KARELIA-AMMATTIKORKEAKOULU
Tietojenkäsittelyn koulutusohjelma

Joonas Rauha
Santeri Honkanen

PELIN KÄÄNTÄMINEN SONYN PLAYSTATION VITALLE

Opinnäytetyö
Toukokuu 2016

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Toukokuu 2016** **Tietojenkäsittelyn koulutusohjelma** Karjalankatu 3 80200 JOENSUU (013) 260 600

Tekijä(t)
Joonas Rauha & Santeri Honkanen

Nimeke
Pelin kääntäminen Sonyn Playstation Vitalle

Toimeksiantaja
Polar Bunny Ltd

Tiivistelmä

Tässä opinnäytetyössä käsitellään pelien käännöstyötä kohdealustalta toiselle. Keskeisenä tavoitteena oli tutkia käännöstyöstä aiheutuvia haasteita. Opinnäytetyössä käännettiin Trivasion-niminen peli Applen iOS-alustalta yhteensopivaksi Sonyn Playstation Vitalle. Trivasion on Unity-pelimoottorilla toteutettu videopeli, joka alun perin toteutettiin yhteensopivaksi iOS:n kanssa.

Opinnäytetyössä analysoidaan käännöstyöhön vaikuttaneita tekijöitä, siinä käytettyjä työkaluja sekä käännöstyön laajuutta. Työssä vertaillaan tehtyjä ratkaisuja muihin vastaavan tyyppisiin käännöstöihin ja pohditaan mahdollisesti eriävien ratkaisujen syitä. Kohdelaitteen ja monialustatuen omaavan pelimoottorin merkitystä käännöstyölle korostetaan.

Opinnäytetyötä varten tehty Trivasionin käännöstyö osoitti, että samanlaisen pelituntuman toteuttamiseksi täytyy mahdollisesti tehdä suuriakin muutoksia pelin tekniseen toteutukseen. Käännöstyön huolellinen analysointi tarjoaa tietoa käännöstyön tärkeimmistä seikoista sekä Unity-pelimoottorin tarjoamista eduista käännöstyölle.

Kieli suomi	Sivuja 58 Liitteet 0

Asiasanat
Pelinkehitys, käännöstyö, pelin kääntäminen, Unity3D, Playstation Vita

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
May 2016
Degree Programme In Business Information Technology

Karjalankatu 3
80200 JOENSUU
FINLAND
358-13-260 600

Author (s)
Joonas Rauha & Santeri Honkanen

Title
Porting a Game to Sony Playstation Vita

Commissioned by
Polar Bunny Ltd

Abstract

This thesis covers the porting process of video games from one target platform to another. The specific objective was to examine the challenges arising from the porting work. In this thesis a video game named Trivasion was ported from Apple's iOS platform to Sony's Playstation Vita, Trivasion being a video game developed with Unity game engine which was originally aimed for iOS.

This thesis analyzes the factors which contributed to the porting work. In addition, also tools used and and the extent of the porting process are analyzed. This is achieved by finding similar approaches and studying reasons behind various solutions.

Both the significance of a cross-platform game engine and target platform are emphasized. The porting process of Trivasion revealed that in order to achieve a similar feeling in game, one might have to make drastic changes in the technical implementation of the game. Thorough analysis of the porting process offers insight in most important factors of the porting process as well as significance of the Unity game engine for game porting.

Language
Finnish

Pages 58
Appendices 0

Keywords
Game development, porting, porting a game, Unity3D, Playstation Vita

Sisältö

Käsitteet

DirectX — on Microsoftin Windows-käyttöjärjestelmälle ja Xbox-pelikonsoleille kehittämä, erityisesti peleihin tarkoitettu ohjelmointirajapinta tietokoneohjelman ja laitteiston välille. DirectX tarjoaa yhtenäisen rajapinnan mm. 3D-grafiikkaa, ääntä ja ohjauslaitteita varten. (Microsoft 2016a)

DrawCall — kun Unity3D piirtää objektin ruudulle, täytyy pelimoottorin tehdä piirtokutsu (draw call) grafiikka API:lle (esim. OpenGL tai Direct3D). Piirtokutsut ovat usein raskaita prosessorille, joten usein optimointitilanteissa keskitytään vähentämään niiden määrää. (Unity Technologies 2016a)

Fysiikkamoottori — (engl. Physics engine) on tietokonepelien osana hyödynnetty ohjelmisto, joka simuloi kappaleiden putoamista, törmäyksiä ja muuta vuorovaikutusta.

Globaali valaistusjärjestelmä — (engl. Global illumination tai "GI") tai epäsuora valaistus on yleinen nimitys ryhmälle algoritmeja 3D-grafiikan luomisessa, sen tarkoitus on luoda realistisempi valaistus ympäristöihin. Algoritmit ottavat huomioon paitsi valon, joka tulee suoraan valonlähteestä, myös valon käyttäytymisen, kun se osuu muihin materiaaleihin riippumatta niiden heijastavuudesta.

Gameplay loop — on kollektiivinen joukko toimia, joita pelaaja tulee tekemään tietyn ajan kuluessa peliä pelatessaan.

HDR-renderöinti (engl. High-dynamic-range rendering, high-dynamic-range lighting, HDR rendering tai "HDRR") on tietokonegrafiikan piirtämiseen tarkoitettu valaistustekniikka. Se toteuttaa laskutoimituksia laajemman dynaamisen arvoalueen sisällä, mikä auttaa yksityiskohtien kontrastisuhteen säilyttämisessä tietyissä tilanteissa.

Kehityslaitteisto (engl. Software development kit, SDK tai "devkit") on joukko ohjelmiston kehitykseen tarvittavia työkaluja, jotka on yleensä räätälöity jonkin tietyn alustan kehitystarpeisiin. Se voi sisältää ohjelmistoja ja laitteistoa.

Pelimoottori (engl. Game engine tai "engine") on videopelin ohjelmistokehys, jonka päälle pelinkehittäjät voivat rakentaa pelejä.

Koontiversio (engl. Software build tai "build") on termi, jolla tarkoitetaan jostakin ohjelmistosta koostettua rakennelmaa, jolla on haluttu toiminto. Yleensä lähdekoodista kasataan itsenäinen ohjelmisto, jota pyöritetään halutulla laitteella.

Level of Detail tai "LoD" on tietokonegrafiikan optimointiin tarkoitettu tekniikka, jonka avulla objektien yksityiskohtaisuutta lasketaan tiettyjen parametrien mukaan. Jos esimerkiksi jokin renderöitävä asia on todella kaukana, ei sitä tarvitse renderöidä sen täydellä tarkkuudella, koska katsoja ei huomaa yksityiskohtien määrässä eroa.

Maailmakoordinaatisto on pelimoottoreissa usein käytetty karteesinen koordinaatisto, jolla kuvataan asioiden sijaintia 3D tilassa x-, y- ja z-akselien arvojen perusteella.

Mecanim-väliohjelmisto on Unityssa käytetty animaatiotyökalu graafisella käyttöliittymällä (Unity Technologies 2016b).

Meshi (engl. Polygon mesh tai "mesh") on kokoelma verteksejä, reunoja, sivuja ja polygoneja, jotka määrittävät esim. pelihahmon muodon.

NovodeX on fysiikkamoottori, jonka Ageia osti vuonna 2004.

Networking-väliohjelmisto (engl. Networking middleware) on pelien sisäisestä tiedonsiirrosta huolehtiva ohjelmisto. Se voi huolehtia esim. kahden eri pelaajan pelitilannetietojen lähettämisestä koneiden välillä, jolloin he pystyvät pelaamaan samaa pelitilannetta samanaikaisesti.

Occulsion culling on tekniikka, joka mahdollistaa objektien renderöimättä jättämisen, jos ne ovat toisten objektien takana näkymättömissä. Tämä on nykyään varsin suosittu optimointimenetelmä suurien kohtauksien renderöintiin, jotka soveltuvat varsinkin alueisiin jotka ovat monimutkaisia syvyys-akselilla.

OpenGL (engl. Open Graphics Library) on ohjelmointirajapinta interaktiivisen tietokonegrafiikan tuottamiseen (OpenGL 2016).

Polygoni (engl. Polygon tai "poly") on monikulmio, jonka jokaisessa kulmassa on verteksi.

Powerup on pelissä oleva toiminnallisuus, joka on usein jonkin kerättävän asian muodossa. Kun pelaaja kerää kyseisen asian, saa hän yleensä keräyshetkellä hyötyä itselleen esim. lisäkykyjen tai pisteiden muodossa.

Piirtoetäisyys	(engl. Draw distance, render distance tai view distance) on termi, joka määrittelee suurimman etäisyyden, jolle asioita renderöidään 3D-tilassa.
Pikseli	(engl. Pixel) on fyysinen piste rasterikuvassa tai pienin digitaalisen kuvan luova elementti, jota voidaan kontrolloida.
PhysX	on entinen fysiikkamoottori, joka tunnettiin NovodeX-nimellä. Kun Nvidia osti Ageian vuonna 2008, sai se nimen PhysX (nVidia 2008).
Reunanpehmennys	(engl. Anti-aliasing) on tekniikka, jota käytetään realismin lisäämiseksi digitaaliseen kuvaan tasoittamalla rosoisia reunoja, kaarevia linjoja ja vinoviivoja.
Renderöinti	(engl. Rendering) on kuvan luomista tietokoneellisesti. Se on ohjelmallinen kuvaus tiedosta, joka voi koostua esim. geometriasta, tekstuurista, valaistuksesta ja niiden katselukulmasta.
Reitinhaku	(engl. Pathfinding tai "pathing") on peleissä käytetty tekoälyn osa, joka pitää huolen pelimaailmassa olevien objektien tavasta liikkua.
Ruudunpäivitysnopeus	(engl. Frate rate, frame per second tai "fps") on taajuus tai määrä (yleensä sekunnin aikana), jolla laite renderöi peräkkäisiä kuvia ruudulle.
Shader	on ohjelma, joka kertoo tietokoneelle, kuinka jokin asia piirretään tietynlaisella ja uniikilla tavalla. Sillä voidaan esim. tasoittaa yksinkertaisten meshien varjostusta tai luoda koko visuaalisen ilmeen muokkaavia tehosteita.

Sprite	on kaksiulotteinen bittikartta, jota käytetään osana pelin graafista ulkoasua.
Splash screen	on graafinen elementti, joka näytetään usein pelin käynnistyessä. Kyseinen ikkuna voi sisältää esim. pelin tehneen tiimin nimen, tai alustan logon jolla peliä pelataan.
Tarkistuslista	(engl. Checklist) on lista, jossa määritellään julkaisijan tai laitevalmistajan vaatimukset tuotteelle, ennen kuin se voi siirtyä eteenpäin julkaisuprosessissa.
Terrain-moottori	(engl. Terrain engine) on pelimoottorin osa, jonka avulla pelinkehittäjät voivat rakentaa peleissä esiintyviä maastoja.
Verteksi	(engl. Vertex) on tietorakenne, joka kuvaa esimerkiksi paikkaa 3D-tilassa.
Vektori	(engl Vector) on välimatka kahden pisteen välillä maailmakoordinaatistossa.

1 Johdanto

Tässä opinnäytetyössä tutkitaan videopelien käännöstyöhön liittyviä haasteita sekä mahdollisuuksia. Opinnäytetyössä käännettiin Trivasion-niminen videopeli Applen iOS-alustalta Sonyn Playstation Vitalle. Käännöstyön vaiheita sekä siihen käytettyjä työkaluja analysoidaan. Myös käännöstyön laajuus on tarkastelun alaisena. Trivasion on Unity-pelimoottorilla toteutettu videopeli.

Opinnäytetyössämme tarkastelemme projektia laajasti sen koko keston ajalta. Keskitymme Playstation Vitan alustalähtöiseen suunnitteluun, ja sen tuomiin mahdollisuuksiin ja haasteisiin käännösprojektissa. Vastasimme pääasiallisesti projektin teknisestä toteutuksesta, joten keskitymme lähinnä siihen tässä työssä. Pyrimme myös vastaamaan seuraaviin tälle työlle asetettuihin tutkimuskysymyksiin: Kuinka Unity-pelimoottori vaikutti käännöstyöhön? Oliko projektin käännösprosessin tekninen laajuus toimiva sekä voimmeko suositella käyttämiämme menetelmiä? Kuinka kohdealusta vaikuttaa käännöstyöhön?

Opinnäytetyön keskeisenä teemana on pelin kääntäminen kohdealustalta toiselle. Useampien kohdealustojen käyttäminen on peliyrityksen kannalta kannattava toimenpide, koska siten saavuttaa huomattavasti suuremman kohdeyleisön kuin keskittymällä yhdelle kohdealustalle. Nykyisin käännöstyötä on pyritty helpottamaan pelimoottoreilla, kuten Unreal Engine ja Unity3D, jotka tukevat laajaalaisesti eri kohdealustoja. Opinnäytetyössä keskitytäänkin erityisesti käännöstyöhön Unity-pelimoottorilla ja verrataan työtä klassisempaan käännöstyöhön, jossa jopa pelimoottori voidaan joutua kääntämään kohdealustalle yhteensopivaksi.

Tässä opinnäytetyössä tarkastellaan projektin aikana käytettyjä työkaluja ja menetelmiä. Kerromme ensiksi hieman yleisellä tasolla alustakääntämisestä ohjelmistokehityksessä, minkä jälkeen paneudumme tarkemmin käyttämiimme työkaluihin ja suunnitteluprosessiin. Erityisesti käsittelemme Unityllä tapahtuvan käännöstyön teknisiä aspekteja sekä tekemiämme ratkaisuja.

2 Julkaisualustojen vertailu

Tässä luvussa kerromme lyhyesti eri laitevalmistajien alustoille kehittämiseen vaadittavista lisensseistä, työkaluista ja tukiverkostoista. Julkaisualustan valinta vaikuttaa projektiin todella paljon, oli kyse sitten käännöstyöstä tai täysin uudesta projektista.

Nykyisin pienempien itsenäisten kehittäjien on helpompi aloittaa pelin kehitys tai käännöstyö suuremmille laitevalmistajille. Tilanne on muuttunut varsinkin viimeisen kymmenen vuoden aikana. Erityisesti tähän on vaikuttanut kehitystyökalumarkkinoiden koventuminen, mikä on johtanut kehitystyökalujen hinnan merkittävään laskuun.

Aikaisemmin pelin kehitysprosessi vaati valtavan määrän aikaa omien kehitystyökalujen tekemiseen tai vaihtoehtoisesti suuria summia rahaa kyseisten työkalujen lisensointia varten. Näiden seikkojen lisäksi yritykseltä vaadittiin hyvin vakaa suhde laitevalmistajaan tai todella merkittävä näyte tuotteen myymiseksi alustalle. Myös kehityslaitteiston hinnat olivat korkealla ja kehittäjien piti maksaa lisensointikustannukset sekä lisäkustannukset pelin päivittämisestä.

Nykyään laitevalmistajat ja kolmannen osapuolen markkinapaikat ovat ymmärtäneet pienempien yksityisten pelinkehittäjien arvon, kun Minecraftin kaltaiset yhden ohjelmoijan projektit saattavat tuoda kohdealustalle uskomattoman määrän näkyvyyttä ja asiakkaita (Hill 2014).

2.1 Sony

Sony Interactive Entertainment (SIE) tarjoaa ilmaiseksi mahdollisuuden rekisteröityä pelinkehittäjäksi. Jotta viralliseksi SIE-kehittäjäksi pystyy rekisteröitymään, täytyy kehittäjällä olla virallinen yritystunnus ja verotustiedot. Yritykseltä vaaditaan myös staattinen IP-osoite, jonka kautta kehittäjät pääsevät käsiksi SIE:n kehitys- ja julkaisuresursseihin. Kehittäjältä vaaditaan myös piilotettu (non-public) domain-sähköpostiosoite, eli yleiset domainit kuten Gmail tai Outlook eivät toimi. (Sony Interactive Entertainment 2016a) Rekisteröityminen vaatii myös varsin laajan hakemuksen täyttämisen, johon täytyy liittää esimerkiksi dokumentaatio yrityksen perustamisesta sekä lyhyt yhteenveto projektin sisällöstä ja aikataulusta. Näiden lisäksi kehittäjän täytyy hyväksyä SIE:n käyttöehdot (Sony Interactive Entertainment 2016a).

Rekisteröitynyt SIE-kehittäjä pystyy hoitamaan pelin julkaisun itse. Julkaistavan tuotteen täytyy kuitenkin vielä ennen myyntiin hyväksymistä läpäistä SIE:n tarkistuslista, joka on osa SIE:n laadunvarmistusprosessia. SIE-kehittäjänä pystyy itse määrittelemään tuotteen hinnan, eikä esimerkiksi Playstation Networkin (PSN) kautta pelin julkaisuun tule erillisiä lisensointimaksuja. SIE tarjoaa rekisteröityneille kehittäjille pääsyn alustakehittäjille suunniteltuun web-palveluun, jonka kautta pääsee käsiksi virallisiin resursseihin ja alustakohtaisiin tukipalveluihin. SIE:n kautta on myös ollut mahdollista lainata kehitystyöhön räätälöityjä pelikonsoleita määräajaksi, kun tuotteen kehitys on siinä vaiheessa että testaaminen oikealla alustalla on ajankohtaista. (Campbell 2013) SIE tukee tällä hetkellä kehitystyötä PlayStation 3, PlayStation 4, PlayStation Vita ja PlayStation TV -alustoille.

2.2 Microsoft

Microsoft tarjoaa pelinkehittäjille samankaltaisen palvelupaketin kuin Sony. Microsoftin julkaisuohjelman nimi on ID@Xbox, joka mahdollistaa ilmaisen rekisteröitymisen Microsoft kehittäjäksi. Onnistuneen rekisteröitymisen jälkeen pelien sertifiointi, julkaisu ja päivittäminen on ilmaista alustojen markkinapaikkojen kautta. ID@Xbox mahdollistaa ilmaisen kehityksen Xbox One ja Windows 10 laitteille, vain Universal Windows Platform nimikkeen alaisuudessa olevien ohjelmistojen kehittämiseen vaaditaan erillinen maksu. (Microsoft 2016b)

Microsoft tarjoaa kaksi ilmaista kehitystyöhön räätälöityä konsolia ilmaiseksi kehittäjille, joiden tuotteen konsepti on läpäissyt Microsoftin asettamat standardit. Microsoft lähestyy kehitystyökaluja hieman SIE:tä poiketen, sillä kaikki kaupallisesti myydyt konsolit toimivat tietyn ohjelmiston asentamisen jälkeen valmiina kehityslaitteistona. (Microsoft 2016b)

2.3 Apple

Apple tarjoaa mahdollisuuden julkaista pelejä ja sovelluksia iOS-alustoilleen App Store-markkinapaikan kautta. Sonyn ja Microsoftin julkaisuohjelmiin verrattuna Applen vastaava on huomattavasti tiukempi, niin hintansa kuin vaatimustensa puolesta. Apple on erittäin tarkka, erityisesti sisällön ja käyttöliittymän yhtenäisyyden kanssa, minkä tulee vastata toiminnallisuudeltaan sen käyttöjärjestelmien ympäristöjä. (Apple Inc. 2016a)

Applen alustoille julkaiseminen on myös kalliimpaa kuin esimerkiksi Sonyn tai Microsoftin alustoille. Applen kehityslisenssi maksaa 99 $ vuodessa, minkä lisäksi Apple ottaa 30 % sovellusten myyntituloista. Apple ei tarjoa kehityslaitteistoa, joka vaaditaan toimivien koontiversioiden tekemiseen. Toisin kuin SIE:n konsoleille kehitettäessä, kelpaavat kaupallisesti myydyt laitteet kehityslaitteistoksi. Kalliimmista lisenssimaksuista huolimatta tarjoaa Apple todella laajan ja toimivan tukiverkoston kehittäjille. (Apple Inc. 2016b)

2.4 Android

Android-alustoille julkaistaessa Googlen Play Store on saavuttanut melkein totaalisen monopoliaseman. Yksi suurimmista syistä tähän on Googlen käyttämä turvallisuuspolitiikka, kun muiden kolmansien osapuolten kauppapaikoista ladattavien sovellusten turvallisuus voi vaihdella paljon. (Hill 2015).

Google Playn julkaisuprosessissa on paljon samoja piirteitä kuin SIE:lla, Microsoftilla ja Applella. Tästä kolmikosta App Storen julkaisuprosessissa on eniten yhtäläisyyksiä Play Storen kanssa, sillä kyseisten kauppapaikkojen kohdealustat ovat älypuhelimia ja tablet-laitteita. Suurimmat erot Applen App Storen ja Googlen Play Storen välillä ovat Play Storen alhaisemmat kriteerit sovellusten tarkastuksen suhteen sekä pienemmät lisensointikustannukset. Kun Apple tarkastaa itse kaikki sen kauppapaikassa julkaistavat sovellukset, siirtää Google enemmän vastuuta laadukkaan sovelluksen julkaisemisesta kehittäjälle (Google Inc. 2016a).

Google Playn julkaisulisenssi maksaa 25 $ vuodessa, jonka lisäksi isäksi Google Play ottaa 30 % sovellusten myyntituloista (Google Inc 2016b). Normaaliin sopimukseen ei sisälly kehityslaitteistoa, joka vaaditaan toimivien koontiversioiden tekemiseen, vaan kehittäjän täytyy hankkia tai lainata tarvittava laitteisto sovellusten testaamiseksi omatoimisesti. Android-alustoille kehitykseen kelpaavat kaupallisesti myydyt laitteet kehityslaitteistoksi, samaan tapaan kuin Applen iOS-alustoille (Google Inc. 2015).

2.5 Steam

PC:n tilanne julkaisualustana on kallistunut vahvasti digitaalisen jakelun suuntaan, varsinkin Steamin kaltaisten markkinapaikkojen vallatessa suurimman osan asiakaskunnasta. Vaikka Valven Steam alusta on yksi suurimmista, on sitä lähtenyt haastamaan joukko pelijulkaisijoiden omia sekä joitakin yksityisiä markkinapaikkoja, kuten Good Old Games ja Humble Bundle.

Valven Steam markkinapaikassa ohjelman julkaisuun tarvitaan rajoittamaton (tilillä on ostettu yli 5 $ edestä sisältöä) Steam tili, jolle on ostettu 90 € maksava Steam Greenlight lisenssi. Lisenssi mahdollistaa pääsyn Steamin kehitystyökaluihin, ja sallii tuotteen listaamisen osana Steamin Greenlight julkaisuprosessia. Steam Greenlight eroaa muiden markkinapaikkojen laaduntarkastusprosessista varsin radikaalisti, sillä asiakkaat ilmaisevat kiinnostuksensa siellä oleviin tuotteisiin suoraan antamalla äänensä tuotteelle joko puolesta tai vastaan.

Kun julkaistava tuote on päässyt läpi Greenlight prosessista, on kehittäjällä varsin vapaat kädet tuotteen tai idean jatkokehityksestä ja hinnoittelusta (Valve Corporation 2016). Valven ottama osuus tuloista on salassapitosopimuksen alaista tietoa, mutta löytämiemme lähteiden mukaan se on 30 % tuotoista (Senior 2013).

3 Pelien kääntäminen

3.1 Unityn merkitys käännöstyölle

Yleisesti ottaen Unitylla kohdelaitteen vaihtamista voidaan pitää melko helppona prosessina. Toisaalta juuri käännöksen moniongelmallisuuteen Unity pyrkii tarjoamaan ratkaisua pelimoottorilla, jolla on laaja laitetuki. Ainoana varteenotettavana kilpailijana Unitylla on Unreal Engine, josta löytyy myös laaja kohdelaitetuki. Laajan kohdelaitetuen ansiosta Unity-projektien käännöksessä vältytään yleisimmiltä alkuvaiheen haasteilta. Yleisesti alkuvaiheeseen kuuluu useampi askelma. Aivan ensimmäisen peli täytyy saada pyörimään kohdelaitteella – usein tässä vaiheessa grafiikoita ei ole vielä implementoitu lainkaan, vain pelimoottoria pyöritetään kohdelaitteella. Tätä myös pidetään useimmiten vaikeimpana askelmana, sillä tässä vaiheessa myöhemmin mainitut väliohjelmistot aiheuttavat eniten ongelmia.

Myös ohjelmointikieli itsessään voi aiheuttaa ongelmia varhain. Ohjelmointikieli, jolla alkuperäinen versio pelistä on tehty, ei välttämättä ole yhteensopiva uuden kohdelaitteen kanssa. Esimerkiksi Playstation 3 ei tue C#-ohjelmointikieltä, jota esimerkiksi Windows ja Xbox360 tukee - toisin sanoen Xbox360 peliä käännettäessä Playstation 3:lle täytyy projekti kirjoittaa kokonaan uudestaan C++-ohjelmointikielellä tai rakentaa yhteensopivuus C#:n ja Playstationin välille. Lähdemateriaalista päätellen usein on järkevämpää kääntää koko projekti C++:lle, sillä tämä takaa parempia optimointimahdollisuuksia.

Muita ongelmia ovat esimerkiksi niin kutsutut väliohjelmistot, joita usein käytetään muun muassa äänten hallintaan. Myös lähdekirjastot aiheuttavat usein ongelmia käännöstyössä, erityisesti tilanteessa jossa niitä ei ole lähdekoodin muodossa saatavilla.

Näiden askelmien jälkeen usein impelentoidaan grafiikka peliin. (Wawro 2014) Erityisesti tilanteissa, joissa peliä käännetään konsoleilta tai tietokoneelta mobiilille tai käsikonsoleille, aiheutuu grafiikan implementoinnissa performanssiongelmia. Grafiikan optimointi kuitenkin lukeutuu enemmän loppuvaiheen käännöstyöhön ja on myös huomion arvoinen seikka Unityllä tehtävässä käännöstyössä.

3. 2 Käännöstyö Unity-pelimoottorilla

Kuitenkin nämä ongelmat ovat liki kaikki olemattomia Unity-projektien kohdalla. Tietenkin joitakin eroja kohdealustojen välillä on, mutta yleensä ei koko projektia joudu ohjelmoimaan uudestaan. Esimerkkinä hienoisesta eroavaisuudesta olkoon se, että iOS alustat eivät tue yli kolmeulotteisia taulukoita, ainakaan Unity-projekteissa. Kuitenkin useaan muuhun pelimoottoriin verrattuna nämä ongelmat ovat melko pieniä. Unitylla käännettäessä on kuitenkin syytä huomioida joitakin seikkoja, jotka liittyvät olennaisesti myös muuhun käännöstyöhön. Yleisesti ottaen näiden seikkojen korjaamisesta puhutaan loppuvaiheen käännöstyönä.

Eräs perustavanlaatuinen asia, johon aina käännöksessä tulee kiinnittää huomiota, ovat kontrollit (Wawro 2014). Kontrollit ovat erittäin merkittävä tekijä, kun pohdimme kohdealustan merkitystä käännöstyölle. Luonnollisesti kosketusnäyttölaitteelle ja pelikonsolille ei voi tehdä täysin samanlaisia kontrolleja, kun huomioidaan kuinka erilaisia nämä laitteet ovat.

Siinä missä toinen käyttää kosketusnäyttöä pääasiallisena kontrolloinnin välineenä, käyttää toinen ohjainta. Unity ei myöskään tue kontrollien suoraa siirtämistä kosketusnäytöltä ohjaimelle. Ohjaimen ja näppäimistön välille voidaan luoda jossain määrin yhteneväinen määrittely ohjelmointitasolla, mikäli käytetään Unityn Input Manageria. Kuitenkin mikäli hiirtä käytetään ohjauksessa, täytyy tälle luoda erikseen oma toiminnallisuutensa, eikä analogiohjaukseen käytettyä toiminnallisuutta voi sille suoraan hyödyntää. Kontrollit täytyy aina luoda kohdealustan ehdoin, mahdollisimman luontevan pelikokemuksen takaamiseksi.

Toinen huomionarvoinen seikka on käyttöliittymä. Ensinnäkin käyttöliittymän tulee olla käytettävissä kohdealustalla. Toisekseen kohdealustan näytön rajoitukset ja mahdollisuudet tulee huomioida. (Wawro 2014) Kohdealustalla käyttöliittymän käytettävyydestä täytyy huolehtia lähinnä kontrollointiaspektista. Kosketusnäyttökontrollit eivät toimi, mikäli laitteessa ei ole kosketusnäyttöä. NGUI-liitännäinen ja nykyinen Unityn oma käyttöliittymäjärjestelmä kuitenkin takaavat itsessään laajat käyttömahdollisuudet. Käytännössä käyttöliittymä voidaan tehdä toimimaan samaan aikaan kontrolleriohjauksella, hiirellä ohjattavaksi sekä kosketusnäytöllä kontrolloitavaksi. Tärkeämmäksi seikaksi nousee kohdealustalle optimoitu käyttöliittymän ulkoasu. Tässä on tärkeää sovittaa käyttöliittymä kohdealustan näytölle sopivaksi. Näytön koko ja resoluutio vaikuttavat tähän luonnollisesti. Tärkeää onkin sovittaa käyttöliittymäelementit sen kokoisiksi, että ne ovat selvästi erotettavissa kohdealustan näytöllä, mutta eivät liian isot. Toisekseen elementtien kokoon vaikuttaa myös kohdealustan kontrollointimahdollisuudet. Esimerkiksi kosketusnäytölle vaaditaan isommat käyttöliittymäelementit kuin PC:lle, johtuen siitä että PC:llä on isompi näyttö ja tarkempi kontrollointi kuin tabletilla tai puhelimella.

Muita yleisiä optimointiseikkoja täytyy myös huomioida Unity -käännöksessä. Esimerkiksi pelin grafiikka täytyy optimoida kohdealustan näytön resoluution mukaan. Mobiililaitteista ja handheld-laitteista harva pystyy vielä FullHD-resoluutioon. Uusimmat konsolit ja PC:t kuitenkin pystyvät tähän, PC:t jopa korkeampiin resoluutioihin. Täten pelin grafiikka täytyy sovittaa kohdealustalle sopivaksi. Myös kohdealustan suoritusteho täytyy huomioida grafiikassa. Polygonien määrä on tässä olennainen tekijä. Unity itsessään tarjoaa jossain määrin grafiikansovitusta kohdealustalle, mutta olisi aina parempi saada alkuperäiset graafiset elementit valmiiksi sovitettuna, sillä Unityn omat pakkaustekniikat voivat aiheuttaa grafiikan tason merkittävän laskun, jota ei välttämättä aiheutuisi valmiiksi optimoidulla grafiikalla.

Myös kohdelaitteen suoritusteho nousee olennaiseen osaan käännöstyötä tehdessä. Unity on kuitenkin pelimoottori, jolla pystyy tekemään graafisesti vakuuttavia pelejä. Tästä johtuen esimerkiksi pelin valaistus on tärkeää miettiä kohdelaitteen mukaisesti. Myös objektien yksityiskohtaisuus, reunanpehmennykset, objektien määrä ja piirtoetäisyys nousevat avainkysymyksiksi. Unity tarjoaa kuitenkin mahdollisuuden grafiikan laadun määrittelyyn itse pelimoottorista, mikä pätee kaikille objekteille. Näin esimerkiksi objektien yksityiskohtaisuutta voidaan vähentää sekä laatutaso-, että kohdelaitekohtaisesti. Tämä nopeuttaa käännöstyötä huomattavasti, kun jokaiselle koontiversiolle ei tarvitse erikseen määritellä grafiikan laatuja. Näin myös peliä voidaan optimoida toimimaan paremmin, esimerkiksi korkeammalla ruudunpäivitysnopeudella. Tärkeimpiä tekijöitä, joihin grafiikan laatu vaikuttaa, on materiaalien yksityiskohtaisuus ja piirtoetäisyys (Garvin 2012).

Syynä grafiikan laadun määrittelyyn kohdelaitekohtaisesti on ensisijaisesti laitteen muodostavien komponenttien kokonaisuus, joka aiheuttaa mahdollisesti niin kutsuttuja pullonkauloja. Kohdelaitteen komponentit eivät yksinkertaisesti ole kykeneviä suoriutumaan pelin aiheuttamasta rasituksesta, käytännössä estäen pelin suorituksen laitteella. Siinä missä uuden sukupolven konsolit pystyvät suoriutumaan lukuisista hiukkastehosteista, yksityiskohtaisista objekteista, joissa on shaderein luotua kiiltoa ja muuta näyttävyyttä lisäävää efektiä, on totuus erittäin toisenlainen esimerkiksi mobiililla.

Mobiililla täytyy tinkiä hieman kaikesta, alkaen pelin laajuudesta (objektien määrä, pelikentän koko), päättyen grafiikkaan ja äänenlaatuun. Grafiikan laadun laskemisella tähdätään parempaan suoritukseen, mutta myös pelin koon pienentämiseen, jolloin sen siirtäminen mobiililaajakaistayhteyksillä on mahdollista. Äänenlaadusta tinkimisellä tähdätään nimenomaan vain ja ainoastaan koontiversion koon hallintaan, sillä äänenlaadulla ei ole mainittavaa vaikutusta pelin suorituksessa. (Garvin 2012)

3. 3 Käännöstyön perusteet

Oletuksena käännöstä tehtäessä on, että peli on käännettävissä. Lienee sanomattakin selvää, että uusille konsoleille tehtyä seikkailupeliä ei vain voi lähteä kääntämään mobiilialustalle. Syynä tälle on tietenkin liian monen tekijän summa, jolloin peliä ei yksinkertaisesti ole mahdollista muuttaa mobiiliyhteensopivaksi.

Käännöstyötä suunniteltaessa onkin ensimmäinen askel miettiä kohdealustan tarjoamat mahdollisuudet, rajoitteet sekä käännöksestä saavutettava hyöty. Yleensä hyöty on isomman kohdeyleisön tavoittaminen. Pelin kuitenkin täytyy olla käännettävissä kohdealustalle, ilman pelin perimmäisten tekijöiden muutostöitä. Tällöinhän kyseessä on kokonaan uusi peli. Että peliä voidaan pitää käännöksenä, täytyy perustoiminnallisuuden, perusilmeen ja pelin mahdollisen tarinan olla samat, kuin alkuperäisessäkin versiossa. Toki joitakin lisäyksiä ja vähennyksiä ominaisuuksiin voidaan tehdä, mutta suuria muutostöitä pitäisi välttää. Tämän takia kohdealustan rajoitteet esimerkiksi kontrolleihin voivat viedä peliltä pois jonkin sille tärkeän toiminnallisuuden, kun sen toteuttaminen ei ole mahdollista kohdelaitteen mahdollistamilla kontrolleilla. Tällöin käännöstyön mielekkyyttä täytyy pohtia tarkoin.

Käännöstyöstä puhuttaessa nousee myös keskustelunarvoiseksi seikaksi milloin ei voida enää puhua käännöstyöstä, vaan kyseessä on enemmänkin samaan pelisarjaan sijoittuva uusi peli, tai jopa kokonaisuudessaan uusi peli. Mikäli alkuperäiseen peliin joudutaan tekemään radikaaleja muutoksia pelin toimimiseksi uudella kohdealustalla, ei enää voitane puhua vain käännöstyöstä. Tämä pätee erityisesti siinä tilanteessa, kun pelimekaanikoita joudutaan muuttamaan rajusti kohdelaitteen takia. Tämä toki on pitkälti periaatekysymys, esimerkiksi pelimekaanikoiden rajut muutokset, mutta sama tarina kuin alkuperäisessä pelissä voi oikeuttaa käännöksestä puhumisen. Kuitenkin yleinen käytäntö ainakin isoilla pelitaloilla on ollut tehdä omat handheld-laite optimoidut pelit, jotka vain sijoittuvat samaan pelisarjaan.

Esimerkiksi Ubisoftin Assassin's Creed pelisarjan handheld-laitteille tehdyt pelit pyörivät samalla pelimoottorilla kuin täysiveriset konsoliversiot, mutta hieman kaikesta alkaen grafiikan laadusta päättyen hahmojen määrään, on tingitty. Erittäin erilaisesta pelikokemuksesta johtuen tässä tapauksessa ei enää puhuta käännöksestä, vaan on päätetty tehdä kokonaan uusi pelisarjaan liittyvä peli.

4 Projektin taustatekijät

4.1 Kehitysalusta

SIE:n Playstation Vita käsikonsoli on pääasiallinen kehitysalusta, jolle teimme pelin käännöstyön. SIE on japanilainen monikansallinen Sony Corporationin tytäryhtiö, joka on erikoistunut moniin eri alueisiin videopelialalla. Yritys perustettiin vuonna 1993 ennen alkuperäisen PlayStationin pelikonsolin julkaisua. Yritys vastaa pääasiassa PlayStation konsoleihin liittyvän laitteiston ja ohjelmiston kehityksestä, valmistamisesta ja myymisestä. SIE:n alaisuudessa toimii myös useita pelistudioita jotka valmistavat pelejä yrityksen valmistamalle konsoleille (Sony Interactive Entertainment 2016b). Yrityksen valmistamia konsoleita ovat PlayStation (Euroopassa 1995). PlayStation 2 (2000), PlayStation 3 (Euroopassa 2007), PlayStation 4 (2013) PocketStation (julkaisu vain Japanissa 1998), PlayStation Portable (2005), PlayStation Portable Go (2009) Playstation Vita (2011). (Sony Interactive Entertainment 2016c)

4.2 Toimeksiantaja

Teimme opinnäytetyön aiheena olevan pelin käännöstyön Polar Bunnylle. Polar Bunny on vuonna 2013 Joensuussa perustettu pelistudio, jonka perustajajäsenet olivat vanhoja Tripworks pelistudion työntekijöitä. Perustajajäsenet työskentelivät ennen Polar Bunnyn perustamista pääosin opetuspelien parissa. Yrityksen ensimmäinen julkaisu oli PC:lle ja MAC:lle Steamin kautta julkaistu Parcel.

4.3 Unity3D-Pelimoottori

Käytimme Unity-pelimoottoria käännösprojektin aikana. Päädyimme kyseiseen ratkaisuun osaksi Unityn laajan alustatuen takia. Laaja alustatuki helpottaisi myös mahdollisten tulevien käännöstöiden toteutusta. Polar Bunnyllä ja projektitiimillä oli myös eniten kokemusta kyseisen moottorin käytöstä aikaisemmista projekteista, joten emme joutuneet opettelemaan uuden moottorin käyttöä alusta asti. Lisäksi Polar Bunnyllä oli laaja kirjasto Unityn liitännäisiä, joista oli suuri hyöty projektin tekemisessä.

Unity sai alkunsa 2000-luvun alussa, kun kolme nuorta tanskalaista ohjelmoijaa kokoontui samaan kellariin rakentamaan uutta pelimoottoria. 5 vuotta myöhemmin Unity Technologies niminen yritys julkaisi ensimmäisen, ominaisuuksiltaan sangen vaatimattoman versionsa uudesta pelimoottoristaan. Pelimoottorin ominaisuuksiin kuului OpenGL-renderöijä, sen fysiikkamoottori pohjautui Novodexiin (nykyisin PhysX) ja se sisälsi tuen äänille ja C#-scriptaamiselle. Pelimoottorin ensimmäinen versio ei saavuttanut kovin suurta huomiota ja tuki ainoastaan OS X-käyttöjärjestelmää. Pelimoottorin ensimmäinen versio päivitettiin kuitenkin nopeasti tukemaan huomattavasti suositumpaa Windows-käyttöjärjestelmää. Samalla se sai myös selaintuen erillisen liitännäisen kautta. (The Internet Archive 2016; Brodkin 2013)

Unity sai seuraavan suuremman päivityksensä vuonna 2007 Unity 2.0 muodossa, joka julkaistiin ensimmäisessä sittemmin vuosittaisessa Unite konferenssissa. Uusi versio sisälsi terrain-moottorin, networking-väliohjelmiston (RakNet), reaaliaikaisen shader-renderöijän sekä yksinkertaiset käyttöliittymätyökalut. Samalla Unity Technologies julkaisi Unity Asset Server-palvelunsa, joka mahdollisti projekti resurssien jaon kehittäjien välillä helpommin (Unity Technologies 2016c).

Kun Apple julkaisi App Storen vuonna 2008, Unity Technologies kehitti pelimoottorilleen tuen iPhone-puhelimille. Tämä kasvatti Unityn suosiota merkittävästi. (Unity Technologies 2016d)

Kun vuonna 2009 Unityn suosio jatkoi kasvamistaan, Unity Technologies ilmoitti vuosittaisessa Unite konfrenssissaan muuttavansa aikaisemmin maksullisen ”indie” lisenssiversionsa maksuttomaksi (Unity Technologies 2016e; Unity Technologies 2016f).

Kolmas suurempi Unity 3.0-päivitys julkaistiin vuonna 2010. Tässä päivityksessä Unity pelimoottoriin integroitiin Ruotsalaisen Illuminate Labsin Beast tuotenimikkeellä kulkevaa valaistusteknologiaa. Sama päivitys sisälsi myös Suomalaisen Umbran occulsion culling-teknologiaa (Unity Technologies 2016g). Myöhemmin vuonna 2010 Unity Asset Store näki päivänvalon. Kyseessä on kauppapaikka peleissä käytettäville projektiresursseille, minne kuka tahansa voi laittaa myyntiin omistamaansa ohjelmakoodia, ääniä tai taidetta muiden kehittäjien käyttöön (Unity Technologies 2016h; Freeman 2010). Vuonna 2012 vielä ennen seuraavaa suurempaa virallista päivitystä, Unityyn lisättiin muun muassa uusi hiukkassysteemi, reitinhaku framework, LoD- hallinta (Level of Detail) 3D objekteille, HDR-renderöinti, global illumination-ominaisuudet ja uudelleen kirjoitettu occulsion culling-tuki (Unity Technologies 2016i).

Virallinen Unity 4.0 versio julkaistiin myöhemmin vuonna 2012. Uuden version suurimpia uudistuksia julkaisun yhteydessä olivat Mecanim-animaatiojärjestelmä, sekä DirectX 11 -tuki. Saman vuoden konferenssissa Unity Technologies ilmoitti siirtyvänsä nopeampaan päivitysrytmiin. (Unity Technologies 2016j)

Seuraavan kahden vuoden aikana pelimoottorin olemassa olevia ominaisuuksia on paranneltu, ja sille lisättiin myös tuki monille uusille alustoille kehittämiseen. Unityn versio 4.6 toi tullessaan uudet käyttöliittymätyökalut, jotka ovat erityisesti opinnäytetyömme aiheena olevan projektin kannalta tärkeät. (Unity Technologies 2016k)

Vuonna 2015 Unity 5.0 toi tullessaan useita odotettuja ominaisuuksia. Näistä merkittävimpiä ovat reaaliaikainen globaali-valaistusjärjestelmä, fysiikkaan pohjautuvat shaderit sekä uusi äänimiksaaja (Unity Technologies 2016l).

4.4 Lisenssit

Unitystä on tarjolla kaksi pääasiallista lisenssiversiota, ilmainen- ja ammattilisenssi. Unityn kahden eri lisenssin väliset erot eivät ole kovin suuret. Ilmaislisenssillä saa kaikki pelimoottorin tärkeimmät ominaisuudet ja se tukee suurinta osaa julkaisualustoista. Ammattilisenssi kuitenkin tuo paljon pieniä etuja ja osa sen ominaisuuksista on pidemmälle kehitettyjä. (Unity Technologies 2016m) Yksi suurimmista ammattilaislisenssin tuomista eduista on Unity Profiler, joka on ohjelmakoodin optimointiin tarkoitettu työkalu (Unity Technologies 2016n).

Näiden kahden lisenssin lisäksi Unityn ammattilisenssiin on olemassa omat erilliset maksulliset alustalisenssinsä iOS- ja Android-käyttöjärjestelmille. Kaikki maksulliset lisenssit myydään erikseen joko 1140 € kertamaksuna, tai 57 € kuukaudessa maksavana tilauspalveluna. (Unity Technologies 2016o)

4.5 Unityn perusominaisuudet

Unity on monipuolinen pelimoottori, joka sisältää laajan kirjon ominaisuuksia. Ominaisuudet mahdollistavat suuren määrän eri toiminnallisuuksien välisiä interaktioita. Pelimoottoreita voitaisiin ajatella laajempana versiona hiekkalaatikosta, jossa vain tekijän mielikuvitus, aika ja raha ovat rajana lopputulokselle. (Unity Technologies 2016p)

4.6 Unity Editor

Unity Editor on Unity-pelimoottorin graafisen käyttöliittymän runko. Se koostuu useista eri ikkunoista ja välilehdistä, joita käyttäjä voi liikutella ja muokata varsin vapaasti. Näitä muokkaamalla on helppo koota omiin tarkoituksiin sopiva käyttöliittymä. Käyttöliittymä voi koostua esimerkiksi pelimaailman kolmiulotteisesti kuvaavasta Scene-ikkunasta, Hierarchy-listasta josta pääsee kätevästi käsiksi kaikkiin Scenen objekteihin, Inspector-ikkunasta jonka kautta Scenen sisältämiä objekteja muokataan ja Project-listasta joka näyttää kaikki projektin sisältämät resurssit (kuva 1). (Unity Technologies 2016q)

Kuva 1. Unity3D:n graafinen käyttöliittymä.

Eräs Unityn suurimmista vahvuuksista on sen editorin helppokäyttöisyys. Sen käytettävyys nojaa vahvasti vedä ja pudota -periaatteeseen. Tämä antaa kuitenkin päällisin puolin valheellisen kuvan pelimoottorin monipuolisuudesta, sillä kaiken voi tehdä myös suoraan ohjelmakoodilla.

Unityn3D:n perusasennus sisältää laajan kirjaston valmiita, hyvin dokumentoituja ominaisuuksia (Unity Technologies 2016r). Tämän lisäksi käyttäjien tekemiä lisäosia on ladattavissa laajasti esimerkiksi Unityn omasta sovelluskaupasta (Unity Technologies 2016s).

4.7 Fysiikka

Unity on käyttänyt aikaisemmin muokattua PhysX 2.8.3 -versiota, joka päivittyi Unityn versiossa 5.0 käyttämään PhysX 3.3 -versiota. Uusi fysiikkamoottori on huomattavasti tarkempi ja nopeampi, kuin sen aikaisempi versio. (Anthony 2014)

Jotta asioille tai esineille pystytään rakentamaan uskottava fyysinen käyttäytyminen, täytyy niiden kiihtyä, törmäillä ja toimia yleisesti tunnettujen voimien mukaisesti. Tämä on toteutettu Unity3D:ssä päällisin puolin yksinkertaisesti. Käyttäjän ei tarvitse kuin liittää oikeat fysiikkaa simuloivat komponentit haluttuun objektiin ja antamaan näille tarvittavat parametriarvot, niin pelimoottoriin sisäänrakennettu fysiikkamoottori hoitaa loput. (Unity Technologies 2016t)

Rigibody on fysikaalisen käyttäytymisen kannalta yksi tärkeimmistä komponenteista. Kun Rigibody-komponentti lisätään objektiin, alkaa se käyttäytymään pelimoottorissa simuloitujen painovoimalakien mukaisesti. (Unity Technologies 2016u)

Kun objektiin lisätään Collider -komponentti, saa se komponentin tyypin mukaan määräytyvän muodon, joka simuloi törmäykset mahdollistavaa pintaa. Jos esimerkiksi pallon muotoinen objekti, johon on liitetty pelkkä Rigibody -komponentti, tippuu tasaista pinta -objektia kohti, ilman että kumpaankaan objektiin on lisätty Collider -komponenttia, läpäisee pallo -objekti pinta -objektin, koska objektien törmäys ei rekisteröidy mihinkään.

Koska kaikki fyysinen käyttäytyminen vaatii monimutkaisia laskutoimituksia, suositellaan pelejä tekeviä kehittäjiä käyttämään mahdollisimman yksinkertaisia Collider -muotoja. Pelejä käännettäessä tehokkaammalta laitteelta vähemmän tehokkaalle, on pelin sisäisen fysiikan optimointi yksi mahdollinen tapa parantaa pelin suorituskykyä. Mitä yksinkertaisempia laskutoimituksia käytetty laite joutuu laskemaan, sitä vähemmän tehoja se laitteelta syö. (Unity Technologies 2016v)

4.8 Hiukkastehosteet

Hiukkastehosteet ovat pieniä, yksinkertaisia spritejä tai meshejä joita hiukkasjärjestelmä piirtää ja liikuttelee suuria määriä kerralla. Jokainen hiukkanen edustaa vain pientä osaa suuremmasta jatkuvasti muuttuvasta kokonaisuudesta, jolloin hiukkaset yhdessä muodostavat halutun lopputuloksen. Jos käytämme savuefektiä esimerkkinä, se voisi muodostua monista yksinkertaisista savutekstuureista, jotka yksinään näyttävät savutuprahdukselta. Kun näitä lisätään useita, niin syntyy niistä oikeanlaisen liikkeen kanssa näyttävä savupilvi. (Unity Technologies 2016w)

Hiukkastehosteiden käyttäminen on varsin yksinkertaista. Sitä käsitellään pelimoottorissa omana komponenttinaan, joka vain lisätään haluttuun objektiin. Komponentin kanssa työskenteleminen on siihen valmiiksi rakennettujen parametrien kuten ajoituksen, hiukkasen kokoon, liikkeen ja värin muokkausta. Näihin parametreihin tehdyt muutokset ovat havaittavissa tosiajassa, jolloin niiden kanssa työskentely on todella intuitiivista. (Unity Technologies 2016x)

Hiukkasefekteihin liittyy vahvasti myös optimointinäkökulma. Lukuisat, monimutkaiset hiukkasefektit voivat aiheuttaa pelille performanssiongelmia, joka pelissä näkyy lähinnä FPS-määrän joskus jopa erittäin voimakkaana laskemisena. Kun hiukkaskomponentti piirretään ruudulle, luo se joka kerta uuden DrawCall-kutsun. Samasta komponentista luodut tehosteet piirtyvät kuitenkin samalla DrawCall-kutsulla.

DrawCall-kutsujen määrää hiukkastehosteiden kanssa työskennellessä voidaan siis minimoida, käyttämällä samaa hiukkastehostetta ja sen komponenttia eri paikoissa. Tämä optimointimenetelmä tuo kuitenkin omia hankaluuksiaan, sillä samaa hiukkastehoste-komponenttia käytettäessä sen sijainti interpoloituu, eikä sitä näin ollen voida kutsua moneen paikkaan samanaikaisesti. Tämän ongelman voi kuitenkin välttää käyttämällä ParticleSystem.Simulate() funktiota ennen hiukkastehosteen piirtämisen ajoa. (Banderous 2012)

Hiukkasten poolaaminen on myös mahdollista, joskin tekniseltä toteutukseltaan tiimin tuolloisen osaamistason ulottumattomissa. Käännöksessä käytetyt hiukkasefektit olivat kuitenkin sen verran yksinkertaisia suhteessa Playstation Vitan tehoihin, että optimointitarvetta ei Trivasionissa tässä tapauksessa esiintynyt. Hiukkasefektien yhtäaikainen määrä myös ruudulla oli hyvin rajallinen, mikä sulki suuren performanssiongelmia aiheuttavan tekijän pois.

4.9 Äänet

Äänet ovat tärkeä osa tunnelman luomisessa peleihin. Unityn ääniominaisuuksiin kuuluu niiden reaaliaikainen miksaus ja masterointi sekä niiden muokkaaminen valmiita tehosteita lisäämällä. Unity tukee AIFF-, WAV-, MP3- ja Ogg-äänitiedostomuotoja. Jos ei omaa laajempaa kokemusta äänien kanssa työskentelystä, voi äänet lisätä projektiin pakkaamattomassa muodossa, jolloin niiden pakkaaminen jää itse pelimoottorin tehtäväksi.

Kuten oikeassa elämässä, jokin asia tuottaa ääntä, jonka jokin toinen asia kuulee tai havaitsee. Tätä simuloidaan Unityssä käyttämällä äänilähteitä ja äänikuuntelijaa. Nämä ovat omia komponenttejaan jotka lisätään haluttuun objektiin. Esimerkiksi autoa simuloivassa objektissa voi olla auton ääniä sisältävä äänilähde -komponentti, kun taas pelaajan hahmoa kuvaava objekti sisältää äänikuuntelija -komponentin. Tässä tilanteessa kun auton liikkuu pelimaailmassa, muokkaa Unity äänen automaattisesti, jotta pelaaja pystyisi arvioimaan äänilähteen sijainnin suhteessa omaan sijaintiinsa pelkän äänen perusteella. (Unity Technologies 2016y)

4.10 Ohjelmakoodi

Pelimaailman sisällä olevien objektien käyttäytymistä muokataan niihin liitetyillä komponenteilla. Nämä komponentit ovat ohjelmakoodista koostuvia scriptejä, joita pelimoottorissa on jo sisäänrakennettuna laaja valikoima. Unityn ominaisuudet on mielestämme dokumentoitu varsin laajasti ja selkeästi.

Unity tukee pääasiallisesti kahta kieltä. Nämä ovat C# sekä UnityScript, joka on JavaScriptistä Unityä varten muokattu versio. Unityllä tehtyjä ohjelmia voi myös kirjoittaa käyttämällä molempia, mutta tämä ei ole erityisen suositeltava käytäntö. Jos projektissa käytetään useita ohjelmointikieliä, vaikeuttaa se projektin jatkokehitystä sen myöhemmässä vaiheessa. (Unity Technologies 2016z)

Unityn vakio ohjelmointiympäristö (IDE) on MonoDevelopin erikoisversio. Sen ominaisuuksiin kuuluvat esimerkiksi täydentävä tekstinsyöttö ja värikoodattu syntaksin ymmärtäminen. Vaikka MonoDevelop tulee Unityn perusasennuksen mukana, voi sen korvata jälkikäteen haluamallaan ohjelmointiympäristöllä. (Unity Technologies 2016å)

4.11 Versionhallinta

Unityn ammattilisenssin mukana tuleva Team license sisältää Unityn sisäänrakennetun versiohallinnan, joka on toteutettu yhteistyössä Plastic SCM:n kanssa (Unity Technologies 2016ä; Unity Technologies 2016ö). Ilmaislisenssin käyttäjät pystyvät liittämään Unityn ulkopuoliseen kolmannen osapuolen versionhallintajärjestelmään.

Ulkopuolista versionhallintaa käyttöönotettaessa täytyy Unity pakottaa näyttämään meta-tiedostot ja käyttämään tekstitiedostoja binääritiedostojen sijasta. Nämä asetukset löytyvät Edit->Project Settings->Editor -valikosta. Tämän lisäksi pitää varmistaa että ProjectSettings ja Assets kansiot on linkitetty versionhallinnan alaisiksi. Muut kansiot kuten juurihakemisto täytyy jättää versionhallinnan piiristä pois, erityisesti Library -kansio tulisi lisätä versionhallinnan ignore -listaan. (Unity Technologies 2016a2)

5 Muut työkalut

Trivasion-projektissa käytettiin versionhallintaan TortoiseSVN-nimistä ohjelmistoa. TortoiseSVN on Subversion periaatteella toimiva versionhallintajärjestelmä. Subversion on versionhallinta, mikä on kehitetty korvaamaan CVS versionhallinta. Subversion versionhallinnalle ominaisesti myös TortoiseSVN pohjautuu keskitettyyn versionhallintaan, jossa kaikki asiakaspäätteet muokkaavat yhteistä, keskitettyä versioarkistoa projektista.

TortoiseSVN on julkaistu alun perin vuonna 2002, kirjoittamishetkellä uusin versio on julkaistu joulukuussa 2015. Ohjelmisto on julkaistu GNU General Public Licensen alaisena ja on ilmainen ladata ja käyttää. TortoiseSVN:lle on myös luotu useita liitännäisiä, joilla sen voi integroida suoraan eri kehitysohjelmistoihin, kuten esimerkiksi Visual Studioon (VsTortoise). TortoiseSVN oli valintamme projektin versionhallintaan, sillä Polar Bunnylla oli olemassa serveri-infrastruktuuri TortoiseSVN:lle. Näin ei tarvinnut luottaa ulkopuolisen palveluntarjoajan tarjoamiin serveriratkaisuihin, vaan tiedostot saatiin pidettyä lähiverkon sisällä. TortoiseSVN oli myös laajalti jo käytössä Polar Bunnylla ja sen käyttöön oli saatavilla asiantuntija-apua yrityksen sisäisesti. Näistä syistä johtuen päädyimme käyttämään TortoiseSVN:ää versionhallintatyökaluna Trivasionin käännöstoteutuksessa.

Unityssä on myös mahdollista käyttää MonoDevelopin ja Visual Studion sijasta erinäisiä tekstieditoreita koodin muokkaamiseen ja luomiseen. Esimerkiksi Sublime Textin, Notepad++:n ja Atomin yhdistäminen Unityyn on hyvinkin helppoa - ne tarvitsee vain linkittää Unityyn asetuksista ja tämän jälkeen Unity avaa koodi-tiedoston automaattisesti valitulla ohjelmistolla.

Näihin tekstieditoreihin on myös mahdollista saada liitännäisillä syntaksin korostus ja koodin automaattinen täyttö. Esimerkiksi Atomille löytyy Omnisharp-niminen lisäosa, millä saa syntaksin korostuksen ja koodin automaattisen täytön toimimaan. Atomille ei kuitenkaan vielä ole lisäosaa, joka tunnistaisi Unityn omat luokat, esimerkiksi Vector-luokat, ja osaisi korostaa näitä. Mutta Sublime Textille tällainen lisäosa on olemassa. Toki nämä lisäosat ovat kaikki yhteisön tekemiä, joten toiminnallisuus ei välttämättä ole aina aivan toivotulla tasolla, eikä asiakastuki ole yhtä kattavaa kuin Visual Studiolla tai MonoDevelopilla.

Suurin ongelma kuitenkin muodostuu koodin debuggaamisesta, joka käytännössä on mahdotonta näillä ohjelmistoilla. Tällöin joutuu turvautumaan lähinnä Debug Logeihin ja printteihin, mikäli haluaa saada koodista tarkempaa tietoa irti. Tarkempaan, IDE:ssä tapahtuvaan koodin debuggaamiseen joutuu käyttämään yhä edelleen MonoDevelopia tai Visual Studiota.

Trivasionin parissa osa tiimistä käytti ajoittain Sublime Text Editoria tekstinkäsittelytyökaluna. Syitä tähän oli lähinnä Sublimen miellyttävämpi käyttöliittymä ja ohjelmiston värimaailma. Sublimen tummempi värimaailma ja syntaksin korostuksen värit koettiin miellyttävämmäksi pitkäjaksoisessa käytössä verrattuna MonoDevelopin vastaaviin. MonoDevelopin käyttöliittymän suurin puute on ehdottomasti se, että tuon aikaisissa versioissa teemaa ei voinut täysin muokata, minkä vuoksi työkalupalkki jää vaaleaksi. Osa tiimin jäsenistä koki tämän häiritsevänä, minkä vuoksi korvaavaa työkalua haettiin Sublimestä, jolla oli myös kattavin tuki koodin automaattiselle täytölle Unityssä.

6 Trivasion pelin käännösprojekti

6.1 Pelitestaus

Trivasionin ensimmäisen iteraation pohjalta suoritettiin pelitestausta. Tuolloin peli oli vielä kehitetty iOS-mobiililaitteille. Pelitestaus oli ulkoistettu pelitestaukseen erikoistuneelle yritykselle, Suomen alueella toimivalle QA me:lle. Pelitestauksessa pelaajien annettiin pelata Trivasionia iPadeilla. Pelaamisen aikana pelaajien verbaalisia ja nonverbaalisia reaktiota tarkkailtiin. Reaktiot yhdistettiin samalla tiettyihin pelitilanteisiin. Testaamisen jälkeen pelaajille annettiin lomake täytettäväksi, jossa oli pisteytettäviä kysymyksiä sekä tekstikenttiä vapaata palautetta varten. Palautteen pohjalta laadittiin raportti, josta selvisi pelaajien suhtautuminen tiettyihin pelin osiin. Raporttia käytettiin hyväksi pelin uutta iteraatiota suunniteltaessa, jossa peliä pyrittiin muokkaamaan sekä uudelle laitteelle, että ominaisuuksiltaan paremmaksi. Testaus keskittyi lähinnä pelin käytettävyyteen ja vastaaviin ominaisuuksiin, eikä sillä pyritty löytämään ohjelmistobugeja.

Raportti oli käytännössä tiivistetty pisteytys laadittujen kysymysten perusteella, näin antaen selvän kuvan, mitä mieltä suurin osa pelaajista pelistä oli. Vastaukset kysymyksiin esitettiin prosentuaalisesti, mikä antoi selkeän kuvan enemmistön mielipiteestä. Raporttiin oli myös pisteytyksen esityksen lisäksi liitetty relevantteja lainauksia sekä itse pelitestauksesta, että kyselylomakkeen tekstikenttäpalautteista. Näin pelaajien kokemuksia pelistä pyrittiin syventämään raportissa, tarjoten mahdollisimman tarkkaa tietoa raportin lukijalle.

Pelitestauksen palautteen pohjalta pystyttiin identifioimaan ongelmia pelissä sekä mikä pelaajia viihdytti pelissä ja mikä ei. Eräs isoimmista esiin nousseista ongelmista oli vihollisten satunnaisgenerointi, joka noudatti jokaisella pelikerralla samaa kaavaa, luoden viholliset kerta toisensa jälkeen samoihin paikkoihin. Tämä luonnollisesti söi paljon peli-iloa pelaajilta, johtaen siihen, että pelaajat yrittivät opetella ulkoa vihollisten syntymispaikat. (QA ME 2014) Tämä oli suurimpia muokattavia asioita uuteen iteraatioon.

Graafinen ilme sen sijaan keräsi hyväksyvää palautetta, joten sitä ei muutettu seuraavaankaan iteraatioon. Pelimuotojen niukkuus keräsi jossain määrin negatiivista palautetta, joten uusia pelimuotoja suunniteltiin, mitkä sitten implementoitiin peliin sitä mukaa, kun tiimillä oli aikaa. Olemassa olevia pelimuotoja myös rikastettiin uusilla ominaisuuksilla, kuten powerupeilla sekä vihollismuodostelmilla, jotka saapuivat aalloittain. Näin peliä pyrittiin rytmittämään ja pitämään pelaajan mielenkiintoa yllä pidempään, sillä pelaajat antoivat negatiivista palautetta pelin rytmityksestä.

6.2 Konseptitaide

Trivasionin konseptitaide luotiin jo ennen projektin alkuperäistä iteraatiota, vaikka sitä luonnollisesti kehitettiin pelinkehityksen edetessä. Konseptitaiteeseen kuului lähinnä vihollisten ulkoasu sekä pelaajahahmon ulkoasu. Vihollisiksi valittiin värikkäät kolmiot, jotka hohtavat pelikentällä. Värikkyys ja hohto perustuvat arcadehengen luomiseen Trivasioniin, sillä vanhemmat arcadepelit olivat usein erittäin värikkäitä ulkoasultaan. Kirkkaat ja toisistaan erottuvat värit auttavat myös pelaajaa havaitsemaan helpommin ruudulla tapahtuvat asiat. Vihollisten muodollinen monotonisuus on perusteltu minimalistisella ilmeellä, joka peliin haluttiin. Vihollisten muoto perustuu myös pelin nimeen, sillä kolmio on englanniksi triangle ja väistely on evasion, joista Trivasion on yhdistelmä, viitaten kolmioiden väistelyyn pelin keskeisenä elementtinä.

Suuri osa Trivasionin tematiikasta sijoittuu avaruuteen. Konseptitaiteessa muiden elementtien takana olevana taustana toimii avaruus, jossa tuikkivat tähdet. Tämä toteutettiin myös pelin uudessa versiossa melkein sellaisenaan. Pelaajahahmo on lähes valkoinen, hieman sinertävä pallo, jolla on asteittain vahvistuva sininen loiste reunoilla sekä perässä seuraava valohäntä. Pelaajahahmo on suunnitteluratkaisun ansiosta todella helppo erottaa vihollisista. Pelaajahahmon loisteen intensiivisyys vaihtelee pelitilanteessa erikoisominaisuuksien käytön myötä, joka toimi myös pelaajaa visuaalisesti helpottavana suunnitteluratkaisuna.

Myös käyttöliittymästä tehtiin konseptitaidetta peliin sopivan yleisilmeen löytämiseksi. Käyttöliittymän ilme on futuristinen, mikä sopii Trivasionin muuhun tematiikkaan erittäin hyvin. Trivasionia varten tehtiin myös oma fontti, joka loi jo yksinään suuren osan pelin käyttöliittymän ilmeestä.

Pyrimme pitämään pelin aikaisen käyttöliittymä mahdollisimman yksinkertaisena ja upottamaan mahdollisimman paljon visuaalisia vihjeitä jo olemassa oleviin elementteihin. Päädyimme tähän ratkaisuun jotta pelaaja pystyisi keskittymään olennaiseen, eikä ruutu olisi täynnä pelitilannetta sekoittavaa informaatiota.

Erikoisominaisuuden käyttämiseen tarkoitettua resurssia varten keskellä ruutua oli kapea, sininen, muodoltaan pyöreä mittari. Tämän mittarin ympärillä oli asteikkoa symboloiva kehä, joka pyöri aina kun resurssia ladattiin, näin antaen pelaajalle visuaalisen vihjeen, että resurssia ladataan lisää.

Myös elämät oli merkattu tämän keskelle ruutua sijoitetun mittarin välittömään läheisyyteen, aivan sen alle. Elämien indikaattorina toimivat pienet pallot, jotka olivat samanväriset kuin resurssimittari. Kun elämän menetti, pallo tyhjeni jättäen jälkeen vain pienen kehän. Näin pelaajan oli helppo tarkistaa, montako elämää hänellä oli jäljellä.

Pisteet oli sijoitettu resurssimittarin keskelle, aivan keskelle ruutua. Näin pistetilanne ja pistekertoimen määrä oli helppo tarkistaa nopeatempoisessa pelitilanteessa. Pisteiden ja pistekertoimen näyttämisessä käytettiin Trivasionin omaa fonttia.

Grafiikkaa ei tarvinnut erityisesti optimoida laitealustakohtaisesti. Lähinnä muokkaustarve käännöstilanteissa kohdistui käyttöliittymäelementteihin. Koska sekä aikaisempi iOS-alusta, että uusi Playstation Vita ovat kannettavia laitteita, ovat ne ominaisuuksiltaan ja käytettävyydeltään hyvin lähellä toisiaan. Koska näyttöjen koko on molemmilla laitteilla melkein sama, kokoeron tuoma muokkaustarve ei ole yhtä merkittävä kuin esimerkiksi televisiota käyttäville laitteille käännettäessä.

Molemmat kohdealustat tukevat myös kosketusnäyttöohjausta, joten käyttöliittymäelementtien koko pystyttiin pitämään perustellusti kosketusnäytölle optimoituna. Mikäli kyseessä olisi ollut käännöstyö esimerkiksi konsolilta käsikonsolille, olisi tilanne ollut todennäköisesti hyvin erilainen.

Esimerkiksi Uncharted: Golden Abyssissä objekteja häivytetään ruudulta, mikäli ne ovat pitkän matkan päässä. Johtuen Playstation Vitan pienestä ruudusta, objekteja häivytetään verrattain läheltä pelaajahahmoa verrattuna esimerkiksi Unchartedien Playstation 3 versioihin. Golden Abyssissa graafisia objekteja myös kierrätettiin paljon, näin ollen säästäen muistia huomattavasti. Kyseinen ratkaisu myös auttoi pientä tiimiä selviämään työtaakasta. (Garvin 2012)

Myös Trivasionissa kierrätettiin paljon graafisia objekteja, mutta performanssinäkökulman sijaan syynä oli ainoastaan tiimin pieni koko ja tuotantoaikataulu. Kyseisessä projektissa grafiikalle ei ollut suurta merkitystä kohdealustalla. Playstation Vitan korkean suorituskyvyn ansiosta mobiiliversion graafisen tason saavuttaminen ei aiheuttanut ongelmia. Grafiikan yksityiskohtaisuutta, objektien määrää ja piirtoetäisyyttä olisi jopa pystytty kasvattamaan tehokkaamman kohdealustan ansiosta. Näin ei kuitenkaan tehty johtuen suunnitellun pelin luonteesta, johon kyseisten ominaisuuksien upottaminen olisi ollut liki mahdotonta.

6.3 Graafiset resurssit

Projektin graafisen ilmeen yksinkertaisuuden takia emme tarvinneet kovinkaan suurta määrää graafisia resursseja pelin kehitysvaiheessa. Tärkeimmät tekijät olivat viholliskolmiot sekä pelaajahahmona toimiva ympyrä. Pystyimme aloittamaan tärkeimpien pelimekaniikkojen rakentamisen, kuten pelihahmon ja viholliskolmioiden liikkumisen, näiden avulla. Vihollisten ollessa erivärisiä kolmioita ja pelaajahahmon ollessa valkoinen ympyrä, ei niiden luominen ollut kovinkaan vaativa työ.

Eniten työtä aiheutui käyttöliittymän elementeistä, joita oli lukuisia, kun huomioidaan sekä valikot, että pelinaikainen käyttöliittymä. Eniten pelaajalle tietoa antava käyttöliittymä koostui lukuisista animoiduista ja/tai reaaliaikaisesti päivittyvistä elementeistä, joista suurin osa oli luotu ja suunniteltu erityisesti Trivasionin käännösversiota varten.

Vihollisten grafiikan olisi voinut toteuttaa yhdellä vaalealla kolmiolla, jonka väri olisi määritelty Unityn sisällä kaikkien vihollistyypien aikaansaamiseksi. Koska pelin iOS-version reaaliaikaista loiste-efektiä ei saatu toimimaan pelin Playstation Vita versiossa, päätettiin viholliskolmiot värittää jo grafiikkaa luodessa, että niihin valmiiksi lisätystä hehkusta saataisiin mahdollisimman toimiva.

Loiste-efektin toimimattomuus johtui efektin luoneen liitännäisen sopimattomuudesta uudelle Unityn versiolle, eikä vastaavia ominaisuuksia omaavia liitännäisiä löytynyt. Tiimillä ei myöskään ollut kokemusta omien shadereiden luomisesta, joten päädyimme toteuttamaan loisteen suoraan graafisiin elementteihin toteutuksen ajankäytöllisen tehokkuuden vuoksi. Näin loiste saataisiin toimimaan kolmioihin tehokkaimmin. Luonnollisesti tämä ratkaisu ei ollut aivan yhtä kaunis kuin reaaliajassa luotu loiste, mutta ratkaisu kuitenkin todettiin toimivaksi ja paremmaksi kuin ei loiste-efektiä ollenkaan.

Trivasionissa grafiikka implementoitiin peliobjekteihin käyttämällä Unityn omaa sprite rendereriä. Johtuen Unityn melko laajasta tuesta eri tiedostomuodoille, oli grafiikan implementointi yksinkertaista. Siihen vaadittiin vain graafisen resurssin lataaminen pelin resurssikansioon, minkä jälkeen sille määriteltiin sprite muoto, esimerkiksi yksi tai useita spritejä samassa tiedostossa. Tämän jälkeen sprite vain kiinnitettiin peliobjektiin, jolla oli sprite renderer-komponentti. Sprite renderer mahdollistaa esimerkiksi objektin värin muuttamiseen suoraan Unityn sisällä. Peliobjektin kokoa voitiin vielä tässä vaiheessa säätää muuttamalla sen skaalausta. Skaalaus mahdollistaa sen, että peliobjektien koot pysyvät suhteessa toisiinsa, vaikka graafisten resurssien pikselimäärät vaihtelisivat.

Luonnollisesti kovin suuria muutoksia näin ei kuitenkaan kannata tehdä, koska objektien tarkkuus vaihtelisi suuresti, aiheuttaen sen että osa objekteista näyttäisi terävämmiltä kuin toiset.

Pelaajahahmon grafiikka itsessään oli yksinkertainen valkoinen pallo, jolla oli ohut sininen kehä. Pelaajahahmolle luotiin häntä hiukkasefektillä, mikä venyi pallon liikkuessa nopeammin ja puolestaan lyheni pallon liikkuessa hitaammin.
Hiukkasefektejä käytettiin monen muunkin efektin luomiseen, esimerkiksi vihollisten räjähtäminen luotiin hiukkasefekteillä. Myös hyökkäystoiminnon käyttämisen jälkeen tapahtuva pelaajahahmosta ulospäin lähtevä, kolmiot tuhoava räjähdysaalto luotiin hiukkasefekteillä. Hiukkasefektit ovat erittäin hyödyllisiä tämän kaltaisten visuaalisten tehosteiden luomiseen. Ne ovat todella nopea ja helppo tapa luoda visuaalisesti erittäin näyttäviä kokonaisuuksia.

7 Ääni

7.1 Musiikki

Trivasionia varten oli tuotettu musiikkia, jota pelissä voitaisiin käyttää taustamusiikkina. Tuotantovaiheessa kappaleita oli vain yksi, mutta peliä kehitettiin niin, että myöhemmin olisi helppo lisätä kappaleita joista voitaisiin luoda soittolista. Käytännössä tämä tarkoitti Master Audio-nimisen liitännäisen käyttöä. Kyseistä liitännäistä voi käyttää sekä efekti-/tehosteäänien hallinnointiin, että taustamusiikin soittolistojen hallinnointiin. Master Audiosta löytyy lukuisia säätöjä ja asetuksia, joiden avulla musiikin ja äänet saa toistettua juuri haluamallaan tavalla, juuri haluttuun aikaan. Esimerkiksi taustamusiikkia voi vaihtaa sekä aikasidonnaisesti, että tilannesidonnaisesti. (Dark Tonic 2016)

Trivasionin musiikki oli nopeatempoista, sopien hyvin pelin nopeatempoiseen luonteeseen. Pelimusiikki oli aavistuksen rockhenkistä, mutta kuitenkin konemusiikillisilla elementeillä runsaasti höystettyä instrumentaalimusiikkia. Raskas kitaririffi melko yksinkertaiseen konerumpuraitaan yhdistettynä antoi musiikille vahvan industrialrock-tunnelman. Tämän tyylilajin musiikki sopi hyvin Trivasionin scifi-teemaan, joskin tietynlainen synkkyys musiikissa aiheutti virkistävän kontrastin Trivasionin värikkään graafisen ilmeen kanssa. Kontrasti ei kuitenkaan ollut liian suuri ollakseen teemaa rikkova tai häiritsevä.

Pelimusiikissa tärkeää on teemaan sopivuus sekä sen käyttö teknisesti ajateltuna pelissä. Musiikki on tärkeä osa pelin teeman luomisessa - liian hidastempoinen musiikki yhdistettynä Trivasionin nopeatempoiseen toimintaan olisi aiheuttanut teemallisen ristiriidan, joka osaltaan olisi mahdollisesti syönyt peli-iloa. Teknisestä näkökulmasta tarkasteltuna pelimuusikin täytyy olla niin sanotusti loopattavaa, eli kappaleen täytyy olla sellainen, että sen voi luontevasti toistaa alusta heti kappaleen loputtua, mikäli tahdotaan muodostaa yhtenäinen musiikillinen jatkumo.

Toki kappaleen vaihtaminen tai pieni tauko kappaleiden välissä on varmasti hyvä ratkaisu monissa peleissä, mutta Trivasionin teoriassa ikuisesti jatkuvassa, hyvin samankaltaisena pysyvässä pelitilanteessa musiikin vaihtaminen tai tauottaminen aiheuttaisi ikävän katkeamisen audaalisessa kokemuksessa, mikä pahimmillaan voi johtaa pelaajan keskittymisen herpaantumiseen ja siten mahdollisesti pelin päättävään virheeseen. Tämän kaltaisessa pelissä pelimuusikin vaihtaminen onkin paras hoitaa uuden pelin aluksi tai sitten ajoittamalla musiikin vaihto pelin suvantokohtaan, esimerkiksi aaltojen tullessa ruudulle. Esimerkiksi aaltovaiheelle voisi olla musiikkinsa, jolloin pelaaja voisi päätellä pelitilanteen vaihdoksesta pelkästään musiikin perusteella. Samalla myös pelaajaa saataisiin rauhoitettua hidastempoiseen pelivaiheeseen, kun musiikki olisi myös rauhallisempaa. Muusikin vaikutus on kuitenkin suuri kuulijaan, ja nopeatempoinen musiikki saa pelaajan aistimaan nopeatempoisuuden myös audaalisesta ärsykkeestä. Aivan samoin kuin rauhallinen musiikki ei sovi pelillisesti nopeatempoiseen tilanteeseen, ei nopeatempoinen, jopa uhkaava, musiikki sovi lainkaan rauhallisempaan tilanteeseen.

7.2 SFX

Audio Manager on myös erinomainen työkalu efektiäänien toistamista varten. Audio Managerin avulla on mahdollista tehdä kaikki Unityn mahdollistamat toimenpiteet äänelle graafisella käyttöliittymällä. Äänet voidaan soittaa tietyllä laukaisimella, joka määritetään scriptissä. Tämä voi esimerkiksi olla vihollisen tuhoutumiseen käytetty metodi, jonka suorituksen aikana äänen toistokäsky lähetetään Audio Managerille. Audio Managerilla on mahdollista luoda omat äänilistat erilaisiin tilanteisiin. Esimerkiksi pelaajahahmon tuhoutumiseen käytettäviä ääniä voi olla useita listalla. Tällöin pelaajan tuhoutuessa Audio Manager soittaa sattumanvaraisesti yhden äänen listalta, mikäli näin on sen asetuksissa määritetty. Listaa voidaan toki myös käyttää siten, että äänet käydään järjestyksessä läpi.

Kuitenkin, käytettynä kummin tahansa, on äänimaailma huomattavasti rikkaampi, kun tilannetta audaalisesti kuvaavia mahdollisuuksia on useampia. Audio Manager myös mahdollistaa käytetyn äänen sävelkorkeuden/taajuuden muokkaamisen. Täten vaihtelua saadaan aikaan jopa yhdellä äänitiedostolla, sen taajuutta vaihtelemalla.

Trivasionin efektiäänet koostuivat suurimmalta osin vihollisen ja pelaajan tuhoutumiseen käytettävistä äänistä, jotka kuvastivat räjähdystä. Molemmille oli Audio Managerilla luodut listat, joilta tuhoutumisen tapahtuessa toistettiin sattumanvaraisesti ääni, joka kuvasi objektin räjähdystä. Listoilla oli useampia ääniä, rikkaamman äänimaailman luomiseksi, sillä yhden ja saman äänen toisto aiheuttaisi pian kyllästymisen ääneen tällaisessa tilanteessa, jossa sitä toistetaan usein ja se on erittäin dominoiva elementti äänimaailmassa. Äänet olivat kuitenkin tarpeeksi lähellä toisiaan, jolloin pelaaja oli selvää, mitä tapahtumaa ääni kuvasti. Näin ollen vältettiin ristiriitaisuus visuaalisen ja audaalisen kokemuksen välillä, joka potentiaalisesti voi heikentää pelikokemusta radikaalisti. Esimerkiksi linnunlaulu räjähdyksen audaalisena vaikutuksena ei sopisi lainkaan Trivasionin kaltaiseen arcadepeliin, joka ei tähtää huumoriarvoisuuteen.

Myös erikoisominaisuuksille oli omat ääneensä, esimerkiksi hyökkäysboostin aikana toistettiin alkuun kovempana, loppua kohti vaimenevana eräänlaista, verbaalisesti kuvattuna lähinnä “tshuuuuum”:ia muistuttavaa ääntä. Tämän ääni päätettiin vakioida samankaltaiseksi joka tilanteessa, sillä se ei ole yhtä dominoiva tekijä äänimaailmassa kuin räjähdyksiä kuvastavat äänet. Eikä tätä ääntä myöskään toisteta läheskään yhtä usein kuin lukuisia räjähdysääniä, joita pelitilanteesta riippuen tapahtuu lukuisia jopa alle sekunnin intervallein.

Trivasionissa käytettiin myös puhuttuja, pelin etenemistä kommentoivia ääniä. Nämä äänet kommentoivat pelaajan menestystä pelissä, esimerkiksi pelaajan tuhotessa lukuisia vihollisia tämä ääni kommentoi “Great!”. Äänitiedostot olivat naisen puhumia, joista oli muokattu aavistuksen konemaiselta kuulostavia, että ne sopisivat paremmin Trivasion scifi-teemaan. Näitä ääniä ei kuitenkaan toistettu kovinkaan usein, sillä tämän tyyppisten äänien liikakäytöllä voidaan pahimmillaan nostaa pelaajan turhautumista erittäin nopeasti. Oienellä intervallilla toistettu palkitsemisääni voi turhauttaa pelaajaa hyvin nopeasti palkitsemisen sijasta.

Äänet eivät aiheuttaneet itsessään mitään tarvetta käännöstyölle. Audio Manager liitännäinen toimii suoraan sekä Vitalla, että iOS:lla. Myöskään ääniformaatin muutoksia ei tarvittu. Optimointinäkökulmasta musiikki kannattaa mieluummin streamata suoraan levyltä esilataamisen sijasta. Näin säästetään latausajoissa huomattavasti. Ääniefektit sen sijaan kannattaa esiladata, koska niiden toistossa pieninkin viive voi rikkoa immersion. (Unity Technologies 2016b2)

8 Gameplay

8.1 AI/Viholliset

Trivasionin tekoäly vastaa lähinnä vihollisten ja powerupien generoinnista kentälle. Vaikka jokaisella vihollisella on periaatteessa oma tekoälynsä, on se erittäin suoraviivainen. Jokaisen vihollisen oma tekoäly vastaa vain vihollisobjektin liikuttamisesta suoraa linjaa pitkin sekä osumahavainnoinnista (collision detection). Sen sijaan peliin rytmiin ja haasteeseen vaikuttaa huomattavasti enemmän vihollistengeneroinnista vastaava scripti, joka täyttää jo yksinkertaisen tekoälyn merkit.

8.2 Tekoäly

Trivasionin tekoäly voidaan jakaa karkeasti ottaen kahteen eri vaiheeseen. Ensimmäisessä vaiheessa viholliset satunnaisgeneroidaan pelialueelle. Toisessa vaiheessa viholliset luodaan tiettyihin muodostelmiin, jotka sitten aalloittain saapuvat pelialueelle. Vihollisten syntymistavan aikasidonnaisella vaihtelulla Trivasionia pyrittiin rytmittämään ja siten lisäämään pelin mielenkiintoa.

Uusittua vihollisten synnyttämisjärjestelmää (spawning) lähdimme toteuttamaan pelitestauksen palautteen perusteella. Trivasionin idea lyhyesti ilmaistuna on väistellä edestakaisin liikkuvalla pelaajan kontrolloimalla hahmolla erityyppisiä vihollisaaltoja. Vanhempi iteraatio oli saanut negatiivista palautetta näennäisen satunnaisesti syntyvistä vihollisaalloista, (QA ME 2014) sillä pelaajat olivat huomanneet muutaman yrityksen jälkeen, että viholliset syntyivät aina samoille linjoille eli viholliset tulivat ruudulle joka pelikerralla samalla tavalla samaan aikaan kuin edelliselläkin pelikerralla. Tämä söi huomattavasti immersiota reagointikykyä koettelevasta pelistä, jolloin pelaajat pyrkivät mieluummin opettelemaan ulkoa vihollisten syntymäjärjestyksen ja siten ennakoimaan liikkeensä pelissä. Tähän ongelmaan ensimmäinen ratkaisu oli tietenkin satunnaistaa täysin vihollisten syntyminen peliin.

Trivasion käyttää vihollisille linjoja, joita pitkin nämä liukuvat ruudulla laidasta laitaan. Vihollisten syntymisen ja ruudulle saapumisen täydellisellä satunnaistamisella oli ongelmana useamman vihollisen ilmestyminen samalle linjalle, mikä jätti paljon tyhjää tilaa ruudulle, jossa pelaajan oli helppo pitää hahmoaan turvassa. Tämä johtui siitä, että täydellisessä satunnaistamisessa jokaisen linjan käyttäminen viholliselle oli yhtä todennäköistä joka kerta, kun uusi vihollinen syntyi peliin. Tämä ongelma ratkaistiin luomalla niin kutsuttu pseudo-random-algoritmi. Samankaltaista algoritmia käytetään esimerkiksi lotto-ohjelmissa. Tämä algoritmi pohjautuu Fisher-Yatesin algoritmiin (Knuth 1969), joka karkeasti selitettynä tarkoittaa sitä, että luodaan allas, josta arvo, esimerkiksi numero, poistetaan satunnaisesti. Samaa numeroa ei voi kuitenkaan enää toiste altaasta poistaa, koska sitä ei siinä enää ole. Trivasionin tapauksessa tämä tarkoitti sitä, että linjat laitettiin taulukkoon, josta satunnaisesti poimittiin linja, jolle vihollinen syntyisi. Samalle linjalle ei kuitenkaan voisi enää syntyä uutta vihollista ennen kuin kaikki kahdeksan linjaa on täytetty vihollisilla. Tämän jälkeen prosessi alkaa alusta, taulukko sekoitetaan uudestaan ja viholliset luodaan jälleen eri järjestyksessä linjoille. Tämän toteutuksen ansiosta jokainen pelikerta olisi erilainen ja Trivasionissa menestyminen pohjaa täysin reagointikykyyn.

Trivasion ei kuitenkaan nojaa täysin satunnaisgeneroituihin vihollisiin, vaan peliä rytmittävät ajoittain syntyvät vihollismuodostelmat. Näitä muodostelmia oli useampia, eroten toisistaan lähinnä muodoltaan. Vihollismuodostelmia oli esimerkiksi nuolen muodossa lentävät viholliset sekä puoliympyrän muodostavat viholliset. Nämä aallot muodostuvat vihollisista, jotka syntyvät eri linjoille tietyllä intervallilla, näin muodostaen halutun muodon. Muodostelmien peliä rytmittävän luonteen takia ne ovat aikasidonnaiset, toimien palkintona pelaajalle, kuitenkin luoden uuden haasteen.

Kun pelaaja on selvinnyt satunnaisgeneroitujen vihollisten keskellä tarpeeksi pitkään, aktivoitiin satunnaisesti eri vihollismuodostelmia synnyttävä aaltovaihe. Haaste syntyy pelaajalle siitä, että näissä muodostelmissa on vain yksi linja vapaana pelaajalle, jolloin hänen täytyy ajoittaa hahmonsa liike siten, että hän osuu tietylle linjalle juuri oikealla hetkellä. Samalla myös pelin rytmitys muuttuu täysin, koska jatkuvan reagoinnin sijasta pelaajalta vaaditaan enemmän tarkkaa ajoitusta ja suunnitelmallisuutta. Toisaalta nämä toimivat palkintona, sillä muodostelmat ovat näyttäviä ja ne antavat pelaajalle valtavia määriä pisteitä.

Muodostelmien tekninen toteutus kulminoituu IEnumerator toteutuksen ympärille. Ajoittimia tietenkin käytetään pelivaiheen muuttamiseen satunnaisgeneroiduista vihollisista muodostelmissa eteneviin vihollisiin. Itse vihollisaallot kuitenkin luotiin käyttämällä IEnumerator-pohjaista metodia. Käytännössä tämä metodi suoritti For-lausetta täyttäen vihollisten käyttämiä linjoja vihollisobjekteilla siten, että ne luotiin linjoille tietyin aikavälein toisistaan, siten luoden erinäisiä muodostelmia. Esimerkiksi puoliympyrän luominen suoritettiin niin, että linjoja lähdettiin täyttämään sekä ylhäältä että alhaalta, kuitenkin niin että pelaajalle jäi sattumanvaraisesti ylin tai alin linja vapaaksi, jotta muodostelman väistäminen oli mahdollista. Linjoja täytettiin järjestyksessä alati kiihtyvällä tahdilla, jolloin keskemmälle siirryttäessä vihollisobjektit luotiin käytännössä lähemmäksi toisiaan. Tällä menettelyllä saatiin aikaan haluttu puoliympyrä-muodostelma. Vihollisobjektien luomisen ajoittaminen oli mahdollista IEnumerator-metodien mahdollistaman ajallisen keskeytyksen ansiosta. Tämä tarkoittaa sitä, että metodin suorittaminen voidaan keskeyttää halutuksi ajaksi, jopa kesken For-lauseen. Pysäytysaikaa säätelemällä vihollisobjektien luomisen intervallia voitiin säädellä, täten mahdollistaen aaltojen muodostamisen. (kuva 2)

```
void Update () {

    if (canSpawn)
    {
        StartCoroutine(spawnEnemies(1f));
    }
}

IEnumerator spawnEnemies(float _spawnInterwall) {

    for (int i = 0; i < enemyLines.Length; i++)
    {
        SpawnEnemy(enemyLines[i]);
        yield return new WaitForSeconds(_spawnInterwall);
    }
}
```

Kuva 2. Yksinkertainen esimerkki IEnumerator-toteutuksesta vihollisten generointiin. canSpawn booleania käytetään kontrolloimaan IEnumeratorin kutsumista, ettei sitä kutsuttaisi joka framella.

Luonnollisesti muodostelma-aaltojen muodot muuttuivat pelimuotojen mukaisesti. Esimerkiksi Survival-muodossa luotiin seiniä, joiden läpi pelaaja pystyi kulkemaan yhden linjan kokoista väylää pitkin. Näin ollen pelaaja pakotettiin hyödyntämään ajanhidastus-ominaisuutta, joka mahdollistaa suunnan muutoksen y-akselilla pelaajahahmolle. Toisaalta hyökkäysmuodossa pelaaja pakotettiin hyödyntämään tilapäistä kuolemattomuutta ja vihollisten tuhoamisominaisuutta luomalla yhden sarakkeen kokoisia seiniä, jotka täyttivät jokaisen linjan. Näin pelaaja pyrittiin tutustuttamaan paremmin ominaisuuksiin, joita hänellä oli pelissä käytettävissä.

8.3 Powerupit

Powerupit olivat erikoisominaisuuksien lisäksi yksi pelin käännösversioon tulleista lisäyksistä. Koska erikoisominaisuuksien käyttämiseksi pelaajan täytyi kerätä resurssia, päätettiin pelialueelle myös luoda powerup-objekteja. Näistä objekteista pelaaja saisi hankittua lisää resurssia käyttöönsä, mitä voi sitten käyttää pelimuodosta riippuvan erikoisominaisuuden aktivoimiseen.

Powerupeista pelaaja pystyi myös hankkimaan uusia elämiä, jotka lisäsivät aina yhden uudelleen yrityksen peliin, ilman pisteiden menetystä. Myös pistekertoimen kasvattamiselle oli olemassa omat powerupit. Powerupien ulkoasu erosi merkittävästi vihollisten ulkoasusta niiden muistuttaessa enemmän tähtiä kuin viholliskolmioita. Selkeällä ulkoasun eroamisella pelinäkymä selkeytyi, jolloin pelaajan oli helpompi tunnistaa kerättävät objektit välteltävistä objekteista pelkän visuaalisen vihjeen pohjalta.

Powerupit luotiin ruudulle satunnaisiin sijanteihin ja ne olivat liikkumattomia objekteja, joiden elinaika oli lyhyt, vain muutaman sekunnin. Turhautumisen välttämiseksi powerupit rakennettiin hakeutumaan pelaajahahmoa kohti, kun tämä vain liikkui tarpeeksi lähelle niitä. Poweruppien hakeutuminen toteutettiin niin, että kentällä aktiivisena olevat powerupobjektit jäljittivät etäisyyttä vektorin pituutta laskemalla omasta sijainnistaan maailmakoordinaatistosta pelaajaobjektin sijaintiin. Kun vektorin pituus edellä mainittujen kahden pisteen välillä oli tarpeeksi lyhyt, powerupobjekti aloitti liikkumaan kohti pelaajahahmoa. Vektorin pituuden laskemisessa käytettiin yleistä vektorin pituuden laskemiseen käytettävää kaavaa (kuva 3). Vaikka kaava sisältääkin neliöjuuren laskutoimituksen, joka on prosessorille merkittävästi hitaampaa, kuin esimerkiksi kertolaskun laskeminen, ei tästä ilmennyt suorituskykyongelmia. Poweupien määrä toki on vähäinen suhteessa Vitan prosessoritehoon, mistä syystä suorituskykyä ei tarvinnut huomioida tässä toteutuksen osa-alueessa. Myös powerupit synnytetään pelialueelle IEnumerator-metodilla, joskin powerupin sijainti arvotaan pelialueelta powerupia luodessa, jossa se pysyy paikoillaan kunnes pelaaja saapuu lähelle tai se poistetaan powerupin elinajan päätteeksi.

```
void Update () {

    if (Vector2.Distance(transform.position, player.transform.position) < 3f)
    {
        transform.position = Vector2.MoveTowards(transform.position, player.transform.position, speed * Time.deltaTime);
    }

}
```

Kuva 3. Kuvassa vektorin pituuden avulla tarkistettu etäisyys pelaajahahmon ja powerupin välillä. Pituuden ollessa tarpeeksi lyhyt, liikutetaan powerupia kohti pelaajahahmoa.

Poweruppien tarkoituksena oli antaa pelaajalle mahdollisuus hankkia jokin edellä mainituista eduista, mutta samalla pakottaa pelaaja liikkumaan suuremmalla riskillä pelialueella. Powerupit kun saattoivat ilmestyä keskelle pahinta satunnaisgeneroitua vihollislaumaa tai aivan ruudun alalaitaan, kun muodostelma-aallon ohittava reitti sijaitsi ruudun ylälaidassa. Näin ollen pelaaja joutuu tekemään riskinarviointia liittyen siihen onko järkevää käydä noutamassa powerup vai pysytellä turvallisemmalla alueella.

Powerupien peliin ilmestymisen oli aikasidonnainen. Luonnollisesti pistekertoimeen vaikuttavat powerupit ilmeistyivät kentälle useimmin. Seuraavaksi useimmin ilmestyivät hyökkäys- tai ajanhidastusresurssia lisäävät powerupit. Kaikkein harvimmin ilmestyivät pelaajalle uuden elämän antavat powerupit. Aikasidonnaisuus oli kuitenkin vielä aavistuksen sattumanvaraistettu, että mitään tiettyä kaavaa ei niiden ilmestymiselle syntynyt, näin ollen tehden ilmestymishetken tarkan arvioinnin mahdottomaksi. Esimerkiksi pistekerroinpoweupin ilmestyminen vaihteli muutamasta sekunnista kymmeneen, kun taas elämän lisäävän powerupin ilmestymisaikaväli vaihteli kahden ja kolmen minuutin välillä. Aikavälin satunnaistaminen toteutettiin Unityn Random.Range-metodilla, jota kutsuessa määritettiin pienin ja suurin mahdollinen luku, jonka se voi palauttaa. Tämän jälkeen metodia kutsuttiin aina powerupin ilmestymisestä vastaavassa metodissa, siten satunnaistaen ajan uudelleen joka ilmestymiskerran jälkeen. Ilmestymisajan ja -paikan satunnaistamisella sekä powerupin lyhyellä eliniällä pelaaja pakotettiin jälleen puhtaasti reagointipohjaiseen ratkaisuun pelitilanteen käsittelemiseksi.

8.4 Pelin rytmitys

Pelivaiheiden rytmityksestä vastaavat ajastimet. Ajastimet vaihtavat pelimuodon satunnaisgeneroiduista vihollisista muodostelma-aaltoihin tietyin väliajoin. Ajastimien käyttö tässä tapauksessa oli perusteltua, koska tavoitteena oli aikasidonnaisuus peliä rytmittävänä tekijänä. Muita mahdollisia vaihtoehtoja olisi toki ollut sitoa pelivaiheet pisteisiin, mutta aikasidonnaisuus koettiin sopivammaksi Trivasionin tyyliseen, kilpailullista tuntua omaavaan peliin. Tällöin pistemäärä ei vaikuta pelin vaikeuteen ja on siten tasapuolisempi, koska silloin esimerkiksi nopeasti pisteitä keräämällä ei pääse helpompaan vaiheeseen.

Ajastimet luotiin yksinkertaisesti käyttämällä arvoa, joka määritti halutun aikamäärän tietyn toiminnallisuuden tapahtumiseen. Kun peli oli käynnissä, vähennettiin tuosta arvosta joka framella framen suorittamiseen kulunut aika käyttämällä Unityn Time.deltaTime-funktiota, kunnes arvoa määrittänyt luku oli vähennetty nollaan. Tällöin laukaistiin haluttu toiminnallisuus, jonka jälkeen ajastimen arvo määritettiin alkuperäistä vastaavaksi. Tässä ratkaisussa oli huomioitava kuitenkin kesken suorittamista jääneet IEnumerator-metodit, joiden suorittaminen ei vaiheen vaihtuessa ollut suotavaa. Käytännössä tämä tarkoitti, että vaiheen vaihtuessa vihollisten generoinnista vastaavat IEnumerator-metodit täytyi sulkea käyttämällä StopCoroutine-metodia. Jos IEnumerator-metodeja ei olisi suljettu, ne olisivat jatkaneet suoritustaan loppuun asti. Tämä olisi aiheuttanut vaiheiden käyttämien IEnumerator-metodien päällekkäin suoritusta, koska aiemman vaiheen IEnumeratorin kutsu olisi ollut edelleen toiminnassa seuraavan pelivaiheen IEnumeratoria kutsuttaessa.

Varsinaisesti tekoälyn siirtäminen kohdealustalta toiselle ei vaatinut mitään alustan luomia erityistoimenpiteitä. Vitan tekoälyn toteutuksena olisi voinut olla aiemmankin version tekoäly. Kuitenkin pelitestauksen tulosten perusteella tekoälyyn oli syytä kiinnittää huomioita, sillä Trivasionin kaltaisessa yksinkertaisessa pelissä tekoälyllä on merkittävä vaikutus pelin mielekkyyteen pelaajalle. Tekoälyn toteutuksen ratkaisut on tehty silmällä pitäen pelaajien kokemuksia ja palautetta vanhemman version tekoälystä.

Tekoälyn satunnaistaminen oli luonnollisin ensiaskel paremman pelikokemuksen takaamiseksi. Uutena ideana tekoälyyn toteutettiin myös muodostelma-aallot, joilla pelaajalle pyrittiin antamaan hengähdystauko ja mahdollisuus suunnitelmallisempaan toimintaan pelin sisällä. Tekoälyn laajan muokkaustarpeen takia oli perusteltua laatia koko tekoäly uudestaan. Alkuperäisen tekoälyn perustavanlaatuisten ongelmien takia sen muokkaaminen oli mahdotonta. Näin ollen tekoälyn kohdalla teknisen toteutuksen laajuus oli oikein arvioitu ajankäytöllisesti ja laadullisesti tarkastellen.

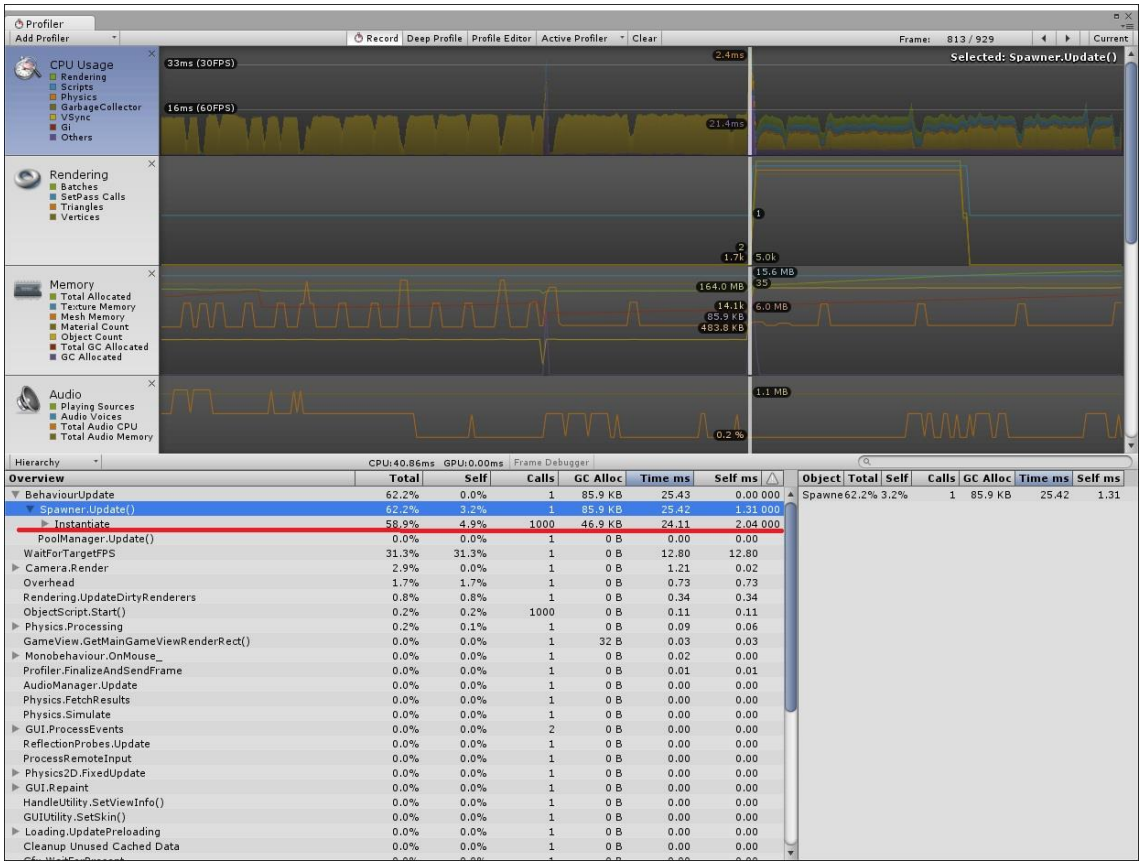
8.5 Optimointi

Vaikka Trivasion kehitettiin Playstation Vitalle, pyrittiin sitä optimoimaan mahdollisimman paljon, että rasitus konsolin tekniikalle olisi mahdollisimman vähäinen. Eräs tapa tähän oli karsia resursseja vihollisten generoinnista käyttämällä vihollisten poolausta. Käytännössä tämä tarkoittaa, että kun pelaaja käynnistää pelin ladataan viholliset altaaseen. Kun vihollista tarvitaan pelissä, se otetaan altaasta ja aktivoidaan haluttuun sijaintiin maailmakoordinaatistossa. Samantapaista lähestymistä käytettiin myös Uncharted: Golden Abyssissa, jossa pelimaailma rakennettiin uudelleen kierrätetyistä palikoista. Näin kyseisessä pelissä onnistuttiin myös vaikuttamaan performanssiin, kun objektimäärä saatiin pidettyä kohtuullisena kierrätyksen ansiosta. (Garvin 2012)

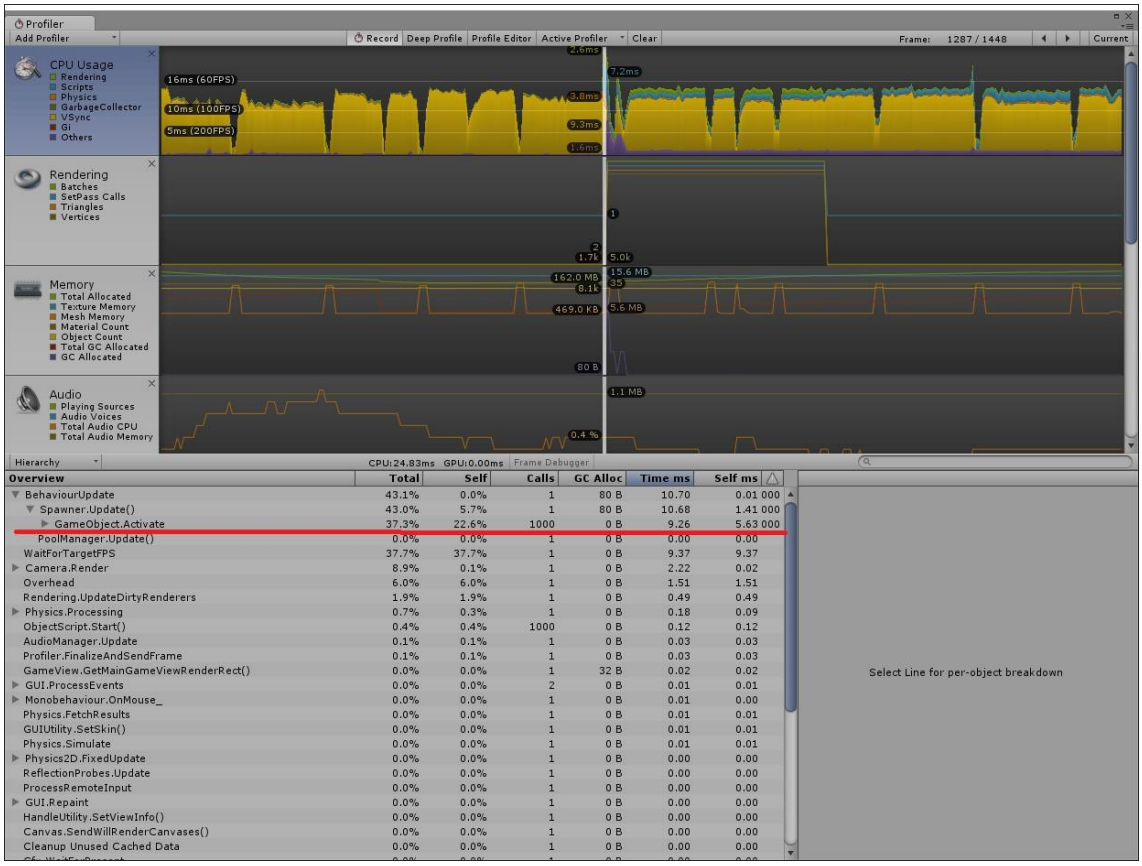
Kuvasta 4 voimme todeta Instantiate-metodilla toteutetun tuhannen peliobjektin aiheuttavan prosessorirasitusta siinä määrin, että kyseisen toiminnallisuuden toteutus kestää noin 24 millisekuntia. Kuvasta 5 näemme vastaavan toiminnallisuuden toteutuksen kestävän noin 9 millisekuntia, kun käytämme hyväksi objektien esilatausta eli poolausta. Kyseisessä yksinkertaisessa demossa, josta kuvat otettiin, objektit olivat erittäin yksinkertaisia, joiden toiminnallisuus rajoittui vain objektien liikuttamiseen. Mikäli kyseessä olisi ollut enemmän toiminnallisuuksia omaava objekti, olisi prosessorirasitus ollut huomattavasti suurempi.

Huomionarvoisimpana seikkana esiin optimointia pohdittaessa nousee Start-metodin käyttö toiminnallisuuden luomisessa. Mitä enemmän Start-metodia käytetään per objekti, sitä suotavampaa on esiladata kyseisten objektien instanssit ennen varsinaista pelin suoritusta. Näin prosessoriresursseja voidaan säästää muulle toiminnallisuudelle, ja täten taata sulavampi pelikokemus. Kyseisistä kuvista myös näkee suoraan suoritusajan korrelaation FPS -asteikolla.

Vihollisten poolaukseen käytimme Poolmanager nimistä liitännäistä (Path-o-logical Games 2011). Peliobjektien poolaus on erittäin tehokas tapa säästää resursseja, kun luodaan samasta objektista lukuisia instansseja, verrattuna esimerkiksi Unityn omaan peliobjektien instantisointiin (Unity Technologies 2016c2). Poolaus-toteutustavassa peliobjektit ovat käytännössä olemassa kokoajan, eikä niitä tarvitse ladata tarvittaessa uudestaan joka kerta. Kun peliobjekti deaktivoidaan, se palautetaan altaaseen, josta se on käytettävissä uudestaan sitä tarvittaessa. Poolaus on erityisen tehokas tapa käsitellä peliobjekteja, joita tarvitaan massoittain pelissä. Tällaisia voivat esimerkiksi olla pelissä aseen ammukset tai muut vastaavat. Trivasionissa vihollisia käytetään massoittain, minkä takia poolauksen käyttö oli perusteltua. Koko Trivasionin vihollismanageri nojaa Poolmanageriin. Poolmanagerin käyttö ei paljoa eroa normaalista objektien instantisoinnista, samaan tarkoitukseen käytetään vain hieman eri metodia.



Kuva 4. Tuhannen peliobjektin luominen instantiate metodia käyttäen.



Kuva 5. Tuhannen peliobjektin luominen poolatuista objekteista.

8.6 Kontrollit

Trivasionissa pelaajan kontrolloima hahmo on pallo, joka liikkuu automatisoidusti ruudun ylä- ja alalaidan väliä. Pelaajalla on kuitenkin mahdollisuus vaikuttaa hahmon liikkeeseen x-akselilla (sivuttaissuunnassa) halunsa mukaisesti. Attack -pelimuodossa pelaajalla on mahdollisuus aktivoida hyökkäys, jolloin pallo muuttuu tuhoutumattomaksi ja tällä voi tuhota vihollisia. Hyökkäyskyky päättyy pallon räjähtämiseen, jolla pystyy tuhoamaan viholliset kyvyn latausajan mukaan kasvavalta alueelta.

Survival -pelimuodossa kyky on ajanhidastus, joka helpottaa vihollisten väistelyä ja mahdollistaa y-akselin suunnan vaihtamisen. Molempien toiminnallisuuksien käyttöä kuitenkin rajoitetaan antamalla niiden käyttöön rajoitetusti resurssia, joka latautuu aikasidonnaisesti.

8.7 Pelaajahahmon liikkuvuus

Kontrollien suurin ongelma oli saada niistä joustavan tuntuiset, mutta tarkat. Pallon tuli liikkua liukuvan oloisesti, mutta pelaajalle kuitenkin haluttiin mahdollisimman tarkat kontrollit pallon liikuttamiseen. Näin Trivasionille pyrittiin saamaan omanlaisensa tuntuma kontrolleihin, joka kuitenkin olisi opeteltavissa niin, että erittäin tarkka ohjaus on mahdollinen.

Hyvän ohjattavuuden merkki on luonnollisuus. Kontrollien rekisteröitymisen pitää vaikuttaa ohjattavaan objektiin niin, että ohjaus tuntuu luontaiselta ja loogiselta. Ohjattavuus ei saa olla liian hankala opittavaksi ja sen täytyy olla johdonmukainen. Esimerkiksi peliobjektin äkkinäiset liikkeet kesken ohjauksen rikkovat luonnollisuutta ohjauksesta eivätkä siksi ole toivottava ilmiö. Trivasionissa peliobjektin liike on tasanopeuksista aina kun ohjauskäsky pelaajalta rekisteröidään. Peliobjekti liikkuu aina saman matkan samassa ajassa, kun ohjauskäsky rekisteröidään.

Kontrolleissa erittäin omaleimainen ratkaisu on, että pelaajalle ei oletuksena anneta kontrollia pallon vertikaaliseen liikkeeseen. Pallo siis liikkuu koko ajan, lakkaamatta, ruudun ylä- ja alalaidan väliä. Pelaaja voi kuitenkin kontrolloida pallon horisontaalista liikettä haluamallaan tavalla, väistelläkseen horisontaalisilla linjoilla liikkuvia vihollisia. Horisontaalinen liike on aavistuksen kiihtyvää, eikä se pysähdy heti ohjauskäskyn päätyttyä. Näin pallolle on pyritty saamaan massan tuntua ja ohjauksesta tekemään hieman haastavampaa. Tästä huolimatta tarkka ohjaus on edelleen mahdollista, sillä pallon liikkeet ovat vakioidut.

Esimerkiksi yhtä pitkä ohjauskäsky vasempaan suuntaan aiheuttaa aina yhtä pitkän pallon liikkeen vasemmalle ohjauskäskyn päättymisen jälkeen. Tämä ominaisuus on myös toteutettu erittäin hienovaraisesti, joten liikkeet eivät vaikuta luonnottoman suurilta. Kuitenkin tällainen ohjattavuus erityisesti tiukoissa pelitilanteissa tuo oman haasteensa ja jännittävyytensä peliin.

8.8 Näppäinten toiminnallisuus, pelihahmon ohjaus

Vanhemmassa iteraatiossa hahmoa ohjattiin kosketusnäyttöohjauksella, mutta koska uutta versiota kehitettiin Playstation Vitalle, päädyttiin uudessa versiossa hyödyntämään Playstation Vitan pelikonsoleille ominaisia kontrolleja (Wikimedia 2016). Tämä tarkoitti analogisten tattien sekä näppäinten hyödyntämistä. Tämä myös luonnollisesti tarkoitti kontrollien täydellistä uusimista vanhasta iteraatiosta, joka nojasi täysin kosketusnäytön käyttöön ainoana kontrollointimahdollisuutena. Kontrollit ovatkin eräs tärkeimpiä seikkoja käännöstyössä Unitylla, sillä ne täytyy aina suunnitella kohdealustakohtaisesti.

Uusi versio käyttää hahmon liikkeen kontrollointiin analogeja, jotka ovat luontevin tapa toteuttaa ohjaus pelikonsolilla - analogit on sijoitettu laitteeseen siten, että pelaajan käyttäessä analogeja peukaloilla jäävät muut sormet vapaiksi esimerkiksi hartianappien käyttöön.

Analogiohjauksessa tärkeäksi seikaksi muodostui tarvittavan analogin liikkeen määrä, joka liikuttaisi pelaajahahmoa ruudulla. Unity antaa laajan tuen analogiohjauksen toteutukselle (Unity Technologies 2016d2), joten esimerkiksi dead spacen säätö on erittäin helppoa. Dead spacella voidaan säädellä tarvittavaa analogin fyysistä liikettä, joka vaaditaan ohjauskäskyn rekisteröimiseen. Analogit myös mahdollistavat liikkeen määrän tarkan seuraamisen, joten pelaajahahmon kontrolloinnista saadaan erittäin tarkkaa.

Esimerkiksi näppäinohjauksessa on mahdollisuus vain näppäimen painalluksen rekisteröintiin, joka käytännössä tarkoittaa puhdasta tosi-epätosi tilannetta ohjauskäskylle. Analogiohjauksessa voidaan kuitenkin lukea kuinka paljon analogia on liikutettu tietyllä akselilla ja näin ollen määrittää haluttu toiminnallisuus. Esimerkiksi pelihahmo voi liikkua hitaammin pienemmästä analogin liikkeestä ja nopeammin suuremmasta liikkeestä. Trivasionissa kuitenkaan analogin liikkeen määrällä ei ole vaikutusta peliobjektin liikkeeseen, muuten kuin suunnallisessa merkityksessä. Tällaiseen ratkaisuun päädyttiin siksi, että pelin nopeatempoisuus vaati nopeasti reagoivaa ohjausta, joten pallon liike päätettiin vakioida tietyksi jokaisella rekisteröidyllä käskyllä.

Muiden toiminnallisuuksien kuin ohjattavuuden kanssa tuli myös huomioida pelin nopeatempoisuus. Erikoisominaisuudet, joita pelaajalla on käytössään, haluttiin sitoa sellaisiin nappeihin, jotta pelaajan olisi helppo painaa niitä irrottamatta otettaan analogeista. Näin ollen pelaajalla on mahdollisuus jatkuvasti hallita pallon liikettä, mutta samalla myös käyttää erikoisominaisuuksia. Tämä mielessä pitäen päädyimme sitomaan erikoistoiminnallisuuden Vitan tarjoamaan hartianappiin. Hartianapin käyttö etusormella on helppoa ja luonnollista konsolipelaajille samalla kun peukalot ovat vapaana analogiohjaukselle. Toinen vaihtoehto olisi ollut käyttää Vitan takaa löytyvää kosketusalustaa, joka on yhtä hyvin saavutettavissa peukaloiden ollessa sidotut ohjaukseen kuin hartianapitkin. Toisaalta tämä ratkaisu olisi vaatinut perinteisille konsoleille tehdyille versiolle pelin kontrollien muokkaamista, koska niiden ohjaimista ei tuota kosketusalustaa löydy, joten päädyimme käyttämään vain hartianappeja.

Muu vähemmän tärkeä toiminnallisuus päätettiin sitoa nappeihin, jotka eivät ole niin nopeasti käytettävissä ohjaustilanteessa kuin hartianapit. Esimerkiksi pelinaikaisen UI:n koon valitseminen suuren ja pienen väliltä löytyy rasti-painikkeen alta. Pelin keskeyttäminen ja pelivalikko löytyy luonnollisesti start-napin alta, joka on melkeinpä vakio sijoituspaikka kyseiselle toiminnallisuudelle ja kyseiselle valikolle Playstation-peleissä.

Pelivalikon kautta peliä voi jatkaa tai palata pelin päävalikkoon. Valikoissa päätimme hyödyntää Vitan kosketusnäyttöä, joten kaikki valikot toimivat sekä normaalilla nappiohjauksella, että kosketusnäyttöä käyttämällä.

Kontrollien muokkaaminen kohdealustalle sopiviksi on ensiarvoisen tärkeä tehtävä käännöstyössä (Wawro 2014). Erityisesti Trivasionissa kontrollien täydellinen uudelleensuunnittelu korostui kun käänsimme mobiilialustalta konsolille, joiden kontrollointimahdollisuudet ovat hyvin erilaiset. Siinä missä mobiililla kontrolloidaan kosketusnäytön avulla, oli PS Vitalla käytettävissä kosketuspaneeli laitteen takana, analogit, napit ja kosketusnäyttö. Tämä on myös eräs tärkeimmistä tekijöistä Trivasionissa, kun tarkastelemme kohdealustan merkitystä käännöstyölle.

9 Käyttöliittymäsuunnittelu

9.1 Käytettävyydestä

Itse käytettävyys käsitteenä on hyvin laaja. Sen määrittely on hyvin vaikeaa, koska sen eri osa-alueet riippuvat itse käytettävästä kohteesta ja sitä käyttävästä määrittelijästä. Käytettävyydelle on paljon standardeja ja määritelmiä. Esimerkiksi ISO 9241-11 -standardi määrittelee käytettävyyden kolmella kriteerillä: vaikuttavuus, tehokkuus ja tyytyväisyys, jolla tietyt määritellyt käyttäjät saavuttavat määritellyt tavoitteet tietyssä ympäristössä. Jakob Nielsen on laajentanut edellä mainittua standardia lisäämällä siihen opittavuuden, muistettavuuden ja käyttäjän tekemien virheiden vähyyden -kriteerit (Routio 2007). Käytettävyydestä löytyy valtavasti tietoa niin kirjallisuudesta, kuin internetistäkin.

Yksinkertaisimmillaan käytettävyyteen kuuluu sovelluksen tai minkä tahansa asian kehittäminen käyttäjäystävällisemmäksi, pyrkimyksenä saavuttamaan ihmisen ja asian välille sujuva yhteistoiminta.

Pelien käytettävyydestä puhuttaessa joudutaan miettimään asiaa hieman eri kannalta kuin esimerkiksi hyötysovellusten kohdalla. Hyötysovelluksen pyrkiessä puhtaasti auttamaan käyttäjää saavuttamaan lopputulos mahdollisimman tehokkaasti, helposti ja varmasti, joudutaan pelien kohdalla asiaa ajattelemaan enemmänkin elämyksen kannalta. Hyötysovellukset palvelevat jonkun olemassa olevan ongelman ratkaisemista, kun taas pelit ovat ajanvietteeksi luotujen ongelmien voittamista. Eniten pelit eroavat hyötysovelluksista juurikin niiden elämyksellisen luonteen takia. Hyötysovelluksen kohdalla sen visuaalisella ilmeellä ja äänimaailmalla on huomattavasti vähemmän merkitystä kuin kun niitä vertaa peleihin. Usein hyötysovellusta palveleekin paremmin yksinkertainen ja pelkistetty suunnittelu ja toteutus, kun taas pelien kohdalla elämyksen kokonaiskuvalla on erittäin suuri merkitys. Pelaaja pyritään saamaan uppoamaan peliin ja pelimaailmaan, jolloin pelin käytettävyyden täytyy kulkea muun pelisisällön kanssa käsi kädessä. Pelin täytyy tarjota käyttäjälle käytettävyydeltään hyvät työkalut pelin pelaamiseen. Useissa tapauksissa pelin pitääkin olla haastava, mutta yleensä haaste ei saa tulla käytettävyyteen liittyvistä asioista. Peli ei siis saa olla haastava vääristä syistä.

Pelien käytettävyyden suunnittelu kulkee käsi kädessä koko muun pelisuunnittelun kanssa. Usein pelistä pyritään tekemään helposti opittava, mutta vaikeasti täysin hallittava. Pelaajalle täytyy opettaa pelin sisällölliset perusteet joilla pelaaja pystyy suoriutumaan peliin luoduista haasteista ilman turhautumista. Yleensä pelaajalle pyritään jättämään asioita kertomatta pelin toiminnasta, jotka ovat kuitenkin mahdollista oppia jo entuudestaan pelaajalle opetettuja työkaluja käyttämällä. Tällä pyritään saavuttamaan palkitseva etenemis- ja oppimiskäyrä.

9.2 NGUI: Next-Gen UI kit

Tasharen Entertainment on pieni itsenäinen pelinkehitys studio Kanadan Ontariosta. Studio on julkaissut kaksi peliä (Lyashenko 2011a): Windwardin, joka on proseduraalisesti luotu toiminnallinen purjehduspeli sekä Starlinkin, joka on reaaliaikainen strategiapeli tähtien ja galaxien valloittamisesta (Tasharen Entertainment 2015a). Studio on tehnyt myös kaksi pelinkehityksessä käytettävää työkalupakettia, joiden päivittämistä yritys ei ole kuitenkaan jatkanut. Tästä samasta syystä ne eivät ole enää saatavilla Unityn Assets Storesta (Tasharen Entertainment 2015b). Studio on myös tehnyt Tasharen Networking-liitännäisen pelien verkko-ominaisuuksien luomiseen. Näistä huolimatta Tasharen Entertainment on kuitenkin ehkä tunnetuin sen NGUI: Next-Gen UI kit-liitännäisestä Unity-pelimoottorille (Tasharen Entertainment 2015c).

NGUI: Next-Gen UI kit on Tasharen Entertainmentin luoma liitännäinen käyttöliittymien suunnitteluun ja tekemiseen Unity-pelimoottorilla. NGUI ei vaadi Unityn maksullista Pro-lisenssiä, vaan toimii myös ilmaisella Unity lisenssillä (Lyashenko 2011b). Liitännäinen tukee iOS-, Android-, Blackberry-, Windows-, Windows Phone- ja Flash-alustoja. Sitä myydään kolmena eri lisenssiversiona, jotka ovat Standard, Professional ja Site License. NGUI on ohjelmoitu C#-ohjelmointikielellä ja se ladataan täydessä lähdekoodi-muodossa, jolloin on mahdollista muokata liitännäinen täysin omiin tarpeisiin sopivaksi.

Käännösprojektin alussa päätimme käyttää NGUI-liitännäistä. Päädyimme tähän ratkaisuun koska Polar Bunnyllä oli jo entuudestaan ostettuna lisenssi, joten liitännäisen käyttäminen oli helpoin ja kustannustehokkain tapa toteuttaa projektin käyttöliittymä. Tämän lisäksi tiimillä oli jo aikaisempaa kokemusta liitännäisen käytöstä aikaisemman projektin pohjalta.

NGUI on kevyt ja nopea tapa luoda käyttöliittymä Unityllä toteutettavaan ohjelmistotuotteeseen. Liitännäinen sisältää laajan valikoiman valmiita scriptejä, joiden ominaisuudet toimivat suoraan Inspectoriin liitettynä. Inspector on Unityn sovelluskehitysympäristön tapa hallita ohjelmistossa toimivien kohteiden sisältämiä toimintoja ja tietoa. Tämä ominaisuus mahdollistaa ideoiden nopean kokeilun, kehityksen ja testauksen.

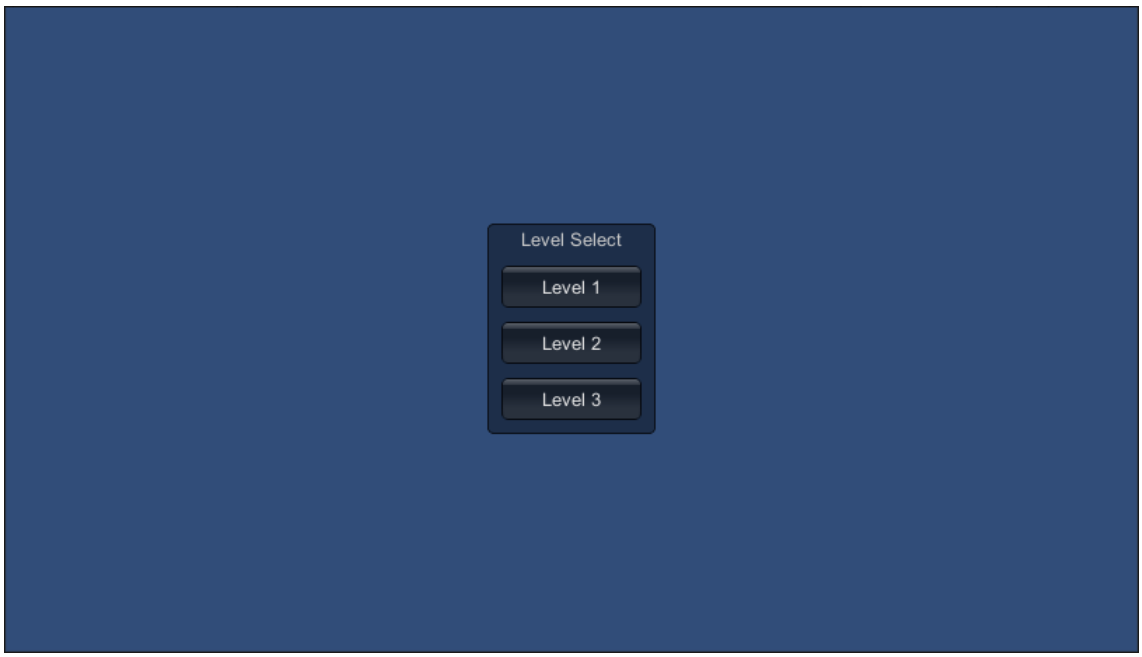
NGUI hyödyntää tekstuuri atlas tekniikkaa (Ivanov, I-A 2016), joka mahdollistaa todella pienen DrawCall määrän. Liitännäisen mukana tulee sen oma suoraan editorissa toimiva työkalu tekstuuri atlaksien luomiseen ja muokkaamiseen projektin sisältämistä kuvista.

NGUI on mielestäni hyvin dokumentoitu (Tasharen Entertainment 2014) ja sen käyttämiseen löytyy paljon Tasharen Entertainmentin omia opetusvideoita (Lyashenko 2014a). NGUI on ollut varsin laajassa käytössä Unity kehittäjien keskuudessa, joten sen käyttämiseen löytyy myös valtavasti ohjeita ja opetusvideoita kolmansien osapuolien tekemänä.

9.3 UnityGUI

Aloittaessamme käännösprojektia Unityn omat käyttöliittymätyökalut soveltuivat parhaiten eri toiminnallisuuksien testaamiseen ja kehittäjälähtöisen käyttöliittymän tekemiseen. Kun esimerkiksi testasimme pelin tallennus-ominaisuutta, oli helpompi toteuttaa napit eri pelitilojen välillä liikkumiseen (kuva 6) yksinkertaisilla UnityGUI scripteillä, kuin ryhtyä rakentamaan suurella vaivalla valikkoja ominaisuutta varten, mikä ei tulisi edes olemaan lopullisessa tuotteessa.

Toinen kehittäjälähtöinen ominaisuus oli reaaliajassa päivittyvä numero, joka kertoi pelin suorituksen aikaisen FPS-lukeman. Kyseisen luvun avulla pystyimme päättelemään mahdollisia ongelmakohtia ohjelmiston suorituskyvyssä ja tarvittaessa puuttumaan niihin. Lopullisen käännöksen kohdalla ei kuitenkaan ollut tarvetta antaa pelaajalle kyseistä tietoa, joten sen toteuttaminen vanhalla UnityGUI versiolla oli tehokas kehitysratkaisu, vanhan UnityGUI ollessa nopea ja helppo tapa toteuttaa yksinkertaisia osia käyttöliittymään.



Kuva 6. UnityGUI ennen versiota 5.0 (Kuva: Jeremiah van Oosten).

Unity Technologies palkkasi NGUI:n kehittäjän osaksi Unityn suurta UnityGUI-päivitystä. ArenMook:na tunnettu kehittäjä NGUI:n takana työskenteli omien sanojensa mukaan Unity Technologies:lla 13 kuukautta (Lyashenko 2014b). Tämä on luultavasti osasyynä käyttöliittymätyökalujen samankaltaisuuteen. Molemmat työkalut sisältävät miltei samat ominaisuudet, vain hieman eri tavalla nimettynä.

En suosittelisi enää käyttämään NGUI liitännäistä jos projektia rakennetaan Unityn versiolla 5.0 tai sitä uudemmalla. Uuden Unity-version käyttöliittymätyökalut (kuva 7) ovat vähintäänkin yhtä hyvät kuin NGUI liitännäisessä, elleivät jopa paremmat. Lisäksi NGUI:ta ei ole enää päivitetty Unity-version 4.6.7 jälkeen (Tasharen Entertainment 2015), joten mahdolliset yhteensopivuusongelmat uudempien versioiden kanssa ovat hyvin todennäköisiä.

Kuva 7. UnityGUI version 5.0 jälkeen (Kuva: Johansen Rune Skovbo).

9.4 Käytettävyys

Pelin käyttöliittymän suunnittelussa lähtökohdaksi otettiin alustan rajoitukset. Playstation Vitan ruudun ollessa 5 tuuman kokoinen, pyrimme käyttämään mobiililaitteissa hyväksi todettuja ratkaisuja. Valikoiden tulisi olla mahdollisimman selkeät ja helppokäyttöiset sekä kaikki ylimääräinen informaatio tulisi minimoida. Pidimme myös mielessä Playstation Vitan oman käyttöliittymän (kuva 8), jolloin pelaajan olisi luontevampi siirtyä laitteen valikoista itse pelin valikkoihin.

Kuva 8. Playstation Vitan käyttöliittymä (Kuva: Sony).

Pelin uusia valikoita suunniteltaessa pyrittiin niiden käytettävyyskokemus pitämään yhtenäisenä laitteen valikoiden kanssa. Käytimme kuitenkin pelin valikoissa sen omia teemoja tietyn ulkonäön luomiseksi yhtenäisen immersion tukena. Pelin valikoissa tapahtuva liike eri ikkunoiden ja valikkoalueiden välillä mukailee Vitan valikoissa käytettyjä vastaavia.

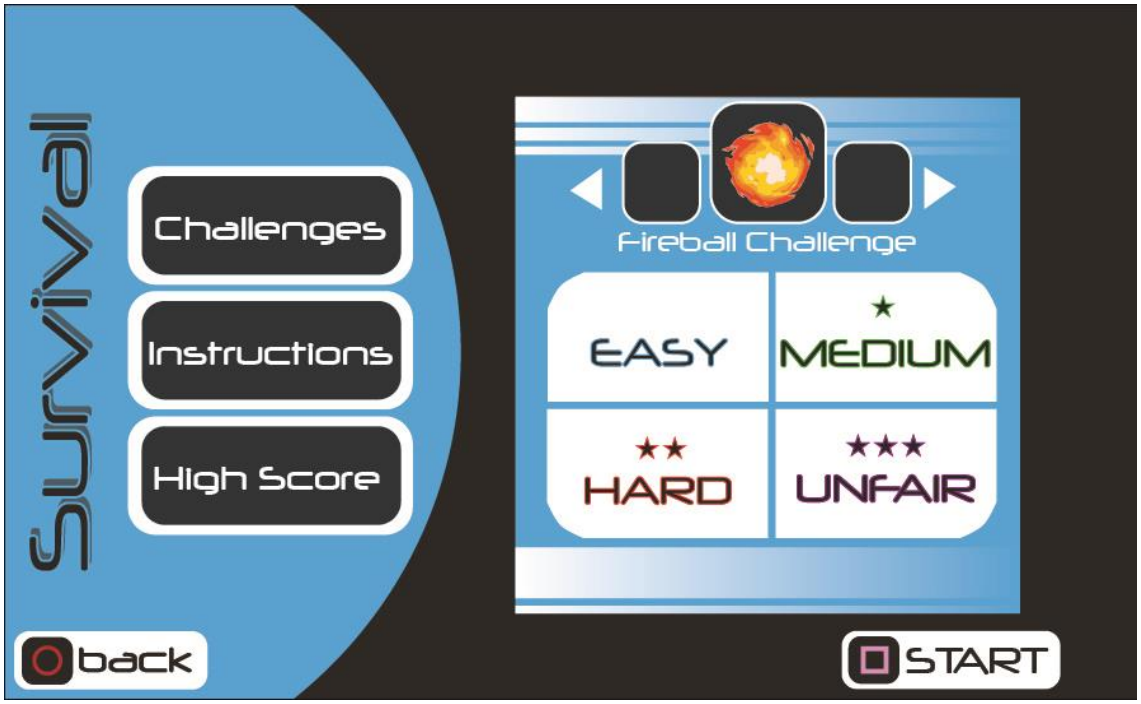
Valikoiden ikkunoista ja napeista pyrittiin tekemään mahdollisimman suuret, selkeät ja yksinkertaiset, laitteen ruudun koko huomioon ottaen. Pyrkimyksenä oli myös tukea laitteen eri ohjausmekaniikkoja mahdollisimman laajasti. Pelaaja pystyy käyttämään pelin valikoita käyttäen nuolipainikkeita tai analogista sauvaa tai painamaan suoraan käyttöliittymän elementtejä laitteen ruudulta. Tällä pyrittiin varmistamaan mahdollisimman matala oppimiskäyrä erilaisiin ohjaustapoihin tottuneille pelaajille. Pelin valikoiden ja itse pelin käyttöliittymän äänimaailma suunniteltiin yhtenäiseksi muun tematiikan kanssa.

Tiettyihin pelissä tapahtuviin toimintoihin liitetään tietty äänivihje. Yksinkertaisena esimerkkinä pelaajalle annetaan tieto valikkonapin painamisesta tietyllä äänellä. Tai tietokoneellisesti muokattu ihmisääni kertoo uuden ennätyspistemäärän saavuttamisesta pelaajalle.

9.5 Ulkonäkö

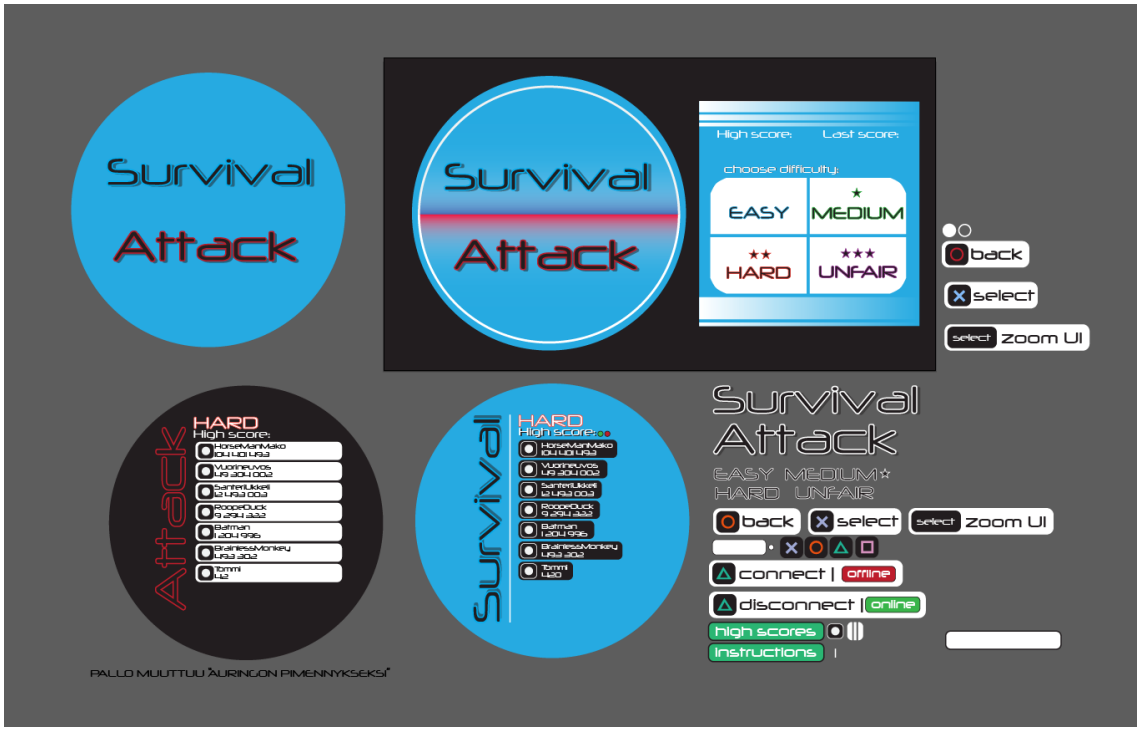
Pelin ulkonäkö oli jo todella pitkälle suunniteltu pelin aikaisemmassa versiossa, eikä sen graafiseen tyyliin haluttu tehdä liian isoja muutoksia. Pystyimme käyttämään pelin vanhoja taideresursseja uuden version kehityksen aikana, niin kutsuttuina tilapäisinä resursseina. Pelin kaikki taide oli kuitenkin tarkoitus tehdä uudestaan korkealaatuisempana julkaistavaan lopulliseen tuotteeseen.

Arhi Makkonen toimi projekti tiimimme johtajana, jonka lisäksi hän vastasi myös projektin taideresursseista. Aluksi keskustelimme mitä teemoja haluamme säilyttää vanhasta versiosta ja mitä pitäisi suunnitella täysin uudestaan. Arhi Makkonen teki keskustelujemme pohjalta peliin konseptitaidetta (kuva 9), jonka avulla pystyimme näkemään miltä ideamme mahdollisesti näyttäisivät toteutettuina. Ideoiden kokeilu ennen toteutusta on pelialalla yleinen käytäntö. On kustannustehokasta testata ideoita ennen varsinaisten resurssien ja ajan suurempaa käyttöä.



Kuva 9. Konseptikuva pelin valikoista (Kuva: Arhi Makkonen).

Kun olimme löytäneet oikeat ulkonäölliset ratkaisut pelin eri osa-alueille, pystyimme tekemään tarkemmat listat taideresursseista joita pelissä tarvitsemme. Koska pelien ulkonäkö koostuu muuttuvista ja interaktiivisista osista (kuva 10), täytyy sen taide koota pienemmistä paloista kuin palapeli.



Kuva 10. Pelin valikoissa käytettyä taidetta (Kuva: Arhi Makkonen).

9.6 Rakentaminen

Käännösprojektin alussa käytimme NGUI: Next-Gen UI kit liitännäistä valikoiden sekä pelin käyttöliittymän luomiseen.

Kun Unity päivittyi versioon 5.0, päivitys toi tullessaan uudistuneet käyttöliittymätyökalut sekä uuden teksturointi- ja valaistusjärjestelmän. Testattuamme Unity 5:n päivitettyjä ominaisuuksia, huomasimme sen ratkaisevan monia kehityksen aikana ilmenneitä ongelmia, siispä teimme päätöksen kääntää projekti uudelle Unity-versiolle.

9.7 Menut

Ensimmäinen versio pelin alkuvalikoista rakennettiin käyttämällä NGUI -liitännäistä. Pyrimme käyttämään kehitystyössä liitännäisen sisäänrakennettuja ominaisuuksia eri toiminnallisuuksia toteutettaessa. Tällä minimoimme mahdolliset tekniset ristiriidat oman ohjelmakoodimme ja liitännäisen ohjelmakoodin välillä.

Päävalikon kantava idea oli esittää pelin kaksi pelimuotoa sen pääasiallisena runkona. Ensimmäisenä pelin käynnistyessä pelaaja näkee Polar Bunnyn ja Unityn logot, joita kutsutaan yleisesti splash screeneiksi. Nämä veivät muutaman sekunnin, minkä jälkeen pelaaja näki pelin alkuvalikoiden vakioasetelman. Tämä koostui varsin suurikokoisesta pelin logosta ruudun yläosassa ja pallosta ruudun keskellä, mikä oli jaettu kahteen osaan, näin muodostaen napit pelin kahdelle eri pelimuodolle. Näiden lisäksi ruudun alanurkissa sijaitsi napit pelin asetusvalikkoon siirtymiseen ja pelin lopputekstien näyttämiseen. Näiden valikkoelementtien taustalla pyöri avaruutta simuloiva hiukkasefekti, jolla luotiin illuusio jatkuvasta liikkeestä syvässä avaruudessa.

Pelin valikoissa pystyi liikkumaan käyttäen analogisia ohjaustapoja tai koskettamalla haluttua valikkoelementtiä suoraan Playstation Vitan ruudulta. Pelaajan valittua haluttu pelimuoto, tapahtui valikkosiirtymä kyseisen pelimuodon valikkoympäristöön. Tämä toteutettiin aktivoimalla kyseistä siirtymää koskevat, kaikkia irrallisia valikko elementtejä varten tehdyt animaatiot samanaikaisesti. Näin luotiin dynaamiselta ja orgaaniselta vaikuttava valikkosiirtymä.

Pelimuotojen valikkoympäristöt seurasivat valikoiden yhteistä teemaa, käyttäen paljon samoja elementtejä toistensa välillä. Kun pelaaja valitsi päävalikosta pelimuodon, siirtyy päävalikkoa hallitseva pelimuotopallo ruudun laitaan ja sen takaa tuodaan ruudulle pelimuotoa koskeva vaikeusastevalikko. Nyt ruudun reunassa sijaitseva pallo muuttuu ”takaisin päävalikkoon”-painikkeeksi, ja uusi ruutua hallitseva valikko sisältää painikkeet pelimuodon eri vaikeusasteille.

Kahdelle pelimuodolle on tasapainotettu neljä eri vaikeusastetta, nämä ovat Easy, Normal, Hard ja Unfair. Eri vaikeusasteiden valintapainikkeet värikoodattiin ja niiden sijoittelu vaikeusastevalikossa oli molempien pelimuotojen kohdalla sama. Tällä pyrittiin pitämään valikoissa liikkuminen loogisena ja helposti opittavissa olevana kokemuksena. Kun pelaaja painaa vaikeusaste painiketta, käynnistää tämä kyseisen pelimuodon pelaajan valitsemalla vaikeusasteella. Tämä toiminnallisuus toteutettiin liittämällä vaikeusaste painikkeisiin scripti (Kuva 11), joka kertoi Switch Case-metodia käyttäen menuManagerScript:ille pelimuodon ja halutun vaikeusasteen.

```
void OnClick() {

string button = gameObject.name;

    switch(button) {

        case "Button_survEasy":

            mms.playingAttack = false;
            mms.playingSurvival = true;
            mms.playingClassic = false;
            mms.difficulty = 1;

            Application.LoadLevel (1);

        break;

    }
}
```

Kuva 11. Vaikeusaste painikkeen toiminnallisuus.

Pelin tauko- ja tallennusominaisuuksien ajo toteutettiin myös napin painalluksesta Switch Case-metodia käyttäen. Pystyimme aktivoimaan halutut toiminnallisuudet yhden scriptin kautta. Pelitauko-napin painaminen (kuva 12) asettaa pauseGameScript:ssä olevan paused-muuttujan arvon epätodeksi ja ajaa tämän jälkeen samassa scriptissä sijaitsevan PauseGame()-metodin. Pelaaja pystyy myös asettamaan pelin taukotilaan painamalla Playstation Vitan ”Start”-painiketta. Pelimuodosta poistumiseen tarkoitettu nappi seuraa samaa toimintalogiikkaa, sen kohdalla kutsutaan pelitilanteen tallentamiseen sekä päävalikkoon siirtymiseen vaadittavia metodeja.

Pelin taukotila toteutettiin käyttämällä Unityn Time.timeScale luokkaa. Tämän luokan arvoa muuttamalla pystytään määrittämään kaiken ruudunpäivitysnopeuteen sidotun toiminnallisuuden simulointinopeus. Timescalen perusarvo on 1,0 joten jos sen arvoksi asetetaan esimerkiksi 0,5, simuloidaan kaikki ruudunpäivitysnopeuteen liitetty toiminnallisuus puolella normaalista nopeudesta. PauseGame()-metodi asettaa myös tietyt peliobjektit aktiiviseksi tai epäaktiiviseksi, riippuen taukotilasta. Liitimme myös pisteiden tallennukseen vaadittavan toiminnallisuuden pelin taukotilan aktivoinnin yhteyteen. Tämä optimoi tallennus metodin ajosta johtuvaa tehonkulutusta itse gameplay loopin ulkopuolelle (kuva 13).

```
void OnClick() {

string button = gameObject.name;

    switch(button) {

        case "Button_pauseResume":

            pgs.paused = false;
            pgs.PauseGame();

        break;

        case "Button_pauseExit":

            mms.Save();
            Application.LoadLevel(0);

        break;

    }
}
```

Kuva 12. Tauko -pelitilan painikkeiden toiminnallisuus.

```
public void PauseGame()
{
    hsms.UpdateScoreFromScoreScript ();

    if(paused == true)
    {
        sms.enabled = false;

        Time.timeScale = 0.0f;
        Time.fixedDeltaTime = 0.02F * Time.timeScale;

        boostBar.SetActive (false);
        boostBarInner.SetActive (false);
        lives.SetActive (false);

        pauseScreen.SetActive (true);
    }
    else if (paused == false)
    {
        sms.enabled = true;

        Time.timeScale = 1.0f;
        Time.fixedDeltaTime = 0.02F * Time.timeScale;

        boostBar.SetActive (true);
        boostBarInner.SetActive (true);
        lives.SetActive (true);

        pauseScreen.SetActive (false);
    }
}
```

Kuva 13. PauseGame metodi tekee taukotilan aktivoimisen.

Kun pelaaja siirtyy päävalikosta pelin asetusvalikkoon, käynnistyy SettingsScript joka hallinnoi valikkonäkymän päivityksen ja pelaajan siellä tekemien muutosten tallentamisen muuttujien kautta tallennustiedostoon (kuva 14). Kuvassa on yksinkertainen versio asetusvalikon toimintalogiikasta.

Kun pelaaja siirtyy valikkoon, hakee scripti tiedon aikaisemmin tallennetuista äänen ja musiikin voimakkuuksista. Koska asetusvalikossa olevien liukusäädinten arvo on float -luku 0 ja 1 välillä ja tieto tallennetaan 0 ja 100 välillä, täytyy arvo kertoa tai jakaa käyttötavasta riippuen.

```
void Awake () {

    music = MenuManagerScript.menuManager.musicVolume;
    sound = MenuManagerScript.menuManager.soundVolume;
    musicBar.value = music / 100;
    soundBar.value = sound / 100;
}

void Update () {

    UpdateMusicAndSoundBar()
}

public void UpdateMusicAndSoundBar() {

    music = musicBar.value * 100;
    sound = soundBar.value * 100;

    MenuManagerScript.menuManager.musicVolume = music;
    MenuManagerScript.menuManager.soundVolume = sound;
}
```

Kuva 14. Ääniasetus valikon toiminnallisuus.

10 Save systeemi

Unityn omana sisäänrakennettuna tallennusjärjestelmänä toimii PlayerPrefs luokka. Kyseinen luokka tallentaa vain int-, float- ja string-tyypin listoja, joten päädyimme käyttämään omaa ratkaisuamme pelitietojen tallentamiseen.

Pelin sisäisten tietojen tallennus ja niiden lataus tapahtuu menuManagerScript:ssä. Kirjoitimme scriptiin PlayerData-luokan (kuva 15) jonka sisällä pystymme määrittelemään mitä tietoja haluamme ajon aikana tallentaa käyttäen kyseistä luokkaa Save()-metodin sisällä.

Kirjoittamamme Save() metodi luo binäärikääntäjän, jotta pystymme muuttamaan tallentamamme tiedon binäärimuotoon. Päädyimme tähän ratkaisuun koska tallennetut pelitiedot kannattaa suojata suoralta muokkaukselta.

Ilman binääriksi kääntöä pelaaja pystyy muokkaamaan mitä tahansa pelitietojaan vain avaamalla tiedoston tekstinkäsittelyohjelmalla. Seuraavaksi määritetään osoite johon binääritiedosto halutaan tallentaa. Application.dataPath kertoo pelille mihin tiedosto tallennetaan riippuen alustasta jolla peliä ajetaan. infoFileLocation on string muuttuja, jonka tilalla voidaan käyttää tiedostosijainnin osoitetta, esimerkiksi "/playerInfo.dat". Seuraavaksi luodaan PlayerData-luokasta olio johon pelin aikana muuttujiin kerätyt tiedot tallennetaan. Lopuksi tiedostossa olevat tiedot käännetään binäärimuotoon ja tiedosto suljetaan (kuva 16).

```
[Serializable]
class PlayerData
{
    public string playerName;

    public float musicVolume;
    public float soundVolume;
}
```

Kuva 15. PlayerData luokka johon pelin tietoja tallennetaan.

```
public void Save()
{
    BinaryFormatter bf = new BinaryFormatter ();

    FileStream file = File.Create (Application.dataPath + infoFileLocation);

    PlayerData data = new PlayerData ();

    data.playerName = playerName;
    data.musicVolume = musicVolume;
    data.soundVolume = soundVolume;

    bf.Serialize (file, data);
    file.Close ();
}
```

Kuva 16. Save-metodi, joka tallentaa ja kääntää tietoa binäärimuotoon.

11 Pohdinta

Unityn merkitys pelin käännöstyölle Trivasionin kohdalla oli erittäin suuri. Käytännössä Unityn käytön takia pystyimme loikkaamaan suoraan vaiheeseen, joka yleisesti tunnetaan loppuvaiheen käännöstyönä. Tässä vaiheessa pelin yhteensopivuus viimeistellään kohdealustalle muokkaamalla kontrolleja sekä grafiikkaa. Unityn erittäin laajan laitetuen ansiosta käännöstyössä pystyy hyödyntämään monia valmiita komponentteja, kuten graafisia resursseja sekä yleisesti ottaen liki kaikkea olemassa olevaa skriptiä. Toisaalta kohdelaitteen mahdollisesti asettamat rajoitukset voivat tulla vastaan ja niihin on syytä reagoida käännöstyössä. Näitä ovat yleensä kontrollit sekä suorituskykytekijät. Suorituskykytekijöistä suurimmat rajoitukset keskittyvät grafiikan yksityiskohtaisuuden, piirtoetäisyyden ja objektien määrän ympärille. Ilman Unitya olisi käännöstyö vaikeutunut huomattavasti, mikä olisi osaltaan lisännyt käännöstyöhön kuluvaa aikaa.

Trivasionin kohdalla käännösprosessia lähdettiin suorittamaan hyvinkin ruohonjuuritasolta. Projekti aloitettiin tekniseltä toteutukseltaan aivan alusta. Ohjenuorana käytimme pelimateriaalia Trivasionin aiemmasta iteraatiosta sekä kyseisen iteraation pohjalta laadittua pelitestausdokumenttia. Perusteita, miksi projekti päätettiin aloittaa tekniseltä osioltaan täysin alusta, on lukuisia.

Ensinnäkin kohdealustat erosivat toisistaan suuresti, minkä takia kontrollien sovittaminen laitteelta toiselle vaati kontrollien täydellisen uusimisen. Tästä syystä kontrolleihin vaikuttava ohjelmakoodi päätettiin laatia kokonaisuudessaan alusta, että pystyimme takaamaan parhaan mahdollisen kontrolloinnin Playstation Vita-alustalle. Tämä koettiin nopeimmaksi tavaksi taata varmasti laadukas toteutus pelin kontrollointiin.

Toisekseen Trivasionin alkuperäisen iteraation tekoäly sai pelitestauksessa suurta kritiikkiä. Tutustuimme hetkisen Trivasionin alkuperäiseen tekoälyyn, joka keskittyy lähinnä vihollisten synnyttämiseen kentälle, ja päätimme laatia myös sen kokonaan uudestaan. Syynä tähän oli alkuperäisen iteraation tekoälyn toteutuksen fundamentaaliset ongelmat, joiden takia sitä ei voinut korjata. Koimme jälleen nopeammaksi tavaksi laatia tämän osa-alueen kokonaan uudestaan laadukkaan toiminnallisuuden takaamiseksi.

Kolmanneksi graafiset resurssit täytyi uusia teknisen puolen toteutuksen ongelmien takia. Alun perin vihollisten, eli kolmioiden, hehkuefekti luotiin pelinaikaisesti lisäosalla. Harmillisesti tuo lisäosa ei enää tukenut Unityn uutta versiota, jolla me työstimme Trivasionin uutta iteraatiota. Pelimoottorin versiota oli vaihdettu uudemman version parannusten myötä. Täten graafisia resursseja muokattiin lisäämällä hehku valmiiksi grafiikkaan pelin aikaisen luomisen sijasta.

Käännöstyöksi tekemämme muutokset olivat suuria. Kaikkia tekemiämme ratkaisuja ei täten voikaan perustella pelkällä kohdealustan vaihdolla - siitä aiheutui lähinnä kontrollien täydellinen uusiminen. Olemme kuitenkin perustelleet tekemämme ratkaisut muilla vaikuttavilla tekijöillä ja koemme näiden olleen tilanteeseen parhaat mahdolliset ratkaisut. Kuitenkin koko työmme tarkoituksena oli sekä kääntää peli mobiililta Playstation Vitalle, että parannella peliä kokonaisvaltaisesti ennen julkaisua.

Käyttämämme menetelmät ovat kyllä täysin sovellettavissa käännöstyöhön yleisesti, mutta näkisimme osan niistä laajuudeltaan liian suuriksi jos puhutaan suorasta käännöstyöstä. Trivasionin kohdalla merkityksellistä on seikka, että peliä myös kehitettiin käännöstyön aikana eteenpäin, eikä se ollut missään nimessä saavuttanut lopullista muotoaan ennen käännöstyön aloitusta.

Pelien kääntämisessä on omat etunsa ja haittansa, kokonaan uuden peliprojektin tekemiseen verrattuna. Käännöstyössä voidaan hyödyntää jo olemassa olevia komponentteja, mikä osaltaan nopeuttaa käännöstyötä. Toisaalta jotkin pelin komponentit eivät välttämättä ole lainkaan sopivia uudelle kohdealustalle, jolloin vastaavan toiminnallisuuden toteuttamiseksi täytyy kehittää uusia ratkaisuja.

Mitä enemmän olemassa olevaa materiaalia voi kierrättää käännöstyössä, sitä edullisemmaksi uusien asiakkaiden tavoittaminen eri alustojen kautta muodostuu. Toisaalta mikäli käännöstyön vaatimat muutokset kyseiseen peliin ovat erittäin suuria ja perustavanlaatuisia, voi olla parempi ratkaisu kehittää kokonaan uusi peli.

Käännösprojektin laajuus määräytyy pelikohtaisesti - kehittäjien tulee ottaa monia asioita huomioon ennen varsinaisen toteutusprosessin aloittamista. Kun pelistä päätetään tehdä käännös jollekin toiselle alustalle, määräytyy koko projektin luonne alkuperäisen alustan ja tulevan kohdealustan mukaan. Yleensä pelaajalle pyritään antamaan sama kokemus kaikilla alustoilla. On kuitenkin pidettävä mielessä että tämä voi vaatia huomattavia muutoksia itse pelaajalle näkyvän rajapinnan takana.

Kohdealustan valinnalla on suuri merkitys käännösprojektiin. Käännös- tai monialustaprojektia tehdessä on hyvä ottaa huomioon mahdollisimman varhaisessa vaiheessa eri alustojen eroavaisuudet. Eri laitevalmistajilla on kohdelaitteiden ominaisuuksien lisäksi toisistaan eroavat tukiverkostot ja ne vaativat omat lisenssinsä sekä kehitystyökalunsa. Myös julkaisusopimukset täytyy tehdä alusta- ja projektikohtaisesti.

Kaikilla kohdealustoilla on omat piirteensä jotka tulisi ottaa huomioon koko projektin ajan. Tämä helpottaa itse projektin toteuttamisen lisäksi mahdollisten tulevien versioiden tekemistä eri alustoille. Projektin alussa tehdyillä valinnoilla saattaa olla hyvinkin pitkäkestoinen vaikutus projektin ja sen immateriaalisten oikeuksien tulevaisuuteen.

Teimme projektimme SIE:n Playstation Vitalle, jolloin olimme jo tutustuneet syvemmin SIE:n tarjoamiin tukiverkostoihin ja laitevaatimuksiin. Jos projekti haluttaisiin Playstation Vitan lisäksi kääntää muille alustoille, olisivat esimerkiksi Playstation 3 ja Playstation 4 olleet hyviä vaihtoehtoja, koska ne ovat hyvin lähellä toiminnallisuudeltaan Playstation Vitaa, jolloin käännöstyön vaatimien muokkausten määrä olisi pysynyt pienenä. Projektiorganisaation ei tarvitsisi kyseisessä tilanteessa tutustua täysin uuteen alustaan ja sen tuomiin haasteisiin ja mahdollisuuksiin, vaan käännöstyössä pystyttäisiin keskittymään suoraan eri laitteille julkaisuun vaadittaviin toteutusratkaisuihin.

Lähteet

Apple Inc. 2016a. App Store Review Guidelines. https://developer.apple.com/app-store/review/guidelines/. 15.4.2016.

Apple Inc. 2016b. Program Membership Details. https://developer.apple.com/programs/whats-included/. 15.4.2016.

Anthony. 2014. HIGH-PERFORMANCE PHYSICS IN UNITY 5. http://blogs.unity3d.com/2014/07/08/high-performance-physics-in-unity-5/. 15.4.2016.

Brodkin, J. 2013. How Unity3D Became a Game-Development Beast http://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. 15.4.2016.

Banderous. 2012. Moving a ParticleSystem - why does Unity interpolate its position? http://forum.unity3d.com/threads/moving-a-particlesystem-why-does-unity-interpolate-its-position.134283/. 15.4.2016.

Campbell, C. 2013. So how much does it cost to develop for PlayStation 4? Polygon. http://www.polygon.com/2013/7/24/4553842/so-how-much-does-it-cost-to-develop-for-playstation-4. 15.4.2016.

Dark Tonic. 2016. Master Audio
http://www.darktonic.com/p/master-audio.html. 15.4.2016.

Freeman, W. 2010. Unite 2010: Tech outfit introduces in-engine marketplace. http://www.develop-online.net/news/unity-launches-unity-asset-store/0108154. 15.4.2016.

Google Inc. 2016a. Launch Checklist http://developer.android.com/distribute/tools/launch-checklist.html. 15.4.2016.

Google Inc. 2016b. Sales, transaction fees, & currencies. https://support.google.com/googleplay/android-developer/answer/112622?hl=en&ref_topic=6075663. 15.4.2016.

Google Inc. 2015. Google Play -kehittäjien jakelusopimus. https://play.google.com/about/developer-distribution-agreement.html. 15.4.2016.

Hill, O. 2014. Yes, we're being bought by Microsoft. Mojang. https://mojang.com/2014/09/yes-were-being-bought-by-microsoft/. 15.4.2016.

Hill, S. 2015. Tired of Google Play? Check out these alternative Android app stores. http://www.digitaltrends.com/mobile/android-app-stores/. 15.4.2016.

Ivanov, I-A. 2016. Practical Texture Atlases. UBM Tech. http://www.gamasutra.com/view/feature/2530/practical_texture_atlases.php?print=1. 26.4.2016.

John Garvin, Jeff Ross, Francois Gilbert, Chris Reese. 2012. Post Mortem: Sony Bend Studio's Uncharted: Golden Abyss. Gamasutra
http://www.gamasutra.com/view/feature/181082/postmortem_sony_bend_studios_.php. 21.2.2016.

Lyashenko, M. 2011a. About. Tasharen Entertainment. http://www.tasharen.com/?page_id=8. 3.11.2015.

Lyashenko, M. 2011b. NGUI: Next-Gen UI kit. Tasharen Entertainment. http://www.tasharen.com/?page_id=140. 3.11.2015.

Lyashenko, M. 2014a. Tutorial Overview. Tasharen Entertainment. http://www.tasharen.com/forum/index.php?topic=6754.0. 3.11.2015.

Lyashenko, M. 2014b. Topic: New Unity GUI and NGUI. Tasharen Entertainment. http://www.tasharen.com/forum/index.php?topic=8666.0. 3.11.2015.

Microsoft. 2016a. DirectX
https://msdn.microsoft.com/library/windows/apps/hh452744. 26.4.2016.

Microsoft. 2016b. Welcome to ID@XBOX http://www.xbox.com/en-us/Developers/id. 15.4.2016.

nVidia. 2008. NovodeX / PhysX
http://www.nvidia.com/object/io_1202895129984.html. 26.4.2016.

OpenGl. 2016. OpenGL
https://www.opengl.org/. 26.4.2016.

Path-o-logical Games. 2011. Pool Manager
http://poolmanager.path-o-logical.com/. 12.4.2016.

QA ME. 2014. Playtesting report for Trivasion. 17.12.2015.

Routio, P. 2007. Vuorovaikutteisen tuotteen käytettävyys. http://www2.uiah.fi/projects/metodi/058.htm. 26.4.2016.

Sony Interactive Entertainment. 2016a. PlayStation® Partner Registration. https://partners.playstation.com/apex/PO_AccountAppliPTR?lang=en. 15.4.2016.

Sony Interactive Entertainment. 2016b. Company Profile. http://www.sie.com/en/corporate.html. 15.4.2016.

Sony Interactive Entertainment. 2016c. History of Sony Interactive Entertainment. http://www.sie.com/en/corporate/history.html 26.4.2016.

Senior, T. 2013. Steam and GOG take 30% cut of revenue vs. Humble Bundle 5% says Fez creator. http://www.pcgamer.com/steam-and-gog-take-30-revenue-cut-suggests-fez-creator-phil-fish/. 15.4.2016.

Tasharen Entertainment. Windward Press-kit. Tasharen Entertainment. http://www.tasharen.com/presskit/sheet.php?p=windward. 3.11.2015.

Tasharen Entertainment. 2014. NGUI: Next-Gen UI kit, Class List. Tasharen Entertainment. http://www.tasharen.com/ngui/docs/annotated.html. 3.11.2015.

Tasharen Entertainment. 2015a. Starlink. Google Play Store. https://play.google.com/store/apps/details?id=com.Tasharen.SG. 3.11.2015.

Tasharen Entertainment. 2015b. TNet: Tasharen Networking. Unity Asset Store. https://www.assetstore.unity3d.com/en/#!/content/6059. 03.11.2015.

Tasharen Entertainment. 2015c. NGUI: Next-Gen UI. Unity Asset Store. https://www.assetstore.unity3d.com/en/#!/content/2413. 3.11.2015.

Tasharen Entertainment. Space Game Starter Kit. Unity Asset Store. https://www.assetstore.unity3d.com/en/#!/content/2081. 3.11.2015.

Tasharen Entertainment. Ship Game Starter Kit. Unity Asset Store. https://www.assetstore.unity3d.com/en/#!/content/2074. 3.11.2015.

The Internet Archive. CORE FEATURES. https://web.archive.org/web/20050309172942/http:/otee.dk/site/Tech/5.html. 15.4.2016.

Unity Technologies. 2016a. Draw Call Batching. Unity Documentation. http://docs.unity3d.com/Manual/DrawCallBatching.html. 3.11.2015.

Unity Technologies. 2016b. Mecanim
https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/animate-anything. 26.4.2016.

Unity Technologies. 2016c. Watch all the sessions from our premier Unite conference in San Francisco. http://unite.unity.com/archive/2007. 15.4.2016.

Unity Technologies. 2016d. Watch all the sessions from our premier Unite conference in Copenhagen. http://unite.unity.com/archive/2008. 15.4.2016.

Unity Technologies. 2016e. Unity 2.6 Released And Now Free! https://unity3d.com/company/public-relations/news/unity2.6-press. 15.4.2016.

Unity Technologies. 2016f. Watch all the sessions from our premier Unite conference in San Francisco. http://unite.unity.com/archive/2009. 15.4.2016.

Unity Technologies. 2016g. Watch all the sessions from our premier Unite conference in Montreal. http://unite.unity.com/archive/2010. 15.4.2016.

Unity Technologies. 2016h. Watch all the sessions from our premier Unite 2012 conference in Amsterdam. Enjoy! http://unite.unity.com/archive/2012. 15.4.2016.

Unity Technologies. 2016i. Unity 4.0. https://unity3d.com/unity/whats-new/unity-4.0. 26.4.2016.

Unity Technologies. 2016j. Unity 4.6. https://unity3d.com/unity/whats-new/unity-4.6. 26.4.2016.

Unity Technologies. 2016k. Unity 5.0. https://unity3d.com/unity/whats-new/unity-5.0. 26.4.2016.

Unity Technologies. 2016l. GET UNITY. http://unity3d.com/get-unity. 15.4.2016.

Unity Technologies. 2016m. The Profiler Window. http://docs.unity3d.com/Manual/Profiler.html. 15.4.2016.

Unity Technologies. 2016n. Configure your Unity Pro Subscription. https://store.unity3d.com/subscribe?currency=EUR. 15.4.2016.

Unity Technologies. 2016o. Unity Manual. http://docs.unity3d.com/Manual/index.html. 15.4.2016.

Unity Technologies. 2016p. Learning the Interface. http://docs.unity3d.com/Manual/LearningtheInterface.html. 15.4.2016.

Unity Technologies. 2016q. Welcome to the Unity Scripting Reference! http://docs.unity3d.com/ScriptReference/. 15.4.2016.

Unity Technologies. 2016r. Importing from the Asset Store. http://docs.unity3d.com/Manual/AssetStore.html. 15.4.2016.

Unity Technologies. 2016s. Physics. http://docs.unity3d.com/Manual/PhysicsSection.html. 15.4.2016.

Unity Technologies. 2016t. Rigidbodies. http://docs.unity3d.com/Manual/RigidbodiesOverview.html. 15.4.2016.

Unity Technologies. 2016u. Colliders. http://docs.unity3d.com/Manual/CollidersOverview.html. 15.4.2016.

Unity Technologies. 2016v. Particle Systems http://docs.unity3d.com/Manual/ParticleSystems.html. 15.4.2016.

Unity Technologies. 2016w. Using Particle Systems in Unity. http://docs.unity3d.com/Manual/PartSysUsage.html. 15.4.2016.

Unity Technologies. 2016x. Audio Overview. http://docs.unity3d.com/Manual/AudioOverview.html. 15.4.2016.

Unity Technologies. 2016y. Creating and Using Scripts. http://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. 15.4.2016.

Unity Technologies. 2016z. MonoDevelop. http://docs.unity3d.com/Manual/MonoDevelop.html. 15.4.2016.

Unity Technologies. 2016å. Configure your Unity Pro Subscription. https://store.unity3d.com/subscribe. 15.04.2016.

Unity Technologies. 2016ä. Asset Server (Team License). http://docs.unity3d.com/Manual/AssetServer.html. 15.4.2016.

Unity Technologies. 2016ö. Using External Version Control Systems with Unity. http://docs.unity3d.com/Manual/ExternalVersionControlSystemSupport.html. 15.4.2016.

Unity Technologies. 2016a2. Object Pooling
https://unity3d.com/learn/tutorials/modules/beginner/live-training-archive/object-pooling. 15.4.2016.

Unity Technologies. 2016b2. Input Manager
http://docs.unity3d.com/Manual/class-InputManager.html. 15.4.2016.

Valve Corporation. 2016. What is Steam Greenlight? http://steamcommunity.com/workshop/about/?appid=765&section=faq. 15.04.2016.

Wawro, A. 2014. What exactly goes into porting a video game? BlitWorks explains. Gamasutra. http://gamasutra.com/view/news/222363/What_exactly_goes_into_porting_a_video_game_BlitWorks_explains.php. 13.1.2016.

Wikimedia. 2016. Vita Layout Map
https://upload.wikimedia.org/wikipedia/commons/8/89/PlayStation_Vita_Layout.svg. 15.4.2016.