Petri Liuska

SISÄLLÖNTUOTANNON TYÖKALUJEN TOTEUTTAMINEN KAUPALLISESSA PELIPROJEKTISSA

Opinnäytetyö
Kajaanin ammattikorkeakoulu
Luonnontieteiden ala
Tietojenkäsittely
Syksy 2014

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ
TIIVISTELMÄ

Koulutusala Luonnontieteiden ala	Koulutusohjelma Tietojenkäsittely
Tekijä(t) Petri Liuska	
Työn nimi Sisällöntuotannon työkalujen toteuttaminen kaupallisessa peliprojektissa	
Vaihtoehtoiset ammattiopinnot Pelimoottoriohjelmointi	Toimeksiantaja KAMK Kajability OSK / wearebind
Aika Syksy 2014	Sivumäärä ja liitteet 26

Nykypeleissä on valtava määrä sisältöä ja sisällöntuotanto onkin yksi pelinkehityksen pääosa-alueista. Merkittävä osa kaupallisen pelin budjetista kuluu sisällön tuottamiseen, etenkin koska laatuvaatimukset ovat korkeat ja kasvavat kaiken aikaa teknologian kehityksen mukana. Sisällöntuotannosta on vastuussa monia eri rooleja ja roolien kirjo vaihtelee projektista toiseen, mutta yhteistä heille on se, että kaikki heistä käyttävät työkaluja. Työkalujen tekemisestä vastaavat siihen erikoistuneet ohjelmoijat. Sisältö, jonka luomiseen tarvitaan omia työkaluja, on yleensä pelikohtaista. Omien työkalujen kanssa on tärkeintä, että ne ovat luotettavia ja niillä saa työn tehtyä.

Opinnäytetyön tavoitteena oli luoda pelisuunnittelijalle työkalut, joilla on mahdollista toteuttaa kaikki Pump-Action Popcorn -pelin power upit ja aseet. Vaatimuksena oli, että ne tarjoaisivat mahdollisimman paljon vapautta, ja niiden käyttäminen piti liittää saumattomasti osaksi Unityn käyttöliittymää. Pelisuunnitteludokumenteissa oli olemassa sisällöstä alustavia suunnitelmia, joiden pohjalta määritettiin, minkälaista sisältöä työkaluilla pitäisi pystyä tekemään. Työkalujen suunnittelussa otettiin huomioon helppokäyttöisyys ja toteutuksen helppo laajennettavuus.

Pelisuunnittelija oppi käyttämään työkaluja nopeasti, ja käyttöönottovaihe oli lyhyt ja helppo. Ne mahdollistivat itsenäisen työskentelyn ja suunnittelija pystyi toteuttamaan visionsa mukaista sisältöä. Työkalut olivat kaikki osana Unityn käyttöliittymää, ja suunnittelijan ei koskaan tarvinnut avata erillistä editoria. Työkalut eivät rajoittaneet suunnittelijan luovuutta, ja työkaluille asetetut vaatimukset täyttyivät. Muutamia kertoja suunnittelija tarvitsi työkaluihin uuden ominaisuuden tai löysi ongelman, mutta tehdyt muutokset olivat varsin pieniä.

On helppo aliarvioida omien työkalujen tekemisen tärkeys ja se, miten paljon niillä voidaan säästää aikaa. Monissa projekteissa työkalujen toteuttamiselle varattu aika on liian lyhyt. Jo lyhyessä ja keskeneräiseksi jääneessä projektissakin huomasi, että työkaluihin keskittyminen kannattaa. Se, että suunnittelija pystyy itse tekemään tehokkaasti sisältöä, vähentää huomattavasti ohjelmoijan työkuormaa ja antaa hänen keskittyä tärkeämpiin asioihin. Suunnittelijan visio ei vääristy välikäden vuoksi, vaan sisällöstä tulee juuri sellaista kuin hän itse haluaa.

Kieli	Suomi
Asiasanat	Sisällöntuotanto, peliohjelmointi, työkaluohjelmointi
Säilytyspaikka	☒ Verkkokirjasto Theseus ☐ Kajaanin ammattikorkeakoulun kirjasto

KAJAANIN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

THESIS
ABSTRACT

School Natural Sciences	Degree Programme Business Information Technology
Author(s) Petri Liuska	
Title Making Content Creation Tools for a Commercial Game	
Optional Professional Studies Game Engine Programming	Commissioned by KAMK Kajability OSK / wearebind
Date Autumn 2014	Total Number of Pages and Appendices 26

There is a huge amount of content in a modern game and that's why content creation requires so much effort. It takes a big portion of the budget of a modern game, especially because the quality requirements are high and growing all the time as the technology advances. There are many roles involved and what they all have in common is that everyone uses tools. Tools programmers are the ones responsible of making them. Content that requires custom tools is usually game-specific. The most important thing with custom tools is that they are reliable and you can get the work done with them.

The goal for this thesis was to make a set of tools for a game designer, which he could use to create all the power ups and weapons in Pump Action Popcorn. It was required that the tools provide as much freedom as possible and they are seamlessly integrated in Unity Editor. There were some rough designs of the game content in the game design document which were utilized in specifying what kind of content one should be able to produce with the finished tools. Ease of use and extensibility were the main principles when the tools were designed.

The designer had no trouble in using the tools and the commissioning was very effortless. The tools made it possible for the designer to create content without help according to his vision. All the tools were included in the user interface of Unity. All the requirements for the tools got fulfilled. There were times when the designer needed a new feature or found a problem with the tools, but all the changes made were minor.

It is easy to underestimate the value of custom tools and the time they save in the long run. In many projects the amount of time used for developing the tools is not enough. Even in a project short and unfinished, it was easy to see that it pays back when you focus on the tools. The fact that the designer can effectively produce content without help greatly decreases the programmer's involvement in content creation, so he can focus on more important things. Also, without middlemen, all the content made by the designer is just as he wants it.

Language of Thesis	Finnish
Keywords	Content creation, game programming, tools programming
Deposited at	☒ Electronic library Theseus ☐ Library of Kajaani University of Applied Sciences

SISÄLLYS

SYMBOLILUETTELO

Instanssi	Olio-ohjelmoinnissa luokan edustaja. Luokka määrittelee muuttujat ja metodit, instanssi antaa muuttujille arvot ja mahdollistaa metodien kutsumisen.
Komponentti	Pelimaailman objektiin kiinnitetty yksittäinen käyttäytymistä säätelevä osa. Unity-pelimoottorissa nämä voivat olla esimerkiksi scriptejä.
Power Up	Pelihahmon kykyjä tehostava väliaikainen vaikutus.
Prefab	Pelimaailman objektista yksittäisten muuttujien tarkkuudella tallennettu muotti, josta voidaan luoda instansseja pelimaailmaan.
Scene	Yksittäinen tapahtumapaikka pelissä, joka sisältää pelimaailman objekteja.
Screen shake	Näytön kuvan tärisyttäminen, jota käytetään tehokeinona peleissä.
Scripti	Unity-pelimoottorissa ajettavaa ohjelmakoodia sisältävä tiedosto.
Sprite	Kaksiulotteinen kuva, jota käytetään graafisena elementtinä pelissä.

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on toteuttaa työkalut, joilla pelisuunnittelija voi tuottaa itsenäisesti sisältöä kaupalliseen peliprojektiin. Työkalujen täytyy tarjota tarpeeksi monipuolisuutta, jotta niillä voidaan toteuttaa pelin kaikki aseet ja power upit. Niiden pitää myös sulautua saumattomaksi osaksi Unity-pelimoottorin käyttöliittymää. Tavoitteiden saavuttamiseksi määritellään, mitä työkaluilla täytyy pystyä tekemään, suunnitellaan tekniset ratkaisut ja toteutetaan lopulta työkalut C#-ohjelmointikielellä.

Teoriaosuudessa esitellään sisällöntuotantoa, erilaisia siihen kuuluvia rooleja ja käytettäviä työkaluja. Valmiiden työkalujen lisäksi tuodaan esille tilanteita, joissa omien työkalujen tekeminen on tarpeen. Erityisesti huomiota annetaan käytännöille, joita omien työkalujen tekemisessä on hyvä soveltaa. Opinnäytetyö käsittelee aihetta siinä määrin syvällisesti, että siitä saa parhaiten kaiken irti, jos omaa perustiedot pelinkehityksestä, Unity-pelimoottorin käyttämisestä ja ohjelmoinnista.

Opinnäytetyön tilaaja, Bind, on seitsemän hengen pelinkehitystiimi, joka toimii Kajaanissa, Kajaanin ammattikorkeakoulun tiloissa. Liiketoiminta tapahtuu KAMK Kajability OSK:n alla aputoiminimellä. Jäsenet tekevät opiskelujensa ohella kaupallisia pelejä ja alihankintatöitä. Lisäksi suurena apuna ja tukena on myös toinen Kajaanin ammattikorkeakoulun tiloissa toimiva osuuskunta, Kajak Games.

2 SISÄLLÖNTUOTANTO

Nykyaikaisessa, kaupallisessa pelissä on valtava määrä sisältöä. Pelisisältö kattaa kaikki pelimaailman ympäristöt, alueet, hahmot, esineet ja asiat. Sisällöntuotanto onkin yksi nykyaikaisen pelinkehityksen pääosa-alueista, ja suuri osa kaupallisen pelin budjetista kuluu siihen. Ainoa tärkeä asia ei ole sisällön määrä, vaan myös laatuvaatimukset ovat korkeat ja kasvavat kaiken aikaa teknologian kehityksen mukana. Vuosi vuodelta on mahdollista lisätä yhä enemmän yksityiskohtia ja yleisön vaatimuksesta siinä mennään äärirajoilla. (Togelius, De Nardi & Lucas 2007, 2; De Carli, Bevilacqua, Pozzer & d'Ornellas 2011, 1; Labschütz, Krösl, Aquino, Grashäftl & Kohl 2011, 1)

Teknisestä näkökulmasta tarkasteltuna sisältö tarkoittaa 3D-malleja, animaatioita, kuvatiedostoja, äänitiedostoja, pelikenttiä jne. Jokaista sisältötyyppiä kohden on yleensä oma erikoistumista vaativa työkalunsa. Sisällön luominen ei siis rajoitu pelkästään yhteen taiteenalaan, vaan sen tuottamiseen tarvitaan monen eri alan asiantuntijoita. (Hastings, Guha & Stanley 2009, 1; Gregory 2009, 49, 5-6.)

2.1 Sisällöntuotannon roolit

Sisällöntuotannosta vastuussa olevien roolien työpanosten merkitys on suuri, koska sisältö käytännössä määrittelee pelin (Gregory 2009, 5). Edellä listattuna muutamia sisällöntuotannon rooleista:

- *Konseptitaiteilijat* maalaavat kuvia, joiden tarkoituksena on kommunikoida muulle tiimille, miltä lopullinen peli voisi näyttää. Työ aloitetaan jo varhaisessa vaiheessa, mutta se jatkuu yleensä läpi koko kehityksen, ohjaten pelin visuaalisuuden muodostumista. Lopputulos näyttää yleensä ainakin jollain tasolla samalta kuin konseptitaide. (Gregory 2009, 6.)

- *3D-mallintajat* rakentavat kolmiulotteisen, yleensä kolmioista koostuvan, geometrian kaikille pelimaailman esineille, asioille ja ympäristöille. Suurin haaste on tehdä mahdollisimman hyvää jälkeä, mutta mahdollisimman vähillä kolmioilla. Tyypillisesti mallintaja erikoistuu joko etualan tai taustojen mallintamiseen. Etualalla viitataan peli-

maailman dynaamisiin tai interaktiivisiin elementteihin, kun taas taustoilla viitataan staattisiin ympäristöihin. (Gregory 2009, 6; Pickthall 2012.)

- *2D-graafikot* maalaavat kaksiulotteisia kuvia, joita käytetään pintakuvioina 3D-mallintajan rakentaman geometrian päällä. Pintakuviointi lisää malliin yksityiskohtia. Peleihin, joissa ei käytetä 3D-grafiikkaa, 2D-graafikot piirtävät kaksiulotteisia spritejä. (Gregory 2009, 6; Pickthall 2012.)

- *Valotaiteilijat* valaisevat pelimaailman sijoittelemalla sinne valonlähteitä. Työskentelemällä valaistuksen värin, voimakkuuden ja suunnan kanssa he vaikuttavat eri kohtausten taiteellisuuteen ja niiden välittämään tunteelliseen lataukseen. (Gregory 2009, 6.)

- *Animaattorit* herättävät pelimaailman eloon laittamalla hahmot ja esineet liikkumaan. He toimivat välillisesti pelimaailman näyttelijöinä samaan tapaan kuin animaatioelokuvissa. Lisähaasteena on otettava huomioon pelimoottorin reaaliaikaisuudesta johtuvat tekniset rajoitukset. (Gregory 2009, 6; Stankowicz 2013.)

- *Äänisuunnittelijat*, *ääninäyttelijät* ja *muusikot* ovat vastuussa pelin äänimaailmasta, mukaan lukien kaikki yksittäiset ääniefektit, pelihahmojen puheäänet ja pelin musiikki. Äänet jäävät usein vähemmälle huomiolle kuin pelin visuaalisuus. (Gregory 2009, 6, 44.)

- *Pelisuunnittelijat* työskentelevät eri yksityiskohtatasoilla ja suunnittelevat pelikokemuksen interaktiivisen osuuden. Jotkut keskittyvät suurempaan kokonaiskuvaan eli tarinaan, sen sanelemaan kenttien järjestykseen sekä pelaajan pitkän tähtäimen tavoitteisiin. Toisten vastuulla ovat yksityiskohdat, yksittäisten kenttien rakenne ja muodot, vihollisten ja muiden elementtien sijoittelu ynnä muu. (Gregory 2009, 7.)

- *Käsikirjoittajat* ovat joissain peleissä mukana rakentamassa tarinaa ja kirjoittamassa dialogia. Työssä on usein apuna pelin tarinan ja isomman kokonaisuuden parissa työskentelevä suunnittelija. (Gregory 2009, 7.)

Roolien kirjo vaihtelee tiimistä toiseen, riippuen projektin tarpeista ja laajuudesta sekä tiimin koosta. Pienemmässä tiimissä, joka työskentelee esimerkiksi mobiilipelin parissa, yksi henkilö saa usein hoidettavakseen useita eri rooleja. Konsolipeliä kehittävässä, moninkertaisesti suu-

remmassa tiimissä yhdellä työntekijällä saattaa usein olla vain yksi tarkkaan määritelty rooli. (Chandler & Chandler 2011, 41.)

2.2 Sisällöntuotannon työkalut

Kaikki pelinkehitysprosessiin osallistuvat käyttävät ainakin jonkinlaisia työkaluja, ja osa tässä osiossa käsiteltävistä asioista onkin sovellettavissa myös sisällöntuotannossa käytettävien työkalujen ulkopuolelle. Työkalun tarkoitus on aina helpottaa työtä tai parantaa sen jälkeä. Kummatkin johtavat työhön käytettävän ajan vähenemiseen. Säästetty aika voidaan käyttää laadunvalvontaan, joka parantaa tuotteen kokonaislaatua, tai tuotteen parissa työskentely voidaan yksinkertaisesti vain lopettaa aikaisemmin. (Moar 2004.)

Tehokkaimmat työkalut muuttuvat luonnolliseksi osaksi työntekoa häiritsemättä sitä. Käyttäjät voivat keskittyä tekemiseen, unohtaen täysin edes käyttävänsä työkalua. Tästä syystä, niin kauan kun työskentelyssä ei esiinny ongelmia, käyttäjiä ei usein oikeastaan edes kiinnosta minkälaisia työkaluja projektissa käytetään. (Moar 2004.)

Käytettävä työkalu riippuu luotavan sisällön tyypistä. Esimerkiksi hahmot, rakennukset ja ympäristö ovat usein 3D-malleja ja tekstuureja. Tällainen sisältö on niin yleistä ja lähes kaikille peleille yhteistä, että siihen on tarjolla hyviä valmiita työkaluja, eikä sellaisia tarvitse tehdä itse. (Gregory 2009, 6, 49-53; Hall 2007, 565.)

Esimerkkejä valmiista työkaluista:

- Unreal Engine 4 on pelimoottori, jolla voidaan kehittää pelejä konsoleille, PC:lle, Macille, iOS:lle ja Androidille (Develop 2014, 7).

- Maya on 3D-mallintamiseen, animointiin, simulointiin, renderöintiin suunnattu työkalu, jota voidaan käyttää pelihahmojen ja efektien luomiseen (Develop 2014, 16).

- Photoshop on alan standardi 2D-grafiikan tekemiseen (Thorn 2013, 37; Pickthall 2012).

- FMOD on yksi eniten käytetyistä äänisisällön luomiseen tarkoitetuista työkalupaketeistä, joka pyrkii ammattimaisuuteen ja joustavuuteen (Develop 2014, 25).

Työkalun tekeminen itse on aina viimeinen vaihtoehto, johon pitäisi turvautua vain, jos ei ole olemassa mitään riittävän hyvää valmista työkalua saman työn tekemiseen tai jos valmiin työkalun hinta ylittää oman vastaavan kehityskustannukset. Kaikki aika, joka käytetään vastineen luomiseksi valmiille kaupalliselle työkalulle, on suoraan pois muusta kehitystyöstä. Sisältö, jonka luomiseen tarvitaan omia työkaluja, on usein pelikohtaista, juuri käsillä olevalle pelille ominaista. (Moar 2004; Gregory 2009, 47, 49; Hall 2007, 565.)

Tavallisesti sisällöntuotannon työkalujen käyttäminen rajoittuu pelinkehitystiimiin, mutta joskus työkalut jaetaan pelaajalle ja pelin ympärille rakentuva yhteisö saa ainakin osan sisällön luomiseen kuuluvasta vastuusta. Joissain peleissä sisältöä luodaan automaattisesti sitä varten kehitellyillä algoritmeilla. Vaikka pelin pelaajakunta tai älykäs algoritmi sisällön luojana kuulostaisi helpolta ja houkuttelevalta, se on usein se vaikeampi tie. (Hastings, Guha & Stanley 2009, 1; Graft 2012; Lambe 2012.)

Pelinkehittäjälle suunnattu työkalu vaatii usein merkittävän määrän harjoitusta ja tietoutta, jotta sen oppii hallitsemaan. Kuinka monella pelaajalla riittää mielenkiintoa työkalujen opettelemiseen tai kuinka monella pelinkehittäjällä on resursseja luoda helpommin käytettävät työkalut nimenomaan pelaajia varten? Algoritmin avulla automaattisesti luodun sisällön ongelma taas on se, että lopputulosta on vaikea hallita. Usein pieni määrä todella hyvää sisältöä saa vastapainokseen suuren määrän käyttökelvotonta kohinaa. (Hastings, Guha & Stanley 2009, 1; Lambe 2012.)

2.2.1 Työkalujen tekijät

Työkalujen tekemisestä vastaavat siihen erikoistuneet ohjelmoijat. Heille tärkeimpänä vaatimuksena ovat hyvät kommunikointitaidot. Heidän tehtävänsä on tehdä pelinkehityksen luoville rooleille ihanteelliset työkalut visionsa saavuttamiseen. Työkaluohjelmoijan roolissa poikkeuksellista muihin ohjelmoijiin verrattuna on se, että heidän asiakkaansa työskentelevät saman projektin parissa. (Moar 2004; Hefty 2007.)

Hyvän kommunikaation saavuttamiseksi jokaisesta asiakasryhmästä (graafikot, suunnittelijat jne.) olisi hyvä olla yksi ensisijainen kontakti, joka tapaa säännöllisesti työkaluohjelmoijien kanssa. Tapaamisissa keskustellaan uusien ominaisuuksien lisäämisestä työkaluihin, niiden tärkeysjärjestyksestä ja valmistumisen aikataulusta. Yksi kontaktin tärkeimmistä tehtävistä on antaa hyväksyntänsä valmistuneille ominaisuuksille ja erityisesti niiden käyttöliittymälle ennen

kuin ne otetaan käyttöön. Palaute työkalujen käyttäjiltä on hyvin tärkeää, koska työkalut tulevat heille paljon tutummiksi kuin tekevälle osapuolelle. Työkaluja käytetään usein tavoilla, joita niitä toteuttaessa ei osattu edes kuvitella. (Moar 2004; Hefty 2007.)

Työkaluissa on tärkeintä, että ne toimivat virheettömästi, ovat luotettavia ja niillä saa työn tehtyä. Niiden ei tarvitse olla ulkonäöltään kauniita ja viimeisteltyjä, mutta käyttökokemuksen pitäisi olla mahdollisimman hyvä. Työkalujen virheet ja ongelmat maksavat projektille koko ongelmien korjaamiseen kuluvan ajan. On tärkeää huomioida, että tämä tapahtuu usealla eri tasolla, koska samaan aikaan kun työkalujen käyttäjät eivät voi jatkaa työtään, ohjelmoijat eivät voi työskennellä tärkeämpien asioiden parissa. (Gregory 2009, 49; Moar 2004.)

Iso osa työkaluohjelmoijan työstä on tutkia mihin pelinkehityksessä kuluu aikaa ja kehittää ratkaisuja, joiden avulla toiminnan tehokkuus kasvaa. Tähän kuuluu usein toistuvien, täysin automatisoitavissa olevien työtehtävien tunnistaminen. Parhaassa tapauksessa ongelmat voidaan huomata jo ennen kuin ne alkavat viivästyttää projektia. Tässä on hyödyksi kokemus aiemmista projekteista. (Moar 2004; Hefty 2007.)

Työhön kuluvan ajan vähentämiseen on työkaluja kehitettäessä sovellettavissa kolme sääntöä (Moar 2004):

1. Käytettävyys on aina tärkeintä, mutta yhden henkilön työn tehostuminen ei saisi koskaan vaikuttaa huonontavasti muiden tehokkuuteen.

2. Työkalu sekä havaitsee virheet että reagoi niihin niin aikaisin kuin mahdollista.

3. Ennakoitu ja aikataulutettu viivästys tulee aina vähemmän kalliiksi kuin yllätyksenä tuleva. Toisin sanoen, vaikka huolellisuus viivästäisi työn valmistumista, se säästää kuitenkin lopulta aikaa.

Työhön kuluvan ajan vähentäminen täytyy huomioida kaikilla työkalun osa-alueilla, aina käyttöliittymäsuunnittelusta teknisiin ratkaisuihin. Yleisimmät toiminnot pitäisi saada suoritettua mahdollisimman vähillä klikkauksilla, ja käyttökokemusta häiritsevien viiveiden minimoimiseksi pitäisi käyttää riittävän nopeita ja tehokkaita algoritmeja. (Moar 2004.)

2.2.2 Työkalujen tekemisen ajoittaminen

Suurimman osan työkalujen kehittämiseen liittyvästä työstä pitäisi tapahtua projektin esituotantovaiheen aikana. Lopputuloksena pitäisi syntyä vakaa, luotettava ja tehokas kattaus työkaluja, joita käyttämällä peli on mahdollista toteuttaa. Parhaassa tapauksessa, lopputulosta hyödyntämällä, kaikista pelissä esiintyvistä erilaisista sisällön tyypeistä voidaan tehdä esimerkkeinä toimivia näytteitä, mutta on varsin selvää, että se ei aina käytännössä onnistu. (Moar 2004; Ward 2010; Sztajer 2013.)

Ihanteellisimmassa tilanteessa, esituotannon jälkeen, tuotannon alkaessa työkaluihin ei enää tehdä mittavia muutoksia, vaan enimmäkseen korjauksia ja huoltotöitä. Jos kuitenkin ajatellaan realistisesti, kaikki pelin järjestelmät eivät välttämättä ole tässä vaiheessa valmiita tai ollenkaan olemassa, jolloin myös työkaluja täytyy muuttaa tai tehdä kokonaan uusia. Merkittävien muutoksien pitäisi kuitenkin olla enimmäkseen poikkeuksia. (Moar 2004; Ward 2010.)

Tuotantovaiheen jälkeen, jälkituotantovaiheessa, työkalujen tekemiseen käytettävien resurssien pitäisi olla minimissään. Tarve uusien työkalujen kehitykselle ja vanhojen huoltamiselle on tässä vaiheessa äärimmäisen pieni. Poikkeuksena sääntöön ovat pelit, joiden mukana toimitetaan työkaluja loppukäyttäjän eli pelaajan käytettäväksi. (Moar 2004.)

3 TYÖKALUJEN TOTEUTTAMINEN PUMP-ACTION POPCORN -PELIIN

Työskentelin Bind-tiimissä Pump-Action Popcorn -pelin parissa ohjelmoijana ja vastuualueinani olivat osa pelin keskeisen toimintalogiikan ohjelmoimisesta ja työkalujen rakentaminen pelisuunnittelijan käytettäväksi. Kehityksessä käytettiin Unity-pelimoottoria ja C#-ohjelmointikieltä.

Bind on seitsemän hengen pelinkehitystiimi, joka toimii Kajaanissa ja omien tuotteidensa kehittämisen ohessa tekee myös alihankintatöitä. Suurin osa jäsenistä opiskelee vielä, ja osittain siitä syystä liiketoiminta tapahtuu toistaiseksi KAMK Kajability OSK:n alla aputoiminimellä.

Pump-Action Popcorn (kuvio 1) on kaksiulotteinen areenaräiskintä, jonka pääpaino on paikallisessa moninpelissä. Tärkeässä asemassa ovat myös pelikentälle sijoiteltujen pisteiden valtaus ja kaatuneiden kavereiden herättäminen takaisin mukaan toimintaan. Projekti tehtiin rajoitetussa ajassa Intelin Level Up -kilpailuun. Kilpailusta ei kuitenkaan tullut menestystä ja projekti on toistaiseksi jäissä.

Kuvio 1. Kuvankaappaus Pump-Action Popcorn -pelistä

3.1 Tavoitteet

Päätavoitteena oli luoda suunnittelijalle työkalut, joilla on mahdollista toteuttaa kaikki pelin power upit ja aseet kirjoittamatta riviäkään koodia. Jotta työkaluilla tuotettu sisältö ei jäisi yksipuoliseksi, haluttiin, että ne tarjoaisivat mahdollisimman paljon vapautta ja rajoittaisivat tekemistä mahdollisimman vähän. Työkalujen käyttäminen pitäisi liittää saumattomasti osaksi Unityn käyttöliittymää niin, että työkaluja käyttääkseen ei tarvitse avata erillistä editoria.

Työkaluilla pitäisi pystyä määrittämään vain miten aseet toimivat, eikä niillä ole tarkoitus vaikuttaa siihen miltä ne näyttävät. Sen sijaan aseiden ulkonäön suunnittelevat ja toteuttavat graafikot niin, että visuaalisuus sopii yhteen mekaanisen toteutuksen kanssa ja sopii pelille asetettuun teemaan ja graafiseen tyyliin.

3.2 Määrittely

Pelisuunnitteludokumenteissa oli olemassa jo muutamia alustavia asesuunnitelmia, joiden pohjalta oli varsin helppo määrittää, minkälaisia erilaisia aseita työkaluilla pitäisi olla mahdollista tehdä. Power upeja oli suunnitteludokumentissa käsitelty huomattavasti niukemmin, lähinnä oli listattu muutamia spekuloituja ominaisuuksia, joihin ne voisivat vaikuttaa. Nyt siis kartoitettiin sitä koko aseiden ja power upien kirjoa ja monimuotoisuutta, jota pelin olisi tarkoitus tarjota.

Aseista tehtiin listaus määritelmiä mahdollisimman erilaisille aseille, joita työkalulla pitäisi pystyä luomaan:

- **Pistooli** ampuu kerran nappia painamalla yhden ammuksen. Kaikista muista aseista poiketen tässä aseessa on rajaton määrä ammuksia. Jokaisen laukauksen jälkeen on pieni odotusaika ennen kuin voidaan ampua uudestaan.

- **Konekivääri** ampuu sarjatulella ammuksia, kun nappia pidetään pohjassa.

- **Assault rifle** ampuu lyhyitä sarjoja, joiden välissä tauko. Nappia voidaan ampuessa pitää pohjassa.

- **Sinko** ampuu törmäyksestä räjähtävän, suoraan matkustavan ammuksen. Ase ampuu jokaisella yksittäisellä painalluksella yhden ammuksen. Jokaisen laukauksen jälkeen on pitkä odotusaika ennen kuin voidaan ampua uudestaan.

- **Kranaatinheitin** ampuu painovoiman vaikutuksen alaisen räjähtävän ammuksen hieman yläviistoon. Ase ampuu jokaisella yksittäisellä painalluksella yhden ammuksen. Jokaisen laukauksen jälkeen on pitkä odotusaika ennen kuin voidaan ampua uudestaan.

- **Haulikko** ampuu useita ammuksia kerrallaan. Ammukset leviävät eivätkä matkusta täysin suoraan. Ase ampuu jokaisella yksittäisellä painalluksella yhden ammuksen. Jokaisen laukauksen jälkeen on keskipitkä odotusaika ennen kuin voidaan ampua uudestaan.

- **Sniper rifle** ampuu nopean kohteita lävistävän ammuksen. Ase ampuu jokaisella yksittäisellä painalluksella yhden ammuksen. Jokaisen laukauksen jälkeen on pitkä odotusaika ennen kuin voidaan ampua uudestaan.

Power upeista ei syntynyt samanlaista esimerkkien listausta, vaan sen sijaan keskityttiin ominaisuuksiin, joihin niiden pitäisi kyetä vaikuttamaan. Vaikutukset rajattiin pelaajien liikkumisnopeuteen, hyppykorkeuteen ja heidän kantamiensa aseiden ominaisuuksiin. Aseiden eri ominaisuuksia ei vielä ollut yksittäisten muuttujien tasolla suunniteltu, joten niiden suhteen ei vielä voitu mennä yksityiskohtiin. Päätettiin myös, että eri power upien vaikutukset kasautuisivat päällekkäin eli aktiivisena voisi olla useampia power upeja kerrallaan.

3.3 Suunnittelu

Määrittelyvaiheen jälkeen tiedettiin tarpeellisella tarkkuudella, mitä työkaluilla pitäisi pystyä saamaan aikaan ja oli aika suunnitella, millainen käyttöliittymän ja teknisen toteutuksen yhdistelmä täyttäisi tarpeet. Suunnittelussa pyrittiin ottamaan huomioon käyttöliittymän helppo lähestyttävyys ja teknisen toteutuksen helppo laajennettavuus.

Ensimmäinen, lähes itsestään selvä päätös oli käyttää C#-ohjelmointikieltä. Syynä päätökselle oli yksinkertaisesti se, että se tarjoaa eniten ominaisuuksia ja kaikilla tiimin ohjelmoijilla oli

siitä enemmän kokemusta kuin muista Unityn tukemista kielistä. Seuraavaksi mennään jo aika syvälle teknisen toteutuksen yksityiskohtiin.

3.3.1 Aseet

Päätettiin, että jokainen ase olisi instanssi samasta luokasta, eli niistä jokainen voitaisiin määrittää käyttämällä samoja muuttujia, eli perintää ei käytettäisi eri aseiden luomiseen. Tätä ajatusta tuki vahvasti Unityn prefab-ominaisuus, jonka avulla eri aseet eli saman luokan instanssit eri muuttujien arvoilla olisi helppoa säilöä projektissa. Tällä lähestymistavalla myös käyttöliittymä rakentuisi lähes itsestään, koska scriptissä on mahdollista määrittää halutut muuttujat muokattavaksi editorin käyttöliittymässä, joka näyttää ne luokan alla listauksena. Suurin edessä siintävä työ tuntui olevan oikeiden muuttujien valitseminen.

Korostettakoon vielä selkeyden vuoksi, että ellei erikseen tarkenneta, muuttujalla tarkoitetaan käyttöliittymässä suunnittelijan muokattavaksi tarkoitettua ominaisuutta. Ei pidä siis luulla, että käsiteltäisiin koko sitä muuttujien kirjoa, jonka luokka pitää sisällään. Niin yksityiskohtainen lähestymistapa olisi tämän aiheen yhteydessä täysin tarpeeton.

Toinen vaihtoehto yhden aseluokan sijaan olisi koostaa aseet useista pienemmistä komponenteista, jotka toteuttaisivat jokainen vain yhden toiminnon. Sitä harkittiin, mutta muuttujien määrän oletettiin pysyvän kuitenkin niin pienenä, että yhden luokan käyttäminen olisi helpommin hallittavissa oleva ja selkeämpi ratkaisu erityisesti käytettävyyttä ajatellen. Olisi suunnittelijalle selkeämpää, että jokainen ase koostuisi aina yhdestä samasta komponentista, josta löytyisivät selkeästi kaikki muokattavissa olevat ominaisuudet.

Tietynlainen erottelu päätettiin kuitenkin aseiden osalta toteuttaa: osa ominaisuuksista erotettiin erilliseen ammusluokkaan. Jokaiselle aseelle asetettaisiin muiden ominaisuuksiensa lisäksi ammustyyppi, joka pitää sisällään omat ominaisuutensa. Tällaisina ominaisuuksina nähtiin ammuksen tekemä vahinko, vaikuttaako siihen painovoima, lävistääkö se kohteita ja räjähtääkö se osumasta.

Aseen muuttujat

Ensimmäisiksi aseen muuttujiksi listattiin ammusten määrä, tulinopeus, kantama, ammuksen lähtönopeus, ammus-prefab ja se, vaatiiko jokainen laukaus erillisen napinpainalluksen, vai tulittaako ase automaattisesti, kun nappia pidetään pohjassa. Näillä muuttujilla olisi jo mahdollista luoda pistooli ja konekivääri. Assault riflen kohdalla kuitenkin havaittiin tarve uusille muuttujille: tulinopeuden lisäksi piti pystyä määrittämään yksittäisten sarjojen välissä oleva odotusaika sekä yksittäisen sarjan ammusten lukumäärä.

Singon ammuksen räjähtämisestä selvittiin pelkällä uudella ammustyypillä, koska se on ammuksen ominaisuus. Kranaatinheittimen ammukset ovat muuten samanlaisia, mutta niihin vaikuttaa painovoima. Pelkkä uusi ammustyyppi ei kuitenkaan riittänyt, vaan täytyi lisätä myös uusi muuttuja aseeseen määrittämään ammukselle lähtösuunta. Kranaatinheittimen ammukset laukaistaan tavallisuudesta poiketen yläviistoon, ennen kuin painovoima lopulta tuo ne alas.

Seuraava ase, jonka kohdalla huomattiin, että nykyiset muuttujat eivät riitä, oli haulikko. Sen piipusta lähtee kerrallaan useita ammuksia, joten tarvittiin muuttuja ammusten määrälle laukausta kohden. Haulikon ammukset myös leviävät, ja siksi tarvittiin muuttuja sille, paljonko satunnaisuutta ammuksen lähtösuuntaan lisätään. Myös lähtönopeuteen lisättävälle satunnaisuudelle luotiin muuttuja, koska vaikuttaa luonnollisemmalle, että kaikki ammukset eivät lennä yhtä pitkälle. Sniper riflestä ei tullut enää aseeseen uusia muuttujia, vaan selvittiin taas pelkällä ammustyypin muuttamisella; se määritettiin lävistäväksi.

3.3.2 Power upit

Ensimmäinen päätös power upien osalta oli sama, mikä tehtiin aikaisemmin jo aseissa: kaikki muuttujat yhteen luokkaan, ei perintää tai pienemmistä osasista koostamista. Power upien suunnittelu oli mahdollista aloittaa vasta kun aseet oli jo saatu toteutettua ainakin alustavasti. Syynä tälle oli se, että muuttujien täytyi olla suurin piirtein lukkoon lyötyjä, koska juuri niihin power upien oli tarkoitus vaikuttaa.

Valittiin, että power upit vaikuttavat aseiden osalta ammusten tekemään vahinkoon, ammusten määrään, kantamaan, viiveeseen laukausten välillä, viiveeseen sarjojen välillä, ammusten leviämiseen ja ammusten lähtönopeuteen. Pelihahmon muuttujista valittiin hypyn voimak-

kuus, hypyn kesto ja liikkumisnopeus. Muuttujien suhteellisen suuresta määrästä huolimatta arveltiin, että niitä pitäisi vielä myöhemmin lisätä.

Melko nopeasti havahduttiin siihen, että eri ominaisuuksien tehostaminen ei olisi mielekästä samalla lähestymistavalla. Joidenkin kanssa olisi tärkeää pystyä asettamaan muuttujan arvolle kerroin ja toisten kanssa olisi kätevämpää lisätä siihen jokin arvo. Tämä päätettiin hoitaa siten, että luotaisiin muuttujaa kuvaava luokka, johon olisi mahdollista määrittää arvo ja sen käsittelytapa, kerrotaanko vai lisätäänkö. Jokainen power upin muuttuja, joka vaikuttaa johonkin ominaisuuteen, olisi siis tämän luokan instanssi.

Power upien vaikutukset hoitavan järjestelmän osalta tehtiin päätös, että sille olisi mahdollista syöttää lista sallituista power upeista, joista se arpoisi jonkin, kun sellaista pyydettäisiin. Järjestelmä manipuloisi suoraan pelaajien ja aseiden muuttujia, kun power upin vaikutukset tulevat voimaan. Power upien vaikutusten pinoutumisen yksityiskohdat päätettiin jättää mietittäväksi myöhemmin samanaikaisesti toteuttamisen ohessa.

3.4 Toteutus

Tarpeettoman toiston välttämiseksi tässä osiossa tuodaan pääasiassa esille vain suurimmat muutokset ja odottamattomat asiat, joita tuli vastaan, kun suunnitellut ominaisuudet toteutettiin käytännössä. Mainitsematta jääneistä asioista voidaan siis olettaa, että niiden toteuttaminen sujui suoraviivaisesti suunnitelmien mukaan. Toteuttaminen työnä koostui käytännössä enimmäkseen ohjelmakoodin kirjoittamisesta. Osana työtä oli myös Unityn editorin käyttämistä enimmäkseen scenejen muokkaamiseen ja prefabien luomiseen.

Aseiden toteuttaminen oli suoraviivaisin osuus. Se meni suunnitelmien mukaan, eikä suurempia mainitsemisen arvoisia yllätyksiä tullut. Tästä syystä aiheelle on turha omistaa omaa osiotaan. Uusia muuttujia lisättiin ja pieniä muutoksia tuli, mutta ne liittyivät enemmän muihin kokonaisuuksiin, joita käsitellään edellä.

3.4.1 Power upit

Power upien vaikutusten pinoutumisjärjestelmän suunnittelutyö oli jätetty yksityiskohdiltaan varsin väljäksi. Toisaalta se oli hyvä, koska olisi todennäköisesti mennyt enemmän aikaa

miettiä sitä pidemmälle teoriassa, kun käytännössä kokeilla ja todeta, mikä toimii hyvin. Syynä tähän oli se, että pelissä järjestelmineen oli paljon osia, joiden kanssa power upien piti toimia yhteen. Asiaa ei helpottanut lainkaan se, että kokonaisuus haki vielä muotoaan.

Ensimmäinen lähes itsestään selvä asia tehdä oli pitää kirjaa aktiivisista power upeista listan muodossa. Kun uusi power up aktivoidaan, sen vaikutukset lisätään niiden instanssien muuttujiin, joita se koskee. Kun taas jokin power upeista poistuu, sen vaikutukset vähennetään muuttujista. Kuten huomata saattaa, ensimmäinen vedos oli todella naiivi ja yksinkertainen.

Ensimmäinen järjestelmästä löytynyt ongelma oli se, että osa power upien vaikutuksista suorittaa kertolaskun muuttujille ja osa taas yhteenlaskun. Tästä syystä power upien hankkimisjärjestyksellä oli suuri merkitys niiden vaikutuksen suuruuteen. Esimerkiksi jos jonkin muuttujan arvoon lisätään ensin luku ja sen jälkeen se kerrotaan jollain luvulla, lopputulos on suurempi kuin tilanteessa, jossa kertolasku olisi suoritettu ensin. Ratkaisuna, aina kun uusi power up aktivoidaan tai vanha poistetaan, kaikkiin asioihin pelimaailmassa, joihin power upit voivat vaikuttaa, palautetaan oletusarvot. Sitten käydään ensiksi läpi kaikki yhteenlaskettavat vaikutukset ja lisätään ne oletusarvoihin, ja vasta sen jälkeen käsitellään kertolaskut. Tämä mahdollistaa sen, että vaikutus on aina sama huolimatta siitä, missä järjestyksessä power upit aktivoituvat.

Ratkaisua toteuttaessa törmättiin kuitenkin uuteen ongelmaan: oletusarvoja ei säilötä missään. Päädyttiin tekemään luokka niistä arvoista, joihin power upit voivat vaikuttaa. Tästä luokasta säilötään yksi instanssi oletusarvoina ja toinen instanssi aktiivisina arvoina. Aina kun halutaan palauttaa oletusarvot, ne voidaan helposti kopioida oletusarvojen instanssista aktiivisiin arvoihin. Tämä muutti hieman aseiden ja pelaajahahmon toteutusta, muun muassa paketoi käyttöliittymässä suunnittelijan syöttämät arvot ”Default Values”-objektin alle.

Seuraava ongelma oli se, että kun pelaaja kerää uuden aseen, se on tietysti uusi instanssi, jonka muuttujilla on oletusarvot. Tämä oli kuitenkin onneksi yksinkertainen ratkaista: ase ilmoittaa power up -järjestelmälle, että se on juuri luotu ja tarvitsee oikeat arvot. Ilmoituksen saadessaan power up -järjestelmä lisää aseen muuttujiin oikeat vaikutukset ja arvot ovat sitten ajan tasalla.

Järjestelmä toimi jo muuten virheettömästi, mutta huomattiin yksi poikkeus: vahinko. Vahinko on ammuksen ominaisuus ja tuntui kömpelöltä, että jokainen ammuksen instanssi olisi suoraan sidoksissa power up -järjestelmään. Tämä siitä syystä, että ammuksia on pelitilan-

teessa kerrallaan olemassa todella paljon ja yksittäisen ammuksen elinaika on varsin lyhyt. Ratkaisuna ammus-luokalle esiteltiin staattiset jäsenmuuttujat, joita power up -järjestelmä manipuloi. Muuttujat pitävät sisällään kertoimen ja lisän, joita käyttämällä jokainen ammus laskee helposti itse omaan vahinkoonsa power upien vaikutuksen, kun se on tarpeellista.

Vielä viimeistelynä power upiin lisättiin myös kaksi booleania: vaikuttaako se pelaajaan ja vaikuttaako se aseeseen. Ne lisättiin, koska on turha edes käsitellä aseeseen vaikuttavia muuttujia, jos power upin on tarkoitus vaikuttaa vain pelaajaan, ja päinvastoin. Lisäksi olisi suunnittelijalle helpompi vain klikata vaikutukset pois päältä ilman, että tarvitsee käsin pyyhkiä kaikkia arvoja pois.

3.4.2 Räjähdykset ja ammuksen laajentaminen

Isoin suunnittelussa ohitettu olennainen asia olivat räjähdykset. Niiden tarkempi tekninen määrittely oli jätetty huomiotta, ja oli vain päätös siitä, että jotkin asiat voivat räjähtää. Ominaisuus päätettiin toteuttaa uutena ammustyyppinä, koska ammus oli varsin lähellä räjähdyksen keskeisiä ominaisuuksia: se aiheuttaa vahinkoa kohteisiin, joihin se osuu. Tähän tarvittiin ammukselle kuitenkin myös uusi ominaisuus: objektin luominen törmätessä. Esimerkiksi singon ammus törmätessään tuhoutuu ja luo uuden räjähdystyyppisen ammusobjektin. Yksi uusi ominaisuus ei kuitenkaan vielä riittänyt, vaan lisättiin tuki myös ammuksen ajastetulle tuhoutumiselle. Olisi nimittäin aika kummallista, jos räjähdys jäisi pelimaailmaan odottamaan, kunnes se törmää johonkin, ennen kuin tuhoutuu.

Räjähdysten toteuttaminen ammuksina teki niistä myös uudella tavoin laajennettavia: voitaisiin tehdä ammuksia, jotka törmätessään jakautuvat pienemmiksi ammuksiksi, jotka edelleen jakautuvat vielä pienemmiksi jne. Lisäksi ajastettu tuhoutuminen avasi mahdollisuuden käsikranaatin tavoin käyttäytyvälle aseelle. Tämä vaatii kuitenkin sen, että lisättiin tuki objektin luomiselle ammuksen tuhoutuessa ajastetusti. Kranaatin ei nimittäin ole tarkoitus räjähtää vielä törmäyksestä. Tuntui myös hienolta idealta, että ammus voisi toistuvasti määrätyn ajan välein luoda uusia objekteja koko olemassaolonsa ajan, joten sekin mahdollistettiin. Nyt siis ammuksella oli kolme tapaa luoda objekteja: törmätessään, tuhoutuessaan ajastetusti ja toistuvasti määrätyn ajan välein. Nämä lisäykset kasvattivat ammuksen muuttujamäärää ehkä jopa jo liiankin suureksi, mutta toisaalta ne laajensivat mahdollisuuksia erilaisten ammusten luomiseen huomattavasti.

Ehkä turhimpana asiana ammukselle lisättiin ominaisuus, jolla se voidaan tuhota törmäyksen jälkeen viiveellä. Eli se ei tuhoudu välittömästi törmätessään, vaan odottaa vielä määrätyn ajan verran ennen kuin tuhoutuu. Mitään käytännön käyttötarkoitusta tälle ominaisuudelle ei ollut mietitty, se vain näytti kivalta ja se jäi sitten vain olemaan varmuuden vuoksi.

Kaikki uusista lisäyksistä eivät olleet vain mekaanisia; yhdellä oli pelkästään visuaalinen merkitys. Graafikoiden pyynnöstä ammus sai ominaisuuden, jolla voidaan määrittää, määräytyykö sen spriten rotaatio liikesuunnan mukaan, kuten on luotimaisten ammusten kanssa sopivaa, vai satunnaisesti. Satunnainen rotaatio lisää variaatiota sellaisten ammusten ulkonäköön, joiden visuaalisella rotaatiolla liikesuuntaan nähden ei ole merkitystä. Ero tulee hyvin esiin, kun verrataan esimerkiksi luotia ja palloa.

3.4.3 Rekyyli, screen shake ja ääniefektit

Asiat, joita ei suunnitellessa ollut otettu huomioon, eivät kuitenkaan loppuneet vielä. Sellaisia olivat rekyyli, screen shake ja isoimpana kokonaisuutena ääniefektit. Kaikki kolme toteutettiin kaiken aiemmin suunnitellun päälle sen kummempia miettimättä, ja ne lisäsivät uusia muuttujia aseille, power upeille ja jo ennestäänkin varsin suureksi paisuneelle ammukselle. Erityisesti ääniefektien toteuttaminen olisi sujunut paremmin, jos se olisi otettu huomioon jo suunnitteluvaiheessa.

Rekyyli ja screen shake olivat onneksi suhteellisen suoraviivaisia ominaisuuksia toteuttaa. Aseelle lisättiin muuttuja sille, paljonko se aiheuttaa rekyyliä eli paljonko pelihahmo lennähtää ampuessaan taaksepäin. Power up sai muuttujan, jolla voidaan tehostaa aseiden rekyyliä, jotta aseen tehoa lisäävien power upien teho välittyisi pelaajalle paremmin. Aseelle lisättiin myös screen shakea säätelevät muuttujat, eli paljonko näytön kuva tärisee ammuttaessa ja kuinka kauan. Screen shake aiheutti lisämuuttujia myös ammukselle, koska ammus-luokkaa käytettiin räjähdysten toteuttamisessa ja haluttiin, että ruutu tärisisi räjähdyksen ilmestyessä peliin. Power up sai muuttujan myös screen shaken tehostamiseen, syynä sama kuin rekyylinkin kanssa: voidaan paremmin välittää pelaajalle, että nyt on käytössä enemmän voimaa.

Ääniefektit

Ääniefektit aiheuttivat kaikkeen muuhun nähden eniten töitä, ja niiden implementoimiseen vaaditun työmäärän suuruus yllätti. Olin työskennellyt Unityn äänipuolen kanssa rajoitetusti,

ja nyt yritin käyttää sitä ensimmäistä kertaa mihinkään näin laajaan. Aiemmista projekteista opitut ääniin liittyvät käytännöt osoittautuivat riittämättömiksi ja äänille oli pakko ohjelmoida oma järjestelmänsä, joka niitä hallinnoi. Vaikka peli oli kaksiulotteinen, haluttiin käyttää kolmiulotteisia ääniefektejä, koska haluttiin, että osa pelitilanteesta välittyy pelaajille jo pelkkiä ääniefektejä kuuntelemalla.

Ääniefektit lisäsivät muuttujia aseeseen, koska haluttiin, että suunnittelija voi määrittää aseen laukaukselle haluamansa ääniefektin. Myöhemmin huomattiin, että jotkin aseista ampuvat ammuksia niin nopealla tahdilla, että niille tämä lähestymistapa ei toimi. Koska jokaista laukausta kohden toistetaan ääniefekti, yksittäiset ääniefektit puuroutuivat rumaksi äänivalliksi. Ongelma ratkaistiin lisäämällä mahdollisuus toistaa jatkuvasti valittua ääniefektiä koko sen ajan, kun ampumisnäppäintä pidetään pohjassa.

Ammus tarvitsi muuttujan, johon sille voidaan asettaa ääniefekti, joka toistetaan sen tuhoutuessa. Erityisesti räjähtävien ammuksien kanssa tämä oli tärkeää. Ei mennyt kauaakaan, kun huomattiin tarve vielä toisellekin muuttujalle. Haluttiin, että ammus voisi pitää jotain tasaista ääntä, esimerkiksi kohinaa. Lisättiin siis muuttuja, johon voidaan asettaa ääniefekti, jota toistetaan jatkuvasti ja joka liikkuu ammuksen mukana. Lähestyvä kohina tekee tuhovoimaisista ja hitaista ammuksista pelaajan näkökulmasta entistä uhkaavampia.

Koska äänet olivat aika läheisesti liitoksissa moniin asioihin pelissä, kuten aseisiin, ammuksiin ja pelaajan toimiin, täytyi aika ajoin muuttaa niiden teknisiä toteutuksia, kun äänijärjestelmän keskeisiin toimintoihin tehtiin muutoksia. Vasta kolmas versio äänijärjestelmästä toimi niin, että se tuki kaikkia haluttuja ominaisuuksia niin, että käyttökokemus oli riittävällä tasolla suunnittelijan käytettäväksi.

3.5 Testaus

Kehitystiimin pienuudesta johtuen ei nähty tarpeelliseksi ylläpitää liian byrokraattista ja monimutkaista testauskoneistoa. Testaus hoidettiin koekäyttämällä työkaluja ensin toteuttavan ohjelmoijan toimesta, ja kun ne vaikuttivat virheettömiltä, luovuttamalla ne suunnittelijan käyttöön. Luovutettaessa opastettiin lyhyesti miten työkalua käytetään.

Aina kun suunnittelija törmäsi ongelmaan tai huomasi haluavansa jonkin uuden toiminnon, se lisättiin ohjelmoijan työtehtävien listaan. Kun tehtävä tuli ohjelmoijalla listassa vastaan,

hän teki tarvittavat muutokset, testasi työkalun ensin itse ja luovutti uuden version suunnittelijalle. Joskus suunnittelija löysi ongelmia jo työkaluja luovutettaessa ja ne lisättiin tehtävälistaan välittömästi.

Tehtävälistan korjaukset ja uudet ominaisuudet järjestettiin aina tärkeysjärjestykseen, pyrkien siihen, että työkaluihin tehtyjä muutoksia ei tarvitsisi odotella kauaa toimettomana, vaan suunnittelijalla olisi aina jotain työtä. Useat ominaisuuksista eivät ehtineet edes valmistua, kun niistä löytyi jo jokin ongelma, josta suunnittelijan kanssa täytyi keskustella ennen kuin työtä voitiin jatkaa.

3.6 Lopputuloksen analysointi

Lopputulosta voitiin aloittaa analysoimaan, kun työkalujen ensimmäiset versiot valmistuivat ja suunnittelija otti ne käyttöön. Enimmäkseen analysointi koostui suunnittelijan työskentelyn havainnoinnista ja hänen työnsä jäljen tarkastelusta. Lopputuloksen onnistuneisuus voitiin todeta vertaamalla havaintoja alussa asetettuihin tavoitteisiin.

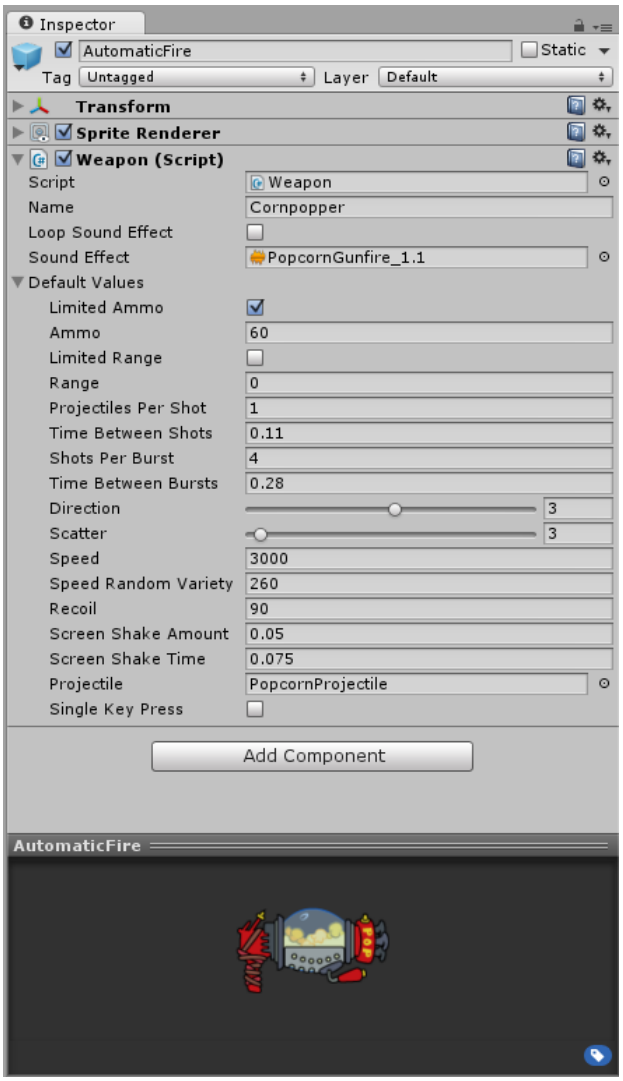
Suunnittelija oppi käyttämään työkaluja nopeasti ja tarvitsi vain hyvin vähän opastusta. Käyttöönotossa näytettiin, mistä työkalut löytyvät ja kerrottiin lyhyesti perusteet niiden käyttämisestä. Tarkempaa opastusta ei vaadittu, ja työkalut tuntuivat selittävän itse itsensä. Ne mahdollistivat täysin itsenäisen työskentelyn ja suunnittelija pystyi toteuttamaan visionsa mukaiset aseet ja power upit. Työkalut olivat kaikki osana Unityn käyttöliittymää ja suunnittelijan ei koskaan tarvinnut avata erillistä editoria. Työkalut eivät rajoittaneet suunnittelijan luovuutta ja tarjosivat hänelle paljon vapautta. Havaintojen perusteella voidaan todeta, että työkaluille asetetut vaatimukset täyttyivät.

Muutamia kertoja suunnittelija tarvitsi työkaluihin jonkin uuden ominaisuuden tai löysi jonkin ongelman, mutta aihetta käsiteltiin tarkemmin jo testausosiossa ja se kuuluu luonnollisena osana käyttöönottoon ja sovelluskehitykseen. Työkaluihin tehdyt muutokset ja löydetyt ongelmat olivat kaikki varsin pieniä, ja kokemuksiin edellisistä projekteista peilaten, olisi ollut huolestuttavampaa, jos mitään ei olisi tarvinnut muuttaa.

3.7 Esimerkkejä sisällöstä

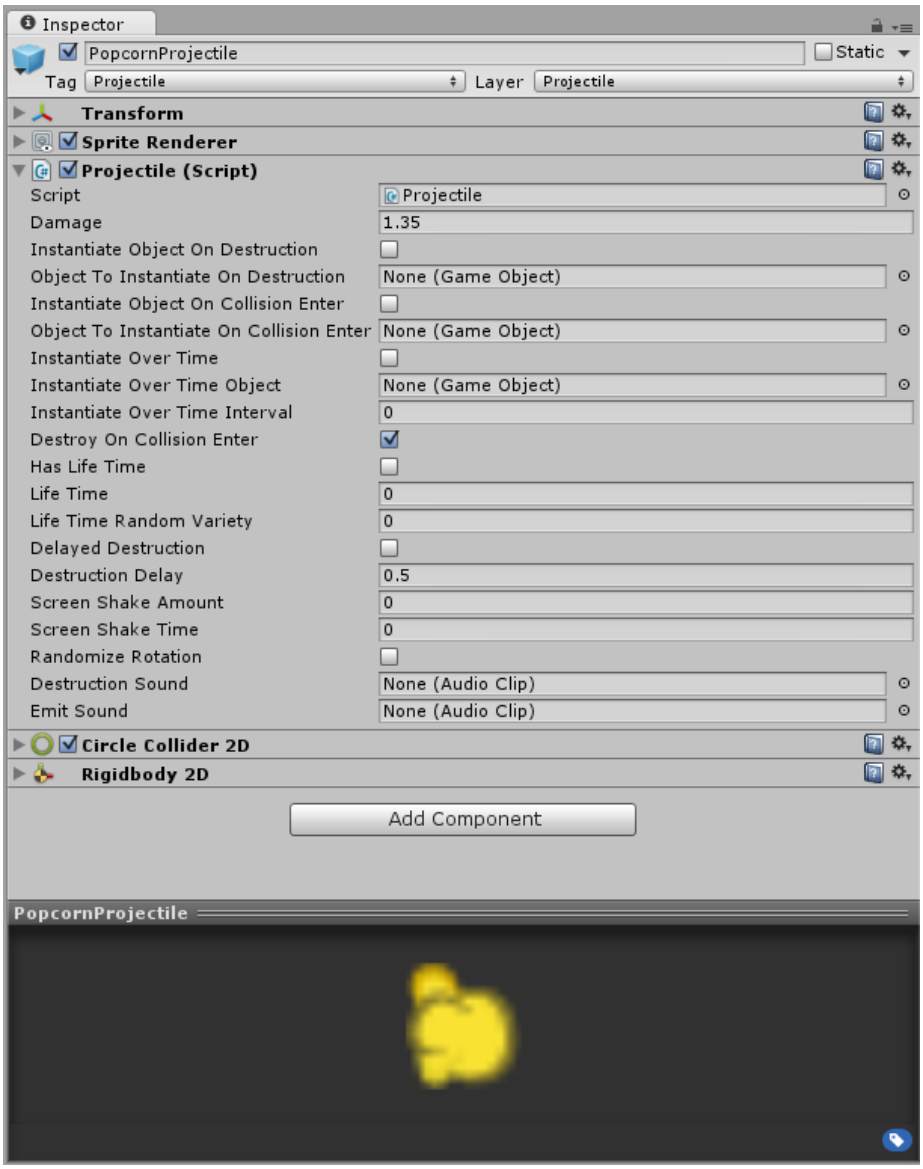
Tässä luvussa käsitellään muutamia esimerkkejä suunnittelijan työkaluilla toteuttamasta pelisisällöstä, jotta työkaluista, niiden käyttöliittymästä ja käytännön käyttökokemuksesta saisi paremman käsityksen. Kutakin sisältötyyppiä on edustamassa yksi esimerkki, eli esittelyssä on yksi ase, yksi ammus ja yksi power up. Esimerkeissä ovat nähtävissä työkalujen käyttöliittymät ja suunnittelijan syöttämät arvot.

Kuviossa 2 on esitettynä suunnittelijan tekemä ase, joka ampuu sarjatulella popcornammuksia. Ammukset lentävät rajattomalle etäisyydelle, mutta niitä on käytettävissä vain rajattu määrä. Ampumissuunta ja ammusten leviämä ovat asetettavissa liukusäätimillä, koska siten on helpompi pitää niiden arvot toimivalla alueella. Kuvion alaosassa on myös nähtävissä artistin aseelle suunnittelema ja toteuttama ulkoasu.



Kuvio 2. Suunnittelijan toteuttama ase Unityn käyttöliittymässä

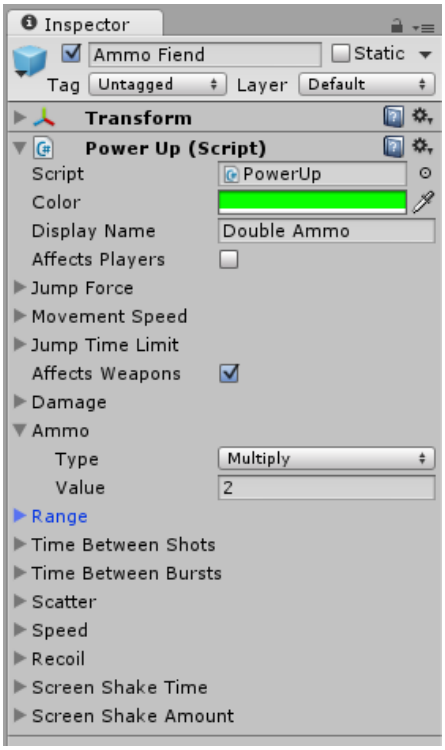
Kuviossa 3 on nähtävissä odottamattoman suureksi kasvanut ammuksen käyttöliittymä. Jälkikäteen tarkasteltuna osan muuttujista saisi karsittua pois, mutta se on jäänyt toteuttamishetkellä huomaamatta. Esimerkiksi ”Instantiate Object On Destruction” ja muut objektien luomista koskevat valintalaatikot ovat turhia, koska on itsestään selvää, että jos kenttään ei ole asetettu mitään objektia, niin mitään ei myöskään luoda. On kuitenkin tärkeää näyttää työkalut sellaisina kuin suunnittelija niitä käytti, siksi virheet jätettiin korjaamatta.



Kuvio 3. Suunnittelijan toteuttama ammus Unityn käyttöliittymässä

Seuraavan sivun kuvion 4 power up on yksinkertainen ja siksi esimerkkinä hyvä. Se kaksinkertaistaa aseen ammusten määrän. Koska se ei sisällä mitään muita vaikutuksia, muut kentät pienennettiin, jotta käyttöliittymä ei veisi liian paljon tilaa. Tämä power up vaikuttaa vain aseeseen, ei mihinkään pelihahmon ominaisuuksiin. Jälkikäteen tarkasteltuna olisi vielä selke-

ämpää, jos voitaisiin pienentää kaikki pelaajan arvot yhden otsikon alle, josta ne saataisiin myös näkymään tarvittaessa.



Kuvio 4. Suunnittelijan toteuttama power up Unityn käyttöliittymässä

4 POHDINTA

On helppo aliarvioida omien työkalujen tekemisen tärkeys ja se, miten paljon niillä voidaan säästää aikaa. Monissa aiemmissa projekteissa työkalujen toteuttamiseen varattu aika on ollut liian lyhyt, koska kaikki muu työ on nähty tärkeämpänä. Nyt jälkeenpäin on helppo spekuloida, että jos työkaluihin olisi käytetty enemmän aikaa, sitä muuta työtä ei välttämättä olisi ollut niin paljon tai ainakaan sen tekemiseen ei olisi kulunut niin paljon aikaa.

Jo näin lyhyessä projektissakin huomasi, että työkaluihin käytetty aika kannattaa. Mikäli projektille olisi ollut enemmän aikaa, edut olisivat olleet vieläkin selvemmät. Se, että suunnittelija pystyi itse tuottamaan tehokkaasti sisältöä, vähensi huomattavasti ilmiötä, jossa suunnittelija kuvailee, ja ohjelmoija tekee. Välikäsi vääristää visiota, väärinkäsitykset aiheuttavat ongelmia ja ohjelmoija ei voi keskittyä tärkeämpiin asioihin. Lisäksi tällä lähestymistavalla rajat sille, mitä on mahdollista tehdä ja mitä ei, on pakko määrittää suhteellisen tarkasti jo projektin alussa. Ei mene aikaa siihen, että varaudutaan liian pitkään aivan kaikkeen, koska mahdollisuuksia ei ole rajattu.

Yhden komponentin toteutustapa toimi aseen ja power upin kanssa varsin hyvin, mutta erityisesti ammuksen kanssa olisi hyödytty siitä, että se olisi koostettu pienemmistä palasista. Sille kertyi paljon muuttujia, joihin liittyvä toimintalogiikka olisi ollut helppo eristää erillisiin komponentteihinsa.

Ääniefektit olisi pitänyt ottaa huomioon jo suunnitteluvaiheessa, niin niiden kanssa ei olisi mennyt niin kauan. Toisaalta tuli yllätyksenä, että aiemmissa projekteissa opitut käytännöt eivät soveltuisi vähän isompaan projektiin. Eniten yllätti kuitenkin ehkä se, että miten vähän Unityssa oli valmiita toimintoja tähän tarkoitukseen, etenkin kun Unityn seuraavan, vieläkin julkaisemattoman, version huomattavasti kehittyneempi äänijärjestelmä oli samaan aikaan paljon esillä.

Työkalut onnistuivat kuitenkin pienistä virheistään huolimatta paremmin kuin itse peli. Pelin isoin ongelma on se, että kaikki power upit tuntuvat samanlaisilta ja on usein hankala tiedostaa niiden eri vaikutuksia. Se, että power upeja voi olla todella paljon päällekkäin, pakottaa suunnittelun puolesta sen, että ne ovat kaikki yksistään aika mietoja. Ongelmaa pahentavat entisestään riittämättömät visuaaliset indikaattorit.

Projektiin käytetyistä työtunneista, tehdyistä huomioista ja opituista asioista on huomattavasti apua jos projektia lähdetään joskus vielä jatkamaan. Monet asiat tehdään kuitenkin todennäköisesti alusta asti uudestaan. Usein parhaan lopputuloksen saavuttaakseen asia pitää tehdä ensin kerran valmiiksi, sitten aloittaa alusta ja lopuksi tehdä se opitun tiedon valossa uudestaan. Vaikka juuri tätä projektia ei enää koskaan jatkettaisiinkaan, siitä opittuja asioita voidaan varmasti soveltaa kaikkeen tulevaan pelinkehitykseen.

LÄHTEET

Chandler, H. & Chandler, R. 2011. Fundamentals of Game Development. Yhdysvallat: Jones & Bartlett Learning.

De Carli, D. M. & Bevilacqua, F. & Pozzer, C. T. & d'Ornellas, M. C. 2011. A survey of procedural content generation techniques suitable to game development. NTIC-Universidade Federal do Pampa, Universidade Federal da Fronteira Sul & Universidade Federal de Santa Maria.

Develop 2014. Develop 100: The Tech List. http://content.yudu.com/A2xcc7/Dev100TechList2014/resources/index.htm (Luettu 25.7.2014)

Graft, K. 2012. User-generated content: When game players become developers. Gamasutra. http://www.gamasutra.com/view/news/179493/Usergenerated_content_When_game_players_become_developers.php (Luettu 22.6.2014)

Gregory, J. 2009. Game Engine Architecture. Yhdysvallat: A K Peters, Ltd.

Hall, J. B. 2007. XNA Game Studio Express: Developing Games for Windows and the Xbox 360. Yhdysvallat: Cengage Learning.

Hastings, E. J. & Guha, R. K. & Stanley, K. O. 2009. Evolving Content in the Galactic Arms Race Video Game. University of Central Florida.

Hefty J. 2007. The Making of the Making of Quake Wars: Enemy Territory. http://pc.gamespy.com/pc/enemy-territory-quake-wars/770741p1.html (Luettu 8.8.2014)

Labschütz, M. & Krösl, K. & Aquino, M. & Grashäftl, F. & Kohl S. 2011. Content Creation for a 3D Game with Maya and Unity 3D. Vienna University of Technology.

Lambe, I. 2012. Procedural Content Generation: Thinking With Modules. Gamasutra. http://www.gamasutra.com/view/feature/174311/procedural_content_generation_.php (Luettu 17.8.2014)

Moar, D. 2004. Growing a Dedicated Tools Programming Team: From Baldur's Gate to Star Wars: Knights of the Old Republic. Gamasutra. http://www.gamasutra.com/view/feature/130468/growing_a_dedicated_tools_.php (Luettu 20.5.2014)

Pickthall J. 2012. Get started in videogame art. Creative Bloq. http://www.creativebloq.com/career/getting-started-game-art-1012940

Stankowicz, J. 2013. Skinned Animation: Runtime. https://josephstankowicz.wordpress.com/2013/06/07/skinned-animation-runtime/ (Luettu 18.9.2014)

Sztajer P. 2013. The 15 Steps of (Particulars) Pre-production. Gamasutra. http://www.gamasutra.com/blogs/PaulSztajer/20130324/189171/The_15_Steps_of_Particulars_Preproduction.php (Luettu 5.10.2014)

Thorn, A. 2013. Learn Unity for 2D Game Development. Yhdysvallat: Apress

Togelius, J. & De Nardi, R. & Lucas, S. M. 2007. Towards automatic personalised content creation for racing games. University of Essex.

Ward, J. 2010. Tools Programmer Fundamentals. The Toolsmiths. http://thetoolsmiths.org/2010/01/18/tools-programmer-fundamentals/ (Luettu 5.10.2014)