Tampereen ammattikorkeakoulu

Pelimoottorin rakentaminen C++ -kielellä

Ville Paakkunainen

OPINNÄYTETYÖ
Kesäkuu 2021

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

Paakkunainen, Ville:
Pelimoottorin rakentaminen

Opinnäytetyö 18 sivua, joista liitteitä 0 sivua
Kesäkuu 2022

Opinnäytetyössä perehdytään pelimoottorin rakentamiseen. Tavoitteena työssä oli määritellä, suunnitella ja rakentaa toiminnallinen pelimoottori, jonka avulla voidaan jatkokehitysvaiheessa rakentaa erilaisia pelejä. Toisena tavoitteena oli oppiminen pelinkehityksen mahdollisuuksista ilman jonkin ulkoisen tahon tarjoamaa valmista pelimoottoria. Pelimoottori rajattiin tukemaan vain 2D-pelinkehitystä.

Työ toteutettiin C++-ohjelmointikielellä. Kieli valittiin sen prosessoinnin nopeuden sekä tekijän osaamisen perusteella. Työssä vältettiin ulkoisten ohjelmistokehyksien käyttämistä, mutta päädyttiin kuitenkin käyttämään SDL Simple DirectMedia Layer -kirjastoa kehityksen nopeuttamiseksi. SDL-kirjasto soveltui hyvin C++-kehitykseen. Kirjastoa käytetään työssä lähinnä median esitykseen, eli kuvan piirtämiseen ja äänentoistoon.

Työn suunnitteluun käytettiin paljon aikaa. Jo varhaisessa vaiheessa tuli ymmärrys, että moniosainen ja koodirivimäärältään suuri projekti täytyy jäsennellä hyvin, jotta kehitys on sujuvaa. Näin ollen arkkitehtuurisen ratkaisun löytämiseen käytettiin paljon aikaa. Työssä päädyttiin käyttämään pääarkkitehtuurina viestiväylä-järjestelmää, joka välittää muiden järjestelmien viestejä toisillensa. Tämä mahdollisti sen, että järjestelmät, kuten ikkunanhallintajärjestelmä, äänijärjestelmä tai syötejärjestelmä, olivat irrallisia. Järjestelmien välille ei edes voitu aiheuttaa riippuvuuksia, koska järjestelmät eivät tienneet toistensa olemassaolosta. Täten järjestelmiä voitiin kehittää eriaikaisesti ja ohjelmakoodin rakenne saatiin pidettyä selkeänä.

Lopputuloksena työssä saatiin 2D-pelejä pyörittävä pelimoottori. Pelimoottori ei sisällä yhtä monta ominaisuutta kuin kaupalliset ratkaisut, mutta sisältää riittävät toiminnallisuudet monimuotoisten 2D-pelien toteutukseen.

Asiasanat: pelimoottori, pelinkehitys, simple directmedia layer, ohjelmistoarkkitehtuuri

ABSTRACT

Tampere University of Applied Sciences
Degree Programme in Information and Communications Technology
Software Technology

Paakkunainen, Ville:
Game Engine Development

Bachelor's thesis 18 pages, appendices 0 pages
June 2022

The main goals of the thesis were to develop a game engine capable of running 2D games and learning game development without the use of a commercial engine. The plan for continued development is to create a game on top of the created engine in addition to continuing work on the engine itself.

The language chosen for this project was C++. C++ offers fast processing speed, which is crucial for games. The writer of the thesis was also familiar with the language, which allowed for a faster development time. A decision to avoid the use of third-party libraries was also made, but a library for rendering media was deemed to be required if the project was to be finished. For this purpose SDL Simple DirectMedia Layer library was selected. SDL is used in handling rendering, sound output and input management among other things.

The architecture of the game engine was one of the main focus points in the thesis. A large piece of software with multiple systems had to be carefully planned out before any development could be done. A well-executed planning phase was crucial in the success of the thesis. The game engine was split in to different systems, each handling their own focus area. Between these systems a message system was created. The messages system, also known as a message bus, was used in communication from system to system. The systems could this way be developed independently.

As a result of the thesis a game engine capable of running many different kind of 2D games was created. The engine might not implement the same amount of functionalities as other commercial solutions, but it provides a good base to build on. Because of the focus on the software architecture, continued development is also easy to pick up. There are plans create a game on the engine and continue development on the engine.

Key words: game engine, sdl, simple directmedia layer, software architecture

SISÄLLYS

LYHENTEET JA TERMIT

SDL	Simple DirectMedia Layer
SFML	Simple and Fast Multimedia Library

1 JOHDANTO

Tässä opinnäytetyössä oli tavoitteena rakentaa 2D pelimoottori siten, että mahdollisimman monet osa-alueet ovat itse toteutettuja. Opinnäytetyön tekijällä oli halu oppia, miten voitaisiin rakentaa tietokonepeli käyttämättä valmiita moottoreita sen tekemiseen. Itse pelin tekeminen ei kuitenkaan mahtunut tämän työn mittoihin, vaan tässä työssä keskityttiin pelin alla olevaan pelimoottorin toteutukseen.

Nykypäivänä yhden tai vain muutaman kehittäjän tiimeissä tehdyt pelit ovat varsin yleisiä, tavallisten isojen studioiden kehittämien pelien rinnalla. Toisinaan jopa yksittäiset kehittäjät ovat onnistuneet tekemään paremmin menestyviä pelejä, kuin alan suurimmat osalliset. Usein näillä pienillä tiimeillä tai yksittäisillä kehittäjillä on kuitenkin käytössänsä jo olemassa oleva pelimoottori. Pelimoottorit ovat tavallisesti jonkin kansainvälisen yrityksen valmiita tuotteita, joiden kaupallisesta käytöstä joutuu maksamaan. Esimerkiksi Unity Technologiesin Unity pelimoottorin kaupallinen käyttö maksaa 399 € vuodessa per kehittäjä, kun sen avulla tehdystä tuotteesta saatu tuotto ylittää 100 000 dollaria (Unity Technologies, 2022). Aina maksu pelimoottorin käytöstä ei ole kertasumma, vaan tuotto-osuus myydyn pelin tuotoista. Hinta valmiin pelimoottorin käytölle voi siis menestyksestä riippuen olla suuri.

Toinen motivaatio itse tehdyn pelimoottorin rakentamiselle oli oppiminen. Valmista pelimoottoria käyttämällä oppii myös varmasti paljon pelien tekemisestä sekä sen mekaniikoista, mutta iso osa oppimisesta on myös käytettävän työkalun, eli pelimoottorin käyttöliittymän, opettelua. Opinnäytetyön tekijälle oli tärkeämpää oppia pelien alla olevista tekniikoista ja mekaniikoista, kuin opetella käyttämään jotain valmista tuotetta.

Näillä rajauksilla ja tavoitteilla lähdettiin rakentamaan 2D pelimoottoria, jonka avulla voitaisiin jatkokehitysvaiheessa tehdä pelejä.

2 TYÖN TAVOITE JA TARKOITUS

2.1 Työn tavoite ja tarkoitus

Tämän opinnäytetyön tavoitteeksi asetettiin pelimoottorin toteutus. Pelimoottori toteutettiin tukemaan vain 2D pelien kehitystä, eikä siihen toteutettu verkkopelaamisen tukea. Tällä työn rajauksella mahdollistettiin toimivan ohjelmistokehyksen rakentaminen rajatuilla resursseilla ja yhden ohjelmistokehittäjän työllä. Työn toisena tavoitteena oli oppiminen pelienkehityksen tekniikoista ja pelimoottorien toiminnasta. Omaa pelimoottoria lähdettiin rakentamaan muun muassa uuden oppimisen halusta. Työn toteutuksessa toivottiin opittavan enemmän tekemällä itse kaikki pelinkehityksessä tarvittavat työkalut, kuin että työssä olisi käytetty valmiita pelimoottoreita. Toteutettu pelimoottori mahdollistaa jatkokehitykselle useita erilaisia vaihtoehtoja. Pelimoottoria voidaan kehittää sekä sillä voidaan jo luoda erilaisia pelejä. Pelin tekeminen itse rakennetulla pelimoottorilla olisi yksi oiva jatkokehitys idea.

2.2 Työn toteutus

Työn toteutuksen suurimpina haasteina nähtiin resurssien rajallisuus. Työtä oli toteuttamassa vain yksi ohjelmistokehittäjä, eikä tämäkään täysipäiväisesti. Täten työn ja tehtävien rajaukseen kiinnitettiin erityisen paljon huomiota.

2.2.1 Käytetyt tekniikat

Yksi työn tavoitteista oli rakentaa kaikki osat mahdollisimman alusta alkaen. Ohjelmointikieleksi valittiin C++ kehittäjän osaamisen sekä sen prosessoinnin nopeuden takia. C++ soveltui työhön myös, koska se on niin sanotusti matalan tason kieli, jota tarvittiin jotta voitiin välttää ulkoisien ohjelmistokehyksien käyttöä.

Työssä käytettiin kuitenkin SDL Simple DirectMedia Layer -kirjastoa, joka auttoi kuvan piirtämisessä, äänien toistamisessa sekä käyttäjän syötteen kaappaamisessa. SDL kirjasto otettiin käyttöön, sillä muutoin työn laajuus olisi ollut yhdelle kehittäjälle rajallisilla aikaresursseilla mahdoton. Muita kirjastovaihtoehtoja, joita työn suunnittelussa tutkittiin oli muun muassa SFML Simple and Fast Multimedia Library. Työhön valittiin kuitenkin SDL, sillä se on paljon enemmän käytössä pelinkehitysteollisuudessa sekä sille löydettiin tarvittavat tukevat resurssit. SDL kirjastoa käyttää pelinkehityksessä muun muassa alan suurimpiin kuuluva Valve, ja sitä on käytetty pelien kuten Counter Strike: Global Offensive, Portal, Portal 2 ja useiden Half-Life pelien kehityksessä. Koska pelienkehittäjä Valve käyttää SDL kirjastoa yhä aktiivisesti, sen kehittäjät myös osaltaan kontribuoivat SDL kirjaston kehitykseen ja näin voitiin varmistua siitä, että SDL kirjaston tuki ei tule loppumaan lähivuosina. Tämä mahdollistaa pidempiaikasemmankin projektin, jota voidaan kehittää eteenpäin ja jonka tuloksia voidaan hyödyntää vielä useita vuosia. SDL on myös alusta riippumaton, joten kehittäjän ei tässä työssä tarvinnut perehtyä eri käyttöjärjestelmien aiheuttamiin mahdollisiin ongelmiin.

3 TEORIA

3.1 Pelinkehitys

Gregory (2018) esittää, että pelit voisivat olla monien tieteilijöiden mukaan todellisen elämän interaktiivisia tietokonesimulaatioita. Gregory pohjustaa ajatustaan sillä, että kuten fyysisessä maailmassa, myös pelimaailmassa on maailma muovattu matemaattisten yhtälöiden mukaisesti. Virtuaalimaailma on yksinkertaistaus ja mallinnus fyysisestä maailmasta. Pelit ovat myös dynaamisia, eli niillä on vaihteleva tila, joka on riippuvainen pelin sisäisestä ajasta ja paikasta.

Pelinkehitys toteutetaan usein olemassa olevien pelimoottoreiksi kutsuttujen ohjelmistojen avulla. Pelimoottorit tarjoavat rajapinnat muun muassa digitaalisen tiedon näytölle sopivaan esitysmuotoon muuntamiseen eli renderöimiseen, käyttäjän syötteen kaappaamiseen, äänien toistamiseen, fysikaalisiin laskelmiin, skriptaamiseen sekä moniin muihin ominaisuuksiin (Tyler 2022). Tällaisia pelimoottoreita ovat esimerkiksi Unity, Unreal Engine. Nämä molemmat ovat vapaasti käytettävissä ei-kaupallisissa tarkoituksissa. Ne ovat toteutettu C ja C++ -ohjelmointikielillä.

3.1.1 Pelinkehityksen roolit

Pelien kehitys tapahtuu perinteisesti melko kookkaissa ryhmissä tai tiimeissä joissa on usein selkeät roolit. Yksittäiset tiimien osalliset voivat kuitenkin tehdä töitä useammalla roolilla, ja rooleja voidaan vaihdella.

Pelinkehityksen roolit voidaan jakaa viiteen peruskategoriaan: insinööreihin, taiteilijoihin, pelisuunnittelijoihin, tuottajiin sekä hallintoon ja muihin tukeviin rooleihin liittyviin jäseniin. Nämä viisi perusroolia voidaan vielä jakaa erilaisiin alaryhmiin. Esimerkiksi insinöörit, eli kehittäjät, voidaan yleisesti jakaa kahteen ryhmään. Toinen ryhmä kehittää peliä, ja toinen ryhmä pelin kehityksessä käytettäviä työkaluja. Pelinkehityksessä taiteilijat voidaan jakaa ainakin yhdeksään eri

alakategoriaan, joita ovat muun muassa piirtäjät, 3D mallintajat, muusikot ja äänisuunnittelijat sekä ääninäyttelijät. Pelisuunnittelun rooliin kuuluu ainakin juonen kirjoittajat, pelimekaniikan suunnittelijat sekä ohjaajat. Pelinkehityksessä tuottajien rooli on vaihteleva. Tuottajat voivat toimia aikatauluttajina, henkilöstöresurssien hallinnoijina tai he voivat myös olla suoraan osallisina pelin rakentamisessa. Usein tuottajat toimivat myös pelinkehittäjien ja kaupallisen yhteistyön välitilassa. Hallinnon jäsenet toimivat yrityksen päätäntäelimessä, esimiestehtävissä ja valvovissa asemissa. Muita pelinkehitystä tukevia rooleja ovat markkinoinnin roolit, julkaisijat ja IT osaston henkilöstö. (Gregory, 2018).

3.2 Pelimoottorien kehitys

Unity pelimoottorin kehityksestä vastaava Unity Technologies määrittelee pelimoottorin työkaluiksi ja toiminnoiksi, joiden avulla ohjelmistokehittäjä pystyy kehittämään ammattimaisesti ja tehokkaasti eri pelejä (2022 Unity Technologies).

3.2.1 Pelimoottorien historia

Pelimoottoria terminä alettiin käyttää 1990-luvun puolivälillä muun muassa huippusuosittuun id Softwaren tekemään Doom-peliin viitaten. Doom oli suunniteltu arkkitehtuurillisesti siten, että sen keskeisimmät ohjelmistokomponentit olivat selvästi erotettuja pelimekaniikoista, tekstuureista ja muista asioista, joista pelaajan pelikokemus koostuu. Erottelu tuotti lisäarvoa, sillä id Software sai tuotettua lisää pelejä tekemällä vain pieniä muutoksia keskeisiin ohjelmistokomponentteihin. Tämä synnytti myös niin kutsutun ”modaus”-yhteisön, jossa yksittäiset tekijät tai pienet tiimit tekivät uusia tai muunneltua pelejä, muokkaamalla vanhoja pelejä tai käyttämällä niiden moottorin rajapintoja. Vuosituhannen lopussa oltiin jo tehty pelejä, joissa oli tarkoituksenmukaisesti suunniteltu pelin toteutus niin, että pelin sisäisiä ohjelmistokomponentteja voidaan käyttää myös jatkossa uusien pelien kehityksessä. Pelin ja pelimoottorin erottelu toi myös pelejä tekeville yrityksille lisätuottoja, koska se mahdollisti niin pelin myynnin, kuin myös pelimoottorin lisensoinnin muiden käyttöön. (Gregory, 2018.)

4 2D PELIMOOTTORIN TOTEUTUS

4.1 Kehitysympäristö

Pelimoottori toteutettiin C++-ohjelmointikielellä. C++ valittiin ohjelmointikieleksi sen prosessoinnin nopeuden takia. Peleissä ruudun päivitystaajuudella on loppukäyttäjälle suuri merkitys, joten prosessoinnin nopeus on keskiössä. Kehitysympäristönä käytettiin Microsoft Visual Studiota. Versionhallinnassa käytettiin gittiä.

4.2 Käytettävät kirjastot

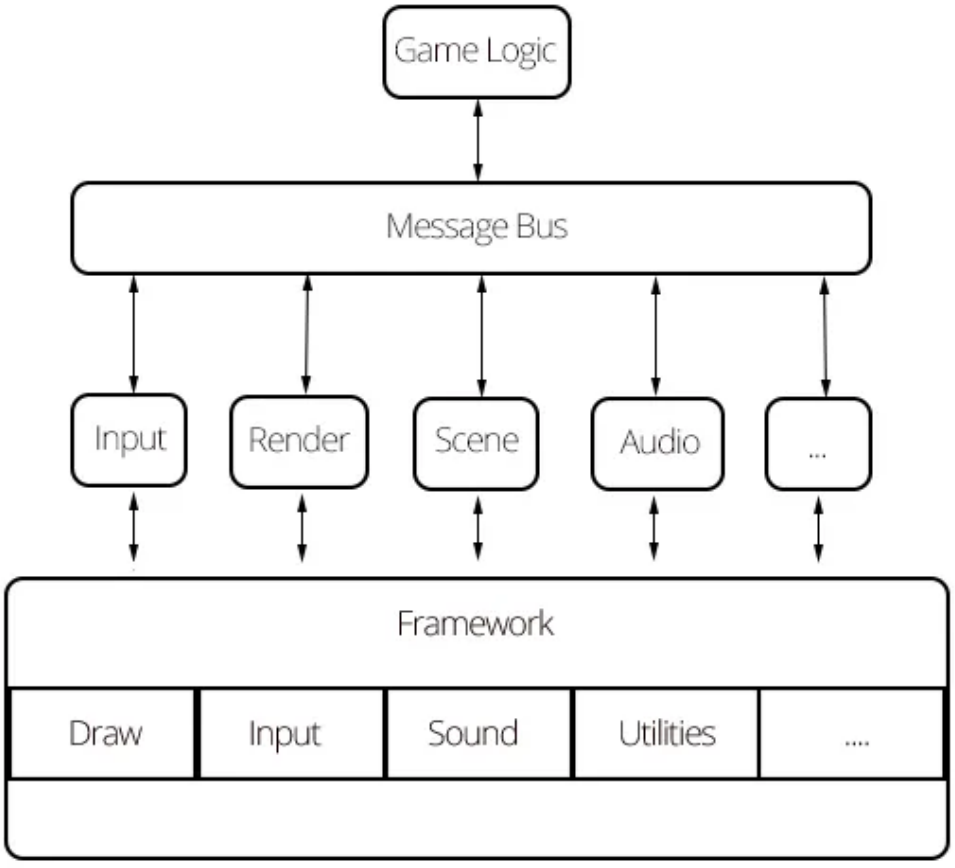
Toteutuksessa oli tavoitteena käyttää mahdollisimman vähän ulkoisia kirjastoja. Ohjelmaikkunan luontiin, renderöintiin, tekstuurien ja muun median käyttöön päädyttiin kuitenkin valitsemaan SDL2-kirjasto kehityksen nopeuttamiseksi. Mikäli SDL2-kirjastoa ei olisi käytetty, olisi työmäärä ollut mahdottoman suuri toteuttaa tämän opinnäytetyön mitoissa. SDL2-kirjaston ja sen lisäosien SDL_mixer, SDL_image ja SDL_ttf lisäksi toteutuksessa käytettiin vain C++ standardikirjastoja.

4.2.1 SDL

Akronyymi SDL tulee sanoista Simple DirectMedia Layer. SDL helpottaa ohjelmistokehitystä tarjoamalla rajapinnat alhaisen tason funktioiden käyttöön. SDL on kirjoitettu C kielellä, ja se käyttää tekstuurien piirtämiseen ja äänen toistamiseen OpenGL ja Direct3D ohjelmistokehyksiä. Käytettäväksi grafiikka- ja äänikirjastoksi valittiin SDL muun muassa sen alustariippumattomuuden takia. Näin toteutetulla pelimoottorilla tehdyt pelit ovat natiivisti *cross-platform*, eli järjestelmäriippumattomia. SDL on myös yleisesti käytössä peliteollisuudessa.

4.3 Ohjelman arkkitehtuuri

Toteutetun ohjelmiston arkkitehtuuri suunniteltiin pitkäaikaista jatkokehitystä ajatellen. Arkkitehtuurisuunnitelma tehtiin Michael Kissnerin kirjoittamaan blogikirjoitukseen perustuen. Kissnerin esittämässä arkkitehtuurissa pelimoottori jaetaan toisistaan riippumattomiin järjestelmiin, jotka kommunikoivat viestiväylän avulla. Järjestelmiin jakaminen mahdollistaa eri osien eri aikaisen kehityksen, eikä kehittäjän tarvitse ymmärtää kaikkien järjestelmien toimintaa yhtäaikaisesti. Ohjelmiston kehittäminen on helpompaa, kun voidaan keskittyä yhteen järjestelmään kerrallaan. Tämä helpottaisi myös työnjakoa, mikäli ohjelmistolla olisi useampi kehittäjä. Jos yksi kehittäjä vastaa syötejärjestelmästä, ei toisen tarvitse sen toimintaa syvemmin ymmärtää.



Kuva 1 Kissnerin esittämä arkkitehtuurinen ratkaisu pelimoottorille.

4.4 Ohjelman suorittaminen

C++ ohjelmien suoritus alkaa main() -funktion toteutuksesta. Tavoitteena oli toteuttaa mahdollisimman yksinkertaistettu main() -funktio ohjelmakoodin lukemisen ja jatkokehityksen helpottamiseksi sekä niin sanotun ”spagettikoodin” välttämiseksi. Täten funktio koostuu vain eri järjestelmien deklaraatioista ja näiden järjestelmien update() -funktion kutsuista silmukan sisällä. Update-kutsun toimintaan perehdytään kappaleessa 4.6. Silmukan lopussa kysytään lopetuslipun tilaa ikkunajärjestelmältä.

```
#pragma once
#include "Messagebus.h"
#include "Window.h"
#include "Input.h"
#include "File.h"
#include "Audio.h"

int main(int argc, char* args[])
{

            Messagebus msgbus;
            Window window(&msgbus);
            Input input(&msgbus);
            Audio audio(&msgbus);
            Gamelogic gamelogic(&msgbus);

            bool endgame = false;

            while (!endgame) {
                        window.update();
                        audio.update();
                        input.update();
                        gamelogic.update();
                        msgbus.notify();
                        endgame = window.getEndgame();
            }

            return 0;
}
```

Ohjelmakoodin suorituksen lopetuslipun tila päätettiin säilyttää ikkunanhallintajärjestelmässä, koska ikkunan sulkemisen ajateltiin olevan viimeinen suoritettava asia ohjelman ajoa lopetettaessa. Ikkunan sulkeutuminen on myös loppukäyttäjälle selvä merkki siitä, että suoritettava prosessi on loppunut.

4.5 Järjestelmät

Ohjelma on jaettu viiteen eri järjestelmään, jotka ovat toisistaan riippumattomia. Järjestelmät eivät kommunikoi suoraan keskenään, eivätkä sinänsä tiedosta toistensa olemassaoloa. Tämä mahdollisti järjestelmien itsenäisen ja eri aikaisen kehityksen. Näin myös vältyttiin järjestelmien välisten riippuvuuksien aiheuttamalta kokonaisuuden monimutkaistumiselta. Kukin järjestelmä vastaa tietystä osasta kokonaisuutta.

4.5.1 Messagebus

Messagebus on viestiväyläjärjestelmä. Viestiväylä on yksi kriittisimmistä järjestelmistä ohjelmassa. Sen avulla voidaan välittää tietoa järjestelmältä toiselle, ilman, että järjestelmät kommunikoisivat keskenään. Viestiväylä toteutettiin käyttämällä Observer -suunnittelumallia.

Viestiväylä tallettaa sille saapuneet viestit jonoon, josta ne käsitellään järjestyksessä. Kukin järjestelmä saa luettavakseen kaikki väylältä tulevat viestit, ja reagoi niihin tarvittaessa. Viestien tietotyyppinä on *enum,* mutta tarvittaessa viestiin voidaan liittää niin paljon tietoa ja eri tietotyyppejä kuin vaaditaan. Suuren tietomäärän siirtäminen viestiväylän kautta ei kuitenkaan ole tarkoituksenmukaista. Esimerkiksi käyttäjänsyötteestä vastaava järjestelmä *Input* kommunikoi käyttäjän syötteen viestinä viestiväylään. Kun käyttäjänsyöte on hiiren liike, tarvitaan viestin lisäksi hiiren kursorin koordinaatit. Tällöin viestiin täytyy lisätä muuttuja tai muuttujia.

4.5.2 Window

Window eli ikkunanhallintajärjestelmä vastaa ohjelmaikkunan luonnista, sen päivittämisestä ja sen sulkemisesta. Järjestelmä käyttää SDL kirjastosta kirjastoja SDL.h, SDL_image.h ja SDL_ttf.h. Näistä ensimmäinen on yleinen SDL kirjasto, jota tarvitaan muiden SDL kirjastojen käyttämiseen. Toinen kirjasto mahdollistaa kuvan piirtämisen ikkunaan. Kolmannen kirjaston avulla voidaan kirjoittaa

truetypefont -tyypin tekstiä. Ikkunanhallintajärjestelmä päätettiin asettaa tässä työssä ikään kuin pääjärjestelmäksi. Ikkunan luonti aloittaa ohjelman ajon, ja sen sulkeminen päättää ajon. Täten ikkunanhallintajärjestelmä lähettää viestiväylään ensimmäiset käynnistysviestit, jotka ohjaavat muita järjestelmiä käynnistymään ja alustamaan toimintansa.

4.5.3 Input

Input eli syötejärjestelmä vastaa käyttäjän syötteen käsittelystä. Käyttäjän syötteitä ovat tässä tapauksessa näppäimistö- ja hiiren nappien painallukset, sekä hiiren kursorin liike. Mikrofonisyötettä ei toteutettu tässä työssä, sillä sen ei ajateltu olevan merkityksellinen useimmissa 2D peleissä, joissa ei ole verkkopelaamista. Järjestelmä lähettää syötteet viesteinä viestiväylään, jotta muut järjestelmät voivat reagoida niihin. Syötejärjestelmä ei juurikaan käytä muilta järjestelmiltä tulevia viestejä. Kuitenkin muista järjestelmistä voidaan laittaa käski viestiväylään syötteen kaappaamisen lopettamiseksi. Tämä voi olla hyödyllistä joissain tapauksissa, kun käyttäjän syötteestä ei olla kiinnostuneita muissa järjestelmissä.

4.5.4 Audio

Audiojärjestelmän avulla voidaan toistaa tietokoneen toistolaitteesta ääntä. Audiojärjestelmä käyttää SDL kirjaston SDL mixeriä. Audiojärjestelmän toimintaa nojaa vahvasti viestiväylään, ja eritoten pelilogiikan lähettämiin viesteihin. Kun pelilogiikkajärjestelmä ilmoittaa viestiväylään jostakin tapahtumasta, äänijärjestelmä toistaa tälle tapahtumalle ennalta määrätyn äänitiedoston. Tällaisia voivat esimerkiksi olla pelattavan hahmon hyppääminen, liikkuminen tai pelimaailman tapahtumat.

4.5.5 Gamelogic

Gamelogic eli pelilogiikkajärjestelmä sisältää kaiken mikä tekee pelistä ”pelin”. Kun toteutetun pelimoottorin avulla tehdään peliä, pelikehittäjän ei tarvitse huolehtia mistään muusta järjestelmästä. Kaikki peliin ja sen logiikkaan liittyvä voidaan pitää pelilogiikkajärjestelmän sisällä, ja pelikehittäjän täytyy tietää vain millaisia eri viestejä voi mikäkin järjestelmä vastaanottaa. Mikäli kehittäjä kokee, ettei hänen haluamalleen toiminnolle löydy valmista ratkaisua, on kehittäjän helppo lisätä järjestelmiin ominaisuuksia. Ominaisuuksien lisääminen ei sinänsä vaikuta järjestelmien toimintaan.

Tämän opinnäytetyön puitteissa pelilogiikkajärjestelmä on jätetty vain rungoksi jatkokehitystä varten. Tässä opinnäytetyössä on keskitytty rakentamaan pelimoottoria itse pelin sijaan.

4.6 Update -suunnittelumalli

Pelimoottorin toiminnallisessa pääsilmukassa kutsutaan eri järjestelmien update() -funktiota. Funktion toteutus on kullakin järjestelmällä tehty samalla idealla. Update funktion on tarkoitus käsitellä järjestelmän hallinnoimaa tietoa yhden ruutupäivityksen ajalta. Esimerkiksi ikkunanhallintajärjestelmässä update-funktiossa kutsutaan kaikkien renderöitävien asioiden renderöinti kutsua. Syötejärjestelmä lähettää sille siihen mennessä rekisteröidyt syötteet, kun taas äänentoistojärjestelmä aloittaa ajoitetut äänet.

5 POHDINTA

Työn teknisessä toteutuksessa onnistuttiin pääosin hyvin. Kaikki tarvittavat järjestelmät, joita pelien rakentamisessa tarvitaan, ovat toteutettuna ohjelmistossa. Arkkitehtuurilliseen suunnitteluun käytettiin paljon resursseja, ja se osoittautui oikeaksi ratkaisuksi. Nyt toteutetun ohjelmiston tila on sellainen, että jatkokehitykseen ryhtyminen on helppoa. Mukaan voitaisiin ottaa myös muita kehittäjiä, kun olemassa oleva toteutus on selkeästi toteutettu ja dokumentoitu.

Työssä hankaluuksia tuotti koko ohjelmiston koko ja työn määrä. Yhdelle kehittäjälle tekemistä on täydellisessä pelimoottorissa liikaa. Siksi tässäkin työssä jouduttiin tekemään kompromisseja, kun kaiken toteuttaminen ei ollut mahdollista. Huolellisesti tehty suunnittelu helpottaa kuitenkin jatkoa, ja mahdollisesti muiden kehittäjien mukaantuloa.

Jos tämän työn päätavoitteena olisi ollut tehdä peli, olisi tässä tavoitteessa epäonnistuttu, mikäli se oltaisiin haluttu rakentaa tässä työssä tehdyllä moottorilla. Resurssit eivät olisi riittäneet siihen, vaikka moottori siihen kykenisikin. Tämän työn pelimoottorilla kuitenkin tehtiin erilaisia kokeiluja, mutta ei mitään mitä voisi kutsua peliksi tai edes pelattavaksi. Pelkästään pelien kehittämisestä kiinnostuneiden kannattaa varmasti tutkia olemassa olevia moottorivaihtoehtoja, sen sijaan, että he käyttäisivät aikaa oman moottorin tekemiseen.

Tästä työstä on opittu paljon. Luettu materiaali on opettanut erilaisia ohjelmistosuunnittelumalleja ja arkkitehtuurisia ratkaisuja. Käytännön tekeminen, eli tässä tapauksessa ohjelmointi, on vahvistanut allekirjoittaneen ohjelmointitaitoja merkittävästi. Eritoten työstä on opittu, kuinka tärkeää on tavoitteiden tarkka rajaus ja ohjelmiston suunnittelu. Vaikka ohjelmointimaailmassa ollaan siirtymässä ketterään kehitykseen tarkan etukäteissuunnittelun sijaan, on suunnittelulla silti roolinsa. Tämä pitää erityisesti paikkaansa, kun kokemattomampi tekijä ryhtyy isoon projektiin.

LÄHTEET

Unity Technologies. 2022. Choose the plan that is right for you. Verkkosivu. Viitattu 27.5.2022. https://store.unity.com/compare-plans

Gregory, J. 2018. Game Engine Architecture. 3. painos. Yhdysvallat: A K Peters/CRC Press.

Unity Technologies. 2022. Game development terms. Verkkosivu. Viitattu 20.5.2022. https://unity.com/how-to/beginner/game-development-terms

Tyler, Dustin. 2022. How to Choose the Best Video Game Engine. Verkkosivu. Viitattu 20.5.2022. https://www.gamedesigning.org/career/video-game-engines/

Kissner, Michael. 2015. Writing a Game Engine from Scratch – Part 1: Messaging. Verkkosivu. Viitattu 20.5.2022. https://www.gamedeveloper.com/programming/writing-a-game-engine-from-scratch---part-1-messaging