VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Jussi Viitala

RÄÄTÄLÖIDYN PELIMOOTTORIN KEHITTÄMINEN

Tekniikka
2020

VAASAN AMMATTIKORKEAKOULU

Tietotekniikka

TIIVISTELMÄ

Tekijä	Jussi Viitala
Opinnäytetyön nimi	Räätälöidyn pelimoottorin kehittäminen
Vuosi	2020
Kieli	suomi
Sivumäärä	38
Ohjaaja	Jani Ahvonen

Opinnäytetyö on tehty opiskelijan omasta pelimoottoriprojektista, joka on suunniteltu mahdollisimman yksinkertaiseksi ja sillä tehtyjä pelejä pitää pystyä muokkaamaan ja laajentamaan helposti. Valmiit saatavilla olevat pelimoottorit sisältävät usein paljon ominaisuuksia, jotka hyödyttävät vain tietynlaisia pelejä. Lisäksi projektin tavoitteiden toteuttaminen on hankalampaa, sillä pelimoottorit ovat monimutkaisia ja niiden käyttämät ohjelmointikielet tuovat usein lisää rajoituksia.

Pelimoottorin eräs erottava tekijä on renderöintitekniikka, jossa piirretään 2D-kuvia päällekkäin pienellä y-tason poikkeamalla edelliseen antaen näin melko hyvän 3D-vaikutelman. Tämä metodi vaatii suurta määrää piirtokutsuja näytönohjaimelle ja se on eräs syy toteuttaa pelimoottorin ydinosa C-kielellä. Varsinainen pelilogiikka ja resurssien hallinta toteutetaan Lua-skriptauskielellä.

Kaikkia ominaisuuksia ei ole vielä toteutettu, mutta projektin tässä vaiheessa voi todeta, että toteutustavalla on ollut selviä hyötyjä antaessa vapaammat kädet toiminnan toteutukselle, kuitenkaan lisäämättä työtaakkaa kohtuuttomaksi.

Avainsanat Game Engine, C, Lua

VAASAN AMMATTIKORKEAKOULU

UNIVERSITY OF APPLIED SCIENCES

Tietotekniikan koulutusohjelma

ABSTRACT

Author	Jussi Viitala
Title	Tailor Made Game Engine
Year	2020
Language	Finnish
Pages	38
Name of Supervisor	Jani Ahvonen

This thesis is based upon a game engine project, designed to be simple and games made with it should be easy to modify and expand upon. Available game engines often include a large set of features and many of them only benefit specific type of games. In addition, their bulky design and language choices bring additional challenges for the design goals of this project.

One unique feature of the engine is the rendering where 2D images are drawn in an offset from the last in y-axis to create an illusion of 3D. This method requires quite a lot of draw calls so the core part of the engine is implemented in the C language. The actual gamelogic and resource management is made with the Lua-scripting language.

Not all of the features have been implemented yet, but at this point it can be stated that there is a certain benefit in this type of game engine development that gives the developer full control of the design without being too laborious.

Keywords Game Engine, C, Lua

SISÄLLYS

KUVA- JA TAULUKKOLUETTELO

TERMIT JA LYHENTEET

SDL	Simple DirectMedia Layer, alustariippumaton laitelähei-nenkirjasto äänen, näppäimistön, hiiren, joystickin ja gra-fiikkalaitteiston hallintaan /1/.
GCC	GNU C Compiler eli GNU C-kääntäjä.
RGBA	Väriformaatti, Red, Green, Blue, Alpha.
Skripti	Tässä dokumentissa skriptillä viitataan yleensä Lua-skrip-teihin eli tulkattavaan ohjelmakoodiin.
FPS	Frames Per Second, ruudunpäivitysnopeus.
VRAM	Video Random Access Memory, videomuisti.
Tile	Kartan yksi ruutu, joka voi olla neljää eri tyyppiä.
Tilemap	Tileistä koostuva karttadata.
Tileset	Kokoelma erilaisia tilejä, joista kartta koostuu.
Kerros	Tilemap kontekstissa tämä tarkoittaa yhtä kerrosta tilemapissa.
UI	User Interface, käyttöliittymä.
Canvas	Piirtoalusta eli tekstuuri, johon voidaan piirtää grafiikkaa.
Slice	Yksi viipale, josta tile koostuu. Yksi kerros koostuu 32 viipaleesta.
Megaslice	Kartan renderöintiä nopeuttava iso tekstuuri, johon slicet aluksi piirretään.
Pikseli	Kuvapiste bittikarttagrafiikassa.
Vokseli	Pikselin kolmiulotteinen vastike.
API	Application Programming Interface, ohjelmointirajapinta.

1 JOHDANTO

Nykypäivänä on saatavilla paljon valmista ohjelmistoa pelikehityksen helpottamiseksi pelimoottoreiden ja erilaisten kehyskirjastojen muodossa. Saatavilla olevista pelimoottoreista monet ovat erittäin kattavia ominaisuuksiltaan, myös täysin ilmaiset ja avoimeen lähdekoodiin perustuvat. Tästä huolimatta oman pelimoottorin tekeminen saattaa joskus olla varteenotettava vaihtoehto. Sen suurimpina etuina on parempi tuntemus ohjelmasta sekä sen kehityksen ohjaus. Valmiissa ratkaisuissa saattaa usein olla resurssit kohdennettu ominaisuuksiin, jotka eivät palvele kyseisen projektin tarpeita. Lisäksi ominaisuudet saattavat muuttua toiminnaltaan tai ne voidaan poistaa kokonaan.

Tässä projektissa yksi kantavista ajatuksista oli yksinkertainen arkkitehtuuri, johon sisällytetään vain tarvittavat toiminnot, jolloin vältytään valmiiden ratkaisujen tuomasta redundanssista. Lisäksi yksi motivoiva tekijä on ollut tekijän itsensä haastaminen ja ohjelmointitaitojen kartuttaminen.

Projektin toimeksiantaja on tekijä itse ja sen on tarkoitus tulla kaupalliseen käyttöön mahdollisesti useampaankin peliprojektiin. Varsinainen kehitys on alkanut toukokuussa 2019 ja on vuoden 2020 aikana todennäköisesti tilassa, jolloin varsinaisen pelin tekeminen voidaan aloittaa. Tämä opinnäytetyö perustuu toistaiseksi tehtyyn työhön, jossa on suurin osa lopullisesta toiminnallisuudesta. Pelimoottori on saanut nimekseen SliceEngine, viitaten sen viipalepohjaiseen karttarenderöintiin.

2 KÄYTETYT TEKNOLOGIAT

2.1 Kirjastot

2.1.1 SDL2

Simple DirectMedia Layer on alustariippumaton laiteläheinen kirjasto äänen, näppäimistön, hiiren, joystickin ja grafiikkalaitteiston hallintaan. Sitä on käytetty monissa videotoisto-ohjelmissa, emulaattoreissa ja videopeleissä. Tässä projektissa kirjastoa on käytetty ikkunan luontiin, piirtämiseen ja syötteiden hallintaan. SDL on kirjoitettu C-kielellä ja se on vapaa kirjasto /1/.

2.1.2 SDL_image 2.0

SDL_image 2.0 on SDL-kirjastoa laajentava kuvanlatauskirjasto, joka mahdollistaa useamman kuvaformaatin lataamisen kuin SDL2. Tässä projektissa sitä on käytetty pääasiassa PNG-kuvien lataamiseen /2/.

2.1.3 SDL_mixer 2.0

SDL-kirjastoa laajentava äänimiksauskirjasto, joka mahdollistaa rajoittamattoman määrän yhtäaikaisesti soitettavia kanavia sekä yhden musiikkikanavan. Lisäksi se helpottaa eri tyyppisten äänitiedostojen lataamista. Tässä projektissa sitä on käytetty toistamaan musiikkitiedostoja, kuten Ogg Vorbis- ja MOD-tiedostoja, sekä helpottamaan äänien toistoa. Lisäksi sitä käytetään äänien ryhmien hallinnoimiseen /3/.

2.1.4 Lua 5.3

Lua on tehokas, kevyt ja sulautettava skriptauskieli, joka tukee proseduuriohjelmointia, olio-ohjelmointia, funktionaalista ohjelmointia, datavetoista ohjelmointia ja datan kuvausta. Lua on tulkattava kieli ja se on kirjoitettu C-kielellä. Lua sopii hyvin skriptauskieleksi C-kielellä toteutettuun ohjelmaan ja sillä voidaan kutsua C-kielisiä rekisteröityjä funktioita. Lua on ilmainen ohjelmisto. Tässä projektissa Lua on keskeinen kirjasto ja sitä käytetään konfiguraatio-, data- sekä skriptitiedostoissa /4/.

2.2 Kääntäminen

2.2.1 GCC

GNU-kääntäjäkokoelma sisältää C, C ++, Objective-C, Fortran, Ada, Go ja D käyttöliittymät sekä näiden kielten kirjastot (libstdc ++, ...). GCC kirjoitettiin alun perin GNU-käyttöjärjestelmän kääntäjäksi. GNU-järjestelmä on kehitetty olemaan 100 % vapaa ohjelmisto, ilmainen siinä mielessä, että se kunnioittaa käyttäjän vapautta /9/. GCC toimii tämän projektin pääsääntöisenä C-kääntäjänä, mutta kehitysvaiheessa käytetään myös Clangia.

2.2.2 Clang

Clang-projekti tarjoaa kielen käyttöliittymän ja työkaluinfrastruktuurin C-kielten perheelle /10/. Sitä on tässä projektissa käytetty kehitysvaiheessa toisena kääntäjänä, sillä se antaa usein selkeämpiä virheviestejä.

2.2.3 MinGW

MinGW on minimalistinen kääntämisympäristö natiiveille Windows-ohjelmille / 5/. MinGWtä käytetään pelimoottorin Windows-version kääntämiseen.

2.2.4 Cmake

Cmake on avoimeen lähdekoodiin perustuva alustariippumaton tuoteperhe ohjelmien kääntämiseen, testaamiseen ja pakkaamiseen. Se helpottaa eri käyttöjärjestelmäkäännöksiä generoimalla natiiveja makefilejä käyttämällä yksinkertaista alustaa ja konfiguraatiotiedostoja /6/.

2.3 GDB

GNU Project debugger on ohjelma, jolla voidaan nähdä suoritettavan ohjelman sisälle suoritushetkellä tai kun ohjelma kaatuu /7/. Käytettyä debuggaustyökalua käytetään pääsääntöisesti VSCoden kanssa. Kääntäjälle on asetettu liput ”-O0 -ggdb”, jotka parantavat debuggausinformaatiota.

2.4 Git

Git on ilmainen ja avoimeen lähdekoodiin perustuva versionhallintaohjelma /8/. Projektin säilytyspaikkana on käytetty GitHubia, jossa kirjoitushetkellä käyttäjä voi luoda maksutta yksityisiä säilöjä (en. Repository).

3 MÄÄRITTELY

3.1 Vaatimusmäärittely

Projektilla pyritään palvelemaan kehittäjänsä tarvetta suoraviivaiselle datakeskeiselle ja omintakeisen visuaalisen lopputuloksen omaavalle pelimoottorille. Pelimoottorin päälle rakennettujen pelien tulisi olla loppukäyttäjän helposti muokattavissa ja laajennettavissa avoimen datarakenteen ansiosta. Toissijaisena tavoitteena on ollut kehittäjän ohjelmointitaitojen kehittäminen ja ylläpitäminen.

Taulukossa 1 on listattuna ominaisuuksia luokiteltuna niiden prioriteettien mukaan ja koska pelimoottori on vielä kesken, puuttuvat ja puutteelliset ominaisuudet on merkitty tähdellä (*). Taulukossa mainitut suoritinvaatimukset ovat viitteellisiä.

Taulukko 1. Vaatimustaulukko.

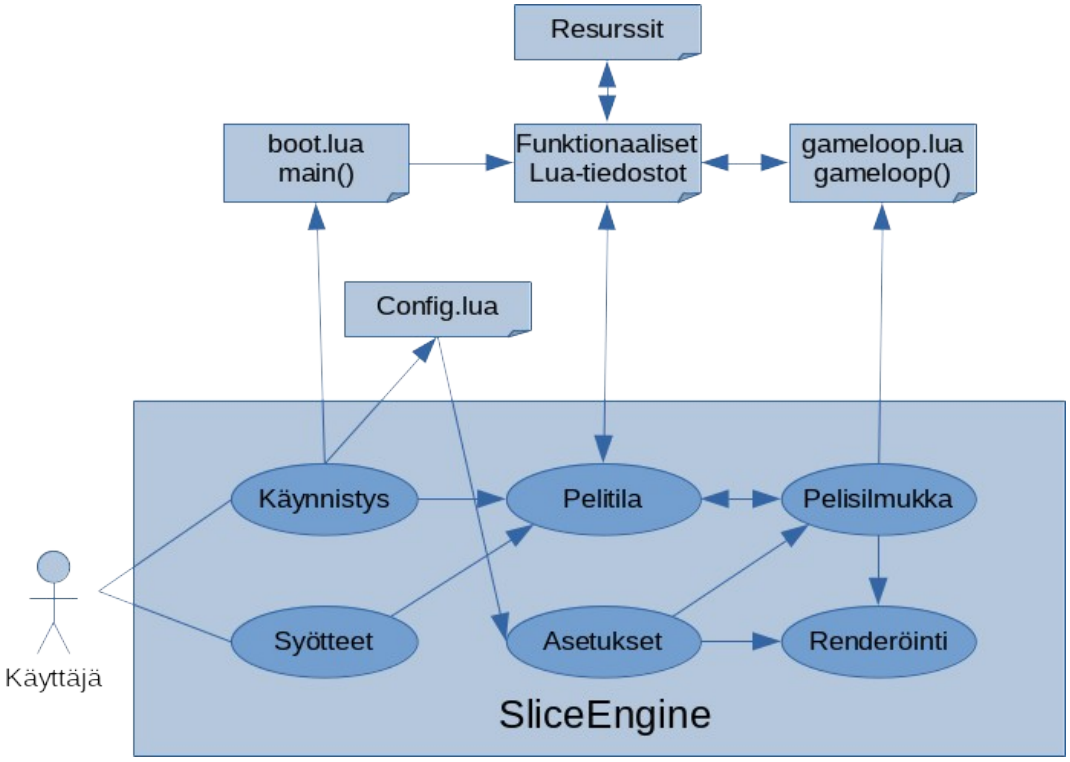
Prioriteetti	Ominaisuus

Pakollinen	• Tasainen 60 FPS pc:llä, jossa 2 GHz suoritin ja 512 MB VRAM, piirrettäessä neljä kerrosta
	• Lua-rajapinta
	• Keskeisimmät Lua-tiedostot
	• Aktiivistenobjektien piirtäminen
	• Käyttöliittymä, johon kuuluu
	  ◦ Käyttöliittymätekstuuri
	  ◦ Bittikarttafontti
	  ◦ Teksti
	  ◦ Canvas
	  ◦ Graafinen osoitin
	• Näppäimistön ja hiiren lukeminen
	• Tile-pohjainen valaistus
	• Musiikin toisto
	• Äänien toisto (myös tilalliset äänet)
	• * Reitinhaku
	• * Lokitiedostojen luonti

Tärkeä	• Tasainen 60 FPS pc:llä, jossa 2 GHz suoritin ja 256 MB VRAM, piirrettäessä kaksi kerrosta • * Animoitavat tilet • * Lua-apufunktiot ja luokat
Ehdollinen	• * Mac-versio • * 32:n bitinversiot • * Raspberry Pi-versio, jossa OpenGl ES tuki • * Peliohjaintuki • * Yksikkötestit

3.2 Toiminnallinen määrittely

Pelimoottorin binääriosuus sisältää vain tärkeimmät primitiiviobjektit, joita tarvitaan pelitilanteen renderöintiin ja syötteiden lukemiseen. Varsinainen pelin toiminnallisuus ohjelmoidaan Lua-kerroksessa, josta voidaan luoda näitä objekteja ja muokata niiden ominaisuuksia. Lisäksi varsinaiset peliasetukset, kuten näppäinasetukset ja äänenvoimakkuuden säätö tehdään Lua-kerroksessa. Tämä työ keskittyy pääosin kuvaamaan ohjelman toimintaperiaatetta, eikä ohjelmakoodia käydä läpi kuin osittain. Lisäksi ohjelman laajuuden vuoksi käydään sen toiminta läpi melko yleisellä tasolla ja esimerkiksi Lualla toteutetun karttaeditorin toiminta jätetään tämän dokumentin ulkopuolelle.



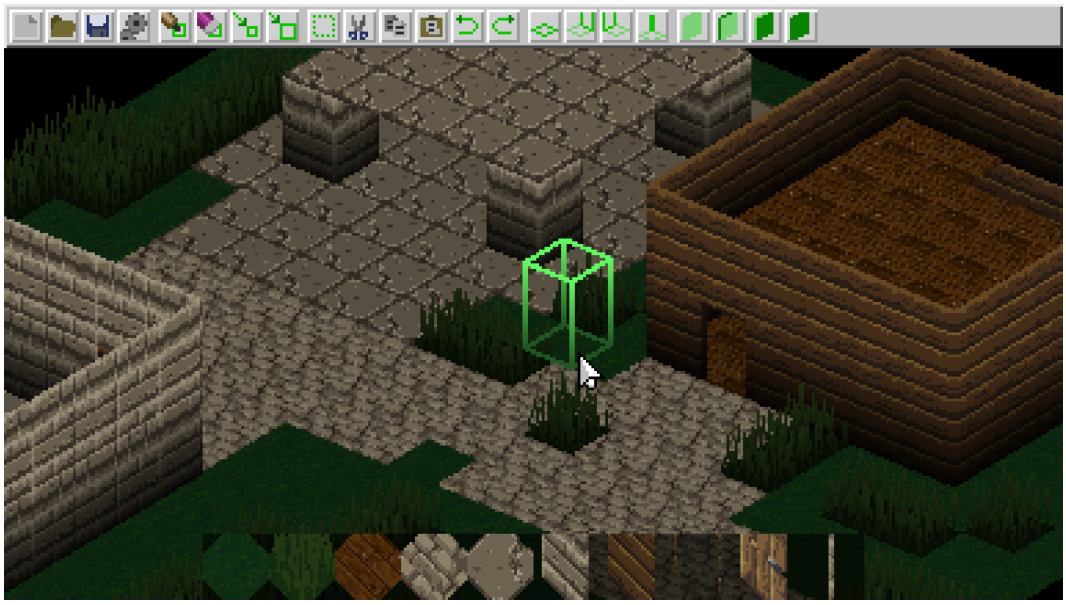
Kuva 1. Käyttötapakaavio.

Kuvassa 1 on esitetty käyttötapakaavio, josta ilmenee pelimoottorin ja Lua-kerroksen rajapinta sekä käyttäjän asema toiminnassa. Käyttötapakaavion ”Pelitila” sisältää kaikki pelimoottorin sisäiset resurssit joita tarvitaan pelitilan esittämiseen peli-ikkunassa. Näihin kuuluu mm. kartta, käyttöliittymä ja aktiiviset objektit. ”Asetukset” sisältää pääasiassa renderöintiasetukset, kuten ikkunan koon, käytettävän resoluution, kuvasuhteen sekä kuvakulman. Lisäksi se sisältää virkistystaajuuden sekä vaihtoehtoiset tiedostopolut datakansioon ja ”boot.lua”-tiedostoon. ”Funktionaaliset Lua-tiedostot” sisältävät kaikki varsinaisen pelikoodin sekä mm. tiedostojen latausmetodit, kun halutaan jakaa data moduuleihin. ”Resurssit” kuvaa sekä datatiedostoina käytettyjä Lua-tiedostoja, kuten tilet ja kartat, sekä kuvatiedostoja ja ääni- sekä musiikkitiedostoja.

3.3 Piirtotekniikka

Suurin yksittäinen ominaisuus, joka on tehnyt oman pelimoottorin kehittämisestä varteenotettavamman vaihtoehdon, on käytetty piirtotekniikka. Piirtämällä tekstuureja toistensa päälle yhden pikselin erotuksella y-tasossa, saadaan aikaa vaiku-

telma kolmiulotteisesta tilasta. Metodia voisi pitää eräänlaisena primitiivisenä vokselirenderöintinä, jolla saadaan aikaan varsinaista vokselirenderöintiä muistuttava lopputulos. Haittapuolena tällä tekniikalla on se, että joudutaan tekemään tuhansia piirtokutsuja, mikä aiheuttaa kuormaa prosessorille. Tällöin saadaan selvää hyötyä käyttämällä laiteläheistä ohjelmointikieltä kuten C-kieltä, jolla pelimoottorin keskeisimmät toiminnot, kuten renderöinti on tehty.



Kuva 2. Pelimoottorilla tehty karttaeditori.

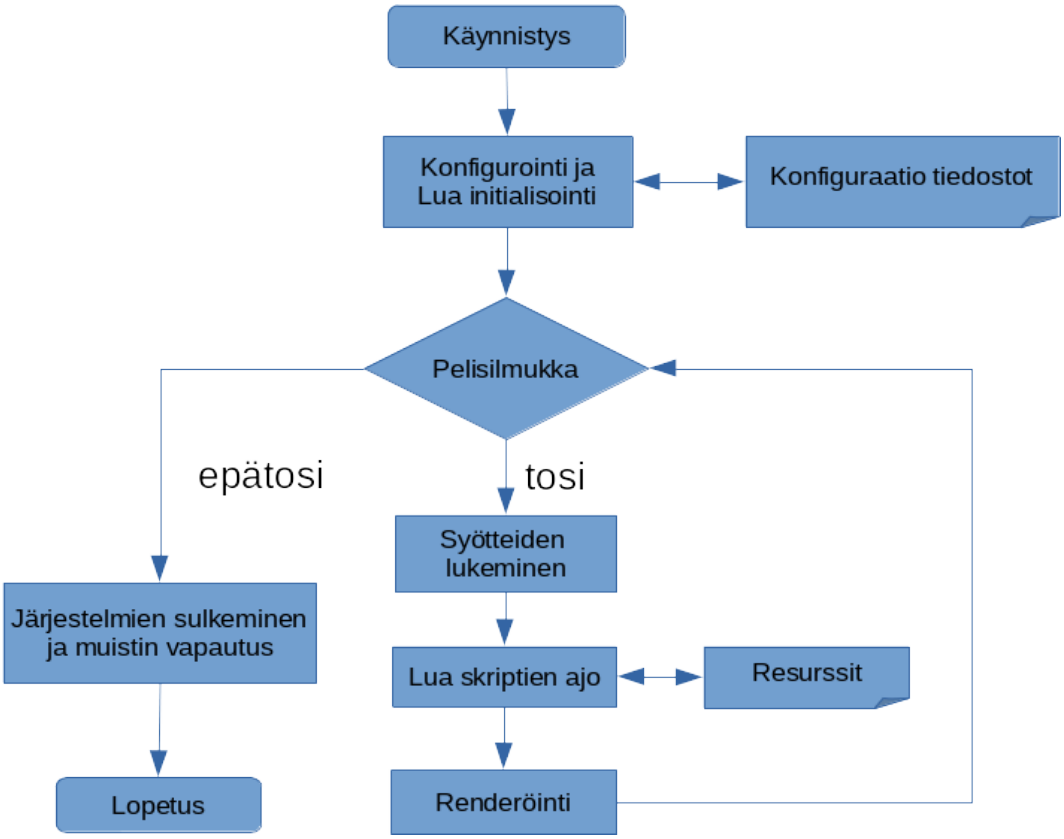
Kartta voidaan renderöidä isometrisenä tai military perspektiivistä, mutta muitakin kuvakulmia olisi mahdollista toteuttaa. Kuvasta 2 nähdään tekniikan lopputulos isometrisesta kulmasta piirrettynä. Lopputulos jäljittelee ulkonäöltään lähinnä 90-luvun alkupuolen DOS-pelejä ja pieni resoluutio on käytännössä vaatimus tälle piirtometodille. Suuremmilla resoluutioilla tämä metodi kävisi kohtuuttoman raskaaksi, sillä piirtokutsujen määrä nousisi eksponentiaalisesti. Tuhansien piirtokutsujen takia tekniikka on käytännössä kuitenkin mahdollinen nykyisilläkin tietokoneilla vain laitteistokiihdytettynä, eikä toimi hyvin pelkällä ohjelmistopohjaisella renderöinnillä.

Koska pelimoottorilla on tarkoitus tehdä pelejä, jotka toimivat vanhemmallakin tietokoneilla, käyttää se useita optimointitekniikoita vähentämään laitteistokuormaa ja piirtokutsuja. Näistä lisää myöhemmissä kappaleissa.

4 TOTEUTUS

4.1 Yleiskuva

Pelisilmukka noudattaa yleistä mallia, jossa ensin luetaan syötteet, jonka jälkeen pelitilanne päivitetään ja lopuksi suoritetaan renderöinti.



Kuva 3. Toimintaperiaate.

Pelimoottorin kaikki konfiguraatio-, skripti- ja datatiedostot ovat Lua-tiedostoja. Lua-skripteillä ohjataan pelimoottorin objekteja, joihin kuuluu:

- Tilet
- Valot
- Aktiiviset objektit
- UI
- Fontti
- Teksti

- Canvas
- Äänet

Objektit varastoidaan muistiin yksinkertaisiin taulukkorakenteisiin ja niihin viitataan indeksiarvolla. Nämä objektien rakenteet huolehtivat muistinhallinnasta lisättäessä uusia objekteja, sekä suljettaessa vapauttavat kaikki varaamansa muistit.

```
int SE_ui_element_add( UIElement *element ) {
    UIManager *uim = se_res->ui_manager;
    int i = 0;

    while ( uim->elements[i] != NULL ) {
        i++;
    }
    /* When slot is found, put element there. */

    element->ID = i;

    uim->elements[i] = element;
    /* Assume first entry. */
    if ( uim->first == NULL ) {
        uim->first = uim->elements[i];
        uim->last = uim->elements[i];
    }
    else {
        uim->last->next = uim->elements[i];
        uim->elements[i]->last = uim->last;
        uim->last = uim->elements[i];
    }

    if ( i == uim->size ) {
        uim->size++;
    }
    /* Allocate more space for elements */
    if ( uim->size == uim->alloc_size ) {
        uim->alloc_size += OBJECT_ALLOC_SIZE;
        uim->elements = realloc( uim->elements, uim->alloc_size * sizeof( UIElement* ) );
    }

    return i; /* return pos in array. */
}
```

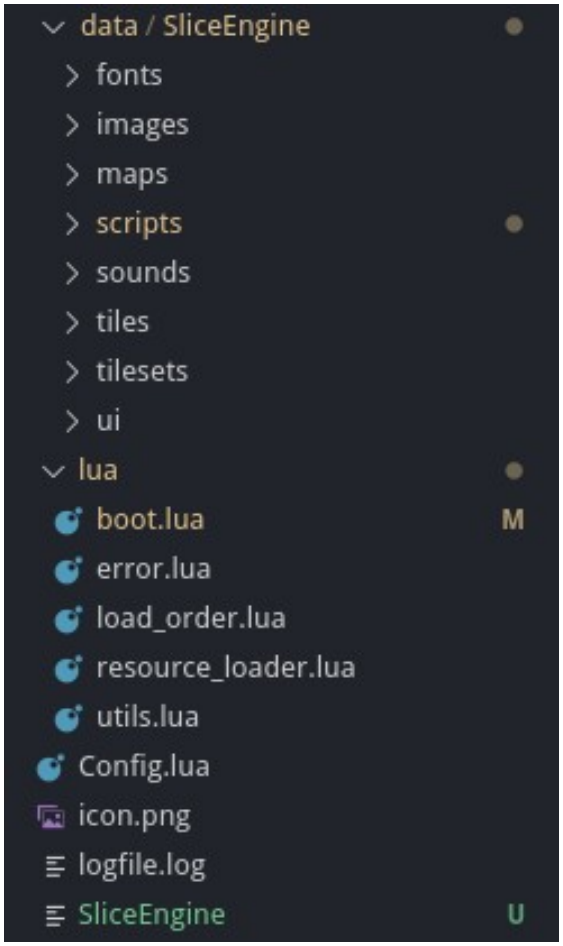
Kuva 4. Esimerkki UI-elementin lisäävästä funktiosta.

Kuvassa 4 on esimerkki UI-elementin lisäämisestä taulukkoon. Funktio etsii objektille tyhjän paikan ja palauttaa löydetyn paikan indeksin. Tällä metodilla uudet objektit voidaan lisätä minne tahansa tyhjään kohtaan taulukkoa, jos objekteja on poistettu. Mikäli objekti asetetaan viimeiseen paikkaan, varataan taulukolle lisää muistia. Kaikilla objektitaulukoilla on samankaltainen toteutus.

```
typedef struct {
    bool running;
    bool is_map_loaded;
    SDL_Window *window;
    iVec2 win_size;
    SDL_Renderer *renderer;
    Textures *textures;
    Map *map;
    Camera *cam;
    Lua *lua;
    InputManager input_manager;
    ObjectManager *object_manager; /* 3d objects in world. */
    UIManager *ui_manager;
    SFManager *spritefont_manager;
    AudioManager *audio_manager;
    DrawManager *draw_manager;
    Profiler profiler;
} EngineResources;

EngineResources *se_res;
```

Kuva 5. EngineResources struct.

Pelimoottorin resurssit kuuluvat kuvassa 5 näkyvään globaaliin muuttujaan, josta niihin voidaan viitata mistä tahansa ohjelman koodista yhdestä muistiosoitteesta.



Kuva 6. Pelimoottorin kansiorakenne, jossa SliceEngine on käynnistystiedosto.

Kuvan 6 kansiorakenteesta nähdään ”lua”-kansion sisältämät pelimoottorin toiminnan kannalta tärkeimmät Lua-tiedostot, jotka tullaan lataamaan ensimmäisenä. Kaikki datatiedostot sijaitsevat datakansion alla, johon ne voidaan jäsentää halutulla tavalla, esimerkiksi omiin moduuleihinsa.

4.2 Lua-rajapinta

Lua on toteutettu sen kirjastoissa virtuaalikoneena lua_State, johon viitataan kaikissa sen funktioissa yleensä vakiintuneella lua_State *L muuttujalla. Lua-skriptit on yhdistetty C-funktioihin sen kirjastoon kuuluvalla lua_register funktiolla.

```
lua_register( L, "SE_is_key_pressed", SE_linput_is_key_pressed );
lua_register( L, "SE_is_key_held", SE_linput_is_key_held );
lua_register( L, "SE_is_mouse_pressed", SE_linput_is_mouse_pressed );
lua_register( L, "SE_is_mouse_held", SE_linput_is_mouse_held );
lua_register( L, "SE_is_mouse_released", SE_linput_is_mouse_released );
lua_register( L, "SE_is_mouse_moving", SE_linput_is_mouse_moving );
lua_register( L, "SE_get_mouse_pos", SE_linput_get_mouse_pos );
lua_register( L, "SE_get_mouse_rel_pos", SE_linput_get_mouse_rel_pos );
lua_register( L, "SE_get_mouse_scroll_x", SE_linput_get_mouse_scroll_x );
lua_register( L, "SE_get_mouse_scroll_y", SE_linput_get_mouse_scroll_y );
```

Kuva 7. Lua-funktioiden yhdistäminen C-funktioihin.

Kuvassa 7 on esimerkki Lua-funktion yhdistämisestä C-funktioon funktio-osoittimella. Kuvan esimerkissä on yhdistetty syötteiden funktioita, kuten Lua-funktio SE_is_key_pressed, joka kutsuu C-funktion SE_linput_is_key_pressed funktion. Lua käyttää rajapintaan pino-toteutusta, jossa mm. argumentit tuodaan virtuaalikoneen pinoon, josta ne voidaan lukea C-funktiossa. Myös funktioiden mahdolliset palautusarvot asetetaan tähän pinoon.

```
-- This function will be called by the engine.
function gameloop( delta, inputs )
    if ( inputs ) then
        if ( SE_get_text_input() == -1 ) then
            mouse_input()
            hotkey_input()
            move_camera( delta )
        end
        if ( SE_is_key_pressed( CONTROLS_CONF.key_quit ) ) then
            SE_quit()
        end
    end
end
```

```
int SE_linput_is_key_pressed( lua_State *L ) {
    if ( lua_isstring( L, -1 ) ) {
        InputManager *im = &se_res->input_manager;

        if ( im->key_just_pressed[ SDL_GetScancodeFromName( lua_tostring( L, -1 ) ) ] ) {
            lua_pushboolean( L, true );
            return 1;
        }
        else {
            lua_pushboolean( L, false );
            return 1;
        }
    }
    return 1;
}
```

Kuva 8. Lua- ja C-funktio.

Jokainen C-ohjelmassa esiintyvä Lua-funktio palauttaa int-muuttujan sekä ottaa argumenttina lua_State-osoitinmuuttujan eli Lua-virtuaalikoneen. Lua-pinosta voi-

daan lukea negatiivisella indeksillä argumentteja pinon päältä alaspäin, kun halutaan esimerkiksi viitata viimeisimpään pinoon laitettuun arvoon. Funktion int- palautusarvo kertoo palautusarvojen määrän. Pelimoottori käyttää funktioiden nimissä etuliitettä ”SE” ja Lua API-funktioiden etuliite on ”lua”. Ohjelma kutsuu kuvan 8 Lua-funktiota ”gameloop” jokaisella pelikierrolla ja käyttää argumentteina delta-aikaa, joka on aika edellisen ja nykyisen pelikierron välillä, sekä lähettää mahdolliset syötteet inputs-muuttujassa. Syötteidenhallintaa tarkastellaan tarkemmin ”Syötteidenhallinta” kappaleessa.

```
int SE_lua_run_gameloop( double delta ) {
    lua_State *L = se_res->lua->lua_state;
    lua_getglobal( L, "gameloop" );

    if ( lua_isfunction( L, -1 ) ) {
        lua_pushnumber( L, delta );
        SE_input_push_inputs( L );

        if ( lua_pcall( L, 2, 0, 0 ) != 0 ) {
            sprintf( se_error_mes, "Lua error. %s", lua_tostring( L, -1 ) );
            SE_console_message( se_error_mes, SE_MESSAGE_PRINT );
            return -1;
        }
    }
    else {
        sprintf( se_error_mes, "Error. No Lua gameloop found!" );
        SE_console_message( se_error_mes, SE_MESSAGE_PRINT );
        return -1;
    }
    lua_pop( L, -1 );

    return 0;
}
```

Kuva 9. Gameloop-kutsu.

Kuvassa 9 nähdään gameloop-funktion kutsu C-koodista. SE_input_push_inputs-funktio asettaa syötteet Lua-pinoon ja lua_pcall-funktio kutsuu Lua-kerroksen gameloop-funktiota näillä argumenteilla. Funktiokutsu tapahtuu suojatussa tilassa (protected call), jolloin mahdollisista koodivirheistä saadaan käyttäjälle hyödyllistä tietoa.

4.3 Käynnistys ja konfigurointi

Käynnistettäessä ensimmäisenä alustetaan SDL ja Lua, jonka jälkeen konfiguroidaan pelimoottori. Pelimoottori käyttää konfigurointi- ja datatiedostoina Lua-tiedostoja ja datatiedostot ovatkin yksi Luan suuri vahvuus.

```
ENGINE_CONF = {
    -- If paths are not set, defaults are used.
    --data_path = "",
    --lua_boot_path = "",
    -- 0 = SDL_RENDERER_SOFTWARE
    -- 1 = SDL_RENDERER_ACCELERATED
    -- 2 = SDL_RENDERER_PRESENTVSYNC
    renderer = 1,
    win_scale = 4,
    -- resolution = { 160, 120 },
    resolution = { 384, 216 },
    -- resolution = { 320, 200 },
    -- resolution = { 426, 240 },
    -- 0 = SDL_WINDOW_FULLSCREEN
    -- 1 = SDL_WINDOW_FULLSCREEN_DESKTOP
    -- 2 = SDL_WINDOW_BORDERLESS
    -- 3 = SDL_WINDOW_RESIZABLE
    window = 3,
    display = 0,
    -- 0 = Military
    -- 1 = Isometric
    projection = 1,
    fps = 60,
    -- 0 = Ignore aspect
    -- 1 = Keep aspect
    aspect = 1,
    show_fps = 0
}
```

Kuva 10. Pelimoottorin konfigurointi-tiedosto.

Kuvasta 10 voidaan nähdä tiedosto, jonka pohjalta pelimoottori tekee tärkeimmät peruskonfiguraatiot käynnistysvaiheessa. Käyttäjä voi halutessaan määritellä datatiedostopolun ”data_path”, josta datatiedostot tullaan lataamaan. Lisäksi voidaan määritellä Lua-käynnistystiedosto ”lua_boot_path”, jonka pelimoottori kutsuu en-

simmäisenä Lua-tiedostona ja josta muut Lua-järjestelmät voidaan käynnistää. Näistä molemmat ovat vapaaehtoisia määrityksiä ja mikäli niitä ei ole erikseen asetettu, käyttää pelimoottori vakiopolkuja ”data/” ja ”lua/boot.lua”, jotka ovat suhteessa käynnistystiedostoon kuvan 6 mukaan. Muita asetuksia, jotka eivät ole välttämättömiä pelimoottorin toiminnalle, voidaan ladata myös käyttäen Lua-tiedostoja.

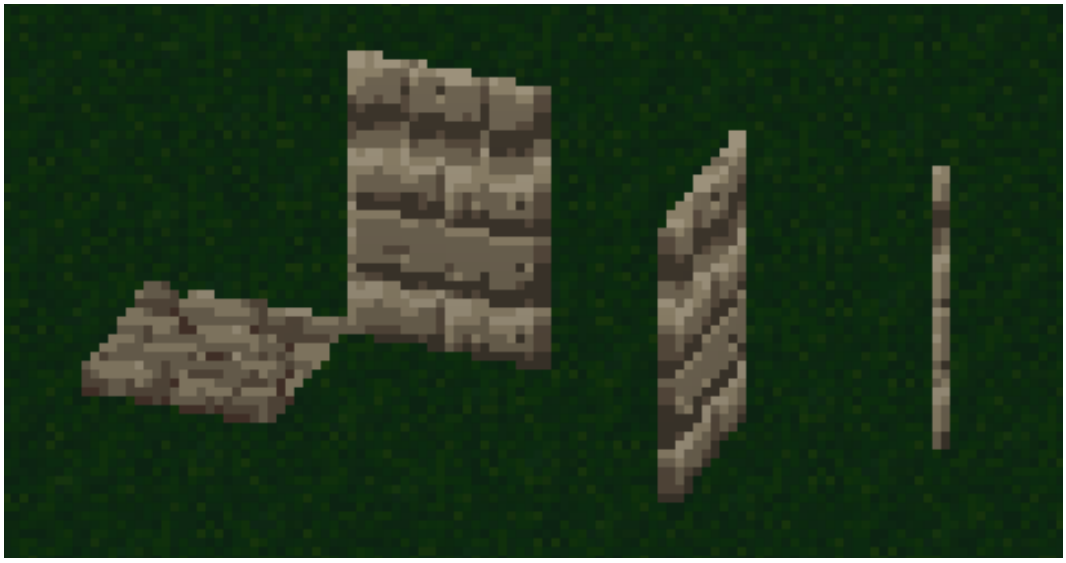
```
--Bits from SDL_GetMouseState
CONTROLS_CORE = {
    mouse_left = 1,
    mouse_right = 4,
    mouse_middle = 2
}

-- Controls
CONTROLS_CONF = {
    mouse_primary = CONTROLS_CORE.mouse_left,
    mouse_secondary = CONTROLS_CORE.mouse_right,
    mouse_pan = CONTROLS_CORE.mouse_middle,
    cam_move_up = "w",
    cam_move_down = "s",
    cam_move_left = "a",
    cam_move_right = "d",
    cam_rotate_right = "e",
    cam_rotate_left = "q",
    cam_floor_up = "r",
    cam_floor_down = "f",
    cam_mouse_pan = "Left Shift",
    cam_move_speed_keyboard = 250,
    cam_rot_speed_keyboard = 100,
    cam_move_speed_mouse = 1,
    cam_rot_speed_mouse = 25,
    mouse_scroll_speed = 6,
    key_quit = "escape"
}
```

Kuva 11. Kontrollien konfiguraatiotiedosto.

Kuvassa 11 on esimerkki Luan sisäisesti käytetystä konfiguraatiotiedostosta, josta voidaan nähdä esimerkiksi kuvassa 8 käytetty näppäin ”CONTROLS_CONF.key_quit”.

4.4 Kartta

Pelimoottorin kartat ovat ruudukkopohjaisia (en. Tile based) ja näitä ruudukkoja voi olla myös useampina kerroksina. Näille ruuduille on olemassa vakiintunut termi ”tile” ja niiden koostamaa rakennetta kutsutaan termillä ”tilemap”. Kartta koostuu neljästä tilemapista, jotka on nimetty pelimoottorissa, SE_GRID_TILEMAP, SE_WE_WALL_TILEMAP, SE_NS_WALL_TILEMAP ja SE_CORNER_WALL_TILEMAP.



Kuva 12. Neljä eri tilemäppiä.

Oman tilemapin käyttäminen seinille selkeyttää kartan rakennetta ja esimerkiksi valon vaikutus lasketaan kaikille tilemapille eri tavalla. Yhteen ruutuun voidaan myös piirtää monta tilea.

Kuva 13. Kerroksen korkeus.

Jokainen kerros koostuu 32 viipaleesta eli tasosta, joihin piirretään kaksiulotteinen tekstuuri. Käytännössä kartan yksi tile koostuu tilavuudeltaan 24 * 24 * 32 vokselista. Kuvasta 13 voidaan havainnoida tämä piirtotekniikan antama illuusio kolmiulotteisesta rakenteesta ja päätellä, että seinien piirtäminen on moninkertaisesti vaativampi prosessi kuin lattian. Optimoinnissa tulee kiinnittää tähän erityistä huomiota.

Kuva 14. Lähdetekstuuri.

Kuvassa 14 nähdään esimerkki lähdetekstuurista, josta voidaan käyttää mitä tahansa suorakulmion muotoista osaa piirrettäessä tilen viipaleet.

```
return {
    texture = "images/tiles/wooden_wall01.png",
    draw_height = 32,
    center = { 12, 0 },
    floor_height = 32,
    occluders = { "0", "0", "0", "0" },
    pf_weight = 1,
    shade = "tiles/shades/wall01.lua",
    src = {
        { 0, 31, 24, 1 },
        { 0, 30, 24, 1 },
        { 0, 29, 24, 1 },
        { 0, 28, 24, 1 },
        { 0, 2, 24, 1 },
        { 0, 1, 24, 1 },
        { 0, 0, 24, 1 }
    },
    dst = {
        { 0, -1, 24, 2 },
        { 0, -1, 24, 2 },
        { 0, -1, 24, 2 },
        { 0, -1, 24, 2 }
    }
}
```

Kuva 15. Esimerkki tile-tiedostosta. Tiedostoa on typistetty kuvaan sopivaksi.

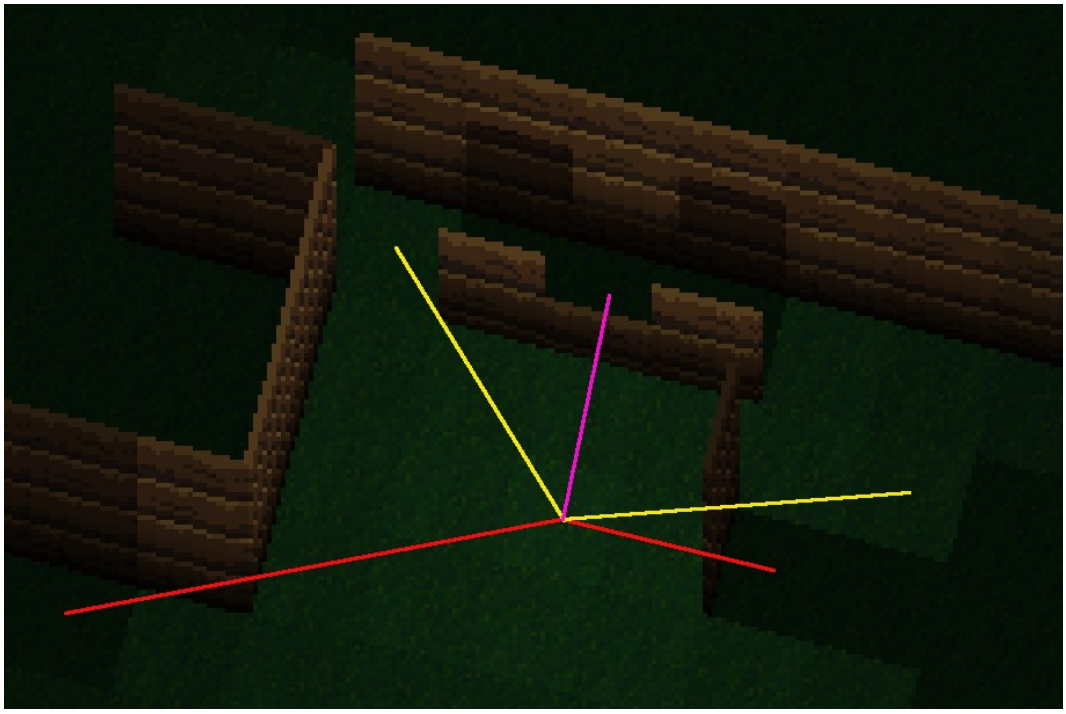
Kuvassa 15 on esimerkki tile-tiedostosta, joka sisältää mm. nämä mainitut suorakulmiot src (source) lähdetekstuuri ja dst (destination) kohde. Suorakulmio on kuvattu lua_table tyypillä { x, y, leveys, korkeus }. Kuvasta voidaan havaita, että lähdesuorakulman korkeutta on venytetty kaksinkertaiseksi (Huomaa, että tiedoston x- ja y-koordinaatit eivät vastaa kuvan 14 tekstuuria). Nämä suorakulmiot tallennetaan pelimoottorin sisällä SDL_Rect-muuttujaksi, jota voidaan käyttää suoraan SDL-renderöintifunktioissa SDL_RenderCopyEx().

Kuva 16. Megatekstuuri viiden ruudun kokoisena.

Kuvasta 16 nähdään pelimoottorin sisällä megatekstuuriksi kutsutut tekstuurit, jotka on tässä kuvassa pienennetty havainnollistamisen helpottamiseksi viiden tilen kokoiseksi. Yksittäiset tilet piirretään ensin megatekstuuriin, jotka sitten piirretään jokaisella pelikierrolla peli-ikkunan tekstuuriin yhden pikselin poikkeamalla y-tasossa. Näin voidaan välttää jokaisen tilen piirtäminen joka pelikierrolla ja päivitetään megatekstuurit ainoastaan tarpeen mukaan, esimerkiksi kameraa liikutettaessa. Mikäli arvioidaan kuvan tilannetta, tarvitaan vain 32 piirtokutsua sen sijaan, että tehtäisiin 5 * 5 * 2 + 12 * 32 = 434 piirtokutsua lattialaattojen ollessa kaksi pikseliä korkeita. Pelkän piirtokutsun lisäksi pelimoottorin täytyy tarkistaa jokaisen tilen ja jokaisen viipaleen data paitsi tilet, jotka on merkitty tyhjiksi.

4.5 Valaistus

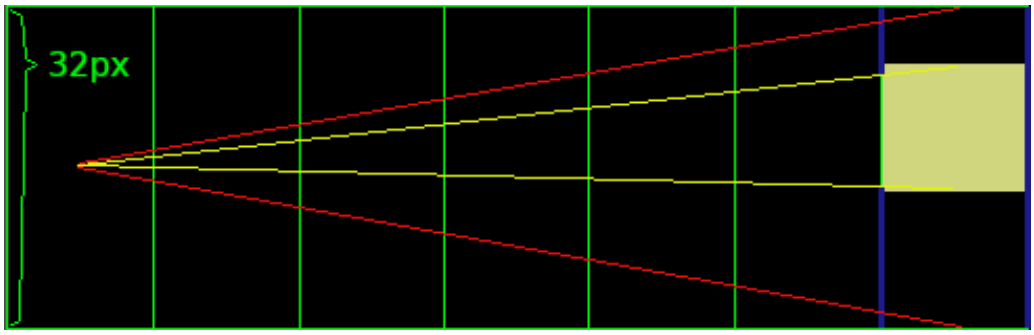
Valaistus toimii tile-pohjaisesti käyttäen säteensuuntausta (en. ray casting). Valonlähteenä toimivasta tilesta testataan ympärillä olevat ruudut mahdollisilta esteiltä, kuten seiniltä, ja katsotaan saavuttaako valo testattavan tilen. Laskettu valaistus tallennetaan kartan lightmap-arvoihin, jotka sisältävät 32 SDL_Color-tyypin muuttujaa jokaiselle tilelle. Ligtmappeja on kaksi, baked ja dynamic. Baked on tarkoitettu suurimmalle osalle kartan valaistuksesta, joka ei muutu tai muuttuu harvoin ja dynamic on tarkoitettu usein päivittyville valoille. Näiden lisäksi on olemassa pohjavalaistus ”Ambient light”, joka on SDL_Color-tyypin muuttuja, joka määrittää koko kartan pohjavalaistuksen. Renderöintivaiheessa lopullinen valaistus määrittyy näiden valoarvojen jokaisen kanavan korkeimmasta arvosta, joka on 8-bittinen RGBA-arvo.



Kuva 17. Valaistustestaukset. Punainen viiva on ei valoa, keltainen viiva on täysi valaistus ja violetti viiva on osittainen valaistus.

Kuvassa 17 havainnollistetaan tilen valaistustestausta. Kuvasta nähdään ettei tilen valaistus ole ainoastaan binäärinen arvo, vaan jokainen kerroksen viipale voi olla valaistu myös erikseen. Koska kerroksen korkeus on 32 yksikköä, voidaan käyttää 32-bittistä muuttujaa valaistuksen maskina. Testaus tehdään aina ylhäältä katsottu-

na tilen keskeltä, mutta valonlähteen korkeutta voi muuttaa. Koska valonlähde testataan aina tilen keskeltä, voidaan sen toteutuksessa soveltaa esimerkiksi viivan piirtoalgoritmia ja siihen onkin käytetty pohjana orthogonaalista viivan piirtoesimerkkiä /11/.

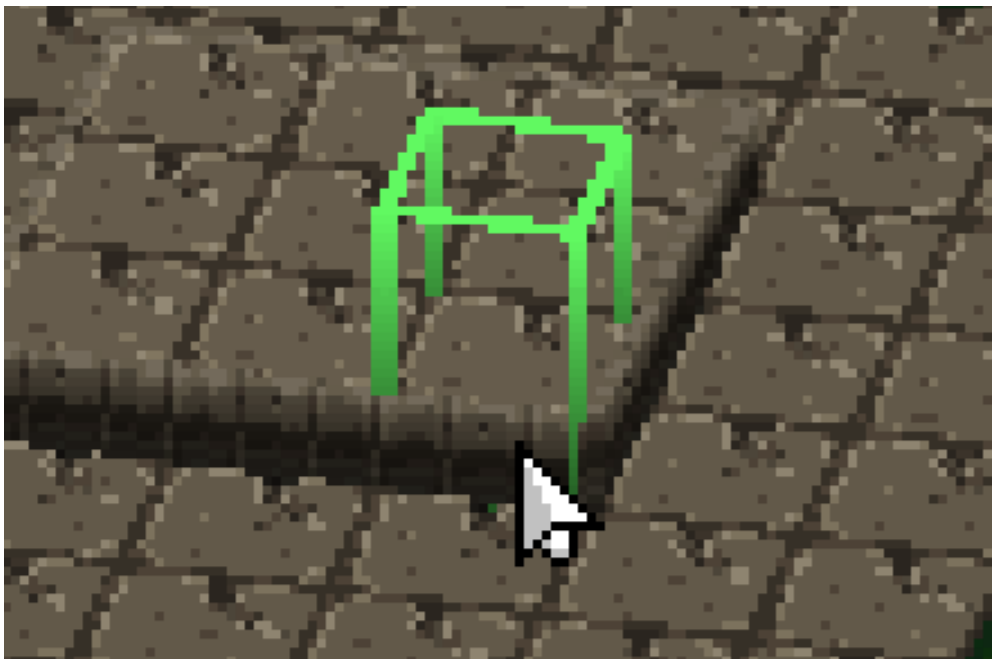


Kuva 18. Valonsäteet sivusta kuvattuna, jossa siniset viivat kuvastavat seiniä ja keltainen alue valaistua aluetta.

Kuvasta 18 voidaan nähdä oikeanpuolimmaisen tilen osittainen valaistuminen ja seinien langettamat varjot.

4.6 Aktiivisetobjektit

Aktiivisetobjektit piirretään samaan aikaan kuin kartan megaslicet, joka tapahtuu jokaisella pelikierrolla.



Kuva 19. Aktiivinenobjekti.

Kuvasta 19 nähdään 3d-kursorin sulautuminen karttaan, sillä kartan megatekstuurit piirretään limittäin objektien viipaleiden kanssa. Objektit sisältävät maksimissaan 32 viipaletta ja niistä jokainen voidaan piirtää tekstuurin eri osista käyttäen omaa väriä ja kulmaa. Muutettaessa näitä arvoja, voidaan objektit animoida reaaliaikaisesti.

4.7 Käyttöliittymä

4.7.1 Käyttöliittymäelementti

Käyttöliittymäelementit muodostavat graafisen käyttöliittymän. Elementtien piirtojärjestystä voidaan muuttaa helposti, sillä ne voivat viitata toisiinsa linkkilistan avulla. Mikäli elementti halutaan piirtää päällimmäiseksi, voidaan se siirtää listassa viimeiseksi. Myös hiiren painallukset tarkistetaan tässä järjestyksessä ja ainoastaan päällimmäisen elementin klikkaus rekisteröityy.

Kuva 20. Esimerkki nine patch-textuurista, jossa näkyy tekstuurin osat värikoodattuna.

Elementti voi sisältää yhden tekstuurin, josta voi piirtää haluamansa osan joko suorakaiteena tai nine patchinä. Nine patch metodi piirtää elementin kuvan 20 mukaisesti käyttäen halutun kokoista suorakaidetta. Tämä on erityisen hyödyllinen mm. ikkunan piirtämiseen, jossa minkä tahansa kokoisen elementin reunat piirtyvät samanlaisina. Lisäksi elementti voi sisältää tekstielementin ja canvaselementin. Näihin elementteihin viitataan niiden indeksillä ja ne piirretään elementin mahdollisen pohjatekstuurin jälkeen.

4.7.2 Bittikarttafontti

Bittikarttafonttia (en. Spritefont) käytetään tekstin kirjoittamiseen tavallisen fontin tapaan.

Kuva 21. Fontin tekstuuri.

```
return {
    tex = get_resource_path( "images/fonts/font.png" ),
    charset = " !\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOP
    char_size = { 6, 8 }
}
```

Kuva 22. Fontin lua-tiedosto.

Kirjainsetti määritellään kuvan 22 tiedoston mukaisesti ”charset” muuttujalla. Tämän järjestyksen ja määritetyn kirjaimen koon perusteella voidaan kuvan 21 tekstuurista piirtää oikea kirjainta vastaava symboli lähtien vasemmalta oikealle rivi kerrallaan. Formaatti tukee 32-bitin merkkejä.

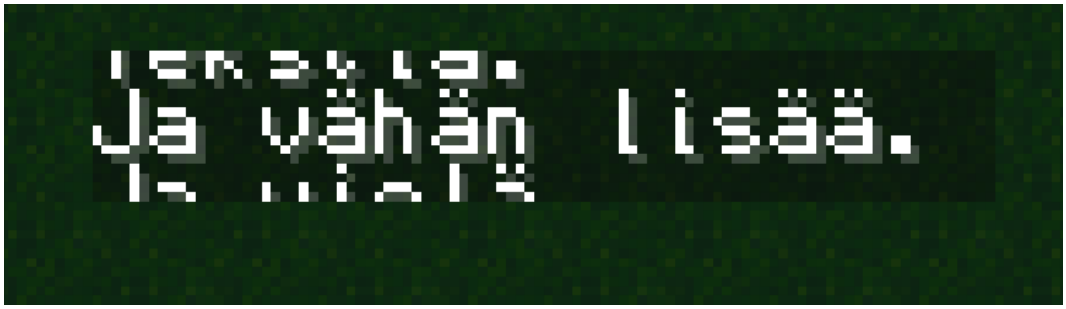
4.7.3 Teksti

Käyttöliittymän tekstielementti piirtää tekstiä käyttäen bittikarttafonttia. Tekstille määritellään mm. koko muistissa sekä suorakaiteen muotoinen piirtoalue.

```
typedef struct {
    Uint32 c; /* Character supporting unicode size characters. */
    SDL_Color color;
} SFChar;

typedef struct {
    int ID;
    int font;
    SDL_Rect rect; /* Used as virtual texture rect. Draw only part of chars in the edge. */
    iVec2 pos; /* Scroll position in px. */
    iVec2 spacing;
    iVec2 text_size; /* Size of allocated text size.
        Note that this needs to be only visible rendered text size.*/
    SFChar **text; /* Characters in memory. If resizable text area,
        should be the max size. */
    iVec2 cur; /* Last cursor pos. Mainly to be used with append. */
} SFText;
```

Kuva 23. SpriteFont char- ja text structit.

Tekstistä voidaan piirtää pikselin tarkkuudella haluttu kohta. Lisättävälle tekstille voidaan määritellä väri ja se voidaan kirjoittaa alkuun tai jatkoksi loppuun. Kuvassa 23 on tekstin kirjoittamiseen käytetyt structit.



Kuva 24. Esimerkki tekstin renderöinnistä.

Teksti renderöidään uudestaan jokaisella pelikierrolla kirjain kerrallaan ja se rajoittuu kuvan 24 mukaisesti, sille määritellylle alueelle.

4.7.4 Canvas

Canvas on käyttöliittymäelementti, johon voidaan piirtää tekstuureja, tekstiä tai grafiikkaprimitiivejä, kuten pisteitä, viivoja ja suorakulmia. Lisäksi toinen canvas voidaan piirtää osaksi tai kokonaan toiseen canvaselementtiin.

Kuva 25. UI Canvas esimerkki.

4.8 Syötteidenhallinta

Syötteiden lukemiseen käytetään SDL-funktiota SDL_PollEvent, josta luetaan event.type-arvoja, kuten SDL_QUIT, SDL_KEYDOWN, SDL_KEYUP, SDL_MOUSEMOTION, SDL_MOUSEBUTTONDOWN.

```
#define KEY_COUNT 322

typedef struct {
    Sint32 x;
    Sint32 y;
    Sint32 last_x;
    Sint32 last_y;
    Sint32 rel_x;
    Sint32 rel_y;
    Sint32 scroll_x;
    Sint32 scroll_y;
    Uint32 buttons;
    Uint32 buttons_clicked;
    Uint32 buttons_released;
    bool moving;
    int over_ui; /* -1 = false and else, is id of element */
} MouseState;

typedef struct {
    bool key_pressed[KEY_COUNT];
    bool key_just_pressed[KEY_COUNT];
    MouseState mouse_state;
    bool has_input; /* If has any input. Can be used in Lua to handle input code. */
    int text_input_target; /* ID of current input spritetext. */
} InputManager;
```

Kuva 26. Input structit.

Näiden luettujen arvojen perusteella päivitetään kuvassa 26 näkyvien structien tilaa ja näitä arvoja voidaan lukea Lua-skripteillä.

```
while ( SDL_PollEvent( &event ) ) {
    switch ( event.type ) {
        case SDL_QUIT:
            se_res->running = false;
            return;
        case SDL_KEYDOWN:
            im->key_pressed[event.key.keysym.scancode] = true;
            im->key_just_pressed[event.key.keysym.scancode] = true;
            im->has_input = true;

            break;
        case SDL_KEYUP:
            im->key_pressed[event.key.keysym.scancode] = false;
            im->has_input = true;
            break;
```

Kuva 27. Näppäinsyötteiden lukeminen SDL_PollEvent-funktiolla.

Pelimoottorissa on kaksi vaihtoehtoa lukea syötteitä Lua-koodista. Ensimmäinen keino on tarkistaa syötteen tila Lua-rajapinnasta erillisellä funktiolla, kuten nähtiin kuvassa 7. Toinen keino on tuoda kaikki syötteet kerralla Lua-ympäristöön pelimoottorista Lua-table muodossa. Näin voidaan merkittävästi vähentää C-kutsuja Lua-ympäristöstä, koska syötteitä tullaan lukemaan useissa paikoissa Lua-koodissa jokaisella pelikierrolla. Lisäksi tämä Lua-tablen arvo on nil, mikäli mitään syötteitä ei ole. Tämän tiedon avulla voidaan ohittaa huomattava määrä koodia, joka käsittelee syötteitä.

```
void SE_input_push_inputs( lua_State *L ) {
    InputManager *im = &se_res->input_manager;
    iVec2 pos = SE_input_get_mouse_win_pos();
    dVec2 rel_pos = SE_input_get_mouse_win_rel_pos();

    if ( im->has_input ) {
        lua_createtable( L, 0, 3 ); /* inputs. 3 is for mouse, keyboard, text_input */
        /* mouse */
        lua_createtable( L, 0, 11 );

        lua_pushinteger( L, pos.x ); /* Pushes table value on top of Lua stack */
        lua_setfield( L, -2, "x" );
        lua_pushinteger( L, pos.y );
        lua_setfield( L, -2, "y" );
        lua_pushnumber( L, rel_pos.x );
        lua_setfield( L, -2, "rel_x" );
        lua_pushnumber( L, rel_pos.y );
        lua_setfield( L, -2, "rel_y" );
        lua_pushinteger( L, im->mouse_state.scroll_x );
        lua_setfield( L, -2, "scroll_x" );
        lua_pushinteger( L, im->mouse_state.scroll_y );
        lua_setfield( L, -2, "scroll_y" );
        lua_pushinteger( L, im->mouse_state.buttons );
        lua_setfield( L, -2, "held" );
        lua_pushinteger( L, im->mouse_state.buttons_clicked );
        lua_setfield( L, -2, "pressed" );
        lua_pushinteger( L, im->mouse_state.buttons_released );
        lua_setfield( L, -2, "released" );
        lua_pushboolean( L, im->mouse_state.moving );
        lua_setfield( L, -2, "moving" );
        lua_pushinteger( L, im->mouse_state.over_ui );
        lua_setfield( L, -2, "over_ui" );

        lua_setfield( L, -2, "mouse" );
```

Kuva 28. Hiiren syötteiden kirjoittaminen Lua-tableen.

Kuvassa 28 nähdään tämän input tablen valmistelu, joka viedään Lua-ympäristöön kutsuttaessa gameloop-funktiota C-koodista.

4.9 Äänet ja musiikki

Pelimoottori käyttää äänikirjastona SDL_mixer-kirjastoa. Kirjasto tukee useita ääniformaatteja ja sillä voi soittaa yhtä musiikkikanavaa ja useita äänikanavia samanaikaisesti. Normaalin äänentoiston lisäksi pelimoottori tukee karttaan lisättäviä paikallisia ääniä, jotka ottavat kameran paikan ja kulman huomioon sekä toistaa ääniä stereona ja vaimentaa niitä paikan mukaan. Äänet voidaan myös määritellä ryhmiin, jolloin niiden asetuksia, kuten äänenvoimakkuutta voidaan helpommin säätää. Musiikki- ja äänikanavilla on kaikki tärkeimmät asetukset, kuten pause, stop, resume ja äänenvoimakkuus.

5 JOHTOPÄÄTÖKSET

Ennen projektin aloittamista oli epäilys, että työtaakka olisi kohtuuton tämän tyyliselle toteutukselle, mutta kokemus on ollut erittäin positiivinen ja tulokset ovat olleet hyviä. Järjestelmät on ollut mahdollista kehittää haluamakseen ja pelimoottorin toiminta on erittäin yksityiskohtaisesti kehittäjänsä ymmärtämä, joten mahdolliset muutokset on suhteellisen vaivatonta toteuttaa. Työhön on liittynyt monia haasteita ja se on kehittänyt tekijänsä ammatillista osaamista. Myös tämän dokumentin kirjoittamisessa on ollut oma haasteensa määriteltäessä aluerajaukset.

On ollut odotettavaakin, että monia toteutettuja ominaisuuksia on poistettu niiden osoittautuessa turhiksi. Näihin kuuluu mm. omat datatiedostoformaatit sekä käyttöliittymän tekstielementin tekstinmuokkaus. Näistä molemmat on paljon helpompi toteuttaa Lua-koodissa, jolloin C-toteutuksia voidaan huomattavasti suoraviivaistaa. Tämä lähestymistapa antaa lisäksi käyttäjälle enemmän vapauksia määritellä toiminta haluamakseen, sillä käyttäjällä on pääsy Lua-kerrokseen.

Kehitystä tullaan vielä jatkamaan ja esimerkiksi kartan renderöinti ei ole vielä lopullisessa muodossa. Lisäksi joitain ominaisuuksia, kuten reitinhaku puuttuu vielä kokonaan ja se on tarkoitus toteuttaa A* algoritmilla. Virheenhallinnassa on parantamisen varaa, erityisesti Lua-rajapinnassa, jossa käyttäjä kutsuu pelimoottorin funktioita. Yksikkötestaus jätettiin tietoisesti pois, sillä se olisi hidastanut ohjelman kehittämistä ja osoittautunut monesti ylimääräiseksi työksi, koska monia toimintoja on poistettu tai niiden toimintaa muutettu huomattavasti. Kuitenkin, ohjelman koodin vakiinnuttua, testaus voidaan mahdollisesti vielä liittää projektiin ja se lisäisi myös ohjelman luotettavuutta.

LÄHTEET

/1/ About SDL. Viitattu 12.3.2020. https://www.libsdl.org/

/2/ Description. Viitattu 12.3.2020. https://www.libsdl.org/projects/SDL_image/

/3/ Description. Viitattu 12.3.2020. https://www.libsdl.org/projects/SDL_mixer/

/4/ Introduction. Viitattu 12.3.2020. https://www.lua.org/manual/5.3/manual.html

/5/ Home of the MinGW.org and MSYS Projects. Viitattu 17.4.2020. http://mingw.org/

/6/ About Cmake. Viitattu 17.4.2020. https://cmake.org/overview/

/7/ What is GDB? Viitattu 17.4.2020. https://www.gnu.org/software/gdb/

/8/ Gitin verkkosivut. Viitattu 17.4.2020. https://git-scm.com/

/9/ GCC, the GNU Compiler Collection. Viitattu 8.5.2020. https://gcc.gnu.org/

/10/ Clang: a C language family frontend for LLVM. Viitattu 8.5.2020. https://clang.llvm.org/

/11/ 2.1 Orthogonal steps. Viitattu 8.5.2020. https://www.redblobgames.com/grids/line-drawing.html