

Juho Lommi

Ohjelmistokehyksen toteuttaminen 2D-mobiilipeleille



Tradenomi
Tietojenkäsittely
Syksy 2019



KAMK • University
of Applied Sciences

Tiivistelmä

Tekijä: Juho Lommi

Työn nimi: Ohjelmistokehysten toteuttaminen 2D-mobiilipeleille

Tutkintonimike: Tradenomi (AMK), tietojenkäsittely

Asiasanat: Ohjelmistokehys, mobiili, videopeli, ohjelmointi

Tässä opinnäytetyössä toteutettiin ohjelmistokehys 2D-mobiilipeleille C++-ohjelmointikielellä. Ohjelmistokehysten lähdekoodissa hyödynnettiin SDL2- ja OpenGL ES -ohjelmointirajapintoja ominaisuuksien, kuten käyttäjäsyötteen, renderöinnin, äänentoiston ja peli-ikkunan toteuttamiseksi. Tavoitteena oli kehittää ohjelmointitaitoja ja saada aikaiseksi ohjelmistokehys, jota voisi jatkossa hyödyntää tulevien peliprojektien kehityksessä.

Opinnäytetyön raportissa esiteltiin ohjelmistokehysten ominaisuuksia, lähdekoodirakenteita sekä teoriaa, jota hyödynnettiin LMGF-ohjelmistokehysten kehityksessä. Raportissa keskityttiin erityisesti ohjelmistokehysiin, joita hyödynnetään videopelien tekemiseen. Opinnäytetyöhön toteutetusta LMGF-ohjelmistokehyksestä käytiin läpi lähdekoodiin ja projektikansioon liittyviä rakenteita, kehysten alustamisen ja käyttöönoton vaiheita, toteutetut ominaisuudet ja mahdollisia parannusehdotuksia tulevaisuutta ajatellen.

LMGF-ohjelmistokehysten lisäksi opinnäytetyössä toteutettiin esimerkkiprojekti, jonka avulla LMGF:n toimivuutta testattiin käytännössä. Raportissa annettiin tietoa siitä, kuinka LMGF:n lähdekoodia hyödynnettiin pelin oman lähdekoodin kanssa pelin toimivuuden kannalta tärkeiden ominaisuuksien toteuttamiseksi. Esimerkkiprojektista rakennettiin onnistuneesti Android mobiilisovelluksen lisäksi myös Windows PC:llä suoritettava versio.

Päätelmässä todettiin, että ohjelmistokehys ja sitä varten tehty esimerkkiprojekti toteuttivat suunnitellut tavoitteet, vaikka lisäominaisuuksia olisi aina mahdollista toteuttaa tulevaisuudessa.

Abstract

Author: Lommi Juho

Title of the Publication: Creating a software framework for 2D mobile games

Degree Title: Bachelor of Business, Information Technology

Keywords: Software framework, mobile, video game, programming

In this thesis, a software framework for 2D mobile games was written using the C++ programming language. SDL2 and OpenGL ES application interfaces were utilized inside the framework's source code in the implementation of features such as user input, rendering, audio and game window management. The goal was to learn programming related skills and to create a working framework, that could be utilized when creating new game projects in the future.

The thesis report introduced basic qualities, source code structures and theory that were applied while creating the LMGF framework. The report focuses mainly on software frameworks that are used for making video games. Basic structure of the LMGF framework and its code architecture were explained, as well as the necessary steps for accessing and initializing the framework, along with the implemented features and suggestions for possible future expansions.

In addition, an example project utilizing the LMGF framework was also made for testing the implemented features in action. Insight was given on how the functionalities of the framework were used in tandem with the game's own source code to execute the necessary features needed for the game. The example project was successfully built for both Android mobile and Windows PC platforms.

In conclusion all the planned goals were met for both the framework and the example project, even though there is always room for more polish and features in the future.

Alkusanat

Videopelien tekeminen on parhaimmillaan luovaa työtä, jossa pelin kehittäjä voi itse määritellä omat työskentelymenetelmänsä ja käytetyt työkalut haluamallaan tavalla. Tutkimalla ja kokeilemalla eri työskentelytapoja jokainen voi laajentaa tietotaitoa ja osaamistaan paremman työtehokkuuden saavuttamiseksi.

Sisällys

1	Johdanto	1
2	Ohjelmistokehys.....	2
2.1	Yleisiä piirteitä ja rakenteita	3
2.2	Käyttö pelialalla.....	7
2.3	Käyttö lähdekoodissa	10
2.3.1	Esimerkki MonoGame.....	10
2.3.2	Esimerkki Cocos2d-x	13
3	Lommi Mobile Game Framework	15
3.1	Kuvaus	15
3.2	SDL2 ja OpenGL ES 2.0.....	17
3.3	Rakenne	19
3.4	Alustaminen ja käyttöönotto	21
3.5	Käyttäjäsysteemi	23
3.6	Renderöinti	24
3.7	Äänentoisto	25
3.8	Muut ominaisuudet.....	25
3.9	Alustatuki.....	26
3.10	Ohjelmistokehityksen laajentaminen.....	27
4	Cephalopod – esimerkkiprojekti LMGF:ää käyttäen	28
4.1	LMGF:n käyttö peliprojektin lähdekoodissa	29
4.2	Android-sovelluksen rakentaminen	32
4.3	PC-alustalle rakentaminen.....	35
5	Tulosten analysointi	36
6	Yhteenveto.....	37

Lähteet

Liitteet

Termit ja lyhenteet

2D-peli - Kaksiulotteiseen avaruuteen sijoittuva peli.

3D-peli - Kolmiulotteiseen tila-avaruuteen sijoittuva peli.

Asset – Pelien tekemiseen käytettyjä tiedostoja, joita ovat esimerkiksi tekstuurit, spritet, 3D-mallit, äänitiedostot.

C – Dennis Ritchien suunnittelema yleiskäyttöinen ohjelmointikieli.

C++ - Bjarne Stroustrupin suunnittelema C-ohjelmointikielestä jatkettu yleiskäyttöinen ohjelmointikieli, joka tukee mm. olio-ohjelmointiin liittyviä ominaisuuksia.

C# - Microsoftin suunnittelema yleiskäyttöinen .NET-ohjelmistokomponenttikirjaston ympärille kehitetty ohjelmointikieli.

CPU - Prosessori. Tietokoneen osa, joka suorittaa tietokoneohjelman sisäisiä käskyjä ja prosesseja.

Debugging – Virheen etsintää ja korjausta ohjelman lähdekoodista.

Funktio – Itsenäinen osa lähdekoodia, joka sisältää komentoja ja jota voidaan kutsua lähdekoodissa ohjelman suorituksen aikana.

GPU - Grafiikkaprosessori. Mikropiiri, joka suorittaa 2D- tai 3D-grafiikan piirtämistä näytölle.

Instanssi – Esiintymä luokasta tai structista ohjelman lähdekoodissa.

Laskentatehoiltaan parempia kuin yleiset prosessorit.

Käyttäjäsysteemi - Komentojen syöttämistä tietokoneohjelmalle esimerkiksi näppäimistön, kosketusnäytön tai äänen avulla.

Käyttöjärjestelmä - Keskeinen tietokoneen ohjelmisto, joka mahdollistaa tietokoneohjelmien toiminnan. Esim. Windows, Linux, iOS.

LMGF - Lommi Mobile Game Framework. Opinnäytetyön aikana toteutettu ohjelmistokehys 2D-mobiilipeleille.

Lähdekoodi - Lista ohjelmointikielellä kirjoitettuja komentoja, jotka tietokoneohjelma suorittaa.

Makefile – Tiedosto, joka sisältää lähdekoodin kääntämiseen liittyviä komentoja ja asetuksia.

Muuttuja – Englanniksi variable. Tiedon (Datan) varastointitapa lähdekoodin sisällä.

Ohjelmakirjasto - Kokoelma lähdekoodia, jota käytetään tietokoneohjelman suorittamiseen.

Ohjelmistokehys - Ohjelmistotuote, joka muodostaa rungon sen päälle rakennettavalle ohjelmalle.

Ohjelmointiparadigma – Tapa ajatella ja lähestyä ongelmia ohjelmoinnissa ja ohjelmien kirjoittamisessa.

Ohjelmointirajapinta – Tietokoneohjelmaan liittyvä käyttöliittymä, jonka avulla ohjelman käyttäjä voi tehdä yksinkertaisia kutsuja monimutkaisempiin ohjelman toteutuksiin.

Olio-ohjelmointi – Ohjelmointiparadigma, jossa yhdistellään tietoa ja toiminnallisuutta sisältäviä luokkia ohjelmoinnissa esiintyvien haasteiden ratkaisemiseksi.

OpenGL – Ohjelmointirajapinta, joka hyödyntää tietokoneen GPU:ta grafiikan renderöinnissä.

OpenGL ES – OpenGL-kirjaston pienempi alaryhmä, jota käytetään etenkin mobiilipelien yhteydessä.

Osoitin – Englanniksi pointer. Muuttujatyyppi, joka viittaa ohjelman käyttämässä muistiosoitteessa olevaan arvoon.

Parametri – Funktioon syötettyä dataa.

Pelimoottori – Ohjelmistokehystä suurempi kokonaisuus, joka kattaa laajan määrän videopelien tekoa varten tarkoitettuja ominaisuuksia ja järjestelmiä.

Pseudokoodi – Oikeaa toimivaa lähdekoodia kuvailevaa koodia.

Renderöinti – 3D-mallin piirtämistä näyttöpäätteelle 2D-muotoon.

SDL2 – Ohjelmointirajapinta, joka sisältää paljon pelien tekoa varten hyödyllisiä ominaisuuksia eri laitealustoille.

Sprite – Kaksiulotteinen pikseligrafiikasta koostuva kuva.

Struct – Suomeksi tietue. Kokoelma muuttujia.

Taulukko – Englanniksi array. Muuttujista, luokista, tai structeista koostuva tietoa alue ohjelman hyödyntämässä muistitilassa.

Tietokoneohjelma - Lähdekoodista koostuva tehtäväkokonaisuus, jonka tietokone suorittaa. Ottaa usein vastaan käyttäjän syötettä, ja palauttaa sen perusteella jonkun lopputuloksen.

Varjostin - Englanniksi shader. Pieni tietokoneohjelma, jonka avulla toteutetaan renderöintiin vaikuttavia efektejä, esimerkiksi esineen valaisua, varjostusta tai kuvan vääristämistä.

Wrapper funktio – Funktio, joka kutsuu toista funktiota kutsutun funktion yksinkertaistamiseksi.

.apk - Android-sovelluksen tiedostotyyppi.

.ipa - iOS-sovelluksen tiedostotyyppi.

1 Johdanto

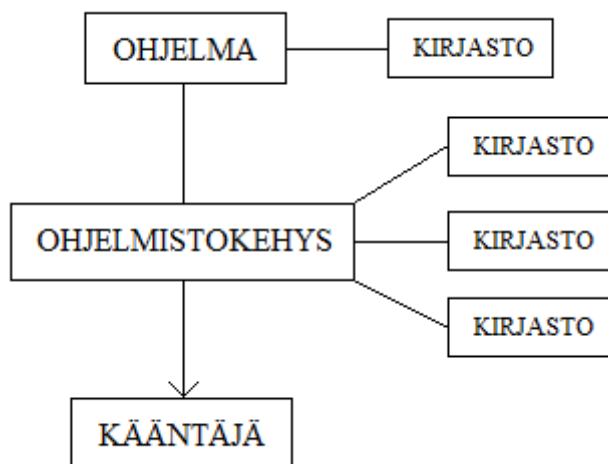
Videopelien suosio ja kehitys on ollut jatkuvassa kasvussa 1970-luvulta alkaen [1]. Vanhat peliohjelmit suorittavat laitteet kuten Atari2600 sisälsivät hyvin vähän muistia ja suorituskyyä nykyaikaisiin pelikonsoleihin verrattuna, jonka takia jokainen peli piti ohjelmoida pitkälti yksittäistapauksena resurssien käytön maksimoimiseksi, eikä suuria ohjelmistokokonaisuuksia pystytty vielä hyödyntämään tehokkaasti. Ajan myötä videopelikonsolien ja tietokoneiden resurssit kasvoivat, jonka seurauksena myös ohjelmia rajoittavat tekijät pienenivät. Laitteiden resurssien kasvu mahdollisti pelien kehitykseen kohdistuvien ohjelmistokehitysten ja pelimoottorien syntymisen. [2.]

Pelien kehitystä varten toteutetut ohjelmistokokonaisuudet sisältävät grafiikan renderöintiin, käyttäjäsyötteen vastaanottamiseen ja datan käsittelyyn liittyviä ominaisuuksia, joita useimmat videopelit tarvitsevat miellyttävän pelikokemuksen saavuttamiseksi. Hyvin toteutetut pelimoottorit ja ohjelmistokehitykset helpottavat pelien tekijöitä tuottamaan uusia pelejä nopeammin, koska niihin kirjoitettua lähdekoodia voidaan käyttää uudelleen useassa eri peliprojektissa. [3.]

Tässä opinnäytetyössä toteutetaan LMGF (Lommi Mobile Game Framework) ohjelmistokehitys, joka kattaa ominaisuudet yksinkertaisten 2D-mobiilipelien kehittämiseksi. Opinnäytetyössä toteutetaan myös esimerkkiprojekti LMGF:ää käyttäen. Tavoitteena on kirjoittaa ohjelmistokehityksen lähdekoodi C/C++-ohjelmointikielellä sellaiseen muotoon, että sitä voidaan hyödyntää Android-sovelluksien rakentamisessa, sekä tutustuttaa lukija ohjelmistokehitysten ominaispiirteisiin ja aiheeseen liittyvään teoriaan. Tarkoitus on, että LMGF:ää voisi hyödyntää tulevaisuudessa uusien peliprojektien tekemisessä ja laajentaa tarvittaessa uusilla ominaisuuksilla. Lisätavoitteina on opetella ohjelmointiin liittyviä taitoja, kuten ohjelmistorakenteiden suunnittelua, OpenGL ES-ohjelmointirajapinnan käyttämistä grafiikan renderöimisessä sekä Android-sovelluksien rakentamiseen liittyviä vaiheita. Opinnäytetyössä käydään läpi ohjelmistokehityksiin liittyvää teoriaa, LMGF:n toteuttamisen vaiheita, LMGF:n käyttöä esimerkkiprojektin avulla sekä lopuksi pohdintaa ja ajatuksia projektin lopputuloksesta. Opinnäytetyö on suunnattu kaikille ohjelmoinnista ja videopelien tekemisestä kiinnostuneille henkilöille. Opinnäytetyön ymmärtämistä helpottaa ohjelmoinnin perusteiden tunteminen sekä kokemukset videopeliprojekteissa työskentelystä. Apuna opinnäytetyössä on käytetty aiheeseen liittyvää kirjallisuutta sekä nettiartikkeleita, joita referoidaan tekstin eri vaiheissa. Lisäksi muista ohjelmistokehityksistä ja pelimoottoreista on haettu vaikutteita LMGF:ää tehdessä.

2 Ohjelmistokehys

Ohjelmistokehys on tietokoneohjelmoinnissa käytetty termi, joka tarkoittaa ohjelmistotuotetta, jonka pohjalle voi rakentaa uusia tietokoneohjelmia. Ohjelmistokehys koostuu tietokoneohjelmasta riippumattomasta uudelleenkäytettävästä lähdekoodista, jonka tarkoitus on nopeuttaa ohjelmistokehitystä tarjoamalla valmiiksi kirjoitettuja lähdekoodin osia, ettei niitä tarvitsisi kirjoittaa uudelleen uusia ohjelmia tehtäessä. Ohjelmistokehys on ohjelmistokirjastoa suurempi kokonaisuus, sillä yksi ohjelmistokehys voi sisältää useita ohjelmistokirjastoja sekä muita pienempiä ohjelmia. Toisin kuin ohjelmistokirjasto, ohjelmistokehys usein määrittää ohjelman suorittamiseen liittyviä rakenteita. [4.] Ohjelmistokehyksen, sitä käyttävän ohjelman ja erinäisten muiden lähdekoodin osien kokonaisuutta kuvataan kuvassa 1.

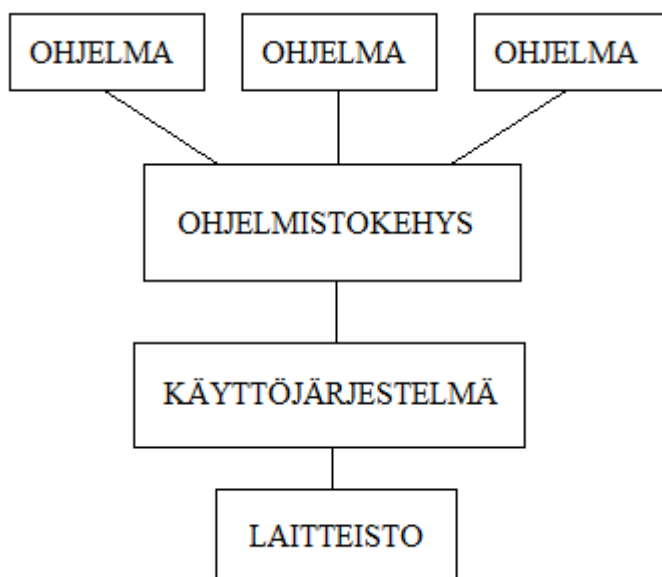


Kuva 1. Ohjelmistokehys ja sitä käyttävän ohjelman lähdekoodi muodostavat kokonaisuuden, joka lopulta käytetään kääntäjän läpi.

Ohjelmistokehyksiä voi soveltaa lähes mihin tahansa ohjelmointia vaativaan työhön. Erilaisia kehyksiä on tuotettu monien eri alojen, kuten liiketalouden, taiteiden ja sovelluskehityksen hyötykäyttöön. [5.] Vanhoissa tietokoneissa ja videopelikonsoleissa ohjelmistokehysten käyttö ei ollut yleistä, sillä laitteistojen muisti ja suorituskyky eivät riittäneet monipuolisten ohjelmistorakenteiden ylläpitämiseksi ja suorittamiseksi. Laitteistojen resurssien kasvun myötä tietokoneohjelmien rajoitteet ovat pienentyneet huomattavasti, mikä on mahdollistanut myös ohjelmistokehyksien käytön yleistymisen. Mobiililaitteilla resursseja on vähemmän verrattuna tietokoneisiin ja pelikonsoleihin, mutta myös niissä resurssit ovat kasvaneet niin suuriksi, että ohjelmistokehyksiä voi helposti hyödyntää myös mobiilisovellusten kehityksessä [6].

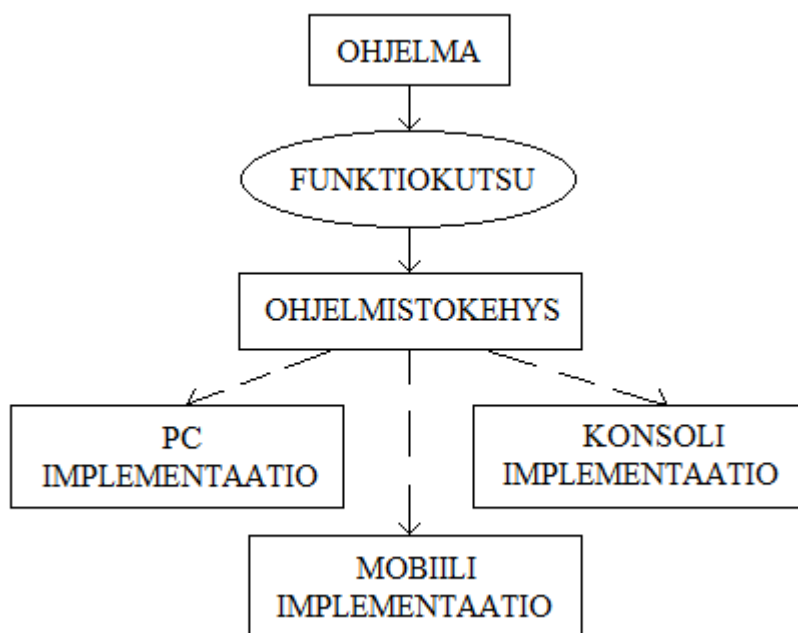
2.1 Yleisiä piirteitä ja rakenteita

Ohjelmistokehystä hyödyntävä ohjelma rakennetaan ohjelmistokehykseen luodun pohjan päälle. Yleinen käyttöperiaate ohjelmistokehyksille on, että kehystä käyttävä ohjelmoija voi hyödyntää kehysten tarjoamaa yleiskäyttöistä lähdekoodia omiin tarkoituksiinsa ilman, että kehykseen tarvitsisi tehdä muutoksia ohjelman kehityksen aikana. [7.] Ohjelmistokehystä on mahdollista käyttää monen eri ohjelman pohjana, kun kehysten lähdekoodi on kirjoitettu riippumattomaksi sitä käyttävästä ohjelmasta. Lähdekoodi säilyy riippumattomana, kun ohjelmistokehys sisällytetään sitä käyttävän ohjelman lähdekoodiin, eikä päinvastoin. Ohjelmistokehysten lähdekoodin ei myöskään tule sisältää instansseja sitä käyttävän ohjelman lähdekoodissa määriteltyihin muuttujiin ja luokkatyyppeihin tai funktioihin. Koko laitteiston, käyttöjärjestelmän, ohjelmistokehysten ja ohjelman riippuvuutta toisistaan kuvataan kuvassa 2. Ylempänä oleva taso on aina riippuvainen alempana olevasta tasosta.



Kuva 2. Monen eri ohjelman lähdekoodi voi hyödyntää saman ohjelmistokehysten lähdekoodia.

Ohjelmistokehykset tarjoavat usein käyttäjälle lähdekoodin sisäisiä funktioita erilaisten komentojen toteuttamiseksi. Ideaalitalanteessa käyttäjän kutsuma funktio toteuttaa monimutkaisempia komentoja laitealustaan tai kehysten ominaisuuksiin liittyen ilman, että käyttäjän tarvitsisi huolehtia toteutukseen liittyvästä logiikasta sen enempää. Esimerkiksi yksi käyttäjäsyötteeseen liittyvä funktio voi sisältää ohjelmistokehysten lähdekoodissa implementaation monelle eri laitealustalle, kuten kuvassa 3 on visualisoitu.



Kuva 3. Ohjelmistokehyksen tarjoama funktio voi toteuttaa erilaisen lähdekoodin sisäisen implementaation esimerkiksi laitealustasta riippuen.

Toisin kuin tavallinen ohjelmakirjasto, ohjelmistokehys sisältää usein ohjelman suoritukseen liittyviä rakenteita. Etenkin videopeleihin käytetyissä ohjelmistokehyksissä grafiikanpiirtoon, ruudunpäivityksiin ja käyttäjäsyötteisiin liittyvä suorituslogiikka on valmiiksi määritetty ohjelmistokehyksen sisäiseen pääsilmutkaan tehokkaan suorituskyvyn ja yleisen toimivuuden saavuttamiseksi [8]. Käyttäjän tulisi tutkia ohjelmistokehyksen suorituslogiikkaa arvioidakseen sen sopivuutta haluamaansa projektiin.

Laitteiston määrittämien rajoitusten puutteessa tärkeäksi osaksi ohjelmistokehysten toteuttamista ovat nousseet suunnittelumallit, joiden mukaan kehysten lähdekoodiin kirjoitetut ominaisuudet voidaan suunnitella, rajata ja toteuttaa helpommin ymmärrettävämmiksi ja muokattaviksi kokonaisuuksiksi. Toimiva ohjelmistokehys nopeuttaa ohjelmistokehitystä, mutta ilman kattavia suunnitelmia kehysten lähdekoodi voi ajan myötä kasvaa niin massiiviseksi ja epäselväksi kokonaisuudeksi, että sitä on lopulta vaikea ymmärtää, käyttää tai ylläpitää. Huonosti toteutetusta ohjelmistokehyksestä voi olla käyttäjälle enemmän haittaa kuin hyötyä. Olio-ohjelmointiin soveltuvat ohjelmointikielet, kuten C++ tai C#, ovat suosittuja kieliä ohjelmistokehysten toteutuksessa, koska ne mahdollistavat lähdekoodin kirjoittamisen suunnittelumallien mukaisesti. Ideaalitulanteessa lähdekoodin modulaarisuus mahdollistaa uusien osien lisäämisen ohjelmistokehykseen ilman, että muutokset vaikuttavat vanhojen ominaisuuksien toiminnallisuuteen, sekä tekee projektin helpommin lähestyttäväksi muille

ohjelmoijille. Modulaarisuuden ansiosta vanhojen ominaisuuksien muokkaaminen tai poistaminen ohjelmistokehityksen lähdekoodista on myös helpompaa. [9.]

Kokemuksen perusteella ohjelmistokehykseen joudutaan tekemään muutoksia, kun halutaan lisätä uusia ominaisuuksia, laitteistotukia, ohjelmakirjastoja tai rakenteellisia muutoksia kehityksen suoritukseen liittyen. Ennen lähdekoodin kirjoittamista on aina tärkeää suunnitella mahdollisimman pitkälle, mihin tarkoitukseen ohjelmistokehystä tullaan käyttämään, että lähdekoodi voidaan alusta alkaen kirjoittaa tavoitetta tukevaan muotoon, ja isoilta rakenteellisilta muutoksilta välttyttäisiin tuotteen kehityksen aikana.

Hyvin rakennettuja ohjelmistokehyksiä voidaan laajentaa käytännössä loputtomiin, ja vanhimpien ohjelmistokehysten pitkäikäisyys on usein kattavan suunnittelutyön ansiota. Lähdekoodin ymmärrettävyys, yhtenäisyys ja modulaarisuus ovat ominaisuuksia, jotka edesauttavat ohjelmistokehityksen pitkäikäisyyttä, jos projektissa on osallisena monta ohjelmoijaa pitkän ajanjakson aikana. [10.] Kirjallinen dokumentaatio on myös hyödyllinen lisä ohjelmistokehystä. Dokumentaation ja helppolukuisen lähdekoodin avulla täysin ulkopuolinen käyttäjä voi ymmärtää, kuinka ohjelmistokehitys toimii jopa vuosia kehityksen valmistumisen jälkeen.

Ohjelmistokehyksiä voi toteuttaa monella eri tyylillä tekijästä riippuen. Ohjelmointikielestä ja suunnittelumalleista huolimatta aina olisi kuitenkin tärkeää noudattaa mihin tahansa ohjelmaan yleisesti hyväksi todettuja ominaisuuksia, joita ovat mm. lähdekoodin muokattavuus, ominaisuuksien riippumattomuus toisistaan, resurssitehokkuus, helppokäyttöisyys, lähdekoodin ymmärrettävyys, alustariippumattomuus, käyttöön otettavuus ja uudelleen käytettävyys [11].

Ohjelmistokehitys on helposti muokattava, kun ohjelmistokehykseen voi lisätä, poistaa tai muokata osia ilman suuria muutoksia ennalta kirjoitettuun lähdekoodiin [12]. Muokattavuutta edesauttaa ohjelmoinnissa pätevien periaatteiden kuten SOLID, DRY (Don't repeat yourself) ja KISS (Keep It Simple Stupid) noudattaminen, sekä modulaarinen lähestymistapa lähdekoodin kirjoittamiseen. Lähdekoodin erottelu pienempiin kokonaisuuksiin voi myös auttaa käyttäjää ymmärtämään ohjelmistokehityksen rakennetta ja toiminnallisuuksia paremmin.

Resurssitehokas lähdekoodi vie mahdollisimman vähän laitteiston muistia suorituksen aikana ja on nopealukuista laitteen prosessorille [13]. Resurssien käytön optimoimisessa on yleisesti kannattavaa keskittyä ensisijaisesti isoimpiin ongelmakohtiin ja lähdekoodin osiin, joita suoritetaan usein ohjelman elinkaaren aikana. Videopelien tekemiseen tarkoitetuissa ohjelmistokehyksissä painopisteet sijoittuvat usein tekstuurien, kolmiulotteisten mallien,

äänitiedostojen sekä muiden asettien säilömiseen muistissa, sekä pääsil mukassa toteutettuun lähdekoodiin, koska sitä suoritetaan jokaisella ruudunpäivityksellä [14]. Jos laitteisto ei anna tiukkoja määritelmiä resurssienkäytön suhteen, käyttäjä joutuu itse määrittelemään, kuinka paljon resursseja ohjelman tulisi käyttää, ja kirjoittamaan lähdekoodi sen mukaisesti. Resurssitehokkuuden saavuttamiseksi ohjelmoija voi hyödyntää tietojenkäsittelyssä toimivia algoritmeja, ohjelmointikielen tietämystä sekä muistinhallintaan liittyviä tekniikoita. Ohjelmointikielet, jotka mahdollistavat manuaalisen muistinhallinnan, kuten C tai C++, soveltuvat hyvin resurssitehokkaaseen ohjelmistokehyksen tuottamiseen, koska ohjelmoija pystyy itse määrittelemään, millä tavalla ohjelmistokehyksen data on säilötyinä ohjelman muistiin, ja manipuloimaan ohjelman suoritusta sen mukaisesti. [15.]

Lähdekoodin ymmärrettävyyttä edesauttaa muuttujien, funktioiden, luokkien ja structien nimeäminen tarkoitusta kuvaavilla tavoilla, sekä yleisesti hyväksi todettujen ohjelmointikäytänteiden noudattaminen aina kun mahdollista. Lähdekoodin kirjoitusasu on hyvä pysyä samanlaisena kaikissa ohjelmistokehyksen lähdekoodin osissa, jotta käyttäjä pystyisi tekemään totuudenmukaisia oletuksia ja ennakkoluuloja kehyksen toiminnallisuuden suhteen. Helppolukuinen lähdekoodi auttaa käyttäjää ymmärtämään ohjelman toiminnallisuutta paremmin, ilman että aikaa joudutaan käyttämään lähdekoodin logiikan selvittämiseen. [16.] Esimerkkejä muuttujien nimeämisestä listauksessa 1 ja 2.

```
int d;
int ds;
int dsm;
int faid;
```

Listaus 1. Huonosti tarkoitusta kuvaavia muuttujia.

```
int elapsedTimeInDays;
int daysSinceCreation;
int daysSinceModification;
int fileAgeInDays;
```

Listaus 2. Selkeämmin tarkoitusta kuvaavia muuttujia.

Ohjelmistokehyksen käyttöönotettavuus riippuu toimintojen määrästä, jonka käyttäjä joutuu tekemään ennen kuin pystyy hyödyntämään kehystä omassa projektissaan. Isommat pelimoottorit ja ohjelmistokehykset tarjoavat käyttäjälle asennustiedostoja, joiden avulla kehyksen käyttöönotolle tarvittavat tiedostot voi ladata ja asettaa oikeisiin kansiorakenteisiin automaattisesti. Jos ohjelmistokehys koostuu vain erillisistä lähdekooditiedostoista, sen

käyttöönottettavuus voi näkyä esimerkiksi kehukseen tehtyjen kutsujen määrässä kehystä käyttävän ohjelman lähdekoodissa.

Helppokäyttöisyys on käyttäjäkohtainen kokemus siitä, kuinka nopeaa, tehokasta ja miellyttävää tuotteen käyttäminen on [17]. Ohjelmistokehityksessä helppokäyttöisyyttä edesauttaa aikaisemmin mainittu lähdekoodin ymmärrettävyys, ohjelmistokehityksen selkeä rakenne, dokumentaatio sekä kaikki työskentelyä nopeuttavat ominaisuudet, kuten debug toiminnallisuudet ja visuaaliset työkalut. Käyttäjäkokeemusta voi kehittää paremmaksi testaamalla tuotetta monen eri kohderyhmän käytössä ja huomioimalla testeistä saatu palaute kehitystyössä.

Ohjelmistokehitys on uudelleenkäytettävä, kun sitä voi hyödyntää monessa projektissa ilman, että kehityksen lähdekoodiin tarvitsee tehdä muutoksia [18]. Jotkut projektit voivat vaatia lisäyksiä ohjelmistokehitykseen, mutta lisäykset tulisi pystyä toteuttamaan siten, että aikaisemmat ohjelmistokehitystä hyödyntävät ohjelmat eivät rikkoutuisi muutosten takia.

Kaikista suunnittelumalleista ja yleisesti hyviksi todetuista käytänteistä huolimatta ohjelmistokehityksen tekijällä on täysi vapaus toteuttaa kehitys juuri haluamallaan tavalla. Tärkeintä on, että ohjelmistokehitys keventää käyttäjän työmäärää eikä päinvastoin.

2.2 Käyttö pelialalla

Nykyään suurella budjetilla tuotetut videopelit sisältävät valtavan määrän ominaisuuksia, joita varten on kirjoitettu paljon erilaista lähdekoodia. Pelejä tuotettaisiin huomattavasti pienempi määrä, jos erinäisiin ominaisuuksiin tehtyä lähdekoodia ei hyödynnettäisi useammassa kuin yhdessä projektissa.

Pelien teon yhteydessä ohjelmistokehitysten lisäksi puhutaan usein pelimoottoreista. Ohjelmistokehitysten tavoin pelimoottorit tarjoavat käyttäjälle valmiita pohjia, ominaisuuksia ja työkaluja uusien peliprojektien toteuttamiselle. Pelimoottorin ja ohjelmistokehityksen ero on häilyvä, mutta usein pelimoottoria pidetään ohjelmistokehityksiin verrattuna suurempana kokonaisuutena kattavamman ominaisuusmäärän takia [19][20]. Voidaan ajatella, että monet pelimoottorit ovat saaneet alkunsa ohjelmistokehityksinä, mutta kasvamisen myötä niistä on alettu käyttää termiä pelimoottori. Termi on kuitenkin huonosti määritelty, ja usein niistä puhutaan toistensa synonyymeinä. Mainittavia esimerkkejä suosituista pelimoottoreista ovat

Unity, Unreal Engine 4 ja GameMaker:Studio, kun taas MonoGame ja Cocos2d on mielletty enemmän ohjelmistokehyksiksi.

Edellä mainittuja esimerkkejä voidaan kuvailla yleiskäyttöisinä pelimoottoreina ja ohjelmistokehyksinä, koska ne tarjoavat valmiita pohjia erittäin monipuolisten pelien toteuttamiselle. Yleiskäyttöiset työkalut tarjoavat ratkaisuja mahdollisimman moneen ongelmaan, toisin kun esimerkiksi yksinomaan ajopelejä varten toteutetut ohjelmistokehykset tai pelimoottorit, joiden tarkoitus on toteuttaa vain ajopelejä varten vaadittavat ominaisuudet mahdollisimman tehokkaasti. Yleiskäyttöiset työkalut ovat eniten käytettyjä, mutta niissäkin on huonot puolensa. Uusia ohjelmistokehyksiä ja pelimoottoreita kehitetään jatkuvasti, koska valmiit ratkaisut eivät välttämättä miellytä kaikkia käyttäjiä, tai ne eivät ole tarpeeksi sopivia tietynlaisten pelien toteuttamiseksi. [21] Joskus peli tarvitsee erittäin tehokasta ratkaisua esimerkiksi grafiikanpiirtoa varten, jolloin yleiskäyttöinen ratkaisu ei välttämättä mahdollista pelin toteutusta halutulla tavalla, etenkin jos käyttäjällä ei ole mahdollisuutta tehdä muutoksia moottorin tai kehyksen lähdekoodiin. Rajauksia ohjelmistokehyksiin ja pelimoottoreihin tehdään useimmiten ohjelmointikielen, laitteistoalustan, ulottuvuuden (2D / 3D) ja peligenren mukaan. [22.]

Ohjelmistotuotteiden käyttöehdoissa ja lisensseissä on myös paljon eroja. Monet ohjelmistokehykset ja pelimoottorit ovat kaupallisia tuotteita, joihin liittyy erilaisia käyttömaksuja, mutta on olemassa myös ilmaisia tuotteita, jotka ovat vapaasti saatavilla internetistä kenen tahansa käyttöön. Nykyään kaupallisten työkalujen tapa tehdä rahaa vaihtelee kertaostosta kuukausimaksuihin ja jopa lisenssimaksuihin käyttäjien tekemistä tuotteista. Työkalujen tekijät voivat melko vapaasti määritellä itsellensä mieluisen tavan myydä tuotteitaan, ja loppupäässä käyttäjät tekevät ostopäätöksen oman käyttötarpeensa arvion mukaisesti. Ohjelmistokehykset kuten MonoGame ovat julkisesti avoimia projekteja, joihin kuka tahansa pystyy ehdottamaan muutoksia projektin lähdekoodiin, kun taas Unity on pelimoottori, johon käyttäjillä ei ole oikeuksia nähdä tai muokata kaikkia lähdekoodin osia. Lähdekoodin näkyvyys on käyttäjänäkökulmasta merkittävä ominaisuus, koska aina ei ole selvää, johtuuko ohjelmassa ilmenevä ongelma käyttäjän vai työkalun lähdekoodista.

Muutama vuosikymmen sitten ohjelmistokehyksien ja pelimoottoreiden käyttö videopelien tuotannossa oli yleistä vain isojen yritysten käytössä, koska niiden kehittäminen vaati paljon aikaa, rahaa ja tietotaitoa. Nykyään internet on mahdollistanut työkalujen saatavuuden helposti jokaisen käyttöön, ja kynnys pelien tekemiselle on madaltunut huomattavasti. Kaupalliset

työkalut pelien tekemistä varten ovat kasvaneet merkittäväksi tulonlähteeksi monelle yritykselle, ja trendi todennäköisesti jatkaa kasvua tulevaisuudessakin. [23.]

2.3 Käyttö lähdekoodissa

Tässä luvussa kuvaillaan, miten ohjelmistokehyksiä voidaan käyttää ohjelman lähdekoodissa.

2.3.1 Esimerkki MonoGame

MonoGame on Microsoft Public Licensellä julkaistu pelinkehitykseen tarkoitettu ilmainen ohjelmistokehys, joka pohjautuu aikaisemmin kehitettyyn XNA-ohjelmistokehykseen. MonoGame ja sitä käyttävät ohjelmat on kirjoitettu C#-ohjelmointikielellä. [24.] Listauksessa 3 esimerkki MonoGamea hyödyntävän ohjelman lähdekoodista.

```

using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace Example
{
    public class ExampleGame : Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        Texture2D texture;
        Vector2 position;

        public ExampleGame()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            position = new Vector2(0, 0);
        }

        protected override void Initialize()
        {
            texture = new Texture2D(this.GraphicsDevice, 100, 100);
            Color[] colorData = new Color[100 * 100];
            for (int i = 0; i < 10000; i++)
                colorData[i] = Color.Red;

            texture.SetData<Color>(colorData);
            base.Initialize();
        }

        protected override void LoadContent()
        {
            spriteBatch = new SpriteBatch(GraphicsDevice);
        }

        protected override void UnloadContent()
        {
        }

        protected override void Update(GameTime gameTime)
        {
            if (Keyboard.GetState().IsKeyDown(Keys.Escape))
                Exit();

            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin();
            spriteBatch.Draw(texture, position);
            spriteBatch.End();
            base.Draw(gameTime);
        }
    }
}

```

Listaus 3. MonoGame esimerkki.

Esimerkkilähdekoodin alussa MonoGame lähdekoodi sisällytetään ohjelmaan `using Microsoft.Xna.Framework`-komennoilla. Kuten kaikissa muissakin ohjelmistokehyksiä hyödyntävissä ohjelmissa, kehyksen lähdekooditiedostot täytyy ensin ladata käyttäjän tietokoneelle, ennen kuin ne voi sisällyttää osaksi projektia. MonoGame on eroteltu moneen eri nimiavaruuteen, että käyttäjä voi halutessaan hyödyntää vain tiettyä kehyksen osa-aluetta, kuten renderöintiin tai käyttäjäsyötteeseen liittyvää lähdekoodia.

Perimällä ohjelmistokehyksessä määritellyn `Game`-luokan `ExampleGame`-luokan sisällä voi yliajaa `Game`-luokassa määriteltyjä `Initialize-`, `LoadContent-`, `UnloadContent-`, `Update-` ja `Draw-`funktioita. Näiden funktioiden sisälle käyttäjä voi kirjoittaa haluamaansa pelilogiikkaa, mutta funktioiden suorittamisjärjestys ja syvempi logiikka tapahtuu ohjelmistokehyksen sisällä määritellyssä pääsilmukassa. [25.]

`GraphicsDeviceManager`, `SpriteBatch` ja `Texture2D` ovat ohjelmistokehyksen sisäisesti määritellyjä luokkia, jotka sisältävät grafiikanpiirtoon liittyvää lähdekoodia. Kyseiset luokat ovat esimerkkejä lähdekoodista, joita on haluttu pystyä uudelleenkäyttämään monessa eri projektissa. MonoGame ja muut tunnetuimmat ohjelmistokehykset sisältävät valtavan määrän muitakin luokkia, joita käyttäjä voi hyödyntää omiin tarkoituksiin projektissaan.

2.3.2 Esimerkki Cocos2d-x

Cocos2d-x on MIT-lisenssillä julkaistu pelejä ja muita sovelluksia varten kehitetty ohjelmistokehys, joka pohjautuu alun perin Python-ohjelmointikielellä kirjoitettuun Cocos2d-ohjelmistokehykseen. Cocos2d-x ja sitä käyttävät ohjelmat on kirjoitettu C++-ohjelmointikielellä. [26.] Listauksessa 4 esimerkki Cocos2d-x:ää hyödyntävän ohjelman lähdekoodista.

```
#pragma once

#include "cocos2d.h"

class Example : public cocos2d::Layer
{
public:
    static cocos2d::Scene* createScene();
    virtual bool init() override;
    CREATE_FUNC(Example);

    void update(float) override;

private:
    cocos2d::Sprite* sprite;
};

cocos2d::Scene* Example::createScene()
{
    auto scene = cocos2d::Scene::create();
    auto layer = Example::create();
    scene->addChild(layer);
    return scene;
}

bool Example::init()
{
    if (!Layer::init())
    {
        return false;
    }

    sprite = cocos2d::Sprite::create("example.png");
    this->addChild(sprite, 0);

    return true;
}
```

Listaus 4. Cocos2d-x esimerkki.

Käyttötavoiltaan Monogamessa ja Cocos2d-x:ssä on paljon samankaltaisuuksia, vaikka ohjelmistokehykset on kirjoitettu eri ohjelmointikielellä.

Esimerkkikoodin alussa ohjelmistokehys sisällytetään ohjelman lähdekoodiin `#include "cocos2d.h"`-komennolla. Perimällä `cocos2d::Layer`-luokan `Example`-luokassa pystytään käyttäjän omiin tarkoituksiin yliajamaan `update()`-, `init()`- ja muita funktioita, jotka suoritetaan ohjelmistokehysten sisäisen pääsilman mukaisesti, käytännössä samalla ajatuksella kuin `MonoGame`-esimerkissä. Kuten `MonoGame`, myös `Cocos2d-x` tarjoaa käyttäjälle hyödyllisiä luokkia, kuten `cocos2d::Scene` ja `cocos2d::Sprite`, joita esimerkkikoodissa hyödynnetään grafiikkatiedostojen lataamiseen ja piirtämiseen näytölle.

`Cocos2d-x` ja `MonoGame` on molemmat kirjoitettu olio-ohjelmoinnin periaatteiden mukaisesti, mikä näkyy luokkaperinnän, datan näkyvyyden rajoittamisen sekä monimuotoisten ja yliajettavien funktioiden käyttämisessä lähdekoodin rakenteessa. Esimerkeistä voi nähdä, että ohjelmistokehysten lähdekoodissa on paljon ohjelmointikielestä, ohjelmistokehysten rakenteesta ja valitusta kirjoitusasusta syntyviä eroavaisuuksia, vaikka kehysten käyttötarkoitukset ovat hyvin samankaltaisia.

Molemmista ohjelmistokehysistä on kirjoitettu internetiin kattava dokumentaatio, jonka avulla käyttäjä voi ymmärtää syvällisemmin ohjelmistokehysten toimintarakenteita sekä tutkia, minkälaisia ominaisuuksia kyseisiin ohjelmistokehysiin kuuluu [27] [28].


3 Lommi Mobile Game Framework

3.1 Kuvaus

LMGF on SDL2:ta ja OpenGL ES:ää hyödyntävä C/C++:lla kirjoitettu 2D-mobiilipelien tekemiseen tarkoitettu ohjelmistokehys, joka tukee Android-mobiili- ja Windows PC -laitealustoja. LMGF toimi harjoitusprojektina ja yrityksenä rakentaa omatekoinen ohjelmistokehys yksinkertaisten 2D-videopeliprototyyppien tekemistä varten. Lähdekoodi kääntyy PC:lle ja Android-sovelluksille helposti muutamaa asetusta säätämällä. Ohjelmointikieleksi valittiin C++, koska se kääntyy hyvin monelle eri laitealustalle, antaa mahdollisuuden tarkalle muistinhallinnalle ja on tunnettu ohjelmointikieli, jota moni ohjelmoija ymmärtää. Vaikka kielenä oli C++, lähdekoodi on C-painotteisesti kirjoitettu. C-painotteisuudella tarkoitetaan tässä tapauksessa sitä, että olio-ohjelmoinnin ominaisuuksia ei ole käytetty ohjelmistokehysten lähdekoodissa lainkaan. Lähdekoodin toteuttamisessa noudatettiin aikaisemmassa luvussa esiteltyjä ohjelmistokehysten ominaispiirteitä ja ohjelmointikäytänteitä.

Kehyksen käytännönläheinen toteuttamissuunnitelma perustuu pitkälti aikaisemmista peliprojekteista saatuihin kokemuksiin, joissa huomattiin, kuinka tietyt lähdekoodin osat esiintyivät monessa eri projektissa lähes identtisellä tavalla. Peliprojektien tuotannon nopeuttamiseksi oli kannattavaa kirjoittaa peliohjelmasta riippumaton ohjelmistokehys, joka sisältäisi valmiiksi yleisesti tarpeelliset ominaisuudet pelin toteuttamiselle. LMGF:ään lisättiin ominaisuuksia, jotka todettiin monessa aikaisemmassa projektissa tarpeellisiksi, ja kehyksen käyttötapoja paranneltiin opinnäytetyön aikana toteutetun esimerkkiprojektin perusteella.

Opinnäytetyön aikana ohjelmistokehykseen tehtiin ominaisuuksiin kuuluu pääsilmukan ja ohjelman suorittamiseen liittyviä rakenteita, käytettyjen ohjelmakirjastojen alustamisen vaiheita, tuki tekstuuriin, fonttien ja yksinkertaisen geometrian renderöinnille OpenGL ES -varjostimia hyödyntäen, tuki äänitiedostojen soittamiselle, käyttäjäsyötteen käsittely mobiili- ja PC-alustoilla sekä datan käsittelyyn liittyvää lähdekoodia. LMGF:n kansiorakenne näkyy kuvassa 4.

 .git	15.9.2019 23:22	Tiedostokansio	
 deps	27.5.2018 20:46	Tiedostokansio	
 extras	27.6.2019 20:58	Tiedostokansio	
 lib	7.7.2019 20:30	Tiedostokansio	
 licenses	15.9.2019 23:22	Tiedostokansio	
 shaders	28.4.2019 16:55	Tiedostokansio	
 .gitignore	25.8.2019 18:21	Tekstitiedosto	1 kt
 app.cpp	25.8.2019 19:29	C++ Source	5 kt
 app.h	25.8.2019 17:42	C/C++ Header	3 kt
 audio.cpp	25.8.2019 16:30	C++ Source	2 kt
 audio.h	25.8.2019 16:30	C/C++ Header	1 kt
 defs.cpp	15.3.2019 20:15	C++ Source	1 kt
 defs.h	25.8.2019 23:59	C/C++ Header	17 kt
 draw.cpp	25.8.2019 23:24	C++ Source	24 kt
 draw.h	25.8.2019 18:00	C/C++ Header	5 kt
 input.cpp	25.8.2019 17:42	C++ Source	5 kt
 input.h	25.8.2019 17:42	C/C++ Header	2 kt
 logo.png	28.4.2019 17:06	PNG-tiedosto	2 kt
 mklink.txt	18.1.2019 14:49	Tekstitiedosto	1 kt
 README.txt	15.9.2019 23:21	Tekstitiedosto	1 kt
 text.cpp	25.8.2019 16:46	C++ Source	2 kt
 text.h	25.8.2019 16:45	C/C++ Header	1 kt
 utils.cpp	26.8.2019 0:04	C++ Source	4 kt
 utils.h	26.8.2019 0:04	C/C++ Header	5 kt
 window.cpp	25.8.2019 16:12	C++ Source	1 kt
 window.h	15.9.2019 22:00	C/C++ Header	1 kt

Kuva 4. LMGF:n kansiorakenne.

Linkki LMGF:n lähdekoodiin löytyy opinnäytetyön liitteistä.

3.2 SDL2 ja OpenGL ES 2.0

SDL (Simple DirectMedia Layer) on C-ohjelmointikielellä kirjoitettu ohjelmointirajapinta, joka tarjoaa käyttäjälle helppokäyttöisiä ja alustariippumattomia rajapintakutsuja, joita voi hyödyntää pelien teossa sekä muissa hyötyohjelmissa. SDL-kirjasto tarjoaa käyttäjälle muun muassa käyttäjäsyötteeseen, äänentoistoon ja grafiikanpiirtoon liittyviä ominaisuuksia. Tässä opinnäytetyössä käytettiin SDL:n zlib-lisenssillä julkaistua SDL2-versiota, joka antaa valtuudet käyttää ohjelmointirajapintaa vapaasti ohjelmakehityksessä myös kaupallisiin projekteihin. SDL-kirjastoa on laajennettu erillisillä kirjastoilla, jotka täydentävät peruskirjaston ominaisuuksia. SDL-kirjaston lisäksi opinnäytetyöprojektissa on hyödynnetty SDL_image-, SDL_mixer- ja SDL_ttf-kirjastoja peli-ikkunan hallintaan, grafiikanpiirtoon, äänentoistoon ja fonttitiedostojen käsittelyyn. SDL on yhteensopiva grafiikanpiirtoon keskittyvien OpenGL-, Direct3D- ja Vulkan-ohjelmointirajapintojen kanssa, mikä teki SDL:stä hyvän vaihtoehdon peliprojektien toteuttamiselle. [29.]

OpenGL (Open Graphics Library) on C-ohjelmointikielellä kirjoitettu alustariippumaton ohjelmointirajapinta, joka on kehitetty ensisijaisesti 3D-renderöintiä varten, mutta sitä voidaan hyvin soveltaa myös 2D-visuaalien renderöinnissä. OpenGL hyödyntää laitteiston GPU:ta laskennan toteuttamiseksi CPU:n sijaan. LMGF:ssä on käytetty OpenGL:n pienempää kokonaisuutta OpenGL ES 2.0 yksinkertaisen 2D-grafiikan, kuten spritejen ja geometrian piirtämiseen. OpenGL ES valittiin, koska se on mobiili- ja PC-alustoille suunnattu pienempi kokonaisuus perus-OpenGL:stä. Varjostimissa käytetty OpenGL ES SL on myös suhteellisen helppokäyttöinen kieli varjostimien kirjoittamiseen. [30.] LMGF tarvitsee molempia ohjelmointirajapintoja toimiakseen, joten ne on sisällytetty LMGF-projektikansion rakenteeseen.

SDL:lle ja OpenGL:lle löytyy myös vaihtoehtoisia ohjelmointirajapintoja ja kirjastoja, joita olisi voitu hyödyntää LMGF:ssä. SDL:n tilalla olisi voitu käyttää esimerkiksi GLFW-ohjelmakirjastoa, joka on tarkoitettu käytettäväksi OpenGL:n kanssa. GLFW tarjoaa käyttäjälle ominaisuuksia käyttäjäsyötteen ja peli-ikkunan hallintaan, mutta muuten ominaisuudet eivät ole yhtä kattavat kuin SDL:ssä.

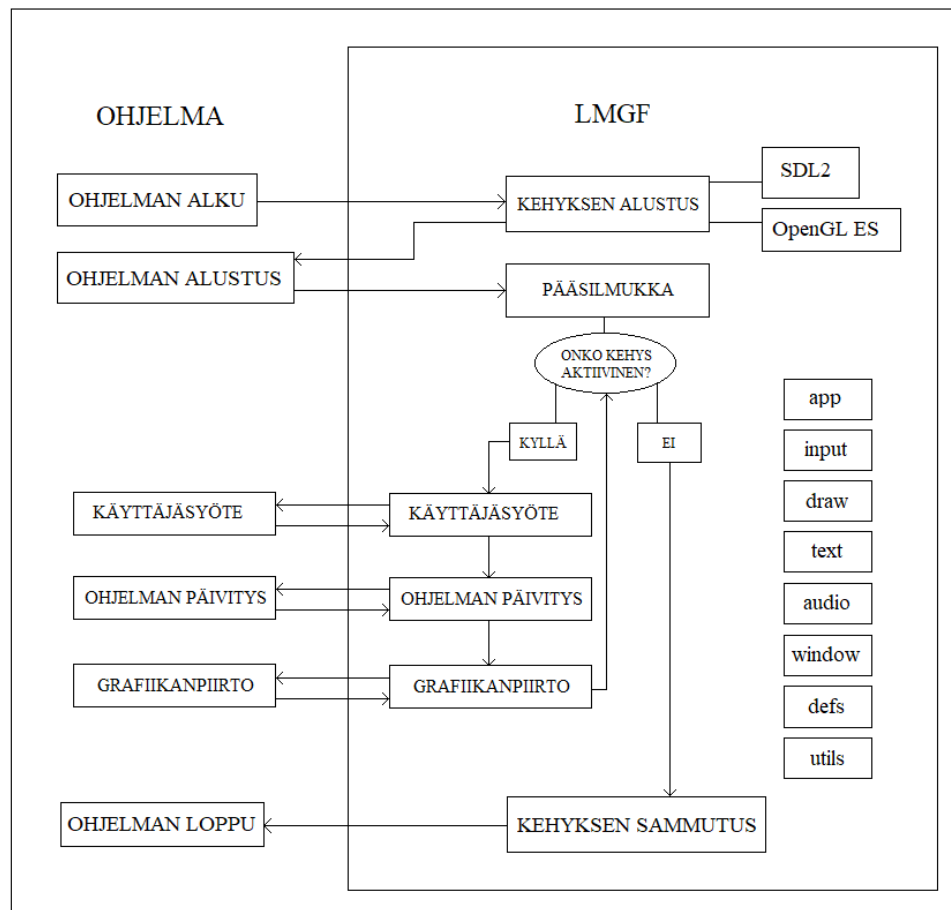
Vaihtoehto OpenGL:lle olisi ollut käyttää Vulkan-ohjelmointirajapintaa, joka myös tukee Windows- ja Android-alustoja. Opinnäytetyöprojektissa haluttiin perehtyä OpenGL:n käyttöön, joten se valittiin Vulkanin sijaan. DirectX olisi ollut toinen vaihtoehto OpenGL:n tilalle, mutta se on kehitetty toimivaksi enimmäkseen vain Windows-pohjaisille alustoille, joten sitä ei haluttu hyödyntää tässä projektissa. SDL ja OpenGL ovat projektille sopivia ohjelmointirajapintoja, koska

tulevaisuudessa olisi mahdollista laajentaa ohjelmointikehystä myös Linux- ja iOS-alustoille, joita molemmat rajapinnat tukevat.

SDL ja OpenGL vaativat jonkin verran alustusta ohjelman lähdekoodissa ennen kuin rajapintojen ominaisuuksia voi hyödyntää ohjelman sisäisesti. Molempien rajapintojen alustus tapahtuu hyvin samankaltaisesti projektista riippumatta, jonka takia niiden alustaminen on toteutettu ohjelmistokehyksen lähdekoodiin. Rajapintojen alustaminen ohjelmistokehyksen lähdekoodissa on yksi esimerkki siitä, kuinka ohjelmistokehyksen lähdekoodirakenteen avulla voi nopeuttaa uusien peliprojektien kehitystä.

3.3 Rakenne

LMGF:n lähdekoodi on jaettu ominaisuuksien mukaan eri tiedostoihin, kuten App, Input, Draw ja Audio. Jaottelu helpottaa tietyn osa-alueen muokkaamista ja laajentamista sekä auttaa ymmärtämään ohjelmistokehyksen rakennetta. Kuvassa 5 kuvataan LMGF:n ja sitä käyttävän ohjelman suoritusjärjestystä.



Kuva 5. LMGF:n ja sitä käyttävän ohjelman suoritusta kuvaava rakenne.

Eri tiedostojen muuttujat on myös nimetty tiedoston nimisellä etuliitteellä, josta tunnistaa, mihin tiedostoon ohjelmistokehyksen funktio kuuluu. Esimerkiksi `Draw_render` on `draw.h`- ja `draw.cpp`-tiedostoissa määritelty funktio, jota käytetään LMGF:n sisällä määritellyssä `App_mainloop()`-funktiossa (Kuvattu listauksessa 5).

```

void
App_mainloop(App *app, Input *input, Renderer *renderer)
{
    app->running = 1;

    while (app->running)
    {
#ifdef MOBILE
        Clock_tick(&app->clock);
#endif
        Input_update(input, &app->window);
        App_update(app);
        Draw_render(renderer, &app->window);
    }
}

```

Listaus 5. LMGF:n lähdekoodiin määritelty App_mainloop() funktio.

App-tiedostot toimivat kehyksen ytimenä. Kaikki tärkeimmät lähdekooditiedostot sisällytetään ja alustetaan app.h-tiedoston alussa. Kehyksen pääsilmukka ja käytettyjen kirjastojen ja rajapintojen alustaminen suoritetaan app.cpp-tiedoston lähdekoodissa. Pääsilmukan suoritusrjestyks on input, update, render järjestyksessä, joka on yleisesti käytetty järjestys videopeleissä. [31.]

Nykyisessä versiossa kaikki LMGF:n tärkeimmät lähdekooditiedostot täytyy kääntää ohjelman lähdekoodin kanssa, eli ominaisuudet ovat pitkälti riippuvaisia toisistaan. Listauksen 6 sisältämässä lähdekoodissa näkyy, kuinka LMGF:n tarvitsemia tiedostoja sisällytetään kehyksen `app.h`-tiedostossa.

```
#pragma once

#include <errno.h>
#include <time.h>

#include "defs.h"

#ifdef MOBILE
#include <unistd.h>
#include "deps/SDL2-2.0.8/include/SDL.h"
#include "deps/SDL2-2.0.8/include/SDL_opengles.h"
#include "deps/SDL2_mixer-2.0.2/SDL_mixer.h"
#include "deps/SDL2_image-2.0.2/SDL_image.h"
#include "deps/SDL2_ttf-2.0.14/SDL_ttf.h"
#else
#include <stdio.h>
#include <Windows.h>
#include "deps/SDL2/SDL.h"
#include "deps/SDL2/SDL_image.h"
#include "deps/SDL2/SDL_ttf.h"
#include "deps/SDL2/SDL_mixer.h"
#include "deps/GL/glew.h"
#include "deps/SDL2/SDL_opengl.h"
#endif

#include "window.h"
#include "utils.h"
#include "input.h"
#include "draw.h"
#include "text.h"
#include "audio.h"
```

Listaus 6. `App.h`-tiedostossa sisällytetyt header-tiedostot.

Lähdekoodin riippuvaisuus kannattaa ottaa huomioon heti ohjelmistokehyksen kirjoittamisen alkuvaiheessa, koska myöhemmin ohjelmistokehyksen perusrakenteita voi olla työlästä muuttaa, jos kehyksen koko on kasvanut paljon.

3.4 Alustaminen ja käyttöönotto

LMGF:n alustamisen tärkeimmät vaiheet ovat ohjelmistokehyksen sisällyttäminen ohjelman lähdekoodiin, tärkeiden struct-instanssien luominen, struct-instansseihin kuuluvien funktio-osoittimien määrittäminen ja ohjelmistokehyksen pääsilman käynnistäminen.

Ohjelmistokehityksen käyttäjän täytyy kutsua `#include "LMGF/app.h"` kerran oman ohjelmansa lähdekoodin alussa sisällyttääkseen LMGF-lähdekoodi projektiinsa. Lähdekoodin sisällyttämisen jälkeen käyttäjän täytyy tehdä instanssi LMGF:ssä määritellyistä `App`-, `Input`- ja `Renderer`-structeista, jotka sisältävät tärkeimmät ohjelman suoritukseen liittyvät funktio-osoittimet ja muuttujat. Useimmissa tapauksissa kehystä hyödyntävä ohjelma sisältää vain yhden instanssin edellä mainituista structeista. Struct-instanssien tekemisen jälkeen käyttäjän tulee kutsua `App_init`-funktioita, jonka sisällä SDL2, OpenGL ES, sekä muut ohjelmistokehityksen ominaisuudet alustetaan käyttövalmiiksi. `App_init`-funktion sisällä asetetaan myös muita ohjelman suorituksen kannalta tärkeitä ominaisuuksia, kuten näytön resoluutio ja asset-resurssikansion tiedostopolku. `Init`-kutsun jälkeen käyttäjän tulee määrittää funktio-osoittimet ruudunpäivitykselle, renderöinnille ja käyttäjäsyötteelle, joita kutsutaan LMGF:n pääsilmaan sisäisesti. Lopuksi LMGF:n pääsilma käännistetään `App_mainloop()`-funktioilla. Listauksessa 7 minimaalinen esimerkki siitä, kuinka LMGF voidaan alustaa käyttäjän tekemän ohjelman lähdekoodissa.

```
#include "app.h"

App app;
Input input;
Renderer renderer;

int main(int argc, char **argv)
{
    App_init(&app, "Example", "../src/LMGF", "../assets/");
    app.Update = Update;
    App_mainloop(&app, &input, &renderer);

    return 0;
}

void
Update()
{
    PRINT("Hello world!\n");
}
```

Listaus 7. Esimerkki LMGF:n alustamisesta ohjelman lähdekoodissa.

LMGF:n alustaminen muistuttaa edellisessä luvussa annetuista esimerkeistä enemmän Cocos2d-x:tä kuin MonoGamea, johtuen C++-ohjelmointikielen käytöstä. LMGF:ssä struct-instanssit ja niiden sisältämät funktio-osoittimet ajavat pitkälti samaa toiminnallisuutta kuin luokkaperintä ja yliajettavat funktiot Cocos2d-x ja MonoGame-ohjelmistokehityksissä. LMGF:ssä toteutetun koodirakenteen myötä moneen ohjelmistokehityksen funktioon syötetään parametrinä osoitin struct-instanssista, että structin sisältämää dataa voidaan muokata ohjelmistokehityksen sisäisessä logiikassa.

Listauksessa 8 esimerkki LMGF:n windows.h-tiedostossa määritellystä Window-structista ja funktioita, jotka ottavat parametrina vastaan osoittimen kyseisestä structista.

```
struct Window
{
    SDL_Window *sdlWindow;
    uint width;
    uint height;
    vec2 resolution;
    float viewport[4];
    float viewportScale;
    float r, g, b, a;
};

void
Window_setResolution(Window *window, uint w, uint h);

void
Window_setViewport(Window *window, uint x, uint y, uint w, uint h);

void
Window_setColor(Window *window, float r, float g, float b, float a);
```

Listaus 8. LMGF Window-struct muuttujan määritelmä ja funktiojulistuksia.

Vaikka Cocos2d-x perustuu vahvemmin olio-ohjelmointipohjaiseen koodiarkkitehtuuriin, molempien ohjelmistokehyksien suoritusjärjestys toimii pääasiallisesti ohjelmistokehyksessä määritellyn logiikan mukaisesti, johon käyttäjä lisää pelilogiikkaa oman ohjelmansa lähdekoodissa.

3.5 Käyttäjäsyste

Käyttäjäsysteeseen liittyvät muuttujat, structit ja funktiot on kirjoitettu LMGF/input.h- ja input.cpp-tiedostoihin. Käyttäjäsysteän käsittelyssä on hyödynnetty SDL2/SDL.h-kirjaston tarjoamaa SDL_Event- ja SDL_PollEvent-rakennetta, jonka avulla käyttäjän antamia komentoja voidaan käsitellä näppäimistöstä, konsolien ohjaimista ja mobiililaitteen kosketuksista. [32.]

Kosketusnäyttöön perustuva käyttäjäsyste on rajoittuneempi verrattuna tietokoneen näppäimistöön tai peliohjaimeen, koska kosketusnäytöllisissä mobiililaitteissa ei välttämättä ole lainkaan fyysisiä nappeja tai ohjaussauvoja, joita voisi hyödyntää pelin mekaniikkojen toteuttamisessa. Kosketusnäytön käyttäjäsysteessä voi kuitenkin soveltaa yhden tai useamman sormen painalluksia sekä sormen liikkeeseen perustuvia mekaniikkoja. [33.]

LMGF:n toteutetussa käyttäjäsyötejärjestelmässä sormen kosketukset tallennetaan ohjelmistokehyksen sisäiseen taulukkoon painallusjärjestyksessä, jonka avulla voidaan rekisteröidä monta eri sormen kosketusta. Annetusta syötteestä seurataan painalluksen sijaintia sekä kosketuksen alku- ja päättymishetkiä. Vertaamalla kosketusten alku- ja loppusijainteja kosketus voidaan tulkita esimerkiksi pyyhkäisy -tai nipistyseleiksi, joita voidaan hyödyntää pelin sisäisissä mekaniikoissa. Käyttäjäsyötteen sijainti suhteutetaan peli-ikkunan kokoon, koska näytön koko ja resoluutio vaihtelee paljon eri mobiililaitteiden välillä.

Useimmat mobiililaitteet sisältävät antureita, kuten kiihtyvyysanturia ja gyroskooppia, joita voidaan myös hyödyntää pelimekaniikkojen toteutuksessa. Antureilla voidaan mitata laitteen fyysistä sijaintia ja kaltevuutta, mutta tämän opinnäytetyön aikana antureihin perustuvaa käyttäjäsyötetukea ei kuitenkaan toteutettu.

3.6 Renderöinti

Renderöintiin liittyvä lähdekoodi on kirjoitettu LMGF/draw.h- ja draw.cpp-tiedostoihin. Grafiikan renderöinti hyödyntää OpenGL ES-ohjelmointirajapintaa SDL2-kontekstissa sekä SDL_Image kirjastoa. OpenGL ES:n avulla renderöintiin liittyvä laskenta suoritetaan laitteen GPU:n avulla, mikä on huomattavasti tehokkaampaa kuin CPU:lla toteutettu renderöinti, koska GPU sisältää paljon enemmän laskentaa suorittavia prosessoreita. [34] LMGF tukee spritejen, fonttien ja yksinkertaisen geometrian, kuten kolmioiden ja neliöiden piirtämistä hyödyntäen OpenGL ES SL kielellä kirjoitettuja varjostimia.

Yksinkertaisesti selitettynä OpenGL:n toteuttama grafiikan renderöinti perustuu puskurijärjestelmään, jossa piirtokomentojen avulla verteksitaulukkoihin asetettu data syötetään varjostimien läpi GPU:lle, jonka jälkeen lopullinen tulos piirretään 2D-pikseleinä näytölle. Vaikka puskureissa syötetty verteksidata on kolmiulotteisessa muodossa (X, Y, Z-koordinaatit), sitä voi silti hyödyntää 2D-grafiikan piirtämiseen projisoimalla piirretty grafiikka ortograafisesti näytölle. [35.]

Koska GPU pystyy käsittelemään paljon dataa yhdellä piirtokomennolla, LMGF:ään toteutettiin komentojen paketoitijärjestelmä, jonka avulla jokaista grafiikanpiirtoon vaadittua komentoa ei lähetetä GPU:lle erikseen, vaan kaikki ruudunpäivityksen aikana tehdyt piirtokomennot pakataan yhteen isoon verteksitaulukkoon, joka ruudunpäivityksen lopussa lähetetään GPU:lle piirrettäväksi. LMGF-grafiikan piirroksessa on käytetty kahta eri verteksitaulukkoa, joista yksi

taulukko on omistettu spriteille ja toinen yksinkertaiselle geometrialle. Taulukot on jaettu kahteen, koska spritet ja geometria sisältävät eri määrän dataa. Piirtokutsuja voi toteuttaa PreRender-, Render- ja PostRender-funktioihin, jonka avulla voi hieman säätää grafiikan piirtojärjestystä.

Tulevaisuudessa renderöintiin liittyvää lähdekoodia olisi hyvä parantaa jakamalla renderöintijärjestyksen erillisiin syvyystasoihin ja muokkaamalla spritejen piirtoon käytetyistä struceista sellaisia, että käyttäjä voisi määritellä niihin omia varjostimiaan.

3.7 Äänentoisto

Äänentoistoon liittyvä lähdekoodi on kirjoitettu LMGF/audio.h- ja audio.cpp-tiedostoihin. Äänentoistossa on hyödynnetty SDL_mixer-kirjastoa, ja ohjelmistokehykseen kirjoitettu lähdekoodi toimii pitkälti vain wrapperina SDL_mixer-funktioille ja muuttujille. Wrapperin teko on hyödyllistä, koska haluttaessa SDL_mixer-kirjasto voidaan vaihtaa johonkin toiseen äänikirjastoon ilman, että wrapper-koodia käyttävän ohjelmaan täytyy tehdä muutoksia. [36.] Nykytilanteessa LMGF tukee yksinkertaisten ääni- ja musiikkitiedostojen toistamista ja ääniasetusten säätämistä. Vaihtoehtoisesti SDL_Mixerin tilalla voisi käyttää esimerkiksi OpenAL-tai FMOD-ohjelmistorajapintoja, jotka ovat molemmat alustariippumattomia ja suunniteltu videopelien äänimaailman toteuttamiseksi.

3.8 Muut ominaisuudet

Äänentoiston, renderöinnin ja käyttäjäsyötteen lisäksi LMGF sisältää tiedostoja, joihin on toteutettu pienempiä ominaisuuksia ja ohjelmointia helpottavaa lähdekoodia.

LMGF/defs.h-tiedostoon on kirjoitettu määritelmiä ja makroja, joita hyödynnetään lähes kaikissa ohjelmistokehyksen lähdekooditiedostoissa. LMGF/utls.h- ja utls.cpp-tiedostoihin on kirjoitettu enimmäkseen matematiikan laskentaan liittyviä funktioita, joita hyödynnetään 2D-renderöinnin ja -fysiikan toteuttamisessa. Utls-lähdekoodi sisältää myös muita hyödyllisiä ominaisuuksia, kuten tiedostojen lukemiseen ja kirjoittamiseen liittyviä funktioita, joiden avulla käyttäjä voi tallentaa dataa erilliseen tiedostoon ohjelman suorituksen aikana. Utls-tiedoston lähdekoodi tulisi olla niin yleiskäyttöistä, että sitä voisi hyödyntää myös

ohjelmistokehyksen ulkopuolisissa projekteissa. Olisi oikeastaan kannattavaa ajatella hyötykäyttöisen lähdekoodin kirjoittamista omaksi ohjelmakirjastoksi, johon voisi aina lisätä uutta ohjelmointia helpottavaa lähdekoodia projektista riippumatta.

LMGF/text.h- ja text.cpp-tiedostot sisältävät fonttitiedostojen lataamiseen ja renderöintiin liittyvää lähdekoodia. Fonttien käsittelyssä on hyödynnetty SDL_ttf-kirjastoa. Myös fonttimuuttujille on tehty wrapperi, että tarvittaessa SDL_ttf:n vaihtaminen johonkin toiseen kirjastoon ei olisi niin työlästä.

LMGF/window.h- ja window.cpp-tiedostot sisältävät peli-ikkunan hallintaan liittyvän Window-structin ja siihen liittyviä funktioita. Window-structin sisältämiä muuttujia käytetään paljon renderöinnin ja käyttäjäsyötteen yhteydessä.

3.9 Alustatuki

LMGF:n tukema Android-mobiili- ja Windows PC-alustatuki toteutettiin ohjelmistokehyksen lähdekoodiin kirjoitetuilla kääntäjäaikaikaisilla komennoilla. PC- ja mobiilialustalle kuuluva lähdekoodi on eroteltu toisistaan `#ifdef MOBILE` -komennoilla, jotka takaavat, että vain halutulle alustalle tarkoitettu lähdekoodi otetaan huomioon koodin kääntämisvaiheessa. Listauksessa 9 pseudokoodia kääntäjäaikaikaisista komennoista.

```
#define MOBILE

#ifdef MOBILE
    // mobiili implementaatio
#else
    // joku muu implementaatio
#endif
```

Listaus 9. Esimerkki kääntäjäaikaisista komennoista.

Parannusehdotus alustariippumattoman lähdekoodin kirjoittamiselle olisi ollut jakaa eri alustaimplementaatiot omiin lähdekooditiedostoihinsa ja käyttää kääntäjäaikaisia komentoja tiedostojen lisäämiseen `#include`-vaiheessa.

Opinnäytetyön aikana LMGF:ään ei toteutettu iOS-alustatukea. IOS-tuki voitaisiin myös toteuttaa kääntäjäaikaisilla komennoilla samalla tavalla kuin muut alustatuet, eli kirjoittamalla lähdekoodi `#ifdef IOS`-alueiden sisälle.

3.10 Ohjelmistokehyksen laajentaminen

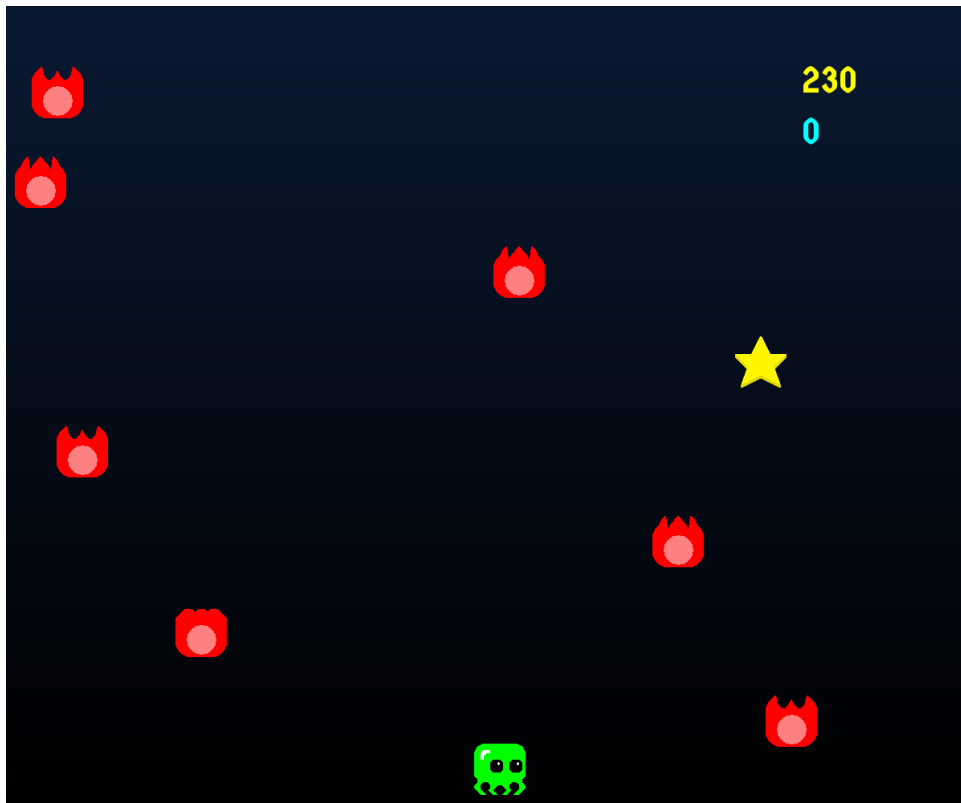
LMGF:n ominaisuuksia voi lisätä tai laajentaa noudattamalla vanhan lähdekoodin rakennetta ja kirjoitusasua. Esimerkiksi jos kehykseen lisättäisiin verkkopelaamiseen liittyviä ominaisuuksia, ne voitaisiin kirjoittaa LMGF/networking.h- ja networking.cpp-tiedostoiksi LMGF:n projektikansioon. Uudet tiedostot tulee sisällyttää ja alustaa app.h- ja app.cpp-tiedostoissa. Lisäksi tiedostopolut täytyy lisätä käyttävän ohjelman makefileen, että ne otetaan huomioon ohjelman kääntämisvaiheessa.

Tämänhetkinen versio LMGF:stä on vielä hyvin minimaalinen, mutta omatekoisessa kehyksessä on se hyvä puoli, että sitä voi laajentaa käytännössä loputtomiin juuri tekijän haluamalla tavalla. Jatkossa hyviä ominaisuuksia ohjelmistokehykseen olisi iOS-laitetuki, mobiilienturien hyödyntäminen käyttäjäsyötteessä, moniajoa tukeva pääsilmutta renderöintiä ja verkko-ominaisuuksia varten, peliobjektien muistinhallintajärjestelmä, käyttäjän määrittelemien varjostimien tukeminen renderöinnissä, konfiguraatitiedostot kehyksen asetuksille, visuaaliset editointi- ja debug-työkalut, sekä paljon muuta. Tarpeeksi pitkälle laajennettuna LMGF:stä voitaisiin alkaa käyttää termiä pelimoottori ohjelmistokehyksen sijaan.

4 Cephalopod – esimerkkiprojekti LMGF:ää käyttäen

Cephalopod on C++-ohjelmointikielellä kirjoitettu arcade-henkinen 2D-mobiilipeli, jonka avulla LMGF:n toimivuutta testattiin käytännössä. Tarkoitus oli hyödyntää LMGF:n kirjoitettuja ominaisuuksia, korjata kehyksessä ilmeneviä puutteita ja täydentää kehiksen ominaisuuksia pelin toteuttamiseen vaadittujen tarpeiden mukaisesti.

Pelissä väistellään vihreällä oliolla peliruudun yläreunasta laskeutuvia tulipalloja, ja samalla kerätään tähden muotoisia pisteitä korkean loppupistemäärän saavuttamiseksi. Graafisesti peli koostuu spriteistä, tekstistä ja yksinkertaisesta geometriasta. Pelin sisäistä grafiikkaa näkyy kuvassa 6. Hahmoa ohjataan koskettamalla ruudun vasenta tai oikeaa puolta sormella. Tietokoneversiossa hahmon ohjaus tapahtuu hiiren vasemmanpuoleisella painikkeella. Eri tapahtumista, kuten pisteiden keräämisestä ja tulipalloihin osumisesta, syntyy ääniefektejä. Pelaajan tavoite on kerätä mahdollisimman suuri pistemäärä, joka tallennetaan laitteen muistiin erilliseen tiedostoon, josta se ladataan uudelleen ohjelman käynnistyessä. Peli loppuu, jos pelaaja osuu tulipalloon. Linkki projektin lähdekoodiin ja .apk-tiedostoon löytyy opinnäytetyön liitteistä.



Kuva 6: Pelikuva Cephalopod-esimerkkiprojektista

4.1 LMGF:n käyttö peliprojektin lähdekoodissa

LMGF:n käyttöönotto ja alustaminen tapahtuu edellisessä luvussa kuvailtujen vaiheiden mukaisesti. Ohjelmistokehys sisällytetään pelin lähdekoodiin `#include "LMGF/app.h"` komennolla, jonka jälkeen tehdään instanssit `App`-, `Input`- ja `Renderer`-structeista. Struct-instanssien luomisen jälkeen ohjelmistokehys alustetaan, ja kehyksen pääsilmutassa suoritettavat funktio-osoittimet asetetaan osoittamaan pelin oman lähdekoodin sisäisiin funktioihin, joissa pelilogiikka toteutetaan. Listauksissa 10 ja 11 lähdekoodia, jota on käytetty LMGF:n alustamiseen Cephalopod projektissa, ja esimerkki `FingerDown()`-funktio-osoittimen käyttämisestä.

```
#include "../LMGF/app.h"

App app;
Input input;
Renderer renderer;

int main(int argc, char **argv)
{
    App_init(&app, "Cephalopod", "../src/LMGF", "");
    Init();
    App_mainloop(&app, &input, &renderer);
    return 0;
}

void Init()
{
    app.Update          = Update;
    renderer.PreRender  = PreRender;
    renderer.Render      = Render;
    input.FingerUp       = FingerUp;
    input.FingerDown     = FingerDown;
    input.FingerMotion   = FingerMotion;
    input.KeyDown        = KeyDown;
    input.KeyUp          = KeyUp;
```

Listaus 10. Pelin funktioita määritellään LMGF-kehyksen sisältämiin funktio-osoittimiin.

```

Void
FingerDown()
{
    switch (gamestate)
    {
        case GS_MENU: gamestate = GS_GAME; break;
        case GS_GAMEOVER: Reset(); break;
        case GS_GAME:
        {
            if (input.touchInputs[input.fingerIndex].position.x <
                app.window.viewport[2] / 2)
            {
                TurnLeft();
            }
            else
            {
                TurnRight();
            }
        }
        }break;
    }
}

```

Listaus 11. Funktio, jota kutsutaan Cephalopod projektissa kun pelaaja koskettaa näyttöä.

Pelin lähdekoodin alussa on määritelty tarpeelliset instanssit peliobjektien spriteille, fonteille, äänitiedostoille ja animaatioille LMGF:ssä määriteltyjä Texture-, Font-, AudioClip- ja Animation-structeja käyttäen. Pelin toteuttamiseksi on kirjoitettu myös omia luokkia, jotka eivät ole osa LMGF-lähdekoodia. Instanssien muuttujiin tarvittavat resurssit ladataan projektin assets-kansiosta LMGF:n tarjoamia funktioita käyttäen. Edellä mainittuja kohtia kuvastetaan listauksissa 12, 13 ja 14.

```

Font *fontConsola;
Font *fontScore;

SoundClip *scoreSound;
SoundClip *deadSound;

SpriteFlip playerFlip;

Texture comboText;
Texture scoreText;

Texture texPlayer;
Texture texFireball;
Texture texSparkle;

```

Listaus 12. LMGF:ssä määriteltyjen struct-tyyppien instansseja.

```

GameManager gameManager;
Player player;
GameObj scoreObj;
GameObj enemies[ENEMY_ALLOCATIONS];
GameObj sparkles[SPARKLE_ALLOCATIONS];

```

Listaus 13. Cephalopod-lähdekoodissa määriteltyjen luokkien instansseja.

```

VoidInit()
{
    fontConsola = Text_loadFont("GermaniaOne-Regular.TTF", 72);
    fontScore = Text_loadFont("GermaniaOne-Regular.TTF", 48);

    scoreSound = Audio_loadSound("score.wav");
    deadSound = Audio_loadSound("dead.wav");

    Draw_loadTexture(&texPlayer, "player.png");
    Draw_loadTexture(&texFireball, "enemy.png");
    Draw_loadTexture(&texSparkle, "sparkle.png");

    Text_create(&scoreText, IntToString(bufScoreText, 0), fontScore,
        colorYellow, 1);
    Text_create(&comboText, IntToString(bufComboText, 0), fontScore,
        colorAqua, 1);

    ...
}

```

Listaus 14. Cephalopod-pelissä käytettyjen resurssien lataamista LMGF-funktioilla.

LMGF ei sisällä automaattista järjestelmää luotujen struct-instanssien ja muuttujien muistinhallinnasta, joten niistä pitää huolehtia pelin omassa lähdekoodissa. Koska LMGF:n lähdekoodissa ei ole referenssiä pelin lähdekoodissa määritellyistä muuttujista moni LMGF:ssä määritelty funktio ottaa parametrina osoittimen tarvittavasta struct instanssista, että pelin lähdekoodissa määritellyn muuttujan dataa voidaan käsitellä LMGF:n sisäisessä lähdekoodissa. Parametrien syöttämistä LMGF-funktioihin kuvastaa listaus 15, jossa Cephalopod-projektin lähdekoodissa määritellyt scoreText-, player.tex- ja enemies.tex-muuttujien osoitteet syötetään Draw_sprite()-funktioihin grafiikan piirtoa varten.

```

void DrawGame()
{
    Draw_sprite(&scoreText,
        app.window.viewport[2] - SCORE_TEXT_OFFSET - scoreText.width,
        64, 0);
    Draw_sprite(&comboText,
        app.window.viewport[2] - SCORE_TEXT_OFFSET - comboText.width,
        128, 0);

    Draw_sprite(player.tex, player.pos.x, player.pos.y,
        player.anim->frames[player.animFrame].rect, playerFlip);

    if (scoreObj.alive)
    {
        Draw_sprite(scoreObj.tex, scoreObj.pos.x, scoreObj.pos.y,
            anim.frames[0].rect);
    }

    for (int i = 0; i < ENEMY_ALLOCATIONS; ++i)
    {
        if (!enemies[i].alive)
        {
            continue;
        }

        Draw_sprite(enemies[i].tex, enemies[i].pos.x,
            enemies[i].pos.y, anim.frames[enemies[i].animFrame].rect);
    }
    ...
}

```

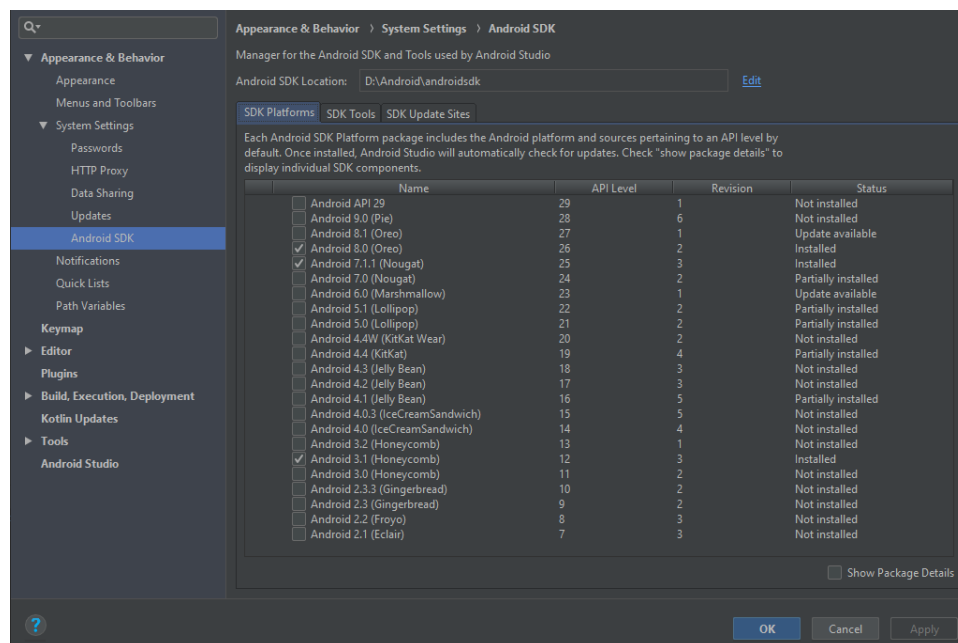
Lista 15. DrawGame()-funktio Cephalopod projektin lähdekoodissa.

4.2 Android-sovelluksen rakentaminen

Android-sovelluksen rakentaminen on monivaiheinen prosessi, joka vaatii useita ohjelmistokirjastoja, kääntäjiä ja tarkkoja kansiorakenteita toimiakseen. Prosessin helpottamiseksi Cephalopod-projektissa hyödynnettiin Android-sovellusten kehitystä varten luotua Android Studio -ohjelmointiympäristöä. Android Studio helpottaa Android-sovellusten rakentamista huomattavasti, sillä se tarjoaa käyttäjälle helppokäyttöisiä liittymiä tarvittavien tiedostojen lataamiseen sekä päivittämiseen. Lisäksi sovellus sisältää tekstieditorin, jolla lähdekooditiedostoja voi muokata, debug-työkaluja ja Gradle-rakennusjärjestelmän, jonka avulla projektikansiossa olevat tiedostot käännetään Android-laitteille suoritettavaan .apk tiedostomuotoon. [37.]

Tärkeimmät ohjelmistokehityspaketit Android-sovellusten kehityksessä ovat Android SDK (Software Development Kit) ja JDK (Java Development Kit). Android SDK sisältää ohjelmakirjastoja, debug-työkaluja ja emulaattoreita, joita hyödynnetään sovellusten kehittämisessä Android-ohjelmointirajapinnalle. Koska Android SDK on kirjoitettu Java-ohjelmointikielellä, sovellusten teko vaatii myös JDK-ohjelmistokehityspaketin toimiakseen. Koska LMGF ja Cephalopod on kirjoitettu C++-ohjelmointikielellä, Android-sovelluksen rakentamiseen tarvitaan myös NDK (Native Development Kit)-ohjelmistokehityspakettia, joka mahdollistaa C- ja C++-koodin käyttämisen Android-sovelluskehityksessä. NDK tarjoaa myös rajapintakutsuja Android-laitteen komponentteihin ja ominaisuuksiin. [38.]

Ensimmäinen askel Android-sovelluksen rakentamisessa on tarvittavien ohjelmistokehityspakettien lataaminen ja asentaminen. JDK-paketti piti ladata erikseen Oraclen omilta nettisivuilta, mutta halutun Android-ohjelmointirajapinnan SDK- ja NDK-version pystyi lataamaan kätevästi Android Studiossa olevan SDK Managerin avulla (kuva 7).



Kuva 7: Android Studion sisäinen SDK Manager.

Ohjelmistokehityspakettien lataamisen ja asentamisen jälkeen Cephalopod-projektikansio tuodaan Android Studion projektinäkömään, jossa projektiin liittyviä asetuksia ja tiedostoja voi muokata. Projekti on alustettu SDL2-2.0.8-tiedostoissa olevan android-project-kansion pohjalle, johon on valmiiksi asetettu kansiorakenteita ja asetuksia SDL2-kirjaston toimimiselle Android-alustalle.

Pelin lähdekoodi, resurssit ja suoritukseen liittyvät tiedostot on eroteltu omiin assets- ja src-kansioihinsa projektikansion sisällä. On tärkeää, että tiedostot ovat oikeissa kansioissa, koska sovelluksen rakentamiseen liittyvät tiedostopolut ovat tarkoin määriteltyjä ohjelman lähdekoodissa ja Gradle-rakennusjärjestelmän asetuksissa. LMGF- ja SDL2-lähdekoodikansiot on sisällytetty osaksi projektikansiota käyttämällä symbolisia linkkejä. Lähdekoodien tiedostopolut on määritelty projektikansiosta löytyvään Android.mk-tiedostoon, jotta ne tulevat osaksi Gradle-rakennusjärjestelmän suoritusta. Listaus 16 kuvastaa Cephalopod-projektissa käytetyn Android.mk-tiedoston sisältämät komennot.

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

OPTIMIZED_C_FLAGS = -Wall -D_USE_MATH_DEFINES -D_REENTRANT -std=c++11 -O3 -lm

LOCAL_CFLAGS += $(OPTIMIZED_C_FLAGS)

LOCAL_MODULE := main

SDL_PATH := ../SDL2

LOCAL_C_INCLUDES := $(LOCAL_PATH)/$(SDL_PATH)/include \
$(LOCAL_PATH)/../SDL2_image \
$(LOCAL_PATH)/../SDL2_ttf \
$(LOCAL_PATH)/../SDL2_mixer

LOCAL_SRC_FILES := ../LMGF/defs.cpp \
../LMGF/app.cpp \
../LMGF/utils.cpp \
../LMGF/input.cpp \
../LMGF/draw.cpp \
../LMGF/text.cpp \
../LMGF/audio.cpp \
../LMGF/window.cpp \
game.cpp

LOCAL_SHARED_LIBRARIES := SDL2 SDL2_image SDL2_ttf SDL2_mixer

LOCAL_LDLIBS := -lGLv1_CM -llog -lGLv2 -lEGL
LOCAL_LDFLAGS := -Wl,--allow-multiple-definition

include $(BUILD_SHARED_LIBRARY)
```

Listaus 16. Cephalopod-projektin Android.mk tiedoston sisältö.

Kun asetukset on asetettu oikein ja lähdekoodissa ei esiinny virheitä, projektin voi kääntää ja rakentaa .apk-tiedostoksi Android Studiosta löytyvällä ”Build and Run”-painikkeella. Apk-tiedoston suorittamiseen tarvitaan joko Android-emulaattori tai Android-käyttöjärjestelmän sisältämä mobiililaite. Opinnäytetyöprojektissa sovelluksen testaamiseen käytettiin Huawei P10

Lite -mobiililaitetta, johon oli asennettuna Android 7.0 -versio. Android Studioon sisältämän LogCat-debug-ikkunan avulla seurattiin ohjelman suorituksen aikana syntyneitä virheviestejä, joiden avulla lähdekoodissa ilmenevät virheet pystyttiin korjaamaan helpommin.

Valitettavasti opinnäytetyöprojektiin ei saatu toteutettua kansiorakennetta, joka tukisi monen Android-sovelluksen kehittämistä samasta projektikansioista. Jokaiseen uuteen peliprojektiin pitää suorittaa samat edellä mainitut vaiheet uudelleen, ja se vie turhaa aikaa projektin aloitusvaiheessa.

4.3 PC-alustalle rakentaminen

Opinnäytetyön aikana Cephalopod-projektista toteutettiin myös Windows PC:llä suoritettava versio. Lähdekoodi kääntyy Android-versiosta PC:lle poistamalla `#define MOBILE` rivin `LMGF/defs.h` tiedostosta ja asettamalla `assets`-kansion tiedostopolku oikein pelin lähdekoodissa. PC-version teko auttoi testauksessa, koska sen alustaminen ja kääntäminen oli nopeampaa kuin mobiiliversion tekeminen. PC-version tekemisessä hyödynnettiin Visual Studio-ohjelmointiympäristöä tiedostojen muokkaamisessa, lähdekoodin kääntämisessä ja PC:llä suoritettavan `.exe` tiedoston rakentamisessa.

PC-version projektikansiorakenne on huomattavasti yksinkertaisempi verrattuna Android-version projektikansioon. Visual Studion sisältämän kääntäjän avulla `.exe`-tiedosto rakennetaan suoraan C/C++-lähdekoodista, joten JDK-, SDK- ja NDK-ohjelmistokehityspaketteja tai Gradle-rakennusjärjestelmää ei tarvita lainkaan. Toimivan `.exe`-tiedoston rakentaminen vaatii kuitenkin `.lib`- ja `.dll`-tiedostot käytetyistä kirjastoista, joten ne on lisätty projektin kansiorakenteeseen.

Kuten Android-projektissa, myös PC-versiolle on oma `Makefile`-tiedosto, johon on määritelty käytettyjen lähdekooditiedostojen tiedostopolut, käytetyt ohjelmakirjastot, kääntäjäkomennot ja muut asetukset ohjelman rakentamista varten.

5 Tulosten analysointi

Nykyään videopelien kehitystä varten on olemassa niin paljon hyviä työkaluja, että omatekoisten pelimoottoreiden ja ohjelmistokehyksien kirjoittaminen on harvoin enää tarpeellista. Omien työkalujen toteuttamisessa on kuitenkin hyviä puolia, joita valmiit työkalut eivät aina voi tarjota kehittäjälle. Etenkin oppimismielessä ohjelmistokehyksen toteuttaminen oli antoisa kokemus, koska opinnäytetyöhön kirjoitetun lähdekoodin suunnittelussa ja toteuttamisessa oli täysi luova vapaus, joka kannusti itsenäistä ajattelua ja ohjelmarakenteisiin liittyvää ongelmanratkontaa. Ohjelmointikehystä tehdessä oli myös välttämätöntä opetella käytettyjen ohjelmointirajapintojen toimivuutta pintaa syvemmmältä, josta saatu tieto on sovellettavissa myös tulevilla projekteilla ja mahdollisesti myös työelämässä.

Omien työkalujen ja vapaan lähdekoodin käyttäminen projekteissa vähentää riippuvaisuutta ulkopuolisista osapuolista, jolloin tekijän ei tarvitse huolehtia esimerkiksi lisenssimaksuista tai versiopäivityksistä. Tekijällä on myös mahdollisuus kehittää omia ohjelmiaan niin pitkälle kuin haluaa ilman ulkopuolisia rajoitteita ja julkaista tuotoksiaan vapaammin. Uusien ohjelmistokehyksien tekijöille on suositeltavaa rajata kehyksen käyttötarkoitus johonkin tiettyyn aiheeseen. Esimerkiksi ohjelmistokehyksen toteuttaminen MMO (Massive Multiplayer Online) peleille voisi olla kannattavaa, koska niitä on olemassa vain hyvin vähän.

6 Yhteenveto

Ensimmäiseksi täysin itsenäisesti toteutetuksi ohjelmistokehykseksi LMGF onnistui hyvin. Tavoitteena oli toteuttaa kevyt ohjelmistokehys 2D-mobiilipelien tekoa varten C/C++-ohjelmointikielellä, ja siihen tavoitteeseen päästiin. Ohjelmistokehyksessä ja esimerkkiprojektissa käytetyn lähdekoodin rakenne, tyyli ja käyttötapa onnistuivat halutulla tavalla, koska tarkoitus oli kirjoittaa lähdekoodi enemmän C-ohjelmointikielen periaatteiden kuin olio-ohjelmointiparadigmojen mukaisesti.

LMGF:n alustaminen ja käyttäminen esimerkkiprojektissa oli vaivatonta, mutta se vaati hyvää tuntemusta ohjelmistokehyksen rakenteesta ja ominaisuuksien toteutustavoista, jonka takia uuden käyttäjän olisi todennäköisesti haastavaa käyttää kehystä omiin tarkoituksiinsa ilman dokumentaatiota. Cephalopod-esimerkkiprojektin lähdekoodissa pystyttiin keskittymään täysin pelin suunnitelman toteuttamiseen, koska tarvittaviin ominaisuuksiin ja ohjelmakirjastojen alustamiseen liittyvä logiikkaa oli kirjoitettu valmiiksi LMGF:n sisäisiin rakenteisiin. Ohjelmistokehyksen käyttö täten nopeutti itse pelin kehittämiseen vaadittavaa aikaa huomattavasti. LMGF:n kirjoitettu lähdekoodi säilyi riippumatonta kehystä käyttävän ohjelman lähdekoodista, joten sitä voi soveltaa useassa eri peliprojektissa.

Opinnäytetyön tekemisen aikana selvisi monia ohjelmistokehyksen rakentamiseen liittyviä seikkoja, jotka voidaan työstä saadulla kokemuksella ottaa paremmin huomioon tulevaisuudessa. Kansiorakenne olisi hyvä suunnitella alusta asti sellaiseksi, että etenkin Android-projekteissa samasta projektikansiosta olisi mahdollista rakentaa monta eri peliprojektia. Tämänhetkisen kansiorakenteen avulla se ei ole mahdollista, ja uuden Android-projektin alustaminen vie aikaa. Ohjelmistokehyksen sisäiset lähdekooditiedostot tulisi kirjoittaa toisistaan riippumattomiksi, että kehystä käytettäessä olisi mahdollista valita käyttöön vain tietyt ominaisuudet, kuten grafiikan renderöinti, käyttäjäsyöte tai molemmat. Opinnäytetyön aikana saatujen havaintojen perusteella ohjelmistokehyksen kasvamisen myötä on myös työläämpää tehdä suuria muutoksia kehysten perusrakenteisiin tarvittaessa, joten lähdekoodin rakenteet kannattaa suunnitella mahdollisimman monipuolisesti tulevaisuutta varten ennen kuin niitä aletaan edes kirjoittamaan.

LMGF:n kirjoitettu lähdekoodi on tyyliltään yhtenäistä, joka helpottaa ymmärtämään ohjelmistokehyksen rakennetta ja käyttöä. Käytetyistä ohjelmointikäytänteistä ja rakenteista tulee huolehtia myös uusia ominaisuuksia lisätessä ja vanhaa lähdekoodia muokattaessa. Opinnäytetyön aikana opittiin OpenGL ES:n käyttöä, vaikka implementaatio jäikin melko

yksinkertaiseksi. Jatkossa olisi kannattavaa tutkia lisää GPU:ta hyödyntävien ohjelmointirajapintojen käyttöä monipuolisempien renderöintijärjestelmien ja näyttävämpien visuaalien toteuttamiseksi.

Lähteet

- [1] Alex Ioana. The incomplete history of video game sales. Luettu 8.9.2019. <https://medium.com/the-peruser/a-brief-history-of-video-game-sales-49edbf831dc>
- [2] Sanjay Madhav. Game Programming Algorithms and Techniques: A Platform-Agnostic Approach. Luettu 8.9.2019. <http://www.informit.com/articles/article.aspx?p=2167437>
- [3] Interesting Engineering. How Do Game Engines Work. Luettu 8.9.2019. <https://interestingengineering.com/how-game-engines-work>
- [4] Techopedia. Software framework. Luettu 8.9.2019. <https://www.techopedia.com/definition/14384/software-framework>
- [5] RIApedia. Software frameworks. Luettu 8.9.2019. <https://riapedia.com/software-frameworks>
- [6] Wikipedia. Comparison of smartphones. Luettu 8.9.2019. https://en.wikipedia.org/wiki/Comparison_of_smartphones
- [7] Techopedia. Software framework. Luettu 8.9.2019. <https://www.techopedia.com/definition/14384/software-framework>
- [8] GamesFromScratch. Cocos2D-x tutorial series: Game Loop, Updates and Action handling. Luettu 8.9.2019. <https://www.gamefromscratch.com/post/2014/10/11/Cocos2d-x-Tutorial-Series-Game-loops-Updates-and-Action-Handling.aspx>
- [9] Mohamed Fayad, Douglas C. Schmidt. Object-Oriented Application Frameworks. Luettu 8.9.2019. <https://www.dre.vanderbilt.edu/~schmidt/CACM-frameworks.html>
- [10] Niteco. 5 Principles to follow when designing the architecture of your application. Luettu 8.9.2019. <https://niteco.com/resources/blogs/5-principles-for-software-architecture/>
- [11] 79xperts. What are the characteristics of a good computer program? Luettu 8.9.2019. <https://www.79xperts.com/blog/good-computer-program/>
- [12] Allan Kelly. Writing Extensible Software. Luettu 13.9.2019. <https://accu.org/index.php/journals/402>
- [13] Koushik Ghosh. Writing Efficient C and C Code Optimizations. Luettu 13.9.2019. <https://www.codeproject.com/Articles/6154/Writing-Efficient-C-and-C-Code-Optimization>
- [14] Unity Technologies. Fixing Performance Problems. Luettu 13.9.2019. <https://learn.unity.com/tutorial/fixing-performance-problems#>
- [15] Paul Hsieh. Programming Optimization. Luettu 13.9.2019. <http://www.azillionmonkeys.com/qed/optimize.html>
- [16] Manas Sadangi. 10 Tips for Improving the Readability of Your Code. Luettu 13.9.2019. <https://dzone.com/articles/10-tips-how-to-improve-the-readability-of-your-sof>
- [17] Bill Cole. What is software usability? Luettu 15.9.2019. https://blog.rocketsoftware.com/2014/11/software-usability/#.XX4-eWZS_fs
- [18] Nidhi Thakkar. Why reusability of software components is essential? Luettu 15.9.2019. <https://blog.e-zest.com/why-reusability-of-software-components-is-essential>
- [19] Stack Overflow. What's the difference between an "engine" and a "framework". Luettu 15.9.2019. <https://stackoverflow.com/questions/5068992/whats-the-difference-between-an-engine-and-a-framework>

- [20] Gamedev.net. Difference between Framework and Engine. Luettu 15.9.2019. <https://www.gamedev.net/forums/topic/679325-difference-between-framework-and-engine/>
- [21] Eric Backeberg. Why exactly are new game engines created, and how do video game companies benefit from that? Luettu 15.9.2019. <https://www.quora.com/Why-exactly-are-new-game-engines-created-and-how-do-video-game-companies-benefit-from-that>
- [22] Wikipedia. List of game engines. Luettu 15.9.2019. https://en.wikipedia.org/wiki/List_of_game_engines
- [23] Reseach briefs. The \$120B gaming industry is being built on the backs of these two engines. Luettu 15.9.2019. <https://www.cbinsights.com/research/game-engines-growth-expert-intelligence/>
- [24] MonoGame. About. Luettu 15.9.2019. <http://www.monogame.net/about/>
- [25] Chis G. Williams. The MonoGame Game Loop (just like the XNA game loop). Luettu 15.9.2019. <http://geekswithblogs.net/cwilliams/archive/2017/02/13/237027.aspx>
- [26] Cocos. Products. Luettu 15.9.2019. <https://www.cocos.com/en/products#Cocos2d-x>
- [27] MonoGame. Documentation. Luettu 15.9.2019. <http://www.monogame.net/documentation/?page=main>
- [28] Cocos. Cocos2d-x v3 Docs. Luettu 15.9.2019. <https://docs.cocos2d-x.org/cocos2d-x/v3/en/>
- [29] SDL Wiki. Luettu 15.9.2019. <https://wiki.libsdl.org/FrontPage>
- [30] OpenGL Wiki. Luettu 15.9.2019. <https://www.khronos.org/opengl/wiki/>
- [31] Robert Nystrom. Game Loop. Luettu 15.9.2019. <https://gameprogrammingpatterns.com/game-loop.html>
- [32] Libsdl.org. Handling the keyboard. Luettu 15.9.2019. <https://www.libsdl.org/release/SDL-1.2.15/docs/html/guideinputkeyboard.html>
- [33] Extra Credits. Designing for a touch screen – What games play best on mobile. Nähty 15.9.2019. <https://www.youtube.com/watch?v=NBHircZu5EI>
- [34] Tina Lee. CPU vs GPU Renderer: Which is better? Luettu 15.9.2019. <https://td-u.com/cpu-vs-gpu-renderer-which-is-better/>
- [35] Joey de Vries. Hello Triangle. Luettu 15.9.2019. <https://learnopengl.com/Getting-started/Hello-Triangle>
- [36] Stack overflow. Where and how is the term used wrapper in programming, what does it help to do? Luettu 15.9.2019. <https://stackoverflow.com/questions/3293752/where-and-how-is-the-term-used-wrapper-in-programming-what-does-it-help-to-do>
- [37] Android Studio. Meet Android Studio. Luettu 15.9.2019. <https://developer.android.com/studio/intro>
- [38] Lazyfoo. Hello mobile. Luettu 15.9.2019. https://lazyfoo.net/tutorials/SDL/52_hello_mobile/android_windows/index.php

Liitteet

LMGF lähdekoodi: <https://gitlab.com/Lommi/LMGF>

Cephalopod lähdekoodi: <https://gitlab.com/Lommi/cephalopod>