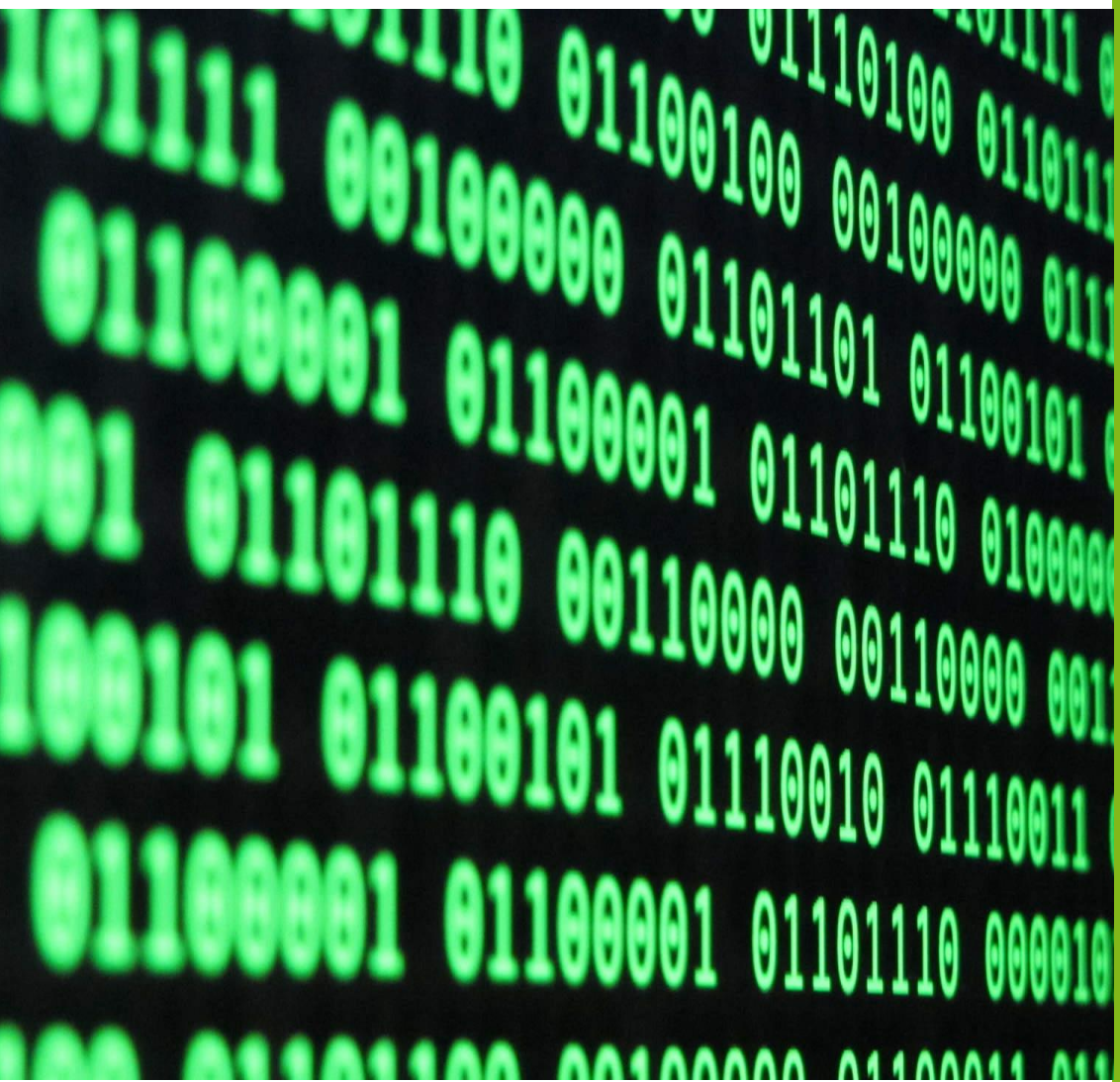
Juha Kallio-Kujala

Pelimoottori opetuskäyttöön



Tradenomi,

tietojenkäsittely

KAJAANIN
AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tekijä: Kallio-Kujala Juha

Työn nimi: Pelimoottori opetuskäyttöön

Tutkintonimike: Tradenomi, tietojenkäsittely

Asiasanat: ohjelmointi, peliohjelmointi, opetusmateriaali, grafiikka, audio

Ohjelmoinnin opettaminen on haastavaa, koska ohjelmointi itsessään on haastavaa ja koska siinä käytettäviä työkaluja ei ole suunniteltu opetusmateriaaliksi. Monessa koulussa ohjelmointia opetetaan pelien tekemisen avulla, pelimoottoria käyttäen. Tämä parantaa huomattavasti oppilaiden mielenkiintoa ja motivaatiota aiheen opiskeluun. Olemassa olevissa pelimoottoreissa on kuitenkin opetuksen näkökulmasta paljon parantamisen varaa, kuten monimutkaisuus, hankala asennusprosessi sekä teknologian mystisyys. Tässä opinnäytetyössä luotiin pelimoottori, joka on suunniteltu käytettäväksi opetusmateriaalina ja siten vastaamaan opetuksen uniikkeihin haasteisiin.

Luotu pelimoottori on peliohjelmoijalle yksinkertainen ja helppo käyttää ja asentaa. Sillä tehtyjen pelien pelaaminen ei vaadi minkäänlaista asennusta, ja se tarjoaa sopivan kirjon toiminnallisuuksia pelien luomiseksi. Pelimoottorin rakenne ja toiminta ovat yksinkertaisia, mikä sallii oppilaan kurkistaa pellin alle ja selvittää, kuinka pelimoottori toimii.

Rakennettu moottori koostuu neljästä moduulista, jotka käyttöjärjestelmän rajapintojen avulla toteuttavat peliohjelmoinnissa hyödyllisen rajapinnan. Lokimoduuli kirjoittaa tekstitiedostoihin peliohjelmoijalle hyödyllistä tietoa sekä huomauttaa tehdyistä virheistä, alustamoduuli luo ikkunan sekä sallii oppilaan reagoida hiiren liikkeeseen ja näppäimistön painalluksiin, grafiikkamoduuli mahdollistaa geometrian, kuvien ja tekstin monitorille piirtämisen ja audiomoduuli äänitiedostojen soittamisen kaiuttimista.

ABSTRACT

Author: Kallio-Kujala Juha

Title of the Publication: Game Engine for Programming Education

Degree Title: Bachelor of Business Administration

Keywords: programming, game programming, education, graphics, audio

Teaching programming is challenging, because programming in itself is a challenging activity and because the tools of the trade have not been designed to be used by novices. Many schools utilize game engines succesfully in substantially improving the morale of students. The existing game engines have many problems that decrease their educational value, like complexity, a difficult installation process and the black-box nature of the technology. This thesis attempts to create an engine that is especially designed to answer to these challenges that are unique to the world of education as opposed to the world of professional game development.

The engine developed for this thesis is simple to use and easy to install. Games made with it require no installation whatsoever, and it provides a suitable collection of functionalities for creating games. The structure and inner workings are simple, which allows the student to peek under the hood and find out how the engine works.

The developed engine consists of four modules that utilize the operating system's application programming interfaces to provide the programmer a useful programming interface for game programming. The logging module writes log messages about useful information and errors into text files while the platform module creates a window and allows the student to react to mouse movement as well as keyboard key presses. The graphics module enables the rendering of geometry, images and text onto the monitor. The audio module is used for playing back sounds from the computer's speakers.

Sisällys

1 JOHDANTO

Ohjelmoinnin opettaminen on haastavaa. Yksi suurimmista syistä on, että ohjelmointi itsessään on haastavaa. Turing-palkittu tietojenkäsittelytieteilijä Fred Brooks tarkastelee esseessään ”Ei hopealuotia” (”No Silver Bullet”) neljää syytä tähän. Ensimmäinen syy on monimutkaisuus. Ohjelmoinnin tarkoituksena on käskyttää tietokoneita, jotka itsessään ovat kenties olemassa olevista laitteista monimutkaisimpia. Mikäli käskyjä poistetaan ohjelmasta ohjelman yksinkertaistamiseksi, ei ohjelma ole enää sama. (Brooks 1987.)

Toinen Brooksin mainitsema syy on ohjelmien vaatimus olla yhteensopivia. Niiden on käytettävä esimerkiksi käyttöjärjestelmien rajapintoja, ohjelmointikielien syntaksia sekä käsiteltävä standardisoituja tiedostoformaatteja. Usein nämä rajapinnat ja standardit ovat tarpeettoman hankalia historiallisista syistä, mutta niiden muuttaminen niiden yleisyyden takia on mahdotonta. (Brooks 1987.)

Tarkastelun kohteena kolmantena syynä on muutettavuus. Tietokoneohjelmalta vaadittava toiminnallisuus muuttuu jatkuvasti, sekä kehityksen aikana että ohjelmaversiosta toiseen. Tyypillisesti uusien tuotteiden uusi versio on kokonaan uusi, sen sijaan että vanhasta versiosta on muokattava uudenlainen. Koska uudet vaatimukset eriävät vanhoista, vanhan version muokkaaminen tukemaan uusia ominaisuuksia on usein vaikeaa. (Brooks 1987.)

Viimeinen syy on ohjelmistojen näkymättömyys. Fyysisten esineiden tarkastelu paljastaa niiden rakenteen, ja piirustukset helpottavat rakenteen hahmottamista. Ohjelmiston rakenne kuitenkin on näkymätön ja vaikeasti merkityksellisellä tavalla visualisoitavissa sen lisäksi, että se tyypillisesti muuttuu ohjelman ajon aikana, kun ohjelma operaatiosta riippuen vie enemmän tai vähemmän muistia. (Brooks 1987.)

Paras tapa oppia ohjelmointia ja siihen läheisesti liittyviä aihealueita on harjoittelu, ja paras tapa saada oppilaat harjoittelemaan on tehdä oppimisesta viihdyttä-

vää ja innostavaa. Ohjelmointikielet tyypillisesti tarjoavat vain 1970-luvulta peräisin olevan komentoriviympäristön ohjelmoinnin opetteluun. Komentoriviympäristö on yksinkertainen ikkuna, johon ohjelman käyttäjä voi kirjoittaa tekstiä. Vain harva oppilas kokee ohjelmoinnin kiinnostavaksi, kun opiskelu koostuu lähinnä ohjelmista, jotka kirjoittavat yksinkertaisia lauseita tähän ikkunaan tai lukevat käyttäjän kirjoittamia yksittäisiä sanoja.

Monet oppilaitokset ovat oivaltaneet, että oppilaiden mielenkiinnon voi herättää pelien tekemisen avulla. Oppilaat kokevat pelien ohjelmoinnin mielenkiintoiseksi, koska pitävät pelien pelaamisesta. Pelien tekeminen saa usein jopa naurua kuulumaan luokasta. Lisätäkseen omaan peliinsä ominaisuuksia, joista ei tunnilla valmista esimerkkiä ole käyty, oppilaat esittävät opettajalle lisäkysymyksiä, keskustelevat ratkaisuista keskenään sekä hakevat esimerkiksi internetistä lisätietoa. Ennen kaikkea he harjoittelevat tyypillisesti enemmän kuin vain mitä pakollinen oppimäärä sisältää. Oppilaat voivat tehdä peliä, jonka juuri he kokevat mielenkiintoiseksi. Lisäksi pelien tekeminen soveltuu erinomaisesti niin yksilö- kuin ryhmätyöksi, ja projektin voi yhdistää muihin oppiaineisiin, kuten kuvien luomisen kuvataiteeseen tai matematiikan peliohjelmoinnissa tarvittavaan matematiikkaan.

Pelimoottoreita, jotka ovat työkaluja pelien tekemiseen, on olemassa monia, mutta harva niistä on tarkoitettu ohjelmoinnin opettamisessa käytettäväksi opetusmateriaaliksi. Ne eivät ota huomioon opettamisen erityishaasteita, eivätkä siksi usein sovellu kovin hyvin tähän tarkoitukseen. Tässä opinnäytetyössä pyritään toteuttamaan pelimoottori, joka on suunniteltu käytettäväksi opetusmateriaalina sekä luokissa opettajan ohjeistuksella että oppilaan itsenäisen opiskelun apuna.

2 OPETUSMATERIAALI

Tässä luvussa käsitellään olemassa olevien pelikirjastojen sekä –moottoreiden soveltuvuutta opetusmateriaaliksi. Niiden huonoja puolia tarkastellaan, ja epäkohtiin esitetään parannuksia.

2.1 Olemassa olevat pelimoottorit

Koulut voivat valita useista olemassa olevista pelimoottoreista, maksuttomista tai maksullisista, jonkin opetuskäyttöön. Riippuen rakennetuista toiminnallisuuksista sekä moottorin rajapinnan suunnittelusta, ohjelmoijan pelintekokokemus moottorista toiseen voi olla hyvinkin erilainen. Suuret pelimoottorit tarjoavat laajoja ominaisuuksia, mutta usein niissä on monia puutteita opetuskäytössä, sillä ne on suunniteltu mahdollisimman tehokasta ammattimaista pelinkehitystä, ei opettamista varten. Esimerkiksi oppilaan kysymykseen ”Kuinka tiedän, onko näppäimistön palautus-nappi painettu alas?” vastaaminen voi vaatia yllättävän monivaiheisen operaation, sillä laajojen toiminnallisuuksien tukeminen on johtanut rajapinnan monimutkaisuuteen.

Opettamisen kannalta ongelmallista on myös, että jotkin operaatiot on tehty liian helpoiksi tai automatisoiduiksi, mikä estää oppilasta miettimästä ongelmaa ja keksimästä siihen oman ratkaisunsa. Esimerkiksi kuvan lisääminen peliin tapahtuu tyypillisesti yhtä editori-ohjelman nappia painamalla. Tällöin oppilas ei saa tilaisuutta miettiä, mitä tapahtuu, jos kuvatiedostoa ei löydykään, milloin kuva ladataan levyltä, milloin kuvan varaama muisti vapautetaan, milloin kuva piirretään ruudulle ja niin edelleen.

Opetuskäyttöön tarkoitetun pelimoottorin rajapinnan tulee olla suunniteltu siten, että oppilaat voivat suorittaa yksinkertaisia operaatioita nopeasti ja helposti, ilman tarpeettoman monimutkaisia systeemejä. Sen on lisäksi vältettävä liikaa au-

tomaatiota, jotta oppilaat suunnittelevat ratkaisuja ongelmiin, joihin heidän kykynsä riittävät.

2.2 Asennus

Lähes jokaisen pelimoottorin käyttäminen vaatii asennuksen. Ohjelmien asentaminen usein hankaloittaa opetusta, sillä tyypillisesti vain koulujen IT-tukihenkilöt voivat asentaa ohjelmistoja tietokoneille. Mikäli nämä henkilöt eivät ole tavoitettavissa, opetus pysähtyy. Osa pelimoottoreista ei vaadi asennusta, mutta niiden kääntäminen ja konfigurointi ei-standardien työkalujen vaatimisen takia ei ole aloittelevalle ohjelmoijalle helppoa. Lisäksi mikäli oppilaille annetaan ohjelmointiläksyjä kotiin tai oppilas haluaa jatkaa ohjelmointia kotona, on asennus tai konfigurointi tehtävä myös siellä. Opetusmateriaalina toimivan pelimoottorin asennuksen tulisi siis olla mahdollisimman yksinkertainen ja minimaalinen.

Viimeinen asennusvaihe tulee eteen pelin pelaajalle. Oman pelin tekemisen hauskimpia hetkiä on antaa peli ystäville tai luokkatovereille pelattavaksi. Moni pelimoottori tekee tästä kuitenkin vaikeaa, mikäli pelimoottorilla tehty peli vaatii tiettyjen komponenttien olemassa olon tietokoneella. Aloittelevan ohjelmoijan on vaikea varmistaa näiden komponenttien löytyminen pelaajan tietokoneelta ja vielä vaikeampi korjata tilanne, mikäli komponentti puuttuu. Ihanteellinen pelimoottori ei vaadi pelaajalta minkäänlaista asennusta pelin pelaamiseksi, tai hoitaa asennuksen automaattisesti, ja mikäli pelimoottori kohtaa toiminnan estävän virheen, se ilmoittaa virheestä pelaajalle ja tarjoaa ohjeistusta virheen korjaamiseksi.

2.3 Teknologia

Silloin tällöin oppilaita kiinnostaa, kuinka pelimoottori on rakennettu ja kuinka se toimii. Mikäli pelimoottorin lähdekoodi ei ole avoimesti luettavissa tai sitä ei ole

kirjoitettu siten, että vasta-alkava ohjelmoija voi ymmärtää koodin, oppilaan oppimismahdollisuudet pelimoottorin teknisen toiminnan suhteen ovat hyvin rajalliset.

Usein ohjelmakoodin ymmärtämiseksi on hyvin olennaista voida kääntää koodi sekä suorittaa sitä virheenjäljittäjän avulla nähdäkseen, mitä koodi suorituksen aikana tekee. Usein koodin kääntäminen vaatii tietynlaisen käännösympäristön, joka täyttää pelimoottorin koodin käännösriippuvuudet. Monen pelimoottorin kääntämiseksi tämän ympäristön luominen on vaikeaa tai vähintäänkin vaivalloista kokeneillekin ohjelmoijille aloittelevista ohjelmoijista puhumattakaan.

3 PELIMOOTTORIOHJELMOINTI

Pelimoottorin ohjelmointia aloitettaessa on oltava selvillä, mistä tehtävistä pelimoottorin tulee suoriutua. Lisäksi on valittava rajapinnat, joita koodi voi käyttää, sekä päätettävä käytettävä ohjelmointikieli.

3.1 Pelimoottori

Pelimoottori on ohjelmointikirjasto, joka rakentaa käyttöjärjestelmän rajapintojen päälle oman rajapinnan. Tämä rajapinta sisältää peliohjelmoijalle hyödyllisiä toiminnallisuuksia pelien luomiseen. Moottorin on oltava suorituskykyinen, sillä tyypillisesti moottorin on piirrettävä monitorille kuva pelistä useita kymmeniä kertoja sekunnissa sekä suoriuduttava monista muista laskennallisesti vaativista tehtävistä. Pelimoottorilta vaaditaan myös äärimmäistä vakautta, koska pelejä pelataan viihteen vuoksi, ja pelin jatkuva uudelleenkäynnistäminen on turhauttavaa. (Gregory 2014, 11, 33.)

Tyypillisesti pelimoottorin rajapinnan kautta ohjelmoija voi reagoida hiiren liikkeeseen, näppäimistön painalluksiin, luoda ikkunan, ladata massamuistista kuvia, piirtää kuvia sekä geometriaa monitorin ruudulle sekä soittaa kaiuttimista ääniä sekä musiikkia. Moottori tarjoaa yleensä myös matematiikkafunktioita, joilla pelin logiikkaa voi ohjelmoida. (Gregory 2014, 11, 33.)

3.2 Ohjelmointikieli ja rajapinnat

Tämän opinnäytetyön pelimoottori on toteutettu C-ohjelmointikielellä, joka on usein pelimoottoreissa käytetty kieli, sillä se antaa edellytykset toteuttaa pelimoottorilta tyypillisesti vaaditut ominaisuudet. Käyttöjärjestelmän rajapinta on usein toteutettu C-kielellä, joten sitä on helppo käyttää moottorista. Kieli kääntyy

suoritinkomennoiksi ja sallii muistin manipuloinnin, mahdollistaen korkean suorituskyvyn, mikä on tärkeää sulavan pelikokemuksen saavuttamiseksi.

Pelimoottori rakentuu käyttöjärjestelmän rajapintojen päälle. Osa niistä on käyttöjärjestelmäkohtaisia ja osa standardisoituja. Esimerkiksi ikkunan luonti on täysin erilainen tapahtuma Windows- ja Linux-käyttöjärjestelmissä, mutta OpenGL sekä C-kirjasto löytyvät kummastakin. Käyttäen näitä rajapintoja, jotka tyypillisesti ovat hankalia syystä tai toisesta, pelimoottori toteuttaa rajapinnan, jota käyttäen pelien ohjelmointi on huomattavasti helpompaa. Lisäksi pelimoottorin rajapinta käsittelee abstraktimpia konsepteja kuin käyttöjärjestelmän rajapinnat. Esimerkiksi pelimoottori voi toteuttaa funktion, jolla käyttäjä voi luoda tekstuurin kuvatiedostosta ja funktion, jolla tekstuurin voi piirtää haluttuun kohtaan ruudulle. Pelimoottorin tehtävänä on lukea tiedosto massamuistista käyttömuistiin, purkaa kuvatiedoston pakkausformaatti, ladata tiedosto näytönohjaimen muistiin, konfiguroida näytönohjain käyttämään kuvaa ja viimein antaa piirtokomento, jolla oikeat pikselit monitorista väritetään. (Gregory 2014, 33.)

4 DIGITAALINEN GRAFIIKKA

Pelimoottorin grafiikoiden ohjelmointi vaatii digitaalisen grafiikan periaatteiden tuntemisen. Tässä luvussa esitellään sen perusteet. Lisäksi tekstuurit eli digitaaliset kuvat sekä fontit eli kirjasimet käsitellään.

4.1 Värit

Digitaalinen RGB väri koostuu punaisesta, vihreästä sekä sinisestä värikanavasta. Värille varattu bittimäärä eli syvyys määrittää, kuinka montaa eri väriä kanavien yhdistelmällä pystytään kuvaamaan. Esimerkiksi 24-bittisessä värissä on kahdeksan bittiä kutakin värikanavaa kohden. Siten 24-bittinen väri voi edustaa noin kuuttatoista miljoonaa eri väriä. Mikäli kullakin kanavalle on varattu 8 bittiä, voi kanava saada arvon nollan ja 255 välillä. Esimerkiksi liila väri on yhdistelmä punaista ja sinistä, joten kanavien arvot järjestyksessä ovat (255, 0, 255). Väriä voidaan tummentaa valitsemalla arvoja lähempää nollaa, kuten (100, 0, 100), sillä kaikkien kanavien ollessa nolla väri on musta. Väri on valkoinen, jos värikanavien arvot ovat värisyvyyden maksimiarvo. Usein laskutoimitusten helpottamiseksi väriä käsitellään normalisoidussa tilassa, jolloin kanavan pienin arvo on nolla ja suurin 1. Väriä tallennettaessa puskuriin tai viimeistään monitorille lähetettäessä se kvantisoidaan takaisin värisyvyyden tarjoamalle välille.

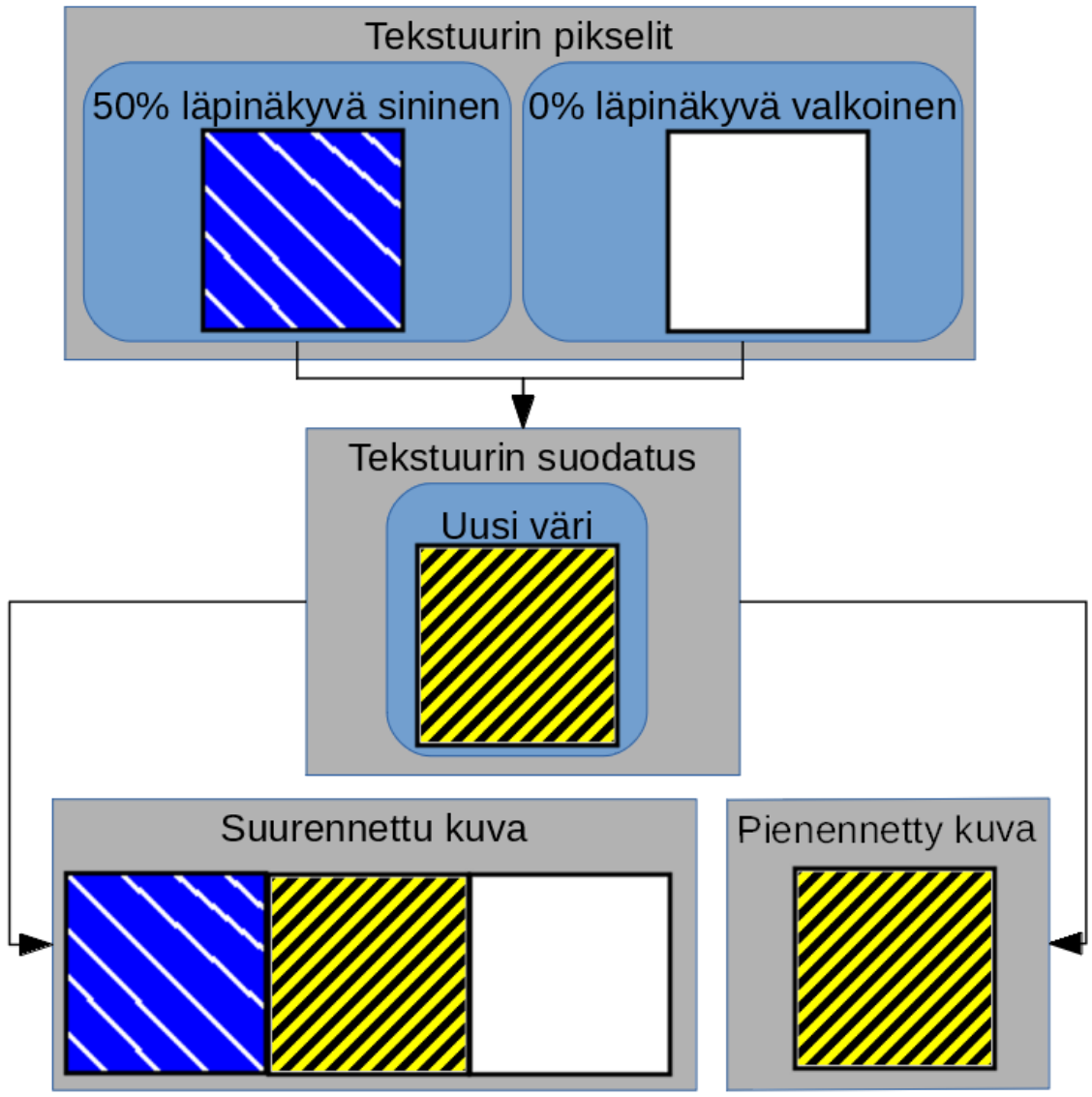
4.2 Tekstuurit

Pelissä piirrettäville artistin luomille kuville eli tekstuureille on useita erilaisia tiedostoformaatteja. Esimerkiksi Bitmap- eli BMP-formaatti tallentaa RGB-kanavien arvot sellaisenaan tiedostoon. Portable Network Graphics eli PNG analysoi kuvan toistuvuuksia ja näiden perusteella tallentaa vain kuvan häviöttömästi uudelleenluomiseksi tarvittavan tiedon, pienentäen tarvittavan tiedoston kokoa. PNG

tukee RGB-kanavien lisäksi alfakanavaa. Kolmas yleinen formaatti on Joint Photographic Experts Group eli JPEG. Myös JPEG pienentää tarvittavan tiedoston kokoa analysoimalla kuvan toistuvuuksia. Erona PNG-formaattiin on, että hyvin toisiaan lähellä olevat värisävyt yhdistetään yhdeksi väriksi, lisäten kuvan toistuvuuksia. Siten JPEG-kuvat vievät vähemmän muistia kuin PNG-kuvat, mutta kuvan laatu on vastaavasti myös huonompi. Kuvien muistivaatimuksen pienentämistä analyysin avulla kutsutaan pakkaukseksi. RGB-arvojen palauttamista pakatusta formaatista yksinkertaisiksi numeroiksi kutsutaan purkamiseksi. Purkaminen on suoritettava, jotta pakattu alkuperäinen RGB arvo saadaan palautettua koodin käyttöön.

Näytönohjaimille erityisesti suunniteltuja häviöllisiä kuvaformaatteja on myös olemassa. Nämä formaatit tukevat huomattavia tiedostokoon pienentämisen ominaisuuksia. Lisäksi näytönohjaimille tarkoitettuja formaatteja ei tarvitse purkaa RGB-arvoiksi ennen niiden käyttämistä, vaan näytönohjain suorittaa purkamisen lennosta hyvin nopeasti juuri ennen pikselin väriarvon lukemista. Esimerkkinä tällaisistä formaateista on formaattiperhe DirectDraw Surface eli DDS.

Artistin tekemät kuvat ovat aina tietyn kokoisia. Usein alkuperäisen kuvan koko on eri kuin pelintekijän kullakin hetkellä ruudulle piirtämä kuva. Esimerkiksi mikäli kuva on 256 pikseliä leveä ja 256 korkea, mutta ohjelmoija antaa piirtokomennon täyttää 512 kertaa 512 pikselin kokoinen neliö, millä väreillä neliö tulee täyttää? Halutun neliön ollessa suurempi tai pienempi kuin alkuperäinen suoritetaan tekstuurin suodatusalgoritmi (kuva 1). Kuvaa suurennetaan esimerkiksi siten, että pikseleiden, joille ei ole vastinetta alkuperäisessä kuvassa, väri otetaan sekoittamalla kahden vierekkäisen pikselin värit keskenään. Jos haluttu neliö on pienempi kuin alkuperäinen kuva, voidaan useampi pikseli yhdistää yhdeksi sekoittamalla niiden värejä keskenään. Uusien pikseleiden luomista tai pikseleiden yhdistämistä sekoittamalla vierekkäisten pikselien värejä keskenään kutsutaan lineaariseksi tekstuurinsuodattamiseksi. Lähimmäisen pikselin suodatus tarkoittaa, että pikselin väriksi valitaan lähimmäisen pikselin väri useamman vierekkäisen värin sekoittamisen sijaan.



Kuva 1. Tekstuurinsuodatustä käytetään, kun tekstuurin pikseleistä muodostuvaa kuvaa suurempi tai pienempi kuva piirretään.

4.3 Fontit

Tekstin piirtämiseen on useita eri lähestymistapoja. Eräs metodi perustuu tekstuuriin, johon on piirretty jokainen fontin tarjoama merkki. Tämä tekstuuri piirretään usein TrueType-fonttitiedoston sisältämän tiedon perusteella. Lisäksi jokaista merkkiä kohden tallennetaan informaatiota, joka kertoo, missä kohtaa kuvaa

merkin pikselit sijaitsevat sekä kuinka paljon väliä kirjaimen ympärille tulee jättää. Tekstiä piirrettäessä muodostetaan neliö jokaista piirrettävää tekstin merkkiä kohden. Ensimmäinen neliö piirretään haluttuun tekstin alkamiskohtaan, ja tämän jälkeen piirtokohtaa siirretään merkin leveyden sekä merkin tarvitsevan välin verran. Neliöihin sisällytetään myös tieto siitä, mistä kohtaa tekstuuria neliön on haettava kuva merkille. Lopulta neliöt piirretään, täyttäen pikseleillä, joihin ne viittaavat tekstuurissa.

4.4 Näytönohjaimet ja OpenGL

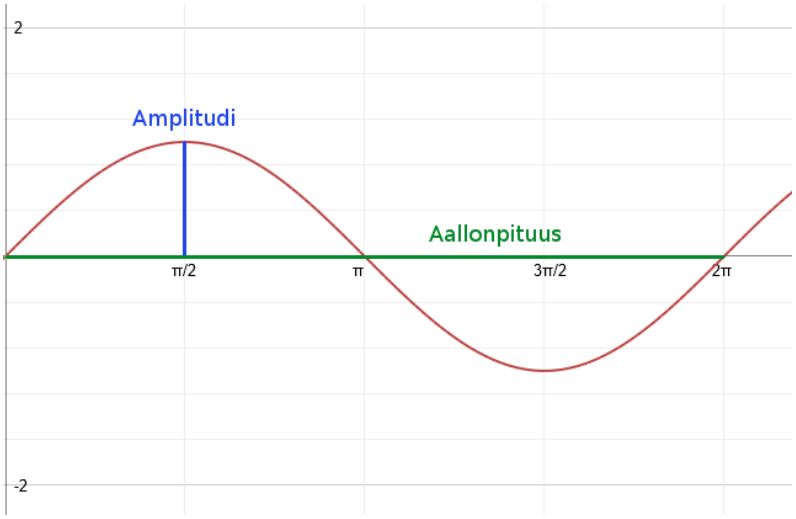
Digitaalisen kuvan muodostaminen monitorin ruudulle monta kymmentä kertaa sekunnissa vaatii hurjan määrän laskutoimituksia joka sekunti. Kuvan luomiseksi suoritetaan monia algoritmeja jokaiselle pikselille, joita useimmissa monitoreissa on miljoonia. Koska näin suuret tietokoneilta vaadittavat grafiikkaan liittyvät tehtävät ovat yleisiä, useimmissa tietokoneissa on asennettuna näytönohjain, joka sisältää grafiikkasuorittimen sekä grafiikalle varattua muistia. Grafiikkasuoritin on erityisesti suunniteltu suorittamaan kuvien piirtämiseen tarkoitettuja laskuoperaatioita ja suorittaa niitä siten huomattavasti nopeampaa kuin tietokoneen kuskussuoritin. Näytönohjaimia on valtavasti erilaisia, eikä niiden suoritinkomentoja ole standardoitu, vaan ne vaihtelevat valmistajasta sekä mallista riippuen. OpenGL-standardi on rajapinta, joka mahdollistaa ohjelmoijan kirjoittaman koodin toimimisen kaikilla näytönohjaimilla, jotka tukevat tätä standardia. Ensimmäinen OpenGL-versio 1.0 julkaistiin 1992, minkä jälkeen standardi on saanut monia uusia versioita näytönohjaimien ominaisuuksien kehittyessä eteenpäin. OpenGL-rajapinnan toteuttavia näytönohjainajureita on useille käyttöjärjestelmille, kuten Windowsille sekä Linuxille.

OpenGL sisältää monia funktioita, jotka sallivat näytönohjaimen toiminnallisuuksien käyttämisen. Tyypillisiä operaatioita ovat OpenGL Shading Language eli GLSL-ohjelmakoodin lataaminen näytönohjaimelle, tekstuurien sekä geometrian siirtäminen näytönohjaimen muistiin, kolmioiden rasterisointi eli piirtäminen ku-

vapuskuriin, tekstuurien suodattaminen ja laskutoimituksien suorittaminen GLSL-koodilla sekä värien alfasekoittaminen. (Sellers, Wright & Haemel, 4–11.)

5 DIGITAALINEN ÄÄNI

Ääni koostuu väliaineessa etenevistä paineaalloista, joiden päätyminen korvaan aiheuttaa äänen aistimisen. Ääniaallon amplitudi määrittää kuinka paljon väliainetta, kuten ilmaa, liikkuu aallon mukana ja siten kuinka kovaa äänen kuulemme (kuva 2). Ääniaallon taajuus kertoo, kuinka monta aaltoa äänen lähteestä säteilee per sekunti ja aiheuttaa äänen sävelkorkeuden aistimuksen. Tyypillinen ihminen kuulee ääniä, joiden taajuus on 20–20 000 hertsiä eli ääniaaltoa sekunnissa. Digitaalinen ääni koostuu siten informaatiosta, joka kuvailee ääniaallon amplitudin sekä taajuuden tietyn ajan välein. Kaiutin vastaanottaa tietokoneesta digitaalista informaatiota, joka kuvaa, kuinka voimakkaasti ja nopeasti kaiuttimen kalvon tulee värähdellä luodakseen ilmaan paineaaltoja, jotka kuuntelijan korvaan saapuessaan toistavat nauhoitetun äänen aistimuksen. (LaMothe 2002, 600–606.)



Kuva 2. Siniääniaallon amplitudi ja aallonpituus kuvaajassa.

Digitaalisen äänen kuvaama informaatio sisältää korkeinta esiintyvää äänitaajutta kohden vähintään kaksi näytteenottopistettä, jotta pisteiden tiheys riittää määrittämään, mistä kohtaa ääniaaltoa piste on otettu. Tämä tunnetaan Shannonin teoreemana. Teoreeman periaatteena on, että jokaisen ääniaallon voi jakaa siniaallon muotoisiksi ääniaalloiksi, ja tarvitaan kaksi pistettä kertomaan, onko arvo

siniaallon nousevalla vai laskevalla harjalla. Koska 20 kilohertsiä on tyypillisesti korkein ihmisen havaitsema taajuus, korkealaatuisen ääni-informaation näytteenottotaajuuden on oltava vähintään 40 kilohertsiä. (LaMothe 2002, 600–606.)

Digitaalista ääntä kuvaavalla informaatiolla on myös näytteenottotarkkuus. Mikäli näytteenottopisteen kertomalle amplitudille on varattu 8 bittiä muistia, pystytään näillä biteillä erottamaan kaksi potenssiin kahdeksan eli 256 eri arvoa. (LaMothe 2002, 600–606.)

Digitaalisella äänellä on myös tietyn monta kanavaa. Yksi- eli monokanavainen ääni kuuluu kaikista kaiuttimista samalla äänenvoimakkuudella. Kaksi- eli stereokanavaisessa äänessä panorointi jakaa äänenvoimakkuuden vasemman sekä oikean kaiuttimen kesken, luoden tilaäänen vaikutelman, jossa pelihahmon vasemmalta kuuluva ääni kuuluu pelaajan vasemmalta. Tämä auttaa pelaajaa eläytymään peliin. (LaMothe 2002, 600–606.)

Tyypillisesti äänet sekä musiikki koostuvat useista toisiinsa miksatuista eli sekoitetuista ääniaalloista. Tätä sekoitusta kutsutaan äänen spektriksi. (LaMothe 2002, 600–606.)

6 TOTEUTUS

Tässä luvussa käsitellään pelimoottorin toteutusta aiemmin esiteltyjen periaatteiden perusteella. Käyttöjärjestelmä sekä ohjelmointikielen valinta perustellaan, ja moottorin rakennetta kuvaillaan. Tämän jälkeen moduulien toimintaan perehdytään läheisemmin yksi moduuli kerrallaan. Niiden pääpiirteinen rakenne sekä toiminta selitetään.

6.1 Käyttöjärjestelmä ja ohjelmointikieli

Tämän opinnäytetyön pelimoottori tarjoaa peliohjelmoijalle neljä käyttöjärjestelmän rajapintojen päälle rakentuvaa moduulia, jotka auttavat oppilasta ohjelmoimaan monia erilaisia pelejä. Nämä C-kielellä ohjelmoidut moduulit käyttävät käyttöjärjestelmän rajapinnoista vain jokaisen Windows 7- sekä myöhempien versioiden tarjoamia toiminnallisuuksia, tai upottavat hyödynnettyjen kirjastojen koodin suoraan pelimoottorin lähdekoodiin. Siten pelin ohjelmointi vaatii ainoastaan C-kielen kääntäjä-ohjelman sekä Windowsin ohjelmistokehityskirjastot. Kummatkin ovat ilmaisia ja helposti asennettavissa.

Tällä pelimoottorilla luodun pelin pelaaminen ei vaadi mitään asennusta, sillä se käyttää käyttöjärjestelmästä vain rajapintoja, jotka ovat aina osa Windowsia ja koska C kääntyy suoritinkomennoiksi, joita suoritin voi suorittaa ilman erillistä tulkkiohjelmaa, toisin kuin esimerkiksi C#, JavaScript tai Java. Lisäksi koska moduulit voi myös helposti kääntää C-koodikirjastoksi, pelimoottoria voi käyttää lähes mistä tahansa opettajan tai oppilaan valitsemasta ohjelmointikielestä, sillä lähes jokainen kieli sisältää tavan kutsua C-funktioita.

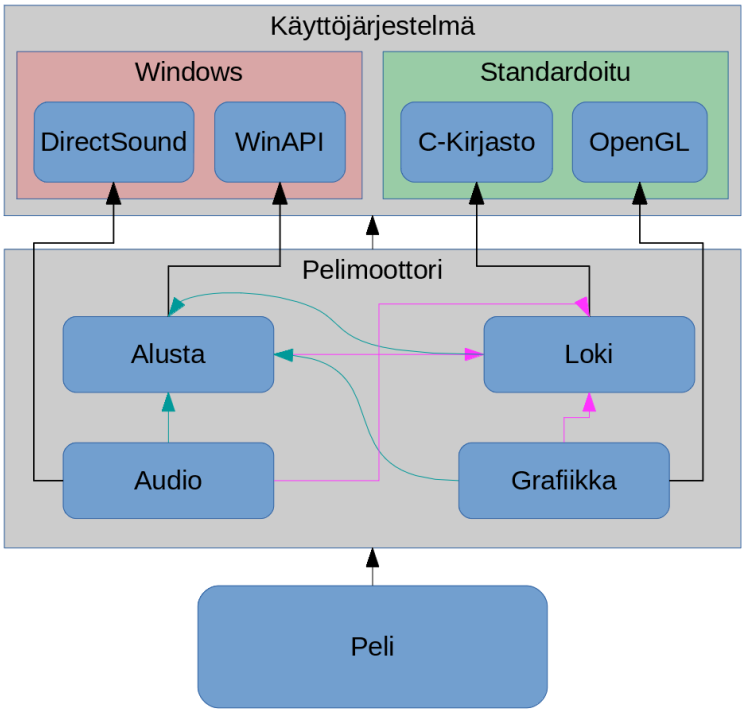
C on varsin pieni kieli, eli siinä on vain vähän käsitteitä ja avainsanoja. Sillä kirjoitettua koodia pystyy ymmärtämään vain vähäisen ohjelmointikokemuksen jälkeen, mikä tekee kielestä hyvin soveltuvan aloitteleville ohjelmoijille. Sen osaa-

minen on oppilaille myös aidosti hyödyllinen taito, sillä kieltä käytetään muun muassa roboteissa, sulautetuissa laitteissa, älypuhelimissa, käyttöjärjestelmissä, työpöytäsovelluksissa, palvelimissa, autoissa, lentokoneissa, raketeissa sekä tietenkin peleissä.

Pelimoottorin koodin kääntäminen lähdekoodista mahdollistaa koodin täyden suorituksen virheenjäljittäjällä. Siten oppilas voi askel askeleelta selvittää, kuinka pelimoottori on rakennettu ja kuinka se toimii. Koodi on lisäksi kirjoitettu yksinkertaisesti ja se sisältää erittäin paljon selventäviä ja ohjeistavia kommentteja, mahdollistaen vasta-alkajan lukea ja ymmärtää pelimoottorin kaikki koodi.

6.2 Rakenne

Moottori sisältää loki-, alusta-, grafiikka- ja audiomoduulin. Lisäksi pelimoottori sisältää vektori- sekä analyyttisen geometrian matematiikkakirjaston, jonka funktioita käyttäen pelin logiikkaa voi ohjelmoida. Moduulien tarjoama toiminnallisuus eroaa toisistaan suuresti, mutta niiden periaate on aina sama. Rajapinta on mahdollisimman yksinkertainen, jotta aloitteleva ohjelmoija pääsee helposti alkuun, mutta ei liian automatisoitu estääkseen oppilasta kohtaamasta mielenkiintoisia ongelmia, joiden ratkaisemiseen hänen kykynsä riittävät. Lisäksi moduulit tekevät kaikkensa tarjotakseen lokitiedostoon hyödyllistä tietoa, mikäli oppilaan tekemä virhe havaitaan. Moduulit toimivat mahdollisimman itsenäisinä yksikköinä, mutta käyttävät tarvittaessa toistensa toiminnallisuuksia (kuva 3). Esimerkiksi kaikki moduulit hyödyntävät lokimoduulia tallentaakseen viestejä hyödyllisistä tiedoista sekä mahdollisista havaituista virheistä.



Kuva 3. Pelimoottorin rakenne.

Pelimoottorin tarjoamaa toiminnallisuutta voi laajentaa luomalla uusia moduuleja. Maininnan arvoisia puutteita verrattuna ammattikäyttöön tarkoitettuihin pelimoottoreihin on muistinvaraajan, tiedostonhallinnan, animaation, tallennuksen sekä kenttäeditorin puuttuminen. Näiden systeemien toteuttaminen pelimoottoriin vaikeuttaisi C-kielen sekä standardin oppimista, sillä esimerkiksi sen sijaan, että oppilas oppisi kirjoittamaan sekä lukemaan tiedostoja, pelimoottorin tallennussysteemi tekisi tämän oppilaan puolesta.

Opettamiseen tarkoitettu pelimoottori innostaa oppilaita harjoittelemaan ja keksimään omia toteutuksia suunnitelmiinsa. Yksinkertainen ja kommentoitu pelimoottorikoodi mahdollistaa sen ymmärtämisen, mikä mahdollistaa ja kannustaa muutosten sekä laajennusten tekoon. Tämä kannustaa ja opettaa oppilaita uskomaan itseensä. Ensimmäinen muutos voi olla pieni, esimerkiksi luotavan lokitiedoston tiedostopolun muutos. Toinen muutos voi kenties olla jo suurempi, kuten uuden audiotiedosto- tai kuvaformaatin tukeminen. Innokkaimmat voivat jopa

tehdä pelimoottoriin uuden hyödyllisen moduulin ja jakaa sen luokkatovereidensa tai ystäviensä käytettäväksi.

6.3 Lokimoduuli

Lokimoduuli on toteutettavista moduuleista pienin sekä yksinkertaisin. Loki tarjoaa rajapinnan kirjoittaa lokiviestejä neljään tekstitiedostoon, jotka edustavat kolmea lokiviestikategoriaa. Virheet, varoitukset sekä huomautukset kirjoitetaan kukin omaan tekstitiedostoonsa. Lisäksi jokainen viesti kirjoitetaan neljänteen tekstitiedostoon, joka sisältää kaikki viestit kronologisessa järjestyksessä. Lokitiedostot auttavat oppilasta korjaamaan ohjelmointivirheitä, joita tulee niin aloitteleville kuin kokeneille ohjelmoijille.

Lokiviestin kirjoitusaika sekä funktio, joka lokiviestin kirjoitti, kirjoitetaan viestin otsikoksi. Tämä helpottaa peliohjelmoijaa korjaamaan pelimoottorin raportoimia virheitä, kuten esimerkiksi puuttuva kuvatiedosto. Mikäli kyseessä on virheen aiheuttama lokiviesti, jos käyttöjärjestelmä on havainnut virheen, lisätään tämä virhe osaksi lokiviestiä. Lisäksi ennen viestin kirjoitusta viestin merkkijonoa verrataan aiemmin kirjoitettuihin viesteihin. Mikäli sama viesti on jo kirjoitettu, lokiin ei kirjoiteta mitään. Pelimoottori suorittaa toimintonsa useita kertoja sekunnissa. Ilman identtisten viestien hylkäystä lokitiedosto täyttyisi nopeasti saman virheen raportoimisesta yhä uudestaan ja uudestaan.

Viestien vertailun nopeuttamiseksi ennen merkkijonovertailua lokiviestin tarkistetta verrataan mahdollisesti kirjoitettavan viestin tarkisteeseen. Näin suhteellisen raskaan merkkijonovertailun sijaan verrataan vain kahta numeroa keskenään. Tarkisteet luodaan tekstin merkkijonosta MurmurHash3-algoritmiä käyttäen. Tarkiste luo eri merkkijonoille eri numeron, kuitenkin joskus luoden eri merkkijonoille saman numeron. Mikäli vertailussa numerot ovat samat, verrataan lokiviestien

merkkijonoja keskenään varmisteen lisäksi. Näin varmistetaan, että merkkijonot todella ovat identtiset.

Lokimoduuli käyttää C-kirjaston toiminnallisuutta merkkijonojen kirjoittamiseksi tiedostoihin. Siten se on valmiina toimimaan ilman alustusfunktion käynnistystä ja on aina käytettävissä kaikille moduuleille. Lokimoduuli kutsuu vain yhtä funktiota toisesta moduulista. Saadakseen käyttöjärjestelmän ilmoittaman virheviestin osaksi virheen lokia lokimoduuli kutsuu alustamoduulista funktiota, joka luo tämän virheviestin.

6.4 Alustamoduuli

Alustamoduuli toteuttaa käsitteenä yksinkertaisia, mutta muun muassa historiallisista syistä johtuen käytännön toteutukseltaan monimutkaisia toimintoja. Se tarjoaa rajapinnan käyttöjärjestelmän pelien kannalta olennaisiin toiminnallisuuksiin. Tähän moduuli käyttää WinAPI-rajapintaa, joka on osa jokaista Windows-asennusta. Rajapinnasta käytetään vain toiminnallisuutta, joka on olemassa Windows 7 -versiosta lähtien. Alustamoduuli on ensimmäinen moduuli, joka pelintekijän tulee alustaa funktiokutsulla. Käynnistyessään moduuli kirjoittaa lokiin ongelmanratkaisutilanteissa hyödyllistä tietoa, kuten tietoa pelaajan suorittimesta, muistista, monitoreista sekä näytönohjaimista.

Tämän jälkeen moduuli luo OpenGL 3.0 -standardia tukevan ikkunan, jonka koko on pelintekijän määrittämä. Tehdäkseen tämän moduuli luo näkymättömän OpenGL 1.1 -ikkunan, mikä mahdollistaa OpenGL 3.0 -funktioiden latauksen. Tämä Windowsin vaatima outo tapa luoda OpenGL 3.0 -ikkuna on hyvä esimerkki historiallisten rajapintojen kummallisuuksista. Funktioiden latauksen jälkeen näkymätön ikkuna tuhotaan, ja OpenGL 3.0 -ikkuna luodaan. Ikkunalle luodaan kaksi 32-bittistä kuvapuskuria, joiden bitit on jaettu tasan punaisen, sinisen ja vihreän värikanavan sekä alfakanavan kesken. Identtisiä puskureita, joihin kuva piirretään, on kaksi, jotta grafiikkamoduulin piirtäessä kuvaa puskuriin monitori

voi lukea edellistä kuvaa häiriöttä toisesta puskurista näyttääkseen sen pelaajalle. Tätä kutsutaan kaksinkertaiseksi puskuroinniksi.

Ikkunan tukema erityisominaisuus on vertikaalinen synkronointi, joka estää näytönohjainta piirtämästä kuvapuskuriin, jota monitori lukee. Tämä estää ilmiön, jossa monitorin näyttämä kuva koostuu kahdesta eri pelimoottorin luomasta kuvasta näytönohjaimen kirjoittaessa kuvapuskuriin monitorin lukiessa sitä. Tyypillisesti näin käy, jos näytönohjain piirtää kuvia nopeampaa kuin monitori ehtii niitä pelaajalle näyttää. Kaksinkertainen puskurointi ei estä tätä, mikäli peli piirtää kuvan yhteen puskuriin ja aloittaa seuraavan kuvan piirtämisen toiseen puskuriin monitorin vielä piirtäessä toisen puskurin kuvaa. Vertikaalisen synkronoinnin huonona puolena on joissakin tilanteissa esiintyvä viive. Pelaaja usein haluaa itse valita, onko vertikaalinen synkronointi aktivoitu, joten pelimoottori tarjoaa mahdollisuuden kytkeä se pois päältä.

Usein pelitekijä haluaa pelin vievän monitorin koko tilan. Alustamoduuli tukee tätä ominaisuutta funktiolla, joka pelintekijän niin halutessa asettaa ikkunan kooksi monitorin koon sekä siirtää ikkunan monitorin vasempaan yläkulmaan.

Alustamoduuli tarjoaa pelintekijälle rajapinnan näppäimistön sekä hiiren hyödyntämiseen. Toteutettava pelimoottori muuttaa käyttöjärjestelmän tapahtumapohjaisen rajapinnan kyselypohjaiseksi. Tämän lisäksi moduuli tallentaa pelimoottorin edellisen pyörähdyksen aikaisen näppäimistön tilan. Siten pelintekijälle tarjotaan käyttöjärjestelmää huomattavasti yksinkertaisempi rajapinta, joka sallii hänen koska tahansa selvittää, onko näppäimistön tai hiiren nappi painettu sekä mikä sen tila oli aiemmin. Hiiren kursorin sijainti ruudulla on myös helposti saatavilla. Alustan oman kursorin voi myös muuttaa näkymättömäksi, mikä mahdollistaa peliin sopivamman kursorin piirtämisen ruudulle.

Tarkka ajan mittaaminen on tärkeää peliohjelmoinnissa. Alustamoduuli tarjoaa tähän kaksi funktiota, jotka palauttavat käyttöjärjestelmän korkeataajuisen laskurin luvun sekä laskurin taajuuden. Näiden lukujen perusteella ohjelmoija voi hyvin tarkasti laskea, kuinka paljon aikaa kahden mittaushetken välillä on kulunut.

Viimeinen alustamoduulin tehtävä on auttaa ohjelmoijaa selvittämään pelin teknisiä ongelmia. Tätä varten, jos peli kohtaa vakavan virheen ja kaatuu, alustamoduuli luo tiedoston, joka sisältää tarkkaa tietoa ohjelman tilasta. Tämän tiedoston perusteella pelintekijä usein löytää virheen syyn ja pystyy korjaamaan sen.

6.5 Grafiikkamoduuli

Pelintekijä alustaa grafiikkamoduulin funktiokutsulla. Alustuksen aikana moduuli luo näytönohjaimen muistiin pistepuskurin kuudesta pisteestä, jotka muodostavat kaksi kolmiota. Nämä kolmiot muodostavat kuvan jälkikäsittelyssä apuna käytettävän koko ruudun peittävän neliön. Pelimoottori varaa myös muistia pistepuskurille, joka koostuu useasta sadasta pisteestä muodostaen kolmioita ja jotka puolestaan muodostavat neliöitä. Pistepuskuri sisältää pisteiden sijainnit ja pisteiden värin. Moduuli luo myös kolme 32-bittistä RGBA-kuvapuskuria, joita käytetään apuna pelimoottorin luoman kuvan piirtämiseen. Kuvapuskureiden nimet ovat etu-, taka- sekä apukuvapuskuri. Moduuli piirtää grafiikkaa luomaansa takakuvapuskuriin, sillä näytönohjain pystyy käyttämään siihen luotavaa kuvaa tekstuurina uuden kuvan luomiseen, mikä on oleellista joidenkin erikoisefektien toteuttamiseksi. Alustamoduulin luomia kuvapuskureita ei voi käyttää tekstuureina, mutta ne ovat ainoat kuvapuskurit, joita monitori voi lukea. Siksi sekä alusta- että grafiikkamoduulin kuvapuskurit tarvitaan. Tämän jälkeen grafiikkamoduuli konfiguroi näytönohjaimen käyttämään ennakkokertolaskettua alfasekoittamista sekä konfiguroi näytönohjaimen muita ominaisuuksia. Viimeisenä näytönohjaimelle ladataan useita GLSL-ohjelmia, jotka suorittavat monia piirtämisessä tarvittavia algoritmeja ja laskutoimituksia.

Rajapinta, jonka grafiikkamoduuli tarjoaa pelintekijälle, sisältää funktioita kahdentyyppisten objektien luomiseen. Käyttäjä voi luoda kuvatiedostosta tekstuurin, jonka voi antaa parametriksi monelle piirtofunktiolle. Kuvatiedostojen purkamiseen pelimoottori käyttää funktiota stb_image-koodikirjastosta, jonka koodi on

upotettu pelimoottorin lähdekoodiin. Mikäli kuvatiedostosta luettavat RGBA-arvot eivät ole ennakkokertolaskettuja alfan suhteen, suorittaa moduuli tämän kertolaskun. Grafiikkamoduuli konfiguroi tekstuurit siten, että näytönohjain suorittaa tarvittaessa pikseleille lineaarisen suodatuksen sekä poistaa artistin silmän tekemän gammakorjauksen, tuoden RGBA-arvot lineaaritilaan.

Toinen luotava objekti on TrueType-tiedostosta luotava fontti, jota käytetään tekstin piirtämiseen. Tämän objektityypin tarvitsema kuva fontin merkeistä luodaan stb_truetype-kirjastolla, joka piirtää kuvan TrueType-fontin merkeistä. Objektin luomisen yhteydessä moduuli lataa piirtämisessä tarvittavan informaation näytönojaimen muistiin. Pelimoottorin havaitsemat virheet objekteja luotaessa kirjoitetaan lokiin, jotta oppilas voi havaita ja korjata ne.

Pelimoottorin rajapinta tarjoaa pelintekijälle useita piirtofunktioita. Ensimmäinen funktio täyttää ikkunan oppilaan haluamalla taustavärillä ja valmistaa pelimoottorin kuvan piirtämistä varten. Loput funktiot vaihtelevat tarvittavien parametrien suhteen. Käyttäjä voi piirtää esimerkiksi tasa- tai liukuvärjättyjä neliöitä, tekstuurilla täytettyjä neliöitä sekä tekstuurin tietyillä pikseleillä täytettyjä neliöitä. Funktiot hyväksyvät RGB-arvojen lisäksi alfa-arvon läpinäkyvyyden säätelemiseksi. Neliöiden lisäksi käyttäjä voi piirtää mitä tahansa muita muotoja tekstuurin alfakanavan avulla, asettaen pikselin RGBA-arvoksi 0, kun pikseliä ei haluta piirrettävän.

Grafiikkamoduulin vastaanottaessa piirtokomennon pelimoottori tarkistaa, että neliö on pelin ikkunan sisällä. Mikäli näin ei ole, funktio ei tee mitään. Seuraavaksi moduuli tarkistaa, mikä tekstuuri on sillä hetkellä merkittynä käytössä olevaksi. Myös ”ei tekstuuria”-lasketaan tekstuuriksi tässä yhteydessä. Jos käytössä oleva tekstuuri on eri kuin piirtokomennon parametrina oleva tekstuuri, pistepuskurin mahdollisesti sisältämistä neliöistä annetaan näytönohjaimelle piirtokomento. Oppilaan antaman piirtokomennon tekstuuri merkitään käytössä olevaksi, ja oppilaan piirtokomennon määrittämä neliö tallennetaan pistepuskuriin.

Neliöitä tallennetaan pistepuskuriin ja piirretään kerralla useampia johtuen siitä, kuinka näytönohjaimet toimivat. Pienen kolmiomäärän piirtäminen on suhteessa huomattavasti hitaampaa kuin suuren kolmiomäärän, joten pelimoottori käyttää pistepuskuria kerryttämään mahdollisimman monta kolmiota ennen piirtokomennon antamista. Piirtokomento annetaan puskurin täyttyessä, tekstuurin vaihtuessa tai kun oppilaan kutsumat funktiot, esimerkiksi kaikki jälkikäsittelyefektit, vaativat, että kaikkien edellisten funktioiden luoma kuva todella on piirrettynä eikä vain odota piirtämistä. Ennen piirtokomennon antamista moduuli kopioi pistepuskurin arvot muistista näytönohjaimen muistiin ja konfiguroi näytönohjaimen käyttämään pistepuskurin sisältämää geometriaa piirtämiseen.

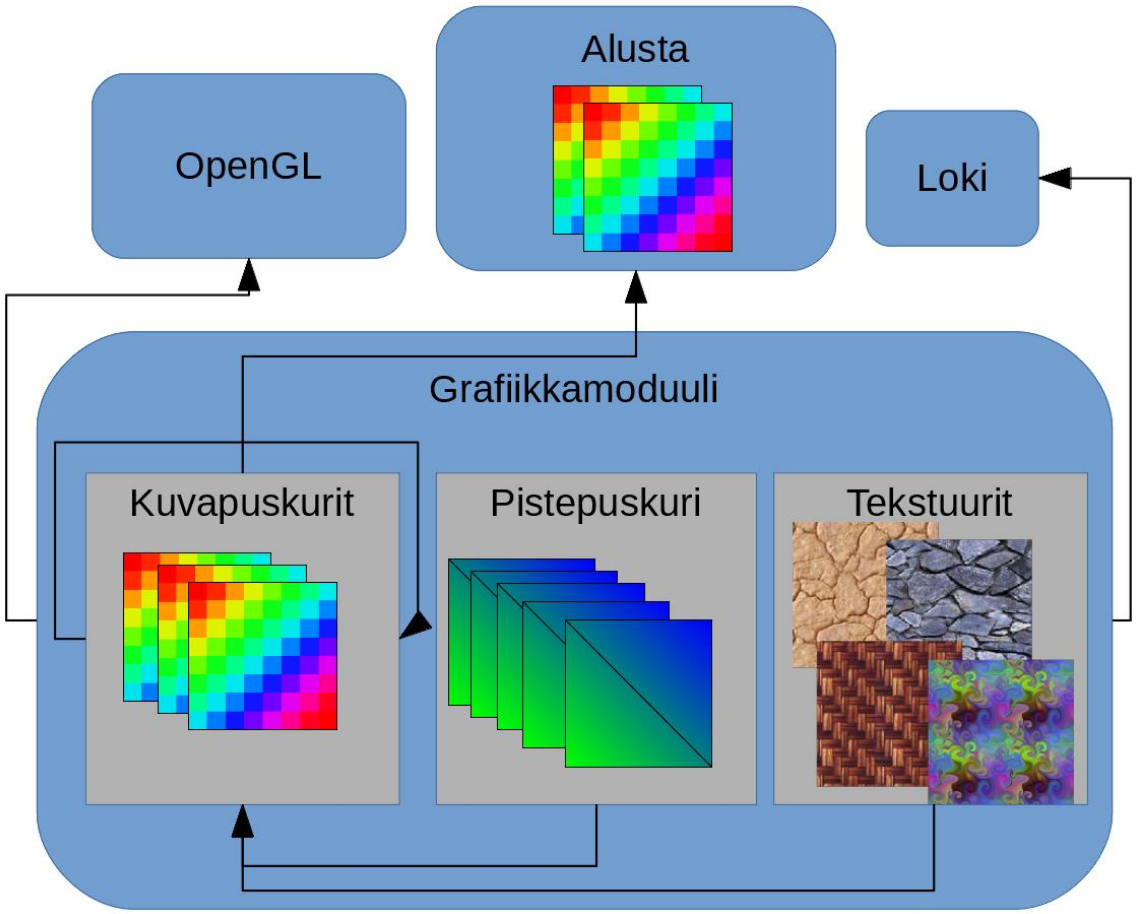
Pelimoottori suorittaa virheentarkistusta piirtokomennoille ja raportoi lokiin, mikäli se havaitsee virheellistä käytöstä. Peliohjelmoijan haluaman tekstin piirtäminen tapahtuu samalla tavalla pistepuskuriin kuin väritettyjen ja kuvallisten neliöiden piirtäminen, sillä teksti piirretään neliöinä, jotka täytetään fontille luodulla tekstuurilla.

Grafiikkamoduuli tarjoaa pelintekijälle laajan kirjon kuvan jälkikäsittelyefektejä, kuten mustavalkoisuus, sumu sekä pikselöityminen. Saadessaan oppilaalta komennon suorittaa kuvalle jälkikäsittelyefekti moduuli vaihtaa takakuvapuskurin sekä etukuvapuskurin nimet keskenään. Sitten näytönohjain piirtää koko peliruudun kokoisen neliön uuteen takakuvapuskuriin, käyttäen uuden etukuvapuskurin tekstuuria kuvana, jolle peliohjelmoijan kutsuman funktion jälkikäsittelyalgoritmi suoritetaan. Jotkut efektit käyttävät lisäksi apukuvapuskuria tarvittaessa. Takakuvapuskuri sisältää aina viimeisimmän pelimoottorin piirtämän kuvan, jonka käyttäjä on funktiokäskyillä luonut.

Kun käyttäjä on kutsunut kaikkia haluamiaan piirtofunktioita ja on valmis näyttämään pelin luoman kuvan monitorilla, kutsuu hän funktiota, joka siirtää kuvan monitorin piirrettäväksi. Funktio suorittaa kuvalle jälkikäsittelyefektinä gammakorjauksen, siirtäen kuvan RGB-arvot algoritmien ja laskutoimitusten tarvitsemasta lineaarisesta tilasta gammakorjattuun tilaan ja piirtää kuvan alustamoduulin taka-

kuvapuskuriin. Viimeiseksi grafiikkamoduuli kutsuu alustamoduulista funktiota, joka vaihtaa ikkunan taka- ja etukuvapuskurit keskenään. Piirtäessään kuvan seuraavan kerran näytölle monitorin gammakäyrä mitätöi tehdyn gammakorjauksen, johtaen lopulta monitorilla näkyvään laadukkaaseen kuvaan.

Kuva 4 kuvaa grafiikkamoduulin rakennetta. Pistepuskurit määrittävät kuvapuskuriin piirrettävän geometrian, joka mahdollisesti saa värinsä tekstuurista. Kuvapuskuri piirretään toiseen kuvapuskuriin erikoisefektejä piirrettäessä. Lopuksi kuvapuskuri piirretään alustamoduulin ikkunan kuvapuskuriin, josta se voidaan siirtää monitorille.



Kuva 4. Grafiikkamoduulin rakenne.

6.6 Audiomoduuli

DirectSound on Windowsin rajapinta, joka tarjoaa audiomikserin. Windows 7 sekä uudemmat Windowsit tukevat tätä rajapintaa ilman käyttäjältä vaadittavaa erityisasennusta. Käyttäen tätä audiomikseriä audiomoduuli tarjoaa käyttäjälle mahdollisuuden toistaa kaiuttimista useita ääniä samaan aikaan ja jopa muuttaa niiden ominaisuuksia pelin aikana. Oppilaan käynnistäessä moduulin funktiokutsulla moduuli alustaa DirectSound-rajapinnan ensisijaisen audiopuskurin, ja siten audiomikserin, toimimaan 16-bittisenä, 44 100 hertsin taajuudella stereona. Tämä on CD-tasoinen audioformaatti, joka kuulostaa ihmiskorvaan laadukkaalta.

Moduuli ei ole toiminnaltaan kovin monimutkainen, sillä audio-ohjelmoinnin vaikein osuus on suorituskykyisen audiomikserin tekeminen. DirectSound tarjoaa tämän, joten moduulin pääasiallinen työ on lukea audiotiedoston formaatin informaatio, ladata se rajapinnan ymmärtämään muotoon ja konfiguroida rajapinta toimimaan käyttäjän toivomalla tavalla. Moduulin pelintekijälle tarjoama rajapinta on suunniteltu yksikertaisemmaksi kuin DirectSound, jotta ohjelmoijan on helppo lisätä ääniä peliinsä.

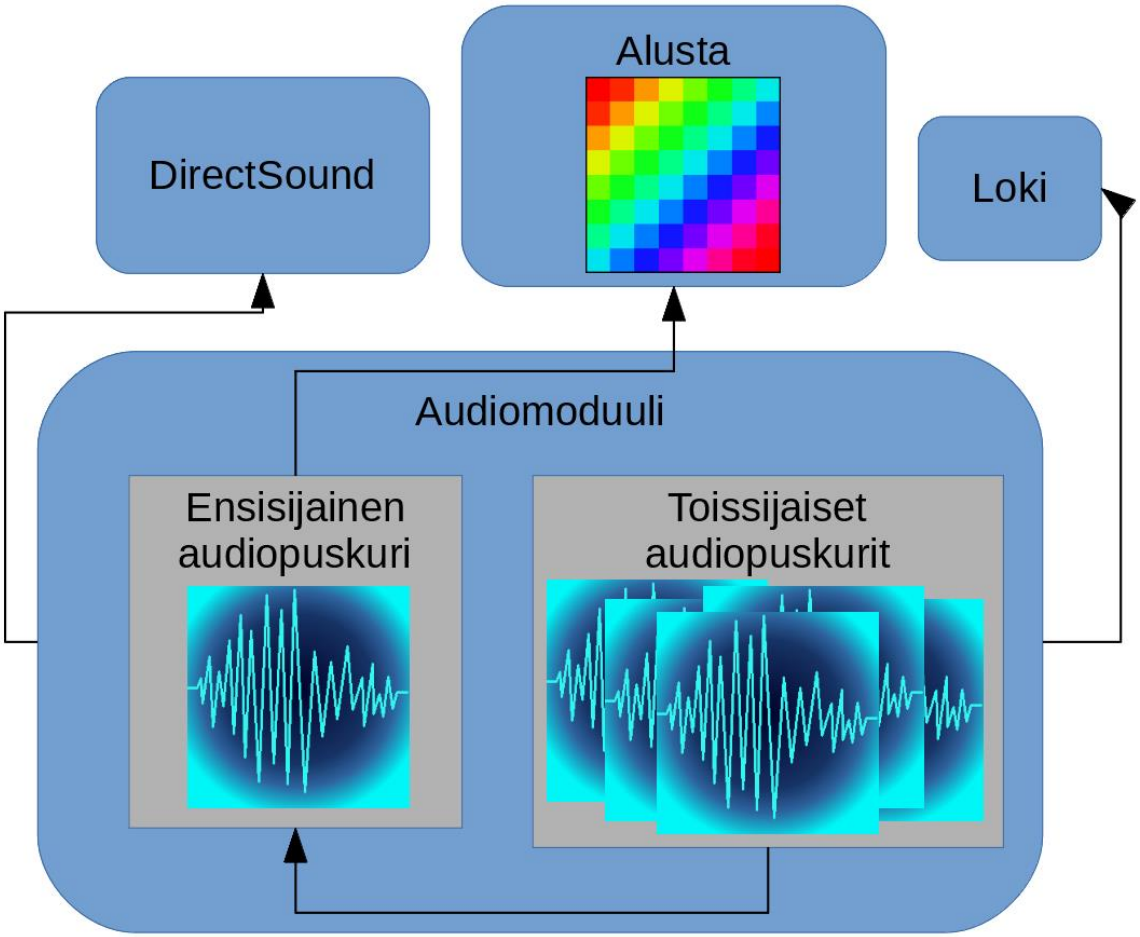
Käyttäjä pystyy luomaan audio-objekteja audiotiedostoista. Moduuli luo toissijaisen DirectSound-audiopuskurin tiedoston formaatin tarjoamien tietojen perusteella. Puskuri on luonteeltaan kehämäinen. Puskurin tietoa toistettaessa, kun puskurin loppu saavutetaan, siirtyy lukupaikka takaisin informaation alkuun. Mikäli audio-objektin luominen mistä tahansa syystä epäonnistuu, lokiin kirjoitetaan siitä virheviesti lisätietoineen, mikä auttaa pelintekijää korjaamaan virheet, kuten puuttuvat tai väärään formaattiin tallennetut tiedostot.

Ohjelmoija voi milloin tahansa funktiokutsulla saada tietoa audio-objektista, kuten sen formaatin, toistetaanko ääntä parhaillaan ja missä kohtaa ääntä toistetaan. Moduuli muuttaa kaiken tämän tiedon intuitiivisempaan muotoon kuin millaisena DirectSound sitä käsittelee. Esimerkiksi DirectSound palauttaa äänen toistokohdan tavumääränä informaation alusta laskettuna. Moduuli laskee tämän ja for-

maatin tietojen perusteella senhetkisen toistokohdan sekunneissa ja palauttaa sen. Siten pelintekijälle palautetaan intuitiivisempaa ja pelin kannalta olennaisempaa tietoa, esimerkiksi, että ääntä toistetaan tällä hetkellä äänitiedoston kohdasta 3,5 sekuntia.

Ohjelmoija voi funktiokutsulla aloittaa audio-objektin toiston, pysäyttää sen, kelata ajassa tiettyyn kohtaan, muuttaa sen äänenvoimakkuutta sekä tuhota objektin vapauttaakseen sen varaaman muistin. Kaiken audiomoduulin miksaaman äänen voimakkuutta voi myös hallita pää-äänenvoimakkuusasetuksella. Moduuli kääntää DirectSound-rajapinnan käyttämän äänen vaimentamisen käsitteen oppilaalle helpommin ymmärrettävämmäksi käsitteeksi, prosenttiosuudeksi alkuperäisestä äänenvoimakkuudesta. Pelintekijä voi muuttaa toistettavan äänen taajuutta, tehden siitä matalamman tai korkeamman mielenkiintoisten äänitehosteiden saavuttamiseksi. Moduuli sallii myös panoroinnin asettamisen. Jokaisessa näistä funktioista moduuli suorittaa ensin audio-objektin tarkistuksen, mahdollisesti havaiten että käyttäjä ei ole onnistuneesti luonut audio-objektia, on jo tuhonnut sen tai sen sisältämä informaatio on korruptoitunut kenties oppilaan vahingossa aiheuttaman muistin ylikirjoittamisen takia.

Kuva 5 kuvaa audiomoduulin rakennetta. Toissijaiset audiopuskurit miksataan ensisijaiseen audiopuskuriin peliohjelmoijan asettamien asetusten mukaisesti. Käyttöjärjestelmä käyttää alustamoduulin ikkunaa apuna toistaessaan ääntä kaiuttimista määrittääkseen, kuinka toiston tulee tapahtua.



Kuva 5. Audiomoduulin rakenne.

7 POHDINTA

Tässä luvussa pohditaan toteutetun pelimoottorin onnistumista. Erityistä huomiota kiinnitetään suunniteltuun rajapintaan, jota oppilas käyttää. Pelimoottorin ja sillä luodun pelin käyttöönottoprosessi, etenkin asennusvaiheen puuttuminen, kuvaillaan ja sen toteuttamiseksi vaadittuja askeleita eritellään. Lopuksi valittua ohjelmointikieltä punnitaan toista mahdollista valintaa vastaan.

7.1 Rajapinta

Opettamiseen tarkoitetun pelimoottorin toteuttaminen oli haastavaa, mutta lopputulos on prototyyppinä onnistunut. Pelimoottorin käyttäminen peliohjelmoijalle on hyvin helppoa rajapinnan yksinkertaisuuden takia. Sen suunnitteluun ja eri lähestymistapojen kokeiluun kului yllättävän paljon aikaa, koska niitä oli kokeiltava käytännössä. Se oli kuitenkin vaivan arvoista, koska sopiva rajapinta opettaa oppilasta ohjelmoimaan ja samalla ajattelemaan ongelmanratkaisua.

Ongelmanratkaisutaidon puutetta havainnollistavat monet Unity3D-pelimoottorin keskustelupalstojen kysymykset. Esimerkiksi kysymys ”Kuinka teen hissin?” kysyy yksinkertaisen kysymyksen. Ohjelmointia paremmin hahmottavat oppilaat osaavat muodostaa kysymyksen eri muodossa. ”Kuinka lisään peliin laatikon, joka ei päästä päällä olevaa kappaletta tippumaan sen läpi, ja kuinka saan kappaleen liikkumaan ylöspäin?” Näihin kysymyksiin löytyy suorat vastaukset Unity3D -dokumentaatiosta, jota hyödyntämällä ohjelmoija voi nopeasti toteuttaa haluamansa toiminnallisuuden peliin, koska tietää, mitä etsiä.

7.2 Asennus

Luodun pelimoottorin asennusprosessi erottaa sen huomattavasti monesta muusta pelimoottorista. Käyttäessään C tai C++-kieliä peliohjelmoija ei tarvitse mitään asennusta, ainoastaan kääntäjän sekä käyttöjärjestelmän koodikirjastot, joihin pelimoottori käännösvaiheessa automaattisesti linkittyy. Tämän toteuttaminen aiheutti jonkin verran päänvaivaa, sillä hyödynnettävät kirjastot oli upotettava suoraan lähdekoodiin. Lopputulos on kuitenkin oppilaan kannalta hyvä.

Tärkeänä etuna valitussa moottorin ohjelmointi- sekä käännöstavassa on, että luodun pelin pelaaminen ei vaadi minkäänlaista asennusta. Siten oppilaat voivat helposti jakaa pelinsä luokkatovereidensa ja ystäviensä pelattavaksi. Lisäksi tämä lähestymistapa sallii kaiken koodin suorituksen virheenjäljittäjällä, mikä mahdollistaa oppilaan selvittää, kuinka pelimoottori toimii.

7.3 Ohjelmointikieli

Valitsin C-kielen, koska ajattelin, että yksinkertaisin ohjelmointikieli on tähän tarkoitukseen paras. Prototyypin toteutettuani huomaan, että kielen yksinkertaisuus on monimutkaisempi käsite kuin sen sisältämien käsitteiden määrä. Esimerkiksi, koska C-kielessä ei ole lähdekoodin käännösaikaisen luonnin konseptia, pelimoottorin koodissa toistuvat monet algoritmit sen sijaan, että ne olisi kirjoitettu vain kerran. Esimerkiksi C++-kielen vektori auttaisi vähentämään lähdekoodin määrää. Lisäksi kenties tärkeämpää kuin pelimoottorin koodin yksinkertaistaminen on oppilaan kirjoittaman koodin yksinkertaistaminen. Peleissä usein tarvitaan listoja asioista, ja jos oppilas joutuu kirjoittamaan saman koodin kerta toisensa jälkeen, voi hänen motivaationsa harjoitteluun nopeasti pudota. Tästä syystä C++ on kenties parempi kieli kuin C ohjelmoinnin aloittamiseen.

Mikäli valittu kieli on monimutkaisempi kuin C, kuten C++, tämä valinta voidaan ottaa huomioon pelimoottorin rajapinnassa, jotta kielen käsitteitä voi esitellä oppi-

laalle yksi kerrallaan. Esimerkiksi pelin taustavärin asettaminen voi olla vain yksi globaali funktio, jonka voi helposti esitellä ensimmäisellä oppitunnilla. Suorakaiteen piirtäminen vaatii struct-käsitteen tuntemisen, sillä piirtofunktioiden parametrinä on muun muassa suorakaide. Kenties tekstuurin luominen voisi tapahtua jäsenfunktiolla, ja niitä käyttämällä oppilas voi lisätä peliin kuvia. On siis mahdollista suunnitella rajapinta siten, että oppilas voi käyttää pelimoottorista uusia toiminnallisuuksia aina opettajan opetettua ohjelmointikielen uuden käsitteen. Tämä vahvistaa oppimista, sillä oppilas joutuu käyttämään oppitunnilla esiteltyä käsitettä. Lisäksi se ylläpitää motivaatiota ja halua oppia lisää, sillä uuden oppiminen johtaa suoraan oman pelin parantumiseen, kun oppilas voi käyttää yhä enemmän pelimoottorin ominaisuuksia.

Kaiken kaikkiaan olen sitä mieltä, että luotu pelimoottori on konseptina ja prototyyppinä onnistunut. Pelimoottori, joka on erityisesti suunniteltu ohjelmoinnin opettamiseen, on hyvä apu opettajalle niin käsitteiden esittelemisessä kuin oppilaan mielenkiinnon sytyttämisessä ja ylläpitämisessä. Toteutusvaiheessa esille nousi useita tapoja, jolla pelimoottorista voi kehittää entistä paremman version.

LÄHTEET

Brooks, F, 1987. No Silver Bullet: Essence and Accidents of Software Engineering. Viitattu 27.8.2016. http://www.cs.nott.ac.uk/~pszcah/G51ISS/Documents/NoSilverBullet.html

Gregory, J, 2014. Game Engine Architecture Second Edition, CRC Press.

LaMothe, A, 2002. Tricks of the Windows Game Programming Gurus Second Edition, Sams Publishing.

Microsoft, 2016. Reference for DirectSound. Viitattu 4.10.2016. https://msdn.microsoft.com/en-us/library/windows/desktop/ee416975(v=vs.85).aspx.

Sellers, G & Wright, R & Haemel, N, 2015. OpenGL SuperBible Seventh Edition, Addison-Wesley Professional.