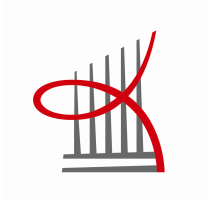


PC-PELIN UUDISTAMINEN AMERIKKA-KESKEISILLE MARKKINOILLE

Kristian Sivonen

Opinnäytetyö
Joulukuu 2013
Tietojenkäsittely

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

KRISTIAN SIVONEN:
PC-pelin uudistaminen Amerikka-keskeisille markkinoille

Opinnäytetyö 31 sivua
Joulukuu 2013

Tämän opinnäytetyön aiheena on Fragment Production -pelifirman Rescue 2013: Everyday Heroes -nimiselle palo- ja pelastustoimintaa käsittelevälle pelille tehty uudistustyö, jossa pelin eurooppalaiset ympäristöt, ajoneuvot ja hahmot muutettiin amerikkalaisiksi. Pelin laatua ja erityisesti sen graafista ulkoasua ja tehokkuutta oli määrä parantaa huomattavasti. USA-versiota Rescuesta oli määrä myydä suositussa Steam-latauspalvelussa.

Tavoitteena on selvittää, miten Fragment Productionin tulevien hankkeiden toteutusta voidaan parantaa ja tehostaa ja mitkä työtavat voidaan todeta jo sellaisenaan toimiviksi. Tarkoituksena on koota kattava raportti projektin etenemisestä, siinä tehdyistä ratkaisuista ja ilmenneistä haasteista. Työn on määrä toimia pohjana sisäiseen käyttöön ja mahdollisesti ulkoiseenkin jakeluun tehtävälle niin kutsutulle post mortem -dokumentille.

Rescue on tehty erittäin suositulla ja ominaisuuksiltaan kattavalla Unity-pelinkehitystyökalulla ja sen ohjelmointikielenä käytetään pääasiassa C#:ia.

Asiasanat: pelit, pelinkehitys, grafiikka, ohjelmointi

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in Business Information Systems

KRISTIAN SIVONEN:
Renewing a PC Game for American-focused Market

Bachelor's thesis 31 pages
December 2013

This bachelor's thesis is about renewing Rescue 2013: Everyday Heroes, a game by Fragment Production Ltd. about firefighting and rescue operations, for American audiences. Environments, vehicles and characters were changed from generally European to American. Moreover, the game was further developed to look and perform better. The plan was to sell the game on the popular digital distribution service, Steam.

The purpose of this thesis was to find out how the new practices found during the project can be adopted to future projects at Fragment and to see if some of the old ways are good as they are. The idea was to thoroughly report the project's progress, the new techniques and working practices found during it as well as its various challenges. This thesis was intended to form a base for a so called post mortem document to be used within the company and perhaps also published.

Rescue was developed with Unity, a popular and versatile game development tool. The main programming language in the project is C#.

Key words: games, game development, graphics, programming

SISÄLLYS

ERITYISSANASTO

verteksi (vertex)	piste kolmiulotteisessa avaruudessa
tekstuuri (texture)	3d-mallin pintakuvio
varjostinohjelma (shader)	näytönohjaimen grafiikkasuorittimen suorittama ohjelma
piirtokutsu (draw call)	näytönohjaimelle annettu paketti piirrettäviä kohteita

1 JOHDANTO

Fragment Production Oy on vuoden 2012 alussa perustettu tamperelainen pelinkehitysyritys, jonka työntekijämäärä liikkuu tämän opinnäytetyön kirjoitusaikaan 15-20 hengen välillä. Sen ensimmäinen julkaistu peli, palo- ja pelastustoimintaa käsittelevä reaaliaikainen strategiapeli Rescue 2013: Everyday Heroes, ilmestyi Saksan markkinoille toukokuun loppupuolella vuonna 2013. Sittemmin Rescue on lokalisoitu Ison-Britannian, Ranskan ja viimeisimpänä Yhdysvaltojen markkinoille.

Lokalisointi on tässä yhteydessä tarkoittanut pelissä esiintyvien hälytys- ja siviiliajoneuvojen ja oikeaa maailmaa jäljittelevien ympäristöjen muuttamista kohdemaan mukaisiksi. Ison-Britannian ja Ranskan lokalisaatioversioissa ympäristöistä on käytännössä vaihdettu alunperin saksalaistyylinen arkkitehtuuri ja muu asiaankuuluva rekvisiitta britannialaiseen tai ranskalaiseen puuttumatta maaston muotoihin tai teihin. Tämä on mahdollistanut näiden maaversioiden verrattain nopean tuotannon.

Rescue: Everyday Heroes on Rescue 2013:n Yhdysvaltoihin suunnattu versio ja nimestä hävinnyt vuosiluku on muutoksista vähäisin. Projekti alkoi toden teolla heinäkuun 2013 lopulla, ja alusta alkaen oli yrityksen sisäisesti tiedossa, että kyseistä versiota pelistä myytäisiin erittäin suositun Steam-latauspalvelun kautta. Tästä syystä, kuten myös yleisestä kunnianhimosta ja halusta hyödyntää täysillä aiemmissa projekteissa haalittua osaamista, USA-versiosta haluttiin tehdä kaikin puolin paras versio pelistä, niin kutsutusti ”Rescue 1.5.” Grafiikan laatua oli määrä nostaa kautta linjan yksityiskohtaisemmilla malleilla ja tekstuureilla, ja aiemmista lokalisaatiopaketeista poiketen useita pelin kenttiä oli määrä rakentaa alusta alkaen uudelleen. Näin saataisiin amerikkalaisista maisemista autenttisimpia ja jälleen kerran näyttävämpiä. Aikaa projektin toteuttamiselle oli aiempien lokalisaatioiden noin kuukaudesta poiketen suunnilleen kaksi ja puoli kuukautta, heinäkuun viimeisestä viikosta lokakuun ensimmäiseen.

Tehtävänimikkeeni yrityksessä oli ollut ensimmäisen Rescuen version julkaisusta lähtien Level Designer eli kenttäsuunnittelija. Olin koonnut sekä Ison-Britannian että Ranskan maakohtaisten versioiden ympäristöjä osana 2-3 kenttäsuunnittelijan ryhmää. Muiden osuus työstä kuitenkin oli vähentynyt tasaisesti heidän siirtyessään uusiin hankkeisiin ja USA-projektista valtaosan olin ainoa kenttäsuunnittelija. Muihin projekteihin oli siirtynyt muutakin henkilöstöä Rescuen parista ja minut mukaan lukien vakituisia USA-

projektin tekijöitä oli lopulta neljä, mikä oli verrattain vähän tiedossa olleeseen työmäärään nähden. Rescuen amerikkalaistaminen oli täten aikaisempaa pidemmästä määräajastaan huolimatta merkittävä haaste aikataulullisesti, mikä edellytti uutta lähestymistapaa projektinhallintaan ja uusien työskentelytapojen kehittämistä. Niin USA-versio kuin aiemmatkin Rescuen versiot on tehty Unity-pelimoottorilla, jota käsitellään omassa luvussaan.

Tämän opinnäytetyön tavoitteena on selvittää, miten Fragment Productionin tulevien hankkeiden toteutusta voidaan parantaa ja tehostaa, ja mitkä työtavat voidaan todeta jo sellaisenaan toimiviksi. Tarkoituksena on koota kattava raportti Rescue U.S. -projektin etenemisestä, siinä tehdyistä ratkaisuista ja ilmenneistä haasteista. Tämän työn on määrä toimia pohjana sisäiseen käyttöön ja mahdollisesti ulkoiseenkin jakeluun tehtävälle niin kutsutulle post mortem -dokumentille.

Aloitan raportoinnin selvityksellä projektiryhmästä ja sen työskentelystä, jatkan selvityksellä käytetyistä työkaluista ja tämän työn ymmärtämiselle tärkeimmistä teknologisista käsitteistä, minkä jälkeen käyn läpi projektissa käytettyjä ja mahdollisesti uudistettuja tekniikoita omissa luvuissaan. Pyrin noudattamaan summittaista kronologiaa aina, kun dokumentin rakenne sen muutoin sallii.

2 PROJEKTIRYHMÄN KOKOONPANO JA TEHTÄVIEN HALLINTA

2.1 Ryhmän jäsenet

Projektiryhmä koostui pääosan ajasta minusta ja kolmesta graafikosta. Muut yrityksen työntekijät auttoivat ajoittain omien osaamisalueidensa tehtävissä kulloisenkin tarpeen ja ehtimisensä mukaan ja hankkeen loppupuolella mukaan liittyi ohjelmoija vastaamaan pelin liittämisestä Steam-palveluun. Projektin valmisteluvaiheessa kaksi graafikkoa oli mallintanut ja teksturoinut pelin ajoneuvoja.

Minun päätehtäväni oli kenttäsuunnittelu, mutta tein lisäksi muita töitä tarpeen ja kykyjeni mukaan. Tämän tekstin kannalta mainitsemisen arvoista on, että ohjelmoin ja muuntelin joitakin skriptejä ja varjostinohjelmia, mistä on luvassa tarkempi selvitys myöhemmissä luvuissa.

Kahden graafikon päätyönä oli mallintaa ja teksturoida kaikkea, mitä tekemiini kenttiin vaadittiin. Yksi graafikko teki käytännössä pelkkää tekstuurityötä, pääasiassa kenttiin tulevia pohjatekstuureja. Tämä pesti vaihtoi noin puolivälissä projektia omistajaa, ja jälkimmäinen työhön astunut henkilö osallistui myös hahmomallinnukseen ja -animaatioon projektin loppuviikoilla.

Koska ryhmä oli verrattain pieni ja työskenteli saman pöydän äärellä, meille syntyi helposti ja nopeasti hyvin tiivis ryhmähenki ja -identiteetti. Tämä auttoi meitä motivoitumaan projektista ja tuntemaan sen todella omaksemme.

2.2 Työn kulku

Kentän tekeminen aloitettiin suunnitelmasta: millainen paikka olisi kyseessä ja mitä siellä olisi. Kentistä oli tulossa jo olemassaolevien korvaajia ja niille oli tulossa tietty määrä tietynlaisia tehtäviä, mikä ohjasi suunnitteluvaihetta. Kenttä, joka oli alunperin kaupunkialue, pysyisi siis kaupunkialueena, mutta olisi silti muuten radikaalisti muuttunut. Suunnittelussa pyrittiin päättämään erilaisten maamerkkien ja muiden avainsijaintien sijoittelu, esimerkiksi pikkukaupungin laidalla nousisi vuoren rinne.

Kun oli selvillä, millainen paikka kentästä olisi tulossa, aloin tehdä sille pohjaksi maastoa Unityn maastotyökalulla, mikä tarkoitti korkeuserojen muotoilemista alunperin tasaiselle pinnalle. Samaan aikaan alkoi pohjatekstuureista vastaavan graafikon työ ja meidän välisemme vuoropuhelu siitä, mitä kumpikin oli tekemässä. Yleinen pähkäilyn aihe oli siinä, miten suunnitelman mukaiset yksityiskohdat sijoiteltaisiin kentälle ja mikä niiden skaala olisi.

Viimeistään tässä vaiheessa, mutta yleensä jo huomattavasti aiemmin, alkoivat mallintajat tuottaa kuhunkin kenttään tarvittavaa rekvisiittaa, kuten taloja, ajoneuvoja, liikennemerkkejä ja kaikkea muuta, mitä realistisessa ympäristössä voisi olettaa näkevänsä. Kun olin saanut kentän pohjan riittävän valmiiksi, aloin sijoitella mallintajilta tullutta materiaalia kentille suunnilleen sitä mukaa kuin sitä tuli. Uusien objektien tarve yleensä kasvoi tässä vaiheessa, kun tajusin jotain oleellista puuttuvan.

2.3 Työn ja tehtävien seuranta

Projektin alusta asti meillä oli Google Drive -palvelussa yhteisesti muokattava taulukko, jossa pidettiin kirjaa kultakin tekijältä tarvittavista asioista ja niiden valmistamiseen kuluvaksi arvioidusta ajasta. Tämä lista muutti jatkuvasti muotoaan, kun asioista luovuttiin tai tarpeet kasvoivat.

Viikottain projektiryhmän ja tuottajan kesken pidettiin palaveri, jossa käytiin läpi, miten ryhmä oli pysynyt aikataulussa ja millä tavoin projektin tarpeet olivat muuttuneet viime tapaamisesta. Tämän lisäksi ryhmän sisäinen kommunikaatio toimi käytännössä kaiken aikaa, sillä istuimme pääosin saman pöydän ääressä. Tämä mahdollisti tarvitessa nopeat suunnanmuutokset ja lyhyet reaktioajat yllättäviin tilanteisiin.

2.4 Merkittävimmät muutokset vanhoista työtavoista

Ennen tätä projektia työtehtävien osoittamiseen ja hallintaan käytettiin Redmine-järjestelmää, joka tarjoaa erittäin kattavat työkalut projektien ja työtehtävien seurantaan ja hallintaan. Se koettiin kuitenkin liian raskaaksi ratkaisuksi tilanteeseen, jossa lähes kaik-

ki asianosaiset olivat saman pöydän ääressä ja pystyivät hoitamaan tarvittavat tehtävien osoitukset ja seurannan yksinkertaisesti puhumalla toisilleen. Käyttämämme Google Drive -taulukko toimi riittävänä työkaluna ryhmän ajanhallintaan ja kommunikointiin ryhmän ulkopuolelle.

Ryhmämme oli aiempaa pienempi, mutta suhteellisesti graafikkopainotteisempi. Tämä oli sinänsä perusteltua, sillä kyse oli etenkin projektin alussa puhtaasti graafisesta sisältöpäivityksestä ilman uusia teknisiä ratkaisuja, joita päädyttiin tekemään myöhemmässä vaiheessa.

Useimmat tässä opinnäytetyössä mainitut uudistukset saivat alkunsa ryhmän jäsenten oma-aloitteisuudesta, eikä niiden keksimiselle tai käyttöönotolle ollut varattu aikaa alkuperäisessä aikataulussa. Koska muutokset kuitenkin paransivat pelin visuaalista laatua huomattavasti, saatiin määräaikaa lykättyä eteenpäin yhteensä noin kuukaudella.

3 TYÖKALUT JA TEKNOLOGIA

3.1 Unity

Unity on pääasiassa 3d-pelien kehittämiseen tarkoitettu pelimoottori ja kehitysohjelmisto, jota käyttää heinäkuun 2013 tietojen mukaan yli kaksi miljoonaa kehittäjää (Unity Technologies 2013e). Se mahdollistaa pelin toiminnallisuuden rakentamisen hyvin pitkällekin valmiista komponenteista graafisessa ympäristössä, mutta tukee myös uusien komponenttien ohjelmoimista C#:illa, JavaScriptillä tai Boo-skriptauskielellä (Unity Technologies 2013d). Pelin sisällössä käytettävien tiedostoformaattien tuki on erittäin kattava (Unity Technologies 2013c).

Unityn editorissa työskennellään kenttätiedostojen parissa, joita kutsutaan Unityssa nimellä scene. Yksi scene sisältää ilmentymiä peliobjekteista, jotka sisältävät komponentteja. Komponentteja voivat olla esimerkiksi graafiset elementit kuten animaatiot, 3d-mallit tai tekstuurit, tai toiminnalliset osat kuten objektin käyttäytymistä hallitsevat skriptit. Kun kerron tässä työssä kenttien muokkaamisesta, tarkoitan näiden scene-tiedostojen käsittelyä Unityn editorityökaluilla.

3.2 Joitakin peruskäsitteitä 3d:stä

3d-malli koostuu vertekseistä, reunoista ja pinnoista. Verteksit ovat pisteitä kolmiulotteisessa avaruudessa. Niiden välille voidaan vetää reunoja (englanniksi edge), jotka ovat yksiulotteisia yhteyksiä verteksien välillä. Kolmen tai useamman reunan välille voidaan luoda polygoni, jolla on pinta (englanniksi face). Verteksillä tai pinnalla on normaali, eli suunta johon se osoittaa. Pinnan voi nähdä tyypillisesti vain siltä puolelta, johon sen normaali osoittaa. Verteksien normaaleja puolestaan käytetään pintojen valaistuksen laskentaan. Kun puhutaan pelkästä 3d-mallin geometriasta, puhutaan polygoniverkosta (englanniksi mesh).

3d-mallin pinnassa on tyypillisesti tekstuuri, eli kuva, joka on piirretty mallin pinnalle UV-kartan mukaisesti. UV-kartta määrittää, mihin kohtiin mallia kuvatiedoston pikselit sijoitetaan. Tekstuuri voi yksinkertaisen pintakuvioinnin lisäksi tai sen sijaan välittää myös muuta pintaan liittyvää informaatiota varjostinohjelmalle.

Varjostinohjelma (englanniksi shader) määrittää, miten näytönohjain lopulta piirtää 3d-mallin tekstuureineen. Varjostinohjelmalla voi tavallisen piirtämisen, valaisun ja varjostamisen lisäksi tuottaa muita tehosteita, kuten luoda optisen vaikutelman monimutkaisemmasta geometriasta, liikuttaa verteksejä tai piirtää mallin pintaan heijastuksia.

Unityssa, kuten tyypillisesti muissakin ohjelmistoissa, varjostinohjelman ja tekstuurin (tai muiden varjostinohjelman vaatimien tai sallimien ominaisuuksien) asettaminen mallille tai mallin osalle tehdään määrittämällä sille materiaali. Materiaali on tiedosto, jossa on viittaukset käytettyyn varjostinohjelmaan ja sen käyttöön annettuihin resursseihin. Objektilla voi olla useita materiaaleja eri osissa mallia tai päällekkäin, jolloin varjostinohjelmissa määritelty, tai määrittelyn puuttuessa Unityn määräämä, piirtojärjestys määrittää, mikä materiaali kulloinkin näkyy. Unityn varjostinohjelmia kirjoitetaan ShaderLab- ja Cg-kielillä (Unify Community Wiki n.d.).

3.3 Piirtokutsut

Piirtokutsu (englanniksi draw call) on grafiikkarajapinnan kautta näytönohjaimelle lähetettävä ohje, johon sisältyy polygoniverkko ja siihen sovellettava varjostinohjelma tarvittavine tekstuureineen. Unityn yhteydessä se tarkoittaa, että piirtokutsu voi koskea vain yhtä materiaalia kerralla.

Piirtokutsut ovat niin sanotusti kalliita, sillä niiden käsittely grafiikkarajapinnassa vaatii joka kerta tietyn määrän työtä tietokoneen suorittimelta (Unity Technologies 2013a). Piirtokutsujen määrän pitämiseksi alhaisena niistä onkin hyvä pyrkiä tekemään mahdollisimman suuria ja kattavia. Unityssa samaa materiaalia käyttävät objektit voi niputtaa samaan kutsuun (englanniksi batching) joko staattisesti editorissa tai ajon aikana dynaamisesti.

4 MODULAARINEN LÄHESTYMISTAPA MALLINTAMISEEN

4.1 Vanhan työtavan ongelmat

Kenttäsuunnittelija pyytää kulloinkin tarvitsemaansa objektia graafikoilta, jotka aikataulunsa salliessa tekevät sen kenttäsuunnittelijan määrittelemään tarpeeseen. Mallintaminen ja teksturointi vie graafikolta aikansa, minkä jälkeen kenttäsuunnittelija saa uuden mallin käyttöönsä. Tämä odotusaika voi olla mallin vaativuudesta riippuen jopa päiviä, joiden aikana kenttäsuunnittelijan pitää keskittyä muihin osiin tekemiään kenttiä. Tämä muodostaa ongelman tapauksissa, joissa odotettava malli on avainasemassa edistymiselle. Tällainen malli voi olla esimerkiksi iso talo, jollaista tarvitsee variaation lisäämiseksi, ja joka pitää saada käyttöön ja kentälle ennen kuin voi nähdä, miten ympäristöä pitää sen mukana ollessa rakentaa.

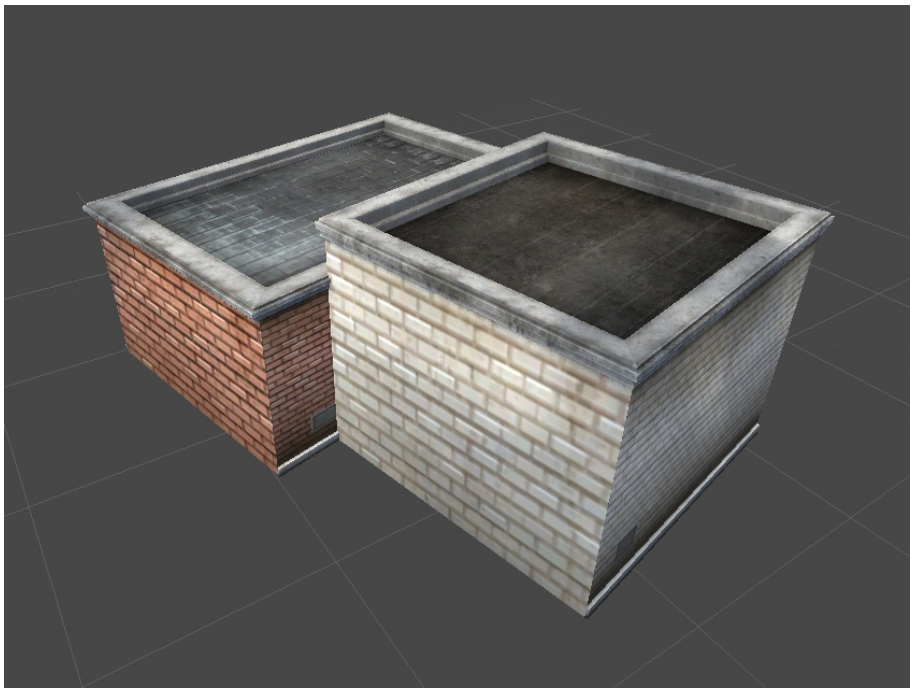
Tämä pullonkaulatilanne toistui ja korostui aiemmissa projekteissa, sillä näkyvän toiston välttämiseksi yhtä mallia ei voinut uudelleenkäyttää kovin usein samassa paikassa. Jokainen uusi talo oli työläs ja hidas tuottaa haluamallamme laadulla, ja kun se oli vihdoin toimitettu kenttäsuunnittelijalle, sille oli käyttöä harmillisen vähän. Mallintamisen aloittaminen hyvissä ajoin ennen muuta työtä auttoi hieman, mutta ennemmin tai myöhemmin kenttäsuunnittelija päätyi kuitenkin odottamaan seuraavan tarvitsemansa mallin valmistumista.

4.2 Uusi lähestymistapa

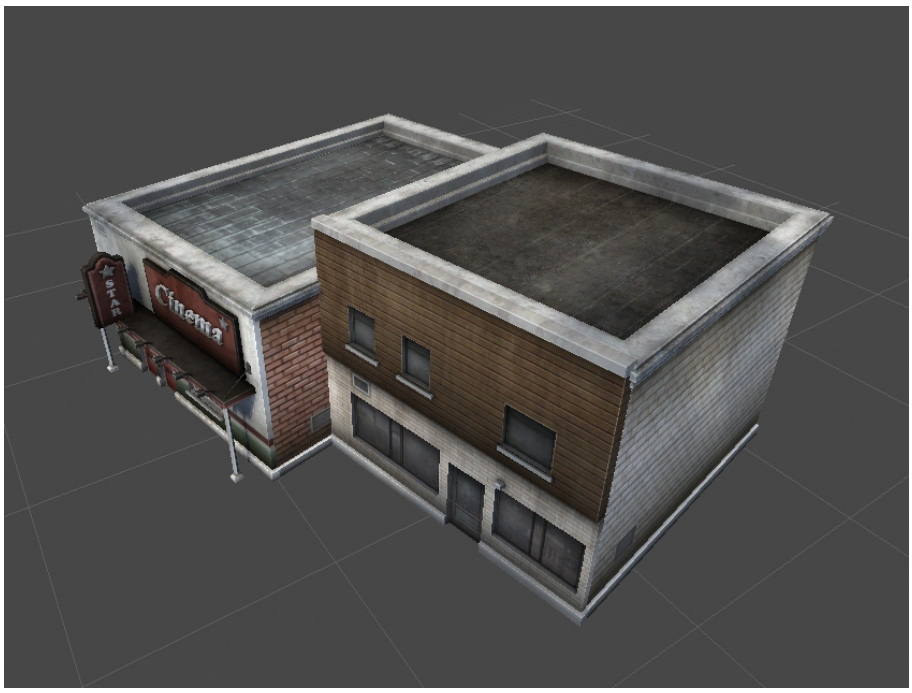
Myös tässä projektissa toimittiin aluksi vanhalla tavalla. Pullonkaulaan valmistauduttiin niin, että projektiin osoitetut graafikot aloittivat amerikkalaistyylisten talojen mallintamisen noin kuukautta ennen muun työn alkua. Tämä auttoi ensimmäisenä aloitetun, kaupunkiteemaisen kentän kokoamisessa huomattavasti, mutta jo sitä tehdessä vanha ongelma alkoi nostaa päätään. Aikataulu todettiin silloisella tahdilla riittämättömäksi, joten kaikilla oli paine keksiä tapoja nopeuttaa prosessia.

Meillä oli jo valmiiksi käytössä markiisien ja mainoskylttien kaltaisia objekteja, joita olin lisännyt taloihin variaation luomiseksi. Niiden ongelmana kuitenkin oli niiden keskittyminen pääasiassa katutason yksityiskohtiin. Pelin kuvakulma on ylhäältä, jolloin varsinkin katoissa tapahtuvan toiston havaitsi erittäin helposti. Tämän tarpeen ilmaistuani toinen mallintavista graafikoistamme teki käyttööni katoille sijoiteltavia ilmastointilaitteita ja sarjan toisiinsa liittyviä ilmastointiputken palasia, joiden mallintamiseen ja teksturointiin kului häneltä yhteensä arviolta puoli päivää. Näillä käänneltävillä, skaalattavilla ja vapaasti toisiinsa liitettävillä palasilla sai kerrostalojen katoista paitsi runsaasti vaihtelevampia, myös mielenkiintoisemman näköisiä.

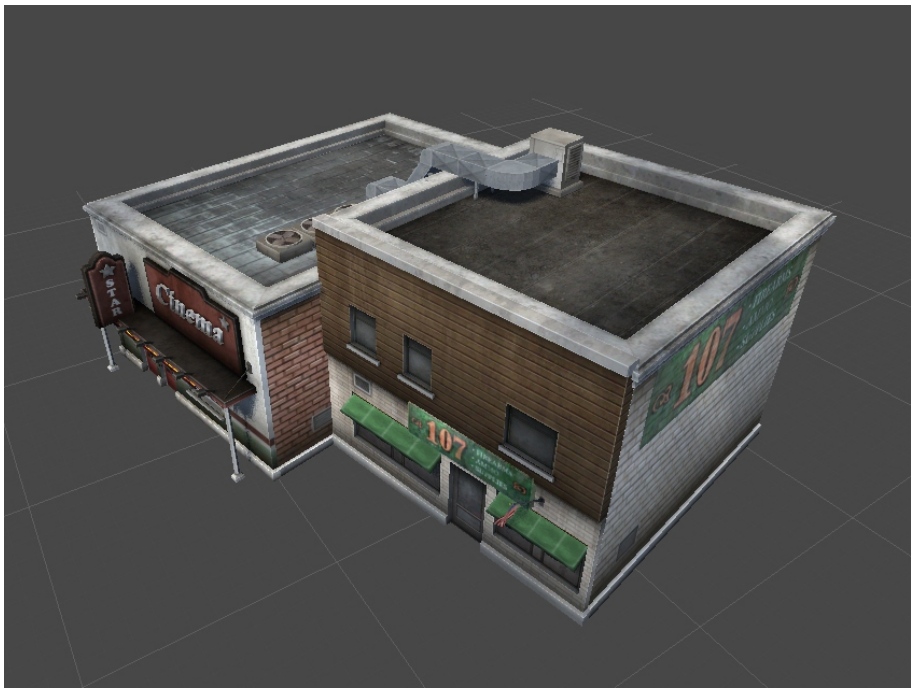
Ideaa vietiin pidemmälle, kun tuli aika tehdä amerikkalaista pikkukaupunkia. Etenkin yhdysvaltalaisen rakentamisen erityispiirteet mahdollistivat modulaarisen lähestymistavan itse taloihin. Graafikot tekivät pikkukaupungin keskustan liikerakennuksia varten vain yksinkertaisen, laatikon mallisen talon, joiden etuseinä oli tyhjä (ks. Kuva 1). Tälle talolle oli kolme vaihtoehtoista tekstuuria. Etuseinän täytteeksi mallinnettiin sarja erilaisia julkisivuja, jotka mitoiltaan ja muodoiltaan sopivat täydellisesti näihin taloihin (ks. Kuva 2). Lisäksi, kuten tähänkin asti, taloihin ripoteltiin pientä rekvisiittaa (ks. Kuva 3).



Kuva 1. Kaksi saman talomallin ilmentymää, joilla on eri skaala ja tekstuuri



Kuva 2. Kuvan 1 talot, joihin on lisätty julkisivut



Kuva 3. Kuvan 2 talot, joihin on lisätty pienempiä kappaleita rekvisiittaa. Myös kattojen välinen ilmastointiputki on koottu modulaarisista osista.

Ilman tätä modulaarisuutta projekti ei olisi tullut valmiiksi määräajassa. Mallinnettavat objektit olivat pienempitöisiä ja monikäyttöisempiä. Vastuu talojen lopullisesta ulkonäöstä ja jopa koosta ja mittasuhteista siirtyi osittain minulle, sillä mitäänsanomattoman näköisten perustalojen rajukin skaalaaminen syvyys- tai korkeussuunnassa oli mahdollista. Varsinainen ulkonäkö taloille tuli julkisivupalasta, katolle asetelluista ilmastointi-

laitteista ja muusta vastaavasta koristelusta. Koska perustalot olivat käytössäni heti, saatoin asetella niitä paikalleen vapaasti ja huolehtia variaatiosta sitten, kun käytössäni oli julkisivupaloja. Minun ei tarvinnut odotella päiväkausia päästäkseni täyttämään kenttää. Pääsin työn pariin hyvin nopeasti, odotusajat lyhenivät, ja ennen kaikkea työn graafinen laatu oli vähintäänkin sama, ellei parempi kuin aikaisemmin.

Pikkukaupungin jälkeen seuraava isotöinen kenttä oli toinen, edellistä laajempi kaupunki, jonka esikuvana toimi Miami. Samalla modulaarisella työtavalla säästettiin runsaasti aikaa myös siinä, ja aiheesta. Määräaika alkoi tuossa vaiheessa jo painaa pahemman kerran päälle, eikä peliä olisi saatu valmiiksi testausta varten, ellei alun perin kenttään suunniteltuja isotöisiä maamerkkejä olisi korvattu suuremmalla määrällä modulaarisesti rakennettuja taloja.

5 PIIRTOKUTSUT JA VARJOT

5.1 Suorituskykyongelma

Rescuen piirtokutsuja oli pidetty kurissa niputtamalla samaa materiaalia käyttävät objektit yhteen piirtokutsuun asettamalla ne Unityn editorissa staattisiksi. Tämä tarkoitti, ettei niille voinut tapahtua ajon aikana mitään muutoksia. Tämä tekniikka toimi, mutta näin yhteen piirtokutsuun niputetuista objekteista langenneet varjot laskettiin edelleen yksitellen. Tämä oli erittäin raskasta tilanteissa, joissa oli useita pieniä objekteja, joista jäi varjo. Käyttämämme Unityn version 4.2 oli määrä korjata tilanne (Unity Technologies 2013f), mutta emme huomanneet mitään parannusta.

Ongelma kärjistyi, kun tein pikkukaupunkikenttään maissipeltoa. Pelto koostui useista erittäin yksinkertaisista malleista, jotka jättivät varjon, ja vaikka ne sai yhdistettyä yhteen piirtokutsuun, ne laskivat määränsä verran varjoja, jotka laskettiin erikseen. Tämä tarkoitti laskettavien varjojen määrän nousua joistakin kymmenistä useisiin satoihin, kun pelaaja näki maissipellon. Se ei käynyt päinsä.

5.2 Objektien yhdistäminen

Avuksi otettiin Unitysta valmiiksi löytyvä Combine Children -skripti, joka yhdistää ajon aikana isäntäobjektinsa alla olevat saman materiaalin jakavat mallit yhdeksi polygoniverkoksi, joka siten laskee vain yhden varjon. Tätä kokeiltiin ensimmäiseksi maissipeltoon.

Combine Children osoittautui hieman hankalaksi ottaa käyttöön. Koska se teki objekteille rajuja muutoksia ajon aikana, ne eivät saaneet olla staattisia, mikä oli juuri päinvastoin, kuin mihin olin tottunut. Kun tätä skriptiä halusi jossain käyttää, piti staattisuus muistaa kytkeä kaikista yhdistettävistä objekteista pois. Muutoin objektit hävisivät näkyvistä ja skripti antoi virheilmoituksen, josta ei ilmennyt mistä oli kyse.

Unityn yläraja vertekseille yhdessä objektissa on 65 000. Jos yhdistettävissä objekteissa oli yhteensä enemmän verteksejä, Combine Children -skripti hävitti ne kaikki ja antoi virheilmoituksen. Tämä tarkoitti, että suuret kokonaisuudet samaa materiaalia käyttäviä objekteja, kuten maissipelto tai metsä, piti jakaa pienemmiksi, alle 65 000 verteksiä sisältäviksi ryhmiksi. Tämä vaati aikaa ja laskemista.

Kun yhdistäminen toimi, parannus oli kuitenkin huikea. Kaikkialla, missä oli esimerkiksi pitkiä aitoja tai edellisessä luvussa käsiteltyjä modulaarisia koristeita, yksittäin laskettujen varjojen määrä väheni murto-osaan ja peli pyöri huomattavasti sujuvammin. Skriptin käyttäminen oli jokseenkin kömpelöä ja aikaavievää, mutta sillä saavutettu hyöty oli vaivan arvoinen.

6 SIIRTOKUVAT TEKSTUUREJA TÄYDENTÄMÄSSÄ

6.1 Siirtokuvat

Projektin alkuvaiheilla projektissa otettiin käyttöön erään Unityn laajennuksen tarjoama tuki siirtokuville (englanniksi decal). Tämä järjestelmä antoi projisoida kaksiulotteisia tekstuureja kolmiulotteisille pinnoille suoraan Unityn editorissa, mikä mahdollisti minulle seinämaalausten, graffitien ja mainosten kaltaisten kaksiulotteisten yksityiskohtien vapaan ripottelun pitkin kenttien pintoja.

Siirtokuvalaajennuksen käyttöliittymässä voi rajata samasta tekstuurista eri kokoisia ja mallisia kappaleita pinnoille projisoitaviksi. Tämä mahdollistaa laajan määrän erilaisia siirtokuvia yhdellä materiaalilla, mikä auttaa pitämään piirtokutsujen määrän matalana. Järjestelmä tekee polygoniverkon alueista, joille kuvat on projisoitu.

6.2 Kallioseinämän parannus

Pikkukaupunkikentän reunalla nousee vuoren rinne. Käyttämällämme Unityn maastotyökalulla ei kuitenkaan voinut tehdä järin hyvän näköisiä kallioseinämiä, vaan ne olivat luonnottoman sileitä, kuten kuvassa 4.

Kuva 4. Kallioseinämä ennen lisäkäsittelyä

Meillä ei ollut aikaa tai työntekijöitä mallintamaan hyvän näköistä kalliota, joten ongelma piti joko ratkaista jollain kekseliäällä tavalla tai jättää sikseen. Aloin tutkia siirtokuvien mahdollisuuksia ja sain erinomaisia tuloksia. Täytin rinnettä eri päin käännetyillä ja skaalatuilla kopioilla projektissa iät ajat ollesta kivestä, jonka väri itsessään oli liian vaalea ympäristöön (ks. Kuva 5).

Kuva 5. Kopioiduilla kivillä täytetty rinne

Lisäsin mukaan siirtokuvajärjestelmän, johon tein hieman läpinäkyvän materiaalin, jossa oli sopiva kalliotekstuuri. Asetin sen projisoitumaan pelkästään asettamiini kiviin ja aloin asetella sillä suuria siirtokuvia kaikkien kivien peitoksi. Kuten kuvasta 6 voi tarkasti katsoen nähdä, siirtokuvan tekstuuri yhdessä siitä läpi näkyvän kiven oman pinnan kanssa vähentää pintakuvion toistoa ja istuttaa kivet ympäristöönsä erittäin hyvin.

Kuva 6. Siirtokuva lisättynä kuvion 5 kiviin

7 VALAISTUSUUDISTUS

7.1 Puutteelliset valaistusasetukset

Toinen mallintavista graafikoista pani projektin ensimmäisen kuukauden aikana merkille, että pelin silloinen valaistus oli erittäin vaatimattoman näköistä, minkä todettiin johtuvan luultavimmin kiireen vuoksi huonosti säädetyistä valoista. Joka kentän valaistuksessa kaikkia pintoja tasaisesti valaiseva ambient-valo oli liian kirkas verrattuna auringonvaloa matkivan suunnatun valonlähteen voimakkuuteen. Tämän takia missään ei näkynyt selkeitä varjoja ja esineiden pinnanmuodot ja kolmiulotteisuus hävisivät tässä ilmeettömässä valossa (ks. Kuva 7).

Kuva 7. Esimerkki vanhasta valaistuksesta

Nopealla kokeilulla Unityssa saimme valaistuksesta huomattavasti miellyttävämmän näköisen. Valitettavasti pelin tehtävien omat asetukset usein korvasivat kenttäkohtaisen valaistuksen pelin ollessa käynnissä. Näille tehtäväkohtaisille asetuksille emme aluksi voineet tehdä mitään, sillä samoja asetuksia käyttivät myös muiden kieliversioiden tehtävät, jotka olivat samassa Unity-projektissa. Jos tehtäväkohtaisia valoja olisi menty muuttamaan tuolloin, olisi se vaikuttanut myös alkuperäiseen saksalaseen versioon pe-

listä, kuten myös Ison-Britannian ja Ranskan lokalisaatioversioihin. Meillä ei ollut aikaa käydä kaikkia lokalisaatioversioita tehtävistä läpi, joten emme ottaneet riskiä, että jokin niistä olisi näyttänyt poikkeuksellisen huonolta omien säätöjemme jälkeen.

Myöhemmässä vaiheessa projektin ulkopuolelta hetkelliseksi viimeistelyavuksi tullut ohjelmoija teki meille USA-versiota varten oman kopion tehtävien valoasetuksista, jolloin tehtäväkohtaiset korjaukset voitiin tehdä. Lopputuloksen voi nähdä kuvasta 8. Kun valokorjaukset vihdoin pääsi tekemään, niihin kului minulta alle päivä. Koko projektiryhmän mielipide oli, että tämä nimenomainen muutos nuorensi pelin ulkoasua viidellä vuodella.

Kuva 8. Kuvion 7 näkymä uusilla valoasetuksilla

7.2 Piirtojärjestysongelma

Peliä testatessa huomattiin valaistusuudistuksen aiheuttavan merkittävän graafisen ongelman: valaistus ei laskenut varjoja pelaajan palomiehiin tai paloautoihin. Syyksi paljastui skripti, joka pakotti ne piirrettäväksi varjojen jälkeen. Tätä virhettä ei oltu havaittu ennen, koska vanha, vaisu valaistus esti näkemästä varjoja missään.

Piirtojärjestystä oli muutettu hyvästä syystä. Pelaajan yksiköissä oli varjostinohjelma, joka näytti niistä väritetyn siluetin, kun ne olivat seinien takana. Edellä mainittu piirtojärjestyksen muutos piti huolen, että siluettia ei näkynyt niissä osissa hahmoja, jotka olivat näkyvissä. Tämä todettiin yksissä tuumin huonoksi tavaksi toimia, joten asia pyrittiin korjaamaan.

Erittäin puutteellinen tietämys Unityn käyttämästä Shaderlab-kielestä tai varjostinohjelmista yleisesti teki vian korjaamisesta hankalaa. Yhden iltapäivän painimisen ja yksien yöunien jälkeisen oivalluksen tuloksena kuitenkin onnistuin siinä. Ratkaisu oli muuttaa varjostinohjelman syvyystestauksen Offset -arvoparia, jolla voidaan pakottaa piirrettäviä pintoja toisten taakse syvyyssuunnassa, vaikka niiden syvyys olisi sama (Unity Technologies 2013b). Näillä arvoilla saatiin siluetti näkymään vain ollessaan itsestään riittävän etäällä olevien asioiden takana. Yrityksen ja erehdyksen kautta päädyttiin siihen pisteeseen, että siluetit näkyivät jokseenkin oikein tärkeimmissä tilanteissa.

Ongelma esiintyi uudelleen, kun siluettia alkoi näkyä läpi yksiköistä, jotka olivat korkeassa maastossa ja siten lähempänä kameraa. Ratkaisuksi osoittautui pienentää kameran ”near plane” -parametria, joka määrittää miten läheltä kameraa asioiden piirtäminen alkaa. Minulle on valitettavasti yhä arvoitus, miksi tämä toimi, sillä tuossa vaiheessa projektia ei ollut aikaa jäädä tutkimaan jo toimivaa ratkaisua.

8 UUSI VESITEHOSTE

8.1 Vanha vesi

Vesi toteutettiin Rescuessa asettamalla yksinkertaiselle tasaiselle pinnalle materiaali, joka näytti vedeltä. Kenttiä, joissa oli vettä, oli pelissä ennen USA-versiota neljä, ja USA-versiossa kaksi lisää. Kentissä olevalla vedellä ei ollut pelillistä arvoa muuten kuin korkeintaan kulkuesteenä.

Käytössä oli Unityn mukana tullut sinänsä monipuolinen vesimateriaali, josta kuitenkin puuttui kiiltotehoste (englanniksi specular). Tästä juontunut kimmellyksen puute ilmeni uusilla valaistusasetuksilla häiritsevästi. Unityn vesimateriaali ei myöskään tarjonnut mahdollisuuksia hallita veden virtausta sijainnin mukaan, mikä sai sen näyttämään hyvin tökeröltä esimerkiksi mutkittelevassa joessa. Tämä ongelma oli jo vanha tuttu, mutta siihen oli tyydytty. Tällä kertaa asialle kuitenkin haluttiin tehdä jotain, sillä ulkoasusta haluttiin paras mahdollinen.

8.2 Uusi varjostinohjelma ja virtauskartta

Otimme käyttöön varjostinohjelman, joka käyttää virtauskarttaa (englanniksi flow map). Virtauskartta on tekstuuri, jonka pikselien väriarvot määrittävät virtauksen suunnan (Vlachos, A. 2010). Käyttämässämme ratkaisussa punainen värikanava ohjasi liikettä vaakasuunnassa, eli x-akselilla, ja vihreä liikettä pystysuunnassa, eli y-akselilla. Varjostinohjelmassa oli myös kaipaamamme kiilto, mutta menetimme veden aiheuttaman vääristymän sen läpi näkyvissä paikoissa. Saavutetut edut todettiin kuitenkin merkittävämmiksi.

Virtauskarttojen piirtäminen käsin on erittäin vaikeaa, mutta onneksemme löysimme niiden tekemiseen ohjelman. Sillä tehtiin virtauskartat kahteen pelin kenttään, joissa oli joki.

8.3 Aallot ja vaahto

Toisella kentistä, joilla virtauskarttoja käytettiin, oli meren ranta, ja halusimme saada aallot lyömään rantaan ja vaahtoamaan. Käyttämästämme varjostinohjelmasta oli olemassa versio, jonka oli tarkoitus lisätä vaahtoa hitaasti virtaaviin kohtiin, mutta emme saaneet sitä toimimaan tyydyttävästi, vaan vaahtoa tuli joka paikkaan. Malli, jolla vesimateriaalimme oli, oli kahdesta kolmiosta koostuva taso, joten mahdollisuutta tehdä aaltoja sen geometriaan ei ollut, ja vaikka olisi ollutkin, emme olisi välttämättä osanneet sitä tehdä.

Silkasta mielenkiinnosta tutkin varjostinohjelman koodia ja sain siitä jopa jonkin verran selvää. Kantapään kautta opiskelemalla sain kurittoman vaahdon muodostumaan hallitummin hitaasti virtaaviin paikkoihin.

Luodakseni vaikutelman aalloista ohjelmoin C#-skriptin (ks. Kuvio 1), joka keinuttaa objektia, johon se on kiinnitetty ja liikuttaa sen sijaintia. Keinumisen ja liikkeen suunta, määrä ja nopeus on käyttäjän päätettävissä. Samassa luokassa muokataan myös varjostinohjelmaan lisäämäni, hieman harhaanjohtavasti nimeämäni foaminess-muuttujan arvoa. Tämä muuttuja on liukulukuarvo, jonka perusteella varjostinohjelma annostelee veden vaahto-osien hajoamista. Hajoamisen olen toteuttanut kääntämällä pinnan normaalin paikoissa, joissa määrittelemäni ehto täyttyy. Tämä tekee pintaan täysin läpinäkyviä reikiä.

```
using UnityEngine;
using System.Collections;

/// <summary>
/// Foamy water system.
///
/// Optional controller script for faking waves and foam behaviour with the Foamy Water shader.
/// Can also be used to just make things in water swing and bop.
/// </summary>
public class FoamyWaterSystem : MonoBehaviour {

    public Vector3 movement;
    public Quaternion rotation;
    public CurveType curveType;
    public float speed;

    private Transform trans;
    private Vector3 originalPosition;
    private Quaternion originalRotation;
    private float timer = 0;
    private float change = 0;
    private int direction = 1;
    private float foaminess = 0;

    private bool foamFound = false;

    void Start () {
        trans = gameObject.transform;
        originalPosition = trans.localPosition;
        originalRotation = trans.localRotation;

        if( gameObject.renderer != null && gameObject.renderer.material.GetFloat("_Foaminess") != null)
            foamFound = true;
    }

    // Update is called once per frame
    void Update () {

        timer += Time.deltaTime * speed * direction;


        change = SimpleEasing.GetValueAtPoint( Mathf.Clamp01( timer ), curveType );

        trans.localPosition = originalPosition + movement * Mathf.Clamp01(change);
        trans.localRotation = Quaternion.RotateTowards(originalRotation, rotation, change);

        if(foamFound && direction < 0)
        {
            foaminess += Time.deltaTime * speed * 2;
            foaminess = Mathf.Clamp01(foaminess);
            gameObject.renderer.material.SetFloat("_Foaminess", foaminess);
        }
        else if(foamFound)
        {
            foaminess -= Time.deltaTime * speed * 3;
            foaminess = Mathf.Clamp01(foaminess);
            gameObject.renderer.material.SetFloat("_Foaminess", foaminess);
        }


        if( timer >= 1.0f || timer <= 0 )
        {
            direction *= -1;
            rotation = Quaternion.Inverse(rotation);
            timer = Mathf.Clamp01(timer);
        }

    }
}
```

Kuvio 1. FoamyWaterSystem-luokka

Keinumisen ja liikkumisen pehmentämiseksi käytin toista ohjelmoimaani luokkaa nimeltä SimpleEasing (ks. Kuvio 2), joka laskee normalisoitujen liukulukuarvojen (englanniksi floating point number, float) funktioita. Sen tarkoituksena on olla äärimmäisen

nopea, joten jätin siitä pois esimerkiksi tarkastukset, onko funktiolle annettu liukuluku ylipäänsä normalisoitu, eikä luokan sisältä kutsuta sen ulkopuolisia metodeja. En ole matemaatikko, joten haluamanilaisten funktioiden aikaansaaminen oli pitkälti arvailun ja kokeilun tulosta, enkä ole varma kutsunko asioita edes oikeilla nimillä, mutta koodini toimii ja vaatii suoritinaikaa erittäin vähän.

```
using UnityEngine;
using System.Collections;

public enum CurveType
{
    ExpSym, ExpSymInv, Exp, ExpInv, Linear
};

/// <summary>
/// Very simple easing curve tool class for normalized float values.
/// Methods return f(x) on different curves.
/// </summary>
public class SimpleEasing
{
    public static float GetValueAtPoint(float x, CurveType type)
    {
        switch(type)
        {
        case CurveType.ExpSym:      // A smooth curve with "slow" beginning and end
            return x * x / ( 2 * x * x - 2 * x + 1 );
        case CurveType.ExpSymInv:   // A smooth curve with steep beginning and end
            return 2 * x * x * x - 3 * x * x + 2 * x;
        case CurveType.Exp:
            return x * x;
        case CurveType.ExpInv:      // Steep rise at start
            return 2 * x - x * x;
        case CurveType.Linear:      // For silly people (or to blend with others, obv.)
            return x;
        default:
            return x;
        }
    }
```

Kuvio 2. SimpleEasing-luokka ja sitä tukeva enumeraatio CurveType

Keinutteluskriptiä käytettiin paitsi vedessä, myös siinä kelluvissa veneissä ja poijuissa. Vaahto oli erittäin tarpeellinen lisä rantamaisemaan ja lopputulosta voi ihastella kuvassa 9. Varjostinohjelman muokkaaminen olisi luultavasti käynyt nopeammin ja paremminkin tuloksin joltain niihin aiemmin perehtyneeltä, mutta sellaista henkilöä ei ollut tuohon aikaan saatavilla.

Kuva 9. Uusi vesi vaahtoineen

9 POHDINTA

USA-Rescuen tekeminen oli minulle, kuten lähes kaikille osallisille, valtava oppimiskokemus, jossa sai niin kutsutusti ison hatun päähänsä. Oli yhtä lailla kunnia kuin taakkakin saada vastuulleen tehdä firman esikoispelistä parasta ja laajimmin levitettävää versiota.

Kommunikaation merkitys tuli selväksi. En usko, että puoliakaan mainitsemistani uudistuksista olisi voitu tai ehditty tehdä, elleivät kaikki olisi olleet jatkuvasti perillä toistensa tekemisistä ja antaneet täyden panoksensa osaamisensa mukaan. Tehtävien seuranta ja suunnittelu auttoi työajan ja -määrän hallinnassa. Tehtäviä osattiin myös poistaa listalta, kun määräajat alkoivat lähestyä.

Projekti oli sikäli erikoislaatuinen, että sen tekninen laajuus ei ollut lainkaan tiedossa aloitettaessa, ja lienee mahdollista spekuloida loputtomiin, miten työskentelyn luonne olisi muuttunut, jos kaikki uudet ominaisuudet olisi suunniteltu tehtäviksi alusta alkaen. Projektin venymisestä ei tullut lisäongelmia tällä kertaa, mutta sitä pitäisi aina välttää. Olisikin parasta, että hankkeita aikataulutettaessa ja budjetoitaessa voitaisiin ottaa huomioon tällainen odottamatta tapahtuva oma-aloitteinen tutkimus- ja kehitystyö.

Luultavasti tulevaisuuden kannalta merkittävin yksittäinen oivallus USA-projektissa oli modulaarisuus. Koska projektiryhmien koot pysynevät lähitulevaisuudessakin pieninä, kyky tuottaa enemmän ja nopeammin käyttökelpoista sisältöä ja jakaa työtä tasaisemmin sen eri vaiheille muodostuu elintärkeäksi. On myös missä tahansa tilanteessa eduksi, jos ryhmät voi tällaisin keinoin pitääkin pieninä, sillä kuten edellä on jo mainittu, nopea kommunikaatio on kullanarvoista. Modulaarisuutta taatusti kehitetään ja hyödynnetään entistä monipuolisemmin ja suunnitelmallisemmin tulevissa hankkeissa.

Pelin yleiseen ulkoasuun eniten vaikuttava tekijä oli puolestaan valaistus – sen parantaminen ensimmäistä kertaa teki toimiston väkeen välittömästi niin syvän vaikutuksen, että uusia ruutukaappauksia pelistä alettiin pitkästä aikaa laittaa firman Facebook-sivulle. Valaistuksen kanssa käyty yllättävän pitkällinen paini oli myös varsin opettavaista ja osaltaan muistutti, miten paljon yllättäviä käänteitä näennäisen yksinkertaisissakin toimenpiteissä voi ilmetä.

LÄHTEET

Unify Community Wiki, n.d. Getting Started with Shaders. Luettu 28.11.2013. http://wiki.unity3d.com/index.php/Getting_Started_with_Shaders

Unity Technologies, 2013a. Draw Call Batching. Luettu 28.11.2013. http://docs.unity3d.com/Documentation/Manual/DrawCallBatching.html

Unity Technologies, 2013b. ShaderLab syntax: Culling & Depth Testing. Luettu 28.11.2013. http://docs.unity3d.com/Documentation/Components/SL-CullAndDepth.html

Unity Technologies, 2013c. The market-leading import pipeline. Luettu 27.11.2013. http://unity3d.com/unity/workflow/integrated-editor

Unity Technologies, 2013d. Unity Scripting. Luettu 27.11.2013. http://unity3d.com/unity/workflow/scripting

Unity Technologies, 2013e. Unity Technologies Doubles Community to Two Million Developers. Luettu 27.11.2013. http://unity3d.com/company/public-relations/news/unity-technologies-doubles-community-two-million-developers

Unity Technologies, 2013f. What's New in Unity 4.2. Luettu 2.12.2013. http://unity3d.com/unity/whats-new/unity-4.2

Vlachos, A. 2010. Water Flow in Portal 2. http://www.valvesoftware.com/publications/2010/siggraph2010_vlachos_waterflow.pdf