TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut
Võrgutarkvara õppetool

Strateegiamäng SDL2 vahenditega

bakalaureusetöö

Üliõpilane: Kristof Mikael Rosenberg
Üliõpilaskood: 112238 IAPB
Juhendaja: Jaagup Irve

Tallinn
2014

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Töö eesmärk on luua C++ abil Linuxi peal töötav SDL2 teeki kasutav programm, mis utiliseerib SDL-i poolt pakutavaid võimalusi graafika ning audio presenteerimiseks. Peab looma SDL2-e abil realiseeritavad klassid, mis tüüpilisi kasutajaliidese loomise juures vajalikke ülesandeid täidavad. Kõik elemendid ei tohi alati alati nähtavad, kuvatavat hulka peab saama muuta. Töö käigus luuakse mäng, kuid mängudisain pole antud töö puhul kõige olulisem.

Kasutajaliidese loomiseks on tihti vajalik realiseerida erinevad akna kujundused ehk stseenid, mida saab vahetada vastavalt vajadusele programmi töö käigus. Stseen koostatakse väiksematest graafilistest elementidest: piltidest ning tekstist. Realiseerima peab ka graafiliste elementide kasutamine nuppudena, mille abil kasutaja saab programmi tööd suunata. Programm peab teatud hetkedel, tihti stseeni muutusel, mängima helisid.

Tulemusena said loodud mitmed stseenid, mille vahelist vahetust realiseeris ühtne põhiline mootor. Programmi töö käigus vahetatakse stseene, mis omavad erinevaid graafilisi elemente. Kõik elemendid pole alati nähtavad ning mõned neist töötavad kui klikitavad nupud. Kasutaja sisendite tagasisidena või teatud stseenide muutumisel mängiti helisid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 42 leheküljel, 8 peatükki, 12 joonist, 4 tabelit.

Abstract

The aim of this work is to create a program in C++ which works on a Linux platform using the SDL2 library. The program has to utilise the possibilities provided by SDL to create a visual user interface. For that there have to be classes that would help create typical user interface elements systematically. Not all graphical elements can be visible permanently, it must be possible to change their visibility. A playable game will be created, but game design is not the main emphasis here.

User interfaces often work by using several sets of graphical elements which could be called „scenes“. They are composed of several displayable components and they can be switched between during runtime. Graphical elements should also be capable of functioning as clickable buttons to be used to divert the course of the program according to the user’s wishes. The program must play sounds at certain points.

As a result, several scenes were created. The current scene can be switched with another by the program’s main engine. Scenes incorporate different graphical elements and not all of them are always displayed. Some of the elements function as clickable buttons. As a feedback for the user inputs and change of stages, sounds are played.

The thesis is in Estonian and contains 42 pages of text, 8 chapters, 12 figures, 4 tables.

Lühendite ja mõistete sõnastik

SDL	***Simple DirectMedia Layer*** Teek, mis pakub programmi ning operatsioonisüsteemi vahelist vahekihti nii, et sama programmikood töötab erinevates toetatud keskkondades.
SDL2	***SDL2.0.3*** SDL uuendus, mida levitatakse zlib litsentsiga.
WAV	***Waveform Audio File Format*** Failiformaat, milles saab hoiustada audiot.
API	***Application Programming Interface*** Liides, mille abil saab realiseerida programmide vahelist suhtlust.
PNG	***Portable Network Graphics*** Kiire ning kompaktne pildiformaat, mis toetab ka läbipaistvat tausta.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

SDL2-e kasutatakse kõiksugu programmide tegemisel, kuid tihti koos mõne muu teegiga, et luua kasutajaliidest. SDL2 küll pakub 2D graafika loomiseks tuge, kuid SDL teegi põhirõhk pole sellel. Tuleb luua teatud klassid, et lihtsustada tüüpilisi kasutajaliidese realiseerimiseks vajaminevaid protsesse. Nendeks võib olla näiteks teksti kuvamine kusagil ekraanil, pildi muutmine vastavalt programmi käigule.

Seda kõike tehakse eesmärgil luua kasutajaliides lihtsale strateegiamängule.

Sarnaseid graafilisi elemente peab saama luua, hoiustada ning kuvada süstemaatiliselt ning polümorfismi ära kasutades. Nendele elementidele peab saama lisada ka kasutajaliidese realiseerimiseks vajalikke funktsionaalsusi näiteks mõned elemendid peavad töötama kui klikitavad nupud. Samuti peaksid elemendid kuuluma teatud stseeni raamidesse ning stseenide peab saama vahetada nii programmi enda kui ka kasutaja sisendite tõttu. Sisendite vastusena peab kasutajale mängima ka teatud helisid.

Luuakse hulk klasse, mis hõlpsustavad luua kuvatavaid kujutisi. Need klassid omavad ühtset baasklassi, millesse suhtutakse, kui nende liidesesse. Seda liidest utiliseerides saab iga stseen oma graafilisi elemente hoiustada konteinerites ning lasta kuvada teadmata, missuguse elemendiga tegelikult tegu on. Stseene saab vahetada vastavalt kasutaja sisenditele, milleks on klaviatuuri klahvidele vajutamine või hiirega klikkimine. Sisukate sisendite puhul mängitakse helisid.

Kõigepealt esitatakse tehnoloogiline ülesehitus, mis kirjeldab valitud tehnoloogiaid ning nende vajalikkust. Seejärel antakse ülevaade programmis kasutusel olevatest põhiobjektidest, nii strateegiamängu realisatsiooniga seotud klassidest kui ka graafilise poole realisatsiooniks vajalikest objektidest. Järgmisena selgitatakse graafiliste elementide lahendustest ning siis ülevaade tulemusest.

1. Tehnoloogiline ülesehitus

Töö keskmeks on SDL2 API vahendid. Kuigi SDL2 kasutatakse tihti, kuna ta suudab töötada erinevatel platvormidel ning operatsioonisüsteemidel, töö koostamise käigus piirdutakse ühe operatsioonisüsteemiga, milleks sai Ubuntu 14.04 (Trusty Tahr). Töö on kirjutatud C++ keeles, kasutades mõningaid C++11 standardi võimalusi.

1.1 SDL2 valik

SDL2 on teek, mille eesmärk on pakkuda programmi ning seda jooksutava süsteemi (operatsioonisüsteemi) vahelist kihti. [1] Seda teeki levitatakse Zlib litsentsiga. [2] SDL-i kasutades võib programmeerija kirjutada koodi, mis ei sõltu keskkonna eripäradest. SDL2 ise tuvastab, missugusel süsteemil ta käivitati, ning teab, kuidas oma põhiülesandeid täita.

SDL2 toetab ka OpenGL tehnoloogiat, mille abil saaks luua keerulisemat graafikat, kuid see pole antud töö raames vajalik ning OpenGL-i kasutamine sõltub siiski operatsioonisüsteemist märgatavalt rohkem, kui SDL2-e pakutavad vahendid. [3]

Ametlikult toetab SDL järgmisi operatsioonisüsteeme: Windows, Mac OS X, Linux, iOS ja Android. [1]

Töö eesmärgiks pole küll teha programm, mis töötaks mitmel platvormil, vaid utiliseerida teeki, mida kasutades on võimalik sama koodi käivitada erinevatel platvormidel. 2014 aasta veebruaris toimunud Steam Dev Days üritusel rääkis tarkvaraarendaja Ryan Gordon sellest, kuidas teha algust tarkvaraarendusega Linuxis. [4] Kui vaadelda Windowsi peal töötavat programmi, mida tahetakse käivitada mõnes Linuxi keskkonnas, siis kõige töömahukamad sammud on SDL2-e ning OpenGL-i kasutusele võtmine. Nende tehnoloogiate implementeerimise käigus pole üldse tähtis operatsioonisüsteemi vahetada, vaid algsel süsteemil saavutada sama funktsionaalsus, mis eksisteeris enne SDL-i kasutusele võtmist. Tulemuseks on programmikood, mida võib käivitada erinevatel platvormidel ilma koodis muudatusi tegemata ehk ei leidu mitte ühtegi otsepöördumist süsteemi enda API poole.

Antud töö käigus ei utiliseerita SDL2-e multiplatvormsuse võimalusi, vaid kasutatakse ära tema kõiksugu võimalusi ühel operatsioonisüsteemil. Sinna alla kuulub põhiliselt SDL versioon 2.0 uuendusega loodud SDL_Texture objektide kasutamine, mida saab otse ekraanile

kuvada. Enne SDL2-e tulekut oli kõik kuvatav SDL_Surface klassides; neid hoiustati arvuti mälus, mitte videokaardi mälus. Seega SDL_Texture kasutamine aitab parandada jõudlust ning utiliseerib mälu paremini. [5]

1.2 SDL2 lisapakettide valik

SDL2-ga koos pakutakse lisakomponente, mis ei sisaldu algses SDL2-e põhipaketis. Antud töö käigus kasutatakse järgmisi: SDL_image, SDL_ttf, SDL_mixer.[1]

1.2.1 SDL_image

SDL_image pakub võimaluse kuvada PNG formaadist mällu laetud pilte ekraanil. SDL2-e põhipaketti kasutades saab pildifaile kuvada ainult kasutades *bitmap image file* formaati. PNG on kokkupakitud ning seega failisuurused kujunevad tavaliselt mõistlikult väikseteks. [6] Ainukene probleemne ala PNG kasutamise juures on animatsioonide realiseerimine. [7] Kuna kuvamiseks kasutatakse selle töö käigus vaid staatilisi kujundeid, pole põhjust muud formaati valida.

Projekti käigus kasutatud kuvandid on kõik hoiustatud PNG formaadis.

1.2.2 SDL_ttf

SDL_ttf eesmärk on kasutada etteantud *font* faili, et sellest luua kuvatav SDL_Texture, mida on võimalik SDL2-e põhipaketi poolt pakutavate funktsioonide kaudu ekraanil näidata. SDL_ttf kasutamise juures tuleb märkida, et muutuva teksti kasutamine pole optimiseeritud, kuna kuvatav objekt tuleb igal muutusel uus genereerida. SDL_ttf konstrueerib SDL_Surface objekti, mis tuleb muuta ümber SDL_Texture objektiks kasutades selleks SDL2-e funktsioone, et parandada jõudlust.

1.2.3 SDL_mixer

SDL_mixer annab võimaluse kasutada ning manipuleerida helisid. [8] Helid jagatakse kanalitesse, mida saab peatada ajutiselt, lõplikult või jätkata. SDL_mixer pakub küll palju võimalusi, kuid selle programmi raamis pole muud vaja, kui audiojuppide mängimine kindlatel hetkedel vastavalt sündmusele ning taustamuusika pidev mängimine.

Kõik audiofailid on WAV formaadis.

[1] Kasutatavad lisapaketid küll kõik töötavad SDL2 versioonidega, kuid kõigil pole uuendatud dokumentatsiooni, kuna API ei muutunud uue versiooni tulekuga.

1.3 Operatsioonisüsteemi valik

SDL2 toetab Linuxit ning C++ töötab sellel tihti kiiremini kui Windowsi peal. Linuxi platvorm kogub populaarsust, kuna järjest rohkem tuntud programme pakuvad ka selle platvormi tuge. Mänguritele võib tähtsaks sammuks lugeda Valve Software firma poolt arendatud Steami Linuxi versiooni välja andmine. Linuxi peal töötavate mängude nimekiri on üha kasvanud ning Steam väidab, et aprillis aastal 2014 kasutas 1,26% kasutajatest põhiliselt mingisugust Linuxit. [7] Kuna kõige populaarseim neist oli Ubuntu 14.04, millel on pikaajaline tugi, sai programmi arendamiseks kasutatavaks operatsioonisüsteemiks valitud just see.

1.4 C++11 kasutamine

SDL on kirjutatud keeles C ning seda saab utiliseerida kasutades mitmeid programmeerimiskeeli, kaasa arvatud C ja C++. [1] Selle töö tegemiseks sai valitud C++.

C++11 võimaldab kasutada mõningaid uusi meetodeid, kuid kompilaatorid ei võimalda algsätteid kasutades C++11 võimalusi utiliseerida, seega tuleb kompileerimisel täpsustada standard. Kasutusel olid näiteks std::to_string() ning std::chrono.

1.5 Akna suuruse valik

Tehnoloogilisest vaatevinklist pole vahet, missugust resolutsiooni kasutada, kuid kuna käesoleva töö eesmärgiks pole luua erinevatele resolutsioonidele sobivaid digitaalvarasid ning kujutise venitamise tulemusena saab pildi visuaalne tase märgatavalt kahjustatud, on mõistlik valida kindel resolutsioon ning töötav programm teha eeldades seda akna suurust. Kuna enamus monitoridest toetavad resolutsiooni 1280x720 ehk 720p-d, tundub see sobiv valik. Ekraani suuruse muutmist kasutajal selle töö raames ei lubata.

2. Programmi sisuline ülesehitus

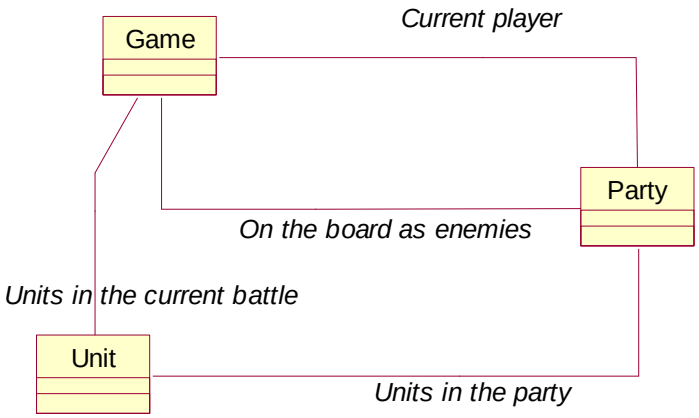
2.1 Üldplaan

Tegemist on strateegiamänguga, kus mängija saab liikuda ruudulisel maailmakaardil, mis on piiritletud seintega, sisaldab rünnatavaid vastaseid, kuid enamjaolt on tühi läbitav ruum. Nii mängija kui ka kõik vastased koosnevad ühest kuni kuuest üksusest, mida saab lahingu käigus utiliseerida. Lahingute käigus tuleb vastaste üksused hävitada, endal peab vähemalt üks üksus ellu jääma. Lahingu kaotamisel tuleb maailmakaarti uuesti alustada. Pärast lahingut pakutakse mängijale võimalust üht oma üksust tugevdada.

Kogu mäng võidetakse, kui kõige tugevam vastane (ingl. k. „Boss“) hävitatakse. See tähendab mängukaardi edukat läbimist.

2.2 Põhiobjektid

Kasutusel on kolm omavahel seotud järgnevat mudelit: Mäng (ingl. k. „Game“), Grupp (ingl. k. „Party“) ning Üksus (ingl. k. „Unit“). Vt „Joonis 1 Mudelite klassidiagramm“, lk 14.



Joonis 1 Mudelite klassidiagramm

2.2.1 Mäng

Mängu ülesanne on olla üldine viide kõigile teistele mudelitele. 2-dimensiooniline massiiv tähistab mänguväljakut, kus on viiteid gruppidele ning viite asukoha järgi massiivis teab ka

selle asukohta maailmakaardil. Samuti on otseviide mängija enda grupile ning teatakse selle asukohta massiivis.

Kui algatatakse lahing, täidetakse Mängu objekti küljes olev üksuste konteiner, mille kaudu lahingustseen saab viited lahingus osalevate üksuste jaoks.

2.2.2 Grupp

Üldjuhul on grupp lihtsalt üksuste kogum, kuid kasutusel on grupi tüüpi määrav enumeratsioon. Tüübi õige määramine kindlustab selle, et kuvatakse maailmakaardil ruut õige illustreeriva pildiga, läbimatuid alasid ei läbita, kuid vastaseid saab rünnata.

Grupi tüübid koos inglise keelse vastega: Mängija („Player“), Vastane („Enemy“), Tugevaim vastane („Boss“), Läbimatu ala („Border“).

2.2.3 Üksus

Üksus võtab osa lahingutest ning omab selleks vajalikke väärtusi, mille põhjal koostatakse vastase üksuse ründamiseks koguründetugevus ning kogukaitsetugevus (vt „Tabel 1 Üksuse atribuudid seletustega“, lk 15). Ründamise puhul kaotavad nii kaitsjad kui ka ründajad hetketugevust vastavalt ründetugevustele ning kaitsetugevustele.

Tabel 1 Üksuse atribuudid seletustega

Eesti keeles	Inglise keeles	Sisu
Hetketugevus	Current Strength	Ründamistugevuse üks määrajatest. Peab olema mittenegatiivne. Väärtusel 0 loetakse üksus „surnuks“ ning ei saa enam rünnakuid läbi viia
Kogutugevus	Strength	Hetketugevus ei saa sellest suuremaks. Küll aga on võimalik tuua hetketugevus sellega võrduma.
Tüüp	Type	Enumeratsiooni kohaselt on tegemist kas üksusega, mis ründab ainult esimeses reas vastaseid („Melee“) või saab rünnata kõiki vastaseid („Ranged“). Esimese rea suremisel saab kõiki rünnata.

Eesti keeles	Inglise keeles	Sisu
Ründetugevuse kordaja	Attack	Koguründetugevuse arvutamiseks vajalik kordaja.
Kaitsetugevuse kordaja	Defence	Kogukaitsetugevuse arvutamiseks vajalik kordaja.
Mudeli nimi	Model Name	Viitab õigele PNG formaadis pildile failisüsteemis ning määrab teksti, mis presenteeritakse kui selle üksuse nimi.

Üksust on võimalik tugevdada neljal viisil:

1) suurendada ründamistugevuse kordajat;
2) suurendada kaitsetugevuse kordajat;
3) viia praeguse tugevuse väärtus maksimaalse tugevuse väärtuseni;
4) lisada konstantne väärtus hetke- ning kogutugevusele.

2.3 Graafilised elemendid

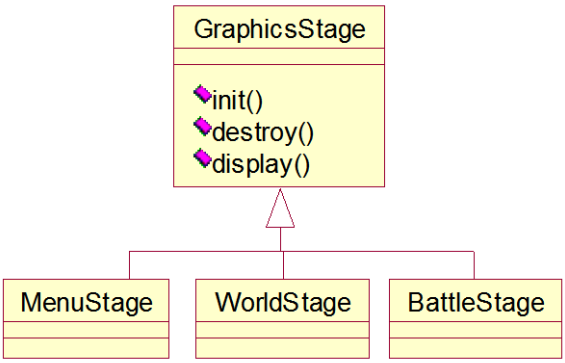
Graafiliste elementide disainimise juures tuli lähtuda kahest põhilisest punktist:

1) neid peab saama säilitada, kuvada ning hävitada kasutades polümorfismi ehk peab eksisteerima üldine liides, baasklass („Element“);
2) neid peab saama luua võimalikult lihtsalt ning lühidalt.

Lähemalt graafiliste elementide struktuurist ning realisatsioonist vt „Graafilised elemendid“, lk 28.

2.4 Stseenid

Stseenid kõik laiendavad üht põhiklassi (ingl. k. vaste „GraphicsStage“). See klass defineerib liidese, mille kaudu saab põhimootor iga stseeni võimalusi utiliseerida. Tähtsad operatsioonid on kuvamine (ingl. k. „display“), loomine (ingl. k. „init“ sõnast „initialize“) ning hävitamine (ingl. k. „destroy“). Vt „Joonis 2 Stseenide klassidiagramm“ lk 17.



Joonis 2 Stseenide klassidiagramm

Programmil on alati olemas „käesolev stseen“, mida põhitsükkel kuvab ning millele sisendid saadetakse sõelumiseks. Läheb ka tarvis „eelmine stseen“ muutujat, mis viitab õigele eelmisele stseenile, et saaks seda uuesti kasutada ilma, et ei lõhuks seda eelnevalt ega looks seda uuesti. Mitme stseeni mälus hoidmine võib küll mälu mõistes kalliks osutuda, kuid niimoodi saab laadimisajalt kokku hoida.

Stseenide vahetust korraldab põhimootori klass. Tüüpjuhtudel käseb ta eelmise stseeni hävitada, loob ning algatab uue stseeni samal ajal kuvades „Laadin“ ekraanipilti ning seejärel laseb põhitsüklil edasi minna. Vahetust käivitab mõne sisendi töötlemine stseeni poolt, mille tulemusena tagastatakse vajalik sündmuse liik (ingl. k. „Event Type“) põhimootorile.

Hiirekliki sündmuse liigi tuvastamiseks tuleb stseenil läbi käia elementide konteinereid ning proovida, kas käesoleva elemendi pindala katab klikkimise koordinaadid x ja y. Kui katab, siis tuleb küsida elemendi käest „On Click Event Type“ muutuja, mis on eelnevalt sellele määratud. Tegemist on enumeratsiooniga, mille väärtustele põhimootor oskab reageerida vastavalt vajadusele. Mõne elemendi puhul on mõistlik sama klikk edastada elemendile, et see ise tagastatava sündmuse tüübi välja selgitaks (näiteks üksust tähistava elemendi korral).

Stseenide kuvamisel tuleb järgida põhimõtet, et hiljem lisatud elemendid jäävad kõige peale, seega peab taustapildi kuvama kõige esimesena ning klikkimise sündmust tuvastades tuleb itereerida kuvamisele vastupidises suunas.

2.4.1 Menüü

Menüüstseen kuvab hulga graafilisi tekste, mis töötavad kui klikitavad menüü esemed (vt „Tabel 2 Menüüs kuvatavad valikud“ lk 18). Neil klikkides suunatakse programmi tööd mõne muu stseeni poole. Vt „Joonis 3 Menüü“, lk 18.

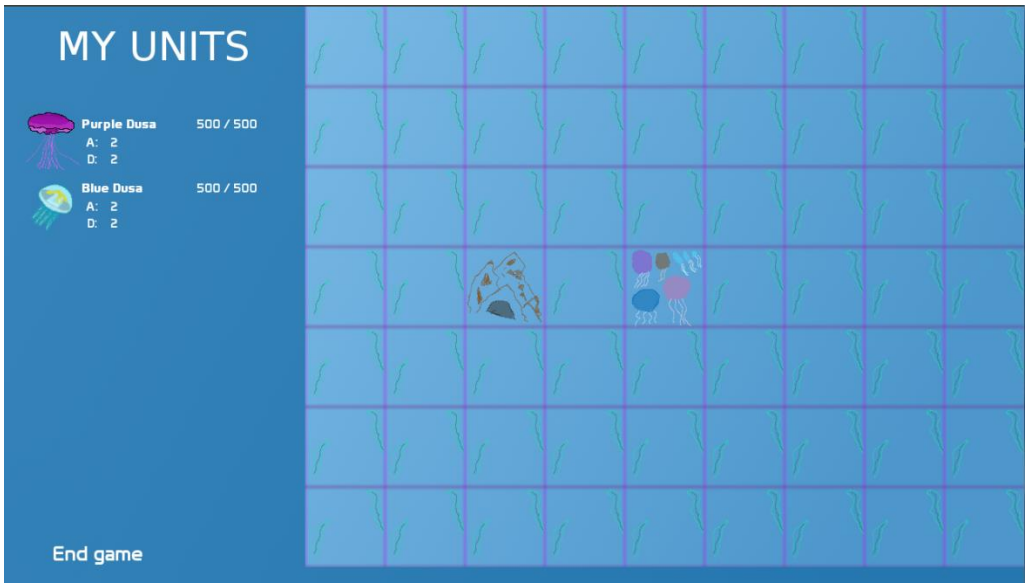
Tabel 2 Menüüs kuvatavad valikud

New game	Muudab käesoleva stseeni maailmastseeniks.
Start a random battle	Alustab ettemääratud lahingustseeni proovimise eesmärkidel.
Exit	Lõpetab programmi töö, vabastab ressursid.

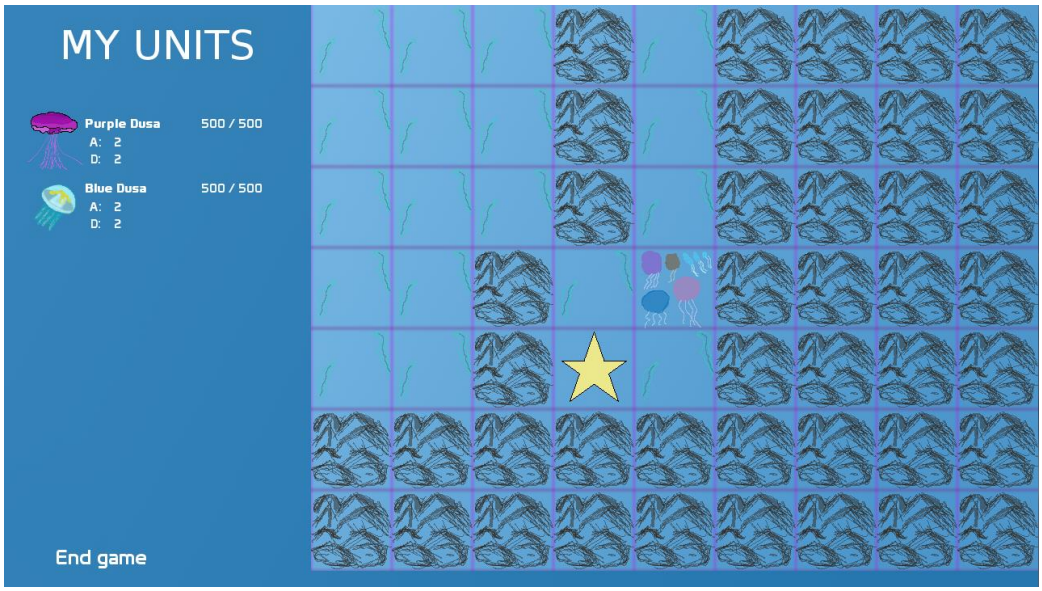
Joonis 3 Menüü

2.4.2 Maailmastseen

Stseeni käigus on võimalik akna vasakul poolel näha oma üksuste praegust olukorda. Kui on selleks võimalus, siis kuvatakse ka nupud võimalike tugevdamiste jaoks igale üksusele. Allpool leidub nupp, millele klikkides lõpetatakse maailm ning minnakse tagasi menüüstseeni. Põhiline osa ekraanist on täidetud ruudulise kaardiga, mille keskmes on mängija ning välja on joonistatud tema lähiümbrus. On selgelt aru saada tühjad ehk läbitavad ning vastase või läbimatu kiviga ruudud. Vt „Joonis 4 Maailmastseen“, lk 19 ja „Joonis 5 Maailmastseen tugevaima vastase juures“, lk 19.



Joonis 4 Maailmastseen



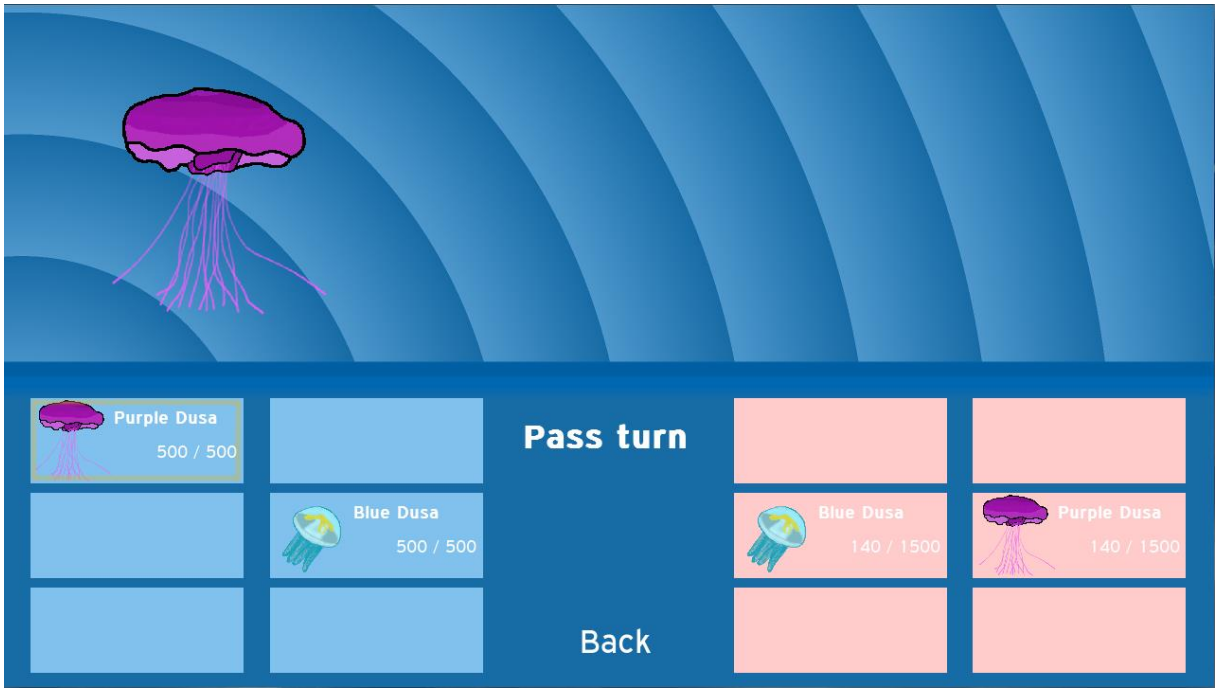
Joonis 5 Maailmastseen tugevaima vastase juures

Mängija saab muuta oma asukohta maailmakaardil ning kui liikumine toimub mõne vastase peale, alustatakse lahingut. Stseen muutub lahingustseeniks.

Maailmastseeni käigus ruudul liikumine käib kasutades klaviatuuri vajutusi. Liikumiseks peab kasutaja vajutama valitud suunale vastavale nooleklahvile, millest automaatselt konstrueeritakse SDL_Event tüüpi objekt. Sellest tuvastatakse vajutatud nupp proovides, kas tegemist on ühe või teise meid huvitava nupuga teades meid huvitavate nuppude SDL_Keycode lipu väärtust. [10] Liikumise tuvastamisel muudetakse mängija asukohta mängu kahemõõtmelises massiivis.

2.4.3 Lahingustseen

Kui lahingustseeni käigus vajutada elusolevale vastase üksusele, siis praegu käija üksus ründab seda sihtmärki. Üldjuhul kaotavad mõlemad üksused elusid vastavalt üksteise koguründe- ning kogukaitsetugevusele. Vastase kõigi üksuste suremise korral likvideeritakse see vastase grupp maailmakaardilt ning antakse mängijale võimalus üks lisakord oma üksusi tugevdada. Kaotuse puhul on maailmakaardi läbimine läbi kukkunud ning kasutaja suunatakse tagasi menüüstseenile.



Joonis 6 Lahingustseen

Lahingustseeni algatamisel luuakse graafilised elemendid, mis tähistavad stseeni ajal kindlaid üksusi (vt „Joonis 6 Lahingustseen“, lk 20). Kuna üksuste jaoks täidetakse graafilistest

elementidest koosnev eraldi konteiner, klikkimise puhul itereeritakse läbi selle konteineri, et tuvastada, missugusel üksusel vajutati. Ründamine toimub vaid siis, kui käija saab rünnata ning vastase üksus on elus. Saab ka valida selle üksusega mitte rünnata hetkel ning ründekord edasi anda.

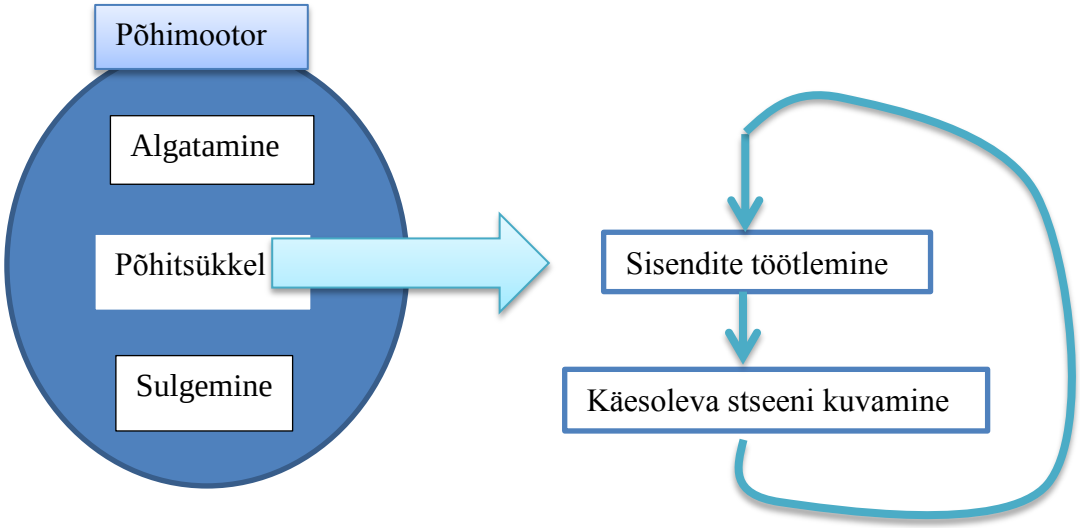
2.5 Audio jupid

Sai loodud klass audio juppide (ingl. k. vaste „Audio Chunk“) mängimise jaoks. Mängimiseks utiliseeritakse SDL_mixeri poolt pakutavaid võimalusi. Igal reaalsel helil, helifailil on oma vastav audiojupi objekt, mille kaudu saab seda kindlat heli mängida.

Helide mängimist korraldab põhimootor, kuna seal on alati ülevaade just läbiviidud sündmusest ning helide mängimine vastavalt käivitatud meetodile on väga otsene realiseerida.

3. Tüüpiline programmi elutsükkel

Esimesena tuleb seadistada põhimootori klass (ingl. k. „Main Engine“). Sedasi luuakse vajalikud SDL võimalused ning objektid, mida saab hiljem kasutada. Seejärel on ettevalmistused tehtud, et programmi põhitöö keskmel olev põhitsükkel käivitada. Tsükkel lõpetatakse, kui soovitakse programm sulgeda. Sulgemisega peab kaasnema kõiksugu ressursside vabastamine. Vt „Joonis 7 Põhimootori töö“, lk 22.



Joonis 7 Põhimootori töö

3.1 Põhimootori algatamine

3.1.1 SDL_Init

SDL_Init(SDL_INIT_EVERYTHING) algatab SDL süsteemi. See on vajalik samm SDL komponentide kasutamiseks. [11] Saab kasutada erinevaid väärtusi või neid kasutada VÕI loogikaga, kuid kasutades *SDL_INIT_EVERYTHING* käivitatakse muude seas ka esmavajalikud süsteemid video, audio ning sündmuste käsitlemiseks.

Kui funktsioon ei tagasta väärtust, mis on suurem kui null, siis võib programmi töö lugeda läbikukkunuks.

3.1.2 SDL_CreateWindow

SDL_CreateWindow loob akna, millel on kindel suurus, asukoht ning teatud võimalused. [12] Akna asukohta määravaks x ning y koordinaadiks võib kasutada SDL-i pakutavat väärtust *SDL_WINDOWPOSITION_CENTERED*. Kõrguseks väärtust 720 ning laiuseks 1280, mis tähendab, et akna resolutsioon on 1280 x 720 ehk 720p. Lisaks kasutame sisuliselt tühja lippu *SDL_WINDOW_SHOWN*, sest teised lipud pole vajalikud antud olukorras.

Kui funktsiooni lõppemisel pole SDL_Window tüüpi objekti tekkinud, siis võib programmi töö lugeda läbikukkunuks.

3.1.3 SDL_CreateRenderer

Funktsiooni eesmärgiks on luua loodud aknale 2D visualisatsiooni konteksti. [9] Tagastatava SDL_Renderer tüüpi objekti viide jäetakse kõigile teistele klassidele vabalt kättesaadavaks staatiliseks muutujaks. Seda objekti kasutatakse, et luua ressursse ning et neid kuvada. Funktsiooni argumente on kolm:

1) SDL_CreateWindow (vt. „SDL_CreateWindow“, lk 23) poolt loodud SDL_Window ehk aken, millega SDL_Renderer siduda;

2) draiveri number, millel visualiseerimine teha. Väärtus „-1“ tähistab, et tuleb võtta esimene lippudega sobiv;

3) Lipud VÕI tehetega seotult.

Lippudeks said valitud *SDL_RENDERER_ACCELERATED*, et kasutataks riistavara kiirendust, ning *SDL_RENDERER_PRESENTVSYNC*, mis piirab näidatavate kaardite arvu sekundis. Virtuaalne sünkronistatsioon (ingl. k. lühendatult „vsync“) tähendab, et kaadrite kuvamine sünkroniseeritakse ekraani tööga – põhiliselt selle värskendussagedusega. Näiteks 60 Hz töötav monitor peaks lubama 60 kaadrit sekundis. Sellise sünkronisatsiooni eesmärk on inimsilmaga nähtavat lõpptulemust natukene parendada.

Meetodi läbikukkumisel ei looda SDL_Renderer objekti, siis võib programmi töö lugeda läbikukkunuks.

3.1.4 IMG_Init

IMG_Init on SDL_image paketiga kaasnev funktsioon, mis algatab selle paketi pakutavad võimalused. [14] Tuleb anda lipud VÕI-tehetega seotult vastavalt sellele, missuguseid formaate tahetakse lugeda. Käesolevas programmis kasutame ainult PNG formaadis faile, seega on vajalik kasutada lippu *IMG_INIT_PNG*.

Tagastatakse kahendsüsteemi väärtuste muster, mis JA tehtega koos lippudega, mida saab hoiustada kui täisarvu tüüpi, peab andma lippudega mittevõrduva kahendväärtuste mustri. Kui aga JA tehte tulemus kattub lippudega, on algatamine läbi kukkunud.

Edukal meetodi läbimise puhul luuakse laadimisekraani kujutis, mida kuvatakse kogu programmi töö käigus aegadell, mil käib üleminek ühelt stseenilt teisele. Seda kujutist kasutatakse ühe kaardi täitmiseks. Seda kaadrit näidatakse kogu ooteaja, see pole eraldi stseen.

3.1.5 TTF_Init

TTF_Init algtab SDL_ttf API võimalused ning tagastab väärtuse „-1“ vea puhul. [15]

3.1.6 Mix_OpenAudio

Avatakse audio tugi, mille abil saab mängida helisid. [8] Avamiseks vajaminevate sisendite väärtused sõltuvad arvutist, mille peal programm käivitatakse. Kuna helisid ei ole palju ning nende ajastamise täpsus ei pea olema nii suur kui mõnel reaalaja märulmängul, valin parema kvaliteedi ning luban langenud jõudlust. Sisendid on järgnevad:

1) Mängitava audio sagedus. Kuigi vaikimisi soovitatakse väärtust „22050“, kasutan siinkohal väärtust „44100“, mida peetakse CD kvaliteeti audio sageduseks.

2) Väljundi formaat, milleks sai valitud vaikimisi pakutud „AUDIO_S16SYS“.

3) Kanalite hulk, milleks sobib antud hetkel väärtus „2“, mis tähistab stereokvaliteediga audiot.

4) Jupi suurus audiopuhvris. See määrab, kui suuri juppe mängitakse tervikuna. Kuna heli pole vaja katkestada programmi poolt, sobib väärtus „4096“, mis tähendab, et üks jupp on 4096 baiti suur.

Meetodi läbikukkumisel tagastatakse väärtus -1, mille puhul võib lugeda programmi töö läbikukkunuks.

Pärast audio algatamist luuakse mängitavad audiojuppide klassid ning käivitatakse lõputultmängiv taustamuusika väikese helitugevusega.

3.2 Põhimootori põhitsükkel

Pseudokoodis kirjutatuna on kogu programmi töö üldiselt järgmine:

```
while(programmJookseb){
        töötleSündmused();
        praeguneStseen->kuva();
}
```

Meetodeid läbides on võimalik tsüklit elus hoidva kahendmuutuja väärtus muuta selliseks, et programm lõpetab tsükli täitmist. Seda eelkõige mõne sündmuse tõttu.

Sündmuste töötlemise käigus ei kulutata nii palju ressursse, kui graafika näitamisel ehk stseeni kuvamisel. Kui aga need kaks tähtsat ülesannet eri lõimedele jätta, peaks tagama, et ühtegi konflikti ei tekiks. Mitmelõimelises programmis võib mõni ressurss muutuda samal ajal, kui seda muudetakse. Järjestikune meetodite läbimine ühe lõime poolt väldib seda.

Sellist järjestikust ülesehitust kasutatakse SDL Wiki koodinäidetes [16], vabalt kättesaadavates õpetustes [17] ning seda on võimalik märgata populaarsete mängude (näiteks Valve Software poolt loodud „Dota 2-e“) kasutamise ajal, kus saab mõne sisendiga kogu graafika peatada mingiks märgatavaks ajaperioodiks.

3.2.1 Sündmuste töötlemine

Soovitatakse kasutada SDL_PollEvent funktsiooni, et töödelda kõiki vahepeal toimunud sündmusi. Sündmused hoitakse järjekorras nii, et saaks eemaldada ühe, see läbi töödelda ning seejärel järgmine võtta. [16] SDL_PollEvent soovib argumendiks viidet SDL_Event tüüpi muutujale, millesse kogu sündmust puudutav informatsioon talletatakse, ning meetod tagastab väärtuste „0“, kui sündmused on otsas. Seega tuleb seda meetodit kasutada tsüklis, mis lõpetatakse hetkel, mil tagastatakse esimene „0“.

Tsükli sees tuleb esimesena selgitada välja sündmuse tüüp. Kui sündmuse tüüp on „SDL_QUIT“, siis on tegemist programmi sulgemisega kasutaja soovil. Tuleb alustada programmi sulgemine ehk ressursside vabastamine.

Antud programmi käigus huvitavad meid põhiliselt sündmuste tüübid „SDL_MOUSEBUTTONUP“ ning „SDL_KEYDOWN“, mille puhul tuleb sündmus edastada käesolevale stseenile töötlemiseks, kuna põhimootoril puudub otsene informatsioon selle kohta, mis järgmisena teha tuleb. Praegune stseen peab tagastama enumeratsiooni ühe võimaliku väärtuse, et põhimootor teaks, kas on näiteks vajalik stseenivahetus, programmitöö lõpetamine või heli mängimine. Kui põhimootor kindla sündmuse tulemusena ei pea midagi tegema, saab tagastada lihtsalt enumeratsiooni väärtuse „NOTHING“, millele ei reageerita.

Hiiresisendite puhul huvitavad meid sündmused, mille tüübiks on „SDL_MOUSEBUTTONUP“ ning nupuks on „SDL_BUTTON_LEFT“.

3.2.2 Stseeni kuvamine

Põhimootoril on alati viide mingisugusele käesolevale stseenile. Siinkohal kasutatakse ära polümorfismi ning kästakse stseenil läbi viia enda kuvamise meetod. Stseen omab enda graafilistele elementidele viiteid ning oskab seda käsku täita. Samuti, kuna sisendite töötlemine käib stseeni enda poolt, teab see, mis staadiumis ollakse, mis vajab kuvamist, mis mitte.

Ühel kaadril on alati üks stseen. Stseeni kuvamise meetod seega alustab puhastades SDL_Renderer objekti. Selleks pakub SDL meetodit SDL_RendererClear, mis soovib argumendiks SDL_Renderer objekti viidet, mille iga stseen saab põhimootori kaudu. [18] Seejärel tuleb lasta igal graafilisel elemendil ennast kuvada, mida tehakse SDL_RenderCopy meetodit kasutades (vt „Lihtne pildi ekraanil kuvamine“, lk 28). Viimasena annab stseen käsu koostatud kaader kuvada kasutades SDL_RenderPresent funktsiooni, mis sarnaselt SDL_RenderClear funktsioonile soovib argumendiks SDL_Renderer objekti. [19]

3.3 Programmi töö lõpp ning ressursside vabastamine

Programmi töö lõppemise puhul tuleks vabastada ressursid, mida töö käigus kasutati. Vabastamist peab tegema ka mingil määral programmi töötamise käigus, et pikaajalisel

jooksmisel ei tekiks probleeme mälu kasutusega. Vt „Tabel 3 Ressursside vabastamise funktsioonid“, lk 27.

Tabel 3 Ressursside vabastamise funktsioonid

Funktsioon	**Põhjus; mida vabastab**
SDL_DestroyTexture	Selle abil saab vabastada kindlaid SDL_Texture tüüpi mällu loetud objekte. Täpsemalt vt „Elemendi hävitamine“ lk 34.
SDL_DestroyRenderer	Funktsioon hävitab graafika kuvamiseks vajaliku konteksti ning vabastab ka sellega seotud SDL_Texture tüüpi objektid. [20]
SDL_DestroyWindow	Sellega hävitatakse SDL_CreateWindow funktsiooniga loodud aken. [21]
TTF_Quit	Suletakse ning koristatakse SDL_ttf-i poolt loodud liides ning selle võimalused. [22]
IMG_Quit	Likvideeritakse kõik SDL_image teegi poolt loodud liidesed ning vabastatakse võimalik mälu. [14]
Mix_CloseAudio	Suletakse ning vabastatakse võimalik mälu. [8] SDL_mixer API-t enam kasutada ei saa.
SDL_Quit	Suletakse kõik SDL poolt pakutavad võimalused, kaasa arvatud 2D graafikaks vajalikud komponendid. [23]

4. Graafilised elemendid

4.1 Lihtne pildi ekraanil kuvamine

Ükskõik millise kujutise näitamine ekraanil käib järgnevalt:

```
SDL_RenderCopy(MainEngine::renderer, this->texture, NULL, this->rect);
```

SDL_RenderCopy funktsioon viib kujutise SDL_Renderer objekti külge ehk kaadri peale. [24] Teine argument peab olema SDL_Texture tüüpi objekt, mis loetakse sisse kasutades SDL_image paketti PNG failist. Selleks on funktsioon IMG_LoadTexture, mis nõuab argumentideks SDL_Renderer objekti ning pildifaili nime. [25] Kolmas argument määrab koha, mida näidata SDL_Texture'i kujutisest. Väärtus „NULL" määrab, et kasutatakse kogu ala. Viimaseks saab määrata kujutise asukohta ning kõrgust-laiust utiliseerides SDL_Rect objekti. Sellel objektil on neli täisarvulist väärtust, mis kõik tähistavad otseselt piksleid ekraanil (vt „Tabel 4 SDL_Rect objekti väärtused" lk 28).

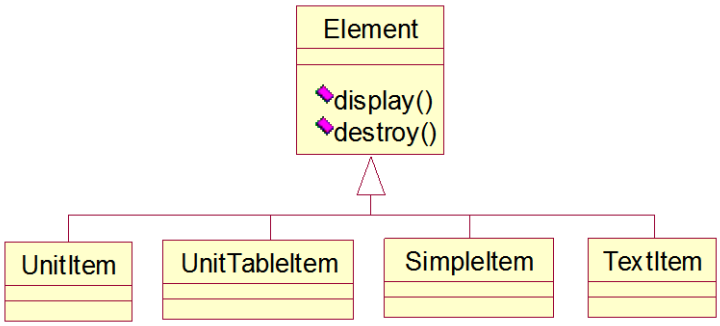
Tabel 4 SDL_Rect objekti väärtused

Väärtus	Sisu
x	Kaugus ülemisest vasakust nurgast horisontaalselt
y	Kaugus ülemisest vasakust nurgast vertikaalselt
w	Objekti laius
h	Objekti kõrgus

4.2 Element ehk teiste graafiliste elementide baasklass

Baasklass defineerib üldise liidese, mis kehtib kõigi teiste elementide kohta (vt „Joonis 8 Graafiliste elementide klassidiagramm", lk 29). Tema laiendused kasutavad vastavalt vajadusele defineeritud funktsioonide erinevaid implementatsioone, kuid saab ära kasutada polümorfismi, et kõiki elemente süsteemselt konteinerites hoida, kuvada või kustutada.

Kõigil elementidel saab määrata ära nähtavuse kahendmuutuja abil. Seega on mõistlik kõikvõimalikke elemente luua stseeni loomisel ning vajadusel nende kuvamine välja lülitada.

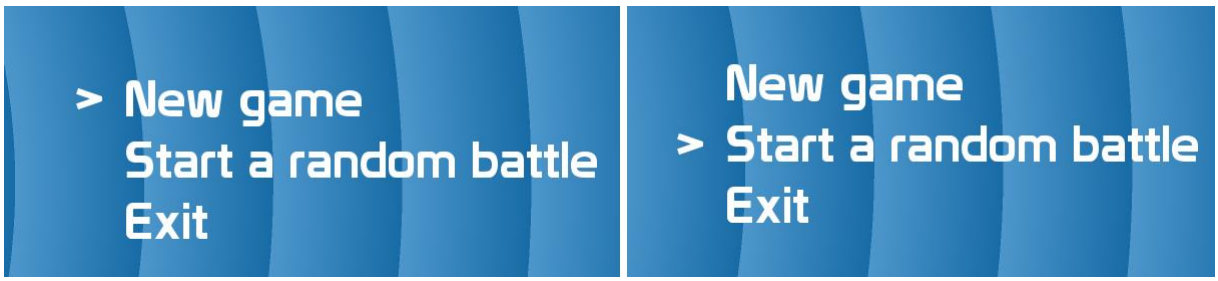


Joonis 8 Graafiliste elementide klassidiagramm

4.3 Elemendiga seotud elemendid

Igal elemendil on konteiner, mille sisse saab lisada teiste elementide viiteid. Sedasi saab realiseerida teistsugust kuvatud pilti vastavalt hiire asukohale. Element kontrollib, kas hiire koordinaadid x ja y, mis saadakse funktsiooniga SDL_GetMouseState [26], kattuvad selle elemendi pindalaga, mis tuletatakse SDL_Rect objektist. Kui saadud koordinaadid eksisteerivad pindala sees, kuvatakse seotud objektid.

Menüüesemete külge on seotud graafilised elemendid, mis kuvavad kasutajale, et kui sellistel koordinaatidel klikkida, valitakse just see menüü valik. Vt „Joonis 9 Seotud elementide kuvamine menüüstseenis“, lk 29.



Joonis 9 Seotud elementide kuvamine menüüstseenis

4.4 Lihtne element

Lihtne element (ingl. k „Simple Item“) tähistab tahet kuvada ekraanil otseselt kindlat SDL_Texture'i näol laetud kujutist. Lihtsa elemendi pindala on võimalik defineerida, kuid selle puudumisel kuvatakse antud SDL_Texture üle ekraani, näiteks taustapildi puhul.

Seda sorti elemendi kuvamiseks on vajalik vaid SDL_RenderCopy funktsiooni kasutamine (vt „Lihtne pildi ekraanil kuvamine“ lk 28).

4.5 Tekstist genereeritud element

Elemendi loomiseks on vaja järgnevaid osi:

1) Viide TTF_Font objektile. Tegemist on SDL_ttf paketi abil avatud kirjatüübiga, mida kasutatakse tekstist kujutise loomiseks. Selleks tuleb kasutada funktsiooni TTF_OpenFont, mis soovib kirjatüübifaili asukohta pluss nime ning kirjasuurust. [27] Samuti võib laetud kirjatüüpi muuta kasutades funktsiooni TTF_SetFontStyle [28], kui soovitakse näiteks paksu või kaldkirja. Viide TTF_Fontile jäetakse selle objekti külge, et teksti muutmisel ei peaks seda uuesti andma.

2) SDL_Color objektina värv, milles tekst peab ilmuma. Tegemist on lihtsa struktuuriga, mis hoiab endas RGB väärtusi ehk täisarvud 0 kuni 255-ni. [29] Näiteks „SDL_Color c = {255,255,255};“ tähendaks valget värvi.

3) Tekst, mida kuvada.

4) Koordinaadid x ning y, mida kasutatakse SDL_Rect objekti väärtuste täitmiseks. Need koordinaadid määravad kujutise pindala vasaku ülemise nurga asukoha aknal.

5) Soovi korral saab määrata, kas element peab algselt nähtav olema, kasutades kahendmuutujat. Vaikimisi on objektid nähtavad.

Neid sisendeid kasutatakse, et luua TTF_RenderText_Blended funktsiooni kaudu SDL_Surface objekt. [30] Funktsiooni sisenditeks on TTF_Font objekt, tekst ning soovitud tekstivärv. Kui soovitakse teksti muuta, tuleb lihtsalt vana tulemus hävitada ning uus luua. Õnneks antud programmis ei planeerita kuskil kiirelt muutuvaid tekste kuvada, seega kiirus pole väga prioriteetne eesmärk.

TTF_Fonti saab kasutada kolme erineva režiimiga, et saada kujutist. [31]

1) „Solid“ tüüpi funktsioonid loovad kiirelt, kuid tõenäoliselt kõige halvemini väljanägeva teksti. Kuna siin programmis tekstid väga kiiresti ei muutu, pole tarvis kasutada sellist lähenemist.

2) „Shaded“ puhul näeb pilt parem välja, kui „Solid“ tüüpi funktsioonidega. Siiski pole visuaalne tulemus kõige parem, kuna jääb nähtav teksti ümbritsev ruut.

3) Kõige aeglasemad, kuid visuaalselt paremad on „Blended“ tüüpi funktsioonid. Loomise aeg on väga sarnane „Shaded“ tüüpi funktsioonidele, seega samamoodi ei sobi kõige paremini kiiresti muutuvate tekstide jaoks, kuid antud programmi raames annab tulemuse, mida me otsime.

TTF_RenderText_Blended loob SDL_Surface objekti, millest tuleb luua kuvamiseks sobiv SDL_Texture, mida realiseeritakse funktsiooniga SDL_CreateTextureFromSurface. [32] Peale neid samme saab vabastada SDL_Surface'i kasutades SDL_FreeSurface funktsiooni. Seda kõike, kuna SDL_Texture on paremini optimiseeritud. [5]

Seejärel on vaja väärtustada veel elemendi SDL_Rect objekti väärtused, millest x ning y olid elemendi loomisel antud. Kõrgus ning laius tuleb võtta loodud SDL_Texture objektist kasutades funktsiooni SDL_QueryTexture, mis kasutab viiteid, et tagastada väärtused. [33]

Mõistlik on luua eraldiseisev meetod, mis loob tekstist SDL_Texture tüübi, et realiseerida teksti muutmist. Teksti muutmiseks tuleb esitada lihtsalt uus tekst. Kui see erineb eelnevalt mällu salvestatud tekstist, siis tuleb hävitada vana SDL_Texture objekt (vt „Elemendi hävitamine“ lk 34) ning luua uus. Kindlasti ka väärtustada elemendi SDL_Rect vastavad kõrguse ning laiuse väärtused, et ei tekiks pildi venimist.

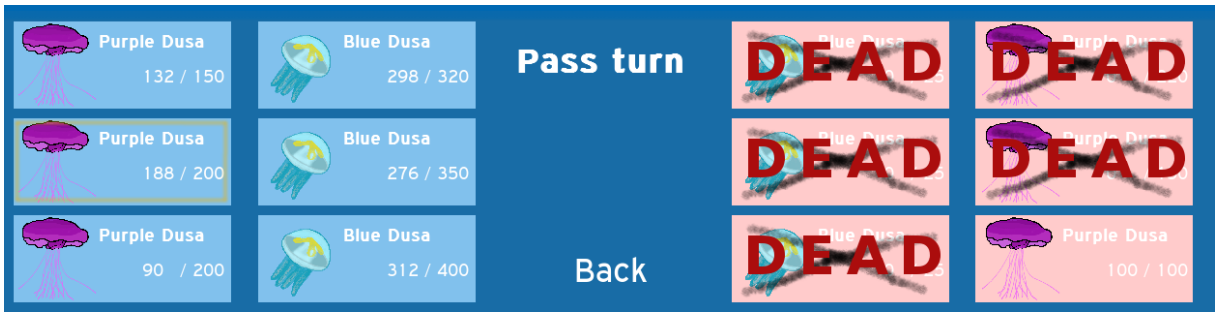
4.6 Elemendid, mis tähistavad üksust

Sai loodud kaks erineva otstarbega elementi, mis mõlemad tähistavad kasutajale mingisugust üksust. Need näevad visuaalselt erinevad välja ning sisaldavad enda pindalas mitut teist sorti visuaalset elementi. Kaks varianti on järgnevad:

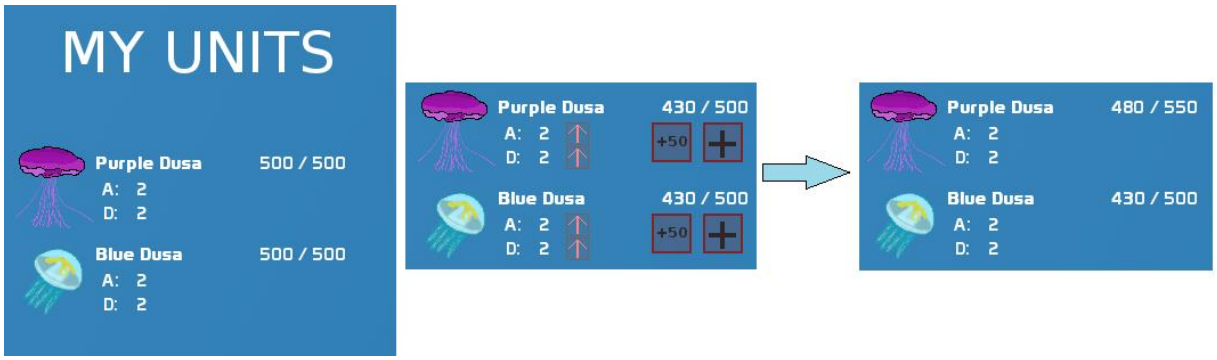
1) Lahingustseeni element (ingl. k. vaste „Unit Item“), mis peab lahingus vajalikku informatsiooni kuvama. Vt „Joonis 10 Lahingustseeni üksused“, lk 32.

2) Maailmakaardi kõrval olev element (ingl. k. vaste „Unit Table Item“), millel kuvatakse hetkeinformatsiooni ning, kui võimalik, ka nuppe, millele klikkides saab seda üksust tugevdada teatud viisil. Vt „Üksus“, lk 15 ja „Joonis 11 Maailmakaardi kõrval olevad üksused“, lk 32.

Üksused omavad näiteks oma pilti, oma nime teksti, hetketugevust, kogutugevust ning muid väärtusi, mida kuvatakse kasutajale. Neid väärusti üksusest kuvatakse kasutades teisi elemenditüüpe. Näiteks on üksuse nimi ning hetketugevus alati tekstist genereeritud elemendina.



Joonis 10 Lahingustseeni üksused



Joonis 11 Maailmakaardi kõrval olevad üksused

Lahingustseenil ning maailmakaardil kuvatavad üksuste elemendid erinevad üksteisest kuvatava info ning paigutuse poolest, kuigi mõlemad võivad eri aegadel samale üksusele viidata.

4.7 Lahingustseenis üksuse animatsioon

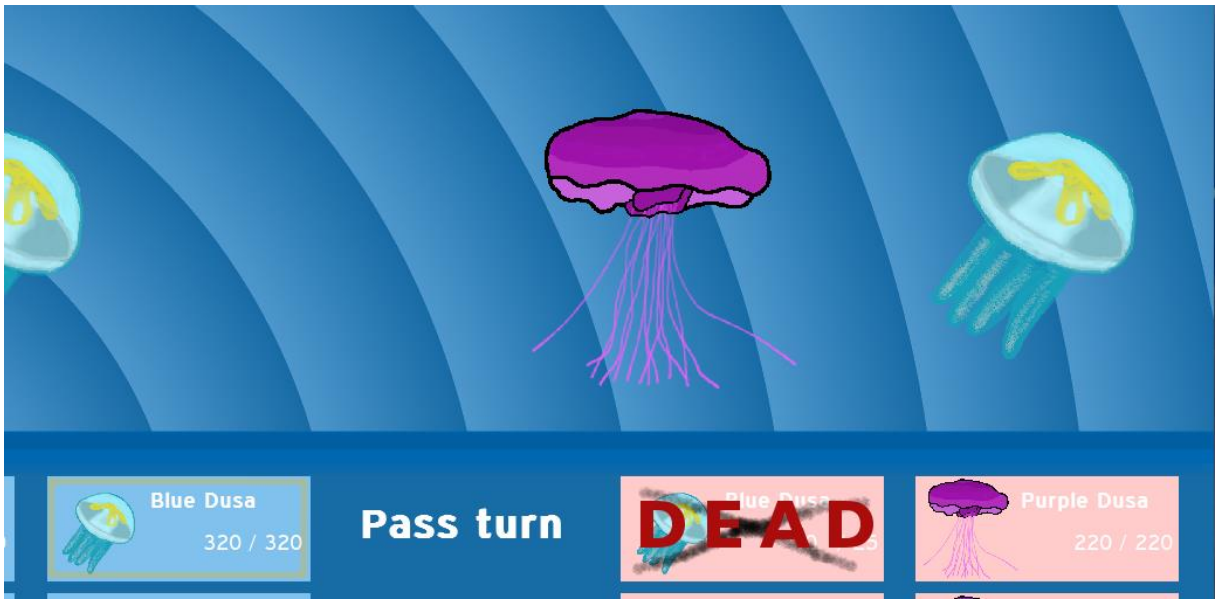
Lahingustseenis kuvatavad elemendid võivad läbida lihtsaid animatsioone ehk liikumisi, et illustreerida lahingus vastase ründamist. Selleks tuleb kuvada üksuse mudel ekraanil ning ründamise puhul läbida teatud liikumine.

Liikumise loomiseks tuleb ära kasutada kahte järgnevat kontseptsiooni:

1) Liikumist saab kirjeldada kasutades selleks kindlate hetkede tõmmiseid. See tähendab, et on teada kindel asukoht ning selles asukohas olemise ajahetk.

2) Kui liikumine on lineaarne, siis teatud ajahetkel on objekti asukohta võimalik arvutada teades, kuna ning kus alustati liikumist, kuna ning kus tuleb lõpetada. See tähendab, et kuvamise hetkel tuleb arvutada asukoht teekonnal.

Liikumise realiseerimiseks sai loodud konteiner, millesse saab lisada teatud punkti kirjeldavaid objekte. Selleks on andmestruktuur, mis teab ajahetke ning koordinaatide paari. Kui kasutada ära hetkeaega ning võrrelda seda liikumise algusaja ning oodatava lõppajaga, siis saab hinnata, kui kaugel ollakse koguteekonnast. Sedasi saab arvutada koordinaadid, mis jäävad alg- ning lõppasukoha vahele. Vt „Joonis 12 Element animatsiooni keskel“, lk 33.



Joonis 12 Element animatsiooni keskel

Kui jõutakse teekonna lõppu, kustutatakse eelmine punkt ning alustatakse järgmist teekonda kuni neid enam pole, mispuhul peidetakse liikuvad elemendid, kuna ründamine on lõppenud.

Animatsiooni alustamiseks antakse viide sihtmärgi mudelile, et animatsiooni käigus saaks seda kuvada. Liikumise lõppedes ei hävitata loodud lihtsat elementi, kuna kasutatud SDL_Texture objekti kasutatakse ka mujal.

4.8 Elemendi hävitamine

Stseenide hävitamisel kutsutakse esile ka iga sellega seotud elemendi hävitamine. Kuna kõiksugu graafika kasutab SDL_Texture objekte, tuleb lihtsalt kästa need mälust vabastada. Selleks on funktsioon SDL_DestroyTexture, mis soovib sisendiks viidet SDL_Texture'ile. [34] Sedasi saab tagada, et isegi programmi pikaajalisel kasutamisel pole vaja muretseda allesjäänud ressursside pärast. Kui elemendiga olid seotud ka teised elemendid, tuleb need samuti hävitada.

5. Audio mängimine

5.1 Audio mängimise klass

Klassi loomisel antakse teksti näol viide audiofailini, mis koheselt Mix_Chunk objekti loomiseks ära kasutatakse. Mix_Chunk tähendab audiojuppi, mida mõnel kanalil on võimalik mängima panna. Loomisel saab määrata ka helitugevuse väärtusest „0“ kuni „128“ („128“ on väärtus, mida märgib ka „MIX_MAX_VOLUME“). Kui Mix_Chunk objekti ei teki, on juhtunud mingisugune viga. [8]

Klass omab kahte meetodid, mis viitavad mõlemad kolmandale. Üks nendest käivitab audiojupi mängimise ühe korra, teine käivitab lõpmatu kordamise, mida kasutatakse taustamuusika puhul.

Mängitav Mix_Chunk luuakse kasutades funktsiooni Mix_LoadWAV. [8] Ainsa sisendina soovitakse sõne näol viidet kindla audiofailini, mis peab selle funktsiooni edukaks töötamiseks olema WAV formaadis. Seetõttu sai otsustatud, et kõik audiofailid luuakse ühtselt WAV formaadis.

5.2 Helijupi mängimine

Klassi loomisel väärtustatakse mängitavaks kanaliks „-1“, et ära kasutada Mix_PlayChannel funktsiooni eripära. [8] Järgnev rida mängib helijupi:

```
this->channel = Mix_PlayChannel(this->channel, this->chunk, loops);
```

Mix_PlayChannel tahab esimeseks argumendiks kanali numbrit, kus jupp mängima panna. Väärtusel „-1“ valitakse esimene vaba. Kuna see meetod tagastab kanali numbri, kus ta mängima pandi, siis peale esimest mängimist jääb heli alati enda kanalile. Teine argument on Mix_Chunk tüüpi objekti viide ning kolmandaks tahetakse täisarvuna väärtust, mis tähistaks, mitu korda audiojuppi mängitakse. Väärtus „-1“ tähendab, et mängitakse lõpmatult, väärtus „0“ puhul üks kord, „1“ puhul kaks korda jne. Kasutusel on vaid väärtused „-1“ (taustamuusika puhul) ning „0“ (kõiksugu muude helide puhul).

Taustamuusika mängimise puhul määratakse helitugevus funktsiooniga Mix_Volume. [8] Sisenditeks soovib see meetod kanali numbrit ning helitugevust, mis on „0“ ning „128“ vahel.

5.3 Objekti sulgemine ning hävitamine

Audiojupi objekti hävitamine toimub antud programmi vältel vaid sulgemise puhul. See tähendab, et võib eeldada, et kõiksugune audio lõpetab töö korraga. See tähendab, et ükskõik missuguse audio lõpetamisel võib kogu selle kanali peatada funktsiooniga Mix_HaltChannel [8]. Seejärel võib olla kindel, et audiojupp on vaba hävitamiseks funktsiooniga Mix_FreeChunk.

6. Tulemuse analüüs

6.1 Hinnang visuaalsele tulemusele

Valmistatud programm näeb visuaalselt ootustepärane välja. Isegi suure kirjasuurusega teiste graafiliste elementide peale loodud ning asetatud kirjutised olid siledate joontega. Programmis kuvatud tagataust nägi õigel resolutsiooni välja korrektne ning palju parem võrreldes venitatud kujutisega, isegi kui venitus toimus väiksemaks. Seetõttu ongi vajalik erinevate resolutsioonide jaoks luua erinev graafiline vara või integreerida vektorjoonistuste kasutamine.

6.2 Kaadrit sekundis

Programmi jõudlust saab paljugi määrata selle kaudu, kas tegemist on paljunõudva või kerge programmiga. Antud juhul saab määrata optimiseeritust kasutades mõõtu „kaadrit sekundis“ ehk inglise keeles „Frames per second“.

Koos lipuga *SDL_RENDERER_PRESENTVSYNC* saab ligikaudu 60 kaadrit sekundis, mis on õigesti töötava programmi puhul oodatav testimiseks kasutata masina peal, kuna kasutatakse 60 Hz monitori. See tähendab, et tüüpilise tsükli läbimise käigus ei leidu kalleid meetodeid (näiteks stseeni laadimine, elementide loomine), mis võiks tüüpilises programmi töös tekitada ebavajalikke jõudlusprobleeme.

Lukustamata kaadrisagedusega saab eri stseenidel tulemused, mis ei lange kunagi alla 1100. See on täiesti oodatav tulemus arvestades, et programmi ei saa lugeda nõudlikuks, kuna kuvamise loogika pole suure keerukusega ning kuvatavate elementide arv pole väga suur.

6.3 Heli kvaliteet

Hinnanguliselt ei suuda vahet teha läbi loodud programmi mängitud helidel ning mõne audiotöötlus programmiga mängitul. Helis pole kordagi kuulda vigu näiteks ragisemist või sahinat. Helid ei jookse programmi töö käigus kokku vaatamata sellele, kui palju samasuguseid sisendeid programmile anda.

7. Kokkuvõte

Töö eesmärgiks oli kasutajaliides kasutades selleks SDL2 API poolt pakutavaid võimalusi. Graafilised elemendid pidid omama kasutajaliideses mingit sorti rolli ehk programmi tööd pidi nende kaudu saama juhtida. See tähendab, et mõned elemendid funktsioneerivad kui klikitavad nupud. Teatud kasutaja sisendid ning stseenide vahetused pidid olema seotud mingisuguse heliga.

Tulemusena loodi hulk klasse, mis aitavad süsteemselt luua ning käsitleda kuvatavaid graafilisi varasid. Nendeks on graafiliste elementide klassid ning ka stseenid, mis käituvad kui graafiliste elementide hulgad. Loodi helide mängimiseks klass, mida saab kasutada programmis sündmuste tähistamiseks.

SDL2 pole loodud kasutajaliideste loomiseks, kuid oma palju häid võimalusi ning tuge selleks. Kui luua hulk klasse, mis abistavad läbida kasutajaliidese koostamiseks vajalikke protseduure, on siiski võimalik luua funktsioneeriv kasutajaliides.

Edasiarendusena saaks lisada puldiga programmi navigeerimise tugi, mida saab realiseerida SDL2-e võimaluste kaudu. Graafiliste varade hulga laiendamisel saaks lubada ka teistsuguseid resolutsioone. Animatsioonide tuge saaks laiendada kõigi graafiliste elementide peale

Võib öelda, et eesmärk saavutati, kuna tulemusena loodi programm, mis kasutab SDL2-e võimalusi kasutajaliidese realisatsiooniks. Kuna kasutajaliidese tegemise maht oli väga suur, jäi strateegiamängu disanimisele vähem rõhku, kui algselt oli planeeritud. Strateegiamängu sisu lisamine, st üksuste ja gruppide lisamine maailmakaardile, on olemasoleva süsteemi kontekstis ülimalt lihtne ning otsekohene.

Põhitulemusteks võib lugeda järgmisi objekte: graafiliste elementide klassid, stseenide klassid, põhimootor, mis koordineerib stseenide vahetust ning helide mängimist, helide mängimise realiseerimise klass, mängitav strateegiamäng.

Eesmärgid saavutati ootustepäraselt. Parandada saaks vaid asjaolu, et audio realiseeritakse ainult põhimootori poolt. Osa sellest funktsionaalsusest peabki sinna jääma, kuid mõistlik oleks audiojupid seostada stseenidega. Programmi laienemisel ning audiojuppide hulga

suurenemisel võib probleemiks osutuda kõigi helide korraga mälus hoidmine, kui selleks pole vajadust, st kõiki helisid kõik stseenid ei kasuta.

Strateegiamängu realiseerimiseks loodi mõned klassid, mida kasutades ning manipuleerides said graafilised elemendid kuvada sisukat informatsiooni kasutajale.

Summary

The aim of this thesis was to use the possibilities of the SDL2 API. Some graphical elements had to have a functional role in the user interface as in they had to have the possibility of affecting the flow of the program in some way. As in functioning as a clickable button. User inut and changes in state had to be signified by sounds.

As a result, several classes were designed. Those would help systematically create and handle displayable graphical element. Those classes were graphical elements' classes and stages, which act as handlers to an array of elements. A single class was created to play sound cues. Those cues were used to signify change in stage or significant user input.

It was time consuming to create a user interface using SDL2, because it is not primarily designed to be used as a tool for creating them, yet it offers a huge variaty of possibilities for the programmer through the API. If one were to create classes, which would go through typical processes of creating and using graphical elements, they would be able to create a functioning user interface.

Overall, the goals were met and the result is a functioning program, that utilizes the SDL2 API to create a user interface. In order to reach that goal, several classes were created. Those classes were graphical elements, stages, a main engine and a class for playing audio chunks.

Due to the amount of work involved in creating a user interface through these methods, not as much time was devoted to the development of the strategy game as was originally planned, yet a playable game was created.

Kasutatud kirjandus

[1] „Simple DirectMedia Layer,“ [Võrgumaterjal]. Available: http://www.libsdl.org/index.php.

[2] M. Adler ja J.-l. Gailly, „Zlib litsents,“ [Võrgumaterjal]. Available: http://www.gzip.org/zlib/zlib_license.html.

[3] J. Barczak, „OpenGL Is Broken,“ 30 mai 2014. [Võrgumaterjal]. Available: http://www.joshbarczak.com/blog/?p=154.

[4] R. Gordon, „Getting Started with Linux Game Development,“ Steamworks Development, 11 veebruar 2014. [Võrgumaterjal]. Available: http://youtu.be/Sd8ie5R4CVE.

[5] „Migration Guide - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/MigrationGuide.

[6] G. Roelofs, „A Basic Introduction to PNG Features,“ [Võrgumaterjal]. Available: http://www.libpng.org/pub/png/pngintro.html.

[7] G. Roelofs, „Current Status of PNG,“ [Võrgumaterjal]. Available: http://www.libpng.org/pub/png/pngstatus.html.

[8] „SDL_mixer 1.2.10,“ [Võrgumaterjal]. Available: http://www.libsdl.org/projects/SDL_mixer/docs/SDL_mixer.html.

[9] Valve Corporation, „Steam Hardware & Software Survey: April 2014,“ [Võrgumaterjal]. Available: http://store.steampowered.com/hwsurvey/.

[10] „SDL_Keycode - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_Keycode.

[11] „SDL_Init - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_Init.

[12] „SDL_CreateWindow - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_CreateWindow.

[13] „SDL_CreateRenderer,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_CreateRenderer.

[14] „SDL_image,“ 3 november 2009. [Võrgumaterjal]. Available: https://www.libsdl.org/projects/SDL_image/docs/SDL_image_frame.html.

[15] „SDL_ttf 2.0.10: TTF_Init,“ 13 november 2009. [Võrgumaterjal]. Available: https://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf_8.html.

[16] „SDL_PollEvent SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_PollEvent.

[17] Lazy Foo' Productions, „SDL Tutorial Mouse Events,“ [Võrgumaterjal]. Available: http://lazyfoo.net/SDL_tutorials/lesson09/index.php.

[18] „SDL_RenderClear - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_RenderClear.

[19] „SDL_RenderPresent - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_RenderPresent.

[20] „SDL_DestroyRenderer - SDL Wiki,“ [Võrgumaterjal]. Available:

https://wiki.libsdl.org/SDL_DestroyRenderer.

[21] „SDL_DestroyWindow - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_DestroyWindow.

[22] „SDL_ttf 2.0.10: TTF_Quit,“ [Võrgumaterjal]. Available: http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf_10.html.

[23] „SDL_Quit - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_Quit.

[24] „SDL_RenderCopy - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_RenderCopy.

[25] TwinklebearDev, „SDL Extension Libraries,“ [Võrgumaterjal]. Available: http://twinklebeardev.blogspot.com/2012/07/lesson-3-sdl-extension-libraries.html.

[26] „SDL_GetMouseState - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_GetMouseState.

[27] „SDL_ttf 2.0.10: TTF_OpenFont,“ 13 november 2009. [Võrgumaterjal]. Available: http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf_14.html.

[28] „SDL_ttf 2.0.10: TTF_SetFontStyle,“ 13 november 2009. [Võrgumaterjal]. Available: http://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf_22.html.

[29] „SDL_Color - SDL_Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_Color.

[30] „SDL_ttf 2.0.7: TTF_RenderText_Blended,“ [Võrgumaterjal]. Available: http://www.libsdl.org/projects/docs/SDL_ttf/SDL_ttf_44.html.

[31] „SDL_ttf 2.0.7: Render,“ [Võrgumaterjal]. Available: http://www.libsdl.org/projects/docs/SDL_ttf/SDL_ttf_35.html.

[32] „SDL_CreateTextureFromSurface - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_CreateTextureFromSurface.

[33] „SDL_QueryTexture - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_QueryTexture.

[34] „SDL_DestroyTexture - SDL Wiki,“ [Võrgumaterjal]. Available: https://wiki.libsdl.org/SDL_DestroyTexture.