MOBIILIPELINKEHITYS COCOS2D-X -SOVELLUSKEHYKSELLÄ

Markku Lehmonen

Opinnäytetyö
Toukokuu 2013

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖN
KUVAILULEHTI

<table>
<tr><td rowspan="3">Tekijä(t)
LEHMONEN, Markku</td><td>Julkaisun laji
Opinnäytetyö</td><td>Päivämäärä
13.5.2013</td></tr>
<tr><td>Sivumäärä
71</td><td>Julkaisun kieli
Suomi</td></tr>
<tr><td></td><td>Verkkojulkaisulupa myönnetty
(X)</td></tr>
<tr><td colspan="3">Työn nimi
MOBIILIPELINKEHITYS COCOS2D-X -SOVELLUSKEHYKSELLÄ</td></tr>
<tr><td colspan="3">Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma</td></tr>
<tr><td colspan="3">Työn ohjaaja(t)
PIETIKÄINEN, Kalevi</td></tr>
<tr><td colspan="3">Toimeksiantaja(t)</td></tr>
<tr><td colspan="3">Tiivistelmä

Opinnäytetyönä tutkittiin mobiilipelien kehittämistä Cocos2d-x -sovelluskehyksessä. Tarkoituksena oli pohtia yleisiä mobiilipelinkehityksen periaatteita ja selvittää kuinka kyseistä sovelluskehystä käytetään mobiilipelien kehitystyössä. Työssä perehdyttiin sovelluskehykseen sekä teorian että käytännön kautta pelinkehityksen yleisiä periaatteita ja tavotteita unohtamatta.

Aluksi kartoitettiin hieman pelinkehityksen yleisiä periaatteita silmällä pitäen alustariippumattoman sovelluskehityksen piirteitä, lähtökohtia ja haasteita. Alkuselvityksien jälkeen siirryttiin tutkimaan valittua sovelluskehystä aluksi yleisestä ja ekosysteemilähtöisestä perspektiivistä. Tämän jälkeen perehdyttiin hieman Cocos2d-x:n arkkitehtuuriin ja käytettyihin ratkaisuihin ja komponentteihin. Lopulta tutkittiin sovelluskehyksen luokkarakenteita pelinkehityksen yleisen teorian kautta ja pohdittiin teorian ja sovelluskehyksen luokkien välisiä yhteyksiä.

Käytännön osuus aloitettiin asentamalla kehitysympäristöt iOS- ja Android-sovelluskehitystyötä varten. Asennustyön jälkeen luotiin molemmille alustoille projektit ja yhdistettiin ne alustariippumattomaksi projektiksi. Tämän jälkeen perehdyttiin käytännön kautta yleisimpiin Cocos2d-x -sovelluskehyksen olioihin, kuten spriteihin, tekstitarroihin ja kontrolleihin.

Lopuksi analysoitiin jo aikaisemmin sovelluskehyksellä toteutetun pelin kokonaisuutta, yksityiskohtia ja ratkaisuja.</td></tr>
<tr><td colspan="3">Avainsanat (asiasanat)
Peliohjelmointi, C++</td></tr>
<tr><td colspan="3">Muut tiedot</td></tr>
</table>

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Authors(s)	Type of publication	Date
LEHMONEN, Markku	Bachelor's Thesis	13.5.2013
	Pages 71	Language Finnish
		Permission for web publication (X)

Title
MOBILE GAME DEVELOPMENT WITH COCOS2D-X FRAMEWORK

Degree Programme
Software Engineering

Tutor(s)
PIETIKÄINEN, Kalevi

Assigned by

Abstract

This thesis examines the mobile game development using Cocos2d-x framework. The goal was to consider some universal principles of mobile game development and find out how to use the framework in development pipeline. Both theoretical and practical aspects of the Cocos2d-x framework were studied with the general design patterns and goals of game development in mind.

The study was started by finding out the general principles of game design and researching the aspects, basis and difficulties of the cross-platform software development. After starting point the chosen framework was put on the table for the examination of the roots and the general ecosystem it is a part of. Next the focus was targeted on the architecture of Cocos2d-x game engine while the chosen technologies and solutions were analyzed. The final theoretical part of this thesis was composed with an investigation of the framework's class hierarchy using the general theory game designs are based on, not forgetting the individual classes typically used in mobile game solutions.

The practical part of this thesis was started by installing the development environment for the iOS and Android software development. After installing the tools, both individual projects for the target platforms were created and combined as a single project usable in cross-platform development. After this the actual usage of the classes and elements of Cocos2d-x framework were examined in practice. This involves the most general game objects like sprites, text labels and controlling systems.

In the final part this thesis takes a look at an already developed mobile game using Cocos2d-x framework when the whole game and its details and solutions had been reviewed.

Keywords
Game Programming, C++

Miscellaneous

SISÄLTÖ

KUVIOT

TAULUKOT

1 TYÖN LÄHTÖKOHDAT

Mobiililaitteiden viime vuosien nopea kehitys on luonut peliteollisuuteen uuden varsin varteenotettavan osa-alueen. Suorituskyvyn kasvu ja uudenlaiset ohjaustekniikat ovat mahdollistaneet entistä näyttävämpien ja viihdyttävämpien pelien valmistamisen. Laitteiden runsas yleistyminen puolestaan on avannut aivan uudet ja monipuoliset markkinat, jonka mahdollisuudet todistavat uudet menestystarinat, kuten Rovio ja Supercell.

Markkinat ovat kuitenkin jakautuneet useamman mobiilialustan kesken. Sovelluksen kehittäminen jokaiselle alustalle erikseen on kuitenkin kustannustehotonta, ja koska pelisovellukset sisältävät paljon toteutusta, joka on monissa tapauksissa käytännöllisesti katsoen universaalia, hakevat ohjelmistokehittäjät ratkaisuja tehtävän työn tiivistämiseksi. Lukuisiin eri tarpeisiin ja tilanteisiin räätälöidyt ratkaisut, kuten sovelluskehykset, tarjoavat valmiita sovelluksia, nopeuttavat kehittämistyötä ja parantavat näin ollen tuottavuutta.

Tässä opinnäytetyössä perehdyttiin yhteen mobiilipelien suosituimpaan sovelluskehykseen, Cocos2d-X:ään. Kyseinen sovelluskehys on hyvin sovellustyyppispesifinen. Toisaalta pelisovellukset ovat äärimmäisen monipuolisia sovelluksia. Näin ollen sen täysivaltaiseksi ymmärtämiseksi on hyvä tuntea ja tiedostaa sekä aihealueen että eri komponenttien lähtökohtia ja yleistä teoriaa. Työn tekeminen aloitettiin perehtymällä alustavasti yleisiin mobiilialustakohtaisiin piirteisiin, kuinka ne vaikuttavat mobiilipelinkehitykseen. Perusteita ei unohdettu missään vaiheessa, vaan ne pidettiin mielessä ja niitä käsiteltiin läpi koko työn sekä teorian että käytännön kautta.

Itse sovelluskehykseen perehtyminen aloitettiin tutustumalla sen historiaan ja ekosysteemiin. Avoimen lähdekoodin projektina Cocos2d-x:n arvo sen käyttäjälle ei muodostu pelkästään tarjottujen ominaisuuksien, vaan myös ympärille rakentuneen yhteisön ja lukuisten muiden projektien ja työkalujen kautta.

Kun sovelluskehyksen lähiympäristö oli kartoitettu, aloitettiin Cocos2d-x -sovelluskehyksen lähempi tarkastelu sen arkkitehtuurista ja yleisistä rakenteista: perehdyttiin teknologioihin, jotka ovat sovelluskehyksen olennaisimpia osia. Koska käytännössä Cocos2d-x:n ydin on API, on hyvä perehtyä myös keskeisimpiin luokkiin ja oliohierarkioihin. Niitä tarkasteltiin sekä itse sovelluskehyksen että yleisten pelisuunnittelun ja pelinkehityksen periaatteiden kautta.

Käytännöllinen osuus aloitettiin asentamalla tarvittavat kehitystyökalut sekä Windows- että OS X -ympäristössä. Koska pelinkehittäjän kannalta mielenkiintoisimmat ja ainoat varteenotettavat alustat ovat Google Android ja Apple iOS, luotiin sovellusprojekti vain niiden pelinkehitykseen. Projektin kautta tutustuttiin, kuinka pelisovelluksia mobiilialustalle käytännöllisellä tasolla rakennetaan.

Kun perusteet olivat jo hallussa, perehdyttiin mobiilipelinkehitykseen ja Cocos2d-X -sovelluskehykseen syventävästi jo ennalta toteutetun Waterly-mobiilipelin kautta.

2 PELINKEHITYS JA MOBIILIALUSTAT

2.1 Mobiililaitteet pelilaitteina

Verrattuna perinteisiin pelilaitteisiin, kuten kotitietokoneisiin ja pelikonsoleihin, omaavat älypuhelimet ja taulutietokoneet monia, jopa ainutlaatuisia piirteitä ja ominaisuuksia. Ilmeisimpiä näistä ovat langattomuus ja laitteiden pienehkö koko, jotka mahdollistavat laitteen liikkuvuuden. Merkittävin tekninen ero ovat käyttäjärajapinnan ohjauslaitteet. Vuoden 2007 jälkeen älypuhelimissa on lähes yksinomaan luovuttu näppäimistöstä. Monikosketusnäytöt mahdollistavat laitteen ohjauksen joko yhdellä tai useammalla sormella joko koskettaen tai hipaisten. Niin sanotut sormieleet eli

gesturet tuovat monipuolisia mahdollisuuksia käyttäjärajapintaan ja tekevät laitteen käyttämisestä luonnollista ja intuitiivista.

Suhteellisen hyvästä tarkkuudestaan huolimatta kosketusnäytöt eivät kuitenkaan sovellu aivan jokaiseen pelinohjaustilanteeseen, vaan tarvitaan nopeampia ja tarkempia ohjausmenetelmiä. Jokainen moderni älypuhelin on varustettu kiihtyvyysanturilla ja gyroskoopilla, jotka tunnistavat laitteen sijainnin, asennon ja liikkeen suunnan ja kiihtyvyyden. Puhelinta itsessään voi siis käyttää eräänlaisena peliohjaimena.

Älypuhelimiin verrattuna taulutietokoneet tarjoavat samat ominaisuudet ja jopa saman käyttöjärjestelmän kuin älypuhelimet. Näin ollen sovellusta kehitettäessä voidaan samainen sovellus kehittää ja ajaa molemmilla laitteilla. Pelejä ja muita sovelluksia kehitellessä on kuitenkin syytä ottaa huomioon yksi tärkeä avaintekijä: näytön fyysinen koko. Vaikka näyttöjen pikseliavaruudet voivat olla puhelimien ja taulutietokoneiden välillä jopa samat, ovat taulutietokoneiden näyttöpaneelit fyysisiltä mitoiltaan 4-6 kertaa suurempia. Pelin käyttöliittymää suunnitellessa tulee siis tarkoin miettiä sen toimivuutta eri kokoluokan laitteilla.

Mobiilialustojen erikoispiirteet eivät kuitenkaan rajoitu pelkkiin laitteiden fyysisiin ominaisuuksiin. Oli kyseessä sitten Microsoftin, Googlen tai Applen tuote, kyseessä on aina kokonainen ekosysteemi, johon itse laitteen ja käyttöjärjestelmän lisäksi kuuluvat muun muassa pilvipalvelut datan säilyttämiseen ja sovellusten ostamiseen. Verrattuna perinteisiin pelilaitteisiin tarjolla ei ole lainkaan mahdollisuutta ostaa sovelluksia fyysisinä medioina. Vastikkeena laitteen käyttäjä voi vaivattomasti ostaa sovelluksia käyttöjärjestelmän valmistajan omista sovelluskaupoista.

Distribuutioväylien, kuten **Google Play** ja **Apple App Store**, vaivattomuus on erityisesti pelinkehittäjien intressi niiden alhaisten käyttökustannuksien ja mahdollisen suuren näkyvyyden ja markkinasegmentin vuoksi, mikä onkin johtanut mobiilipeliyritysten määrän räjähdysmäiseen kasvuun viime vuosina. Eduistaan huolimatta mobiilialustat sisältävät sovelluskehittäjän näkökulmasta merkittäviäkin haasteita.

2.2 Mobiilialustojen diversiteetti ja fragmentaatio

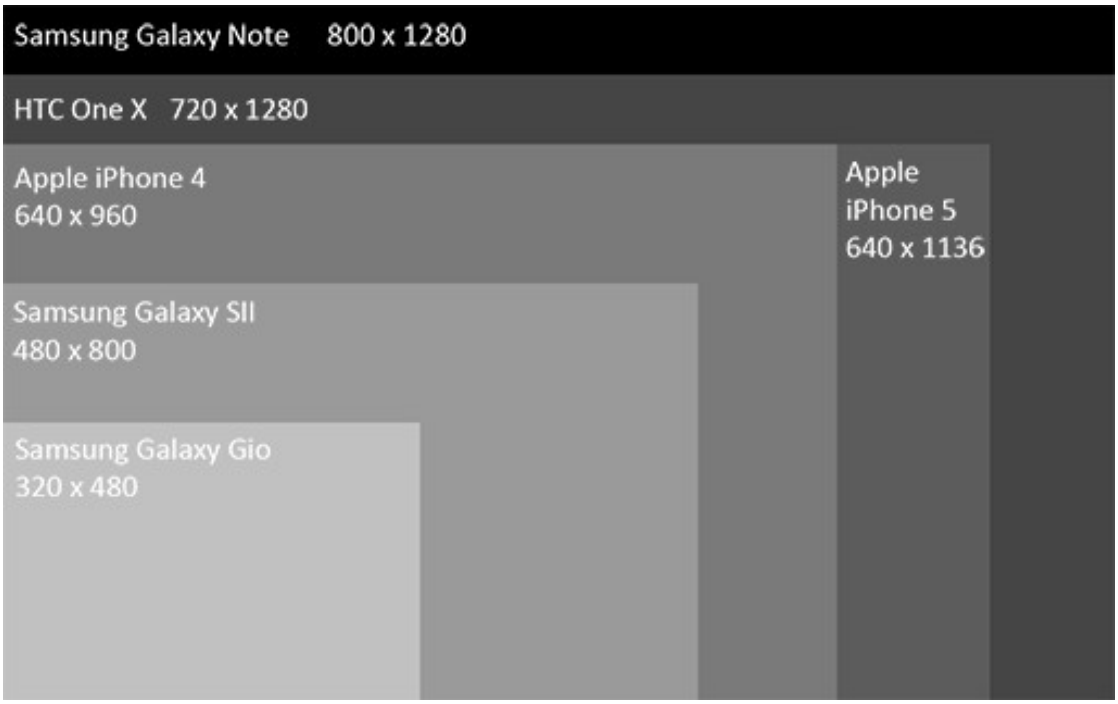
Mobiilimarkkinoiden on arvioitu olevan 80-90 -prosenttisesti kahden alustan, Googlen Androidin ja Applen iOS:n, hallitsema (Usage share of operating systems 2013). Nopean tarkastelun perusteella voisi siis luulla, ettei erilaisten alustojen määrä olisi kovinkaan suuri ongelma mobiilisovelluksia kehitettäessä.

Applen oma laitekanta perustuu kokonaisuudessaan itse suunniteltuun ekosysteemiin. Sekä laitteisto että ydinohjelmistot, kuten käyttöjärjestelmä, suunnitellaan ja jopa toteutetaan saman yrityksen sisällä. Tämän ja suhteellisesti vähäisen laitetarjonnan myötä koko ekosysteemi on eheä, ja mikäli rautatason eroja ensimmäisten ja uudempien sukupolvien laitteiden välillä ei oteta huomioon, toimii sama kehitetty pelisovellus yleisesti ottaen laitteesta riippumatta kunhan laitteeseen asennettu käyttöjärjestelmäversio on vaadittu vähimmäisversio.

Googlen Androidin kohdalla tilanne on kuitenkin toisin. Google ei itse valmista tai edes suunnittele lähellekään kaikkia Android-laitteita, vaan vapaan lähdekoodin ohjelmistona Android on levinnyt useiden laitevalmistajien älypuhelimien ja taulutietokoneiden käyttöjärjestelmäksi. Laitevalmistajat ovat vapaita muokkaamaan käyttöjärjestelmää haluamakseen, ja kun lisätään tähän vielä laitteiden runsas rautatason variaatio, on Android alustana kaikkea muuta kuin eheä.

Alustan fragmentaatio luo ongelmia ohjelmistojen kehittämisen ja erityisesti testaamisen kannalta. Halutulle sovelluksen toiminnolle voidaan joutua tekemään useita mahdollisia toteutuksia, ja mikäli sovelluksen yhteensopivuuden ja toimivuuden suhteen halutaan olla varmoja, joudutaan panostamaan lukemattomia tunteja sovelluksen testaamiseen lukuisilla eri laitemallien ja käyttöjärjestelmäversioiden yhdistelmillä. On myös kokemuksia, että jopa puhelimen myyntimaalla tai liittymän operaattorilla on vaikutusta.

Useat eri laitteet ja laitemallit tietävät useita eri mallisia näyttöpaneeleita, jopa Applen ekosysteemin sisällä, joka on muilta osin kuitenkin varsin yhtenäinen. Ilmeisimpiä ovat näyttöpaneelien erot fyysisissä mitoissa ja resoluutioissa (ks. kuvio 1). Tulee kuitenkin huomata, että mobiililaitteiden välillä vaikeimmat eroavaisuudet löytyvät näyttöjen kuvasuhteissa. Eri resoluutioihin varautuminen on suhteellisen helppoa: pakataan peliin kuvamateriaalit sekä pienille että suurille resoluutioille. Vaihtelut kuvasuhteissa ovat kuitenkin astetta ongelmallisempia, sillä se vaikuttaa olennaisesti näytön mittasuhteisiin ja sitä kautta näytettävän pelimaailman ulottuvuuksiin. Tämä luo ongelmia pelisuunnittelun kannalta.



KUVIO 1. Eri näyttöresoluutioiden vertailua suhteessa toisiinsa

Oli ongelma sitten vaikea tai helppo ratkaista, siihen tulee kuitenkin varautua. Helpoimmat ja yksinkertaisimmat keinot ovat yleensä universaaleja, eivätkä ne ole sidoksissa itse pelin pelisuunnittelullisiin ratkaisuihin. Tämä tarkoittaa, että ne on helppo paketoida yleisesti käytettäviksi tuotteiksi, eikä niitä kannata toteuttaa erikseen jokaisen valmistettavan tuotteen kohdalla. Tämä lisää kehitystyön ja tuotannon tehokkuutta.

2.3 Mobiilipelien yhteisiä piirteitä ja rakenteita

Riippumatta genrestä tai arkkitehtuurista on jokaisella reaaliaikasuoritteisella pelillä yhteisiä piirteitä ja rakenteita. Oli kyseessä sitten Tetriksen kaltainen puzzle tai tasohyppely, tarvitaan pääsuoritussilmukka (ks. kuvio 2), joka kommunikoi käyttöjärjestelmän kanssa, vastaanottaa pelaajan antaman syötteen, käskyttää taustalogiikan suorittavia komponentteja ja määrittelee tilanteen vaatiman äänen ja piirrettävän kuvan.

Interaktiivisena sovelluksena mobiilipelit sisältävät vähintään kaksi komponenttia: grafiikan ja kontrollit. Näiden toimintojen toteuttamiseksi tarvitaan rajapinnat alustan ja pelisovelluksen välille. Pelinkehittäjien onneksi suosituimpien mobiilialustojen rajapinta grafiikanpiirtoon on toteutettu OpenGL ES:llä. Ne käyttävät siis samaa graafisen rajapinnan kirjastoa, mikä helpottaa enimmiltä osin grafiikkaohjelmointia eri alustoille. Täytyy kuitenkin huomata, että esimerkiksi näkymän luominen ja ikkunointi tapahtuu alustakohtaisilla rajapinnoilla.

KUVIO 2. Esimerkki yksinkertaisen pelisovelluksen pääsilmukasta

Harvassa ovat pelit jotka eivät sisältäisi valikoita. Valikot ovat yleisin navigointimenetelmä pelitapahtumien välitiloissa, kuten kentän tai käytettävän pelihahmon valitsemisessa. Niiden toteutukset ovat useimmiten universaaleja ja hyviksi koettuja, eikä niiden toteuttaminen jokaisessa pelijulkaisussa ole erityisen järkevää, ellei haeta omanlaista efektiä, kuten valikon sulautuminen itse peliympäristöön.

Mobiilipelit ovat sovelluksia ja sovelluksiin kuuluvat olennaisena osana erilaiset tilamuutokset. Sovelluksen on kyettävä taustalle siirtyessään keskeyttämään toimintansa

ja palautuessaan aloittamaan sen uudelleen, mahdollisesti jopa täsmälleen samasta suoritusvaiheesta. Erityisesti peleissä on oltava mahdollisuus ns. taukotilalle, johon pelaaja voi tarvitessaan keskeyttää pelaamisen väliaikaisesti.

2.4 Ratkaisuna alustariippumaton sovelluskehys

Pelisovelluksilla on siis genrestä riippumatta sekä yhteisiä haasteita että samoja ominaisuuksia ja toimintoja. Lisäksi on luonnollista, että halutaan kohdistaa sovellus mahdollisimman usealle alustalle, jotta saavutettaisiin mahdollisimman suuri asiakaskunta. Saman toiminnallisuuden toteuttaminen useaan otteeseen jokaisessa uudessa projektissa erikseen on resurssien kannalta tuhovoimaista, ja mikäli sovellus on tarkoitus kehittää ja toteuttaa useammalle alustalle, kasvavat käytetyt työtunnit moninkertaisiksi. Täytyy siis löytää keino työmäärän minimoimiseen: **alustariippumaton sovelluskehys**.

Sovelluskehyksellä tarkoitetaan ohjelmistotuotetta, jonka on tarkoitus olla universaali ja uudelleenkäytettävä alusta tiettyyn tehtävään tai tehtäviin erikoistuneemmalle sovellukselle. Ohjelmistokehys sisältää mahdollisesti kehitystyötä tukevia ohjelmia, kääntäjiä, koodikirjastoja, API:n ja työkaluja, jotka mahdollistavat mahdollisimman tehokkaan sovelluskehitystyön. (Software Framework 2013)

Alustariippumattomuudella tarkoitetaan kykyä suorittaa ohjelmistoa useammalla alustalla. Käytännössä tämä voidaan toteuttaa kahdella tavalla: joko kääntämällä ohjelma alustariippumattomaksi suunnitellulla kirjastolla jokaiselle alustalle natiivi binääri, tai rakentamalla jokaiselle tuetulle alustalle yhteensopivuuskerros tai virtuaalikone. (Cross-platform 2013)

Cocos2d-x -sovelluskehyksessä yhdistyvät molemmat edellä mainitut ominaisuudet. Lisäksi useiden eri sovelluskehittäjien ja henkilöiden käyttämänä se on käytännössä jatkuvasti testauksen alla. Äärimmäisen aktiivisena avoimen lähdekoodin projektina

sovelluskehyksen kehitystyö on jouhevaa, uusia versioita ilmestyy lyhyin aikavälein ja ilmenneet bugit korjataan nopeasti.

3 COCOS2D-X 2.0:N ESITTELY

3.1 Cocos2d-tuoteperhe

Alkuperäinen **Cocos2d** on Python-ohjelmointikielellä ikkunointi- ja multimediakirjasto Pygletin päälle toteutettu sovelluskehys kaksiulotteisten pelien, demojen ja graafisten/interaktiivisten sovellusten kehittämiseen. Se tarjoaa kaikki yleisimmät kaksiulotteisten pelien ominaisuudet, kuten sprite-grafiikan, valikot ja perusliikuttelun. Cocos2d on lisensoitu avoimen lähdekoodin lisenssillä BSD:llä. Siitä onkin sittemmin kasvanut muiden kehittäjien myötä kokonainen tuoteperhe, jonka tuotteet jakavat yhteisen nimen ja filosofian. (cocos2d.org 2013)

Ensimmäinen eri alustalle ja kokonaan toiselle ohjelmointikielelle käännetty Cocos2d-perheen lisäys on iOS- ja Mac OS X -alustoille suunnattu **Cocos2d-iPhone**. Se on toteutettu Objective-C -kielellä, mutta noudattaa samoja suunnitteluperiaatteita ja konsepteja. Esikuvastaan poiketen Cocos2d-iPhone sisältää ydintoiminnallisuuden lisäksi integroituna pari kaksiulotteista fysiikkakirjastoa: Box2D:n ja Chipmunkin. (Cocos2d 2013)

Cocos2d-iPhonen jälkeen syntyi monen alustan tuella varusteltu **Cocos2d-X**. Ohjelmointikielenä Cocos2d-X käyttää C++-kieltä. Sen tukemien alustojen listalta löytyvät tärkeimmät alustat, kuten Windows, Mac OS X, Linux, Android ja iOS. Myös Windows Phone löytyy tuettujen alustojen listalta vähäisestä suosiostaan huolimatta. Yhdessä iPhone-version kanssa Cocos2d-X ovat Cocos2d-perheen elovoimaisimmat jäsenet. Rinnalle on kuitenkin syntynyt muitakin enemmän tai vähemmän elossa olevia kään-

nöksiä, kuten Cocos2d-X:stä syntyneet HTML5-käännös **Cocos2d-HTML5** ja Microsoft XNA-yhteensopiva **cocos2d-XNA**. (Cocos2d-x | Relationships in Cocos2d Family, 2013)

Cocos2d-iPhonen kilpailuvaltti Cocos2d-X:ää vastaan ovat sen astetta aktiivisempi Internet-yhteisö ja runsaampi kirjallisuus. Tulee kuitenkin huomata, että Objective-C:llä kirjoitetut koodiesimerkit ovat pääsääntöisesti käytettävissä myös X:llä, sillä yhteisen suunnittelufilosofian myötä ne kääntyvät lähes suoraan C++-kielelle. Cocos2d-iPhonen ja Cocos2d-X:n API:t sisältävät keskinäisiä eroja, mutta ne ovat lähinnä kosmeettisia eroja funktioiden nimeämiskäytänteissä. Näin ollen molempien yhteisöjen olemassaolo hyödyttää ja hyödyntää toinen toisiaan.

3.2 Ympärille syntyneet ja yhteensopivat työkalut

Cocos2d-iPhonen ja Cocos2d-X:n suosion myötä rinnalle on syntynyt monia varta vasten Cocos2d-kehitystä varten toteutettuja työkaluja. Lisäksi on olemassa lukuisia ohjelmia, jotka ovat yhteensopivia ja käytettävissä Cocos2d-sovelluskehyksissä käytettyjen ratkaisujen ja tuettujen tiedostoformaattien kautta.

CocosBuilder

CocosBuilder on graafinen editori, jonka tarkoitus on nopeuttaa Cocos2d-pelien kehitysprosessia helpottamalla yksinkertaisten ja toistuvien rakenteiden luomista ns. “Saat mitä näet” (WYSIWYG) -filosofian kautta. Sillä voidaan myös toteuttaa kokonaisia pelejä web-selaimilla ajettuina Javascript-versioina.

BMFont

BMFont on sovellus True Type -fonttien muuttamiseksi bittikartoiksi. Näitä bittikarttoja voidaan käyttää tekstuureina, joilta luetaan kirjaisimia oheisen tiedoston koordinaattien mukaisesti.

Hiero

Javalla toteutettu BMFontin kilpailija.

Particle Designer

Editori partikkeliefektin luomiseen. Tukee Cocos2d-sovelluskehyksien lisäksi myös muita pelimoottoreita, kuten Moai, Starling ja Sparrow.

TexturePacker

Editori sprite-lakanoiden luomiseen.

Tiled Map Editor

Editori tekstuuritiileihin perustuvien karttojen suunnitteluun ja toteuttamiseen.

3.3 Lisenssi

Cocos2d-X on MIT-lisensoitu. Vapaana ohjelmistona Cocos2d-X:ää voidaan siis käyttää vapaasti sekä yksityisiin että kaupallisiin tarkoituksiin. MIT-lisenssi antaa käyttäjälle vapaat oikeudet muokata, kopioida ja käyttää lähdekoodia omissa projekteissaan, mikäli lisenssin teksti sisällytetään lähdekoodiin. Koska MIT-lisenssi ei ole niin sanottu copyleft-lisenssi, se ei vaadi lähdekoodin julkistamista missään vaiheessa käyttötarkoituksesta riippumatta. (MIT License 2013)

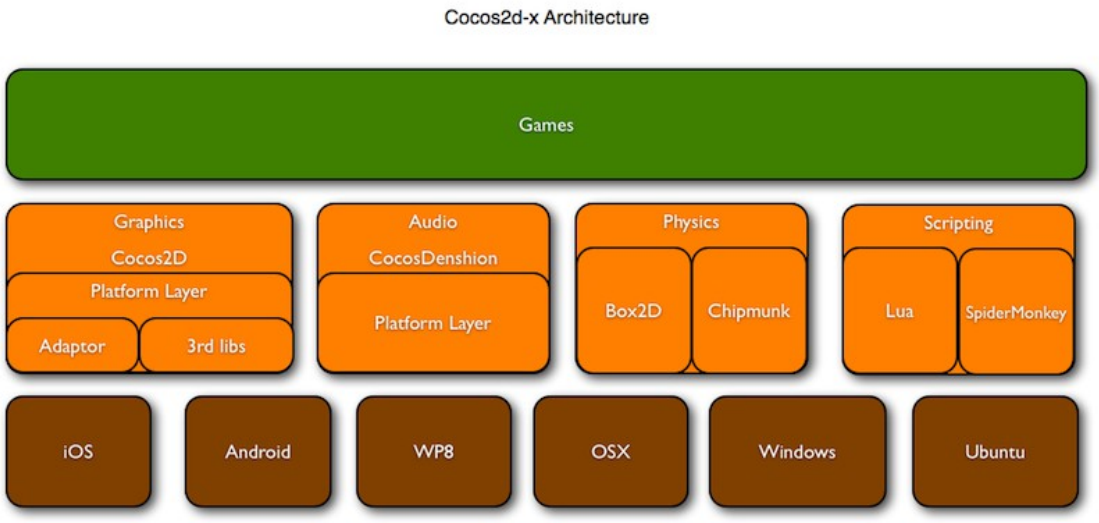
4 COCOS2D-X 2.0 TEORIASSA

4.1 Arkkitehtuuri

Koska pelit ovat reaaliaikasuoritteisina sovelluksina suorituskykykriittisiä, on Cocos2d-

x:n lähtökohtana mahdollisuus kääntää sovelluksesta alustakohtainen natiivikäännös. Kun tästä näkökulmasta halutaan rakentaa alustariippumattomia ratkaisuja, tulee rakentaa kirjasto joka sisältää rajapinnat itse sovelluksen ja kohdealustojen välille. Tätä varten täytyy osata erottaa universaalit toiminnot ja logiikat toisistaan. Pelilogiikat kuten fysiikkamallinnus tai taistelumekaniikat ovat universaaleja pelisovelluksen komponentteja, eikä niiden toteuttamiseen tarvita alustakohtaisia rajapintoja. Riittää että käytetylle ohjelmointikielelle löytyy alustakohtainen kääntäjä. Vastaavasti taas esimerkiksi äänet tai grafiikan piirtäminen vaativat toteutuksessaan alustakohtaisia rajapintoja joilla päästään käsiksi kohdelaitteen laitekomponentteihin.

Pääpiirteissään Cocos2d-X muodostuu neljästä peleille keskeisestä sovelluskomponenttista: grafiikkamoottorista, äänimoottorista, fysiikkakirjastoista ja tukiratkaisuista skriptikielille (ks. kuvio 3). Sekä grafiikka- että äänimoottori sisällyttävät yhteydet alustakohtaisille rajapinnoille. Grafiikkamoottori hyödyntää lisäksi kolmannen osapuolen kirjastoja, kuten matematiikkakirjasto KazMathia. Fysiikkakirjastot ovat alustariippumattomia komponentteja, eivätkä tarvitse minkäänlaista alustakohtaista kerrosta. (Cocos2d-x | Architecture and Directory Structure 2013)



KUVIO 3. Cocos2d-x -sovelluskehyksen arkkitehtuuri (Cocos2d-x Architecture 2013)

OpenGL ES 2.0

Android ja iOS -mobiilikäyttöjärjestelmien kohdalla Cocos2d-x 2.0:n grafiikkamoottorin sydän on OpenGL ES 2.0. OpenGL ES on isoveljensä OpenGL:n kevennetty, suljetuille järjestelmille tarkoitettu grafiikkaan erikoistunut ohjelmointirajapinta. Sitä käytetään esimerkiksi useissa puhelinkäyttöjärjestelmissä kuten Android ja iOS, kuin myös pelikonsoleissa. Versio 2.0 eroaa huomattavissa määrin versiosta 1.0 avoimemman käyttörakenteen vuoksi. Shader-ohjelmat mahdollistavat täydellisen hallinnan kuvan piirtämiseen. (OpenGL ES - The Standard for Embedded Accelerated 3D Graphics 2013)

Fysiikkakirjastot Box2D ja Chipmunk

Fysiikkamoottoriksi Cocos2d-x sovelluskehys tarjoaa kaksi mukaan integroitua vaihtoehtoa. Tämä ei kuitenkaan tarkoita, etteikö kehittäjä voisi käyttää jotain muuta tai omaa toteutustaan. Täytyy kuitenkin muistaa, että Cocos2d-x on sovelluskehys nimenomaan kaksiulotteisille sovelluksille, ja valmiiksi tarjotut fysiikkakirjastot ovat parhaita mahdollisia, saatavilla olevia avoimen lähdekoodin kirjastoja. Se kumpaa kehittäjä haluaa käyttää, on pitkälti makukysymys.

Box2D on vapaa ja ilmainen C++ -ohjelmointikielellä kirjoitettu avoimen lähdekoodin fysiikkamoottori. Sitä on käytetty lukuisissa hyvin menestyneissä kaupallisissa pelijulkaisuissa, kuten Angry Birds, Limbo ja Tiny Wings. Alustariippumattomana kirjastona se ei ole sidottu tiettyyn ympäristöön, vaan sitä voidaan käyttää monipuolisesti niin selainpeleissä kuin konsoli- tai mobiilialustoilla. Kaiken lisäksi se on muiden osapuolien toimesta käännetty lukuisille muilla kielille, kuten Javalle, Lua:lle tai JavaScriptille. (Box2D 2013)

Ominaisuuksillaan Box2d tarjoaa monipuoliset mahdollisuudet jäykkien kappaleiden simulointiin. Se tarjoaa oletuksena muodot yleisimmille kappaleille, kuten laatikoille ja palloille. Perusmuodoista kehittäjä voi rakentaa monimutkaisempia rakenteita käyttämällä liitoksia (joint). Kappaleet reagoivat toisiinsa kappaleisiin vaikuttavien voimien mukaisesti. Box2D:stä poiketen **Chipmunk** on kirjoitettu C-kielellä. Se ei siis tarjoa oliopohjaisia rakenteita ja mahdollisuuksia.

Skriptikielet Lua ja JavaScript

Erityisesti AAA-pelisovellusten kääntöajat voidaan mitata jopa tunneissa. Mikäli pelisovelluksen lähdekoodiin tehdään pienikin muutos, joudutaan koko lähdekoodi kääntämään uudelleen, mikä kuormittaa tuottavuutta. Ei ole epätavallista, että usein muutettavat pelisovelluksen osat, kuten erinäiset asetukset ja pelilogiikat, toteutetaan ajon aikana tulkattavilla skriptikielillä. Tällöin ne ovat myös helpommin niiden henkilöiden muokattavissa, joiden ammattitaito ei perustu ensisijaisesti ohjelmointitaitoon. Tällaisia henkilöitä ovat esimerkiksi pelisuunnittelijat.

Cocos2d-x tukee kahta skriptikieltä: Lua ja JavaScript (SpiderMonkey). Skriptikieliä käyttäessä tulee huomioida niiden tuki Cocos2d-x -sovelluskehyksessä eri alustojen kanssa (ks. taulukko 1).

TAULUKKO 1. Cocos2d-x ja skriptikielien alustakohtainen tuki.

	iOS	Android	WP8	Win32	OS X	Linux
Lua	x	x		x	x	x
JavaScript	x	x		x		

4.2 AppDelegate

AppDelegate on välttämätön osa jokaisen iOS-sovelluksen rakennetta ja sen instanssi luodaan sovelluksen käynnistysvaiheessa. Luonteeltaan AppDelegate on singleton-olio. Sen tehtäviin kuuluvat muun muassa sovelluksen käynnistyksen aikaiset alustukset ja sovelluksen tilamuutokset, kuten sovelluksen sammuttaminen tai siirtyminen taustaprosessiksi ja sieltä takaisin. Vaikka AppDelegaten ja sovelluksen pääsilmukan

välinen kommunikointi on piilotettu, se sisältää sovelluskehittäjälle olennaisia virtuaalisia funktioita, joita suoritetaan aina tietyssä sovelluksen tilamuutoksen vaiheessa. (iOS App Programming Guide: Core App Objects 2013)

Koska iOS on oleellinen pelinkehittäjän kohdealusta, ja tilamuutokset kehittäjälle olennaisia tapahtumia alustasta riippumatta, on AppDelegate toteutettu myös Cocos2d-x:ssä tarjoten alustasta riippumatta yhtenäisen rajapinnan sovelluksen tilamuutoksiin. Cocos2d-kehittäkälle AppDelegate on se paikka, jossa muun muassa koko sovelluksen oliohierarkian ensimmäinen solmuolio CCScene ja OpenGL -piirto alustetaan.

4.3 Singletonit

4.3.1 Singletoneista

Ohjelmistoja kehitettäessä tulee toisinaan vastaan tilanteita, jolloin sovelluksen toiminnan kannalta on tärkeä, että jokin sen yksittäinen komponentti on oltava saatavilla sovelluksen jokaisesta osasta ja sen on tarjottava kysyjästä riippumatta samat attribuutit tietyllä ajan hetkellä. Tällöin käytetään singleton-oliota. Singleton-oliosta on olemassa vain yksi ainoa instanssi. Singletonit ovat yksi kiistanalaisimpia aiheita ohjelmoinnin saralta, eikä niitä tulisi käyttää liian heppoisin perustein. (Singleton Pattern 2013)

4.3.2 CCDirector

CCDirector on kokonaisuuden kannalta ja singleton-luonteensa vuoksi keskeisin Cocos2d-sovelluksen olio. Sen tehtäviin kuuluvat lukuisat pelien keskeisimmät toiminnot, asetukset ja muuttujat, jotka ovat yhteisiä ja joihin on hyvä päästä käsiksi pelisovelluksen jokaisesta nurkasta. CCDirectorin kautta voidaan selvittää kulloinkin suorituksessa oleva CCScene-olio, ja tarvittaessa vaihtaa sen tilalle toinen, joko väliaikai-

sesti tai kokonaan. (Itterheim & Löw 2012, 61)

CCDirector tarjoaa suoran pääsyn Cocos2d-sovelluksen OpenGL-näkymään ja ikkunaan, sekä sen avulla voidaan suorittaa tarvittavat koordinaatistomuunnoksen itse pelisovelluksen ja laitteen kosketusjärjestelmän välillä. Sen kautta voidaan myös vaikuttaa kaikkeen OpenGL-piirtoon yleisellä tasolla, kuten projektioon tai syvyystesteihin (depth tests). (Itterheim & Löw 2012, 62)

CCDirector isännöi myös muita Cocos2d:n singleton-luokkia. **CCScheduler** ja **CCActionManager** huolehtivat pelilogiikan kannalta tärkeistä asioista, ajoituksista ja tapahtumista. **CCTouchDispatcher** ja **CCEventDispatcher** pelaajasyötteen lukemisesta käytetyn alustan omista rajapinnoista. CCDirectorin kautta huolehditaan myös pelin suurpiirteisistä tilamuutoksista, kuten pelin pysäyttäminen, peliin palaaminen ja pelin lopettaminen.(Itterheim & Löw 2012, 95)

4.3.3 Cache-luokat

Resurssien, kuten kuvan ja äänen, lataaminen edellyttää niiden lukemista tallennusmedialta, mikä on tietotekniikan perspektiivissä erittäin hidasta. Lisäksi kyseinen toimenpide sisältää muistinvarausta, jolla on hetkellinen mutta suora vaikutus suorituskykyyn. Koska pelit ovat reaaliaikasuoritteisia, on suorituskyvyllä olennainen merkitys. Mikäli esimerkiksi pelihahmon tekstuuri ladataan muistiin joka kerta kun uusi hahmo syntyy, voivat vaikutukset pelin pyörimisnopeuteen olla dramaattiset. Ratkaisuksi on kehitetty cache eli välimuisti. Cachen periaate peleissä on, että sinne ladataan valmiiksi pelissä tarvittavat resurssit. Näin ollen säästetään suoritusajan päivitysnopeudessa.

Cocos2d-x pitää sisällään neljä cache-luokkaa, joilla pyritään optimoimaan pelisovelluksen toimintaa ja suorituskykyä varmistamalla ettei resursseja ladata muistiin duplikaatteina ja minimoimalla muun muassa CPU:n ja GPU:n välistä liikennettä. Täydellisin esimerkki tästä on **CCTextureCache**, johon ladataan piirrettävät tekstuurit. Cachesta löytyvää tekstuuria ei tarvitse siirtää GPU:lle joka kerta erikseen sitä piirrettäessä.

Cachessa olevan tekstuurin ominaisuuksia, kuten koko tai väriä, ei voi enää muokata. Toisaalta sen piirtoon voidaan vaikuttaa shader-ohjelmilla. (Cocos2d-x | Texture Cache 2013)

Lisäksi Cocos2d-x:stä löytyvät omat cache-luokat sprite-ruuduille, animaatioille ja shader-ohjelmille: **CCAnimationCache**, **CCSpriteFrameCache**, **CCShaderCache**.

4.4 Solmuluokat

4.4.1 Yleisluokka CCNode

Pelit jotka muistuttavat enemmän tai vähemmän reaalimaailmaa, voidaan käsittää eräänlaisina kevytsimulaatioina. Ne sisältävät kaksi- tai kolmiulotteisen aika-avaruuden, johon sijoitetaan toisiinsa vaikuttavia olioita. Näiden olioiden mallintamiseen ja niiden käyttäytymisen simuloimiseen saadaan loistavat työkalut olio-ohjelmoinnista. Cocos2d-sovelluksissa pelimaailman käsitteelliset oliot, olivat ne sitten konkreettisesti näkyviä tai ei, toteutetaan erilaisilla solmuluokilla. Näitä olioita voivat olla esimerkiksi yksilöt kuten koira, tai ryhmät kuten koiralauma. Olioilla voidaan tarkoittaa myös vähemmän konkreettisia olioita kuten säännöstö jääkiekko-ottelussa.

CCNode on eräänlainen yleissolmuluokka, joka toimii jokaisen solmuluokan perustana. Se sisältää jokaiselle solmuluokalle ominaiset attribuutit ja toimintojen toteutukset. Olemukseltaan CCNode-olio on abstraktimainen, sillä se ei sisällä minkäänlaista visuaalista ulkomuotoa. Tämän vuoksi se soveltuu mainiosti ryhmäkäsitteiden, kuten ”parvi” tai ”lauma”, olioiksi tai symboliksi.

Koska solmuluokan on tarkoitus simuloida esimerkiksi jotain yksittäistä reaalimaailman olioon rinnastettavaa instanssia, sisältää CCNode-olio muutamia aika-avaruuteen liittyviä attribuutteja. Jokaisella reaalimaailman olennolla, oli se näkyvä tai ei, on esimerkiksi oma positionsa aika-avaruudessa. Kaksiulotteisessa maailmassa tämä kä-

sittää liukulukuarvot koordinaatiston akseleilla x ja y. Kappaleella on myös rotaatio, joka kertoo kappaleen kulmallisen orientoitumisen ajan hetkellä x suhteessa aika-avaruuden koordinaatistoon. Jokaisella fyysisellä olennolla on myös koko, ja ajan suhteen muuttuva mittaskaala.

4.4.2 Kulissiluokka CCSene

Käsite kulissi on Cocos2d:n kontekstissa abstrakti. Se ei ole mitään konkreettista, mutta koko Cocos2d:n oliohierarkian ensimmäisenä solmuoliona se on välttämätön, ja sisältää jokaisen solmuluokan instanssin joko suoraan tai jälkipolvien lapsina. Se siis toimii eräänlaisena säiliönä kaikille muille pelaajalle näkyville ja pelaajan toimintojen kannalta konkreettisille rakenteille ja olioille, eikä pidä sisällään mitään omaa erityistä toiminnallisuutta. Toimiakseen CCDirector vaatii vähintään yhden kulissin. (Itterheim & Löw 2012, 39,62)

Cocos2d on suunniteltu pyörittämään aktiivisena vain yhtä kulissia kerrallaan. Kulissi voidaan vaihtaa toiseen missä tahansa Cocos2d-sovelluksen suorituksen vaiheessa. Mutta koska uusi kulissi ladataan kokonaisuudessaan muistiin ennen kuin edellinen hävitetään, on muistiin ladattuna yhtäaikaisesti kaksi kokonaista kulissia. Näissä tilanteissa tulee varmistaa että muistia on saatavilla riittävästi. Kulissi voidaan myös syrjäyttää ja jättää muistiin uuden tieltä, jotta siihen voidaan palata myöhemmin. Kulissien vaihdot on mahdollista suorittaa animoiduin efektein CCSceneTransition-luokalla. (Itterheim & Löw 2012, 62-64)

CCScenellä on samat mitat kuin OpenGL-näkymällä. Käytännössä tämä on sama kuin suorituslaitteen näytön resoluutio. Koska kulissi on olennainen osa Cocos2d-sovellusta, ensimmäisen kerran se luodaan heti AppDelegatessa. Periaatteessa kulissin lapseksi voidaan liittää minkä tahansa solmuluokan instanssi, mutta useimmiten tämä on CCLayer tai sen johdannainen. (Itterheim & Löw 2012, 39,69)

4.4.3 Tasoluokka CCLayer

CCLayer on solmuolio, joka voidaan käsittää eräänlaisena kuvatasona. Se periytyy suoraan CCNode-luokasta ollen hyvinkin samankaltainen. Kuten CCScene-olio, CCLayer-olio saa samat mitat kuin Cocos2D-x -sovelluksen käyttämä OpenGL-näkymä. Kuitenkin se omaa yhden erityispiirteen: se kykenee vastaanottamaan pelaajasyötettä. (Itterheim & Löw 2012, 39,68-69)

CCLayer-luokka pitää sisällään toteutukset kykyyn vastaanottaa monikosketusnäytön kosketustapahtumien koordinaatit ja kiihtyvyysanturin arvolukemat. Koska Cocos2d-x ei sovelluskehyksenä rajoitu pelkästään mobiilialustoihin, voidaan työpöytäympäristössä vastaanottaa pelaajasyötettä myös hiireltä ja näppäimistöltä. Jokaisen neljän laitteen lukeminen täytyy aktivoida manuaalisesti. (Itterheim & Löw 2012, 39,68-69)

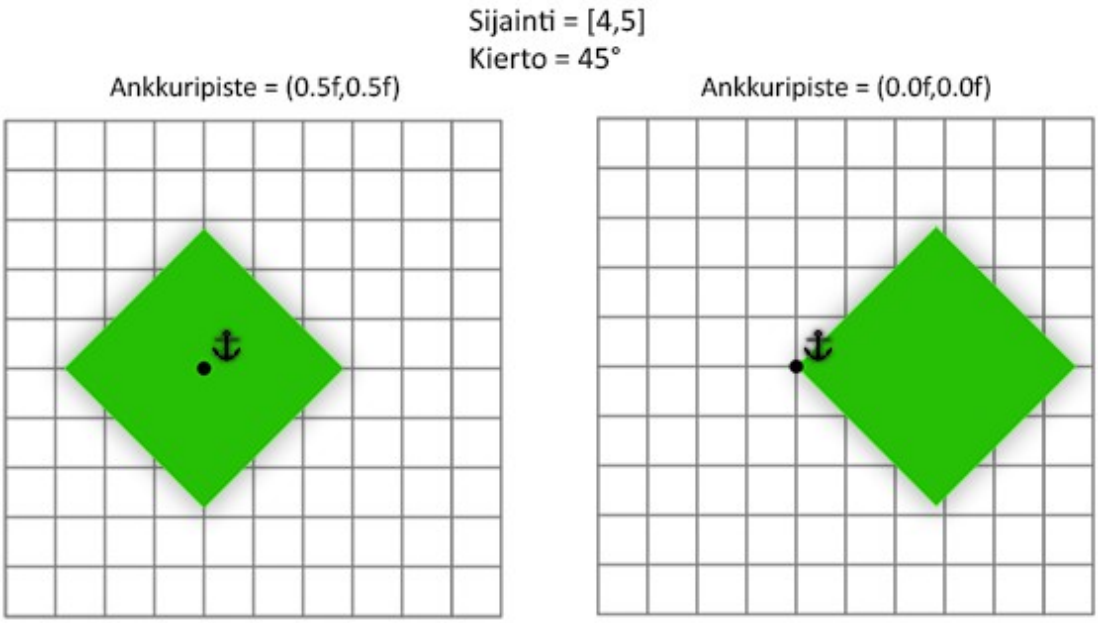
Suorituskyvyn kannalta ei ole olennaista kuinka monta tasoa Cocos2d-sovelluksessa on, yhtä poikkeusta lukuun ottamatta. Syötteen lukutapahtumat ovat verrattain suorituskykyraskaita, eikä niitä tulisi lukea useilla tasoilla samanaikaisesti. Käytännössä riittääkin että vain yksi taso vastaanottaa signaalit. Koska signaalit luetaan alustakohtaisella API:llä, käytetään Cocos2d-sovelluksen omassa “maailmassa” ja piirretään OpenGL-näkymään, tulee ne kääntää käyttökontekstin koordinaatistoon ja vaatimaan muotoon. (Itterheim & Löw 2012, 69)

4.4.4 Sprite-luokka CCSprite

Pelit sisältävät visuaalisen muodon omaavia entiteettejä (pelihahmo, projektiilit tai viholliset), joiden piirtämiseen on kaksiulotteisissa peleissä käytetty perinteisesti spritejä. Spritet ovat pelinäkymään piirrettäviä kaksiulotteisia kuvia. Niiden toteutuksellinen vastine Cocos2d-x -sovelluskehyksessä on luokka **CCSprite**. Cocos2d-x:n kontekstissa käsite sprite on kuitenkin enemmän kuin pelkkä visuaalinen muoto, sillä CCNoden perillisenä se voidaan käsittää täysimittaisena pelioliona, jolla on oma olotilansa aika-avaruudessa.

CCSprite on yleensä Cocos2d-sovelluksen ylivoimaisesti käytetyin solmuluokka. CCNode-oliosta poiketen se omaa oletusarvoisesti konkreettisesti näkyvän ominaisuuden,

piirrettävän tekstuurin. Sen myötä kaikille solmuluokille ominainen attribuutti, **ankkuripiste**, muuttuu merkityksellisemmäksi. Ankkuripiste on eräänlainen sprite-olion nollapiste, jonka kautta määritellään olion sijainti, rotaatio tai skaalaus (ks. kuvio 4).

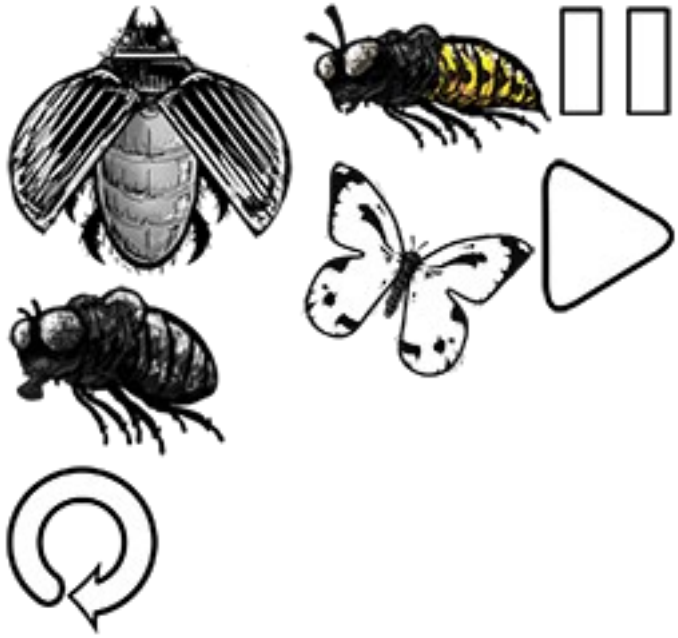


KUVIO 4. CCSprite-olion sijoittuminen koordinaatistoon eri ankkuripisteen arvoilla

Ankkuripisteen koordinaatit, arvoltaan 0.0-1.0, ovat aina suhteelliset CCSprite-olion tekstuurin kokoon. Esimerkiksi ankkuripisteen oletusarvo [0.5,0.5] kohdistuu CCSprite-olion tekstuurin pinta-alan keskipisteeseen. Vastaavasti arvon ollessa [0,0] sijaitsee ankkuripiste vasemmassa alanurkassa. Tällöin CCSprite-olio pyörähtää rotaatiossa suhteessa oman vasemman alanurkkansa ympäri ja se sijoitetaan sijaintikoordinaattiinsa sen kohdalta.

Suorituskyvyn parantamiseksi on hyvä käyttää niin sanottuja sprite-lakanoita CCSprite-olioiden tekstuurien pakkaamiseen (ks. kuvio 5). Sprite-lakana on yksittäinen suurehko kuvatiedosto, johon on sijoiteltu useita yksittäisiä pienempiä tekstuureita. Kun sprite-lakana luodaan esimerkiksi TexturePacker-sovelluksella, luo sovellus kuvatiedoston lisäksi plist-tiedoston, joka sisältää tiedot jokaisen yksittäisen sprite-tekstuurin

lukemiseksi lakanasta.

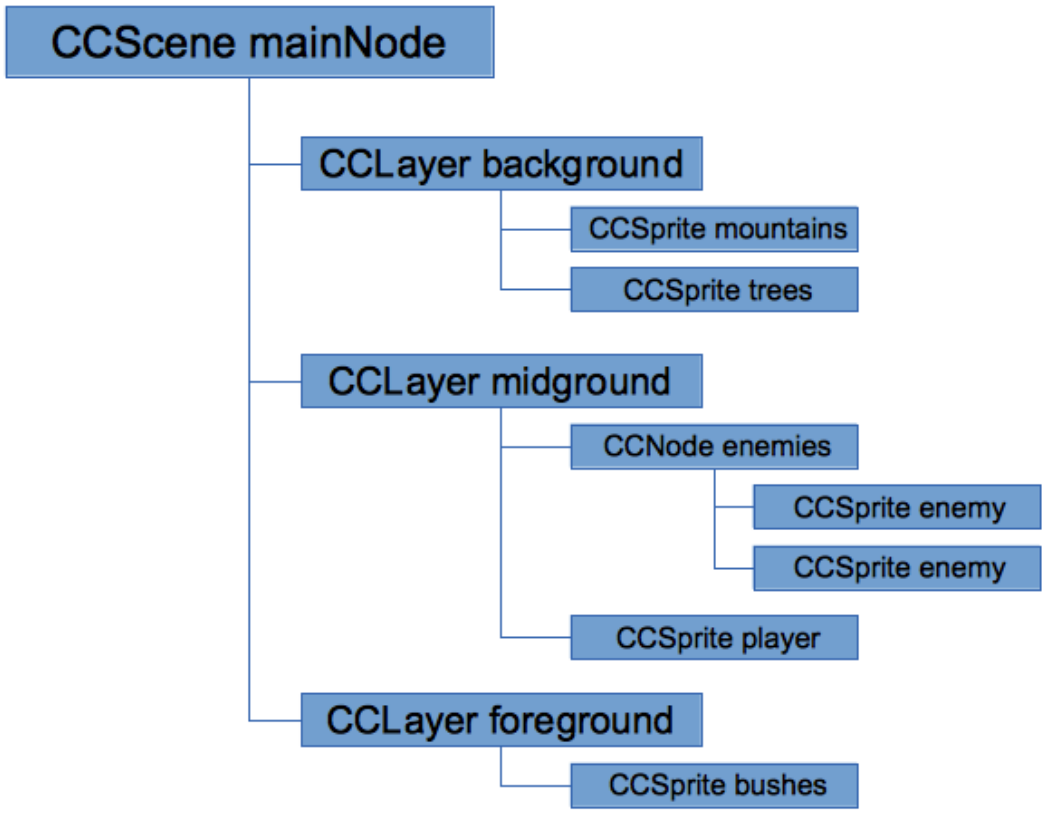


KUVIO 5. Esimerkki sprite-lakanasta

4.4.5 Oliohierarkia

Jokaiselle solmuluokan oliolle on tyypillistä sen sijoittuminen Cocos2d-x -sovelluksen sisäiseen oliohierarkiaan. Tämä tarkoittaa, että jokainen sovelluksen solmuolio on kytkettynä isäntäolioon ollen oma haaransa puurakenteisessa instanssikokoelmassa. Oliohierarkia alkaa aina yhdestä CCScene-oliosta. Se miten loput oliopuusta rakentuu, on täysin kiinni suunnitellusta arkkitehtuurista (ks. kuvio 6). Ainoa rajoite on, että oliolla voi olla vain yksi ainut isäntä. Lapsia oliolla voi olla rajattomasti. Lisäksi niin kauan kuin kyseessä on vapaa solmuolio, voidaan se liittää mihin tahansa

solmuolioon riippumatta siitä, onko kyseessä CCNode, CCLayer vai CCSprite.



KUVIO 6. Mahdollinen oliohierarkian toteutus

Oliohierarkia vaikuttaa ennen kaikkea olioiden piirtojärjestykseen. Lähtökohtaisesti mitä ylempänä hierarkiassa olio on, sitä päällimmäiseksi se piirretään. Lisäksi saman isäntäolion olioiden piirtojärjestykseen voidaan vaikuttaa solmuolion attribuutilla **Z-index**. Mitä suurempi indeksi on, sitä päällimmäiseksi se piirretään. Indeksin oletusarvo on 0, ja mikäli kahdella oliolla on sama arvo, piirretään päällimmäisenä olioista uudempi. Positiivisen arvon omaavat oliot piirretään isäntänsä päälle, kun taas negatiivisen arvon oliot piirretään isäntäolion alle.

Solmuolion liittäminen hierarkiaan ei sinänsä ole välttämätöntä, mutta mikäli näin ei tehdä, se hävitetään Cocos2d:n automaattisissa rakenteissa tarpeettomana ja sen varaama muisti vapautetaan. Olio voidaan toisaalta tilapäisesti poistaa muistinhallinnan piiristä, mutta tällöin tulee huomioida ettei hierarkiaan kuulumatonta oliota piirretä, eikä sille voida suorittaa toimintoluokkiin perustuvia toimenpiteitä.

4.4.6 CCLabelTTF, CCLabelAtlas ja CCLabelBMFont

CCLabelTTF on helpoin tapa tulostaa tekstiä näytölle. Se luo piirrettävän tekstuurin oikeasta True Type -fontista. Tämä on kuitenkin erittäin hidas prosessi, joten tekstin vaihtaminen suorituksen aikana luo selkeän piikin. Mikäli teksti ei suorituksen aikana muutu, toimii ja piirretään CCLabelTTF-olio yhtä nopeasti kuin mikä tahansa muu CCSprite-olio. (Cocos2d-x | Text Labels 2013)

Tilanteissa, joissa tekstitarraa päivitetään usein suorituskykykriittisen ajon aikana, kuten peliolion statusindikaattoreissa, kannattaa käyttää sprite-lakanaan perustuvia ratkaisuja. **CCLabelBMFont** kirjoittaa ja piirtää halutun merkkijonon PNG-kuvan ja fnt-tiedoston perusteella. Fnt-tiedostosta löytyvät kaikki määrittelyt kirjainmerkkien lukemiseen tekstuuriatlaksesta. Yhteensopivan tekstuuriatlaksen ja fnt-tiedoston voi True Type -fontista luoda kolmannen osapuolen työkaluilla kuten **Hiero**. (Cocos2d-x | Text Labels 2013)

Nopeus ei tule kuitenkaan ilmaiseksi. Siinä missä True Type -fontit ovat dynaamisia ja vievät vähän keskusmuistia, ovat tekstuureihin perustuvat staattiset tekstit suurempi rasite muistinkulutukselle että levityspaketin koolle. (Cocos2d-x | Text Labels 2013)

4.4.7 CCMenu ja CCMenuItem

CCMenu periytyy suoraan CCLayer-luokasta, mutta siitä poiketen se hyväksyy lapsikseen vain **CCMenuItem**-olioita. CCMenuItem-oliot ovat valikon painikkeita. Näkyvältä osaltaan painikkeet ovat joko teksti- tai kuvamuotoisia. Kun pelaaja koskettaa painiketta kosketusnäytöltä, painike suorittaa siihen liitetyn funktion. Painike imaisee sii-

hen kohdistuneen kosketustapahtuman, eikä sillä siten ole vaikutusta mihinkään muuhun. Painikkeet sijoittuvat aina suhteessa niiden vanhempaansa CCMenu-olioon. (Itterheim & Löw 2012, 76-78)

CCMenuItem-oliot perustuvat joko tekstuuriin tai fonttiin. **CCMenuItemImage** luodaan käyttäen kuvatiedostoa tai cachetettua tekstuuria, kun taasen **CCMenuItemSprite** käyttää jo olemassa olevaa CCSprite-oliota. Tulee kuitenkin huomata, ettei CCSprite-olion tule ennestään kuulua oliohierarkiaan, sillä se lisätään CCMenuItemSprite-olion lapseksi tämän luontiprosessissa. (Itterheim & Löw 2012, 76-78)

CCMenuItemImagen ja -Spriten fontilliset vastineet ovat **CCMenuItemFont** ja **CCMenuItemLabel**. CCMenuItemFont alustetaan merkkijonolla ja CCMenuItemLabel vaatii alustuksessa valmiin CCLabel-olion. Jokaisella CCMenuItem-oliolla on oletuksena kaksi tilaa, “noclick” ja “onclick”, joille molemmille voidaan määritellä omat erilliset kuvat tai tekstitarransa. Mikäli painikkeelle halutaan kaksi erillistä “noclick”-tilaa (esim. vahvistimen virtanäppäin), voidaan käyttää **CCMenuItemToggle**-oliota. Tällöin painikkeelle voidaan määritellä erillisiksi tiloiksi kaksi erillistä CCMenuItem-sukuista painiketta. (Itterheim & Löw 2012, 76-78)

4.4.8 SpriteBatchNode

Mikäli CCSprite-olio lisätään minkä tahansa tavanomaisen solmuolion, kuten CCLayer- tai CCNode-olion lapseksi, suoritetaan sen piirto CCSprite-olion omalla piirtokäskyllä. Kun tekstuuri piirretään, tarvitaan monenlaisia CPU-suoritteisia toimintoja. Määritellään GPU:lle käytettävä shader-ohjelma, käytettävä tekstuuri ynnä muuta tarpeellista dataa. Ja koska pelit ovat täynnä ulkomuodoltaan samanlaisia peliolentoja, kuten projektiilit, tehdään ne erikseen piirrettäessä turhaa suorituskykyrasitteista työtä.

Batchaus on tekniikka, jolla nopeutetaan useiden peliolentojen piirtotoimintoja piirtämällä ne yhdestä ja samasta cacheen tallennetusta tekstuurista, yhdellä ainoalla piirtokäskyllä. Kun CCSprite-olio lisätään lapseksi CCSpriteBatchNode-oliolle, suoritetaan lapsen piirto vanhemman piirtokäskyllä. CCSpriteBatchNode hyväksyy lapsek-

seen vain CCSprite-olioita, joilla on sama tekstuuri. Tämä ei kuitenkaan tarkoita, että jokaisen CCSprite-olion olisi oltava samanlainen. Sprite-lakanan käyttäminen mahdollistaa lukuisten erilaisten CCSprite-olioiden batchaamisen samalla isäntäoliolla. Olennaista on, että tekstuurit löytyvät samasta samasta tiedostosta. (Itterheim & Löw 2012, 163-165)

Mitä vähemmän CCSpriteBatchNode-olioita, ja mitä enemmän CCSprite-olioita yhdessä CCSpriteBatchNode-oliossa, sitä parempi lopputulos pelisovelluksen suorituskyvyn kannalta. Niin kauan kuin vaatimus samasta tekstuurista täytyy, mitään varsinaisia rajoja batchaukselle ei ole. Yksittäisen CCSpriteBatchNode-olion hyöty kannattaakin pyrkiä maksimoimaan. Käytännössä tätä kuitenkin rajoittavat muut, grafiikkapiirtoriippumattomat tekijät, kuten pelisovelluksen arkkitehtuuri.

5 COCOS2D-X KÄYTÄNNÖSSÄ

5.1 Kehitysympäristön pystyttäminen

5.1.1 Cocos2d-x -paketti, sen rakenne ja sisältö

Cocos2d-x -sovelluskehyksen voi ladata projektin kotisivuilta osoitteesta http://www.cocos2d-x.org. Käytännössä sovelluskehys on zip-paketoitu hakemisto täynnä lähdekooditiedostoja, skriptejä sekä dokumentaatio- ja projektitiedostoja. Paketti voidaan purkaa vapaasti valittavaan sijaintiin, mutta suositeltavaa on tehdä kehitystyötä varten oma hakemistonsa, jonne kasataan sovelluskehyksen lisäksi kaikki alustakohtaiset työkalut ja Cocos2d-x -projektit.

Cocos2d -hakemisto

Cocos2d-hakemisto sisältää sovelluskehyksen ydintoiminnallisuuden lähdekooditie-

dostot: universaalit Cocos2d-x -luokat, projektipohjia, alustakohtaisia kirjastoja sekä kolmannen osapuolen matematiikkakirjaston KazMathin. Juuresta löytyvä include-hakemisto sisältää olennaiset otsaketiedostot sekä platform-hakemisto alustakohtaiset kirjastot Cocos2d-kirjastojen liittämiseksi alustakohtaisiin projekteihin.

CocosDenshion

Tämä hakemisto sisältää äänimoottorin alustakohtaiset lähdekoodi- ja projektitiedostot. Juuresta löytyvä include-hakemisto sisältää otsaketiedostot kirjaston liittämiseksi projekteihin.

Document

Document-hakemisto sisältää offline-dokumentaation. Sen lukeminen vaatii Doxygen-sovelluksen.

Extensions

Extensions-hakemisto sisältää kaikki Cocos2d-x:n laajennetut luokat kuten Box2d- tai Chipmunk-yhteensopivan CCSprite-luokan CCPhysicsSprite. Lisäksi hakemistosta löytyy mm. luokka verkkotoiminnallisuuksiin.

External

Tämä hakemisto sisältää suurimmat kolmannen osapuolen komponentit. Nämä ovat fysiikkakirjastot Box2D ja Chipmunk sekä relaatiotietokantajärjestelmä SQLite.

Samples

Samples-hakemisto sisältää lähdekoodiesimerkit kätevinä, suoritusvalmiina projekteina.

Skriptitiedostot

Juuresta löytyy tärkeitä skriptitiedostoja projektien luomiseen ja ohjelmointiympäristöjen projektipohjien asentamiseen.

Muut

Lisäksi Cocos2d-x -paketista löytyvät mm. hakemistot erilaisille pienille työkaluille, eri sovelluskomponenttien lisensseille ja plugineille.

5.1.2 Kehitysympäristön pystyttäminen OS X -käyttöjärjestelmässä

Seuraavat ohjeet lähtevät olettamuksesta, että käyttöjärjestelmän asennus on tuore, eikä sisällä ennalta tarvittavia työkaluja tai asetuksia.

iOS-kehitysympäristö

iOS-kehitysympäristön pystyttäminen OS X -käyttöjärjestelmässä on kaikkien kehitysympäristöjen asennusprosesseista suoraviivaisin. Asennettuasi Xcode-ohjelmointiympäristön ja ladattuasi Cocos2d-X-paketin suorita paketin juuresta löytyvä skripti terminaalissa komennolla "./install-templates-xcode.sh". Kyseinen skripti kopioi tarvittavat Cocos2d-kirjastot Xcoden käyttämään lokaatioon ja asentaa valmiit sovelluspohjat.

Android-kehitysympäristö

Android-kehitysympäristö pystytetään seuraavasti:

1. Asennetaan Java-kehitykseen tarkoitettu Java Development kit (JDK). Käyttöjärjestelmäkohtaisen asennuspaketin löytää Oraclen verkkosivuilta http://www.oracle.com. Asennettavaksi suositellaan uusinta versiota. Mikäli työasemalta löytyy jo vanhempi versio, kannattaa paketti päivittää.
2. Android-sovelluskehitykseen tarvittavat työkalut ladataan Googlen ylläpitämältä Android-kehittäjille tarkoitetulta tukisivustolta http://developer.android.com. Helpoin vaihtoehto on kaikki tarvittavat työkalut sisältävä ADT Bundleksi nimetty paketti. Se sisältää Eclipse-ohjelmointiympäristön tarvittavilla laajennuksilla, Android SDK -työkalut, Androidin alustatyökalut, viimeisimmän Android-alustan sekä järjestelmäkuvan viimeisimmästä Android-versiosta emulaattoria varten. Paketti voidaan purkaa vapaavalintaiseen sijaintiin, mutta suositeltava käytäntö on luoda hakemisto kehitystyökaluille kotikansion juureen.
3. Ladataan tuettavien Android-versioiden API-paketit. Cocos2d-x 2.0 tukee And-

roid-versioita versiosta 2.1 uusimpaan. Paketit asennetaan Bundle-paketin hakemistosta sdk/tools löytyvällä android-nimisellä sovelluksella.

4. Luodaan Android-emulaattorissa käytettävät virtuaalilaitteet haluttujen API-tasojen mukaisesti.
5. Asennetaan Android Native Development Kit (NDK). Paketti ladataan Android-kehittäjien tukisivustolta ja voidaan purkaa haluttuun hakemistoon. Suositeltava sijainti on saman hakemiston juuri kuin missä ADT Bundle sijaitsee.
6. Sovelluksien kääntämiseksi komentoriviltä tarvitsee asentaa komentorivityökalut Xcoden asetuspaneelin Downloads-välilehdeltä.
7. Viedään Cocos2d-x:n Android-kohtaiset kirjastot (cocos2dx/platform/android/java) Eclipsen valikosta New/Other löytyvällä toiminnolla “Android Project from Existing Code”. Tuleviin projekteihin voidaan joutua vielä lisäämään viittaus projektikohtaisista asetuksista.

Asennusohjeet on tässä opinnäytetyössä selitetty hyvin yleispiirteisesti, sillä esimerkiksi Android-SDK ja Eclipse kehittyvät ja muuttuvat kiivastakin tahtia. Ajantasaisimmat ohjeet löytyvätkin mahdollisesti Cocos2d-x -projektin kotisivuilta.

5.2 Alustariippumaton Cocos2d-x -projekti

5.2.1 Alustarippumattoman projektin luominen

IOS-projektin luominen

iOS-projekti luodaan normaalien käytäntöjen mukaisesti Xcoden kautta. Valittavana on muutamia projektipohjia, jotka sisältävät valmiiksi implementoituja komponentteja kuten fysiikkakirjaston. Komponentit voidaan lisätä projektiin myös jälkikäteen. Tallennussijainnilla ei ole merkitystä, sillä projekti tullaan myöhemmin yhdistämään Android-projektiin alustariippumattoman projektikokonaisuuden luomiseksi.

Android-projektin luominen

Androidille tarkoitettu Cocos2d-x -projekti luodaan siihen tarkoitukseen toteutetulla, Cocos2d-x -paketin juuresta löytyvällä skriptillä “create-android-project.sh”. Skriptin alkuun määriteltyjen ympäristömuuttujien NDK_ROOT_LOCAL ja ANDROID_SDK_ROOT_LOCAL arvoiksi tulee muuttaa oman työaseman Android NDK:n ja SDK:n hakemistopolut absoluuttisina.

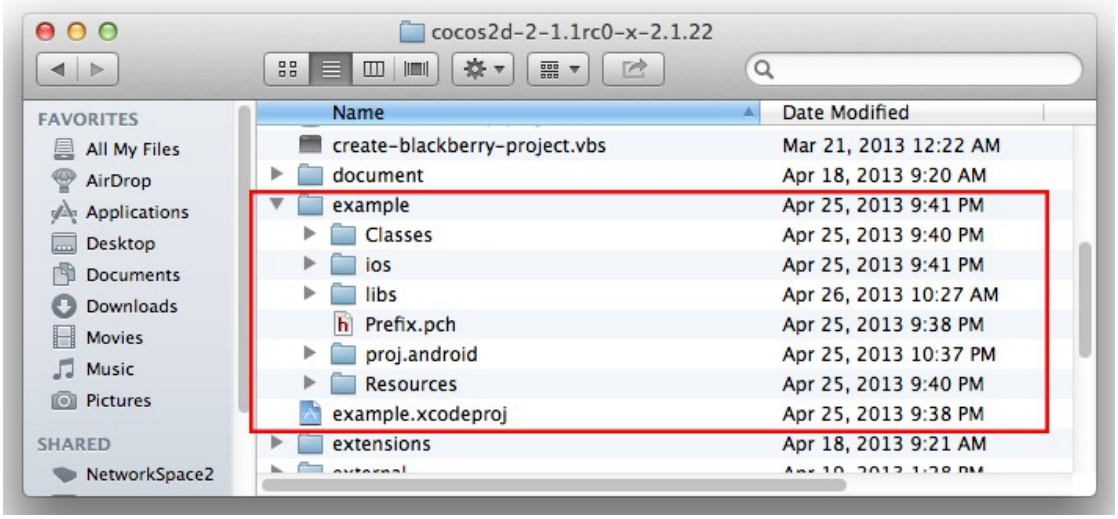
Skriptin suorituksen aikana kysytään tarvittavia tietoja kuten projektin nimi ja käytettävä Android-versio. Mikäli tavoiteltua API-tasoa ei löydy, tulee se asentaa sdk:n tools-hakemiston android-sovelluksella. Projekti voidaan lisätä Eclipseen ajoa varten Android Project from Existing Code -toiminnolla.

Projektien yhdistäminen

Projektien yhdistämiseksi kopioidaan iOS-projektikansion juuresta .xcodeproj-tiedostopäätteinen Xcode-projektitiedosto. Android-projektikansioon kopioidaan iOS-projektikansiosta hakemistot “lib” ja “ios”, sekä lisätään Resource-hakemiston puuttuvat tiedostot iOS-projektikansion vastaavasta hakemistosta.

5.2.2 Alustariippumattoman projektin rakenne

Vastaluodun alustariippumattoman projektin tulisi sisältää asianmukaiset tiedostot ja hakemistot alihakemistoineen (ks. kuvio 7). Käytetyn esimerkkiprojektin nimi on “example”.



KUVIO 7. Vasta luodun ja yhdistetyn HelloWorld-projektin sisältö

Classes

Classes-hakemisto sisältää sovelluskohtaiset, alustariippumattomat luokat, toisin sanottuna pelikohtaisen toteutuksen. Oletusarvoisesti tuoreen projektin hakemisto sisältää otsake- ja lähdekooditiedostot luokille AppDelegate ja HelloWorldScene. Kun hakemistoon lisätään uusia luokkia, tulee niihin viitata hakemistosta proj.android/jni löytyvässä tiedostossa android.mk.

Resources

Resources-hakemisto sisältää sovelluksen resurssitiedostot. Nämä tiedostot sisältävät sovelluksen käyttämää ääntä ja kuvaa. Tähän hakemistoon sijoitetaan myös xml-muotoiset tiedostot kuten sprite-kollaasien plist-tiedostot.

Alustakohtaiset hakemistot ios ja proj.android

Ios- ja proj.android -hakemistot sisältävät alustakohtaisten projektirakenteiden luokat ja tiedostot. Ios-hakemisto sisältää siis iOS-sovelluksen peruselementit kuten AppController ja RootViewController. Proj.android -hakemisto puolestaan sisältää Androidsovelluksen Java-projektin toteutuksen. Lisäksi hakemistosta löytyy skripti natiivin Android-paketin kääntämiseksi.

Libs

Libs-hakemisto sisältää iOS-projektin vaatimat kirjastot. Käytännössä tämä tarkoittaa, että esimerkiksi Cocos2d-kirjastot sijaitseva nyt duplikaatteina kahdessa eri sijainnissa: libs-hakemistossa ja Cocos2d-x -paketin juuressa. Mikäli kovalevyn tallennustila on kriittisen vähäinen, voidaan levytilaa säästää muokkaamalla iOS-projekti käyttämään paketin juuressa olevia kirjastoja. Tämä vaatii mm. XCode-projektitiedoston muokkaamista tekstieditorissa.

5.2.3 HelloWorld-projekti

Kääntäminen ja suoritus

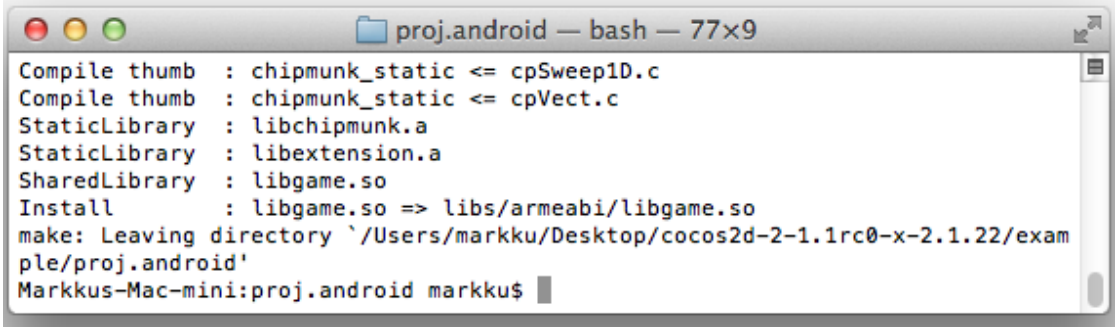
iOS-projekti on kääntövalmis heti tuoreeltaan ja voidaan ajaa iOS-simulaattorissa tai testilaitteessa suoraan XCoden kautta (ks. kuvio 8).

KUVIO 8. HelloWorld-projekti suoritettuna iOS-simulaattorissa

Android-paketin kääntämiseksi Android-projektikansio sisältää skriptitiedoston **build_native.sh**. Skriptitiedoston alkuun tulee ennen suorittamista lisätä tekstieditorilla rivi

export NDK_ROOT = <path/to/Android-NDK>

Käännöksen onnistuessa valmis apk-paketti löytyy Android-projektikansion bin-hakemiston juuresta (ks. kuvio 9). Android-sovellus voidaan nyt suorittaa emulaattorissa tai oikeassa laitteessa käyttäen Eclipseä tai komentorivityökalu ADB:tä.



KUVIO 9. Onnistunut Android-projektin käännösprosessi

Koodin rakenne ja sisältö

Valitusta Cocos2d-x -projektityypistä riippumatta ovat muuntelemattomien Hello-World-projektien perusrakenteet samat muutamia yksityiskohtia lukuunottamatta. Projektin alustariippumattomat luokat käsittävät kaksi välttämätöntä perustaluokkaa: AppDelegaten ja CCLayer-luokasta periytyvän HelloWorldScenen. Eri projektityyppien HelloWorld-projektit sisältävät kuitenkin valitusta tyypistä riippuen omanlaistansa toteutusta.

Oletuksena AppDelegate sisältää kolmen virtuaalisen funktion toteutukset, joita kutsutaan automaattisesti aina tietyissä Cocos2d-x -sovelluksen suoritusyklin vaiheissa. Kun sovelluksen alustakohtainen instanssi on luotu, alustettu ja käynnistetty, kutsutaan universaaleista sovelluksen osista ensimmäisenä funktiota a**pplicationDidFinish-Launching()**. Tässä funktiossa luodaan ja käynnistetään ensimmäiset pakolliset Cocos2d-x -sovelluksen osat, kuten OpenGL-näkymä ja CCScene-luokan instanssi. Lisäksi määritellään oletusarvoinen ruudunpäivitysnopeus ja ruudunpäivitystietojen näyttäminen aktivoidaan.

```
bool AppDelegate::applicationDidFinishLaunching()
```

```
{
    CCDirector *pDirector = CCDirector::sharedDirector();
    pDirector->setOpenGLView(CCEGLView::sharedOpenGLView());

    pDirector->setDisplayStats(true);
    pDirector->setAnimationInterval(1.0 / 60.0f);

    CCScene *pScene = HelloWorld::scene();
    pDirector->runWithScene(pScene);

    return true;
}
```

Funktiota applicationDidEnterBackground() ja applicationWillEnterForeground() kutsutaan sovelluksen siirryttyä taustalle ja juuri ennen sieltä palaamista. Näissä funktioissa pysäytetään sovelluksen päivitys ja mahdolliset taustaäänet ja ääniefektit.

Nimestään huolimatta luokka HelloWorldScene periytyy CCScene-luokan sijaan luokasta CCLayer. Luokka sisältää kuitenkin staattisen funktion **scene()**, jossa luodaan ensimmäinen CCScene-luokan instanssi ja palautetaan sen osoitinmuuttuja. Scenen lisäksi funktiossa luodaan varsinainen HelloWorldScene-luokan instanssi, joka oliohierarkiassa liitetään juuri luodun CCScene-olion jälkeen.

```
CCScene* HelloWorld::scene()
{
    CCScene *scene = CCScene::create();
    HelloWorld *layer = HelloWorld::create();
    scene->addChild(layer);
    return scene;
}
```

Lisäksi HelloWorldScene sisältää funktion **init()**, jossa luokan instanssi ja sen sisältö alustetaan. Viimeinen funktio, **menuCloseCallBack()**, sisältää toteutuksen sovelluksen sulkemiseksi. Funktio on kytketty init()-funktiossa alustettuun käyttöliittymäpainikkeeseen.

5.3 CCDirector

CCDirectorin instanssiin päästään käsiksi staattisen funktion **sharedDirector()** kautta. Olio on saatavilla luokasta riippumatta.

```
void MyClass::pauseGame()
{
    CCDirector::sharedDirector()->pause();
}
```

Tilamuutokset

CCDirector-luokka sisältää seuraavat funktion pelisovelluksen tilojen hallintaan:

- pause()
- resume()
- isPaused()
- stopAnimation()
- startAnimation()
- end()

Funktioilla **pause()** ja **resume()** hallitaan taukotiloja, joissa halutaan säilyttää mm. valikoille tarpeellisten komponenttien kuten valikoiden ja painikkeiden toiminnallisuus. Funktio **isPaused()** kertoo sovelluksen nykytilan. Taukotilassa esimerkiksi animaatioiden ja päivitysfunktioiden toiminta pysäytetään, mutta pääsilmukka jatkaa toimintaansa.

CCDirector-luokan funktioilla **stopAnimation()** ja **startAnimation()** pysäytetään ja käynnistetään koko sovelluksen toiminta. Koska tällöin mm. painikkeet menettävät interaktiivisuutensa, käytetään näitä funktioita lähinnä vain appDelegaten sisällä virtuaalisissa funktioissa, jotka kutsutaan sovelluksen piiloutuessa tai palatessa taustatilasta. Funktio **end()** sammuttaa koko sovelluksen ja vapauttaa sen varaaman keskusmuistin.

Ikkunan koko

Jotta pelisovellukseen voidaan sijoittaa graafisia olentoja, kuten käyttöliittymäelementtejä tai peliolioita, on tarpeellista tietää sovelluksen ikkunan koko. Ikkunan ulottuvuuksien selvittämiseksi voidaan käyttää seuraavia CCDirector-olion funktioita:

- getWinSize()
- getWinSizeInPixels()

- getVisibleSize()

Suositeltavin funktioista on **getWinSize()**, joka palauttaa ikkunan koon Cocos2d-x:n koordinaatistopisteinä. Mikäli Cocos2d-x:n oma sisäänrakennettu moniresoluutiotuki on otettu käyttöön oikein, toimivat funktion palauttamat arvot näytön koosta ja resoluutiosta huolimatta samoissa mittasuhteissa. Funktio **getWinSizeInPixels()** puolestaan palauttaa mitat todellisina näyttöruudun pikseleinä. Funktio **getVisibleSize()** palauttaa ikkunan koosta vain sen näkyvissä olevan osan.

Projektio

Graafisten sovellusten sovelluskehyksenä Cocos2d-x:n olennainen osa-alue on projektio. CCDirectorin funktiolla **setProjection()** voidaan asettaa mm. seuraavat projektion enumeraatioarvot:

- kCCDirectorProjection2D (othogonaalinen)
- kCCDirectorProjection3D (fovy=60, znear=0.5f and zfar=1500)

Arvolla **kCCDirectorProjection2D** tarkoitetaan ortogonaalista projektiota. Ortogonaalisessa projektiossa kolmiulotteinen tila esitetään kaksiulotteisesti, jolloin piirrettävässä kuvassa ei esiinny perspektiiviin kuuluvaa ilmiötä konvergaatiot. Konvergaatiolla tarkoitetaan havainnoitsijan illuusiota samansuuntaisten linjojen lähentymisestä kohti toisiaan niiden loitontuessa horisonttia kohti. Ortogonaalisella projektiolla voidaan mahdollisesti parantaa suorituskykyä tai poistaa kolmiulotteisessa projektiossa havaittavat, mahdollisesti tekstuureissa horisontaalisesti tai vertikaalisesti esiintyvät virheelliset artifaktit.

Cocos2d-x:n projektion oletusarvo on **kCCDirectorProjection3D**, jolloin piirretään aidosti kolmiulotteinen kuva. Oletusarvoisesti näkökartio on asetettu 60 asteeseen. Syvyyden ensimmäinen piirtoraja on 0.5 ja takaraja 1500 koordinaattipistettä.

Scenet

Cocos2d-x -sovelluksen sydämenä sen kautta hoidetaan myös käytössä oleva kulissiolio CCScene. Eri tarkoituksiin sopivia vaihtofunktioita ovat mm.

- runWithScene()
- pushScene()
- replaceScene()

Funktiolla **runWithScene()** käynnistetään ensimmäinen kulissiolio, joka taas voidaan korvata uudella oliolla käyttäen funktiota **replaceScene()**. Mikäli vanha olio halutaan säilyttää ja siihen palata, voidaan käyttää funktiota **pushScene()**.

5.4 Solmuluokat

5.4.1 CCPoint

CCPoint ei varsinaisesti ole niin sanottu solmuluokka, mutta on niiden toiminnan kannalta keskeinen koordinaattiolio. Kaikki sijaintiin liittyvät koordinaatit asetetaan ja palautetaan CCPoint-oliona. Luokka käsittää attribuutteina koordinaatiston arvot **X** ja **Y**.

```
void MyClass::setSamePosition(CCSprite *sprite1, CCSprite *sprite2)
{
      CCPoint position = sprite1->getPosition();
    sprite2->setPosition(position);
}
```

CCPoint-olioita käsitellessä kannattaa muistaa, että Cocos2d-x sisältää runsaasti hyödyllisiä makroja erinäisiin laskutoimituksiin. Nämä ”ccp”-alkuiset makrot suorittavat näppärästi niin peruslaskusuoritukset (ccpAdd, ccpSub), kuin piste- ja ristitulon (ccpDot, ccpCross).

```
CCPoint MyMath::closestPointOnLine(CCPoint A, CCPoint B, CCPoint P,
bool segmentClamp)
{
    CCPoint AP = ccpSub(P, A);
    CCPoint AB = ccpSub(B, A);

    float t = ccpDot(AP, AB) / ccpDot(AB, AB);

    if (segmentClamp)
    {
        if (t < 0.0f) t = 0.0f;
        else if (t > 1.0f) t = 1.0f;
    }

    AB = ccpMult(AB, t);

    return ccpAdd(A, AB);
}
```

5.4.2 CCNode

Luominen ja alustus

CCNodesta ja siitä periytyvistä luokista luodaan instanssi funktiolla **create()**. Tulee kuitenkin huomata, että monilla luokilla kuten CCSprite, on kaikille yhteisen create()-funktion lisäksi omia sen sukuisia funktioita, joilla olio alustetaan käyttäen resursseja kuten tekstuureita. Luomisen jälkeen jokainen solmuluokka liitetään jo olemassa olevan olion lapseksi. Prosessi on jokaiselle solmuluokalle pääpiirteissään sama lukuun ottamatta pieniä luokkakohtaisia yksityiskohtia.

```
void MyClass::createNodeObject()
{
	CCNode *node = CCSprite::create();
	this->addChild(node);
}
```

Oliohierarkia

Mikäli oliota ei lisätä oliohierarkiaan, se hävitetään pääsilmukan loppupuolella. Se voidaan kuitenkin estää kutsumalla luomisen jälkeen funktiota **retain()**, joka erottaa olion Cocos2d-x:n muistinhallinnasta. Olio palautetaan takaisin muistinhallinnan piiriin kutsumalla funktiota **autoRelease()**.

```
void MyClass::createNodeObject()
{
	CCNode *node = CCSprite::create();
	node->retain();   // Remove from AutoReleasePool...
	node->autorelease();    // And it's pooled again!
	this->addChild(node);
}
```

Piirtojärjestys

Solmuoliota lisättäessä oliohierarkiaan voidaan määrittää myös haluttu piirtojärjestys asettamalla **ZOrder**-muuttuja. Muuttuja voidaan asettaa myös halutussa vaiheessa tai muuttaa funktiolla **setZOrder()**. Mikäli haluttu Z-indeksi jätetään kertomatta, on sen arvo oletuksena 0. Mikäli kahdella oliolla on sama indeksiarvo, suoritetaan piirto luontijärjestyksen mukaisesti nuorimmasta vanhimpaan.

```
void MyClass::createNodeObjectWithZIndex(int zIndex)
{
      CCNode *node = CCSprite::create();
      this->addChild(node,zIndex);
}
```

Solmuoliot voidaan poistaa vanhemmastaan ja sitä kautta koko oliohierarkiasta kutsumalla funktiota **removeFromParent()**. Olion omat lapset voidaan puolestaan poistaa funktiolla **removeAllChildren()**. Funktiot eivät kuitenkaan vaikuta olioiden instansseihin, vaan ne poistetaan vasta suorituskierroksen lopulla muistinhallinnassa. Mikäli oliot halutaan poistaa muistista välittömästi, voidaan käyttää funktioiden variantteja **removeFromParentWithCleanup()** tai **removeAllChildrenWithCleanup()**.

5.4.3 CCSprite

Instanssin luominen

CCSprite-olioita luodessa on näppärintä käyttää create()-funktion johdannaisfunktioita:

- create(const char *pszFileName)
- create(const char *pszFileName, const CCRect &rect)
- createWithTexture(CCTexture2D *pTexture)
- createWithTexture(CCTexture2D *pTexture, const CCRect &rect)
- createWithSpriteFrame(CCSpriteFrame *pSpriteFrame)
- createWithSpriteFrameName(const char *pszSpriteFrameName)

Funktioille on yhteistä, että ne vastaanottavat parametrinä tiedostonimen tai valmiin olion, joiden mukaan alustavat CCSprite-olion halutulla kuvalla.

Ankkuripiste

Jokaisella solmuoliolla on niin sanottu ankkuripiste (anchor point), jonka avulla määritellään olion sijainti, kierto ja mittasuhteet. Ankkuripiste asetetaan funktiolla **setAnchorPoint()**. Tulee kuitenkin huomioida, ettei ankkuripisteenä käytetä absoluuttista, vaan suhteellista koordinaattipistettä. Oletusarvoisesti tämä arvo on (0.5f,0.5f), joka tarkoittaa olion ulkomuodollista keskipistettä.

CCSpriteFrame

CCSprite ei itsessään ole piirrettävä tekstuuri, vaan se sisältää sitä varten oman tekstuuriolion. Mikäli CCSprite-olion tekstuuri vaihtuu harvoin, voidaan **CCSpriteFrame**-olio vaihtaa manuaalisesti CCSprite-olion funktiolla **setDisplayFrame()**. Animaatioita ei tällä tavoin kuitenkaan kannata toteuttaa, sillä sitä varten on olemassa omat olionsa.

Batchaus

Mikäli samaa tekstuuria käyttäviä CCSprite-olioita on useampia, voidaan suorituskyvyn nostamiseksi käyttää ns. batchaus-tekniikkaa. Batchausta varten luodaan olio **CCSpriteBatchNode**-luokasta, jota käytetään batchattavien CCSprite-olioiden vanhempana. Jotta CCSprite-olio voidaan lisätä CCSpriteBatchNode-olioon, tulee molempien käyttää samaa tekstuuria. Käytettävän tekstuurin lisäksi create()-funktiossa tulee ilmoittaa myös olioiden maksimimäärä. Ilmoitettu arvo ei ole staattinen vaan se muuttuu dynaamisesti uusien lisättyjen olioiden myötä rajan tullessa vastaan.

```
void MyClass::createBatchedSprites(int spriteCount)
{
	CCSpriteBatchNode *batchNode;
	batchNode = CCSpriteBatchNode::create("star.png", spriteCount);
	this->addChild(batchNode);

	for (int i = 0; i < spriteCount; i++) {
		CCSprite *starSprite = CCSprite::create("star.png");
		batchNode->addChild(starSprite);
	}
}
```

Käytetyn tekstuurin ei tarvitse olla täsmälleen sama, mutta niiden tulee olla samasta kuvatiedostosta. Suositeltavaa onkin käyttää sprite-lakanoita. Näin voidaan useiden erilaisten CCSprite-olioiden piirto suorittaa samalla piirtokäskyllä. Mitään kappalemääräistä rajaa CCSprite-olioille ole, vaan CCSpriteBatchNode-olioon voidaan lisätä rajattomasti.

Batchausta tuleekin käyttää mahdollisimman paljon huomioiden kuitenkin sen piirtojärjestykselliset rajoitukset. Vaikkakin batchatut CCSprite-oliot noudattavat keskenään piirtohierarkiaa, samassa CCSpriteBatchNode-oliossa olevat CCSprite-oliot piirretään samalle kuvatasolle. Tästä syystä CCSpriteBatchNode-olioon kuulumattomat CCSprite-

oliot voidaan piirtää vain joko batchattujen olioiden alle tai päälle, ei niiden väliin.

5.5 Solmuolioiden toiminnot ja tilapäivitys

5.5.1 Ajoitukset ja update()

Jokaisen solmuolion ominaisuuksiin kuuluvat tilapäivitykset. Tilapäivitykset suoritetaan **ajoituksilla** (schedule), jolla tarkoitetaan funktioiden suorittamista tietyllä hetkellä. Ajoitukset voidaan kohdistaa joko omaan funktioon tai CCNoden omaan funktioon **update(float d)**.

Update(float dt)

Mikäli olion päivitys tapahtuu jokaisella ruudulla, suositellaan käytettäväksi valmista funktiota update(float dt). Funktioon lisätään omaa toiminnallisuutta syrjäyttämälle se. Oletuksena funktio ei sisällä minkäänlaista toiminnallisuutta. Funktion suorittaminen käynnistetään solmuolion funktiolla **scheduleUpdate()** ja sammutetaan vastaavasti funktiolla **unscheduleUpdate()**. Funktion parametri **dt** on edellisen ruudun vaatima päivitysaika millisekunneissa eli delta-aika.

```
bool HelloWorld::init()
{
    //    Some default initialization...
    //    Create some own stuff...

    this->scheduleUpdate();

    return true;
}

void HelloWorld::update(float dt)
{
    //      This is now called every frame!
}
```

Olioiden update()-funktiot suoritetaan olioiden luontijärjestyksen mukaan. Oliot voidaan kuitenkin järjestellä haluttuun päivitysjärjestykseen käyttämällä ajoitusfunktiota **scheduleUpdateWithPriority(int priority)**. Mitä pienempi prioriteetti-indeksin arvo on, sitä ensisijaisemmaksi se sijoittuu päivityshierarkiassa.

Kustomoidut ajoitukset

Mikäli halutaan yksittäisiä funktioita, joiden toiminnallisuus ei vaadi delta-aikaa, ja joiden käynnistäminen ja sammuttaminen on satunnaista, voidaan käyttää kustomoitua ajoitusta funktiolla **schedule()**. Suoritettava funktio valitaan **schedule_selector()**-makrolla.

```
bool MyClass::init()
{
    //    Some default initialization...
    //    Create some own stuff...

      this->schedule(schedule_selector(HelloWorld::customizedUpdate),
                                                    1.3f,
                                                    5,
                                                    2.0f);

    return true;
}

void MyClass::customizedUpdate()
{
      //    This is now called 5 times after 2.0f second delay
      //    Interval is assigned to 1.3 seconds
}
```

Mikäli haluttu funktio halutaan ajoittaa suoritettavaksi kerran, voidaan käyttää funktiota **scheduleOnce()**. Funktio vaatii parametreinä suoritettavan funktion ja käynnistysviiveen sekunneissa. Funktioita ajoittaessaan tulee huomata, että samalle funktiolle voi olla vain yksi voimassa oleva ajoitus. Mikäli funktiolle määrätään päällekkäinen ajoitus, vanha ohitetaan.

5.5.2 Toimintoluokat

Toimintoja voidaan suorittaa jokaisen solmuluokan olioille. Toiminnosta luodaan oma olionsa, joka linkitetään haluttuun solmuolioon **runAction()**-funktiolla. Jokaista solmuoliota kohti tulee luoda oma toiminto-olionsa.

```
void MyClass::moveSprites(CCSprite *sprite1, CCSprite *sprite2)
{
      CCMoveTo *moveTo = CCMoveTo::create(2.0f, CCPoint(100,100));
      sprite1->runAction(moveTo);
      sprite2->runAction(moveTo); //      Only this sprite will move
}
```

Luonnossa esiintyvä tilamuutos kuten liike on vain harvoissa tapauksissa tasaista. Esimerkiksi kappaleen liikkeelle lähteminen ja pysähtyminen vaatii aina jonkin asteista kiihtyvyyttä, joten lineaarinen liike vaikuttaa pelaajan näkökulmasta epäluonnolliselta tai jopa epämiellyttävältä. Tästä syystä on suositeltavaa muokata toimintoluokkien käyttäytymistä **CCEaseRateAction**-luokilla. Toiminto-olion osoitinmuuttujan lisäksi sulavoittamisoliota luodessa annetaan suhdeluku joka määrittelee kiihtyvyyden jyrkkyyden.

```
void MyClass::moveSprite(CCSprite *sprite)
{
	CCMoveTo *moveTo = CCMoveTo::create(2.0f, CCPoint(100,100));
	CCEaseInOut *easedMoveTo = CCEaseInOut::create(moveTo, 2);
	sprite->runAction(easedMoveTo);
}
```

Toimintojen niputtaminen

Toimintoja voidaan yhdistellä toisiinsa käyttämällä luokkia **CCSequence** ja **CCSpawn**. CCSequence luo toimintokokonaisuuden jossa sen jokainen toiminto-olio suoritetaan peräkkäin. CCSpawn suorittaa toiminnot yhtäaikaisesti. Oliota luodessa tulee muistaa osoitinmuuttujien jälkeen viitata NULL-parametrilla. Lisäksi tulee huomioida, että CCSpawnilla ei tule niputtaa saman luokan olioita.

```
void MyClass::moveThenRotateSprite(CCSprite *sprite)
{
	CCMoveTo *moveTo = CCMoveTo::create(2.0f, CCPoint(250,150));
	CCRotateBy *rotate = CCRotateBy::create(2.0, 45);
	CCFiniteTimeAction *sqnc = CCSequence::create(moveTo, rotate,
	NULL);
	sprite->runAction(sqnc);
}
```

Mikäli toiminto halutaan suorittaa useamman kerran, voidaan käyttää **CCRepeat**-luokan oliota.

```
void MyClass::rotateSprite(CCSprite *sprite, int repCount)
{
	CCRotateBy *rotate = CCRotateBy::create(0.50f, 90);
	CCFiniteTimeAction *rptRot = CCRepeat::create(rotate,
repCount);
	sprite->runAction(rptRot);
}
```

Mikäli toiminnon halutaan suoriutuvan ikuisesti, voidaan käyttää vastaavaa oliota **CCRepeatForever**. Mikäli halutaan toistaa ikuisesti niputettuja toimintoja kuten sekvenssi, tulee se muuttaa **CCActionInterval**-tyyppiseksi.

```
void MyClass::rotateAndMoveSpriteForever(CCSprite *sprite)
{
	CCMoveTo *moveTo = CCMoveTo::create(2.0f, CCPoint(250,150));
	CCRotateBy *rotate = CCRotateBy::create(2.0, 45);
	CCFiniteTimeAction *seq = CCSpawn::create(moveTo,rotate,NULL);
	CCFiniteTimeAction *rptForever;
	rptForever = CCRepeatForever::create((CCActionInterval*)seq);
	testSprite->runAction(rptForever);
}
```

5.5.3 Animaatiot

Animaatio voidaan luoda manuaalisesti erillisistä kuvatiedostoista. Luodaan **CCAnimation**-olio, jolle animaation yksittäiset ruudut asetellaan yksitellen funktiolla **addSpriteFrameWithFileName()**. Jotta funktio voidaan suorittaa kätevästi silmukassa, kannattaa nimetä tiedostot yhdenmukaisesta numerointia noudattaen. Ruutujen alustuksen jälkeen animaatiolle asetetaan yksittäisen ruudun pyöritysaika. Mikäli animaation on tarkoitus palata alkuperäiseen ruutuun, asetetaan ehto funktiolla **setRestoreOriginalFrame()**. Jotta animaatio voidaan ajaa tyypillisellä funktiolla **runAction()**, muutetaan se CCAction-luokasta periytyväksi **CCAnimate**-olioksi.

```
void MyClass::animateSprite(CCSprite *sprite)
{
	CCAnimation *rawAnimation = CCAnimation::create();

	for (int i = 1; i < 15; i++)
	{
		char imageFileName[128] = {0};
		sprintf(imageFileName, "frame%02d.png", i);
		rawAnimation->addSpriteFrameWithFileName(imageFileName);
	}

	rawAnimation->setDelayPerUnit(0.15f);
	rawAnimation->setRestoreOriginalFrame(true);

	CCAnimate *animated = CCAnimate::create(rawAnimation);
	sprite->runAction(animated);
}
```

Animaation ruudut voidaan ladata myös esimerkiksi sprite-lakanasta. Animaatiosta voidaan kolmannen osapuolen editoreilla luoda myös oma plist-tiedostonsa, johon on määritelty mm. käytettävä sprite-lakana, animaation ruudut ja niiden päivitysnopeus.

```
void MyClass::animateSpriteWithPlist(CCSprite *sprite)
{
	CCAnimationCache *cache;
	cache = CCAnimationCache::sharedAnimationCache();
	cache->addAnimationsWithFile("animations.plist");
	CCAnimation *animation = cache->animationByName("myAnimation");
	CCAnimate *animate = CCAnimate::create(animation);
	sprite->runAction(CCRepeatForever::create(animate));
}
```

5.6 Kontrollit

5.6.1 Kosketusten lukeminen

Kosketustapahtumia käytettäessä tulee valita, halutaanko käyttää kohdistettua kosketusmetodia vai monikosketusta. Mikäli yksinkertainen kosketustapahtuma riittää, on suositeltavaa käyttää kohdistettua kosketustapahtumaa sen yksinkertaisemman rakenteen vuoksi: virtuaalinen funktio **ccTouchBegan()** palauttaa vain yhden kosketustapahtuman. Funktiota **ccTouchesBegan()** käytettäessä on kosketustapahtumien erottelu käyttäjän vastuulla.

Kohdistettu kosketustapahtuma

Aktivoidakseen kohdistetun kosketuksen, tulee kosketuksista vastaavalle oliolle **CCTouchDispatcher** kertoa kosketuksen kohde. Samalla kerrotaan kohteen ensisijaisuusindeksi kosketusten suhteen. Lisäksi määritellään nielaistaanko kosketus. Kun kosketus nielaistaan, se ei vaikuta muihin samassa pisteessä oleviin kohteisiin, olivat ne kohdistettuja tai eivät.

```
bool HelloWorld::init()
{
    //    Some default initialization...

    CCTouchDispatcher *td;
      td = CCDirector::sharedDirector()->getTouchDispatcher();
      td->addTargetedDelegate(this, 0, true);

    return true;
}
```

Kosketuksia luetaan siihen tarkoitetuilla virtuaalisilla funktiolla **ccTouchBegan()**, **ccTouchMoved()** ja **ccTouchEnded()**. Funktiohin syötetään osoitinmuuttujat itse kosketustapahtuman olioon **CCEven** sekä kosketuskoordinaatteihin **CCTouch**. Jotta koordinaatteja voi käyttää, tulee ne muuttaa näyttökoordinaateiksi ja sen OpenGL:n käyttämään koordinaatistoon sopivaksi.

```
bool HelloWorld::ccTouchBegan(CCTouch *pTouch, CCEvent *pEvent)
{
      CCPoint temp = pTouch->getLocationInView();
      temp = CCDirector::sharedDirector()->convertToGL(temp);
```

```
}
```

Monikosketustapahtumat

Lisäksi iOS-kohtaisessa projektissa tulee muistaa kytkeä päälle alustakohtainen monikosketustuki lisäämällä tarvittava koodirivi AppController.mm -tiedostoon. Kyseessä on oletusarvoinen iOS-sovelluksen luokka. Android-sovelluksissa monikosketus on päällä oletusarvoisesti.

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    // Initialization of window and OpenGl-view __glView

    [__glView setMultipleTouchEnabled:YES];

    // More default initialization stuff...

    return YES;
}
```

Cocos2d-x -sovelluksessa monikosketuksen rekisteröinti aktivoidaan CCLayer-luokan funktiolla **setTouchEnabled()**.

```
bool HelloWorld::init()
{
      //    Some default initialization...

      this->setTouchEnabled(true);
      this->setAccelerometerEnabled(true);


      return true;
}
```

Kohdistetusta kosketuksesta poiketen kosketustapahtumat luotaan funktioilla **ccTouchesBegan()**, **ccTouchesMoved()** ja **ccTouchesEnded()**. Funktiot eroavat siinä, että yhden koskituskoordinaatin sijaan palautetaan taulukko useampia.

```
void HelloWorld::ccTouchesBegan(CCSet *pTouches, CCEvent *pEvent)
{
    CCSetIterator it;
    CCTouch* touch;

    for( it = touches->begin(); it != touches->end(); it++)
    {
        touch = (CCTouch*)(*it);

        if(!touch)
            break;

        CCPoint temp = touch->getLocationInView();
        temp = CCDirector::sharedDirector()->convertToGL(temp);

    }
}
```

5.6.2 Kiihtyvyysanturi

Kuten kosketustapahtumat, aktivoidaan kiihtyvyysanturin lukeminen siihen tarkoitetulla CCLayer-olion funktiolla.

```
bool HelloWorld::init()
{
	//	Some default initialization...
	this->setAccelerometerEnabled(true);
	return true;
}
```

Lukemat luetaan syrjäyttämällä virtuaalinen funktio **didAccelerate()**, joka saa parametreinaan **CCAcceleration**-olion osoitinmuuttujan, josta arvot voidaan lukea radiaaneina jokaiselta akselilta.

```
void HelloWorld::didAccelerate(CCAcceleration *pAccelerationValue)
{
	accX = pAccelerationValue->x;
	accY = pAccelerationValue->y;
	accZ = pAccelerationValue->z;
}
```

Kiihtyvyysanturin herkkyydestä johtuen lukuarvot ovat epätasaisia. Mikäli arvoa käytetään peliolennon kiihdyttämiseen, voidaan arvoa käyttää sellaisenaan. Mutta mikäli peliolennon tila, oli kyseessä sitten rotaatio tai paikka, määritellään suoraan kiihtyvyysanturin antamasta arvosta, on suositeltavaa tasoittaa arvoa esimerkiksi laskemalla viimeisimmistä lukuarvoista keskiarvo. Esimerkki soveltuvasta funktiosta on esitetty luvussa 6.2.3.

5.7 Käyttöliittymäelementit

5.7.1 Tekstitarrat

CCLabelTTF

Fonttiin perustuva tekstitarra CCLabelTTF luodaan samoja periaatteita noudattaen kuin mikä tahansa muukin solmuolio. Merkkijonon päivittäminen suoritetaan funktiolla **setString()**.

```
bool HelloWorld::createFontLabel()
{
```

```
	CCLabelTTF *labelTTF;
	labelTTF= CCLabelTTF::create("Hello, fnt-label!", "Calibri",
14);
	this->addChild(labelTTF);
	labelTTF->setString("Updated text!");
}
```

CCLabelBMFont

CCLabelBMFontin tarvitsemat kuvatiedosto ja fnt-tiedosto on helppo luoda TTF-fonteista kolmannen osapuolen työkaluilla kuten Hiero tai BMGlyph. Myös fontin käyttäminen on varsin yksinkertaista.

```
bool HelloWorld::createBMLabel()
{
	CCLabelBMFont::create("HelloWorld!", "font.fnt");
}
```

Tulee kuitenkin humioida, että käytetty fontti on kuvaformaatissa, eikä sen skaalaaminen suuremmaksi onnistu ilman huomattavaa kuvanlaadun heikkenemistä. Mikäli pelissä käytetään runsaasti saman fontin eri käyttökokoja, kannattaa tehdä tarvittavat tiedostot suurimman fontin mukaisiksi ja skaalata niistä luotava tekstitarra oikean kokoiseksi. Skaalaaminen suuntaan tai toiseen vaikuttaa kuitenkin hieman kuvanlaatuun, joten mikäli kokovariaatioita ei ole paljon, on järkevää luoda samasta fontista jokaiselle käytettävälle fonttikoolle omat lähdetiedostonsa.

5.7.2 Valikot

Valikkoa varten tarvitaan isäntäolio **CCMenu**. Sen luominen ei poikkea tavanomaisen solmuolion luomisesta.

```
void HelloWorld::createMenuObject()
{
    CCMenu *menu = CCMenu::create();
    menu->setPosition(CCPoint(wSize.width*0.5f, wSize.height*0.5f));
    this->addChild(menu);
}
```

Valikon painikkeiden luominen on varsin yksinkertaista sekin. Tulee kuitenkin huomata, että mikäli käytetään painikeolioita jotka käyttävät symbolinaan CCSprite- tai CCLabel-olioita, ei symboliolioita lisätä oliohierarkiaan manuaalisesti vaan **CCMenuItem**-olion **create()**-funktiossa. Valikkopainikkeen sijainti voidaan määritellä manuaalisesti, jolloin sijoitus tapahtuu normaalien solmuolioiden sijoittelukäytäntöjen mukaisesti.

CCMenu-luokka sisältää kuitenkin sijoittelun automaattisesti tekeviä funktioita kuten **alignItemsVertically()** tai **alignItemsHorizontally()**. Nämä funktiot kumoavat manuaaliset positiomäärittelyt.

```
void HelloWorld::createMenuButton(CCMenu *menu)
{
      CCSprite *btnSprite = CCSprite::create("btnImg.png");

      CCMenuItemSprite *menuBtn;
      menuBtn = CCMenuItemSprite::create(btnSprite, // Default
                                                    btnSprite, //
On-Touch
                                                    this,

      menu_selector(HelloWorld::doStuff));
    menu->addChild(menuBtn);
    menu->alignItemsVertically();
}
```

5.8 Näyttö

5.8.1 Orientaatio

Koska näytön orientaatio on alustakohtainen, määritellään se järjestelmäkohtaisesti joka alustalle erikseen.

Android

Android-sovelluksessa käytettävä näyttöorientaatio määritellään tiedostossa Android-Manifest.xml. Activity-tagin sisältä löytyy ominaisuus android:screenOrientation, jolle voidaan antaa seuraavat esimerkiksi arvot:

- unspesified
- landscape
- portrait
- user
- behind

Oletusarvolla **unspecified** järjestelmä päättää sisällön mukaisesti mitä orientaatiota käytetään. Lopputulos voi vaihdella laitevalmistajien välillä, sillä yhtenäistä käytäntöä

ei ole. Arvot **landscape** ja **portrait** ovat kiinteät. Niillä määritellään käytettäväksi asennoksi joko puhelimen pysty- tai vaakaorientaatio. **User** määrittää orientaatioksi käyttäjän suosiman orientaation ja **behind** saman kuin sovelluksen alla olevan toisen sovelluksen käyttämän orientaation.

iOS

iOS-sovelluksessa näytön orientaatio määritellään RootViewController-luokasta käsin tiedostossa RootViewController.mm. Versiota iOS 6 vanhemmissa käyttöjärjestelmäversioissa käytetään metodia **shouldAutorotateToInterfaceOrientation**.

```
// Override to allow orientations other than the default portrait
orientation.
// This method is deprecated on ios6
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
    return UIInterfaceOrientationIsLandscape( interfaceOrientation );
//    return
UIInterfaceOrientationIsPortrait( interfaceOrientation );

}
```

Versiossa 6 tulee käyttää metodeja **supportedInterfaceOrientations()** ja **shouldAutorotate()**.

```
// For ios6, use supportedInterfaceOrientations & shouldAutorotate
instead
- (NSUInteger) supportedInterfaceOrientations{
#ifdef __IPHONE_6_0
    return UIInterfaceOrientationMaskLandscape;
#endif
}

- (BOOL) shouldAutorotate {
    return YES;
}
```

Oletusarvoisesti Cocos2d-x -projektista löytyvät molemmat toteutukset yhteensopivuuksien varmentamiseksi.

5.8.2 Moniresoluutiotuki

Moniresoluutiotuki perustuu eri näyttökokoluokille tarkoitettuihin kuvaresurssikategorioihin. Käytettävät resurssikategoriat ja moniresoluutiotuen koko implementointi tapahtuu rajapintaluokassa AppDelegate. Olennainen osa moniresoluutiotukea on ns. suunnitteluresoluutio, jonka mukaan Cocos2d-x -pelisovelluksen oliosijoittelu ruudul-

la toteutetaan. Näin ollen ei tarvita jokaiselle näyttökokoluokalle omia arvoja esimerkiksi pelihahmojen sijoitteluun, vaan voidaan käyttää universaaleja koordinaatistoarvoja. Kaikki käytettävät positioarvot skaalataan automaattisesti CCDirector-oliolle asetetun **contentScaleFactor**-muuttujan mukaisesti.

Eri näyttökokoluokille tarkoitetut resurssit sijoitetaan Resources-hakemistoon omiin alikansioihinsa. Jotta sovellus osaa hakea oikean kokoluokan resurssit, asetetaan sovellukselle oletusarvoiset hakupolut näytön resoluution mukaisesti. Mikäli sovellus ei löydä hakemaansa resurssia tästä ensisijaisesta hakemistosta, etsii se seuraavaksi kyseistä resurssia suoraan Resources-hakemiston juuresta.

```
bool AppDelegate::applicationDidFinishLaunching()
{
    static CCSize sdRes = CCSizeMake(480, 320);
    static CCSize mdRes = CCSizeMake(1024, 768);
    static CCSize hdRes = CCSizeMake(2048, 1536);
    static CCSize desRes = CCSizeMake(480, 320);

    // Initialize director and OpenGL view - Default stuff
    CCDirector* pDirector = CCDirector::sharedDirector();
    CCEGLView* pEGLView = CCEGLView::sharedOpenGLView();
    pDirector->setOpenGLView(CCEGLView::sharedOpenGLView());

    pEGLView->setDesignResolutionSize(desRes.width,
                                      desRes.height,
                                      kResolutionShowAll);

    CCSize frameSize = pEGLView->getFrameSize();

    if(frameSize.height > mdRes.height)
    {
        this->setSearchPath(ccs("hd"));
        pDirector->setContentScaleFactor(hdRes.height/desRes.height);
    }
    else if (frameSize.height > sdRes.height)
    {
        this->setSearchPath(ccs("md"));
        pDirector->setContentScaleFactor(mdRes.height/desRes.height);
    }
    else
    {
        this->setSearchPath(ccs("sd"));
        pDirector->setContentScaleFactor(sdRes.height/desRes.height);
    }

      // Some default initialization - scene and stuff
      //...

    return true;
}
```

Esimerkkitoteutus sisältää myös oman AppDelegateen kuulumattoman funktion hakupolkujen asettamiseen. Funktiolle syötetään hakemiston polku suhteellisena Resources-hakemiston näkökulmasta **ccs()**-makrolla. Makro muuttaa merkkijonon

CCString-olioksi. Cocos2d-olioiden mukaisesti CCString on ns. autorelease-olio, eikä sitä tarvitse poistaa käsin.

```
// This function is not default
void AppDelegate::setSearchPath(cocos2d::CCString *str)
{
    std::vector<std::string> searchPaths;
    CCFileUtils* pFileUtils = CCFileUtils::sharedFileUtils();
    searchPaths.push_back(str->getCString());
    pFileUtils->setSearchPaths(searchPaths);
}
```

5.9 Äänet

Cocos2d-x:n äänimoottori **SimpleAudioEngine** on singleton-olio ja siihen pääsee käsiksi CocosDenshion-nimiavaruuden kautta.

```
CocosDenshion::SimpleAudioEngine::sharedEngine();
```

Ääniresurssit kannattaa esiladata, sillä niiden lataaminen puhelimen muistikortilta on hidasta.

```
void HelloWorld::preloadSoundResources()
{
      SimpleAudioEngine *audioEngine;
      audioEngine = CocosDenshion::SimpleAudioEngine::sharedEngine();
      audioEngine->preloadBackgroundMusic("bgM.mp3");
      audioEngine->preloadEffect("effect.wav");
}
```

Äänimoottorin itsensä käyttäminen on yksinkertaisen suoraviivaista.

```
void HelloWorld::playBackgroundMusic()
{
      SimpleAudioEngine *audioEngine;
      audioEngine = CocosDenshion::SimpleAudioEngine::sharedEngine();
      audioEngine->setBackgroundMusicVolume(0.75f);
      audioEngine->playBackgroundMusic("bgM.mp3", true);
}

int HelloWorld::playSoundEffect()
{
      SimpleAudioEngine *audioEngine;
      audioEngine = CocosDenshion::SimpleAudioEngine::sharedEngine();
      audioEngine->setEffectsVolume(0.75f);

      unsigned int effectId;
      effectId = audioEngine->playEffect("effect.wav", false);

      return effectId;
}

void HelloWorld::pauseAndResumeAudio()
{
      SimpleAudioEngine::sharedEngine()->pauseAllEffects();
```

```
        SimpleAudioEngine::sharedEngine()->pauseBackgroundMusic();

        SimpleAudioEngine::sharedEngine()->resumeAllEffects();
        SimpleAudioEngine::sharedEngine()->resumeBackgroundMusic();
}

void HelloWorld::pauseAndResumeSingleEffect(int effectId)
{
        SimpleAudioEngine::sharedEngine()->pauseEffect(effectId);
        SimpleAudioEngine::sharedEngine()->resumeEffect(effectId);
}

void HelloWorld::killEverything()
{
        SimpleAudioEngine::sharedEngine()->stopAllEffects();
        SimpleAudioEngine::sharedEngine()->stopBackgroundMusic(true);
        SimpleAudioEngine::sharedEngine()->unloadEffect("effect.wav");
        SimpleAudioEngine::sharedEngine()->end();
}
```

Jotta äänet pysäytetään sovelluksen siirtyessä taustalle, tulee asianmukaiset funktiot kuten pauseAllEffects() kutsua tilanvahdoksiin liittyvissä virtuaalisten funktioiden toteutuksissa.

6 KÄYTÄNNÖN SOVELLUS: WATERLY

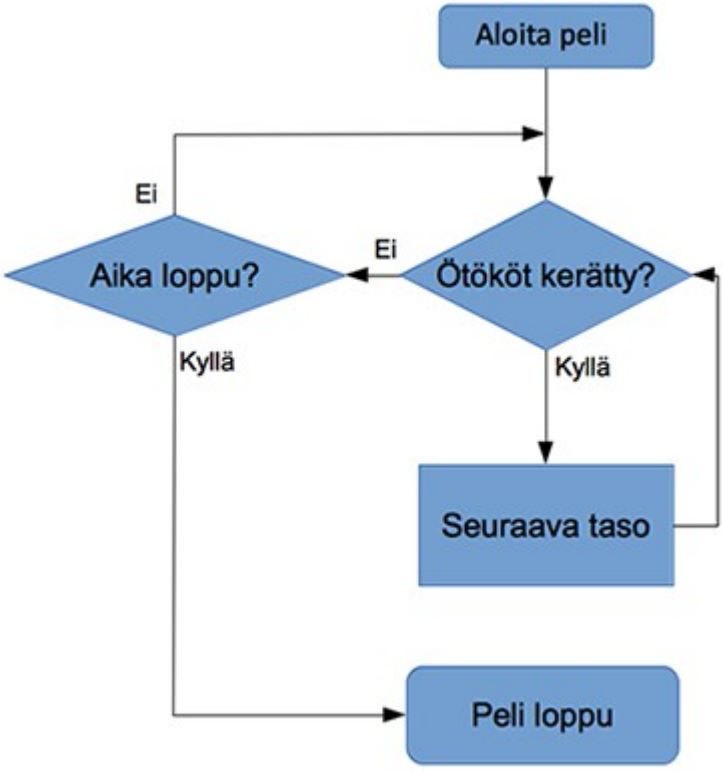
6.1 Waterly pelisovelluksena

6.1.1 Gameplay

Waterly on yksinkertainen toimintapeli, jossa pelaaja ohjaa eräänlaista lummeolentoa puhelinta kallistelemalla (ks. kuvio 10). Pelaajan tavoite on noukkia hyönteisiä lummeolion kieleltä käyttäen. Kieltä ohjataan näyttöruutua napauttamalla, jolloin kieli suoristuu niin kauan kuin pelaaja painaa näyttöruutua tai kieli tavoittaa maksimikorkeutensa.

KUVIO 10. Waterly-toimintapeli pelitilassaan

Jokaisen tason alussa pelaajalla on 10 sekuntia peliaikaa. Jokaisesta ystävällisestä hyönteisestä, kuten kärpänen tai perhonen, pelaaja saa 2 sekuntia lisäaikaa. Ampiaisesta pelaaja saa 3 sekuntia sakkoa ja sekunnit vähennetään peliajasta. Mikäli pelaaja kerää tavoitemäärän hyönteisiä peliajan sisällä, saavuttaa pelaaja seuraavan pelitason. Seuraavalle tasolle siirryttäessä kerättävien hyönteisten tavoitemäärä suurenee. Samoin kasvaa myös hyönteisten, erityisesti ampiaisten, esiintyvyys. Peliajan loputtua pelaaja häviää (ks. kuvio 11).



KUVIO 11. Waterly-toimintapelin karkea pelilogiikka

6.1.2 Valikot

Aloitusvalikko

Kun Waterly käynnistetään, ensimmäinen näkymä on aloitusvalikko (ks. kuvio 12). Aloitusvalikosta pelaaja voi käynnistää pelin, siirtyä hyönteisten esittelyvalikkoon tai sammuttaa pelin. Valikon jokainen painike on animoitu skaalausefektillä valikkotilan elävöittämiseksi. Taustalla on kaikissa valikoissa ja pelitilassa esiintyvä utuista suometsikköä esittävä taustakuva.

KUVIO 12. Waterly-toimintapelin aloitusvalikko

Hyönteisten esittely -valikko

Tässä valikossa esitellään kaikki pelissä esiintyvät hyönteiset. Hyönteisistä kerrotaan leikkisällä kielellä jokaisen hyönteisen käyttäytymisestä ja syötävyydestä (ks. kuvio 13). Valikkoa selataan sormieleellä joka muistuttaa kirjan sivun lipaisevaa kääntämistä yhdellä sormella. Vaihtoehtoisesti valikon selauksen voi suorittaa yläreunan nuolipainikkeilla, jotka samalla ilmoittavat käyttäjän sijainnin valikkopuussa. Vasemmassa alanurkassa on painike alkuvalikkoon palamiseksi.

KUVIO 13. Waterly-toimintapelin valikko hyönteisten esittelyyn

Taukovalikko ja peli päättynyt -valikko

Taukovalikko noudattaa hillityn tyylikästä ja yksinkertaista linjaa. Läpinäkyvältä pohjalta löytyy valikon otsikko ja yksinkertaiset painikkeet, joiden symboliset merkitykset ovat tuttuja pelaajille (ks. kuvio 14). Painikkeet sisällyttävät valikkoon toiminnot peliin palaamisesta, uuden aloittamisesta ja aloitusvalikkoon siirtymisestä.

KUVIO 14. Waterly-toimintapelin taukotilavalikko

Pelin päättymisestä ilmoittava valikko noudattaa samaa yksinkertaista linjaa (ks. kuvio 15).

KUVIO 15. Waterly-toimintapelin ilmoitus pelin päättymisestä

6.2 Waterly teknisesti

6.2.1 Pelitilan maailma

Syvyysvaikutelman luomiseksi Waterlyn pelitilan maailma muodostuu neljästä kerroksesta. Alimmainen kerros on kokonaisena kuvatekstuurina toteutettu taustakuva (ks. kuvio 16).

KUVIO 16. Waterly-toimintapelin taustakuva pelimaailman alimmaisena kerroksena

Taustakuvan erikoisuus on, että se sisältää itse taustamiljöön lisäksi vedenpinnan heijastumakuvan. Koska perspektiivisääntöjen mukaan katsojan havainnoima pelikuva liikkumattomasta kohteesta pysyy katsojan näkökulmasta liikkumattomana vaikka hei-

jastuspinta liikkuisi, olisi turhaa lisätä heijastumakuva erillisenä kuvatekstuurina. Näin säästetään sekä käytetyssä grafiikkasuorittimen muistissa, piirtonopeudessa että lopullisen pelitiedoston asennuskoossa.

Toiseksi alimmainen kerros on veden pintakerros. Pintakerros on toteutettu näytön laidasta laitaan yltävällä CCSprite-oliolla. Olion tekstuuri on läpinäkyvä, jotta ensimmäisen kerroksen taustakuvassa oleva taustamiljöön heijastuskuva näkyisi (ks. kuvio 17).

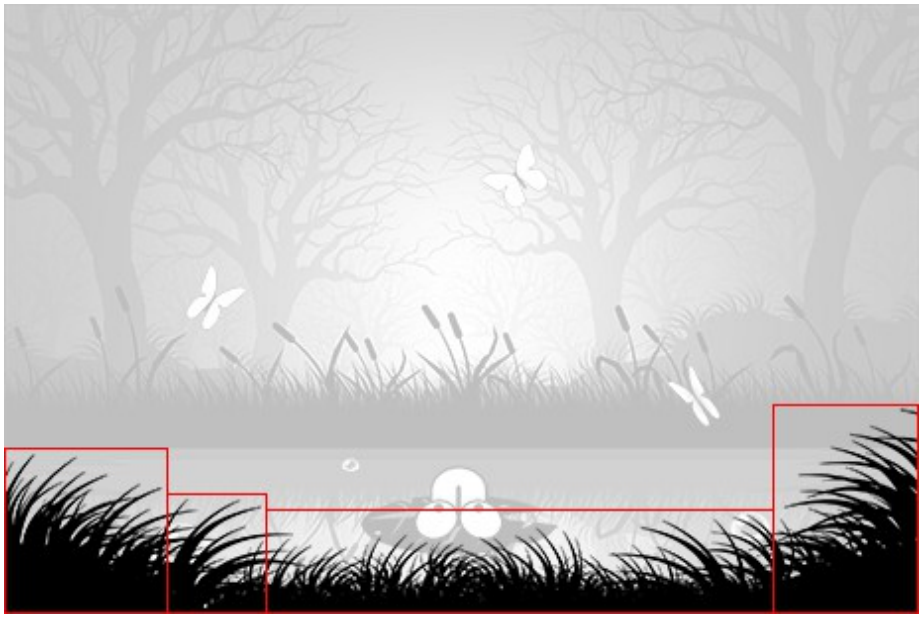
KUVIO 17. Waterly-toimintapelin pelimaailman toiseksi alimmainen kerros

Kerrokseen luodaan illuusio veden aaltoilusta skaalaamalla kuvatekstuuria Y-akselin suuntaisesti sinikäyräliikkeellä. Kerrokseen on oliohierarkkisesti lisätty myös kupla-olioita vesiolion lapsiksi aaltoliikkeen mukailemiseksi. Kuplat syntyvät näkymättömän pieninä ja kasvettuaan täyteen mittaansa poksahtavat rikki.

Kolmannes pelimaailman kerros on peliolioiden oma kuvataso. Peliolioita ovat kaikki hyönteiset sekä itse pelihahmo (ks. kuvio 18).

KUVIO 18. Waterly-toimintapelin pelioliokerros

Viimeinen kuvakerros on etualan ruohikko. Parhaimman suorituskyvyn saavuttamiseksi ruohikko on toteutettu neljällä CCSprite-oliolla. Mikäli ruohikko olisi yhtenäinen kuvatekstuuri, käsiteltäisiin yhdestä CCSprite-olion tekstuurista 2-3 -kertainen määrä näkymättömiä pikseleitä (ks. kuvio 19).



KUVIO 19. Waterly-toimintapelin päällimmäinen, etualakerros ja sen rakenne

6.2.2 Lumme - pelihahmo

Pelihahmo voitaisiin toteuttaa ruutuanimoidulla CCSprite-olennolla. Perinteisen ruutuanimaation ongelma kuitenkin on sen staattisuus. Koska pelihahmo on pelin interaktiivisin olio, ja kaiken lisäksi elävä sellainen, jouduttaisiin toteuttamaan runsaasti eri tilanteisiin kuuluvia animaatioita. Tämä tarkoittaisi sekä enemmän työtä että enemmän rasitettua muistikapasiteettia sekä grafiikkasuorittimen muistissa että puhelimen massamuistissa. Lisäksi interaktiivisissa sovelluksissa esiintyy huomattava määrä erilaisten tilanteiden ja olion toimintojen yhdistelmiä, mikä vaikeuttaa huomattavasti ennakointia ja toteutettavien animaatioiden valintaa ja suunnittelua.

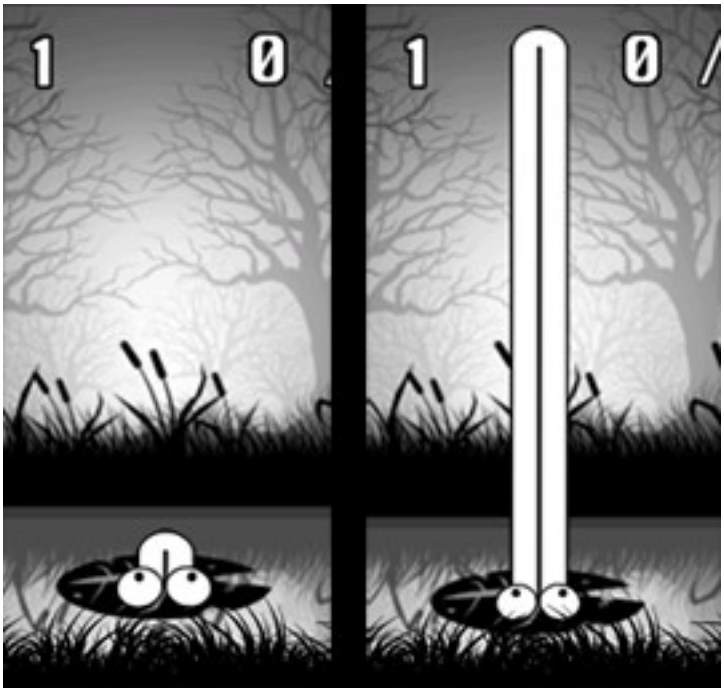
Waterlyn pelihahmo perustuu erillisiin komponentteihin. Pelihahmo koostuu neljästä omasta kerroksestaan, jotka animoidaan erillisillä ominaisanimaatioillaan (ks. kuvio 20). Näin ne toimivat toisistaan riippumatta ja luovat interaktiivisissa tilanteissa useiden animaatioiden kokonaisuuksia elävöittäen pelihahmoa esimerkiksi silmien satunnaisella räpäyttämisellä. Kaikki animaatiot perustuvat ohjelmalliseen CCSprite-olion kuvatekstuurin käsittelyyn jotta ne voisivat reagoida pelaajasta riippuviin, toisin sanoen ennakoimattomiin tapahtumiin reaaliajassa ja animaatiollisesti katsottuna mahdollisimman sujuvasti.

KUVIO 20. Waterly-toimintapelin lummehahmon komponentit

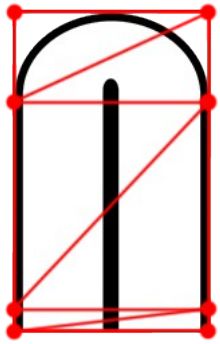
Kustomoitu OpenGL-piirto

Kielen dynaaminen toiminnallisuus asettaa omat vaatimuksensa sen toteuttamiselle

(ks. kuvio 21). Koska kielenkärjen muodon on pysyttävä eheänä puolipallona kielen venyessä, ei kielen toteuttamiseen voida käyttää tavanomaista kahden polygonin tekstuuriin perustuvaa CCSprite-oliota. Lisävaatimukseksi voitaisiin myös lisätä kielen ohentuminen sen pidentyessä niin, että efekti on voimakkaampi kielen keskiosissa.



KUVIO 21. Waterly-toimintapelin pelihahmon kielen toiminta



KUVIO 22. Pelihahmon kielen polygonirakenne

Kieltä varten on siis luotava oma piirrettävä olio. **CTongue** on tätä varten luotu, solmuluokka CCNode-luokasta periytyvä erityisluokka. Vaikka CCNode-olio on olemukseltaan näkymätön, omaa se kaikki valmiudet piirtämiseen, kuten mahdollisuuden asettaa oliolle käytettävän shader-ohjelman. Shader-ohjelmana voi joko käyttää Cocos2d-x:n valmiita ratkaisuja tai tehdä oman. Luokkaan toteutetaan oma, useampaan polygoniin perustuva sprite (ks. kuvio 22).

```
bool CTongue::init()
{
    //      initialize vertices[]
    //      initialize textureCoordinates[]
    CCShaderCache *sc = CCShaderCache::sharedShaderCache();
```

```
    CCGLProgram *shaderProgram;
    shaderProgram = sc->programForKey(kCCShader_PositionTexture);
    this->setShaderProgram( shaderProgram );

    //      initialize mTexture

    return true;
}


void CTongue::update(float dt)
{
      //    update vertices
}
```

Varsinainen piirto tapahtuu syrjäyttämällä CCNode-luokan oma funktio **draw()**. Funktion sisälle rakennetaan aivan normaalia OpenGL ES -rajapintaa hyödyntävää ohjelmallisuutta. Tulee kuitenkin huomata, että Cocos2d-x sisältää paljon sekä staattisia, että CCNode-olion funktioita moniin OpenGL ES:ään liittyviin tehtäviin, kuten käytettävät tekstuurin asettamiseen tai shader-ohjelman käyttämiseen.

```
void CTongue::draw()
{
    ccGLBindTexture2D( mTexture->getName() );

    this->getShaderProgram()->use();
    this->getShaderProgram()->setUniformsForBuiltins();

    ccGLEnableVertexAttribs( kCCVertexAttribFlag_Position |

      kCCVertexAttribFlag_TexCoords);

    glVertexAttribPointer(kCCVertexAttrib_Position, 3, GL_FLOAT,
                                                          GL_FALSE,
0, vertices);

    glVertexAttribPointer(kCCVertexAttrib_TexCoords, 2, GL_FLOAT,
                                                       GL_FALSE, 0,
textureCoordinates);

    glDrawArrays(GL_TRIANGLE_STRIP, 0, 8);

    CC_INCREMENT_GL_DRAWS(1);

}
```

6.2.3 Tasoiteltu delta-aika

Delta-ajalla tarkoitetaan aikaa, joka kului edellisen ruudun päivittämiseen. Sitä käytetään jokaisessa ruudussa kertoimena, jotta saadaan todellisen suuruinen muutos suhteessa kuluneeseen aikaan. Mikäli delta-aikaa ei käytetä, hidastuu ja nopeutuu pelisovelluksen pelaajalle näkyvä pyörimisnopeus sovelluksen oman suoritusnopeuden mukaisesti. Tämä ei monissa peleissä, kuten pelaajan reaktionopeuteen perustuvissa toi-

mintapeleissä, ole suotavaa.

Ihannemaailmassa delta-aika olisi käytettävissä juuri sellaisenaan kuin se annetaan. Todellisuudessa useiden delta-aikojen jatkumo sisältää sovelluksen suorituskykyrasituksen luomia arvopiikkejä ja jopa virhearvioita. Lopputuloksena on epätasainen funktiokäyrän profiili. Koska delta-aika on olennainen osa peliolioiden liikuttelua näyttönäkymässä, syntyy pelaajan näkökulmasta häiritsevän epätasaista liikettä. Peleissä joissa ei ole ennakoimattomia tapahtumia johtuen pelaajan ja pelisovelluksen välisestä interaktiosta, ja joissa pyörimisnopeudella ei ole merkitystä, käytetäänkin tästä syystä kiinteää delta-aikaa.

Waterlyn tapauksessa kiinteä delta-aika on pois suljettu ratkaisu, sillä peli perustuu selkeästi pelaajan kykyyn reagoida tarpeeksi nopeasti pelin tapahtumiin. Yksi tapa tasoittaa delta-aikojen aikajatkumoprofiilia on laskea viimeisempien delta-aikojen keskiarvo. Tällöin delta-aikojen muutokset ovat hillittyjä, ylisuuret piikit karsiutuvat ja pelisovelluksen suorituslogiikka säilyy reaktiivisena suhteessa suoritusnopeuteen. Pelisovellus säilyy siis aikasuoritteisesti stabiilina.

```
float CDeltaTimeSmoother::runStepAndReturnSmoothed(float dt)
{
    deltaTimeSumUp += dt;
    deltaTimeSumUp -= deltaTimes[oldestDeltaTimeIndex];
    float averageDeltaTime = deltaTimeSumUp * 0.05;
    deltaTimes[oldestDeltaTimeIndex] = dt;

    oldestDeltaTimeIndex++;
    if (oldestDeltaTimeIndex >= 20) {
        oldestDeltaTimeIndex = 0;
    }

    return averageDeltaTime;
}
```

Keskiarvon laskemiseen tarkoitetussa funktiossa voitaisiin myös käyttää tavallisen taulukon sijaan std:vector-oliota, jolloin funktion ohjelmakoodi säilyttäisi ammattimaisemman ulkoasun. Toteutettu funktio olisi kuitenkin suorituskykyraskaampi johtuen vector-olion omasta toiminnallisuudesta liittyen muistivarauksen ja alkioiden järjestelyyn. Voitaisiin kuitenkin argumentoida, että alkioiden lukumäärä on sekä lukuarvoltaan (ei kahden potenssi) että suuruudeltaan sellainen, ettei syntynyt suorituskyky-

kyrasite ole suuremmassa kokonaisuudessa merkittävä.

```
float CDeltaTimeSmoother::runStepAndReturnSmoothed(float dt)
{
    deltaSumUp += dt;
    deltaTimes.push_back(dt);

    while (deltaTimes.size() > 20) {
        deltaSumUp -= deltaTimes.front();
        deltaTimes.erase(deltaTimes.begin());
    }

      return deltaSumUp * 0.05;
}
```

7 LOPPUPÄÄTELMÄT

Vaikka mobiilipelit ovat esimerkiksi AAA-konsolipeleihin verrattuina huomattavasti pienempiä, ovat ne silti samoja lainalaisuuksia ja kriteereitä noudattavia poikkitieteellisiä/-taiteellisia kokonaisuuksia. Jokainen pelisovelluksessa käytettävä osaamisalue on oma itsenäinen kokonaisuutensa eikä niihin syvempi perehtyminen tämän opinnäytetyön osalta olisi ollut edes mahdollista. Cocos2d-x -sovelluskehys kuitenkin perustuu näille periaatteille, sillä sen on tarkoitus yhdistää eri komponentit ja muodostaa niistä saumattoman kokonaisuuden: pelisovelluksen.

Cocos2d-x pohjautuu useamman vuoden kypsyneeseen Cocos2d-iPhone -sovelluskehykseen ja on kehittynyt sen rinnalla jo muutaman vuoden ajan. Projekti on aktiivinen ja sen ympärille on kasvanut huomattava käyttäjäympäristö. Sovelluskehys sisältää paljon hyödyllistä toteutusta erilaisten luokkien ja funktioiden muodossa, joihin tässä opinnäytetyössä on luotu vain alustava katsaus.

Sovelluskehys mahdollistaa suhteellisen nopean prototyyppien valmistamisen ja kehitystyön. Monet perustason komponentit löytyvät valmiina ja toimivina toteutuksina. Tämä ei kuitenkaan sulje pois mahdollisuutta laajentaa kokonaisuutta omilla ratkaisuilla. Esimerkiksi omien piirtofunktioiden luominen on vaivatonta, ja niiden helpottamiseksi on sovelluskehykseen pakattu hyödyllisiä funktioita ja toiminnallisuuksia.

Sovelluskehyksen käyttö ei kuitenkaan poista tarpeellisuutta yleisen ja aihekohtaisen teorian tuntemukseen ja hallintaan. Yhteisöineen ja verkosta löytyvine materiaaleineen se kuitenkin antaa tukevan alkuvauhdin aloittelevalle pelinkehittäjälle. Laadukkaine kirjastoineen ja monipuolisine toteutuksineen sovelluskehys jouduttaa ja tarjoaa ratkaisuja erityisesti vakavampien peliprojektien toteuttamiseen.

LÄHTEET

Box2D. 2013. Wikipedia. Viitattu 12.5.2013. http://en.wikipedia.org/wiki/Box2D

Cocos2d. 2013. Wikipedia. Viitattu 23.4.2013. http://en.wikipedia.org/wiki/Cocos2d

Cocos2d.org. 2012. Cocos2d:n sivustolla. Viitattu 23.4.2013. http://cocos2d.org

Cocos2d-x | Architecture and Director Structure. 2013. Cocos2d-x:n sivustolla. Viitattu 23.4.2013. http://www.cocos2d-x.org/projects/cocos2d-x/wiki/Architecture_and_Directory_Structure

Cocos2d-x Architecture. 2013. Kaavakuva Cocos2d-x:n sivustolla. Viitattu 23.4.2013. http://www.cocos2d-x.org/projects/cocos2d-x/wiki/Architecture_and_Directory_Structure

Cocos2d-x | Texture Cache. 2013. Cocos2d-x:n sivustolla. Viitattu 23.4.2013. http://www.cocos2d-x.org/projects/cocos2d-x/wiki/Texture_Cache

Cocos2d-x | Relationships in Cocos2d Family. 2013. Cocos2d-x:n sivustolla. Viitattu 23.4.2013. http://www.cocos2d-x.org/projects/cocos2d-x/wiki/Relationships_in_Cocos2d_Family

Cross-platform. 2013. Wikipedia. Viitattu 23.4.2013. http://en.wikipedia.org/wiki/Cross_platform

iOS App Programming Guide: Core App Objects. 2013. Apple iOS Developer -kirjasto. Viitattu 23.4.2013. http://developer.apple.com/library/ios/#documentation/iphone/conceptual/iphoneosprogrammingguide/AppArchitecture/AppArchitecture.html

Itterheim, S. & Löw, A. 2012. Learn cocos2d 2 – Game Development for iOS. New York: Apress

MIT License. 2013. Wikipedia. Viitattu 12.5.2013. http://en.wikipedia.org/wiki/MIT_License

OpenGL ES - The Standard for Embedded Accelerated 3D Graphics. 2013. Khronos Group -yhteenliittymän kotisivulla. Viitattu 12.5.2013. http://www.khronos.org/opengles/

Singleton Pattern. 2013. Wikipedia. Viitattu 23.4.2013. http://en.wikipedia.org/wiki/Singleton_pattern

Software Framework. 2013. Wikipedia. Viitattu 23.4.2013.

http://en.wikipedia.org/wiki/Software_framework

Usage share of operating systems. 2013. Wikipedia. Viitattu 23.4.2013.
http://en.wikipedia.org/wiki/Usage_share_of_operating_systems