. Keskityn myös siihen, mitä parannettavaa projektissani on ja miten jatkan eteenpäin projektin kehityksen suhteen. Puntaroin myös koko pelintekoprosessia yhden miehen pelikehittäjän näkökulmasta.

2 TYÖVÄLINEET

Tässä luvussa esittelen mahdollisia työkaluja pelinkehittämiseen. Käyn läpi vain joitain suosittuja työvälineitä ja nämä eivät todellakaan ole ainoat mahdollisuudet, vaan lähinnä niitä useiden pienempien ja keskikokoisten kehittäjien käyttämiä. Isommilla pelitaloilla on usein omakehitteisiä pelimoottoreita, joita ei jaeta oman talon ulkopuolelle.

2.1 XNA (C#)

XNA on Microsoftin vuonna 2006 julkaisema .NET-kirjastoista koostuva pelinkehitysympäristö, jossa käytetään C#-ohjelmointikieltä. Sillä voi tehdä pelejä Windows-, Xbox 360- ja Windows Phone-laitteille. XNA poikkeaa perinteisestä peliohjelmoinnista olemalla aloittelijaystävällisempi ja vähemmän työläs ympäristö. (What is the XNA Framework 2006.)

XNA helpottaa peliohjelmointia sillä, ettei tarvitse keskittyä kuin peliohjelman kirjoittamiseen. Perinteisessä peliohjelmoinnissa täytyy ennen varsinaisen pelikoodin kirjoittamista tehdä asioita kuten ikkunan luontia tai grafiikkasovittimien ja näytöntilojen listausta. Nämä asiat eivät liity mitenkään itse peliohjelman sisältöön, vaan ovat pakollisia, että peli näkyisi ja toimisi oikein laitteella. Myös itsestään selvät asiat, kuten pelihahmon näkyminen ruudulla tai näppäinpainalluksien tutkiminen vaativat perinteisessä peliohjelmoinnissa paljon aikaa ja koodia. XNA:ssa nämä hoituvat huomattavasti vähemmällä työllä. (What is the XNA Framework 2006.)

XNA:ta käytetään usein indie-peleissä ja muissa pienemmissä projekteissa. Indiepeleiksi kutsutaan usein pienten ryhmien tekemiä pelejä, joihin ei ole hankittu rahoitusta ulkoiselta julkaisijalta. Tämmöisiä pelejä, toisin kuin isoja pelijulkaisuja, ei myydä näyttävillä grafiikoilla, tunnetuilla ääninäyttelijöillä ja isoilla mainoskampan-

joilla, vaan esimerkiksi hauskalla pelimekaniikalla tai massaviihteestä eroavalla tarinankerronnalla. (Dutton 2012.)

KUVA 1. Fez (Fez 2012.)

Fez (kuva 1) on hyvä esimerkki XNA:lla tehdystä indie-pelistä (Bédard 2009). Pelissä pelataan Gomez-nimisellä hahmolla, joka elää kaksiulotteisessa maailmassa. Maagisen fetsi-hatun saatuaan hän pystyy näkemään kolmannenkin ulottuvuuden. Koko pelimekaniikka perustuukin siihen, että pelaaja pystyy kääntelemään neliskulmaista pelimaailmaa nähdäkseen sen kaikista eri sivuista kaksiulotteisesti. (Fez 2012.)

2.2 Unreal Development Kit

Unreal Engine on Epic Gamesin tarjoama pelimoottori, joka syntyi alun perin *Unreal*-pelin (kuva 2) moottoriksi vuonna 1998, mutta kehittyi siitä myöhemmin yhdeksi yleisimmin käytetyistä ensimmäisen ja kolmannen persoonan räiskintäpelien moottoreista. Unreal Engine kuuluu nykyään myös Unreal Development Kitiin, jolla pystyy tekemään ammattilaistasoisia pelejä helppokäyttöisellä työkalusetillä. Se käyttää UnrealScript-nimistä ohjelmointikieltä. (Thomsen 2010.) Viimeisin julkaistu versio eli Unreal Engine 3.0 mahdollistaa pelien tekemisen Applen iOS:lle, Windowsille, Adobe Flashille, Androidille, Xbox 360:lle, Mac OS:lle, Playstation 3:lle, Playstation Vitalle ja Nintendon Wii U:lle (Platforms 2012).

KUVA 2. Unreal (Unreal #9 2012.)

Unreal Enginellä on tehty nimensä mukaisesti *Unreal*-sarjan kaikki osat, mutta tämän lisäksi sitä on käytetty erittäin moneen hyvin suosittuun peliin kuten *Gears of War*-trilogia, *BioShock*-sarja ja *Mass Effect*-trilogia. Nämä kolme sarjaa ovat niin sanottuja AAA-pelejä eli budjetiltaan ja myyntiluvuiltaan monet isot Hollywood-elokuvatkin syrjäyttäviä pelejä.

2.3 Source Development Kit

Valve Softwaren vuonna 2004 julkaisema Source Engine ilmestyi yhtiön suosittujen jatko-osien *Counter-Strike: Source:*n ja *Half-Life 2*:n (kuva 3) pelimoottorina. Se on jatkokehitetty GoldSrc-pelimoottorista, joka taas on raskaalla kädellä muokattu id Softwaren *Quake*-pelimoottori. Yleiseen käyttöön se päätyi Source SDK-paketin yhteydessä. Sourcella on tehty pelejä myös Xbox 360:lle ja Playstation 3:lle, mutta pääosin kyseessä on kuitenkin PC- ja Mac-pelien maailmaan tarkoitettu pelimoottori. (Thomsen 2009.)

KUVA 3. Half-Life 2 (Half-life 2 2012.)

Vaikkei Source Enginellä tehdäkään paljoa kolmannen osapuolen isoja projekteja, on se erittäin suosittu modaajien keskuudessa. Modaajiksi kutsutaan henkilöitä, jotka sovittavat esimerkiksi karttapaketteja, graafisia muokkauksia tai vaikka kokonaan uusia pelejä jonkin suositun pelin moottoriin ja pelimekaniikkaan. Muun muassa Valven suositut nettipelit *Team Fortress* ja *Counter-Strike* saivat alkunsa tällä tavoin fanien käsissä. (Thomsen 2009.)

2.4 Unity

Unity on Unity Technologiesin kehittämä ilmainen ympäristö, jolla tehdään pelejä kaikille pelaamiseen tarkoitetuille ympäristöille. Sen ensimmäisen version kehitys aloitettiin jo vuonna 2001 ja lopulta julkaistiin vuonna 2005 Applen Worldwide Developers Conference-tapahtumassa. (Fun Facts 2013.) Unityn pelimoottori on suunniteltu skaalautumaan eri laitteiden välillä niin hyvin, että teoriassa sama peli voisi toimia kännykällä ja PC:llä. Tämän takia sillä ei saa aikaiseksi niin yksityiskohtaisinta grafiikkaa kuin joillain kilpailijoiden tuotteilla. Unitystä on myös olemassa maksullinen ammattilaisversio, joka tarjoaa enemmän toiminnallisuuksia. (License Comparisons 2012.)

KUVA 4. Rochard (Rochard 2012.)

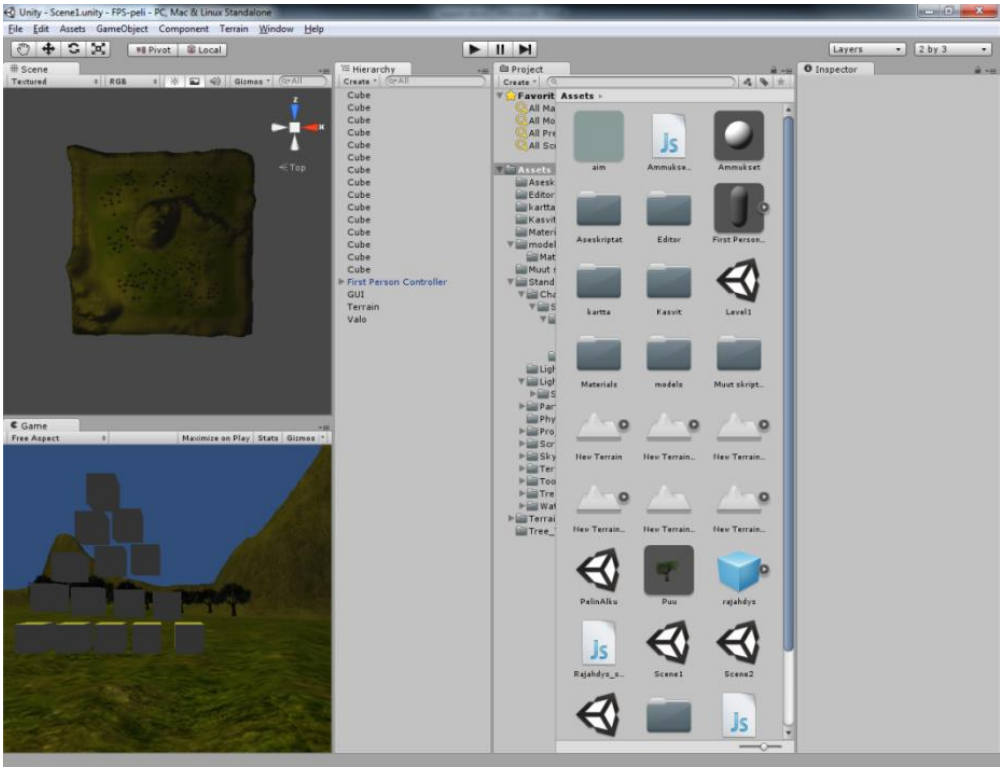
Unityä ovat käyttäneet projekteissaan esimerkiksi suomalainen Recoil Games pelissään *Rochard* (kuva 4) ja Rovion *Bad Piggies*:ssä. Nämä kaksi ovat esimerkkejä siitä, että Unity taipuu myös 2D-peleihin, vaikka kyseessä onkin 3D-pelimoottori. Peleissä on lukittu kamera sivulle ja hahmot voivat liikkua vain kaksiulotteisesti ylös, alas ja sivuille, mutta eivät ollenkaan syvyyssuunnassa. (Rochard's Producer on the Unity Engine 2011.)

3 UNITY

Unityä voisi kuvailla pienkehittäjien tulevaisuudeksi. Sen avulla pelejä voi oppia tekemään muutkin kuin pelkät ohjelmoijat, koska sen käyttöliittymä muistuttaa enemmän 3D-mallinnus- tai piirto-ohjelmia kuin koodieditoria. Unityssä onkin jo valmiina pelimoottori, joten tarvittava ohjelmointi keskittyy lähinnä erilaisiin pelissä tarvittaviin toimintoihin kuten vaikkapa aseella ampuminen tai tietokoneen ohjaamien hahmojen tekoäly. Ohjelmointi suoritetaan käyttäjän haluamalla tekstieditorilla ja koodipätkät lisätään erillisinä skriptatiedostoina projektiin.

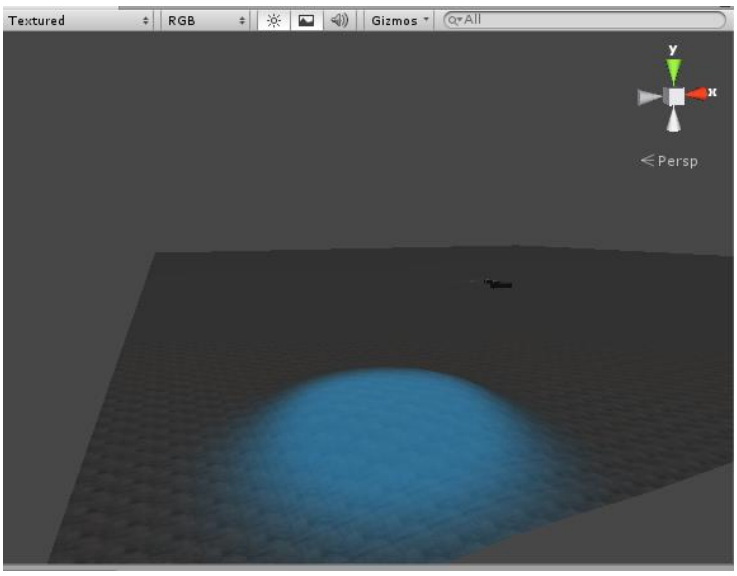
Unityn käyttöliittymä (kuva 5) koostuu erilaisista ikkunoista. Tärkeimmät näistä ovat Scene-kuva, Game-kuva ja Assets-valikko. Scene-kuvassa pystyy katselemaan pelimaailmaa eri perspektiiveista, maalaamaan pinnanmuotoja ja lisäämään objekteja esimerkiksi 3D-malleja siihen. Game-kuvassa taas näytetään pelikuva livenä ja As-

sets-valikon kautta tehdään kaikki objektien, kooditiedostojen ja vastaavien hallinta ja muokkaus.



KUVA 5. Unityn käyttöliittymä

Maailman luonti Unityssä on hyvin yksinkertaista. Se aloitetaan tekemällä nelikulmion muotoinen Terrain ja tätä sitten muokataan piirtotyökalujen kaltaisilla välineillä. 3D-maailman muodot ja tekstuurit maalataan erilaisilla pensseleillä (kuva 6). Maanpintaa voi pensseleillä nostaa tai laskea tietyltä alueelta. 3D-mallit voidaan tuoda erilaisissa formaateissa kuten vaikka FBX-tiedostona ja malleja voidaan liikutella hiirellä haluamaansa paikkaan kartalla.



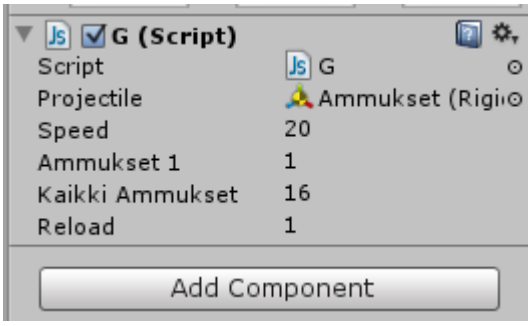
KUVA 6. Pinnanmuotojen maalaaminen Unityssä

Projektiin tuoduille 3D-malleille voidaan asettaa törmäyksentunnistuksen lisäksi fysiikka eli paino ja se vaikuttaako painovoima niihin. Pelimoottorissa mukana oleva *NVIDIA PhysX*-fysiikkamoottori hoitaa täten kappaleiden tippumisen, vierimisen ja pomppimisen reaaliajassa pelimaailmassa. Sillä saa aikaiseksi myös tuuliefektin esimerkiksi vaikuttamaan puiden lehdistöön ja hahmon vaatteisiin. Fysiikan laskeminen monelle kappaleelle yhtäaikaisesti on kuitenkin raskasta tehokkaillekin tietokoneille ja pelin optimoinniksi olisi hyvä pistää fysiikat vain välttämättömille objekteille. (Physics 2013.)

Unityllä pääsee siis hyvin pitkälle ilman ohjelmointia, mutta kuitenkin kaikki vaativampi pelilogiikka ja toiminnot tarvitsevat käyttäjän kirjoittamaa koodia. Ohjelmointi hoidetaan JavaScript-, C#- tai Boo-ohjelmointikielillä. Eri toiminnallisuuksista voi kirjoittaa oman kooditiedoston, jonka liittää sitten Unityn puolella siihen peliobjektiin, johon tämä liittyy.

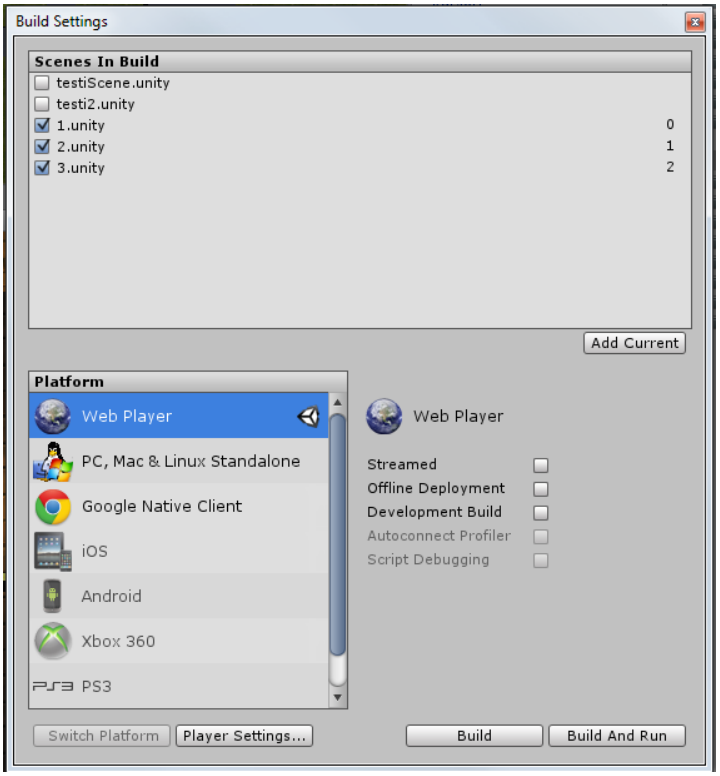
Toiminnallisuus voi olla vaikkapa se, että hiiren vasenta nappia painamalla pelihahmo ampuu hiiren kursorin osoittamaan suuntaan pallon. Tällöin kooditiedosto pitää liittää peliobjektiin, jolla ammutaan (ase) ja määrittää tiedoston näkymässä mainittu ammus sitä vastaavaan 3D-malliin. Kuva 7:ssa aseen 3D-malliin on liitetty G-niminen JavaScriptiä sisältävä tiedosto, jonka ammuksia (Projectile) ovat Ammukset-niminen 3D-malli, jonka lentorataan taas vaikuttaa PhysX-fysiikkamoottori. JavaScript-tiedostossa määriteltyjen muuttujien arvot sekä näkee, että ovat muokattavissa suoraan

Unityn editorin puolella. Niitä voi muuttaa Unityn editorissa ilman, että ne vaikuttavat alkuperäisen tiedoston koodiin.



KUVA 7. Peliobjektiin lisätty JavaScript-tiedosto

Peliprojekti Unityllä koostuu sceneistä. Jos pelissä on esimerkiksi monta tasoa, niin jokainen taso voi olla oma scenensä. Sceneä pystyy vaihtamaan pelin sisällä lyhyellä koodinpätkällä ja näin ollen säästetään koneen tehoja, kun kaikkia tasoja ei tarvitse ladata yhtäaikaisesti. Jokaisessa scenessä voi olla oma karttaeditorilla luotu maailmansa ja vaikka kokonaan erilainen pelimekaniikka.



KUVA 8. Unity-projektin kääntäminen

Kun pelin haluaa julkaista, täytyy se ensin kääntää. Kääntövaiheen asetuksissa (kuva 8) valitaan mitkä kaikki scenet tulevat peliin ja missä järjestyksessä. Yksi scene voi

olla esimerkiksi pelin alkuvalikko ja toinen pelin lopputekstit. Tämän jälkeen valitaan mille alustalle se käännetään. Unityn ilmaisversiossa on mahdollista kääntää nettiselaimessa toimivaan muotoon, tietokoneella ajettavaksi standalone- eli työpöytäsovellukseksi (PC, Linux ja Mac) ja Googlen Native Clientillä toimivaan muotoon.

4 PELIPROJEKTI

Tässä luvussa kerron yleisesti, mitä kaikkea peliprojektiin kuuluu. Peliprojekti alkaa usein suunnitelmalla josta tehdään sitten käsikirjoitus. Vasta sitten kun peli on suunniteltu niin hyvin, että kaikilla on selkeä visio mitä ollaan tekemässä, aletaan tehdä itse ohjelmointityötä ja peligrafiikan valmistamista. Pelin kehitysvaiheessa ollaan jo yleensä tietoisia mille alustoille peli julkaistaan ja missä hintaluokassa, koska optimointi alustoille aloitetaan jo kehityksen alkuvaiheessa. Markkinointi on iso osa peliprojektin jälkituotantoa.

4.1 Suunnittelu

Pelisuunnittelu tarkoittaa ideoiden keksimistä. Sitä miten ideoista tehdään peli ja ideoiden ylläpitoa koko pelintekoprosessin ajan. Pelisuunnitteluun kuuluu myös idean myyminen muille ja muiden palautteen kuunteleminen. Pelisuunnittelija saattaa rakastua omaan ideaansa liian syvästi ja unohtaa, ettei ulkopuolinen pelaaja välttämättä pidäkään siitä tai ymmärrä sitä. (Fulletron 2008.)

Hyvä pelisuunnittelija toimii eräänlaisena tulkkina suunnittelutiimissä. Graafikot ja ohjelmoijat eivät välttämättä ymmärrä toistensa toimialaa, mutta hyvällä pelisuunnitelmalla kaikilla pitäisi olla käsitys projektin kokonaisuudesta. Pelisuunnittelija on myös vastuussa ideoidensa laajentamiseen ja muokkaamiseen kehitystiimin ja ajanpuutteen tuomien rajoituksien mukaan. (Fulletron 2008.)

Mikä siis on pelisuunnitelma? Pelisuunnitelmassa kerrotaan muutakin kuin pelkkä pelin juoni. Siinä on mietitty pelin *gameplay*tä eli pelaajan interaktiota pelimaailmassa, pelimaailman rakennetta eli tasosuunnittelua, hahmojen välistä dialogia, pelin säännöt, pelin kohdeyleisö ja muuta vastaavaa. Pelisuunnitelma voi muuttua projektin edetessä, mutta sen olisi hyvä silti olla melko tarkka ennen itse peliohjelman ja siihen

liittyvien asioiden tekemisen aloittamista jo pelkästään resurssien säästämiseksi. (Fulletron 2008.)

Gameplay on pelin tärkein anti. Vaikka pelin tarina olisikin hyvä, niin pelaaja kyllästyy peliin nopeasti, jos itse pelaaminen on tylsää. Gameplayn määritelmä vaihtelee lähteestä toiseen, mutta yleisesti ottaen sillä tarkoitetaan pelaajan ja pelin välistä interaktiota. Tähän sisällytetään kaikki millä pelaaja voi vaikuttaa pelimaailmassa, kuten päähahmon liikuttaminen, miekan huitominen tai vaikka toisille pelaajille puhuminen.

Pelin kohdeyleisön voi ottaa huomioon valikoimalla pelitestaajaryhmän. Pelitestaajat pelaavat peliä samalla, kun pelisuunnittelija ottaa muistiinpanoja asioista, mitkä asiat ovat pelaajille liian helppoja ja mitkä aiheuttavat pelin etenemisen pysähtymisen liian pitkäksi aikaa. Jos peliä tekevät osapuolet testaisivat peliä itse, niin he eivät osaisi ottaa kantaa pelin ongelmiin niin suoraan kuin henkilöt, joilla ei ole minkäänlaista näkemystä pelin suunnitteluvaiheista.

Pelin tasosuunnittelu pitää tehdä kokonaan pelaaja mielessä. Pelitaso voi näyttää hyvältä, mutta jos se on suunniteltu pelkästään ulkonäön puolesta ilman mitään noteerausta pelimekaanisiin ominaisuuksiin, niin se voi sisältää monia turhauttavia ongelmia pelaajalle. Esimerkiksi tasohyppelypeleissä pelaaja turhautuu, jos jatkuvat häviöt johtuvat viallisesta tasosuunnittelusta, eivätkä hänen omista taidoistaan. Siinä vaiheessa pelkkä kaunis ulkonäkö ei pelasta peliä.

4.2 Käsikirjoittaminen

Pelit ovat muuttuneet yksinkertaisesta pisteidenkeräämisestä elokuvateollisuutta haastavaan popcorn-viihteeseen. Tämän takia nykyajan peleissä on paljon dialogia, tarinankäänteitä ja kinemaattisia välianimaatioita. Jotta tämä kaikki pysyisi kasassa, on peli pakko käsikirjoittaa.

Ennen käsikirjoittamisen aloittamista on kuitenkin hyvä olla jonkinlainen näkemys siitä millainen pelistä tulee. Jos on tekemässä kovin dialogivoittoista käsikirjoitusta, niin pelin tyylisuunta ei voi välttämättä olla tasohyppely tai peli joka rakentuu pelkästään pisteiden keräämisen ympärille. Hyvän roolipelin tekeminen taas ei onnistu ilman isoa määrää hahmojen ja maailman taustatarinaa.

Pelikäsikirjoittaja Richard Danskyn (2012) mukaan pelit ovat ainoa media, joiden käsikirjoituksessa ei kerrota kirjailijan omaa tai päähenkilön tarinaa, vaan pelaajan tarinaa, koska pelaaja omaksuu protagonistin roolin ja on interaktiossa pelin maailman kanssa. Vaikka pelaaja ottaisikin ikonisen ja mediassa tunnetun hahmon kuvakseen pelissä, niin kaikki hahmon tekemät asiat ovat nimenomaan pelaajan tahdosta riippuvaisia.

Nykyään pelien käsikirjoitukset ovatkin melko monimutkaisia ja niistä vastaavat usein ammattilaiset kirjailijat. Parhaana esimerkkinä tästä ovat *Tom Clancy* -sarjan pelit, jotka perustuvat samannimisen kirjailijan teoksiin tai käsikirjoituksiin. Pelikäsikirjoitus eroaa elokuvista ja kirjoista siinä määrin, että pelaajalla voi olla mahdollisuus tehdä valintoja: esimerkiksi valita joko hyvä tai paha polku. Vuonna 2012 ilmestyneessä *Dishonored*-pelissä pelaajalla on mahdollisuus aiheuttaa kaaosta ja olla psykopaattinen massamurhaaja tai sitten olla salamyhkäinen ja näkymätön henkilö, joka ei tapa ketään koko pelin aikana. Pelin maailman kohtalo ja muiden hahmojen vuorovaikutus pelin päähahmoa kohtaan muuttuvat molemmilla eri pelitavoilla aivan päinvastaisiksi.

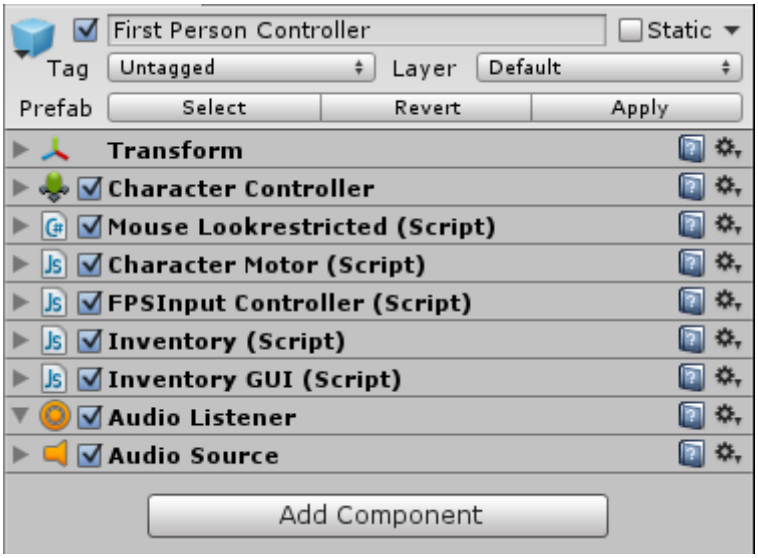
Pelin käsikirjoittaja ei kuitenkaan ole peliprojektin pomo, joka määrää kaikista asioista. Pelien tapauksessa käsikirjoittaminen on vieläkin enemmän yhteistyötä kuin esim. elokuvien tapauksessa. Käsikirjoittaja joutuu työskentelemään pelin tasosuunnittelijoiden, artistien ja pelisuunnittelijoiden kanssa, jotta kaikki sopisi yhteen. (Owen 2013.)

4.3 Peliobjektit

Unityssä pelissä olevia objekteja kutsutaan peliobjekteiksi (GameObject). Peliobjekti ei itsessään tee mitään. Se tarvitsee erilaisia ominaisuuksia voidakseen muuttua hyödylliseksi pelin kannalta. (Gameobjects 2013.) Esimerkiksi pelihahmo ja pelimaailma kaikkine sisältöineen ovat peliobjekteja.

Peliobjektia voisi kutsua säiliöksi. Kuten kuvasta 9 voidaan nähdä, niihin säilötään erilaisia komponentteja, jotka riippuvat ihan siitä minkälainen peliobjekti on kyseessä. (Gameobjects 2013.) Jos kyseessä on ensimmäisen persoonan räiskintäpeli, niin pelin päähahmossa pitää olla komponentteina ainakin pelitilannetta hahmon silmistä kuvaa-

va kamera, hahmon ohjauksen määrittävät koodinpätkät, aseet omine koodeineen ja törmäyksentunnistus. Komponentti on siis hyvin laaja käsite, joka kattaa kaiken aina kamerasta ääniin, ja siitä kooditiedostoihin ja efekteihin.



KUVA 9. Iso määrä komponentteja

Pelaajan näkökulmasta peliobjektit näkyvät vain siis 3D-mallina, muuna grafiikkana tai eivät näy ollenkaan, mutta niissä on tietenkin paljon enemmän sisältöä. Peliobjekti on se kohta pelissä, jossa artistien, suunnittelijoiden ja ohjelmoijien tekemä sisältö yhdistyy. Artistin puolesta se voi olla vaikka 3D-malli vihollisesta, johon ohjelmoija on tehnyt suunnitelmien pohjalta tekoälyn.

4.4 Ohjelmointi

Videopelien alkuaikana 70-luvun alusta 80-luvun puoliväliin videopelin ohjelmoija oli usein myös suunnittelija ja artisti. Videopelit olivat tähän aikaan hyvin yksinkertaisia ja jopa pelin grafiikat piirrettiin koodissa määräämällä pikseleitä tiettyihin paikkoihin. Esimerkiksi Taiton vuonna 1978 julkaisema kolikkopeli *Space Invaders* (kuva 10) on vain yhden miehen, *Tomohiro Nishikadon*, voimin tehty projekti niin ohjelmointia kuin laitesuunnittelua myöten. (Edwards 2007.) Nykypäivänä pelin ohjelmoijat ovat vain osa tiimiä, jotka pistävät artistien piirtämän grafiikan, esimerkiksi 3D-mallit, liikkumaan ruudulla ja ohjelmoivat pelisuunnittelijan määrittelemät säännöt pelimaailmaan.

KUVA 10. Space Invaders (Reyes 2010.)

Pelejä voi ohjelmoida oikeastaan millä tahansa ohjelmointikielellä, koska pelin määritelmä on niin laaja. Nykyään iso osa peleistä kuitenkin ohjelmoidaan C-, C++-, C#- tai Java-ohjelmointikielillä. C++ on näistä käytetyin.

Peliohjelmoinnissa yksi keskeisimmistä komponenteista on *game loop* eli vapaasti käännettynä pelisilmukka. Se määrittelee kuinka monta kertaa sekunnissa peliruutu piirretään, pelaajan painalluksia tarkistetaan ja kaikkea ruudulla liikkuvaa päivitetään. (Gamedev Glossary: What is the ”Game Loop”? 2012.) Unityn tapauksessa peliä päivitetään erilaisissa Update-funktioissa: FixedUpdate, Update ja LateUpdate, jotka luettelin tärkeysjärjestyksessä useimmiten päivitettävästä vähimmiten päivitettävän. (Execution Order of Event Functions 2012.)

4.5 Markkinointi ja levitys

Vielä 10 vuotta sitten, jos halusit pelata videopelejä, sinun täytyi kävellä pelejä myyvään erikoisliikkeeseen tai Anttilan tapaiseen ketjuliikkeeseen. Tämän jälkeen piti toivoa, että haluamaasi peliä löytyy hyllystä ja sinulle sopivaan hintaan. Nykyään tämän kaiken voi hoitaa kotoa käsin vain nappia painamalla.

Ladattavia pelejä myyvät Internet-palvelut ovat hyvin suosittuja PC-pelien puolella. *Steamin*, *Originin* ja *GOG*:n kaltaiset palvelut myyvät pelin digitaalisena ja käyttäjätiliin liitettynä. Sinun ei siis tarvitse huolehtia CD-levyn kunnosta, vaan voit ladata pelin milloin tahansa uusiksi, kunhan muistat vain käyttäjätunnuksesi ja salasanasi.

Tämä sekä matkapuhelinpuolella *Google Play* ja Applen *AppStore* ovat helpottaneet pelintekijöiden talousongelmia paljon. Pelin ei tarvitse enää olla budjetiltaan iso, siitä ei tarvitse painaa montaa fyysistä kopiota ja mikä parasta sen voi hinnoitella itse, koska tuote on digitaalinen ja rahaa ottavia välikäsiä on paljon vähemmän. Tämän takia nykyään pelimarkkinoilla on erilaisia hintaluokkia toisin kuin 20 vuotta sitten. Pelin hinta voi olla melkein mitä tahansa ilmaisesta yhteen euroon ja siitä monen sadan euron keräilypaketteihin.

Budjetiltaan isommat pelit painetaan kuitenkin edelleen levyille, mutta ne ovat tämän lisäksi saatavissa myös digitaalisina. Teknologia ei ole vielä kehittynyt niin paljoa, että jokainen pelaava ihminen ostaisi pelinsä digitaalisesti, mutta pienemmän budjetin peleille se on useassa tapauksessa ainoa levityskanava. Jopa jokaisella uudemmalla videopelikonsolilla on omat verkkokauppansa.

Digitaalinen levitys on tuonut markkinoille myös uusia markkinointimalleja peleille. On olemassa kuukausimaksullisia pelejä kuten esimerkiksi erittäin suosittu *World of Warcraft*. Tätä markkinointimallia käyttävät pelit ovat yleisimmin *Massive Multiplayer Online* -pelejä (MMO) eli isoja pelaajamääriä tukevia nettipelejä, joissa on pelattavaa sisältöä miltei loputtomiin.

Koska kuukausimaksullinen malli ei enää kannata kuin muutamassa harvinaisessa tapauksessa, niin markkinoille tuli *Free-to-play* eli ilmaispelimalli. Tätä mallia edustavat useiden MMO-pelien lisäksi muun muassa monet nettiräiskintäpelit kuten *Team Fortress 2* ja useat Facebook-pelit. Ilmaispelimalli perustuu siihen, että ydinpeli on ilmainen, mutta kaikki lisäsisältö maksaa. Lisäsisältöä voi olla esimerkiksi kosmeettiset esineet pelimaailman sisällä, uudet pelattavat hahmot tai vaikkapa lisää pelattavaa peliin. Esimerkiksi *Lord of the Rings Online* -nettipelin tapauksessa, markkinointimallin muuttaminen kuukausimaksullisesta ilmaispeliksi triplasi pelin tuottamat rahamäärän. (Orland 2011.)

Miten pelistään saa nykyään jaettua informaatiota? Mainostaminen sanan varsinaisessa mielessä on vain isompien pelien hommaa, koska kustannukset tv- ja lehtimainontaan ovat niin isoja. Pienemmät pelit saavat usein julkisuutta peliaiheisten nettisivustojen kautta arvosteluissa ja artikkeleissa. Myös YouTubesta on tullut iso alusta indie-

pelikehittäjille mainostamiseen. Videopelikommentaattori John Bain tekee YouTubeen nimimerkillä *TotalHalibut* videosarjaansa ”WTF is…”, jossa hän esittelee odotettujen uutuuspelien lisäksi tuntemattomampia tuotoksia. Videot keräävät jopa satojen tuhansien katsojamääriä.

Sonyn PlayStation 3:lla, Microsoftin Xbox 360:llä ja Nintendon Wii- sekä 3DS-konsoleilla on kaikilla omat latauspalvelunsa, joita niitä on kritisoitu monesta asiasta. Toisin kuin PC-puolen vastaavissa, niissä ei pelin päivittäminen ole ilmaista kehittäjälle. Veteraanipelinkehittäjä Tim Schafer mainitsi Hookshot inc.:n haastattelussa (2012), että pelin päivittäminen konsolilla voi maksaa jopa kymmeniä tuhansia euroja päivityksen koosta riippumatta. Tässä hintaluokassa olevat päivitysmaksut ovat Schaferin mukaan pienelle kehittäjälle niin isoja, ettei niitä yksinkertaisesti pysty maksamaan. Näin ollen konsoliversiot joistain pienemmän budjetin peleistä ovat täynnä erilaisia vikoja, joita ei pystytä korjaamaan vaikka asia on kehittäjien tiedossa.

Pelikonsolivalmistajat ovat kuitenkin sanoneet tulevaisuuden olevan toista tämän suhteen. Nintendon uusin konsoli eli Wii U tarjoaa lisensoiduille kehittäjille ilmaisen Unity Pro-version ja mahdollisuuden hinnoitella pelinsä itse. (Nutt 2013.) Myös PlayStation 4:n on luvattu sen julkaisevan Sonyn sanoin olevan hyvä ympäristö indiekehittäjille. (Webster 2013.)

5 CASE: PELI

Tässä luvussa kerron kaikki neljännen luvun asiat oman projektini näkökulmasta. En ole pelinkehityksessä ammattilainen, joten tämä ei välttämättä anna täydellistä kuvaa pelinkehittäjien arjesta. Haasteena on kuitenkin alun, lopun ja pelimekaniikan sisältävän ja pelattavissa olevan pelin tekeminen.

5.1 Suunnittelma

Tässä alaluvussa puhun sitä, minkälainen peli tämä tulee olemaan. Kerron lyhyesti pelin ideasta, pelimekaniikasta ja pulmien suunnittelusta. En kuitenkaan paneudu jokaiseen pelin pulmaan, koska niihin tulee muutoksia jatkuvasti. Niitä pitää hioa pelintekoprosessin edetessä, kun huomaa esimerkiksi pelimoottorin puolesta tai suunnitelman monimutkaisuudesta johtuvia ongelmia

5.1.1 Pelin idea

Pelin ideana on ensimmäisen persoonan seikkailupeli kauhuvivahteilla. Ideaan vaikuttivat suosikkipelini kautta aikain: *The Secret of Monkey Island* (kuva 11) ja *Day of the Tentacle*, jotka ovat *LucasArtsin* julkaisemia seikkailupelejä. Niiden kantava idea on etsiä pelimaailmasta erilaisia esineitä ja ratkaista niiden ja erilaisten dialogivalintojen perusteella mitä oudoimpia pulmia hyvin tarinavetoisessa maailmassa. Tämänkaltaiset pelit olivat kuitenkin yleensä kaksiulotteisia pelkästään hiirellä ohjattavia seikkailuita. Oman projektini tapauksessa pelin päähahmoa ohjataan hiirellä ja näppäimistöllä. Sen pelimekaniikka on tästä johtuen lähempänä ensimmäisen persoonan räiskintäpelejä kuin vanhoja seikkailupelejä.

KUVA 11. The Secret Of Monkey Island (The Secret of Monkey Island 2009.)

Koska yksi pelin genreistä on kauhu, tapahtuu iso osa pelistä pimeässä tyrmässä. Se että pelaaja ei näkisi ollenkaan, olisi tietenkin huonoa pelisuunnittelua. Tämän takia pelaajalla on auttavana voimanaan taskulamppu, jolla näkee vain rajatulle alueelle. Taskulampun kanssa olisi kuitenkin liian helppoa edetä, jos sen saisi pitää kokoajan päällä. Tämän takia taskulampussa on kuluvat patterit, jotka latautuvat itsestään vain kun lamppu on poissa päältä. Näin pelaaja joutuu tasapainottelemaan pimeän ja valon tasapainoa itse.

Pelissä ei ole varsinaisia fyysisiä vihollisia. Ainoa vihollinen pelaajalle on pimeys ja sen tuomat pelkotilat. Pelaajan pitää siis ratkoa *puzzleja* eli seikkailupeleille ominaisia loogisia ongelmia taustalla kuuluvan pelottavan ambient-musiikin ja satunnaisten pelottavien äänien keskellä.

Pelissä on myös *inventory* eli inventaario. Siellä näkyvät hallussa olevat esineet ja tiettyä esinettä klikkaamalla saa sen käyttöönsä, joka näkyy käytännössä esineen ollessa hiiren kursorina. Käytössä olevaa esinettä voi testata erilaisiin *triggereihin* eli ”kytkimiin” ympäristössä. Jos käyttää oikean esineen oikeaan kytkimeen, esimerkiksi avaimen oveen, niin pelitilanne muuttuu ja pelaaja pääsee eteenpäin.

5.1.2 Pelimekaniikka

Pelissä on mahdollisuus katsella maailmaa vapaasti kameralla, jonka kulma on ohjattavissa hiirellä. Tämä kamera kuvastaa päähahmon päänliikkeitä. Hahmon fyysisen sijainnin muuttaminen tapahtuu taasen WASD- tai nuolinäppäimillä. F:stä pelihahmo nostaa taskulampun, jonka valokeilaa voi muokata Q- ja E-näppäimillä

Liikkumisen lisäksi pitää huomioida pelin *point-and-click*-elementti. Koska hiirellä ohjataan hahmon katsetta, olisi siinä samalla vaikeaa osoittaa kursorilla jotain kohdetta, koska kameran kulma muuttuu kokoajan. Tämän takia pelissä on mahdollisuus lukita kameran kulma hiiren oikealla napilla ja siirtyä ohjaamaan kursoria, jolla voi vuorovaikuttaa pelin ympäristöön. Kursori näkyy vain napin ollessa pohjassa ja muuttaa väriä, kun se koskettaa jotain objektia, mikä on poimittavissa tai käytettävissä.

Jos pelaaja klikkaa esinettä, joka on poimittavissa, häviää se peliruudusta ja siirtyy pelaajan pohjattomiin taskuihin. Painamalla I:tä aukeaa inventaario, jossa kaikki poimitut esineet näkyvät erilaisina ikoneina. Ikonia klikkaamalla esine tulee käytettäväksi. Kun esine on käytettävissä, sen vuorovaikutus toisiin esineisiin hoituu myöskin klikkaamalla.

KUVA 12. Game Over

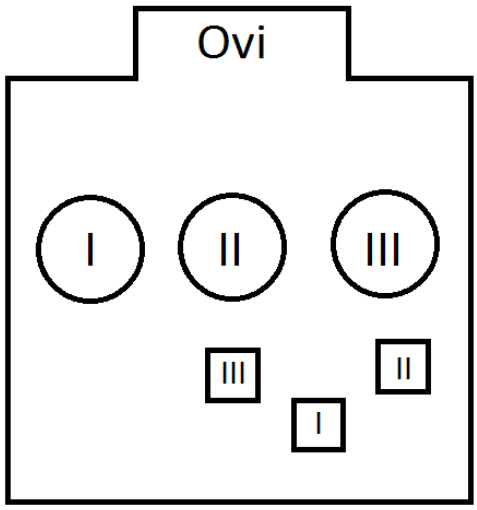
Vaikkei peli rankaise virheistä, niin siinä voi kuolla. Toistaiseksi kuoleminen on kuitenkin mahdollista vain yhdessä kohdassa. Jos pelin alkuvaiheessa sijaitsevassa käytävässä taaksepäin, voikin huomata edellisen huoneen hävinneen. Jos jatkaa siltikin kävelemistä taaksepäin, niin tippuu omaan kuolemaansa ja saa ”Game Over”-tekstin ruutuun (kuva 12).

5.1.3 Pelin pulmat

Pelin pulmien suunnittelu on haastava pulma itsessäänkin. Niitä suunnitellessa pitää ottaa huomioon, että pelaajalla on mahdollisuus ratkaista ne, mutta kuitenkin pelin olematta liian helppo. Pulmissa pitää siis olla tarpeeksi haastetta, että pelaaja viihtyy etsiessään ratkaisua. Kuitenkin jos pulma on liian vaikea, pelaaja ei jaksa enää yrittää muutaman kokeilun jälkeen. Siksi pelin pitääkin viestiä, jos ratkaisu on menemässä edes osittain oikein.

Tämän takia pelissä on hyvä säilyttää eräänlainen vaikeuskäyrä. Pelin alussa on vain yksinkertaisia ja ohjeistettuja pulmia, jotka kuka tahansa pystyy ratkaisemaan. Tässä vaiheessa kuitenkin vain opetetaan miten pelimekaniikka toimii. Kun pelaaja on oppinut pelimekaniikan, pitää sekoittaa pakkaa ja tehdä pulma, joka vaatii omaaloitteisuutta pelaajalta. Tällä linjalla vaikeuskäyrää on hyvä nostaa pelin loppuun saakka nostamatta vaikeustasoa kuitenkaan liian korkealle.

Esimerkiksi tämän pelin ensimmäinen pulma tapahtuu pimeässä huoneessa, josta ei näyttäisi olevan pääsyä ulos (kuva 13). Huoneessa ei ole pelihahmon lisäksi muuta kuin kolme numeroitua nappia ja numeroituja kuutioita. Osaamalla yhdistää numeroidun kuution ja numeroidun napin oikein, ovi aukeaa. Jos kuutiot taas asettaa väärässä järjestyksessä napeille, niin mitään ei tapahdu.



KUVA 13. Ensimmäinen pulma

5.2 Käsikirjoitus

Peli alkaa tilanteessa, jossa pelaajan ohjastama hahmo löytää itsensä pimeästä huoneesta ilman mitään muistikuvaa sinne joutumisesta. Pelaajalla on turvanaan vain taskulamppu ja omat päättelykykynsä. Tästä alkaakin seikkailu erilaisia pulmia ratkoessa (katso luku 5.1.3). Tämä on pohjustus pelin tarinalle.

Pelin taustatarinaa kerrotaan maailmasta löytyvissä kirjoissa, joita on siroteltu ympäri peliä. Koska peli on yhden miehen projekti, taustatarinaa ei ole paljoa, mutta se vähäinen on tehty auttamaan pelaajaa ongelmatilanteissa. Esimerkiksi pelin toisesta huoneesta löytyvässä kirjassa (kuva 14) on neuvoja sen huoneen pulman ratkaisuun. Vihje on kerrottu päiväkirjan muotoon. Kirjat tulevat ruutuun läpinäkyvinä ikkunoina, joissa on nappi sulkemista varten.

KUVA 14 Yksi pelin kirjoista

Näin ollen voisi sanoa pelin käsikirjoituksen valmistuvan samaa tahtia kuin muukin peli. Käsikirjoitan taustatarinaa lisää aina, kun peliin tulee pulmia lisää. Taustatarina on kuitenkin lähestulkoon aina pelivihjeiden muodossa.

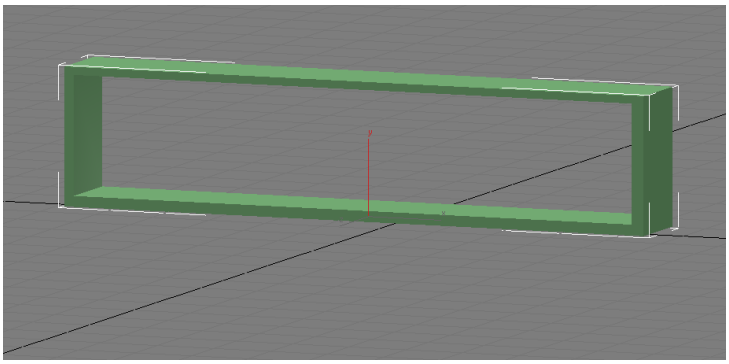
5.3 Peliobjektit

Pelin erillisten 3D-mallien tekemiseen olen käyttänyt Autodeskin *3ds Max Design 2011*:sta. Useat peliobjektit ovat kuitenkin laillisesti erilaisista rojaltivapaista lähteistä hankittuja, eivätkä näin ollen ole omatekemiä. Keskityn tässä luvussa siis lähinnä vain niihin itseni tekemiin asioihin.

Pelin ulkomaailmat (kuva 15) on tehty tietenkin Unityn omalla Terrain-editorilla, mutta niiden yksityiskohtiin ei ole käytetty kovin paljoa vaivaa, koska isoin osa pelistä tapahtuu 3ds Maxilla tehtyjen rakennelmien sisällä. Pelin ensimmäinen terrain näkyy pelaajalle 10 sekunnin ajan, kun hän onnistuu pakenemaan pelin ensimmäisestä rakennuksesta. Tämän jälkeen pelin kohtaus vaihtuu ja pelaajan hahmo siirtyy toisen rakennuksen sisälle siirtymäefektin tahdittamana.

KUVA 15. Pelin ensimmäinen terrain

Kaikki pelin rakennelmat, kuten huoneet ja käytävät, ovat 3ds Maxilla tehtyjä. Niiden luomiseen käytin ohjelman perusmuodoista *Box*- eli laatikkomuotoa. Sen mittasuhteita muuttamalla saa helposti aikaiseksi nelikulmion muotoisia huoneita ja pitkiä nelikulmion muotoisia käytäviä. Ontoksi huoneet ja käytävät sain ohjelmasta löytyvällä Boolean-työkalun *substraction*-operaatiolla, jolla saa poistettua toisesta objektista toisen objektin kokoisen osan pois. Pistämällä esimerkiksi pienemmän laatikon isomman sisälle, saa isommasta laatikosta onton kuten kuvasta 16 voidaan nähdä.



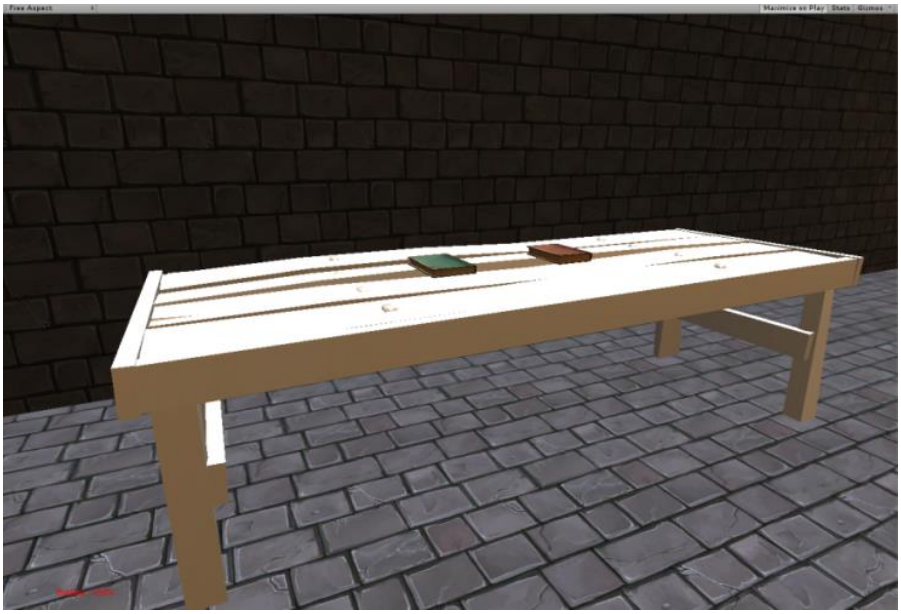
KUVA 16. Laatikko substraction-operaation jälkeen

Kun pelin ensimmäisestä huoneesta pääsee ulos, päätyy pitkään käytävään (kuva 17). Tämä käytävä on toteutettu tekemällä yksi lyhyt pätkä käytävää, jota on sitten aseteltu saumattomasti toistensa perään. Yhden käytävänpätkän varustukseen kuuluu kummal-

lakin seinällä olevat valoa tuottavat soihdut ja kolmet kuvioidut matot. Soihtujen liekkiefekti valaistuksineen on tuotettu Unityn Standard Assets-paketin mukana tulevalla Flame-efektillä.

KUVA 17. Käytävä

Unityn Asset Storesta pystyy ostamaan erilaisia 3D-malleja ja lataamaan ne suoraan käytettäväksi omaan projektiinsa. Asset Storeen pystyy kuka tahansa pistämään myyntiä omia 3D-mallejaan, tekstuurejaan, audio-tiedostojaan tai vaikka kokonaisia peliprojekteja määrittelemäänsä hintaan. Toin omaan projektiini tätä kautta muutamia ilmaisia 3D-malleja lisäämään vähän sisältöä pelimaailmaani. Kuvassa 18 on Asset Storesta ladattu puupöytä ja kaksi kirjaa.



KUVA 18. Asset Storesta ladattuja 3D-malleja

3D-mallien lisäksi pelissä on myös ns. näkymättömiä peliobjekteja. Näihin kuuluvat mm. pelialueen rajaavat *Box Colliderit* eli näkymättömät laatikon muotoiset seinät, joiden läpi pelaaja ei pysty kävelemään. Myös esimerkiksi hiiren kursori on hoidettu omalla peliobjektillaan, koska pelissä ei ole Windowsin oma kursori käytössä.

5.4 Ohjelmointi

Tässä peliprojektissa ohjelmointi on jaettu useaan erilliseen JavaScript-tiedostoon. Jokainen tiedosto ajaa joko tietyn asian päivittämistä jokaisessa ruudulla näkyvässä kuvassa tai sitten ajoittaa tietyn asian tapahtumaan tietyllä hetkellä. Tietty hetki voi olla vaikka, kun pelihahmo kävelee merkatun alueen yli tai tekee jotain. Kaikki koodi on JavaScriptillä kirjoitettua. Päädyin käyttämään JavaScriptiä, koska sillä on niin iso tuki Unityn yhteisössä.

Pelihahmon ohjaamiseen käytetään Unityn mukana tulevaa valmista hahmomoottoria ”First Person Controller”, joka on suunniteltu ensimmäisen persoonan pelejä varten. En ole muokannut yhtään tätä valmista komponenttia, joten en kerro siitä tämän enempää tämän opinnäytetyön aikana. Keskitynkin koodinpätkiin, joita ei tule Unitys-sä valmiina vaan ne on joko itse kirjoitettu tai otettu joistain vapaassa levityksessä olevista projekteista.

Yksi tärkeimmistä ominaisuuksista pelissä on hahmon ja pelin esineiden välinen interaktio. Tämä vaatii monta eri JavaScript-tiedostoa: interaktio.js, Inventory.js, Item.js ja InventoryGUI.js. Näistä tiedostoista jokainen tekee oman asiansa, mutta vain yhdistettynä ne toimivat kunnolla. Tässä luvussa selitänkin pelaajalle selvimmin näkyvät osat näistä kyseisistä JavaScript-tiedostoista.

5.4.1 Interaktio

Isoin osa inspiraatiosta interaktiosysteemin toteuttamiseen löytyy *Sue Blackmanin* kirjasta *Beginning 3D Game Development with Unity* (2011). Interaktio-systeemini liikkeelle lähti hänen kirjassaan esittämistä esimerkeistä. Olen kuitenkin muokannut esimerkkejä toimimaan omassa pelissäni paremmin ja lisännyt niihin paljon uusia ominaisuuksia.

```
98  function OnMouseDown () {
99      //vain jos on oikealla etäisyydellä
100     if (DistanceFromCamera() > triggerDistance) return;
101     // jos esine on kirja
102     if(kirja == true) {
103         kirja_tunniste = true;
104     }
105     // hae tiedot
106     GetComponent(ObjectLookup).LookUpState(this.gameObject, currentState,
107     kursori.GetComponent(KursorinOhjaus).currentCursor.name);
108     if(gameObject.GetComponent("Item") == null)
109     {
110         ProcessObject(currentState);
111     }
112     else {
113     // jos esineen voi poimia ja se on käytettävissä
114     if (gameObject.GetComponent("Item").Kaytettava == true)
115     {
116         ProcessObject(currentState);
117     }
118     }
119 } // OnMouseDown
```

KUVA 19. OnMouseDown-funktio

interaktio.js:ssä määritellään mitä tapahtuu, jos pelaaja vie kursorin esineen päälle tai klikkaa sitä. Kuten kuva 19:ssä nähdään, Unityn skriptauksessa hiirenpainalluksen tunnistus tehdään OnMouseDown-funktiolla. Funktion { }-merkkien sisälle tulee kaikki mitä tapahtuu, kun kyseisen koodinpätkän sisältävää peliobjektia painetaan ruudulla. Peliobjektiin voi olla määritelty useampikin OnMouseDown-funktio erillisissä JavaScript-tiedostoissa.

Rivi *if (DistanceFromCamera() > triggerDistance) return;* tarkistaa onko esine pelaajan näkökulmasta liian kaukana. DistanceFromCamera()-funktiossa määritellään esineen sijainnin etäisyys kameran sijainnista reaaliajassa ja triggerDistance on muuttuja,

johon voi määritellä haluamansa maksimietäisyyden. Jos esine on liian kaukana, seuraavia rivejä ei suoriteta.

Seuraava if-lause tarkistaa onko klikattava esine määritelty kirjaksi, koska olen määritellyt kirjoille omia ominaisuuksia, joita muilla esineillä ei pelimaailmassa ole. Kirjat määritellään Unityn editorin puolella muuttamalla oletuksena olevan epätosi-arvon todeksi. Loppuosa funktiosta hakee peliobjektin tiedot ja tarkistetaan suoritetaanko sille ProcessObject-funktiota. ProcessObject-funktio suorittaa kaikki toiminnallisuudet, mitä tapahtuu esinettä klikkaamalla. Näitä ovat esimerkiksi mahdollinen ääni tai animaatio.

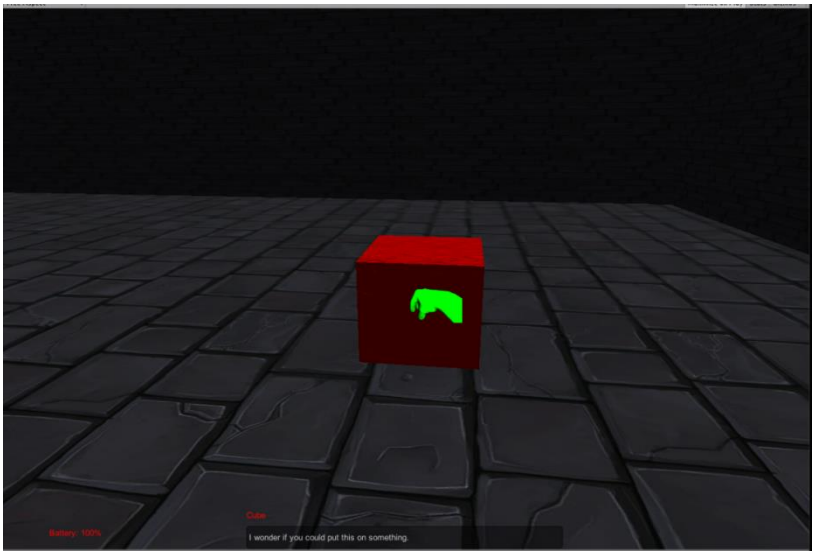
```
function OnMouseOver () {
    // vain, jos hiiren oikea nappi on pohjassa
    if(GameObject.Find("kursori").guiTexture.enabled == true)
    {
    // jos etäisyys ei ole liian iso
    if (DistanceFromCamera() > triggerDistance + moOffset) return;
    // vaihda kursoria, jos tarpeeksi lähellä
    if(DistanceFromCamera() <= triggerDistance) {
    if(gameObject.GetComponent("Item") == null)
    {
        GameObject.Find("kursori").GetComponent(KursorinOhjaus).SendMessage("SwapCursor",
        GameObject.Find("kursori").GetComponent(KursorinOhjaus).nostoCursor);
    }
    else {
    if(gameObject.GetComponent("Item").Kaytettava == true)
    {
        GameObject.Find("kursori").GetComponent(KursorinOhjaus).SendMessage("SwapCursor",
        +GameObject.Find("kursori").GetComponent(KursorinOhjaus).nostoCursor);
    }
    }
    onkoTavoitettavissa = true;
    controlCenter.GetComponent(GameManager).inRange =true;
    }
    // muuten resetoi kursori
    else {
        controlCenter.GetComponent(GameManager).inRange =false;
        GameObject.Find("kursori").GetComponent(KursorinOhjaus).SendMessage("ResetCursor");
    } //else
    controlCenter.GetComponent(GameManager).showText = true;
    // hakee controlCenteristä esineen nimen ja kuvauksen
    controlCenter.GetComponent(GameManager).shortDesc= currentObjectName ;
    controlCenter.GetComponent(GameManager).longDesc = currentObjectDescription;
    kursori.SendMessage("CursorColorChange", true); // kursorin värimuutos
    renderer.material = mouseOverMaterial; // vaihda esineen materiaalia
    } // if kursori
} // MouseOver
```

KUVA 20. OnMouseOver-funktio

Kuvassa 20 on OnMouseOver-funktio, joka on OnMouseDown:n tapaan mahdollista määrittää useassa eri tiedostossa yhden peliobjektin kohdalla. Tämän funktion sisällä oleva koodi vaikuttaa siihen, mitä tapahtuu hiiren kursorin mennessä kyseisen peliobjektin päälle. OnMouseDown ja OnMouseOver ovat siis UnityScriptin vakiofunktioita, joiden toiminnallisuus on määritelty syvemmällä Unityn pelimoottorissa.

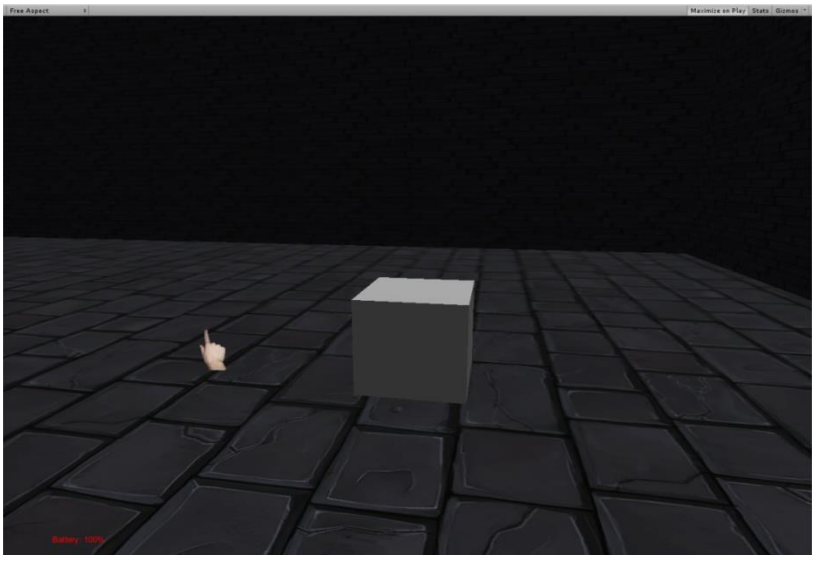
Tämä funktio oman pelini tapauksessa tarkistaa ensin onko hiiren oikea nappi pohjassa (näkyykö kursori) ja sen jälkeen tarkistaa miten kaukana pelaaja on kohteesta. Jos

kohde on tarpeeksi lähellä, muuttaa kursori kuvaansa ja väriänsä. Tämän lisäksi se muuttaa peliobjektin värin ja näyttää sen nimen ja kuvauksen, jos ne on määritelty (kuva 21).



KUVA 21. OnMouseOver toiminnassa

OnMouseOver-funktion vastapainona on myös OnMouseExit, joka ei oikeastaan tee muuta kuin nollaa OnMouseOverin tekemät asiat (kuva 22). Se siis muuttaa kursorin ja peliobjektin takaisin vakioksi, jos kursori poistetaan objektin kohdalta. Tämän lisäksi ruudussa näkyvät mahdolliset peliobjektiin liittyvät tekstit häviävät.



KUVA 22. OnMouseExit toiminnassa

Interaktio.js hoitaa siis pääosin vain kaiken näkyvän interaktion, kuten tekstien näyttämisen, äänien kuulumisen ja kursorin sekä 3D-mallien värin muuttamisen. Itse esi-

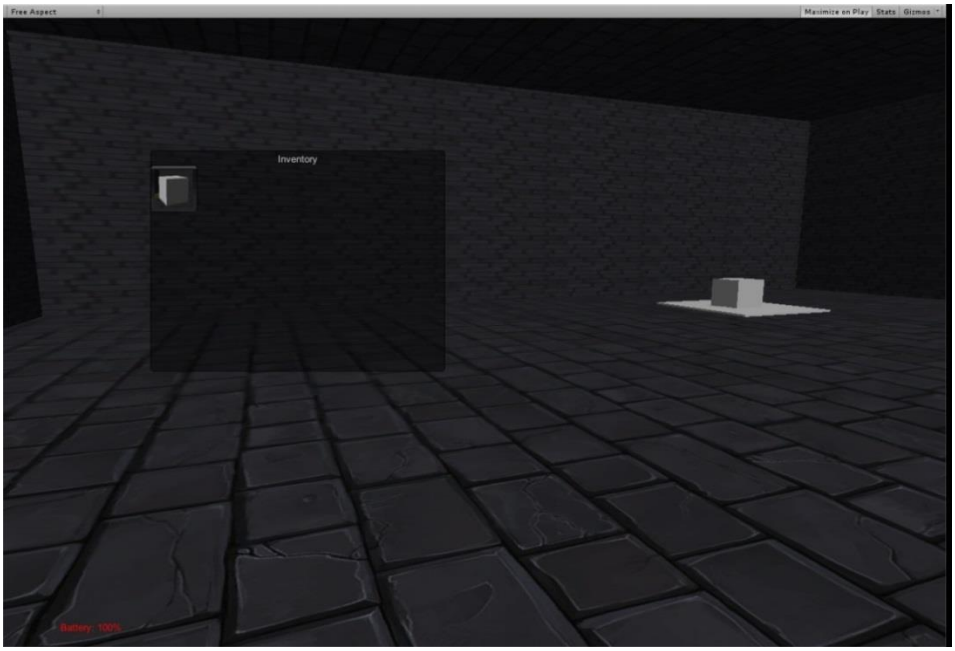
neiden poimimisen ja käyttämisen hoitavat kolme inventaariosysteemiin liittyvää JavaScript-tiedostoa, joista kerron seuraavassa luvussa (5.4.2). Täytyy kuitenkin muistaa, että ne kolme tiedostoa ovat myös yhteydessä Interaktio.js:n ja muuttavat sen muuttujien arvoja, kun esimerkiksi pelaaja poimii esineen.

5.4.2 Inventory

Rooli- tai seikkailupelejä pelaavalle *inventory* – suomeksi *inventaario* – on tuttu käsite, mutta muille se tulee mieleen lähinnä liiketaloudellisena terminä. Nämä molemmat tarkoittavat kuitenkin täsmälleen samaa asiaa eri konteksteissa. Videopeleissä inventaario on luettelo tavaroista, jotka pelihahmolla on käytössään. Nämä voivat vaihdella pelistä riippuen esimerkiksi parannusjuomista miekkoihin ja siitä vaikka kirjoihin tai kultarahoihin.

Koska en ollut koskaan aiemmin tehnyt tämmöistä järjestelmää, niin hain apua Unityn virallisilta foorumeilta. Tätä kautta löysin omiin tarpeisiini sopivan vapaassa jakelussa olevan inventaariojärjestelmän, jonka on tehnyt käyttäjänimi ExDeaDguY. Tämä vaati kuitenkin aika paljon muokkausta, että sain sen toimimaan omassa projektissani haluamallani tavalla, koska se oli alun perin suunniteltu ihan erilaiseen pelityyppiin erilaisella ohjausmekaniikalla.

Tämä järjestelmä koostuu kolmesta eri JavaScript-tiedostosta: yksittäisen esineen määrittävästä Item.js:stä, esineistä taulukon tekevään Inventory.js:stä ja järjestelmän graafisia puolia hoitavasta InventoryGUI.js:stä. Järjestelmässä näytölle tulee ruuduista muodostuva taulukko (kuva 23), joista jokainen ruutu vastaa yhtä esinettä. Jokaiselle esineelle voi erikseen määrittää myös oman ikonin, joka näkyy tässä taulukossa.



KUVA 23. Inventory-näkymä

Tämänkaltainen laatikko (GUI.Box) pelikuvan päälle piirretään UnityScriptiin sisältyvällä OnGUI-funktiolla. Sillä voidaan määrittää tekstiä, 2D-grafiikkaa, painettavia nappeja ja laatikoita varsinaisen peligrafiikan päälle kaksiulotteisena. Invenaarionäkymässä jokainen esine on määritetty GUI.Buttoniksi eli napiksi. Nappi on samanlainen toiminnaltaan kuin esimerkiksi JavaScriptin button. Toisin kuin JavaScriptissä, UnityScriptissä ei kuitenkaan tarvitse kirjoittaa erillistä onclick-funktiota klikkauksen tarkistukseen, vaan se hoidetaan if-lauseella napin piirtävän koodin ympärillä (kuva 24).

```
36      if(GUI.Button(Rect(currentX,currentY,itemIconSize.x,itemIconSize.y),item.inventoryIcon)){
37          esinekadessa = true;
38          kadessaolevaesine = i;
39          GameObject.Find("kursori").GetComponent(KursorinOhjaus).SendMessage("SwapCursor",item.inventoryIcon);
40      }
```

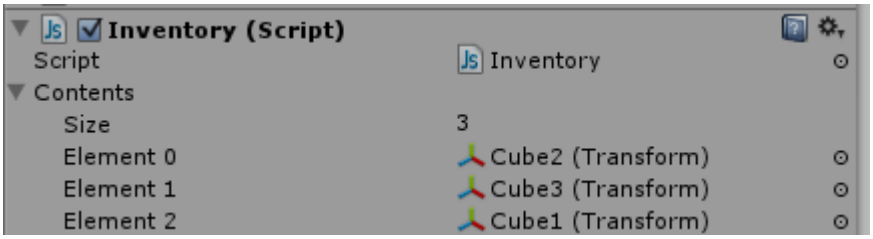
KUVA 24. Painalluksen tarkistus

Esineet ilmestyvät inventaarioon vasta, kun pelaaja on poiminut ne maasta. Mihin esineet siis häviävät poimimisen jälkeen? Esineet eivät oikeasti häviä, vaan ne muuttuvat ainoastaan näkymättömiksi ja ne siirtyvät pelaajan peliobjektin lapsiobjekteiksi. Näin ne siis kulkevat pelaajan koordinaattien mukana, vaikkei pelaaja niitä näekään kokoajan.

```
61  function Laske(){
62      transform.collider.isTrigger=false;// törmäyksentunnistus päälle
63      transform.renderer.enabled=true;// esine näkyy
64      rigidbody.useGravity = true; //painovoima
65      transform.parent=Vastakappale.transform;// siirrä vastakappaleen lapsiolioksi
66      transform.localPosition=Vector3(0, 0, 0.2);
67      Kaytettava = false;
68      }
```

KUVA 25. Esineen käyttöön liittyvä funktio

Esineistä pistetään poimittaessa renderer-, collider- ja useGravity-arvot epätosiksi. Rendererin poistaminen tarkoittaa sitä, ettei objektia piirretä pelikuvassa. Colliderin muuttaminen taas saa törmäyksentunnistuksen olemaan epäaktiivinen ja useGravity otetaan pois, ettei esineille turhaan lasketa painovoimaa, kun ne ovat näkymättömiä. Kun esineen käyttää tai laskee maahan, nämä kaikki arvot pistetään takaisin (Kuva 25). Kuvassa nähdään myös, että esine pistetään vastakappaleen lapsiolioksi ja arvo muuttuja Kaytettava saa arvon false eli epätosi. Nämä kaksi riviä ovat koodissa sen takia, koska kaikki pelin esineet ovat kertakäyttöisiä. Ne rivit takaavat, ettei pelaaja pysty nostamaan käytettyjä esineitä maasta uudestaan.



KUVA 26. Inventaarion sisältö

Inventory.js sisältää Contents-nimisen taulukon (kuva 26), jossa on kaikki pelaajan poimimat esineet Transformina. Transform sisältää tiedot kyseisen peliobjektin sijainnista, kierteestä ja koosta. Taulukkoon lisätään aina uusi rivi, kun esine poimitaan ja poistetaan, kun esine tiputetaan.

5.5 Markkinointi ja levitys

Mitä mahdollisuuksia kaltaisellani Indie-pelinkehittäjällä on tuotoksensa levittämiseen ilman ulkoista rahoitusta? Jos puhutaan PC-markkinoista, niin on olemassa pari uutta trendiä: *crowdfunding* eli joukkorahoittaminen ja keskeneräisen version julkaiseminen rahaa vastaan. Molemmat näistä tavoista vaativat toimiakseen kuitenkin paljon näkyvyyttä pelimediassa.

Joukkorahoittamisella tarkoitetaan rahan keräystä pelaajilta ennen kuin peliprojekti on edes valmis. Pelaajat ovat siis itse sijoittajia ja ottavat riskin pelistä peliä kehittävän tahon lupauksien pohjalta. Yhdysvaltalainen *Kickstarter*-joukkoistamispalvelu tuli kuuluisaksi pelimedioissa, kun Double Fine Adventure-niminen peliprojekti keräsi yli 3,3 miljoonan dollarin rahoituksen käyttäjiltä, vaikka he hakivatkin alun perin vain 400 000 dollarin rahoitusta. (Double Fine Adventure 2012.)

Vähän erilaisempi esimerkki samankaltaisesta ideasta on *crowdsourcing* eli joukkoistaminen. Tätä käyttää erimerkiksi *Steam Greenlight*, jossa pelaajilta ei kerätäkään rahaa vaan ääniä. Pelaajat saavat äänestää mitä pelejä haluaisivat nähdä Steam-latauspalvelusta ostettavana. Nämä pelit voivat olla jo valmiita tai sitten vasta puhtaita peli-ideoita, eikä niiden tarvitse välttämättä olla edes pelejä, vaan esimerkiksi myös pelimodit käyvät. Toisin kuin isomman peliyrityksen, pienen kehittäjän voi olla vaikea saada pelinsä myyntiin Steamiin. Tämä onkin yksi keino, jolla Steam saa palautetta käyttäjiltään, minkälaista sisältöä palvelussa kuuluisi olla. Pelin ilmoittaminen Greenlightiin vaatii 100 dollarin maksun, joka menee kokonaan hyväntekeväisyyteen. Maksulla yhtiön mukaan karsitaan kaikki vitsillä tehdyt peli-ideat. Maksun tarvitsee suorittaa vain kerran käyttäjätiliä kohden, jonka jälkeen käyttäjätilillä voi lähettää niin monta peliä kuin tahtoo. (Steam Greenlight 2013.)

Pelin varhaisen version julkaiseminen rahaa vastaan on trendi, joka tuli suosioon ruotsalaisen Markus Perssonin tekemän *Minecraft*-pelin myötä. Persson julkaisi vuonna 2009 pelin varhaisen version ja pyysi siitä rahaa, jotta voisi lopettaa päivätyönsä ja jatkaa pelinsä kehittämistä työkseen. Pelaaja siis maksoi keskeneräisestä pelistä, mutta oli näin ollen oikeutettu valmiiseen peliin ilman lisäkuluja heti kun se olisi valmis. Peli valmistuikin pari vuotta myöhemmin ja on tuottajatiimilleen tekijälleen monen kymmenen miljoonan omaisuuden. (Rose 2011.) Nyt vuonna 2013 samankaltainen ilmiö on havaittavissa Steam-latauspalvelussa, jossa on myynnissä ja pelattavissa jopa toistakymmentä vielä kehityksessä olevaa peliä.

Ilman isoa markkinointibudjettia, lähestulkoon ainoa tapa saada julkisuutta pelilleen on ottamalla yhteyttä pelimediaan. Lähettämällä arvostelukopioita pelistä tarpeeksi moneen eri mediaan, joku saattaa tarttua haasteeseen ja kirjoittaa siitä arvostelun tai jutun. Tämä kuitenkin vaatii valmista tai edes lähes valmista peliä. Toinen vähän kal-

liimpi tapa on erilaiset pelimessut kuten Seattlessa vuosittain järjestettävä *Penny Arcade Expo*. Tämänkaltaisissa messuissa voi vuokrata pöydän ja esitellä peliään suoraan kuluttajille ja pelitoimittajille.

AppStoren, Google Playn ja Facebookin kaltaiset lähes vapaat levitysympäristöt ovat jo niin laajoja, että sieltä on lähes mahdotonta erottua muiden joukosta ilman erillistä markkinointia. Pelin täytyy joko olla niin hyvä, että se saa julkisuutta pelaajien kertoessa kavereilleen teoksesta tai sitten täytyy vain tuurilla selviytyä voittoon 10:n melkein samanlaisen pelin keskeltä. On myös huomioitava, että vastassa mobiilipelialustoillakin ovat isot markkinointikoneistot, jotka haluavat oman osansa tästä bisnesmallista.

Oman pelini tapauksessa paras levitystapa olisi luultavasti Windows- tai Linux-peli. Mobiili- ja konsolipelit eivät tulisi kysymykseen, koska Unity myy erikseen työkalut mobiililaitteille ja konsoleille. Mobiilialustat vaativat jokainen oman 400 dollarin pakettinsa Unityn verkkokaupasta ja konsoliversioita voi tehdä vain 1500 dollarin arvoisella Unityn Pro-versiolla. (Unity Store 2013.) Lisäksi konsolien tapauksessa pitäisi olla konsolista kehittäjäversio, jolla on mahdollista ajaa omia sovelluksiaan. Tämän lisäksi konsolille julkaistavat pelit vaativat kyseiselle alustalle ominaisen lisenssisopimuksen, joka pitää sopia alustan valmistajan kanssa. (Unity For Consoles 2013.) Unityn ilmaisella versiolla on mahdollista julkaista pelejä kaupallisesti ilman mitään lisensointimaksuja. Ainoastaan jos yrityksen vuosittaiset tulot ylittävät 100 000 dollaria vuodessa, on ilmaisversion käyttö kielletty. (Unity3D.com: Frequently Asked Questions 2013.)

Mac-versio olisi mahdollinen myös, jos minulla olisi käytettävässäni Mac-tietokone. Mac-version kääntäminen Unity-pelistä vaatii nimittäin Mac-käyttöjärjestelmän, jossa on Xcode-ohjelmisto asennettuna. Myös iPhone-versio pitää kääntää tätä kautta, vaikka se vaatiikin erillisen maksullisen pakettinsa.

6 PÄÄTÄNTÖ

Olen jo pitkän aikaa ollut kiinnostunut pelien kehittämisestä. Kuitenkin vasta Unityn kaltaiset kehitysympäristöt antoivat minulle sen mitä halusin: mahdollisuuden kehittää näyttävää peliä ilman hirveitä määriä koodia. Oma kiinnostukseni kohdistuu kuitenkin

pelimekaniikan hiomiseen, eikä pelin ruudunpäivityksen optimoimiseen tietyn näytönohjaimen tietyillä ajureilla ja Unity on täydellinen työkalu tämänkaltaisessa tapauksessa.

Opinnäytetyöni lähtökohtana oli peliprojektin loppuun vieminen. Onko peli tosiaan valmis? Tähän asti aikaan saamani teos sisältää 2 tasoa, joista toinen on enemmän tai vähemmän vain pelimekaniikan opetusta. Pelissä on kuitenkin selvä alku ja selvä loppu. En tätä vielä tällaisenaan myyntiin pistäisi, mutta nyt minulla on ainakin selvä pohja, jota voin laajentaa suuntaan tai toiseen.

Entä kuinka paljon Unity todella helpotti pelinkehitystä? En ole koskaan päässyt sisälle C:n ja C++:n kaltaisiin ohjelmointikieliin. Sen takia en ole tutustunut pelien tekemiseen juurikaan. XNA:n opittuani yksinkertaisella tasolla ymmärsin, ettei peliohjelmointi enää nykyään vaadi samanlaisia tietoja ja taitoja kuin aiemmin. C# on ohjelmointikielenä paljon suoraviivaisempi kuin muut C-kielet. Tämän takia innostuin Unitystä, koska se tukee JavaScriptin lisäksi myös C#:a. Päädyin kuitenkin tekemään oman työni JavaScriptillä, koska iso osa Unity-oppaista, joihin törmäsin, suosii sitä.

Tulenko koskaan tekemään pelistäni niin hyvää, että se kelpaisi mielestäni maksulliseen levitykseen? En luultavasti. Pelin kohtalo on kuitenkin vielä avoin. Iso osa peliä varten tehdystä koodista ja sisällöstä on kuitenkin suunniteltu siinä mielessä, että niitä voisi käyttää uudestaan muissakin projekteissa.

Yksi tärkeimmistä asioista mitä tästä projektista opin, oli tietenkin näkemys peliprojektin laajuudesta. Näin laaja projekti yhden ihmisen työnä on hyvin työlästä. Yksi ihminen pystyisi ehkä tekemään yksinkertaisemman pelin, mutta tarinavetoinen ensimmäisen persoonan peli on mahdotonta ilman monen vuoden sitoutumista projektiin varsinkin, kun sama henkilö tekee 3D-mallit, grafiikat ja ohjelmoinnin.

Koska projektilla oli niin selvä deadline, keskityin heti alusta lähtien yksityiskohtiin kuten 3D-mallien tekstuurit. Tämä on virhe peliä kehittäessä. Pelistä pitäisi saada ensiksi ulos prototyyppi, jota voi testata pelaajilla ja kehittää siitä oikeaan suuntaan. Yksityiskohdat ovat kuitenkin usein vain ulkoasuun liittyviä ja karkeammallakin toteutuksella saisi hiottua pelimekaniikan toimivaksi.

Riittävä pelaajatestaus on ainoa keino saada tasapainotettua tämänkaltainen peli sopivan haastavaksi. Peliä suunnitellessa tekijä usein sokaistuu omille ideoilleen. Kun on tehnyt töitä niiden kanssa kymmeniä tunteja, ei ymmärrä että joku ulkopuolinen ei välttämättä näe asioita yhtä selkeästi. Tämän takia muutamia pelin pulmia ja pelimekaniikkaa pitikin miettiä uusiksi sen jälkeen, kun näytin peliä muille ihmisille. Jos olisin tehnyt näin jo projektin alusta, niin korjaaminen olisi ollut huomattavasti helpompaa.

Peliprojektissa tärkeintä ovat siis tarkat ja yksityiskohtaiset suunnitelmat. Mitä enemmän käyttää aikaa suunnitteluun, sitä vähemmän tarvitsee tuotosta korjata kesken kehityksen. Tiesin tämän jo etukäteen, mutta en siltikään tehnyt tarpeeksi pohjatyötä ennen varsinaisen pelin tekemistä. Ehkä tämä on asia, joka pitää kokea kantapään kautta.

LÄHTEET

Bedárd, Renaud 2009. Behind Fez: Trixels (part two). Theinstructionlimit.com. WWW-dokumentti. http://theinstructionlimit.com/behind-fez-trixels-part-two. Päivitetty: 3.5.2009 Luettu: 29.11.2012.

Blackman, Sue 2011. Beginning 3D Game Development with Unity. New York: Appress.

Dansky, Richard 2012. A Game Writer's Perspective on Game Writing. Jordanmechner.com. WWW-dokumentti. http://jordanmechner.com/blog/2012/01/game-writing-2/. Päivitetty: 25.1.2012. Luettu: 25.12.2012.

Double Fine Adventure. 2012. Kickstarter.com. WWW-dokumentti. http://www.kickstarter.com/projects/doublefine/double-fine-adventure. Päivitetty: 14.3.2012. Luettu: 25.4.2013.

Dutton, Fred 2012. What is Indie? Eurogamer.net. WWW-dokumentti. http://www.eurogamer.net/articles/2012-04-16-what-is-indie. Päivitetty: 18.4.2012. Luettu: 29.11.2012.

Edwards, Benj 2007. Ten Things Everyone Should Know About Space Invaders. 1up.com. WWW-dokumentti. http://www.1up.com/features/ten-space-invaders?pager.offset=1. Luettu: 20.2.2013.

Excecution Order. 2012. Unity3D.com. WWW-dokumentti. http://docs.unity3d.com/Documentation/Manual/ExecutionOrder.html. Päivitetty: 10.10.2012. Luettu: 20.2.2013.

Fez. 2012. Polytron Corporation. WWW-dokumentti. http://polytroncorporation.com/61-2. Luettu: 5.12.2012.

Fulletron, Tracy 2008. Game Design Workshop: A Playcentric Approach to Creating Innovative Games Second Edition. Burlington: Elsevier.

Fun Facts. 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/company/public-relations/. Luettu: 10.4.2013.

Gamedev Glossary: What Is the "Game Loop"?. 2012. Tutplus.com. WWW-dokumentti. http://gamedev.tutsplus.com/articles/glossary/quick-tip-what-is-the-game-loop/. Päivitetty: 30.11.2012. Luettu: 20.2.2013.

GameObjects. 2010. Unity3D.com. WWW-dokumentti. http://docs.unity3d.com/Documentation/Manual/GameObjects.html. Päivitetty: 14.9.2010. Luettu: 2.2.2013.

Half-Life 2. Verkkokauppa.com. WWW-dokumentti. http://www.verkkokauppa.com/productimages/orig/51584_02.jpg Luettu: 27.11.2012.

License Comparisons. 2013. Unity3d.com. WWW-dokumentti. http://unity3d.com/unity/licenses. Luettu: 10.4.2013.

Luban, Pascal 2002. Designing and Integrating Puzzles in Action-Adventure Games. Gamasutra.com. WWW-dokumentti. http://www.gamasutra.com/view/feature/2917/designing_and_integrating_puzzles_.php. Päivitetty: 6.12.2002. Luettu: 3.2.2013.

Nutt, Christian 2013. Nintendo's indie guy tells you how to get your games approved. Gamasutra.com. WWW-dokumentti. http://www.gamasutra.com/view/news/189180/Nintendos_indies_guy_tells_you_how_to_get_your_games_approved.php. Päivitetty: 25.3.2013. Luettu: 10.4.2013.

Orland, Kyle 2011. Turbine: Lord of the Rings Online Revenues Tripled As Free-To-Play Game. Gamasutra.com. WWW-dokumentti. http://www.gamasutra.com/view/news/32322/Turbine_Lord_of_the_Rings_Online_Revenues_Tripled_As_FreeToPlay_Game.php. Päivitetty: 6.1.2011. Luettu: 2.3.2013.

Owen, Phil 2013. What In The World Do Video Game Writers Do? The Minds Behind Some Of Last Year's Biggest Games Explain. Kotaku.com. WWW-dokumentti. http://kotaku.com/5988751/what-in-the-world-do-video-game-writers-do-the-minds-behind-some-of-last-years-biggest-games-explain. Päivitetty: 5.3.2013. Luettu: 14.5.2013.

Physics. 2013. Unity3D.com. WWW-dokumentti. http://unity3d.com/unity/quality/physics. Luettu: 20.2.2013.

Platforms. 2013. Unrealengine.com. WWW-dokumentti. http://www.unrealengine.com/en/platforms/ Luettu: 4.5.2013.

Rochard. Rochardthegame.com. WWW-dokumentti. http://www.rochardthegame.com/wp-content/uploads/2011/10/R4.jpg Luettu: 5.12.2012.

Rochard’s Producer on the Unity Engine. 2011. Rochardthegame.com. WWW-dokumentti. http://www.rochardthegame.com/en/2011/10/21/rochards-producer-on-the-unity-engine/. Päivitetty: 21.10.2011. Luettu: 5.12.2012.

Rose, Mike 2011. Minecraft 'Full Version' Will Release On November 11. Gamasutra.com. WWW-dokumentti. http://www.gamasutra.com/view/news/34003/Minecraft_Full_Version_Will_Release_On_November_11.php. Päivitetty: 8.4.2011. Luettu: 27.3.2013.

Ryes, Eli 2010. Warner Bros to Acquire Big-Screen Rights to SPACE INVADERS Game. WWW-dokumentti. http://geektyrant.com/news/2010/3/2/warner-bros-to-acquire-big-screen-rights-to-space-invaders-g.html. Päivitetty; 2010. Luettu: 5.3.2013.

Store. 2013. Unity3D.com. WWW-dokumentti. https://store.unity3d.com/. Luettu: 16.3.2013.

Stuart, Keith 2012. Interview: Schafer’s Millions. Hookshotinc.com. WWW-dokumentti. http://www.hookshotinc.com/interview-schafers-millions/. Päivitetty: 11.2.2012. Luettu: 23.4.2013.

The Secret of Monkey Island. Worldofmi.com. WWW-dokumentti. http://www.worldofmi.com/thegames/monkey1/16color.gif Luettu: 2.5.2013.

Thomsen, Mike 2009. Ode to Source: A History of Valve's Tireless Game Engine. IGN.com. WWW-dokumentti. http://www.ign.com/articles/2009/09/22/ode-to-source-a-history-of-valves-tireless-game-engine?page=2. Päivitetty: 22.7.2009. Luettu: 25.12.2012.

Thomsen, Mike 2010. History of the Unreal Engine. IGN.com. WWW-dokumentti. http://www.ign.com/articles/2010/02/23/history-of-the-unreal-engine. Päivitetty: 23.2.2010. Luettu: 5.12.2012.

Unity for console games. 2013. Unity3D.com. WWW-dokumentti. http://unity3d.com/unity/multiplatform/consoles. /. Luettu: 16.3.2013.

Unity3D: Frequently Asked Questions. 2013. Unity3D.com. WWW-dokumentti. http://unity3d.com/unity/faq. Luettu: 10.5.2013.

Unreal #9. 2012 Gamechasm.com. WWW-dokumentti. http://www.gameschasm.com/unreal-9.html. Luettu: 5.12.2012.

Webster, Andrew 2013. PlayStation and indie games: a love story. TheVerge.com. http://www.theverge.com/2013/3/29/4162430/sony-playstation-indie-games-love-story. Päivitetty: 29.3.2011. Luettu: 24.3.2013.

What is Steam Greenlight?. 2013. Steamcommunity.com. WWW-dokumentti. http://steamcommunity.com/workshop/about/?appid=765&section=faq. Luettu: 5.5.2013.

What is the XNA Framework. 2006. MSDN Blogs. WWW-dokumentti. http://blogs.msdn.com/b/xna/archive/2006/08/25/724607.aspx. Päivitetty: 25.8.2006. Luettu: 26.11.2012.