FLASH-OHJELMISTOKEHITYS

Perustietopaketti peliohjelmoijille

Pasi Perkiö

Opinnäytetyö
Toukokuu 2012
Viestinnän koulutusohjelma
Vuorovaikutteisuuden suunnittelu

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Viestinnän koulutusohjelma
Vuorovaikutteisuuden suunnittelu

PASI PERKIÖ:
Flash-ohjelmistokehitys: Perustietopaketti peliohjelmoijille

Opinnäytetyö 35 sivua
Toukokuu 2012

Opinnäytetyöni tarkoitus on madaltaa kaikentasoisten peliohjelmoijien kynnystä Flash-pelinkehityksen aloittamiseen. Flash on helposti lähestyttävä, mutta suorituskyvyn kannalta välillä hyvin haasteellinen alusta, mutta mielestäni edelleen hyvin varteenotettava pelinkehitysalusta etenkin nettipelejä ajatellen.

Opinnäytetyöni mediatyönä olen ohjelmoinut Frozenbyten kehittämän Splot-pelin Flashille. Kyseisen projektin aikana olen oppinut paljon pelinkehityksen tyypillisistä haasteista ja kompastuskivistä, joita pyrin tämän opinnäytetyön kautta jakamaan.

Työn kirjallisessa osuudessa käsittelen Flash-ohjelmistokehityksen ja Flashin käyttökohteiden kannalta erityisen tärkeitä aihealueita, kuten kehitysympäristön valintaa, resurssienhallintaa ja optimointia. Luon myös melko hyvän kuvan Flashin yhteydessä käytettävän ActionScript 3 -skriptauskielen pääominaisuuksista.

Asiasanat: Flash-peliohjelmointi, ActionScript 3

THESIS SUMMARY

Tampere University of Applied Sciences
Media Programme
Interaction Design

PASI PERKIÖ:
Flash software development for game programmers

Final thesis 35 pages
May 2012

The purpose of this thesis is to serve as an introduction to developing games with Flash. It's very easy to get started with Flash, but it can be quite challenging in terms of achieving great performance. All in all, I argue that Flash is still a very good platform especially for web games.

For my thesis media project, I have been the sole Flash programmer for the Frozenbyte game Splot. During the development of the game, I've learned a lot about the typical challenges and pitfalls of game development that I will try to address through my thesis on some level.

In the written part of my thesis, I address the most important subjects that one must be familiar with while developing games with Flash. Subjects like choosing a development environment, managing resources and optimization. I also form a pretty good picture of ActionScript 3 - the scripting language used in Flash today.

Key words: Flash game development, ActionScript 3

SISÄLLYS

1 JOHDANTO

Flash on mielenkiintoinen pelinkehitysalusta. Monien kritisoima ja aliarvostama Flash on vihaajistaan huolimatta saanut viime aikoina paljon tuulta alleen myös kaupallisessa pelinkehityksessä. Flash-sovelluksiin törmää nykyään nettiselaimissa Facebookissa, Applen mobiililaittella ja myös työpöydällä Steamissa. Mutta mikä saa alustan pitämään päänsä pinnalla, vaikka jatkuvasti kaikki puhuvat sen kuolemasta?

Kun pari vuotta sitten mietin opinnäytetyölleni aihetta, tavoitteeni oli kirjoittaa jotain peliohjelmoijille. Mielessäni pyöri monia hieman edistyneempiä aihealueita, kuten renderöinti, tietokeskeisen pelimoottorin ohjelmointi ja optimointi. Voi olla että tuolloinen tieto-taitoni ei olisi vielä riittänyt noiden aiheiden asiantuntevaan käsittelyyn, joten päätin, että kirjoitan mieluummin jotain aloitteleville peliohjelmoijille. Samoihin aikoihin työllistyin Frozenbytelle Flash-peliohjelmoijaksi, joka pitkälti saneli myös opinnäytetyöni tarkemman aihealueen.

Opinnäytetyöni tarkoitus on luoda katsaus Flashiin ohjelmistokehitysalustana pelien näkökulmasta koskettamalla olennaisimpia peliohjelmoijan arkipäivään kuuluvia aihealueita, kuten resurssienhallintaa ja optimointia. Luon tekstissä myös lyhyen katsauksen Flashin käyttämään AS3-skriptauskieleen peliohjelmoijille, jotka tuntevat jo jonkin kielen, kuten C++. Kaikkea mahdollista en ole tähän tekstiin saanut kuitenkaan mahtumaan, joten esimerkiksi piirtopuoli jätetään kokonaan huomioitta aihealueen laajuuden vuoksi. Tämä jättää enemmän tilaa perusasioille.

Opinnäytetyöstä hyötyvät eniten jo hieman käsiänsä lianneet Flash-ohjelmoinnista kiinnostuneet peliohjelmoijat, jotka haluavat viedä alustatuntemuksensa seuraavalle asteelle. Opinnäytetyöni ei opeta ketään ohjelmoimaan, vaan oletan lukijoilta monissa kohtaa tekstiä jo valmiiksi hieman perustietoja ohjelmoinnista. Varsinaiset ohjelmakoodiesimerkit ovat melko harvassa, mutta kun niitä esiintyy, käytettävä skriptikieli on ActionScript 3.0. Aikaisemmat versiot kyseisestä kielestä jätän kokonaan huomiotta, koska niiden käytölle ei nykyään ole enää mitään järkevää syytä yhteensopivuuden eikä etenkään suorituskyvyn kannalta.

2 SANASTOA

Mikromaksu (engl. micropayment)
Pienikokoinen, yleensä alle 10 euron arvoinen, rahamaksu. Maksetaan tyypillisesti verkossa esimerkiksi PayPal-maksupalvelun kautta tai luottokortilla.

Pelisilmukka (engl. game loop)
Pelin sydän, jonka tehtävä on päivittää pelin tapahtumat, joka tyypillisesti ajetaan riippumatta pelaajan antamista syötteistä.

Porttaus (engl. porting)
Ohjelmiston sovittaminen toimimaan eri ympäristössä kuin mihin se on alkujaan suunniteltu ja ohjelmoitu.

Skriptikieli (engl. scripting language)
Ohjelmointikieli, joka tyypillisesti voidaan suorittaa lennosta ilman kääntämistä kohdealustan konekielelle. Sen sijaan skriptikieli käännetään eräänlaiseen välimuotoon, jota kutsutaan tavukoodiksi.

Syntaksi (engl. syntax)
Ohjelmointikielen kirjoitussäännöt/kielioppi.

Tavukoodi (engl. bytecode)
Skriptikielien kanssa käytettävä välimuoto lähdekoodin ja konekielen väliltä. Vaatii toimiakseen ajonaikaisen tulkin.

Tapahtumavetoinen ohjelmointi (engl. event driven programming)
Ohjelma kulkee eteenpäin tapahtumien kautta, joita syöttölaitteet kuten hiiri ja näppäimistö tai pelin sisäiset oliot lähettävät.

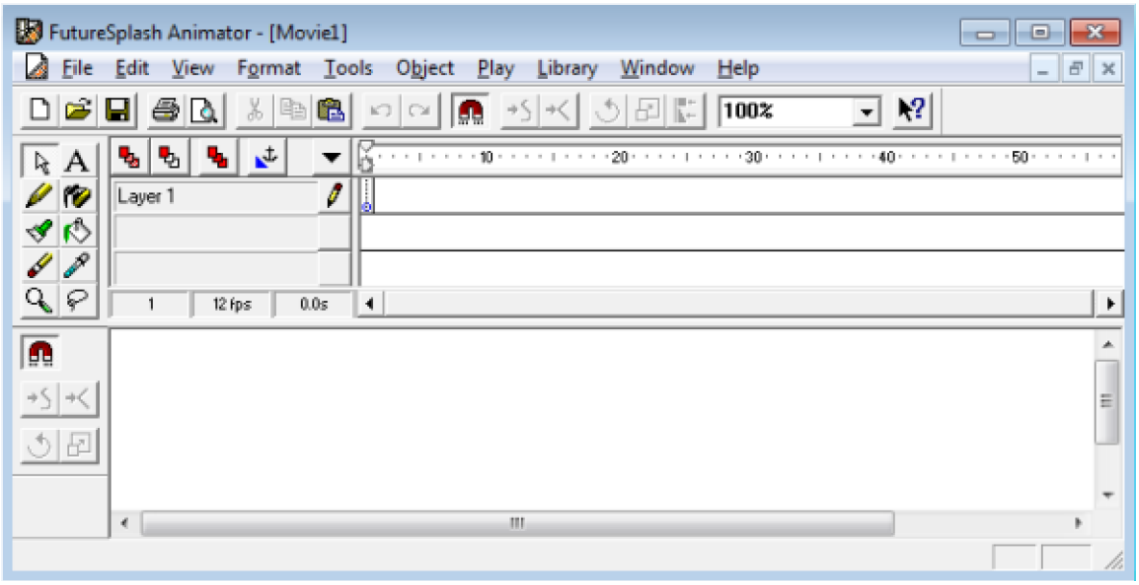
Vahva muuttujien tyypitys (engl. strong typing)
Asettaa rajoitteita sille, miten eri tietotyyppien välisiä operaatioita voidaan suorittaa. Käytännössä pakottaa ohjelmoijan määrittelemään muuttujille tietotyypin, mikäli se on valmiiksi tiedossa.

3 FLASHIN ROOLI PELITEOLLISUUDESSA

Flash on ollut pelialalla pitkään suosittu alusta etenkin peli-prototyyppejä varten, mutta se soveltuu viimeaikaisten teknisten uudistustensa ja uusien markkina-alueiden kuten Facebookin avauduttua paremmin myös kaupalliseen pelinkehitykseen. Kiitos suuren yhteisön ja kattavien koodikirjastojen, Flashilla on kaiken lisäksi helppo päästä alkuun, joten se soveltuu erinomaisesti kaikentasoisille peliohjelmoijille.

3.1. Sisällöntuotantotyökalu

Flashin aikajana alkaa vuodesta 1993, jolloin FutureWave Softwaren perustajakolmikko - Jonathan Gay, Gary Grossman ja Robert Tatsumi - asettivat tavoitteekseen helpottaa tietokoneavusteisen grafiikan luomista. Aika oli otollinen, sillä myös Internet teki kovasti tuloaan ja kaipasi kipeästi liikkuvaa kuvaa. Kolmen vuoden iteroinnin jälkeen tuloksena syntyi FutureSplash Animator -animaatiotyökalu, joka suuren suosion siivittämänä laajeni erillisen FutureWaven kehittämän selainlaajennoksen kautta nopeasti myös Internetiin elävöittämään alkujaan monien suosittujen tahojen verkkosisältöä kuten Disneyn ja Simpsoneiden kotisivuja. Puolen vuoden päästä työkalun julkaisusta, kilpailevaa Shockwave-teknologiaa kehittävä Macromedia osti FutureWaven ja uudelleennimesi työkalun Flashiksi (Gay, Grossman & Tatsumi 2008).



KUVA 1. FutureSplash Animator vuodelta 1996.

Flash on aina ollut vahva työkalu kaksiulotteisen animaation tuotannossa. Pelialalla esimerkiksi Super Meat Boy (2010) ja Plants vs. Zombies (2009) ovat hyödyntäneet Flashia nimenomaan animaatiotyökaluna. Super Meat Boyn kehittäjien (McMillen & Refenes 2011) mukaan kenttäeditorin lisäksi ainoa toinen tärkeä pelin kehitykseen käytetty sisällöntuotantotyökalu oli Flash, josta McMillenin piirtämät animaatiot käännettiin pelimoottorin ymmärtämään muotoon Refenesin ohjelmoimalla työkalulla. Tämän lisäksi myös pelin alkuperäinen prototyyppiversio tehtiin Flashilla (kuva 2).

KUVA 2. Vasemmalla: Meat Boy -peliprototyyppi. Oikealla: Super Meat Boy.

Hyviä työkaluja kaupallisen tason kaksiulotteiseen animaatioon on olemassa markkinoilla melko heikosti. Pieni kilpailu tekisi varmasti hyvää myös Flashille. Mikäli sopivaa työkalua ei löydy, turvautuu moni firma oman sisäisen työkalun kehittämiseen. Näin ongelmaa lähestyi esimerkiksi Ubisoft Rayman Origins -pelin kanssa, jota varten kehitetty UbiArt-teknologia saattaa tulevaisuudessa olla myös lisenssoitavissa (Orland 2011). Ohjelmoijana oman työkalun kehittäminen on aina houkutteleva ajatus, mutta ajankäytön suhteen se ei aina ole perusteltua suhteessa saavutettavaan hyötyyn.

Animaation lisäksi peliteollisuudessa Flashin vaikutusvalta ulottuu myös käyttöliittymäsuunnitteluun. Tätä edisti suurelta osin vuonna 2004 perustettu yritys nimeltään Scaleform, jonka vektorigrafiikka-piirtomoottori Scaleform GFx mahdollisti Flashissa kehitettyjen käyttöliittymien tuonnin konsolipeleihin (MobyGames: Scaleform 2012). Suurin osa kaupallisista korkean profiilin pelimoottoreista kuten Unreal Engine, CryEngine tukevat tänä päivänä Scaleformia (Autodesk Scaleform: Overview 2012). Tätä kautta Flashia on hyödynnetty esimerkiksi Battlefield 3, Batman: Arkham City ja Mass Effect 3 -pelien käyttöliittymien toteuttamisessa (kuva 3).

KUVA 3. Mass Effect 3 -pelin hahmonkehitys-käyttöliittymä.

3.2. Pelinkehitysalusta

Vakavasti otettavana pelinkehitysalustana Flashin edistys on ollut hitaampaa. Alun perin FutureWaven ja Macromedian alaisuudessa ollessaan, skriptaaminen ei ollut Flashissa ensimmäisten Flash-versioiden aikaan käytännössä edes mahdollista, jonka vuoksi monimutkaisempien vuorovaikutteisten sovellusten kuten pelien kehittäminen ei tullut kysymykseenkään (Greene 2007). Käytännön kokemuksesta Flash on myös aina paininut suorituskykyongelmien kanssa, jonka vuoksi sen käyttö vakavaan pelinkehitykseen on ollut aina hieman kyseenalaista.

3.2.1 ActionScriptin synty

Flashin vakiintunut JavaScriptistä vahvasti vaikutteita ottanut skriptikieli ActionScript astui kuvioihin vasta Flash 5 -version myötä vuonna 2000. Tätä ennen skriptaaminen oli rajoittunut animaatioiden aikajanojen kontrollointiin käskyillä kuten "gotoAndPlay" ja "gotoAndStop", jotka ovat säilyneet perintönä myös nykypäivään asti (Greene 2007).

ActionScriptin ensimmäinen versio: ActionScript 1.0 tai AS1 toi mukanaan muuttujat, ehtolauseet ja funktiot. Kielenä AS1 oli silti hyvin rajoittunut. Päällimmäisenä kompas-

tuskivenä AS1 ei varsinaisesti tukenut luokkia ja sitä kautta kunnolla olio-ohjelmointia, jota suurin osa ohjelmoijista oli tuolloin tottunut käyttämään muiden kielien kuten Java ja C++ kanssa. AS1 halusi olla helposti lähestyttävä, jonka vuoksi se oli kielellisesti hyvin anteeksiantava eikä vaatinut esimerkiksi muuttujien tyypin määrittelyä, vaan muuttujat saivat sisältää mitä tahansa tietoa. Ohjelmoijat varmasti ymmärtävät, että tällä on suora vaikutus kielen tulkinnan suorituskykyyn, mutta Flashin tuolloisia käyttökohteita silmälläpitäen tällä ei ollut niin paljon väliä. (Greene 2007).

Seuraava suurempi kehitysaskel: ActionScript 2.0 tai AS2 esiteltiin vuonna 2003 Flash MX 2004 -työkalun ja Flash Player 7 -version julkaisun yhteydessä. AS2 paikkasi aikaisemman version suurimmat puutteet: luokat ja vahvan muuttujien tyypityksen. Että siirtymä vanhaan syntaksiin tottuneille olisi ollut mahdollisimman kivuton, AS2 oli taaksepäin yhteensopiva aiempien kielen versioiden kanssa. Tästä johtuen AS2:n tulo ei parantanut Flashin suorituskykyä juuri yhtään. Samaan aikaan Flashia alettiin kuitenkin käyttää yhä raskaampien sovelluksien kuten pienien pelien kehittämiseen, eikä AS1-aikainen moottori yksinkertaisesti enää pysynyt perässä jatkuvista paikkailuista ja pienistä optimoinneista huolimatta. Aika oli otollinen aloittaa puhtaalta pöydältä ja laittaa koko kieli uusiksi (Greene 2007).

3.2.2 ActionScript 3.0

Mielestäni ActionScriptin historian merkittävien edistysaskel tapahtui Adoben ostettua Macromedian. Adoben alaisuudessa Flash kävi läpi suuren remontin, joka johti ActionScriptin nykyiseen kolmanteen versioonsa vuonna 2006. AS3 hyödyntää täysin uutta virtuaalikonetta nimeltään AVM2, joka kykenee tulkkaamaan AS-koodia jopa kymmenen kertaa nopeammin kuin aikaisemmin. Luvatun suorituskyvyn saavuttaminen tietenkin edellyttää, että ohjelmoijat noudattavat tiukkaa kuria esimerkiksi muuttujien tyypityksen suhteen ja käyttävät hyväkseen aina tilanteeseen sopivia tietorakenteita.

Uudesta järjestelmästä johtuen AS1- ja AS2-kielillä kirjoitettu ohjelmakoodi ei ole lainkaan yhteensopivaa AS3-koodin kanssa. Tämä siirtymä oli monille ei-ohjelmoijille melko kivulias, mutta kompromissina Adobe ei kuitenkaan pakottanut ketään sopeutumaan uuteen tyyliin, vaan mahdollisti myös vanhan skriptikielen käytön jatkamisen, kunhan sitä ei sekoittanut AS3-ohjelmakoodin kanssa. AS1/AS2-koodin käyttäminen oli

kirjoitushetkelläkin edelleen mahdollista, sillä se vain käännetään vanhan AVM1-virtuaalikoneen ymmärtämään muotoon.

(Brimelow 2008; Greene 2007)

3.2.3 Flash Player 11 ja Stage3D-rautakiihdytys

Vaikka AS3 oli merkittävä edistysaskel Flash-pelejä ajatellen, oli se silti monille pettymys, sillä se ei edelleenkään tarjonnut sitä, mitä pelinkehittäjät ehkäpä eniten kaipasivat vakuuttavamman näköisien pelien tekemiseen: rautakiihdytystä grafiikan piirtoon. Ilman tätä esimerkiksi kolmiulotteisen grafiikan piirtäminen oli Flashissa hyvin raskasta ja epäkäytännöllistä, sillä piirto suoritettiin käytännössä kokonaan prosessorin voimin.

Flash oli pahasti ajastaan jäljessä tämän suhteen aina vuoteen 2011 asti, kunnes Adobe viimein vastasi huutoon ja esitteli Stage3D-rajapinnan, joka houkutteli välittömästi myös suuret pelialan jätit mukaan Flash-kelkkaan (kuva 4). Stage3D avasi grafiikkaohjelmoijille kokonaan uuden leikkikentän: indeksipuskurit, verteksipuskurit, pikseli- ja verteksivarjostimet. Kaikki tämä kuulostaa varmasti tutulta jos on aikaisemmin ollut tekemisissä jonkin 3D-rajapinnan kuten DirectX:n tai OpenGL:n kanssa, mutta aihealueeseen perehtymättömillä menee varmasti helposti sormi suuhun (Scabia 2011).

KUVA 4. Unreal Engine 3 -teknologiaa esittelevä Infinity Blade -demo pyörii Flashilla.

Tätä silmällä pitäen Adobe oli koko Stage3D-rajapinnan kehitystyön ajan tiiviissä yhteistyössä aktiivisten Flash-kehittäjien kanssa. Tuloksena uuden Stage3D-rajapintaa tukevan Flash Playerin ilmestyttyä, kaiken tasoiset ohjelmoijat pääsivät heti ottamaan hyödyn irti uudesta piirtorajapinnasta avoimen lähdekoodin kirjastojen, kuten Starling kautta, tuntematta varsinaisesti miten Stage3D toimii alhaisella tasolla (Thibault 2011).

3.3. Levityskanavat

Flashin suosio on aina ollut korkea etenkin aloittelevien pelinkehittäjien keskuudessa, joiden synnyttämää tuotosten tulvaa vastaamaan myös levityskanavat seurasivat pian perässä. Koska Flash-sisältö pyörii nettiselaimessa, oli vain ajan kysymys milloin Flash-animaatioita ja -pelejä varten alkoi ilmestyä omia portaaleja. Näistä suosituimpiin lukeutuvat Kongregate ja Newgrounds (kuva 5), joiden avulla oman tekeleen jakaminen ilmaiseksi miljoonien ihmisten kanssa on hyvin vaivatonta.

KUVA 5. Newgrounds-portaalin peliosion etusivu.

Suositut Flash-pelit keräävät paljon silmäpareja, joka herätti nopeasti myös mainostajien kiinnostuksen. Hyvä esimerkki tästä on Flash-mainontaan erikoistunut yritys MochiMedia, jonka 2007 julkaistun MochiAds-latausruudun kytkeminen peliin on paitsi ilmaista, mutta myös lähes yhtä helppoa kuin itse pelin levittäminen portaaleihin. (Bibby 2007). Tästä syystä mainonnasta tuli harrastelija-pelinkehittäjille houkutteleva tapa kerätä peleillään pieniä tuloja.

Mainostulot jäävät kuitenkin yleensä hyvin pieniksi. Oma pieni Flash-pelini on kahdessa vuodessa tienannut MochiAdsin avulla hieman vajaat 200 euroa, johon on vaadittu noin puoli miljoonaa pelikertaa. Niinpä pelkkiin mainoksiin nojautuminen ei ole Flash-peleissä todennäköisesti kestävä ratkaisu kaupalliseen pelinkehitykseen. Ongelma on, että verkossa ihmiset eivät yleensä ole valmiita maksamaan sisällöstä kovin helpolla, vaan jotain täytyy aina saada ilmaiseksi. Tästä johtuen valtaosa verkossa pelattavista ei-mainosvetoisista Flash-peleistä käyttävät mikromaksuja, joilla pelin ollessa suurimmalta osin ilmainen, voivat pelaajat rahaa vastaan yleensä parantaa pelikokemustaan.

Facebookin saavuttua voidaan sanoa Flash-pelien kultakauden alkaneen. Miljardia käyttäjää tänä päivänä lähestyvä Facebook avasi ohjelmistonkehittäjille vuonna 2007 mahdollisuuden tarjota sen käyttäjille kaupallisia sovelluksia, joiden tuotoista Facebook ottaisi itselleen tietyn siivun. Tämä avasi valtavan kanavan uudentyyppisille sosiaalisille peleille, jotka hyödyntäisivät Facebookin ainutlaatuista käyttäjäkuntaa ja käyttäjien välisiä ystävyyssuhteita ja käyttäjätietoja, joita myös sovellukset pääsisivät tarkastelemaan pelaajien antaessa siihen luvan (Nutt 2009). Koska Flash oli Facebookin nousun aikoihin sen levinneisyyden vuoksi erittäin toimintavarma tapa tuoda interaktiivista sisältöä nettisivuille, valitsi suurin osa pelinkehittäjistä Flashin alustakseen, jota myös Facebook on tukenut (Elman 2009). Myös suomalainen Rovio toi 2012 alkuvuodesta huippusuositun Angry Birds -pelinsä Facebookiin käyttäen Flashia (kuva 6).

KUVA 6. Angry Birdsin Facebook-versio.

4 KEHITYSYMPÄRISTÖ

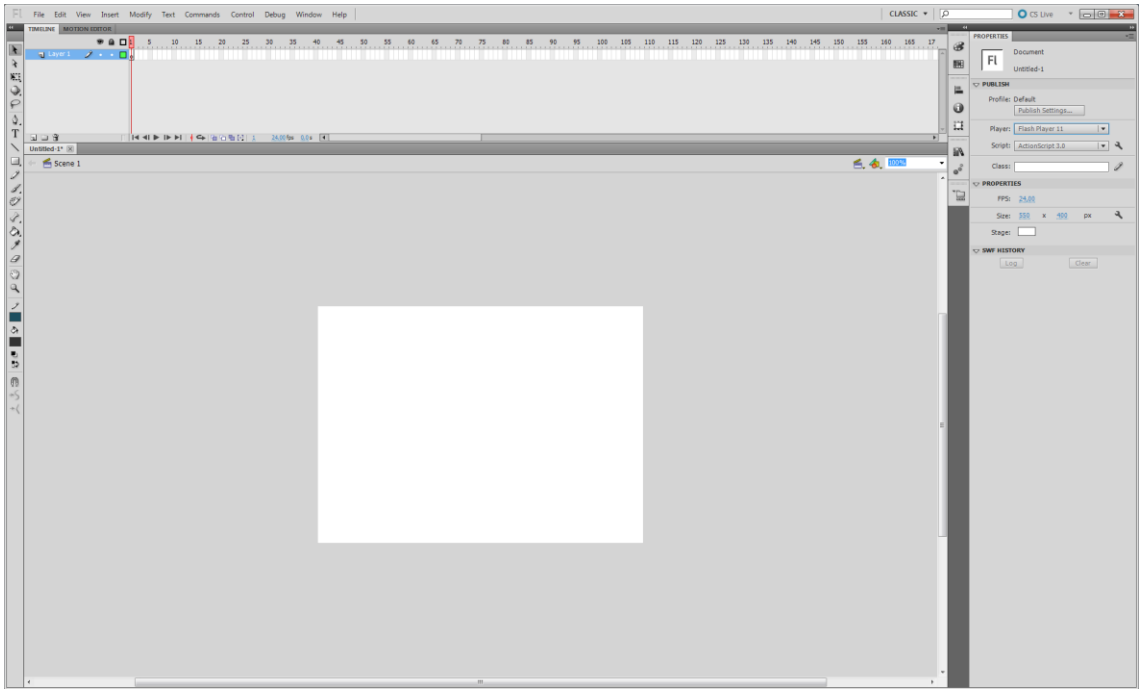
Flash-ohjelmistokehitys, kuten mikä tahansa ohjelmistokehitys, alkaa kehitysympäristön valinnasta. Hyvä kehitysympäristö on mielestäni tehokkaan ohjelmistokehityksen tärkein tukipilari. Kehitysympäristö ei tietenkään sellaisenaan riitä, vaan ohjelmoijan täytyy myös oppia hyödyntämään monipuolisesti kehitysympäristönsä tarjoamia edistyneempiä ominaisuuksia. Jos tätä ei vaivaudu ikinä tekemään, on melkein ihan sama mitä tekstieditoria käyttää ohjelmakoodin kirjoittamiseen.

4.1. Adobe Flash

Aloittelevalle Flash-ohjelmoijalle loogisin työympäristö on varmasti Flashin kehittäjän, Adoben oma kaupallinen Flash-työkalu (kuva 7). Kirjoitushetkellä uusin versio tästä oli Flash Professional CS5.5, joka on osa Adoben Creative Suite -ohjelmistopakettia Photoshopin, Illustratorin ja muiden Adoben ohjelmistojen rinnalla. Flash jakaa näiden kanssa paitsi samantyylisen käyttöliittymän, mutta kykenee myös tuomaan sisältöä suoraan esimerkiksi Photoshopista hävittämättä alkuperäisiä tasotietoja. Tämä yhteensopivuus voi nopeuttaa työskentelyä huomattavasti etenkin staattista sisältöä kuten käyttöliittymiä suunnitellessa.

Myös peliohjelmoinnin näkökulmasta Adoben työkaluilla on helppo päästä alkuun. Pienissä yhden tai kahden hengen projekteissa on monesti kätevää, että koko projekti ja kaikki sen käyttämät resurssit scripteistä ääniefekteihin elävät saman kehitysympäristön sisässä. Ongelma on pitkällä tähtäimellä se, että Adoben tarjoama kehitysympäristö on selvästi alkujaan suunniteltu animaatiota ja web-sisällöntuotantoa varten. Tästä syystä kokemuksen karttuessa Adobe Flash sen kaikkine aikajanoineen ja vektorityökaluineen alkaa nopeasti tuntua tarpeettoman suurelta kokonaisuudelta peliohjelmointiin. Projektien kasvaessa ja koodikannan paisuessa, sujuva kooditiedostojen välinen ja sisäinen navigointi, ohjelmavirheiden tehokas paikantaminen ja nopea pelin uudelleenkääntäminen tulevat entistä tärkeimmiksi. Pian Adobe Flash on vain tiellä. Isoissa projekteissa suuri miinus on etenkin Flash-sovelluksen kääntämiseen menevä aika, joka on omien kokemuksieni mukaan liikkunut jopa useissa kymmenissä sekunneissa. Adoben mukaan (Set Preferences in Flash 2012) kääntämisaikaa voi yrittää nopeuttaa välimuistilla, jolloin vältyttäisiin ainakin fonttien ja äänitiedostojen tarpeettomilta uudelleenpakkaamisilta.

Itse tutustuin ActionScript-ohjelmointiin myös Adobe Flashin kautta, mutta siirryin hyvin nopeasti vaihtoehtoisiin työkaluihin. Adoben kehitysympäristö on siitä huolimatta edelleen varteenotettava vaihtoehto esimerkiksi käyttöliittymien varten. Nettisivuja ja pelien valikoita ohjelmoitaessa suorituskyky ei ole avainasemassa, jolloin Adobe Flashin WYSIWYG-lähestymistapa saattaa nopeuttaa kehitystyötä.

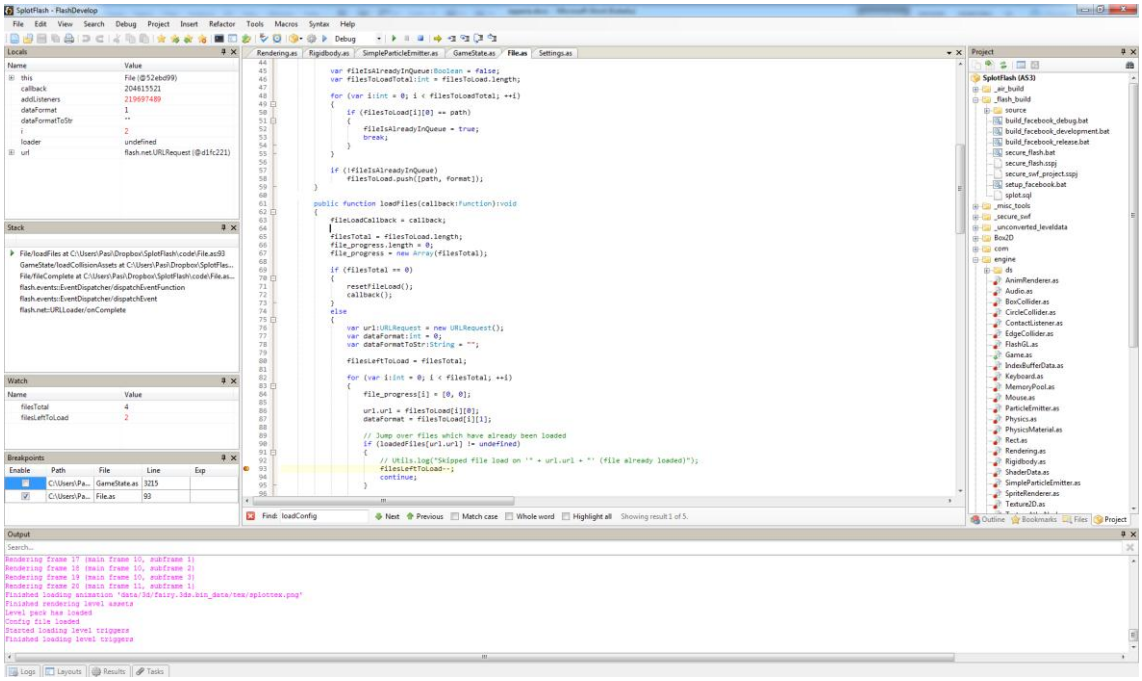


KUVA 7. Adobe Flash Professional 5.5 ja ActionScript 3.0 projektin aloitusnäkymä.

4.2. FlashDevelop

Parempi vaihtoehto vakavaan Flash-ohjelmistokehitykseen on avoimen lähdekoodin ActionScript-kehitysympäristö: FlashDevelop. Edelleen aktiivinen ja jo yli 7 vuotta - eli koko ActionScript 3 -kielen elinkaaren ajan - olemassa ollut FlashDevelop-projekti on ollut oma valintani jo kahden vuoden ajan. Se on parempi paitsi ominaisuuksiltaan, nopeuttaen ohjelmointityötä huomattavasti esimerkiksi aktiivisella kooditäydentäjällä (FlashDevelop Features: Completion 2012), mutta ennen kaikkea se on ilmainen. ActionScript on kielenä spesifikaatioineen avoimen lähdekoodin alainen, jonka vuoksi sitä varten on saatavilla myös ilmaisia kääntäjiä kuten Adoben Flex SDK. Myös FlashDevelop kääntää Flash-ohjelmat tavukoodiksi tämän avulla. Huono puoli FlashDevelopissa on, että se toimii ainoastaan Windowsilla, joten Mac-käyttäjien täytyi ainakin kirjoitushetkellä vielä purra huulta tai tutustua muihin vaihtoehtoihin kuten Eclipseen.

FlashDevelop vastaa ominaisuuksiltaan ammattimaisia kehitysympäristöjä kuten Visual Studiota. Ohjelmistokehitystä ajatellen tärkein yksittäinen valtti on tehokas virheiden etsintä -työkalupakki, johon sisältyy pysäytyskohdat, kutsupino ja muuttujien arvojen tarkkailuikkuna (kuva 8). Myös virallinen Adoben kehitysympäristö kykenee virheiden etsintään, mutta FlashDevelopissa se vain tuntuu luontaisemmalta.



KUVA 8. FlashDevelop-kehitysympäristö virheiden etsintä -tilassa.

Pysäytyskohdat luovat perustan kaikelle virheiden etsinnälle. Niiden avulla ohjelman suoritusta ja muuttujien varastoimia arvoja voidaan tarkastella suurennuslasin kanssa. Pysäytyskohta määrää ohjelman keskeyttämään ohjelmakoodin suorituksen halutun koodirivin kohdalla, jonka jälkeen suoritusta voidaan jatkaa rivi kerrallaan. Mikäli kulloinkin aktiivisella rivillä kutsutaan funktiota, voidaan sen sisään halutessa astua. Oletusarvoisesti funktiokutsut suoritetaan täydellä nopeudella siten, että seuraavan kerran pysähdytään vasta funktiokutsun jälkeisen rivin kohdalla.

Kutsupino on pysäytyskohtien kanssa käytettävä apuväline, joka näyttää kyseisellä hetkellä voimassa olevan funktioketjun tai -polun, mitä kautta pysäytyskohtaan on päädytty. Näin ongelman lähde voidaan mahdollisesti paikantaa tutkimalla ylemmän tason funktioita.

Astellessasi koodisi läpi ja hypellessäsi kutsupinoa ylös ja alas, haluat varmasti myös pitää silmällä tiettyjen muuttujien arvoja. Tätä varten on olemassa **tarkkailuikkuna**,

johon yksinkertaisia muuttujia ja monimutkaisiakin luokkia voidaan asettaa seurattavaksi. FlashDevelopissa - kuten monissa muissakin kehitysmpäristöissä - muuttujien arvoja voidaan tarkastella myös suoraan pysäytetyn ohjelmakoodin seasta.

Kukaan ei kirjoita täydellistä koodia ja pelkästään pieniä inhimillisiä virheitä sattuu kymmeniä päivässä. Yllättävän suuri osa ohjelmoijan työstä on itse asiassa virheiden korjaamista. Niinpä ilman hyviä virheiden etsintätyökaluja työnteko hidastuu huomattavasti. Tätä vastaan hyvän kehitysympäristön tulisi taistella auttamalla paikantamaan ja korjaamaan virheet mahdollisimman nopeasti. Virheiden etsintä on yksi tärkeimmistä taidoista, mitä ohjelmoija voi osata (Gregory 2009, 78.).

4.3. Selainlaajennokset

Valitettavasti jotkut ohjelmavirheet nostavat päätään vasta kehitysympäristön ulkopuolella julkaisuversiossa; Flashin tapauksessa tyypillisesti nettiselaimessa. Ohjelmavirheiden metsästäminen on usein huomattavasti työläämpää, kun enää ei voida turvautua kehitysympäristön virheiden etsintätilan suomiin apukeinoihin virheiden paikantamiseksi. Käytettävissä on Flasin tapauksessa yleensä vain ohjelmoijan itsensä koodin sekaan asettamat tekstitulosteet, joiden avulla voidaan kömpelösti, mutta onnistuneesti tarkastella muuttujien arvoja. Nämä tulosteet ilmestyvät jonkinlaiseen konsoliin, tekstitiedostoon tai molempiin riippuen implementaatiosta.

Flash-pelejä kehitettäessä kätevintä on jos tämä konsoli löytyy suoraan nettiselaimesta. Esimerkiksi Firefoxia varten on olemassa lisäosa nimeltään Flashfirebug, joka itsessään on lisäosa monipuoliselle Firebug-työkalulle. Tämän avulla Flash-peli voi lähettää tulosteita suoraan Firefoxin yhteydessä olevaan konsoli-ikkunaan. Esimerkiksi Facebook-integraation kanssa selaimen virheiden etsintä konsolit tulevat hyvin läheisiksi Flash-pelinkehityksessä.

5 ACTIONSCRIPT 3 PÄHKINÄNKUORESSA

5.1. Lyhyt oppimäärä C++ ohjelmoijille

Yksi pelialan käytetyimmistä ohjelmointikielistä on C++, jota ActionScript 3.0 monen muun olio-ohjelmointikielen, kuten Javan, tapaan muistuttaa hyvin paljon syntaksiltaan. Ominaisuuksiltaan ja toiminnaltaan AS3 ja C++ eroavat kuitenkin merkittävästi toisistaan. Listaan nyt asiasta kiinnostuneille tärkeimmät merkille pantavat eroavaisuudet, jotka tulisi ottaa huomioon siirryttäessä AS3-maailmaan.

Flash on tapahtumavetoinen ympäristö. Ohjelmoija ei suoraan hallitse sovelluksensa viestisilmukkaa, vaan kaikki tapahtuu vastakutsujen kautta (Brown 2006). Jotta Flashin tapahtumiin voisi jotenkin reagoida, täytyy niitä ensin kuunnella. Toisin sanoen ohjelmoija asettaa jollekin Flashin tapahtumalle - kuten näppäimistön painallukselle - vastakutsu-funktion, jota kutsutaan tapahtumaan liittyvällä tiedolla aina kun se käy toteen. Hyvänä esimerkkinä pelin pääsilmukkaa ei ajeta ikuisen while-silmukan sisässä, vaan Flash-ohjelmalle asetetaan tiukka ruudunpäivitysnopeus-tavoite, jonka jälkeen ruudunpäivitys-tapahtumaan reagoidaan vastakutsun kautta.

Flash ei tue säikeitä. Vaikka ohjelmakoodi ajetaan pääasiassa yhdellä säikeellä suorittimesta riippumatta (Brown 2006), käyttää Flash sisäisesti säikeitä esimerkiksi asynkroniseen tiedostojen lataamiseen.

Flashissa ei ole enum-tietotyyppiä. Tämän seurauksena esimerkiksi olemassa olevan C++-ohjelmakoodin porttaaminen Flashiin vaikeutuu hieman. Käytännössä rajoituksen kiertämiseen on olemassa monia enemmän tai vähemmän arveluttavia ja rumia keinoja, joten tarpeen tullen enum on teoriassa mahdollinen myös Flashissa (Brown 2006).

Flash ei tue operaattoreiden ylikuormittamista. Sano hyvästit kauniin näköisille vektorien välisille laskutoimituksille.

Flashissa on automaattinen roskienkerääjä. Ohjelmoijalla ei ole vastuuta varatun muistin vapauttamisesta, vaan Flash pyrkii parhaansa mukaan tekemään tämän automaattisesti. Ohjelmoijan tulisi silti antaa roskienkerääjälle tarvittavia vihjeitä, että se voisi hoitaa hommansa mahdollisimman hyvin.

5.2. Tietotyypit

ActionScript 3.0 perustietotyypit ovat:

- Boolean: tosi tai epätosi.
- int: 32-bittinen kokonaisluku.
- uint: 32-bittinen positiivinen kokonaisluku.
- Number: 64-bittinen IEEE-754 standardin mukainen liukuluku.
- String: UTF-16 merkkijono.
- Object: pohjaluokka, josta kaikki muut luokat (class) periytyvät.
- null: oletusarvo määrittelemättömille luokille ja merkkijonoille.
- undefined: oletusarvo määrittelemättömälle ja tyypittömälle tiedolle.

(ActionScript language and syntax 2012)

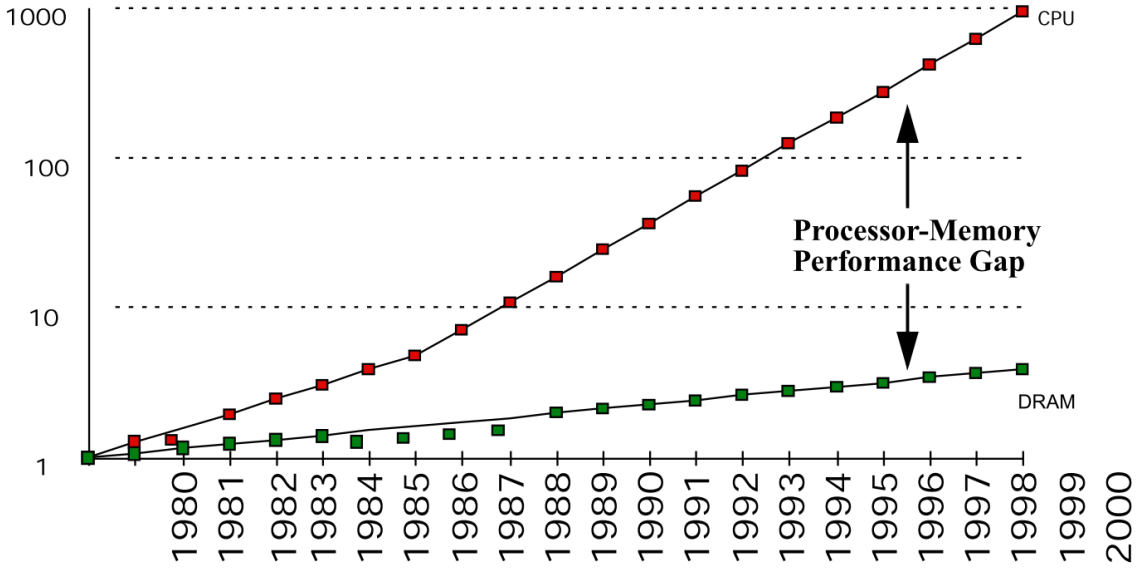
Ennen nykyistä 3.0 versiota, Number-tietotyyppiä käytettiin ActionScriptissä sekä kokonaislukuja että liukulukuja varten. Käytännössä siis kaikki luvut nähtiin liukulukuina. Suorituskyvyn kannalta tämä ei ollut optimaalinen ratkaisu, sillä kokonaislukuja varten on prosessoreissa oma laskentayksikkönsä, jonka aritmetiikka on kokonaislukujen yksinkertaisemman luonteen vuoksi nopeampaa verrattuna liukulukuihin. ActionScriptin 3.0 -remontin yhteydessä tämä suorituskykypullonkaula korjattiin.

Yksi silmään pistävä asia tosin on edelleen 32-bittisen liukuluku-tietotyypin puuttuminen. Monista muista kielistä tällainen löytyy nimellä "float", jolloin 64-bittinen liukuluku tyypillisesti kantaa nimeä "double". Peliohjelmoinnissa on erittäin harvoin tarvetta 64-bittisen liukuluvun suomalle tarkkuudelle. 32-bittisellä liukuluvulla paitsi puolitettaisiin tilantarve, mutta myös todennäköisesti nopeutettaisiin lukujen käsittelyä, sillä suurin osa suorittimista on edelleen luonteeltaan 32-bittisiä ja pienikokoisempi luku tietäisi yksinkertaisesti vähemmän töitä suorittimen liukulukulaskentayksikölle (Gregory 2009, 105.).

5.3. Muistinhallinta

Tänä päivänä suurien pelien nopeuteen vaikuttavat paitsi käytetyt algoritmit ja niiden implementointi, mutta kasvavissa määrin erityisesti se, miten peli hyödyntää muistia.

Pelin suorituskyvyn maksimoimisen perusedellytys on pitää käytettävissä olevat suorittimet aina työllistettynä. Ongelmana on, että jokainen suorittimen käsittelemä bitti joutuu kulkemaan yhden tai useamman muistin läpi, joiden nopeuskehitys ei ole pysynyt suorittimien kelkassa. Tämä kuilu on kasvanut 80-luvulta lähtien tasaisesti erittäin vakavaksi pullonkaulaksi (kuva 9). Tästä syystä prosessori saattaa viettää suuren osan ajastaan odottaessaan tarvitsemaansa dataa toimettomana. Jos suoritin olisi urheiluauto, vastaisi muisti liikennevaloja. Tätä vastaan on taisteltu eritasoisilla suorittimen yhteyteen suunnitelluilla välimuisteilla, jotka pyrkivät säilyttämään hiljattain käytetyt ja lähitulevaisuudessa mahdollisesti tarvittavat muistilohkot mahdollisimman lähellä suoritinta (Garney & Preisz 2010, 58.). Mutta koska Flash ei tue säikeitä ja muistinhallintakin on automaattista, on tätä varten vaikea, ellei mahdoton optimoida.



KUVA 9. Prosessorin ja muistin nopeuskehitys 1980-2000 (Patterson ym. 1997).

Alemman tason ohjelmointikielissä, kuten C ja C++, ohjelmoija ottaa täyden vastuun siitä, milloin ja mistä muistia varataan ja miten paljon. Kun muistia ei enää tarvita, muistilohko vapautetaan ohjelmoijan käskystä takaisin uudelleenkäytettäväksi. Tällä tavalla taitava ohjelmoija pystyy hallitsemaan muistia suorituskyvyn kannalta parhaalla mahdollisella tavalla. Alhaisen tason muistinhallinta tuo mukanaan myös suuren vastuun. Pienetkin huolimattomuusvirheet, kuten muistin vapauttamatta jättämiset, johtavat helposti ohjelman tai jopa koko järjestelmän kaatumiseen.

Flashissa muistinhallinnasta pitää huolen automaattinen roskienkerääjä. Tästä johtuen, Flashissa ei myöskään ole osoittimia, joten ohjelmoija ei pysty tarkalleen tietämään,

mistä kohtaa muistia tilaa varataan. Muistia käytännössä vain pyydetään roskienkerääjältä, joka palauttaa sitä tarvittavan määrän. Tämän jälkeen automaattisen roskienkerääjän päätehtävä on päättää milloin varattua muistilohkoa ei enää käytetä ja se voidaan vapauttaa (Yaiser 2011). Ongelma on, että roskienkerääjä ei voi mitenkään ymmärtää ohjelmasi toimintaa niin pitkälle, että se osaisi suoraan päätellä milloin jonkun tiedon säilyttäminen muistissa on muuttunut tarpeettomaksi. Roskienkerääjässä on siis omat hyötynsä, mutta myös omat haittapuolensa, jonka vuoksi se ei voi milloinkaan olla yhtä tehokas kuin kokonaan ohjelmoijan vastuulle jätetty muistinhallinta.

Flashissa automaattinen roskienkerääjä toimii pääperiaatteeltaan pitämällä kirjaa eri muistilohkojen referenssiviittausten määrästä. Jos siis ohjelmakoodissa mikään muuttuja ei enää viittaa aikaisemmin varattuun muistilohkoon, voidaan muistilohko merkitä vapautettavaksi. Vapautettavaksi merkityt muistilohkot käydään puhdistamassa muistista aika ajoin. Ohjelmoija ei voi mitenkään vaikuttaa näiden puhdistusoperaatioiden ajankohtiin, vaan roskienkerääjä itsenäisesti päättää niiden ajoituksesta (Yaiser 2011). Käytännön kokemuksesta roskienkerääjä herää kyllä viimeistään silloin, kun järjestelmästä alkaa muisti loppua.

Ohjelmoijan vastuulle jää siis turhien referenssien huolellinen mitätöiminen asettamalla käyttämättömät muuttujat null-arvoisiksi, tuhoten referenssin. Poikkeuksena ovat niin sanotut ”heikot referensit”, joita roskienkerääjä ei ota huomioon referenssilaskuissaan (Yaiser 2011). Heikkoja referenssejä voidaan hyödyntää esimerkiksi Dictionary-tietorakenteen ja tapahtumakuuntelijoiden yhteydessä. Jos referenssiviittaukset ovat heikkoja referenssejä lukuun ottamatta mitätöity, voi roskienkerääjä iloisesti vapauttaa muistilohkon.

5.4. Tietorakenteet

Array ja **Vector** ovat Flashin tarjoamat taulukko-tietorakenteet. Erona näillä on, että Arraylle ei voi määritellä vahvaa tietotyyppiä, joten käytännössä ohjelmoijien tulisi aina käyttää Vectoria, joka on pohjimmiltaan vain monipuolisempi versio Arraysta (Optimizing Performance… 2012, 34.). Array on lähinnä olemassa taaksepäin yhteensopivuus syistä, sillä Vector-tietorakenne esiteltiin vasta AS3-kielen yhteydessä. Vectorille voi määritellä sekä sen varastoiman tietotyypin, mutta tarvittaessa myös kiinteän koon.

Dictionary on Flashin versio hajautustaulusta, jonne tieto varastoidaan avain-arvo-pareina. Avaimelle ei tosin määritellä vahvaa tietotyyppiä, vaan samassa Dictionaryssa voi olla avaimia eri tietotyypeistä.

Mikäli kaivataan enemmän erilaisia tietorakenteita, kuten linkitettyjä listoja tai suorituskykyisempää hajautustaulua, on yksi vaihtoehto aina tietorakenteen implementointi itse. Näin teki myös Flash-blogia ylläpitävä Michael Baczynski, joka on jakanut hyvä nipun kaikenlaisia tietorakenteita kaikkien saataville (Baczynski 2012).

5.5. Kääntäminen

Ennen kuin ActionScriptillä kirjoitettu lähdekoodi voidaan suorittaa, täytyy se ensin kääntää Flashin ajoympäristön virtuaalikoneen ymmärtämään binäärimuotoon: ABC-tavukoodiksi (ActionScript bytecode). Kaikki projektin tarvitsemat kooditiedostot käyvät läpi tämän käännöksen, jonka tulokset pakataan yhteen tai useampaan SWF-tiedostoon (Moock 2007, 5.). Matkan lähdekoodista tavukoodiksi ja tavukoodista SWF-tiedostoon suorittavat Adoben kääntäjät, jotka tulevat lähtökohtaisesti valmiina kaikkien Flash-kehitysympäristöjen kuten FlashDevelopin ja Adoben omien Flash-työkalujen asennusten mukana. SWF-tiedostot voivat päätyä sisältämään suoritettavan ohjelmakoodin lisäksi myös muita lähdetiedostoja kuten ääniä ja grafiikkaa, mutta niistä lisää myöhemmin erillisessä osiossa (Resurssienhallinta, 22.).

SWF-tiedostoa ajettaessa, tavukoodi käy läpi vielä toisenkin käännöksen suoritettavan alustan ymmärtämälle konekielelle. Tämä käännös suoritetaan lennosta ja sitä kutsutaan ajonaikaiseksi kääntämiseksi tai JIT:ksi (engl. just in time compilation) (Moock 2007, 5.). Ajonaikaista kääntämistä hyödyntäviä ohjelmointikieliä voidaan kutsua myös "tulkattaviksi kieliksi". Tämä kategorian ulkopuolelle kuuluvista kielistä käytetään nimitystä "käännettävät kielet". Tällaisia kieliä ovat esimerkiksi konsolipelinkehityksessä käytettävät C ja C++, jotka käännetään konekielelle jo ennen ohjelman suorittamista.

Tulkattavuudesta on paljon hyötyä levityksen kannalta. Sen ansiosta Flash-ohjelmat voidaan suorittaa potentiaalisesti jokaisella olemassa olevalla alustalla ja käyttöjärjestelmällä edellyttäen, että Adobe on kehittänyt tulkin ympäristöä varten. Flashin tapauksessa tämä tulkki elää perinteisesti selaimen lisäosana, mutta vuodesta 2008 lähtien

Adobe Air nimisen ajoympäristön kautta Flash-sovelluksia voidaan ajaa lisäksi työpöydältä ja Applen iOS-laitteista käsin.

Tulkkauksessa on myös monia ilmiselviä haittapuolia suorituskyvyn kannalta. Itse JIT-kääntämisprosessi vie tietysti jo oman aikansa, mutta tämän lisäksi kääntäjän tarkoitus on myös tuottaa mahdollisimman optimaalista konekieltä kohdealustaa varten. Normaalisti kääntäjällä on tämän saavuttamiseksi aikaa jopa useita minuutteja ja resursseina roppakaupalla muistia, joita käyttää ohjelmakoodin suorituskyvyn analysointiin. Ajonaikainen kääntäjä on tähän nähden melkoisessa alakynnessä, sillä aikaa saattaa olla käytettävissä vain yksi millisekunti ja hyvin pieni määrä muistia (Garney & Preisz 2010, 305.). Ohjelmoija voi auttaa optimoimalla koodin mahdollisimman pitkälle lähdekoditasolla.

6 RESURSSIENHALLINTA

Flashissa resurssienhallinta - eli pelin käyttämien kuva-, ääni- ja muiden resurssien lataaminen pelin käytettäväksi - voidaan hoitaa karkeasti kahdella eri tavalla.

6.1. SWF-tiedostoon upottaminen

Resurssien upottaminen SWF-tiedostoon helpottaa pelin levittämistä, koska ainoa pelin ajamiseen tarvittava tiedosto on parhaassa tapauksessa yksi SWF-tiedosto. Riippuen tiimisi sisällöntuotanto-vaiheista, upotetut resurssit myös kasvattavat iterointiaikaa huomattavasti pelin kehitysvaiheessa, koska koko peli täytyy kääntää aina uudestaan jokaisen pienenkin sisältömuutoksen päivittymiseksi peliin (Resource Handling… 2010). Upottaminen ei siis ole kovinkaan käytännöllistä. Tämän lisäksi upotetut resurssit kasvattavan pelin latausaikaa huomattavasti, joka on melkoista haaskausta ottaen huomioon, että peli harvoin tarvitsee kaikkia resursseja heti alusta lähtien.

Upottaminen on joskus myös ihan järkevää. Esimerkiksi pelin käyttöliittymän kehitykseen Adoben oma Flash-kehitysympäristö toimii mainiosti ja koska käyttöliittymän käyttämiä kuvaresursseja varmasti tarvitaan joka pelikerralla, on näiden upottaminen ihan perusteltuakin. Mikäli pelin kääntämiseen käytetään Adoben omaa Flash-työkalua, on resurssien upottaminen hyvin vaivatonta. Flash upottaa ja kompressoi kaikki projektissa käyttämät ääni- ja kuvatiedostot SWF-tiedoston luontivaiheessa projektin asetusten mukaisesti automaattisesti mukaan.

6.2. Ulkoisten resurssien lataaminen

Käyttöliittymän ulkopuolisia resursseja varten paljon järkevämpi lähestymistapa on resurssien lataaminen ulkoisista tiedostoista tarpeen tullen esimerkiksi kentänvaihdoksen yhteydessä. Näin pelin resurssien lataamisen tuomaa taakkaa pelipalvelimelle voidaan helpottaa merkittävästi.

Erilaisten ulkoisten resurssin lataamiseen on Flashissa olemassa Loader-olioita, jotka lataavat tiedostot asynkronisesti häiritsemättä pelin kulkua ja kutsuvat jotain ohjelmoijan määrittelemää vastakutsu-funktiota kun lataus on valmis. Tähän ohjelmoija voi rea-

goida haluamallaan tavalla. Asynkronisen luonteensa vuoksi on käytännössä mahdollista ladata taustalla jo valmiiksi esimerkiksi seuraavan kentän sisältöä, joka taas saa pelin kentän välisten siirtymät vaikuttamaan sulavammilta pelaajan näkökulmasta. Loaderit myös lähettävät tiedoston lataamisen aikana tapahtumien kautta päivitystietoja latauksen edistymisestä, jota taas voidaan hyödyntää latausruutujen päivittämiseen.

7 YLEINEN TIEDONSIIRTOMUOTO

Pelit kannattaa suunnitella mahdollisimman tietokeskeisiksi. Tämä tarkoittaa sitä, että kaikki pelin moottorin ulkopuolinen, pelkästään kehitettävään peliin liittyvä sisältö ja pelin pelattavuuteen vaikuttavat arvot eivät tulisi olla kovakoodattuna pelin ohjelmakoodiin. Sen sijaan tällainen data tulisi lukea ulkoisista tiedostoista, joihin pelin designerit ja sisällöntuottajat pääsevät käsiksi (Gregory 2009, 252.). Tämäntyyppisen tiedon tallentamiseen on olemassa monia standardeja, joista suosituimpia ovat XML ja JSON.

7.1. XML

XML on kauan olemassa ollut standardi, jota käytetään laajalti tiedonvälitykseen eri järjestelmien välillä, mutta myös tiedostomuotona esimerkiksi taulukkolaskentaohjelmissa. XML-tiedosto koostuu sisäkkäisistä elementeistä, joihin tieto voidaan sijoittaa joko attribuutteihin tai elementtien sisään. Näiden kahden tyylin sekoittamista tulisi välttää, sillä se haittaa luettavuutta ja monimutkaistaa tiedon jäsentämistä.

XML soveltuu hyvin tekstidokumenttien sisällön esittämiseen, mutta pelitiedon tietorakenteiden kuvaamiseen on olemassa mielestäni parempiakin vaihtoehtoja. XML-tiedostoista tulee XML:n syntaksin vuoksi helposti melko tunkkaisen oloisia. Siitä huolimatta jotkut pelit liikuttavat tietoa XML-muodossa. ActionScriptin koodikirjastot ovat sisältäneet XML-tiedostojen parsimiseen tarvittavat funktiot AS3-kielen esittelystä lähtien (Moock 2007, 397.).

```
<viholliset>
   <käärme>
      <avunhuuto>Pzz</avunhuuto>
      <koko x=”5” y=”8”></koko>
      <nopeus>15.5</nopeus>
   </käärme>
   <örkki>
      <avunhuuto>Örr</avunhuuto>
      <koko x=”15” y=”15”></koko>
      <nopeus>7.0</nopeus>
   </örkki>
</viholliset>
```

7.2. JSON

JSON on kevyemmän ja ilmavamman syntaksinsa ansiosta luettavampi kuin XML. Sen tietotyypit myös vastaavat suoraan ohjelmoinnin perustietotyyppejä: kokonaisluvut, liukuluvut, merkkijonot ja taulukot. Myös JSON-tiedostojen parsiminen onnistuu nykyään ActionScriptin oletus-koodikirjastojen avulla.

```
{
   “viholliset” :
   {
      “käärme” :
      {
         “avunhuuto” = “Pzz”,
         “koko” = [5, 8],
         “nopeus” = 15.5
      },
      “örkki” :
      {
         “avunhuuto” = “Örr”,
         “koko” = [15, 15],
         “nopeus” = 7.0
      }
   }
}
```

8 OPTIMOINTI

Myös optimointi on tärkeä osa peliohjelmointia. Optimoinnin tarkoitus on helpottaa tietokoneen taakkaa pelin prosessorikuormitusta ja muistinkäyttöä vähentämällä ja latausaikoja pienentämällä. Optimoidulla pelimoottorilla ja ohjelmakoodilla ehkäistään kuormituspiikkejä ja kaatumisia, ruudunpäivitysnopeus paranee ja resursseja jää enemmän käytettäväksi muun pelin pyörittämiseen - tavoitteena sujuvampi ja miellyttävämpi pelikokemus. Mitä enemmän suorituskykyä on käytettävissä, sitä enemmän löytyy potentiaalia tarkemmalle fysiikanmallinnukselle, tyylikkäämmille partikkeli- ja muille visuaalisille efekteille, älykkäämmälle tekoälylle, mutta ennen kaikkea paremmalle pelille.

8.1. Profilointi

Optimointi ilman benchmarkkausta ei ole optimointia. Tämä korostuu Flashilla ohjelmoitaessa, sillä suorituskyky vaihtelee alustan (PC, Mac), ohjelmointiympäristön (Air, Flash Player) ja Flash Playerin version (9, 10, 11) välillä. Kaikki tulokset tulisi aina osoittaa konkreettisilla testeillä, jotka pystyvät mittaamaan optimoinnin kohteena olevan toimenpiteen resurssienkäyttöä. Pelkkiin vaistoihin luottaminen siinä, mikä on nopeaa ja mikä ei, voi helposti johtaa jopa suorituskyvyn heikkenemiseen.

Tärkein asia ennen optimointiurakkaan lähtemistä on selvittää, missä osassa ohjelmakoodia vietetään suurin osa suoritusajasta (Optimizing Performance… 2012, 3.).. Suurin ilmeinen säästö piilee tyypillisesti silmukoissa, jotka käydään läpi useita kertoja sekunnissa. Mutta jälleen kerran: älä luota vaistoihisi vaan selvitä konkreettisin keinoin pullonkaula ennen optimointiin ryhtymistä.

Yksittäisen funktion, silmukan tai muun koodinpätkän suoritusaikaa voidaan tarkastella yksinkertaisella ajastimella. Tähän tarkoitukseen voidaan käyttää Flashissa esimerkiksi getTimer() -funktiota, joka palauttaa pelin alkamisesta kuluneen ajan millisekunteina. Itse tapaan hajottaa testattavan kohdan eri versiot (alkuperäinen ja muutetut) omien funktioidensa taakse, joita kutakin iteroin tasavertaisesti ottamalla samalla aikaa. Jotta testi olisi merkityksellinen, iteraatioiden määrän täytyy olla tarpeeksi korkea. Nyrkkisääntönä testin tulisi kestää muutama sekunti. Mikäli haluaa tarkempaa tietoa Flashin-

sovelluksen suorituskyvystä, on Flash-kommuuni ajan kanssa kehittänyt monenlaisia erilaisia ilmaisia profilointi-työkaluja, joiden käyttö on erittäin suositeltavaa (Optimizing Performance… 2012, 91.).

8.2. Tiedostokoko

Flash pelit kehitetään tyypillisesti verkon ylitse pelattaviksi. Peli sijaitsee tällöin fyysisesti pelinkehittäjän tai -julkaisijan palvelimella, josta se joudutaan lataamaan pelaajan koneelle yhdessä tai useammassa osassa pelaamisen aikana. Tästä johtuen pelin ollessa menestys, pelaajasuma helposti ylittää peliä varten varattujen palvelinten kapasiteetin. Suuren kävijämäärän pitäisi olla onnenpäivä, mutta se voi pahimmassa tapauksessa aiheuttaa sen, että palvelinten mennessä tukkoon, peliä ei yksinkertaisesti pääse pelaamaan ja ensivaikutelma on pilattu. Tätä voidaan ehkäistä varaamalla peliä varten riittävästi palvelimia, mutta palvelimet ovat kalliita ylläpitää. Halvempi ja helpompi tapa on ensin minimoida pelin vielä tila ja sitä kautta kuorma palvelimelle.

Pelin käyttämistä resursseista kuva- ja äänitiedostot haukkaavat tyypillisesti valtaosan pelin koosta. Loput tiedostot ovat scriptejä, konfiguraatiotiedostoja, 3D-malleja, fysiikkamalleja ja muuta tarpeellista dataa. Kaiken pelitiedon pakkaaminen minimaaliseen tilaan on etenkin verkon yli pelattavissa Flash-peleissä aina hyvä idea. Pakettien purkamiseen kuluva aika loppukäyttäjän koneella verrattuna suureen säästöön pelin latausajoissa on voitto sekä palvelimelle että pelaajalle. Mitä vähemmän kuormaa, sitä suurempi maksimipelaajamäärä yhtä palvelinta kohti ja sitä halvemmat ylläpitokustannukset. Niinpä rahaa jää enemmän uusiin palvelinhankintoihin ja kaistanleveyteen. Lopputuloksena on mukavampi pelikokemus.

8.2.1 Pakkaaminen

Kuvat kannattaa pääsääntöisesti tallentaa PNG-muodossa. PNG on häviötön toisin kuin JPEG ja monipuolisempi kuin GIF. PNG voidaan käytännössä tallentaa joko 24-bittisenä häviöttömänä kuvatiedostona vaihtoehtoisella 8-bittisellä alfakanavalla tai rajoitetulla väripaletilla indeksoituna. Indeksoimisella voidaan säästää paljon tilaa, mutta mahdollisimman pienen väripaletin valitseminen ilman värihävikkiä voi olla työlästä.

Tätäkin varten on onneksi olemassa työkaluja. Itse tapaan kehottaa artisteja tallentamaan kaikki kuvat 32-bittisinä PNG-tiedostoina, jotka ajan itse jälkeenpäin automatisoidusti Ken Silvermanin kehittämän ilmaisen PNGOUT-komentorivityökalun lävitse. PNGOUT analysoi PNG-tiedoston ja pakkaa sen mahdollisuuksien mukaan tarvittaessa uudelleen pienempään tilaan säilyttäen kaikki alkuperäisvärit (Ken Silverman's Utility… 2012).

Äänitiedostoja varten Flashin natiivisti tukema MP3 on helppo ja laadukas vaihtoehto. Vaihtoehtoisesti myös avoimen lähdekoodin Ogg Vorbista varten on olemassa jotain yhteisön kehittämiä dekoodereita, jos haluaa patentti- tai muista syistä välttää MP3-muotoa. MP3-muodon ollessa kyseessä vaihtelevalla bittivirralla (VBR) pakattu äänitiedosto takaa yleensä parhaan koko/laatu-suhteen keskiverto bittivirran pysyessä 128-192 välimaastossa. Pakkaamiseen voidaan käyttää esimerkiksi ilmaista LAME-komentorivityökalua.

Loppuja tiedostoja varten pakkausalgoritmeista Lempel-Ziv-Markovin LZMA on kirjoitushetkellä yksi parhaista vaihtoehdoista avoimen lähdekoodinsa ja tehokkuutensa ansiosta. Myös vanha suosikki DEFLATE, jota käytetään esimerkiksi PNG- ja ZIP-tiedostoissa, on edelleen käytännöllinen, jos nopeus on tärkeää. LZMA pakkaa tiiviimmin kuin DEFLATE, mutta vie enemmän aikaa sekä pakatessa että purkaessa. Yksi varteenotettava tekijä on Adoben linjaus pakata itse SWF-tiedostot käyttäen DEFLATE:a, joka lie johtanut myös siihen, että Flashissa on valmiina kirjasto DEFLATE-tiedostojen käsittelyyn. Niinpä DEFLATE on helpompi vaihtoehto. Kannattaa silti muistaa, että LZMA:n tuoma lisähyöty saattaa olla kullan arvoista. Molemmissa on puolensa.

8.2.2 Arkistointi

Yhden ison tiedoston lataaminen on nopeampaa kuin useamman pienen. Kun tietoa ladataan levyltä, on tiedoston hakuaika levyltä yleensä käyttöjärjestelmästä ja levystä riippuen suurin aikasyöppö, koska kiintolevyn lukupää joutuu fyysisesti siirtymään oikeaan kohtaan jokaisen tiedoston kohdalla. Tiedostot kannattaa paketoida siten isommiksi kokonaisuuksiksi arkistoihin, jolloin hakuaika saadaan minimoitua (Gregory 2009, 285.). Jakaminen voidaan suorittaa esimerkiksi kenttäkohtaisesti.

Riippuen käytettävästä pakkausalgoritmista, tällainen arkistomuoto saattaa olla valmiiksi saatavilla. DEFLATE johtaa luonnollisesti ZIP-muotoon, jota myös Flash tukee natiivisti. LZMA-kompressointia varten on olemassa 7z, mutta kirjoitushetkessä kukaan ei ole vielä portannut sitä Flashiin. Yksi vaihtoehto on tietenkin tehdä tämä porttaustyö itse, mutta se saattaa olla melko työlästä verrattuna oman arkistoformaatin keksimiseen.

8.3. Mikro-optimointi

Mikrotason optimointi käsittää viilaukset, jotka kohdistuvat tyypillisesti vain yksittäisiin koodiriveihin. Tämä ei toki tarkoita, ettei mikro-optimointi olisi ajan arvoista. Jos jokin raskas matemaattinen operaatio, kuten käänteisen neliöjuuren laskeminen suoritetaan kymmeniä tuhansia kertoja sekunnissa esimerkiksi kolmiulotteisia yksikkövektoreita laskettaessa pelin valaisua varten, on muutaman prosenttiyksikönkin trimmaaminen tällaisen laskutoimituksen suoritusajasta suuri voitto.

Tämän tason optimointikäsittely tulisi tosin monilta osin suorittaa vasta projektin loppuvaiheilla, kun peli on jo toiminnallisesti valmis, mutta ohjelmoijalla on aikaa, halua ja yleensä tarvettakin puristaa ne viimeiset mehut irti raudasta pelin tavoiteruudunpäivitysnopeuden saavuttamiseksi. Haittapuolena mikro-optimoinnilla on nimittäin sen paha tapa vaikeuttaa koodin luettavuutta ja ymmärrettävyyttä (Garney & Preisz 2010, 14.). Jos pelin arkkitehtuuri on vielä työn alla, vaikeaselkoinen ohjelmakoodi hidastaa pelin kehitystä merkittävästi. Puhumattakaan siitä ettei mikro-optimointi ole aina edes vaivan arvoista. Mikro-optimoinnin - kuten minkä tahansa muunkin optimoinnin - hyödyllisyys riippuu pitkälti siitä, miten aikakriittisestä koodinpätkästä on kyse.

8.3.1 Alchemy

Flash Player version 10 myötä Adobe laajensi virtuaalikonettaan Alchemyksi nimetyllä lisäpalikalla. Tämän tarkoitus oli mahdollistaa C- ja C++ -kielillä kirjoitetun ohjelmakoodin kääntäminen ActionScriptin virtuaalikoneen ymmärtämäksi tavukoodiksi. Käytännössä Alchemy sisälsi käskyjä, jotka liittyivät muistinhallintaan, joka on C-sukuisille kielille tärkeää (Elsass 2010). Osoittimet ja dynaaminen muistinvaraus ovat outoja käsitteitä ylemmän tason ohjelmointikielissä kuten ActionScriptissä, mutta C-ohjelmissa

muisti on vakava asia. Alchemyn kautta Adobe avasi potentiaalisesti erittäin suuren kanavan kaikenlaisten C-kirjastojen uudelleenkäyttämiseen Flashissa.

Koska Alchemy on suora lisäys Flashin virtuaalikoneeseen, pääsevät myös ActionScript-ohjelmoijiat nyt teoriassa paremmin käsiksi muistiin. Eri asia on, miten järkevää ja käytännöllistä dynaaminen muistinvaraus on Flash-sovelluksissa.

Kannattaa muistaa, että ActionScript ei voi Alchemysta huolimatta mitenkään yltää C-kielellä kirjoitetun natiivikoodin suorituskykyyn. C-koodi on edelleen vähintään 2-10 kertaa nopeampaa. Tämä ei liene mikään yllätys, sillä ActionScript on ja pysyy tulkattuna kielenä. Vaikka Alchemy nopeuttaa teoriassa muistin käpälöintiä noin kymmenkertaisesti, on itse muistioperaatioilla tyypillisissä Flash sovelluksissa loppujen lopuksi hyvin pieni vaikutus koko ohjelman suorituskykyyn.

Tämän lisäksi Alchemyn käskyt ovat hyvin visusti piilossa asiaan perehtymättömiltä Flash-ohjelmoijilta. Tästä voidaan päätellä, että Adobe ei todennäköisesti tarkoittanut Alchemya käytettävän ActionScriptin avulla. Uusiin käskyihin pääsee käsiksi vain muokkaamalla suoraan tavukoodia. Tätä varten on kehitetty monenlaisia työkaluja, kuten Apparat ja Azoth, jotka prosessoivat julkaistun SWF-tiedoston, lisäten halutut Alchemy käskyt (Elsass 2010). Koko homma tuntuu enemmän hakkeroinnilta kuin ohjelmointityöltä, eikä siitä syystä varmaan istu kovin monen Flash-ohjelmoijan arkipäivään.

8.3.2 Bittisiirrot

Yksittäisiä laskutoimituksia voidaan tehostaa bittisiirroilla, jotka ovat yleisesti ottaen ne mahdollistavissa ohjelmointikielissä erittäin nopeita. Sama pätee AS3:ssa. Mikäli termi on outo, ei syytä huoleen. Aihepiiriä valottaa, jos tutustuu siihen, miten tietokone käsittelee numeroita binääritasolla. Ennen tällaisten temppujen käyttämistä tulisi kuitenkin aina tarkistaa, nopeuttavatko ne oikeasti koodia. Olen koonnut alle yleisimpiä bittisiirtojen mahdollistamia kikkoja.

```
// Negatointi
i = -i;
i = (i ^ -1) + 1; // 300% nopeampi

// Kertominen kahden potenssilla (2, 4, 8, 16, 32, 64, 128, ...)
```

```
i = i * 128;
i = i << 7; // 300% nopeampi

// Jakaminen kahden potenssilla
i = i / 32;
i = i >> 5; // 350% nopeampi

// Jakojäännös kahden potenssilla jaettaessa
i = 131 % 4;
i = 131 & (4 - 1); // 600% nopeampi

// Kokonaisluvun parillisuus
if ((i % 2) == 0) { ... }
if ((i & 1) == 0) { ... } // 600% nopeampi

// Itseisarvo
i = Math.abs(i);
i = i < 0 ? -i : i; // 2500% nopeampi
i = (i ^ (i >> 31)) - (i >> 31); // 3125% nopeampi (!!!)
```

(Baczynski 2007)

Kannattaa muistaa, että Flash Player ja sen matematiikkakirjastot kehittyvät jatkuvasti, eivätkä kaikki näistä tempuista välttämättä enää lukuhetkellä ole hyödyllisiä. Optimointia kannattaa ylipäätään aina mieluummin lähestyä järjestelmä- ja algoritmitasolta ennen kuin edes harkitsee mikro-optimointeja.

9 LOPPUSANAT

Flashin kuolemasta on puhuttu jo vuosia selaimiin upotettujen standardien kuten HTML5:n ja WebGL:n yleistyessä. Flash vaatii selaimilta toimiakseen erillisen laajennuksen eikä se esitä sisältöään minkään standardin mukaan, joten väitteissä on osittain perääkin etenkin koskien verkkosivujen tuotantoa. Hakukoneiden on vaikea haalia sisältöä Flash-pohjaisista sivustoista eikä alasivuihin voida suoraan linkittää ulkoisilta sivustoilta. Flashia tulisi siten käyttää vain tukevana elementtinä verkkosivustolla animaatioiden ja liikkuvan kuvan esittämiseen.

Peliteollisuudessa Flashin asema on tosin mielestäni tänään vahvempi kuin koskaan aiemmin. Tie tähän pisteeseen on ollut pitkä. Flashilla on vahva tausta kaksiulotteisessa vektorigrafiikka-animaatiossa ja käyttöliittymäsuunnittelussa. Niihin käyttötarkoituksiin Flashia tullaan käyttämään vielä pitkään. Myös pelialustana Flash on viimeisen kahden vuoden aikana noussut vahvasti kilpailemaan nousevien tähtien, kuten Unityn ja WebGL:n kanssa. Uudistukset Flashiin etenkin piirtopuolella ovat saaneet myös Unityn tukemaan Flashia omasta selainlaajennuksestaan huolimatta, joka varmasti puhuu puolestaan.

Flashin menestys perustuu kuitenkin pitkälti perustavalaatuisiin arvoihin kuten aloittelijaystävällisyyteen. Porras pelinkehityksen aloittamiseen on Flashilla teknisestä näkökulmasta minimaalinen, joten alusta palvelee hyvin pelinkehityksestä kiinnostuneita kannustamaan harjoitteluun. Moni kritisoi Flash-pelejä niiden keskivertolaadun heikosta tasosta, mutta se on vain luonnollinen seuraus alustan helposta lähestyttävyydestä.

Myös Flashin hitaus esimerkiksi C++-ohjelmointiin verrattuna on merkillepantava miinus, joka aiheuttaa myös itselleni viikoittain harmaita hiuksia, mutta kun ongelmatilanteet otetaan vastaan enemmän haasteina kuin esteinä, niin niistä voi oppia jopa nauttimaan. Siitähän peliohjelmoinnissa pohjimmiltaan on kyse: nerokkaiden ratkaisujen löytämisestä ja optimoinneista alustan ehdoilla. Flash on mielestäni edelleen paras tapa tuoda pelejä verkkoon pelattavaksi. Tulevaisuudessa tilanne voi hyvin muuttua, mutta vaikka Flash pitäisi pintansa vielä 10 vuotta, en osaa vaatia parempaa. Flash on erinomainen ympäristö pelinkehitykseen.

10 LÄHTEET

ActionScript language and syntax. Luettu 18.5.2012.
http://help.adobe.com/en_US/as3/learn/WS5b3ccc516d4fbf351e63e3d118a9b90204-7f9c.html.

Autodesk Scaleform: Overview. Luettu 25.4.2012.
http://gameware.autodesk.com/scaleform/features/overview.

Baczynski, M. 2012. Introduction to ds. Luettu 29.4.2012.
http://lab.polygonal.de/?p=2513.

Baczynski, M. 2007. Bitwise gems – fast integer math. Luettu 29.4.2012.
http://lab.polygonal.de/?p=81.

Bibby, J. 2007. MochiAds: now open to everyone! Luettu 17.5.2012.
http://jayisgames.com/archives/2007/10/mochiads_now_open_to_everyone.php.

Brimelow, L. 2008. Six reasons to use ActionScript 3.0. Luettu 26.4.2012.
http://www.adobe.com/devnet/actionscript/articles/six_reasons_as3.html.

Brown, D. 2006. AS3 programming 101 for C/C++ coders. Luettu 26.4.2012.
http://blogs.adobe.com/kiwi/2006/05/as3_programming_101_for_cc_cod_1.html.

Elsass, P. 2010. AS3 – fast memory access without Alchemy. Luettu 30.4.2012.
http://philippe.elsass.me/2010/05/as3-fast-memory-access-without-alchemy/.

Elman, J. 2009. Bringing Adobe Flash Platform and Facebook Platform Together.
Luettu 29.4.2012. http://developers.facebook.com/blog/post/217/.

FlashDevelop Features: Completion. Luettu 18.5.2012.
http://www.flashdevelop.org/wikidocs/index.php?title=Features:Completion.

Gregory, J. 2009. Game Engine Architecture. United States: A K Peters.

Garney, B. & Preisz, E. 2010. Video Game Optimization.
United States: Course Technology PTR.

Gay, J., Grossman, G. & Tatsumi, R. 2008. Grandmasters of Flash: An Interview With The Creators of Flash. Luettu 22.4.2012.
http://coldhardflash.com/2008/02/grandmasters-of-flash-an-interview-with-the-creators-of-flash.html.

Greene, J. 2007. The Road to Actionscript 3. Luettu 26.4.2012.
http://www.digital-web.com/articles/the_road_to_actionscript_3/.

Ken Silverman’s Utility Page: Compression Utilities. Luettu 18.5.2012.
http://advsys.net/ken/utils.htm.

McMillen, E. & Refenes, T. 2011. Postmortem: Team Meat's Super Meat Boy.
Luettu 22.4.2012. http://www.gamasutra.com/view/feature/134717.

MobyGames: Scaleform Corporation. Luettu 25.4.2012. http://www.mobygames.com/company/scaleform-corporation.

Moock, C. 2007. Essential ActionScript 3.0. United States: O'Reilly Media.

Nutt, C. 2009. The Facebook Doctorine: Gaming And The Future. Luettu 17.5.2012. http://www.gamasutra.com/view/feature/132465/the_facebook_doctrine_gaming_and_.php?page=2.

Optimizing Performance for the Adobe Flash Platform. Luettu 18.5.2012. http://help.adobe.com/en_US/as3/mobile/flashplatform_optimizing_content.pdf.

Orland, K. 2011. Ubisoft's Ancel Planning To Open Up Rayman Origins Tech. Luettu 25.4.2012. http://www.gamasutra.com/view/news/125952.

Patterson, T., Anderson, D., Cardwell, N., Fromm, R., Keeton, K., Kozyrakis, C., Thomas, R. & Yelick, K. 1997. A Case for Intelligent RAM: IRAM. Luettu 18.5.2012. http://iram.cs.berkeley.edu/papers/IRAM.micro.pdf.

Resource Handling in Flash. 2010. Luettu 18.5.2012. http://pixelpracht.wordpress.com/2010/06/14/resource-handling-in-flash-part-two/.

Scabia, M. 2011. How Stage3D works. Luettu 28.4.2012. http://www.adobe.com/devnet/flashplayer/articles/how-stage3d-works.html.

Set Preferences in Flash. Luettu 18.5.2012. http://help.adobe.com/en_US/flash/cs/using/WSd60f23110762d6b883b18f10cb1fe1af6-7f99a.html.

Thibault, I. 2011. Introducing Starling. Luettu 28.4.2012. http://www.bytearray.org/?p=3371.

Yaiser, M. 2011. Garbage collection internals for Flash Player. Luettu 29.4.2012. http://www.adobe.com/devnet/actionscript/learning/as3-fundamentals/garbage-collection.html.