Tallinna Ülikool

Informaatika Instituut

SFML ÕPPEMATERJAL

Seminaritöö

Autor: Lauri Roomere

Juhendaja: Jaagup Kippar

Tallinn 2013

Sisukord

Sissejuhatus

Siinse seminaritöö eesmärgiks on luua eestikeelne SFML (Simple and Fast Multimedia Library) õppematerjal, mis tutvustaks lühidalt SFML'i ja tema kasutusvõimalusi. Õppematerjali käigus kasutan SFML 2.0 versiooni ja MS Visual C++ Express'i. SFML 2.0'i valisin, kuna tegemist on kõige uuema versiooniga, mis hetkel ei ole stabiilseks versiooniks kuulutatud sellepärast, et SFML'i autor ei soovi seda teha ennem, kui ta on veebilehte uuendanud ja kõik juhendid valmis teinud.

Õppematerjal on peamiselt suunatud mänguloojatele, kes soovivad 2D mänge luua SFML 2.0 versiooni kasutades. Soovitatav on kokkupuude C++'i ja programmeerimisega. Kasuks tuleb kindlasti varajasem kogemus mängude loomisega.

Õppematerjali käigus antakse õppurile vajalikud teadmised, mille abil oleks võimalik koostada mõni lihtsam mäng, näiteks Ping-Pong. Õppematerjalis käsitletavad teemad olen valinud oma kogemuste põhjal, mis tekkisid mängude loomisel kõige enam käsitlevatele probleemidele lahenduste otsimisega.

Õppematerjalis käsitletakse lühidalt objektide liigutamise, põrke tuvastuse, kujundite joonistamise, tekstuurimise, objektide värvimise, sprite'de, heli ja muusika kasutamist. Samuti tutvustatakse lühidalt SFML'i ja selgitatakse mis on SFML.

Õppematerjalist jätsin välja mõningad teemad, mida ei käsitlenud puuduva kogemuse, tehniliste probleemide või teadmiste tõttu. Näiteks võrke, juhtkangi (ingl.k joystick) ja 2D kaamera kasutamist käsitlevad teemad, sest juhtkang mul endal puudub ning inimene, kes esmakordselt tutvub SFML'ga peaks esmalt suutma luua paar lihtsat mängu ennem, kui ta hakkab looma kliendi ja serveri suhtlusel baseeruvaid rakendusi. Selle jaoks tuleks lisaks seletada lahti erinevate interneti protokollide tähendused ja kasutamisvõimalused. Õppematerjal ei käsitle kõiki SFML'i komponente.

Teema valiku põhjuseks oli autori isiklik huvi SFML'i ning 2D mängude loomise vastu ja eestikeelse informatsiooni puudumine SFML'st. SFML'i valisin mitmete mängude loomisel abiks olevate teekide seast hea dokumentatsiooni, vajalike komponentide sisaldamise ja lihtsuse pärast. Suurimaks puuduseks pean hetkel SFML'i teekidel mobiiliplatvormide toetuse puudumist. See ei tähenda, et mobiilidele ei ole üldse võimalik SFML'i kasutades rakendusi luua. See on võimalik ja juba ka tehtud, kuid nõuab palju tööd ning loodetavasti on järgmisesse SFML'i versiooni juba mõne mobiiliplatvormi toetus lisatud.

Mõisted

Multimeedium - „Teabe mitme esitusvormi üheaegne kasutamine (tekst, graafika, heli, video jm)“ (Vallaste, 2013).

Teek (ingl.k library) - „Programmeerimises nimetatakse teegiks valmiskompileeritud alamprogramme, mida programm saab kasutada“ (Vallaste, 2013).

Kompileerima (ingl.k compile) - „Kõrgkeeles kirjutatud programmi masinakeelde tõlkima“ (Vallaste, 2013).

Madalatasemepöördus (ingl.k high-level access) - „Tarkvara , mis juhib riistvara otse, ilma et oleks vaja läbida programmi transleerimise kihti“ (Vallaste, 2013).

Kõrgetasemepöördus (ingl.k low-level access) - Tarkvara, mis kasutab madalatasemepöördusega tarkvara.

Tsükkel (ingl.k loop) - Tähendab programmeerimises mingi koodiosa kordamist programmis.

Puhver (ingl.k buffer) - „Mäluosa ajutiseks andmesalvestuseks. Enamiku puhvrite ülesanne on säilitada andmeid selleks, et protsessor saaks neid töödelda enne mingile seadmele (näit. printerile või kõvakettale) saatmist“ (Vallaste, 2013).

Sõne (ingl.k string) - „Arvutiterminoloogias tervikuna käsitletav elemendi- või märgijada. Erinevalt sõnast või arvust ei pruugi stringil olla mingit konkreetset tähendust – string võib näiteks olla ka suvaline sõnaosa“ (Vallaste, 2013).

Liides (ingl.k interface) – „Sõltumatute funktsionaalüksuste vaheline ühispiir, kus kehtivad koostöötingimused, mis puudutavad funktsioone, füüsilisi ühendusi, signaalivahetust jms“ (Vallaste, 2013).

Kasutajaliides (ingl.k user interface) – „Klaviatuur, hiir, arvuti menüüd jne. Kasutajaliides võimaldab kasutajal suhelda arvuti opsüsteemiga“ (Vallaste, 2013).

Tarkvaraliides (ingl.k software interface) – „Keeled ja programmid, mida rakendusprogrammid kasutavad suhtlemiseks omavahel ja arvuti riistvaraga“ (Vallaste, 2013).

Riistvaraliides (ingl.k hardware interface) –„ Juhtmed, pistikud ja pesad, mis võimaldavad riistvarakomponentide omavahelist suhtlemist“ (Vallaste, 2013).

Konverter (ingl.k converter) – „Seade, mis muundab ühe koodide, režiimide, jadade või sageduste komplekti teiseks komplektiks“ (Vallaste, 2013).

Siluma (ingl.k debug) – „ Programmides vigu avastama, lokaliseerima ja kõrvaldama“ (Vallaste, 2013).

1 Mis on SFML?

SFML (*Simple and Fast Multimedia Library*) on porditav multimeediumi teek, mis tagab madala- ja kõrgetasemepöörduse graafikale, helile, sisendile, aknale ja süsteemile (Gomila, 2011). SFML on kirjutatud C++ programmeerimiskeelt ja OpenGL'i (Open Graphics Library) kasutades.

SFML'i saab kasutada mängude ja interaktiivsete rakenduste loomiseks. Esimene SFML'i versioon SFML 1.0 väljastati 2007. aastal. Hetkel on kõige uuemaks stabiilseks versiooniks SFML 1.6, kuid selle õpematerjali käigus kasutame me SFML 2.0 versiooni mis on hetkel RC (väljastuskandidaat) staatuses. Hetkel on SFML 2.0 veel väljastuskandidaadi staatuses seetõttu, et SFML'i autor ei soovi teda ametlikult enne stabiilseks versiooniks kuulutada, kui SFML'i koduleht on uuendatud ja kõik juhendid valmis tehtud.

SFML võimaldab luua rakendusi, mis töötavad Windows'i(98,2000,XP,Vista,7,8), Mac OS X'i ja Linux'i platvormil. Multimeediumi teegi suurenedes lisatakse juurde rohkem operatsiooni-süsteeme, mida toetatakse. SFML'i saab kasutada järgnevate programmeerimiskeeltega:C++, C, Net, Python, D, Ruby.

SFML koosneb 5 moodulist (süsteem, aken, graafika, heli ja võrk), mis koosnevad mitmetest klassidest (vt joonis 1).

Süsteemi moodul – Süsteemi moodulit kasutades saab luua lihtsaid või objekt-orienteeritud programmi lõimesid ja vastastikust välistamist. Porditavat ja täpset aja mõõtmist. Unicode moodulit universaalse teisendusvormingu ja kohalike sõltuvate kodeeringute teisenduseks (Gomila, 2011).

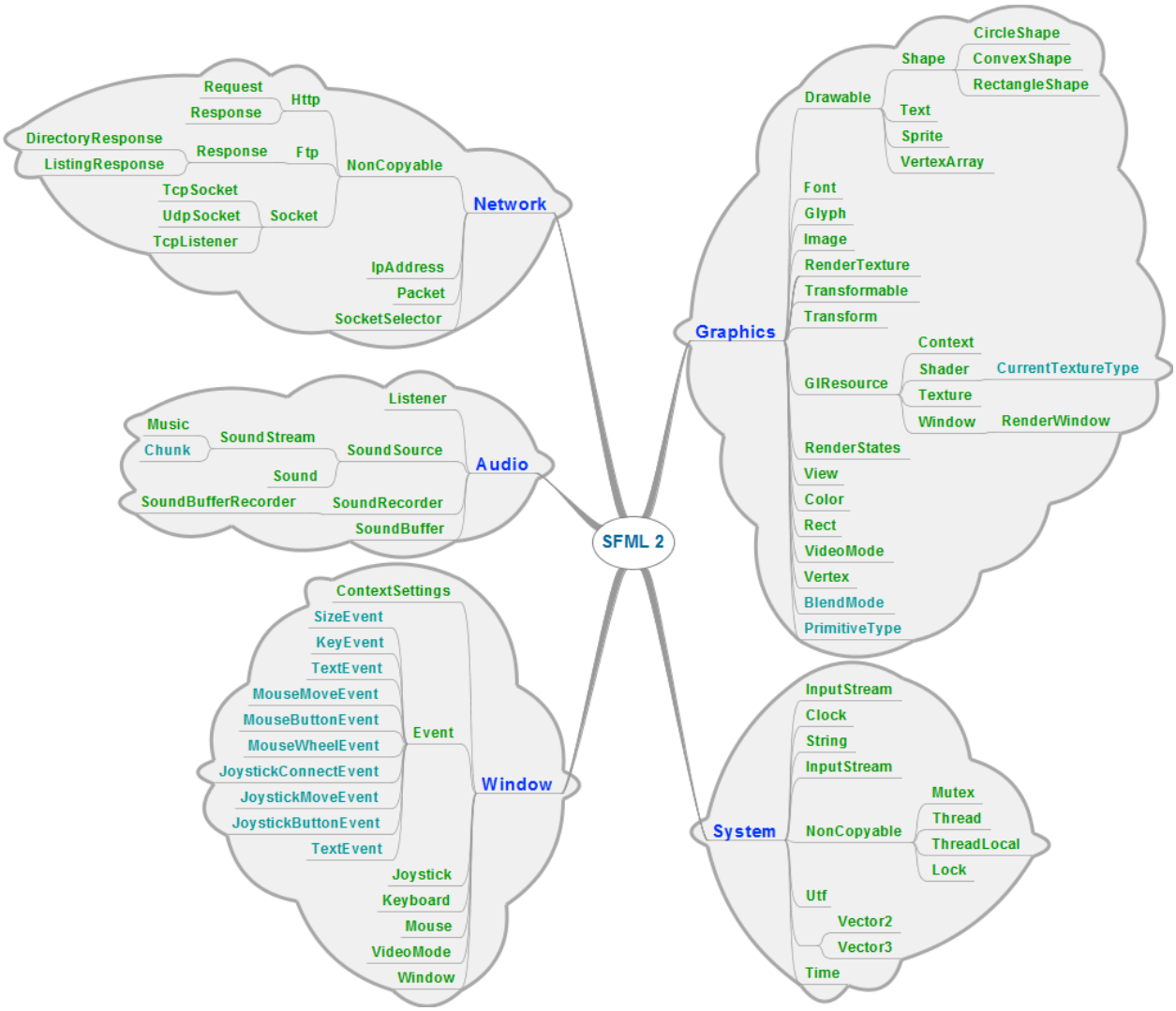
Akna moodul – Saab kasutada porditava paketina, et asendada akna loomist ja sisendit GLUT'i või SDL'i asemel OpenGL'ga. Saab luua mitu visualiseerivat akent. On võimalik integreerida Qt, MFC, wxWidgets, Win32, x11 ja veel mitmete liidestega. Tagab nii sõnumipõhise ja reaalajas toimuva liidese sisendi haldamiseks. Suudab hallata hiirt, millel on 5 hiire nuppu ja nelja juhtkangi 7 telje ning 32 nupuga (Gomila, 2011).

Graafika moodul – Lihtne liidestada OpenGL'ga. Võimaldab kasutada kaasaegseid effekte ja riistvara kiirendust pööramiseks, varjutamiseks, alfasujutamiseks jne. Haldab mälu tõhusalt nii, et kasutajal ei ole vaja muretseda ressursi eluaja või säilitamise pärast. Võimaldab laadida graafika ressursse enne akna loomist. Suudab laadida ja salvestada järgmisi standardseid pildi formaate: bmp, dds, jpg, png, tga, psd. Suudab laadida graafika ressursse failidest mis on mälus. Võimaldab kasutada 3D stseenide vaateid suurendamiseks, transleerimiseks ja kogu maailma pööramiseks. Kasutab lihtsustatud varjutamiskeelt reaalajas toimuvatele efektidele tagajärje

lisamiseks. Kerge manipuleerida graafilise tekstiga rasterfontide abil. Toetab unicode tähemärke. Suudab laadida järgnevaid kirjatüübi faili formaate: ttf, cff, pcf, fnt, bdf, pfr, sfnt, type 1, type 42 (Gomila, 2011).

Heli moodul– Kasutab igal võimalikul juhul riistavara kiirendust. Suudab laadida ja salvestada järgnevaid standardseid heli formaate: ogg, wav, flac, aiff, au, raw, paf, svx, nist, voc, ircam, w64, mat4, mat5 pvf, htk, sds, avr, sd2, caf, wve, mpc2k, rf64. Suudab laadida kõiki heli ressursse otse failidest, mis on mälus. 3D heli võimalused. Haldab mälu tõhusalt nii, et kasutajal ei ole vaja muretseda ressursi eluaja või säilitamise pärast. Toetab mahukate failide voogedastamist. Lisaks on võimalik luua enda jaoks voogedastus klass ükskõik millisele allikale (nt. võrgule). Toetab mitme kanali formaati (mono, stereo, 4.0, 5.1, 6.1, 7.1) (Gomila, 2011).

Võrgu moodul – Teostab porditava kihi üle TCP ja UDP sokli. Lihtne andmeside voogedastusel põhinevatel laiendatavatel pakettidel. Klassid HTTP ja FTP interneti protokollide kasutamiseks (Gomila, 2011).



Joonis 1. SFML 2.0 klasside hierarhia (Baudry, 2012).

SFML’i autorid on Laurent Gomila (peamine arendaja) ja Marco Antognini (OS X arendaja). SFML’il on aktiivne foorum, millest vajaduse korral saab abi leida või teistele nõu anda. Foorum asub aadressil http://en.sfml-dev.org/forums/ .

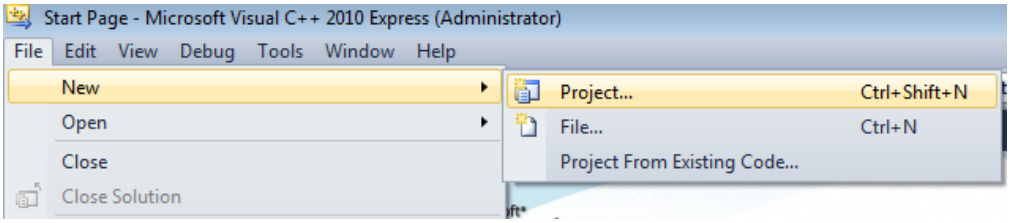
Järgnevalt aadressilt leiab YouTube’i kasutajakontolt CodingMadeEasy inglisekeelsed video juhendid SFML 2.0’i kasutamisest, mida soovitan algajatele ja juba kogenumatele mängu-loojatele: https://www.youtube.com/playlist?list=PLHJE4y54mpC5j_x90UkuoMZOdmmL9-_rg

2 SFML'i ja Visual C++ 2010 Express'i

SFML arendustarkvara on saadaval aadressilt: http://www.sfml-dev.org/download.php. Kui olete SFML'i arendustarkvara allalaadinud, siis paigutage SFML'i kaust oma arvutis sellisesse kohta, kus on võimalik seda kergesti üles leida.

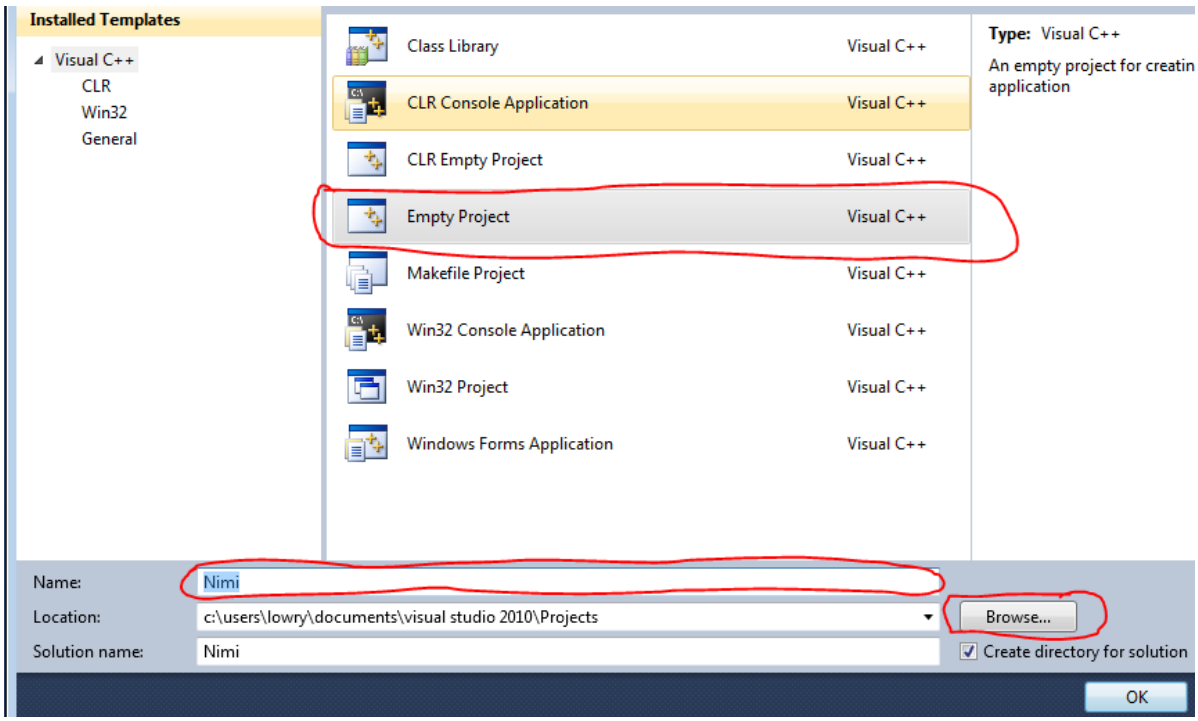
Selle õppematerjali käigus kasutame Microsoft Visual C++ 2010 Expressi, seega oleks vaja alla laadida Visual C++ 2010 versioon. Microsoft Visual studio C++ 2010 Express on tasuta saadaval aadressilt: http://www.microsoft.com/visualstudio/eng/downloads#d-2010-express

Kui vajalik tarkvara on alla laaditud, võib alustada SFML 2.0 ja Microsoft Visual Studio C++ 2010 Express'i linkimisega. Esimeseks sammuks oleks MS Visual C++ 2010 Express avada ja uus projekt luua.Valige *File->New->Project* või vajutage klahvikombinatsiooni *Ctrl+Shift+N* (vt joonis 2).



Joonis 2. Uue projekti loomine

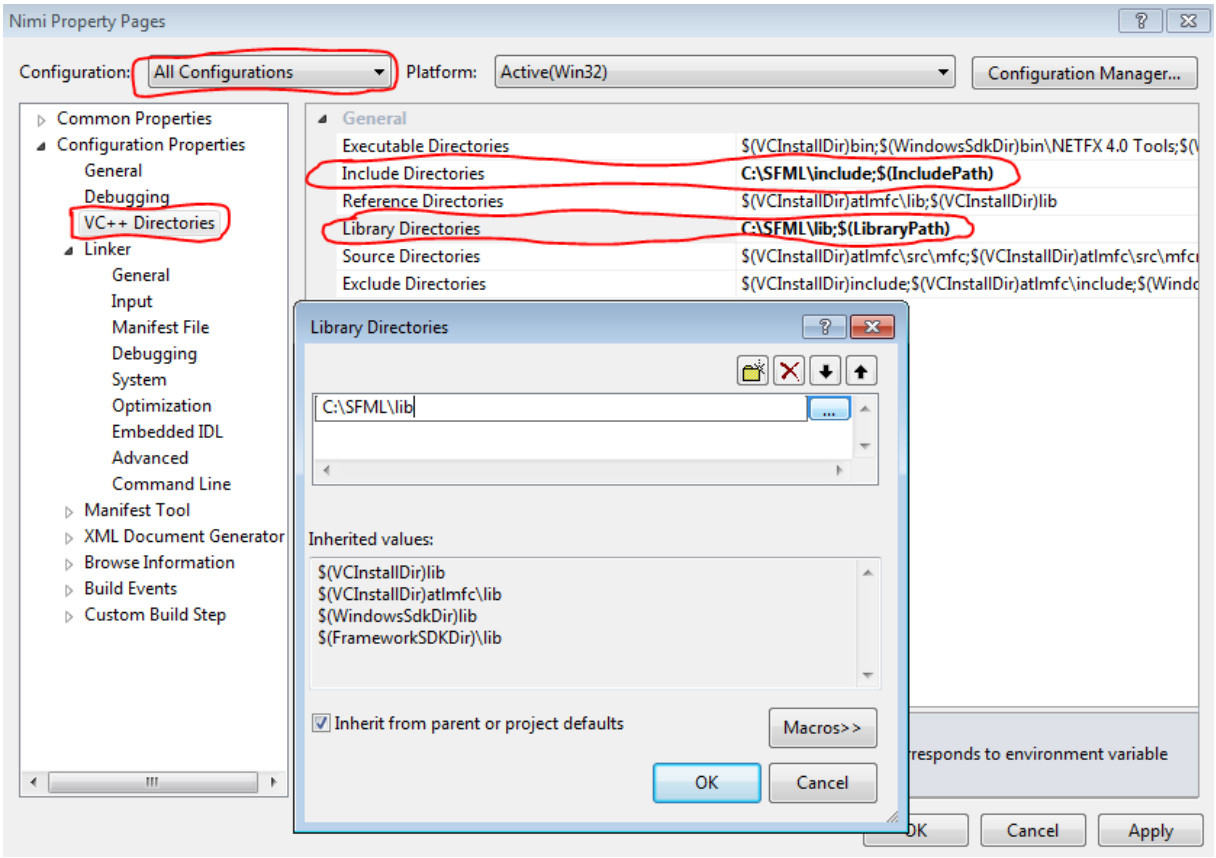
Järgmisena valige *Empty Project* ja lisage oma uuele projektile nimi. Vajaduse korral võite projekti asukoha ära muuta vajutades *Browse* nupule (vt joonis 3).



Joonis 3. Projekti tüübi valimine

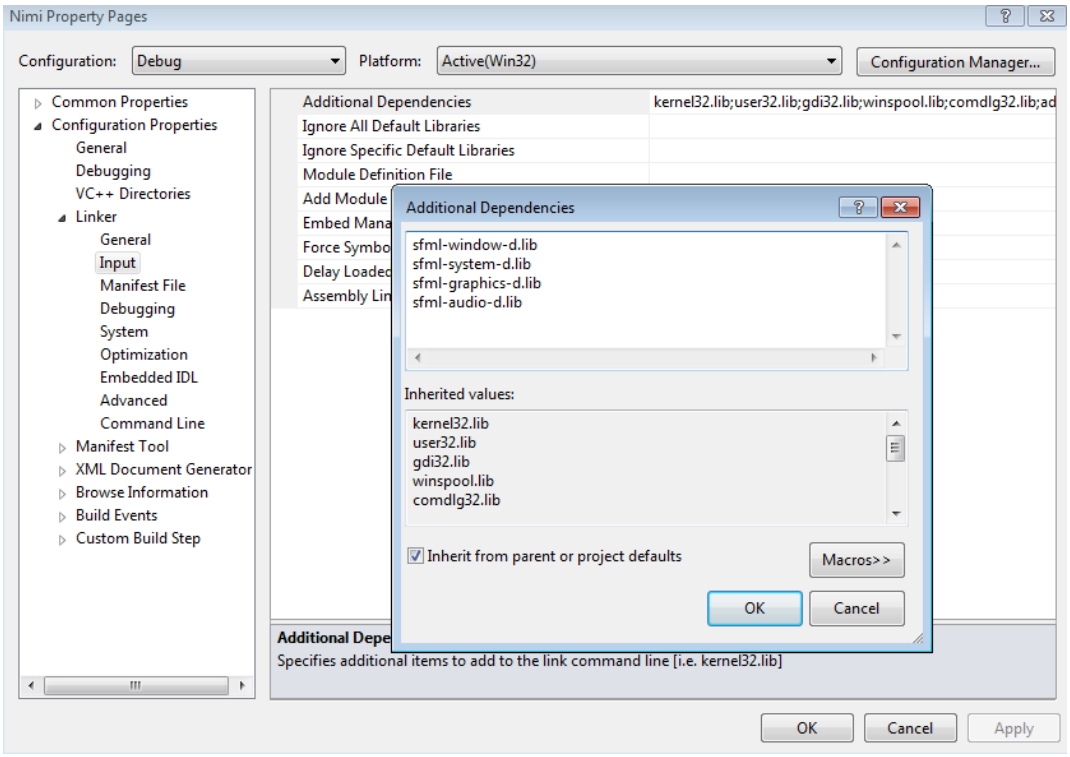
Nüüd tuleb kompilaatorile öelda, kus asuvad SFML päised (.hpp failid) ja linkur (ingl.k linker) mille abil Visual Studio leiab SFML teegid (.lib failid) (vt joonis 4).

Valige ülevalt menüü ribalt *Project->Properties*. Seejärel valige *Configuration* rippmenüüst *All Configurations*, sest määratav tee on silumisel (*debug*) ja avalikustamisel (*release*) sama. Vajutage *Include Directories* peale ja avanevast rippmenüüst valige *edit* ning määrake tee SFML kaustas olevasse *include* kausta. Sama samm tuleks teostada *Library Directories'ga* ja määrata tee lib kausta. Mõlemad kaustad leiate oma SFML kaustast. Minu lib kaust asub järgneval aadressil C:\SFML\lib ja include C:\SFML\include.

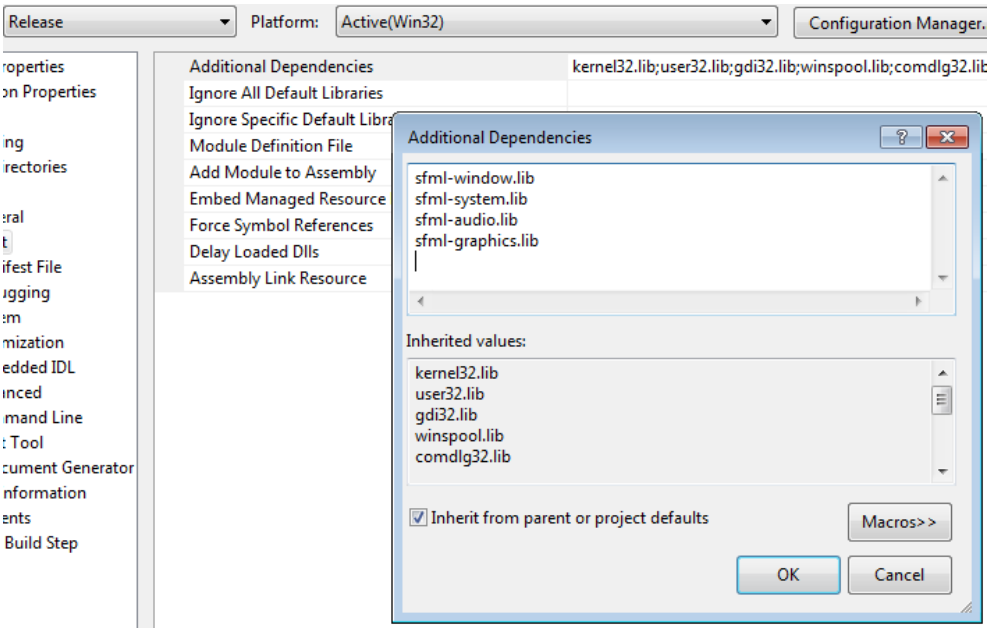


Joonis 4. Include ja Library kaustade lisamine

Eelviimaseks sammuks on rakenduse linkimine SFML teekidega (.lib failid), mida teie koodis vaja läheb. Teegid tuleb lisada *Project Properties Linker->Input->Additional Dependencies* alla. Lisada tuleb kõik vajalikud teegid. Käesoleva õppematerjali käigus läheb vaja järgmisi teeke: sfml-system.lib, sfml-window.lib, sfml-graphics.lib, sfml-audio.lib. Alati on võimalik *Project Properties* teeke vajaduse korral juurde lisada. Silumisele teeke lisades tuleb kirjutada *Input->Additional Dependencies* alla sfml-xxx-d.lib (vt joonis 5) ja väljastamisel sfml-xxx.lib (vt joonis 6).



Joonis 5. Silumisele teekide lisamine.



Joonis 6. Väljastamisele teekide lisamine

Viimaseks sammuks oleks SFML\bin kaustast kopeerida oma projekti juurde SFML DLL failid. Kopeerimata võib jätta sfml-network-2.dll ja sfml-network-d-2.dll'i. Näide sellest, kuhu kausta mina oma DLL failid kopeerisin: C:\Users\Lowry\Documents\VisualStudio 2010\Projects\Nimi\Nimi.

3 Akna loomine

Selles peatükis loome uue akna sf::Window klassi abil. Peale sf::Window klassi on võimalik kasutada sf::RenderWindow klassi, mis on graafika mooduli peaklass ning sisaldab rohkem funktsioone akna haldamiseks, kui sf::Window klass. Selles õppematerjalis kasutame tihti *clear()* ja *draw()* funktsioone, mis kuuluvad sf::RenderWindow klassi. Seetõttu pean vajalikuks lahti seletada, mida need funktsioonid teevad. Funktsiooni *clear()* abil saab muuta akna taustavärvi ning puhastada eelnevalt joonistatud objekte aknalt. Funktsiooni *draw()* abil on võimalik joonistada objekte aknale.

Ülejäänud peatükkides kasutame sf::RenderWindow klassi, kuid ühe näite teeme selles peatükis sf::Window klassi abil. Sf::RenderWindow klassi kasutame ülejäänud materjalis seetõttu, et selle abil on võimalik aknal olevaid objekte joonistada ja töödelda SFML'i graafika moodulisse kuuluvate klasside funktsioone kasutades. Sf::Window klassi kasutades tuleks meil kaasata lisaks SFML'le uus teek, mis tegeleks objektide aknale joonistamisega ja haldamisega nagu näiteks OpenGL (Open Graphics Library). Sf::Window klassi tasuks kasutada juhul, kui ei ole soovi kasutada SFML'i graafikateeki objektide joonistamiseks ja töötlemiseks, vaid selleks soovitakse kasutada mõnda teist graafikateeki.

Akna loomiseks tuleb meil esmalt kaasata oma programmi graafika moodul, et saaksime erinevaid SFML'i klasse kasutada. Selle jaoks tuleb programmi päisesse kirjutada käsklus *#include<SFML/graphics.hpp>*. Kui on loodud ühendus vastava mooduliga, siis on meil võimalik luua aken kasutades sf::Window ja sf::VideoMode'i klasse (vt koodinäide 1) ning lõpuks1 käsuga *window.display()* väljastada joonistatud objekte aknale. Objektide joonistamiseks tuleks sf::RenderWindow klassi abil akent luues kasutada käsklust *window.draw(objekti nimi)*. Sulgudesse tuleks lisada objekti nimi mida soovite joonistada.

Akna loomise näide sisaldab kahte while tsüklit. Esimene while tsükkel kontrollib kas aken on *avatud (while(aken.isOpen()))* ja teine aknal toimuvaid sündmusi *(while(aken.pollEvent))*. Teise tsüklisse asetame koodilõigud, kus on vaja kuulata mingit sündmust. Näiteks, kas klaviatuuri klahvile a on vajutatud ja kui on, siis käivitub mõni uus sündmus.

```
#include <SFML\Graphics.hpp>

int main (){
     // sf::VideoMode (akna laius,akna kõrgus),”Akna pealkiri”)
     sf::Window aken(sf::VideoMode(800,600),"Esimene aken");

     //Jooksutab programmi nii kaua, kuni aken on avatud.
    while (aken.isOpen())
    {
    /*Kontrollib kõiki akna sündmusi mis käivitusid pärast
viimase tsükli iteratsiooni.*/
        sf::Event event;
        while (aken.pollEvent(event))
        {
   /*Kui me vajutame aknal vasakus nurgas olevat X'i siis
programm sulgub*/
          if (event.type == sf::Event::Closed)
                aken.close();
        }
  //Värskendab akent,kuvab objektid aknale.
     aken.display();
    }

     return 0;

}
```

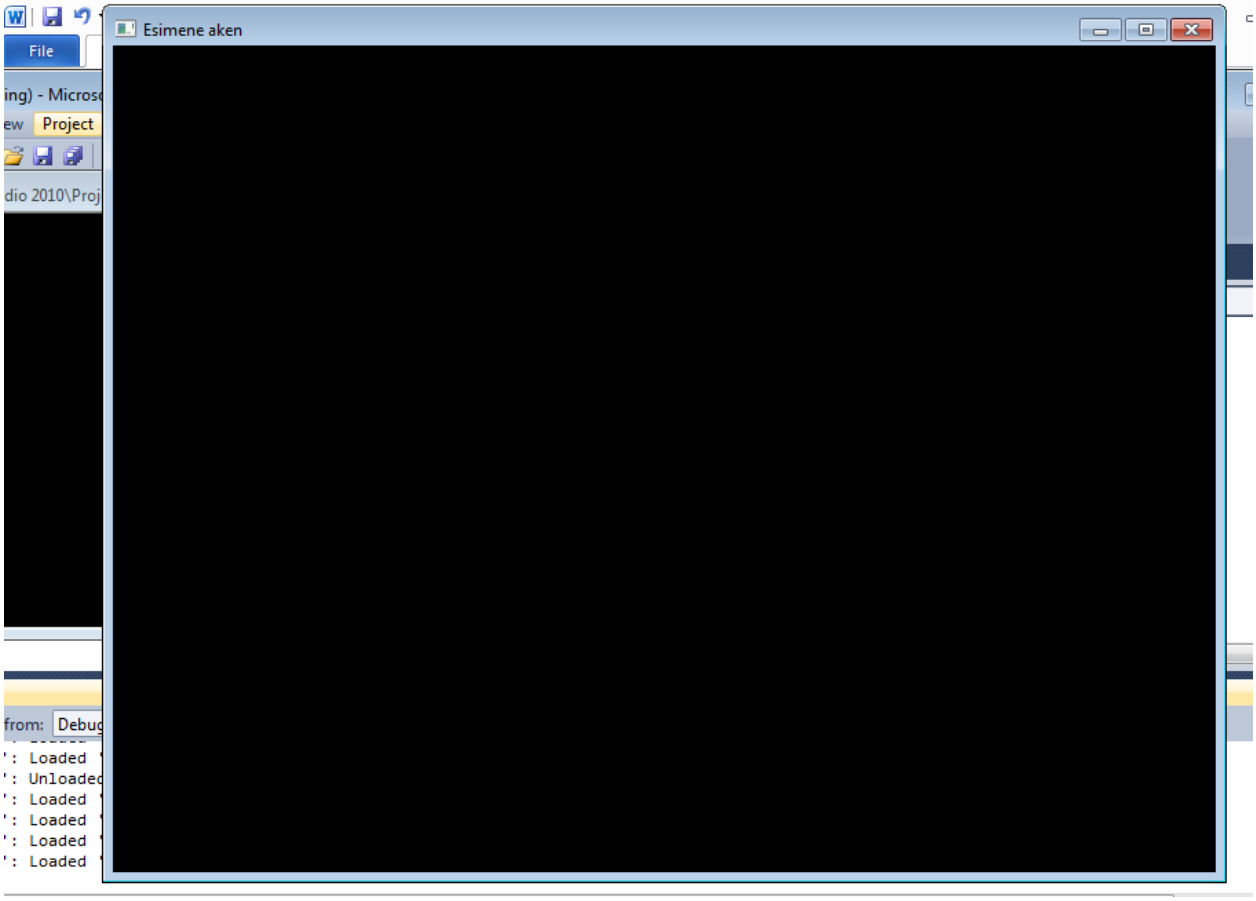
Koodinäide 1. Akna loomine sf::Window klassi kasutades

Lisaks määrame, mitu kaadrit sekundis meie rakendus töötab. Selle jaoks lisame programmi järgneva koodi rea: *aken.setFramerateLimit(60)* (vt koodinäide 2). Hetkel suudab meie rakendus näidata 60 kaadrit sekundis. 60 kaadrit sekundis valisin sellepärast, et tänapäeva kõrgkvaliteediline televisioon ja arvutimonitorid töötavad tavaliselt 60 hertsilisel kaadrisagedusel. Kaadrisagedus määramise abil saate objekte aknal sujuvalt liigutada.

```
sf::Window aken(sf::VideoMode(800,600),"Esimene aken");
aken.setFramerateLimit(60);
```

Koodinäide 2. Kaadrisageduse määramine

Oma programmi kompileerides peaks ilmuma ekraanile uus aken (programmi käivitamiseks vajutage F5) (vt joonis 7).



Joonis 7. Koodinäite 1 tulemus

3.1 Värvimine

SFML'i graafika moodul sisaldab sf::Color klassi värvide kasutamiseks. Sf::Color koosneb neljast 8-bitisest komponendist: r (red), g (green), b (blue) ja a (alpha läbipaistvuseks), mille väärtused on 0...255-ni (Gomila, 2011). Kui anda alpha väärtuseks 0 (nt. *sf::Color color(93,138,168,0)*), muutuvad värvid täiesti läbipaistvaks. Järgnevalt aadressilt leiab erinevate värvide konstantide väärtused RGB's :http://www.colorhexa.com/color-names.

Objektile, saab värvi omistada kahel kujul *sf::Color(0,0,255)* või *sf::Color::blue*. Mõlemal juhul saate sinist värvi objekti. *sf::Color::blue* kasutades kasutan sf::Color klassis olemasolevat värvi ja *sf::Color(0,0,255)* kasutades RGB värvi konstante. Värvi konstante kasutades saate luua erinevaid värve, mida näiteks sf::Color klass ei sisalda. Näiteks soovin saada helesinist värvi. Selle jaoks tuleb kasutada RGB konstante, sest sf::Color klass ei sisalda helesinist värvi. Helesinist värvi saab määrata järgnevalt väärtustega: punast 173, rohelist 216 ja sinist 230 (nt. *objektinimi.setFillColor(sf::Color(173 , 216, 230))*).

Lisades sf::RenderWindow klassi kasutades käsklusele *window.clear()* juurde käskluse *window.clear(sf::Color(255,0,0)）* või *window.clear(sf::Color::Red)*, saame muuta akna taustavärvi.

4 Teksti joonistamine ja töötlemine

Tekstiga töötamiseks on SFML'is kaks klassi sf::Font ja sf::Text. sf::Font tegeleb kirjatüübi laadimisega ja sf::Text teksti joonistamise ning töötlemisega. Lisaks toon näite sellest, kuidas mängu skoori kasutajale kuvada.

4.1sf::Text

Text klass on loodud teksti joonistamiseks ja töötlemiseks. Teksti klass pärib *position, rotation, scale, origin* funktsioonid sf::Transformable klassist. Funktsiooni *setPosition()* kasutades saab muuta teksti asukohta aknal, *rotatate()* pöörab teksti, *setScale()* muudab teksti mõõtmeid ja *setOrigin()* määrab objekti keskpunkti. Lisaks saab määrata teksti poolt kasutatava kirjatüübi, suuruse (pikslites), värvi, stiili (italic, bold, underlined) ja teksti mida väljastada (vt Lisa 4).

Koodinäites 3 loon kõigepealt sf::Text klassi abil objekti millele panen nimeks tekst. Objekti tekst abil on võimalik kasutada erinevaid sf::Tekst klassi kuuluvaid funktsioone (vt koodinäide 3). Pärast määran tekstile kirjasuuruse *setCharacterSize()* funktsiooni abil. Teksti värvimiseks kasutame *setColor()* funktsiooni, mille abil on võimalik värv määrata värvi konstante või sf::Color klassis olemas olevaid värve *kasutades. setPosition(x-koordinaat, y-koordinaat)* funktsiooni abil määrame teksti positsiooni aknal ja *setString()* funktsiooniga saame määrata teksti, mida soovime kasutajale kuvada.

```
#include <SFML\Graphics.hpp>

int main(){

      sf::RenderWindow aken(sf::VideoMode(800, 600), "Tekst");

      sf::Text tekst;
      tekst.setCharacterSize(30);//Kirja suurus
      tekst.setColor(sf::Color::Red);//Teksti värvus
      tekst.setPosition(380, 10);  //Teksti asukoht
      //Tekst mida kuvada
      tekst.setString("Hello World ehk tere,maailm!");
      sf::Event event;

while (aken.isOpen())
     {

      while (aken.pollEvent(event))
         {

                  if (event.type == sf::Event::Closed)
                      aken.close();
               }

                 aken.clear();
                 aken.draw(tekst);
                 aken.display();
          }

           return 0;
}
```

Koodinäide 3. Teksti kasutamise näide

4.2sf::Font

sf::Font klass tegeleb kirjatüübi laadimisega. Kasuks tuleb sf::Font klassi kasutamine juhul kui ei taheta kasutada SFML'i vaikeseadena määratud kirjatüüpi.

sf::Font klass koosneb kahest funktsioonist *LoadFromFile* ja *LoadFromMemory* (vt koodinäide 4). Mõlemad funktsioonid tegelevad fondi laadimisega. Oma näites kasutan ma arvutis olevaid kirjatüüpe Fonts kaustast (vt Lisa2). Kui soovitud kirjatüüp, mida kasutada soovitakse on leitud, tuleks valitud kirjatüüp lisada oma projekti kausta (vt Lisa3).

```
sf::Font minuFont;

// Laeb kirjatüübi faili kettalt
if (!minuFont.loadFromFile("arial.ttf"))
{
    std::cout<<"Kirjatüübi faili laadimine ebaõnnestus"<<endl;
}

// Laeb kirjatüübi faili mälust.
if (!minuFont.loadFromMemory(PointerToFileData, SizeOfFileData))
{
    std::cout<<"Kirjatüübi laadimine mälust ebaõnnestus"<<endl;
}
```

Koodinäide 4. Kirjatüübi laadimise näide.

Kui laadimine õnnestus on tõeväärtuseks tõene (ing.k true) ja vea korral väär (ing false). *If* lause sisse on võimalik lisada juurde veateade, mis eksimuse korral meid koheselt teavitab veast. Koodinäites 4 on toodud vastav näide, mille abil väljastatakse konsooli veateade. Veateate kuvamiseks konsoolis kasutan C++'i käsklust *std::cout<<" Siia lisa oma tekst "<<endl*;. Et seda käsklust kasutada tuleks päisesse lisada *#include<iostream>*, mis tegeleb andmete väljastamise ja sisestamisega C++ keeles.

Eelnevad näited ja seletused tegelevad sõne (ingl.k string) tüüpi muutujate kuvamisega kasutajale. Et kasutajale saadud punktide summat kuvada, tuleks meil täisarvuline (ingl.k integer) muutuja teisendada sõneks. Selle jaoks kasutame c++'i sstream päist, mille sstream klassi abil on võimalik täisarvulisi muutujaid ümber teisendada sõnedeks. Kõigepealt tuleb tekitada sf::Text klassi kasutades uus objekt, mille nimeks olen oma näites pannud skoor ning luua täisarvuline muutuja nimega arv, millele omistan väärtuseks 10. Seejärel kasutame *std::stringstream*'i klassi mille abil tekitame objekti text ja seejärel kasutame käsklust *text.toString(arv)*, mille abil muudame täisarvu sõneks. Tänu sellele saame kuvada sf::String klassi kasutades kasutajale skoori aknale (vt koodinäide 5).

```
#include <SFML\Graphics.hpp>
// Kaasame sstream päise oma programmi päisesse
#include <sstream>

int main(){

sf::RenderWindow aken(sf::VideoMode(800, 600), "Täisarv sõneks");

     // Määrame täisarvulisele muutujale  väärtuse
     int arv=10;
     sf::Text skoor;
     //Loome stringstream klassi objekti
     std::stringstream teisenda;
     //Määrame kuidas kuvada oma sõnum aknale
     teisenda << "SCORE:"<<arv;
     //Teisendame muutuja teisenda sõneks
     skoor.setString(teisenda.str());
      sf::Event event;

while (aken.isOpen())
    {
     while (aken.pollEvent(event))
        {

            if (event.type == sf::Event::Closed)
                aken.close();
        }

          aken.clear();
          aken.draw(skoor);
          aken.display();
    }
     return 0;
}
```

Koodinäide 5. Täisarvu teisendamine sõneks

Ülesanded

1) Loo uus aken sf::RenderWindow klassi kasutades, mille laius on 600 pikslit ja kõrgus 500 pikslit. Lisa akna pealkirjaks oma nimi.
2) Tekita aknale tekst, kus on kirjas sinu nimi, mis on rohelist värvi ning mille kirja suuruseks on 20 pikslit.
3) Ülesandes 1 loodud tekstile lisada alljoon (ingl.k underline) ja paksendatud kiri (ingl. k. bold).
4) Kuva teksti „Hello world ehk tere,maailm!“ (vt koodinäide 3) 4 korda aknale *for* tsüklit kasutades. Iga uus teksti rida, mis on aknale kuvatud peab olema järgmisel real (vt lisa 5). *For* tsükli kohta leiab inglise keelset informatsiooni järgmiselt aadressilt: http://en.cppreference.com/w/cpp/language/for

5 Heli ja muusika mängimine

SFML'il on heli ja muusika kasutamiseks kaks klassi, sf::Sound ja sf::Music. Erinevuseks nende kahe klassi vahel on see, kuidas nad helisid haldavad.

Sf::Sound on klass, mis mängib sf::SoundBuffer'is laetud heli andmeid. Sound klassi tuleks kasutada väikesemahuliste helifailide puhul, mis mahuvad mälusse. (Gomila, 2011). Helifailide suurus võiks jääda alla 1MB sf::Sound klassi kasutades. Pikkade ja mahukate lugude mängimiseks kasutage sf::Music klassi, sest sf::Sound klassi kasutades on helikvaliteet halb.

Sf::Music klass ei lae kogu helifaili andmeid mälusse, voog tuleb otse lähtefailist. Tavaliselt kasutatakse seda klassi tihendatud muusika mängimiseks, mis kestab mitu minutit ja vajab kaua aega laadimiseks ning kasutab sadu megabaite mälust (Gomila, 2011).

SFML 2.0 toetab järgnevaid helifaili formaate: ogg, wav, flac, aiff, au, raw, paf, svx, nist, voc, ircam, w64, mat4, mat5 pvf, htk, sds, avr, sd2, caf, wve, mpc2k, rf64.

5.1 Heli mängimine

Esmalt on meil vaja alla laadida Internetist mõni helifail või kasutada arvutis olemasolevat helifaili. Üks veebilehekülg, millelt saab tasuta ja legaalselt helieffekte alla laadida asub aadressil http://www.mediacollege.com/downloads/sound-effects/ (MediaCollege.com, n.d). Paigutage oma helifail samasse kataloogi, kus asub teie VC++ projekt. Lisaks võiks tekitada uue kausta nimega Data samasse kataloogi, kus asub teie projekt. Lisage helifail Data kausta (vt lisa1).

Esiteks tuleks programmi kaasata audiomoodul, mis võimaldab meil heli ja muusikaga tegeleda. Selle jaoks tuleb lisada päisesse *#include <SFML\audio.hpp>*. Oma programmi kehasse lisage järgnevad koodiread (vt koodinäide 6).

```
sf::SoundBuffer puhver;

    if (!puhver.loadFromFile("Data/laser.wav")){
       return -1;
    }
    sf::Sound heli;
    heli.setBuffer(puhver);
    heli.setVolume(60);
    heli.play();
```

Koodinäide 6. Heliga seotud funktsioonide kasutamise näide.

Algul kutsume välja SoundBufferi ja nimetame ta puhvriks. SoundBuffer klass hoiab audio valimeid (ingl.k audio sample). Järgmisena laeme helifaili puhverisse. Kui helifaili laadimine ebaõnnestub, siis aken sulgub. Käsuga *heli.play()* paneme oma helifaili mängima ja *heli.setVolume(60)* seab helitugevuseks 60. SFML'i helitugevuse puhul on tegu rakenduse sisemise suhtelise mõõtühikuga. Maksimaalne helitugevus on 100 ja minimaalne 0 ehk heli ei ole kuulda. Helitugevust mõõdetakse detsibellides ja helitugevus sõltub arvutipoolt kasutatavast helikaardi ja kõlarite võimekusest.

Käsuga *heli.setBuffer(puhver)* määrame helile puhvri. LoadFromFile käsuga näitame, millist faili laadida ja teed helifaili asukohta.

5.2 Muusika mängimine

Kõigepealt on meil vaja leida pikem muusikapala. Mina kasutan helifaili Kalimba.mp3. Probleem on selles, et SFML 2.0 ei toeta mp3 formaati. Seega tuleb see konvertida ümber näiteks ogg formaati, mida SFML 2.0 toetab. Kasutan selleks järgmisel aadressil asuvat online konverterit http://audio.online-convert.com/convert-to-ogg%20%20. Kui muusikafail konverditud, tõstame faili oma projekti data kausta.

Muusika mängimiseks on meil vaja välja kutsuda sf::Music klass, et me saaksime avada oma faili ja selle käivitada. Kui heli mängimiseks ütlesime puhvrile kuhu laadida helifail, siis sf::Music klassi kasutades tuleb voog otse lähtefailist. Seetõttu kasutame *loadFromFile* asemel *openFromFile* funktsiooni (vt koodinäide 7).

```
#include <SFML\Graphics.hpp>
#include <SFML\audio.hpp>

int main(){
     sf::Window aken(sf::VideoMode(800, 600), "Esimene Aken");
     sf::Music music;
     if (!music.openFromFile("Data/Kalimba.ogg"))
                return -1; // error
     music.play();
     while (aken.isOpen())
    {
        sf::Event event;
        while (aken.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                aken.close();
        }
          aken.display();
    }
     return 0;
}
```

Koodinäide 7. Muusika laadimise ja mängimise näide

Lisaks on võimalik nii heli kui muusikaga kasutada järgnevaid funktsioone: *play*, *pause*, *stop*, *setPlayingOffset* (Gomila, 2011). Töötava näite leiab aadressilt: http://www.tlu.ee/~neira/SEM/Heli.rar

6 Lihtsamate kujundite joonistamine

Selles peatükis tegeleme erinevate kujundite joonistamise, värvimise ja aknale kuvamisega. Lisaks toon näiteid sellest, kuidas objektide asukohta määrata ja erinevat tüüpi kujundeid joonistada ning seletan, mis on geomeetrilised primitiivid.

6.1 Ringi ja ristküliku joonistamine

Peatükis käsitleme meetodeid, mille abil saab määrata ja muuta erinevate geomeetriliste kujundite suurust, värvi ning positsiooni. Esmalt tutvume sf::RectangleShape klassiga ning seejärel lühidalt sf::CircleShape klassiga.

Erinevate klasside abil joonistatud kujundite kasutamise täieliku näite leiate aadressilt: http://www.tlu.ee/~neira/SEM/Kujundid.rar

RectangleShape klass tegeleb ristüliku kujuliste objektidega. Koodinäide 8 kasutab sf::RectangleShape klassi ruudu joonistamiseks.

```
#include <SFML\Graphics.hpp>

int main (){

sf::RenderWindow window(sf::VideoMode(800, 600),"SFML:Kujund ");
      window.setFramerateLimit(60);

      sf::Event event;

      sf::RectangleShape alus1;
      alus1.setSize(sf::Vector2f(20, 20));
      alus1.setPosition(50, 200);
      alus1.setFillColor(sf::Color::Red);
      alus1.setOutlineThickness(2);
      alus1.setOutlineColor(sf::Color(0,0,255));
      while (window.isOpen())
      {
            while (window.pollEvent(event))
            {
                  if (event.type == sf::Event::Closed)
                  {
                              window.close() ;
                  }
            }
            window.clear();
            window.draw(alus1);
            window.display();
        }
        return 0;
}
```

Koodinäide 8. Ruudu joonistamine sf::RectangleShape klassi abil.

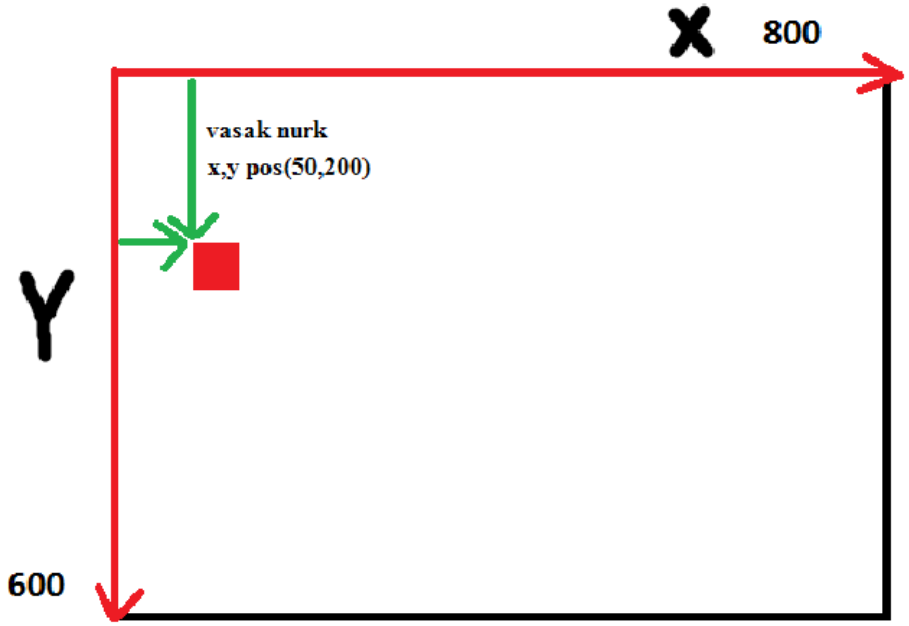
Koodis on näha, et kõigepealt kutsume välja sf::RectangleShape ja paneme talle nimeks alus1. Ristkülikule lisame veel suuruse *alus1.setSize(sf::Vector2f(laius,kõrgus))*.

Ruudu saame me selle abil ,et ristküliku laiuseks on 20 pikslit ja kõrguseks 20 pikslit(tekib ruut). *alus1.setPosition(50,200)* määrab ära, kuhu me oma objekti aknal paigutame (vt joonis 8).

Anname risküliku info tema x ja y koordinaadi asukoha kohta. Jooniselt on näha, et arvutimaailmas on alla suunduv y telg positiivne, mitte negatiivne. *Alus1.setFillColor(sf::Color::Red)* annab ruudule punase värvuse, kasutades sf::Color klassi.

Koodinäites 8 olen lisanud ruudule piirjoone paksuse *alus1.setOutlineThickness(piirjoone paksus ujukomaarvuga)* ja piirjoone värvi *alus1.setOutlineColor(sf::Color(0,0,255))*, mis on sinist värvi.

Viimaste sammudena lisame käskluse *window.draw()*, mis joonistab etteantud objekti aknale ning enne seda *window.clear()*, mis puhastab ja täidab akna musta värviga.



Joonis 8. Objekti asukoha määramine aknal.

Ringi joonistamine on üldiselt sarnane ristküliku joonistamisega. Ainuke erinevus on selles, et *setSize* asemel on *setRadius(ujukomaga arv)* ja kasutatakse sf::CircleShape klassi (vt koodinäide 9).

```
sf::CircleShape alus2;
     alus2.setRadius(20.0);
     alus2.setFillColor(sf::Color::Magenta);
     alus2.setPosition(400,300);
```

Koodinäide 9. Näide sf::CircleShape klassi kasutamisest.

6.2 Geomeetriliste primitiivide joonistamine

Geomeetrilisteks primitiivideks nimetatakse lihtsamaid geomeetrilisi objekte (nt. joon või punkt). Geomeetriliste primitiivide joonistamiseks kasutame sf::VertexArray klassi. VertexArray klass on massiiv, mis koosneb tippudest ja geomeetrilistest primitiividest . VertexArray abil loodud objekte ei ole võimalik transformeerida sf::Transformable klassi kuuluvate funktsioonide abil (nt. pöörata funktsiooni *rotate()* abil või liigutada funktsiooniga *move()*).

```
//sf::VertexArray objnimi(geomeetriline primitiiv, tippude arv)
sf::VertexArray joon(sf::LinesStrip, 4);
//Esimese tipu koordinaadid
joon[0].position = sf::Vector2f(100, 10);
//Teise tipu koordinaadid
joon[1].position = sf::Vector2f(200, 50);
//Kolmanda tipu koordinaadid
joon[2].position = sf::Vector2f(100, 50);
//Neljanda tipu koordinaadid
joon[3].position = sf::Vector2f(100, 10);
//Esimese joone värv.
joon[0].color = sf::Color::Blue;
```

Koodinäide 10. Joonte abil kolmnurga joonistamine.

Ülaltoodud koodi abil saame joonistada joonte abil kolmnurga, mille üks külg on sinist värvi (vt koodinäide 10).

Järgnevalt on loetletud geomeetrilised primitiivid, mida VertexArray sisaldab:

sf::Points- individuaalsete punktide kogum.

sf::Lines- individuaalsete joonte kogum.

sf::LineStrip- omavahel ühendatud punktide kogum (kasutab eelnevat punkti, et moodustada joon kahe punkti vahel).

sf::Triangles- individuaalsete kolmnurkade kogum.

sf::TrianglesStrip- omavahel ühendatud kolmnurkade kogum, kus üks punkt kasutab kahte eelnevat punkti kolmnurga moodustamiseks.

sf::TrianglesFan - omavahel ühendatud kolmnurgad, kus kolmnurk saadakse ühise keskpunkti ja kahe eelneva punkti abil.

sf::Quads- individuaalsete nelikute kogum.

Kolmnurka, mida on võimalik värviga täita saab joonistada, sf::Triangles, sf::TrianglesStrip või sf::TrianglesFan klassi kasutades. Koodinäites 11 joonistame sf::Triangles shape klassi kasutades kolmnurga, mis on punast, rohelist ja sinist värvi.

```
sf::VertexArray kolmnurk(sf::Triangles,3);
     kolmnurk[0].position=sf::Vector2f(400,400);
     kolmnurk[0].color=sf::Color::Red;
     kolmnurk[1].position=sf::Vector2f(400,300);
     kolmnurk[1].color=sf::Color::Green;
     kolmnurk[2].position=sf::Vector2f(500,400);
     kolmnurk[2].color=sf::Color::Blue;
```

Koodinäide 11. Kolmnurga klassi abil kolmnurga joonistamine.

Ülesanded

1) Joonista kolmnurk, ruut ja ristkülik.
2) Väljasta ekraani keskele ring, mis oleks rohelist värvi. Akna taust värvi siniseks.
3) Joonista sf::Lines klassi kasutades ristkülik, mille jooned oleks punast värvi.
4) Selles ülesandes hakkame joonistama maja. Maja seina, katuse ja akna värvid värvib igaüks enda maitse järgi. Maja joonistamist alustame ristkülikuga, mis oleks maja sein ja selle seina peale asetame kolmnurga (kasutage kolmnurga joonistamiseks sf::Triangles klassi), mis on meie maja katuseks. Kui katus ja sein on valmis, lisame majale akna. Akna loomiseks peame joonistama veel ühe ristküliku. Väiksema ruudu sisse joonistame omakorda risti sf::Lines klassi abil (vt. lisa 6).

7 Piltide kasutamine ja objektidele lisamine

Sfml'is on piltide salvestamiseks, laadimiseks ja töötlemiseks sf::Image ning piltide kuvamise jaoks on sf::Texture klass. Nende kahe klassi abil saame lisada varem valmis joonistatud objektide peale pildid või tekstuurid. SFML'i poolt toetatud pildi formaadid on bmp, png, tga, jpg, gif, psd, hdr, pic.

7.1 Tekstuur

Sf::Texture klassis on pildiandmed salvestatud graafikakaardi mälusse. Seetõttu toimub tekstuuri joonistamine objektile kiiresti. Tekstuuri klassi miinuseks on see, et sf::Image klassis on lihtsam pilte töödelda kui sf::Texture klassis. Tekstuuri saab laadida *loadFromFile* (laeb tekstuuri kettal olevast failist), *loadFromStream* (laeb tekstuuri voost), *loadFromMemory* (laeb tekstuuri mälus olevast failist) ja *loadFromImage* (laeb tekstuuri pildist) funktsioonide abil (Gomila, 2011).

7.2 Pilt

Sf::Image klass omab funktsioone pikslite lugemiseks, kirjutamiseks, laadimiseks ja salvestamiseks. See klass kasutab RGBA 32 bittist värvusruumi, mis tähendab, et lisaks punasele (ingl.k red), rohelisele (ingl.k green) ja sinisele (ingl.k blue) värvile on lisaks alfakanal(ingl.k alpha channel). Alfakanali abil saame muuta pildi läbipaistvust. Pilte ei saa joonistada *draw()* funktsiooni abil. Selle jaoks tuleb töödeldud pilt laadida tekstuuri (vt koodinäide 12).

```
#include <SFML\Graphics.hpp>

int main(){

sf::RenderWindow window(sf::VideoMode(800, 600), "Laadimine");

sf::Image pilt;

if (!pilt.loadFromFile("Data/pad.png")){

      return -1;

     }
//Laeb tekstuurile pildi
sf::Texture tekstuur;
tekstuur.loadFromImage(pilt);

sf::RectangleShape objekt;
//Määran tekstuuri, adresseerides(&).
objekt.setTexture(&tekstuur);
objekt.setSize(sf::Vector2f(50,50));
objekt.setPosition(550,370);//Määran x ja y positsiooni

while (window.isOpen())
    {

        sf::Event event;
        while (window.pollEvent(event))
        {

            if (event.type == sf::Event::Closed)
                window.close();
        }

          window.clear();
          window.draw(objekt);
          window.display();
    }



     return 0;
}
```

Koodinäide 12. Tekstuurimine, pildi laadimine ja tekstuuri objektile omistamine

Ülesanded

1) Tekita ring ja laadi ringile mõni pilt ringikujulisest objektist nagu näiteks pall. Ringi raadiuseks pane 50 pikslit.
2) Värvi ülesandes 1 tekitatud objekt punast värvi.

8 Klaviatuuri ja hiire sisendi kasutamine

Selles peatükis käsitleme, kuidas pöörduda klaviatuuri klassi ning hiire klassi poole, et nende abil objekte liigutada oma rakenduses.

Samas pean vajalikuks lisada, et sf::Keyboard ja sf::Mouse klassid ei tegele objektide liigutamisega. Erinevate kujundite joonistamise, töötlemise ja loomisega seotud klassid sisaldavad käsklust *objektiNimi.move(x-koordinaat,y-koordinaat)*, mille abil on võimalik objekte liigutada. Sf::Keyboard ja sf::Mouse klassid tegelevad sisendiga (klahvile ja hiirele vajutuste kuulamisega).

8.1 Klaviatuur

Tööks klaviatuuriga kasutame sf::Keyboard klassi. See klass sisaldab ainult ühte funktsiooni *isKeyPressed*, mis kontrollib klahvi seisundit. Fikseerides, kas klahvile vajutatakse või on klahvist lahti lastud.

Järgnev koodijupp liigutab klahvi A vajutades ristküliku vasakule poole 1 piksli võrra ja klahvi B vajutades pöörab 30 kraadi (vt koodinäide 13).

```
if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
                {
/*Kui klahvile A on vajutatud liigub shape1 objekt vasakule 1 px
võrra*/
                   shape1.move(-1, 0); // shape1.move(x,y);

                }

if (sf::Keyboard::isKeyPressed(sf::Keyboard::R))
                {
//Kui klahvile R on vajutatud pöörab shape1 objekt 30 kraadi.
                   shape1.rotate(30.0);// shape1.move(ujukomaarv);

                }
```

Koodinäide 13 . sf::Keyboard klassi kasutuse näide klahvile vajutuse kuulamisest ja sündmuste käivitamisest.

Hetkel kontrollime *if* lause abil, kas klahvile on vajutatud. Kui klahvile on vajutatud, siis liigub objekt vasakule. Klahvile mitte vajutamise korral objekt peatub.

Lisaks on võimalik liigutada objekte sf::Event::KeyPressed ja sf::Event::KeyReleased abil. Soovitav on kasutada sf::Keyboard klassi, sest see tagab sujuvama liikumise.

Järgnevalt aadressilt on võimalik leida näidisrakendus liikumise kohta kommenteeritud koodiga: http://www.tlu.ee/~neira/SEM/ ja valida Liikumine1.rar.

8.2 Arvutihiir

Hiire kasutamiseks on meil vaja kasutada sf ::Mouse klassi. Nagu ka sf::Keyboard, koosneb sf::Mouse staatilistest funktsioonidest. SFML toetab viite hiirega seonduvat nuppu: vasak, parem, keskmine (ratas) ja veel kahte lisanuppu.

Järgnev koodijupp kontrollib, kas klahvile on vajutatud ja klahvile vajutuse korral muudab risküliku värvi roheliseks (vt koodinäide 14).

```
if (sf::Mouse::isButtonPressed(sf::Mouse::Left))
{
//vasakule hiire nupule klõpsates muutub ristküliku värv
    ristkylik.setFillColor(sf::Color::Green);
}
```

Koodinäide 14. Klahvi vajutust kontrolliv koodijupp.

Lisaks klaviatuuri ja hiire abile on võimalik erinevaid objekte liigutada kaadrisagedusega (vt lisa 7).

Hiire koordinaadid meie loodud aknal on võimalik kätte saada hiire klassis oleva funktsiooni *getPosition()* abil. Funktsiooni sisse tuleb lisada sf::RenderWindow klassi abil loodud akna *objekt (nt. getPosition(akna objekti nimi))*, et saaksime hiirekursori koordinaadid meie loodud aknalt, mitte kogu arvuti ekraanilt.

Järgnevalt loome punast värvi ristküliku, mis hiire vasakule nupule vajutades muutub roheliseks. Ristküliku laiuseks määran 50 pikslit ja kõrguseks 40 pikslit. Lisaks loome objekti hiir ja hiljem küsime hiire x ja y koordinaadid aknalt loodud hiire objekti abil (vt koodinäide 15).

```
//Teegid
#include <SFML/Graphics.hpp>

int main (){
	sf::RenderWindow window(sf::VideoMode(800,600,32),
		"Nupu loomine");
	window.setFramerateLimit(60);
	//Muudab hiire kursori nähtamatuks
	sf::Event event;

	sf::RectangleShape ristkylik;
	ristkylik.setPosition(350,300);//Positsioon
	ristkylik.setSize(sf::Vector2f(100,50));//Objekti suurus
	ristkylik.setFillColor(sf::Color::Red);//Värv
	sf::Mouse hiir;//Loome objekti hiir
```

Koodinäide 15. Ristküliku ja hiire objekti loomine

Järgnevas koodiosas kontrollime *if* lause abil, kas hiire koordinaadid asuvad meie loodud ristküliku sees ja kas hiire nupule vajutus on toimunud. Kui hiire nupule vajutus on toimunud, siis värvime loodud ristküliku roheliseks.

Kontrollimiseks kasutame järgmist valemit: hiire x koordinaat > ristküliku x koordinaat && hiire x koordinaat < ristküliku x koordinaat + ristküliku laius && hiire y koordinaat > ristküliku y koordinaat && hiire y koordinaat < risküliku y koordinaat + ristküliku kõrgus.

Valemis kasutatud märk && tähendab sõna ja. Kui hiire x koordinaat on 355 ja y koordinaat 305 siis järgneva valemi abil teame, et hiire kursor asub ristküliku sees.

Hiire x ja y koordinaadid saame funktsioonide *getPosition(window).x* ja *getPostition(window).y* abil. Ristkülikul ei ole vaja getPosition() funktsiooni sisse lisada akna nimetust, sest funktsioon tagastab ruudu koordinaadid ainult aknal. Ruudu laiuse ja kõrguse tagastavad funktsioonid *getSize().x* ning *getSize().y* (vt. koodinäide 16).

```
if(hiir.getPosition(window).x> ristkylik.getPosition().x &&
hiir.getPosition(window).x < ristkylik.getPosition().x+
ristkylik.getSize().x && hiir.getPosition(window).y>
ristkylik.getPosition().y && hiir.getPosition(window).y<
ristkylik.getPosition().y+ ristkylik.getSize().y &&
hiir.isButtonPressed(sf::Mouse::Left) ){

                    ristkylik.setFillColor(sf::Color::Green);

                    }
```

Koodinäide 16. Kontroll, kas hiir asub nupul ja hiire nupule on vajutatud

Kuna hetkel jääb meie nupp pärast vajutust punaseks, siis lisame else lause oma *if* lausele, mis ütleb, et kui nupule ei ole vajutatud ning hiir ei asu nupul, siis nupp on roheline (vt. Koodinäide 17).

```
if(//siin asub kontroll, mis on eelnevalt kirjeltatud){

            ristkylik.setFillColor(sf::Color::Green);
}else{
            ristkylik.setFillColor(sf::Color::Red);
      }
```

Koodinäide 17. Else lause nupu värvi muutmiseks.

Terviklik kood asub lisades punktis 8.

Ülesanded

1) Joonista ekraanile ruut, mida on võimalik liigutada vasakule, paremale, üles ja alla klaviatuuri abil.
2) Eelmises ülesandes joonistatud ruudule lisada võimalus ruudu pööramiseks. Ruudu pööramiseks kasuta käsku: *objektinimi.rotate(pöörde nurk ujukomaarvuga (nt. 20.0))*.
3) Kolmandas peatükis loodud näitele lisada klaviatuuri nuppudele erinevad funktsioonid (*stop, play, pause*), mis vastava nupu vajutuse korral käivitab funktsiooni.
4) Lisaks kuva kasutajale välja, mis nupp mida teeb ja mis nupule hetkel on vajutatud. Ülesannete 3 ja 4 lõpptulemuse leiab aadressilt: http://www.tlu.ee/~neira/SEM/%dcle3ja4Tulemus.rar

5) Kontrolli kas vasakule hiire nupule on vajutatud. Kui hiirenupule on vajutatud, väljasta konsooli aknasse cout<<”Hiire nupule vajutati”;. Programmi päisesse kaasa *#inlcude <iostream>* ja enne main funktsiooni kirjuta *using namespace std ;*, et saaks kasutada nimeruumi (*cout* ette ei pea lisama *std*).

9 Põrke tuvastus

Põrke tuvastusega seonduvas peatükis võtame läbi lihtsaima põrke tuvastusega seonduva funktsiooni ning toome vastava näite. Oma näites kasutame ristkülikut ja ringi. Näidis on leitav järgmisel aadressil: http://www.tlu.ee/~neira/SEM/kollisioon.rar

Põrke tuvastamiseks leidub erinevaid algoritme ja lahendusi. Selles õppematerjalis kasutame me selle jaoks sf::Shape klassi kuuluvat funktsiooni *getGlobalBounds().intersect*, mille abil on võimalik teada saada, kas kaks objekti on omavahel põrkunud (vt koodinäide 18).

```
/*Kontrollime,kas ring põrkub ruut1-ga.Kui põrkub, siis ring
liigub 5 x ja y koordinaadis tagasi.*/

if(ring.getGlobalBounds().intersects(ruut1.getGlobalBounds())
        ==true){
     //Ring liigub -5 pikslit mööda x,y koordinaati.
     ring.move(-5,-5);
     //Muudame kokkupõrke korral ringi värvi
     ring.setFillColor(sf::Color::Green);
}
```

Koodinäide 18. Põrke tuvastus getGlobalBounds() funktsiooni abil.

GetGlobalBounds funktsioon annab objekti mõõtmed ja kontrollib antud hetkel, kas ring ja ruut on omavahel kokku põrganud. Kui kokkupõrge on toimunud, siis muutub ringi värv roheliseks.

Kui tahame tekitada palli, mis põrkab meie loodud akna vahel, tuleb meil leida palli x või y positsioon ja võrrelda palli y positsiooni akna y positsiooniga. Ykiirus ja Xkiirus muutujad määravad palli kiiruse vastavalt mööda x ning y koordinaati (vt koodinäide 19).

```
if(pall.getPosition().y < 0){
                Ykiirus=-Ykiirus;
                Xkiirus=0;
          }

if(pall.getPosition().y > 570){
                Ykiirus=-Ykiirus;
                  Xkiirus=0;
          }

pall.move(Xkiirus,Ykiirus);
```

Koodinäide 19. Põrke kontrollime palli ja akna vahel.

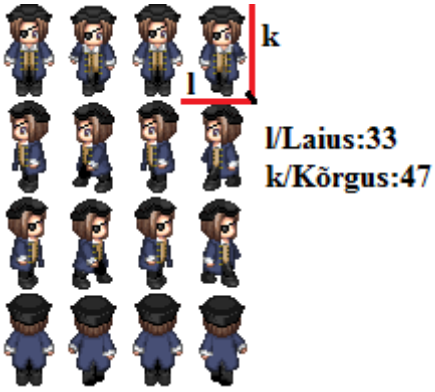
Kui palli positsioon on väiksem kui null, siis pall põrkub ülevalt alla, teise if lause korral ülesse. Akna kõrgusest, mis oli 600 pikslit peab lahutama 30 pikslit (30 pikslit on palli raadius), et tekiks aknal pallipõrke kujutis. Eeltoodu tehte mittesooritamisel näeksime, kuidas pall aknast välja liigub ja siis tagasi põrkub.

Ülesanded

1) Joonista ring ja pane ring akna vahel mööda y koordinaati edasi tagasi põrkuma.
2) Joonista kaks kujundit. Esimest kujundit peab saama klaviatuurilt juhtida. Kontrolli , kas kujundid põrkuvad omavahel. Kujundite põrkumise korral põrkub liigutatav kujund tagasi.
3) Joonista aknale ristkülik ja kontrolli, kas kursor on risküliku sees. Kui kursor on ristküliku sees, siis vaheta ristküliku värvi.

10 Sprite

Arvutigraafikas nimetatakse animeeritud pilte sprite'deks. Sprite on üks pilt või piltide jada, mille abil tekitatakse ekraanil olevale objektile piltidest animatsioon. Näiteks inimese liikumise jäljendamiseks, kes saab liikuda üles, alla, vasakule ja paremale on meil vaja mitut asendit ja seisundit ühe objekti jaoks. Järgneval joonisel on inimese liikumist kujutav sprite'i leht (ingl.k sprite sheet).



Joonis 9. Inimese liikumist kujutav sprite'i leht.

Sprite sheet on kogumik piltidest või joonistustest, kus objekt on mitmes asendis. Vastava sisendi korral ehk nupuvajutusel vahetub pilt ja tekib kujutlus kõndivast inimesest või mõnest muust liikuvast objektist.

SFML'is tekitatakse sprite sheet'i peale ristkülik, mis katab kogu sprite sheet'i ja nupuvajutusel näidatakse vastavat osa pildist ehk üht osa ristkülikust, mis sprite sheet'i katab tehakse nähtavaks sprite sheet'i peal oleva alamristküliku abil.

SFML'is tuleb esmalt pilt laadida. Selle jaoks kasutame käsklust *loadFromFile()* ja sf::Texture klassi (vt koodinäide 20). Pildi taust tuleks eelnevalt mõnes pilditöötlusprogrammis läbipaistvaks muuta. Näiteks vabavaralise pilditöötlusprogrammi GIMP abil.

Inglise keelse videoõpetuse sellest, kuidas GIMP'i kasutades tausta läbipaistvaks muuta leiab järgmiselt aadressilt: http://www.youtube.com/watch?v=jMF8fIjW7Ls

```
sf::Texture pilt1;

if (!pilt1.loadFromFile("Data/pirate_2m.png")){
        return EXIT_FAILURE;

      }
```

Koodinäide 20. Pildi laadimine sf::Texture klassiga.

Järgmisena kasutame sf::Sprite klassi, millel on samad funktsioonid, mis teistel joonistamisega seotud klassidel nagu näiteks sf::RectangleShape klassil. Lisaks määrame talle tekstuuri ja positsiooni. (vt koodinäide 21).

```
int width=33;//Alamristküliku laius.
int heigth=47;//Alamristküliku kõrgus
int left=0;
int top=0;

sf::Sprite sp;

sp.setTexture(pilt1);

sp.setTextureRect(sf::IntRect::Rect(left,top,width,heigth));
sp.setPosition(200,200);
```

Koodinäide 21. Tekstuuri ja positsiooni määramine.

Kui pildi laadimine on õnnestunud, määrame väärtused muutujatele ülevalt,vasakult (top, left) piltide vahel navigeerimiseks. Algselt on meie tegelane nagu ka joonisel 9 ülesse liikumise suunas ehk muutujate top ja left väärtusteks on null. Kui tahaksime järgmisena saada pilti teisest asendist, peaksime liitma top'le juurde 47 pikslit. Kui pilt on teinud 4 sammu siis tahame, et pilt oleks jälle algseisundis. See tähendab, et sprite on 4*33=132 (seisundite arv*laius) pikslini jõudnud sprite sheet'il ja me peame ütlema, et sprite oleks jälle seisundis, kus top=0 ja left=0. Et saada sprite vasakule liikumise animatsioon peab top=48 ja left=0. Ning algab sama tsükkel, kus iga kord liidame left'le 33 pikslit juurde (vt koodinäide 22).

Näidises kasutatava pildi leiab aadressilt: http://www.tlu.ee/~neira/SEM/pirate_m2.png

```
while (play==true)
      {

                  while (window.pollEvent(event))
            {
                  if(event.type== sf::Event::Closed)
                  {
                        play=false;//lõpetab mängu tsükli

                  }
      if (event.type == sf::Event::KeyPressed && event.key.code ==
      sf::Keyboard::Down)

                  {
                        sp.move(0,5);
                        top=143;
                        left+=33;

                  }

      if (event.type == sf::Event::KeyReleased && event.key.code
      == sf::Keyboard::Down)
                  {
                        sp.move(0,0);

                  }

      if(left>=132)
                  {

                  left=0;

                  }

       }
      window.clear();
      //Värskendame alamristküliku muutujaid
      sp.setTextureRect(sf::IntRect::Rect(left,top,width,heigth));
      window.draw(sp);//Joonistame sprite
      window.display();//Kuvame objektid aknale


      }
```

Koodinäide 22. Sprite'i liigutamise koodinäide.

Töötav näidis on saadaval aadressil:http://www.tlu.ee/~neira/SEM/SEM_Sprite.rar

Kuna hetkel ekraanile kuvatav sprite on üsna väikeste mõõtmetega, siis saate kasutada *sp.setScale(2,2)* käsku, et muuta sprite kaks korda suuremaks. Kood tuleks lisada peale sprite'le muutujate omistamist ja enne sprite aknale joonistamist.

Õppematerjali proov

Valminud SFML õppematerjali sai testitud informaatika tudengite abil, C++ kursuse raames. Prooviloengu käigus ilmusid välja vead, mida autor ise ei märganud. Loengul katsetati valmis-olevaid näiteid, lahendati ülesandeid ja seadistati ülesse keskkond rakenduste loomiseks SFML'ga. Loengus kasutasime MS Visual C++ Expressi 2008 ja SFML 2.0'i. Seetõttu ei saanud õppematerjali teist peatükki täielikult kasutada, sest MS Visual C++ Express 2008 SFML'ga linkimine erineb MS Visual C++ Express 2010'st .

Järgnevalt hakkasid õppurid õppematerjaliga lähemalt tutvuma. Peamisteks probleemideks olid õppurite poolt tehtud pisivead või minu halvasti sõnastatud ülesannete ja seletuste pärast tekkinud vead. Näiteks üritati sf::Sound klassi laadida mahukat muusika faili, mis tuleks laadida sf::Music klassi kasutades. Põhjuseks võis olla kiirustamine, mille tõttu jäi peatüki algus lugemata, kus on kirjutatud sf::Music ja sf::Sound klassi erinevustest või ei suutnud mina kõigile arusaadavalt selgitada sf::Music ja sf::Sound klassi erinevusi.

Loengu lõpus lahendasid tudengid ülesandeid. Ülesanded koostasin eelnevates peatükkides käsitletud teemade põhjal. Õppematerjalile lisasin Lisade peatüki juurde mõnede ülesannete tulemustest pildid, et tulemus oleks üheselt mõistetav ja lahendajal oleks selgem arusaam sellest, milline peab olema lahenduse lõpptulemus. Ülesannete lahendamisega said tudengid väga hästi hakkama, mistõttu võib ülesannete lahendamist lugeda edukaks. Lisaks sai autor tagasisidet mõningate näidete kohta, mis üliõpilaste sõnul oleks võinud olla pikemad.

Algselt lisasin ma peatükkidesse lühikesed koodinäited, et lugejal oleks lihtsam keskenduda ning jälgida seda koodiosa mida peatükis lahti seletatakse. Pärast prooviloengut lisasin peatükkidesse juurde tervikud koodinäited lisaks lühikestele koodinäidetele, et oleks aru saada, kus milline koodiosa tervikus koodis asub. Sellega sai kõrvaldatud oht, et lugejal jääb koodist mõni vajalik koodiosa puudu ja proovimise korral programm ei käivitu.

Kõige rohkem probleeme tekkis minul SFML 2.0'i kasutama õppides sprite'de kasutamisega, sest mul puudus eelnev kogemus ja täpne arusaam sellest kuidas sprite kasutada. Olulist informatsiooni leidisin sellel hetkel SFML'i foorumis olevatest näidetest, kuidas sprite kasutada. Seetõttu lisasin sprite käsitleva peatüki õppematerjali, sest sprite'de abil on võimalik muuta mängu reaalsemaks ja huvitavamaks.

Kokkuvõte

Õppematerjalis tutvustatakse lühidalt SFML'i graafika, akna, audio ja süsteemi mooduliga seonduvaid klasse ning funktsioone. Õppematerjal käsitleb ainult väikest osa SFML'st ja tema kasutamise võimalustest. Lisaks sisaldas õppematerjal MS Visual C++ 2010 Express'ga uue projekti loomist ja SFML'ga linkimist. Viie peatüki lõpust leiab ülesanded, mis aitavad õpilase poolt omandatud teadmisi kinnitada ja proovile panna.

Õppematerjali testiti prooviloengus, kus informaatika tudengid lahendasid ülesandeid ja tutvusid õppematerjaliga. Prooviloengus tulid välja mõned vead, mida õppematerjal sisaldas ja mis said hiljem eemaldatud. Prooviloengust saadud tagasiside oli positiivne.

Õppematerjali luues omandasin teadmisi mängude ja õppematerjali loomisest. Ühtegi õppematerjali ei ole ma varem loonud. Õppematerjali luues oli kõige raskemaks ülesandeks valida teemasid mida käsitleda. SFML'i puhul on tegemist suure multimeediumi teegiga mis hõlmab mitmeid mooduleid, mistõttu ei ole võimalik seminaritöös kõike käsitleda arvestades seminaritöö koostamiseks antud aega.

Teemad valisin lähtudes erinevates õppematerjalides käsitletavate teemade põhjal ja mulle probleeme valmistanud teemade põhjal. Nagu sissejuhatuses sai kirjeldatud, siis üheks probleemiks osutus minul SFML'ga alustades sprite'de kasutamine. Eelneva kogemuse puudumisel või vastava tehnika puudumise tõttu jätsin välja veel mitmed teemad. Näiteks juhtkangi käsitlevat teemat ei lisanud ma kaheksandasse peatükki seetõttu, et mul puudus juhtkang. Lisaks paljud inimesed ei oma juhtkangi ja need kes omavad saavad kasutada minu poolt lisatud viidet video juhendite lingile, mis on lisatud esimesse peatükki, kus kasutaja CodingMadeEasy tutvustab SFML'i. Videodes tutvustab YouTube kasutaja CodingMadeEasy kuidas erinevaid SFML'i komponente kasutada ja mida on nende komponentide abil võimalik teha. Videod on inglisekeelsed.

Õppematerjal on eelkõige mõeldud tutvustama SFML'i ja tekitama huvi selle vastu. Õppematerjali läbi töötanud õpilane peaks olema võimeline looma mõne lihtsama töötava rakenduse.

Kasutatud materjalid

Baudry, C. (2012). SFML 2-Tutorial-Introduction. Kasutamise kuupäev: 19.12.2012. a., allikas: http://riseagain.wordpress.com/2012/07/15/sfml-2-tutorial-introduction/

CodingMadeEasy. C++ Sfml 2.0 Made Easy. Kasutamise kuupäev: 26.12.2012. a., allikas: https://www.youtube.com/watch?v=kAbkFY6lwAY&list=SPHJE4y54mpC5j_x90UkuoMZOdmmL9-_rg

Convert audio to OGG format. Kasutamise kuupäev: 26.12.2012. a., allikas: http://audio.online-convert.com/convert-to-ogg.

Gomilia, L. (2011). Documentation. Kasutamise kuupäev: 15.11.2012. a., allikas http://www.sfml-dev.org/documentation/2.0/

Gomilia, L. (2011). Features. Kasutamise kuupäev: 15.11.2012. a., URL http://www.sfml-dev.org/features.php

Gomilia, L. (2011). sf::Texture Class Reference. Kasutamise kuupäev: 08.01.2013. a., allikas: http://www.sfml-dev.org/documentation/2.0/classsf_1_1Texture.php

Gomilia, L. (2011). SFML. Kasutamise kuupäev: 15.11.2012. a., allikas: http://www.sfml-dev.org/.

Gomilia, L. (2011). Tutorial-Audio-Playing sounds and musics. Kasutamise kuupäev: 15.11.2012. a., allikas: http://www.sfml-dev.org/tutorials/2.0/audio-sounds.php

Gomilia, L. (2011). Tutorials. Kasutamise kuupäev: 24.12.2012. a., allikas: http://www.sfml-dev.org/tutorials/2.0/

Harbour, S.J. (2008). Advanced 2D Game Development. Boston: Course Technology PTR.

http://riseagain.files.wordpress.com/2012/07/sfml-2-hierarchy-of-classes1.png

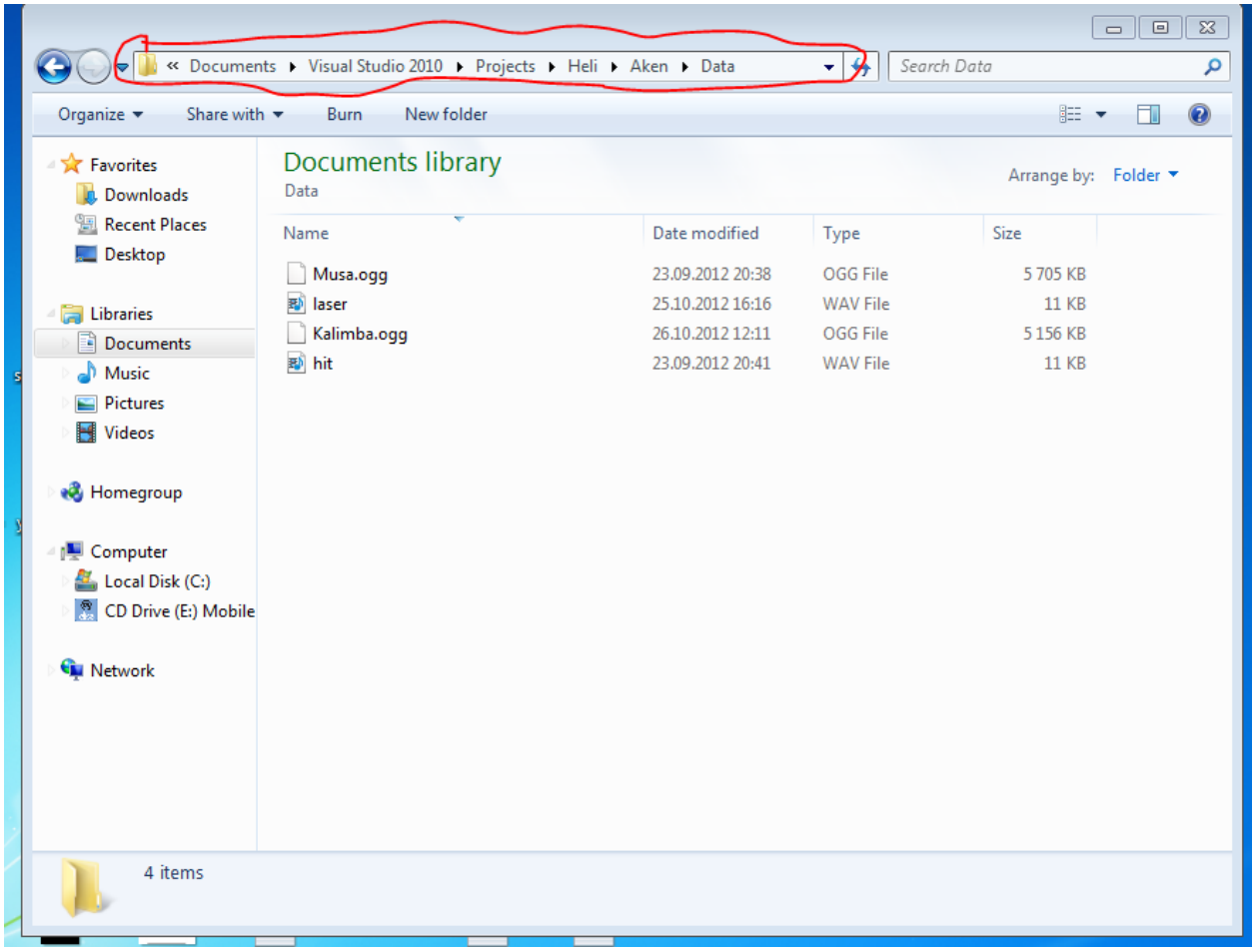
Mbrsolution. (2012). Create transparent background using gimp 2.8. Kasutamise kuupäev: 12.01.2013. a., allikas: http://www.youtube.com/watch?v=jMF8fIjW7Ls.

MediaCollege.com. Free Sound Effects. Kasutamise kuupäev: 25.01.2013. a., http://www.mediacollege.com/downloads/sound-effects/.

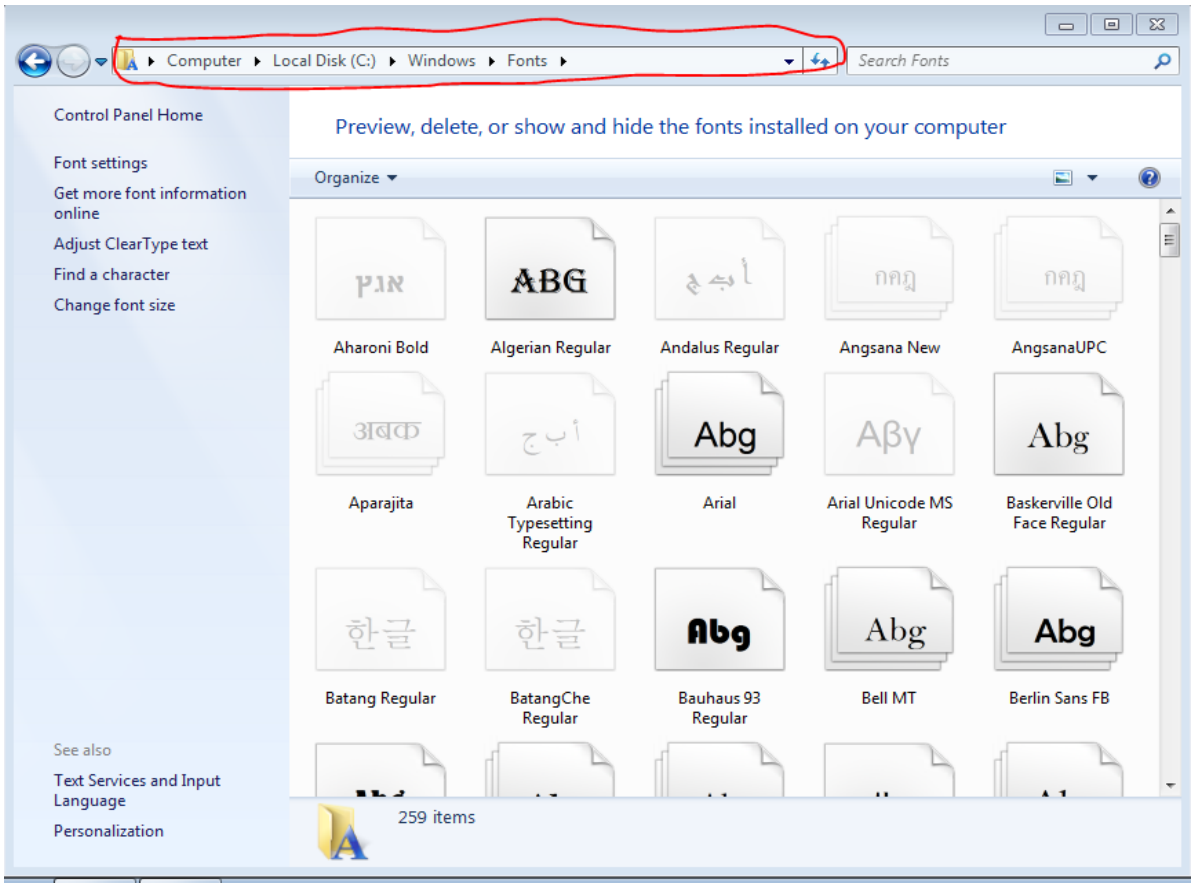
Sithjester's RMXP Resources. Kasutamise kuupäev: 23.01.2013. a., allikas: http://untamed.wild-refuge.net/rmxpresources.php?characters

Vallaste, H. (2013). E-teatmik. Kasutamise kuupäev: 16.11.2012. a., allikas: http://www.vallaste.ee/.

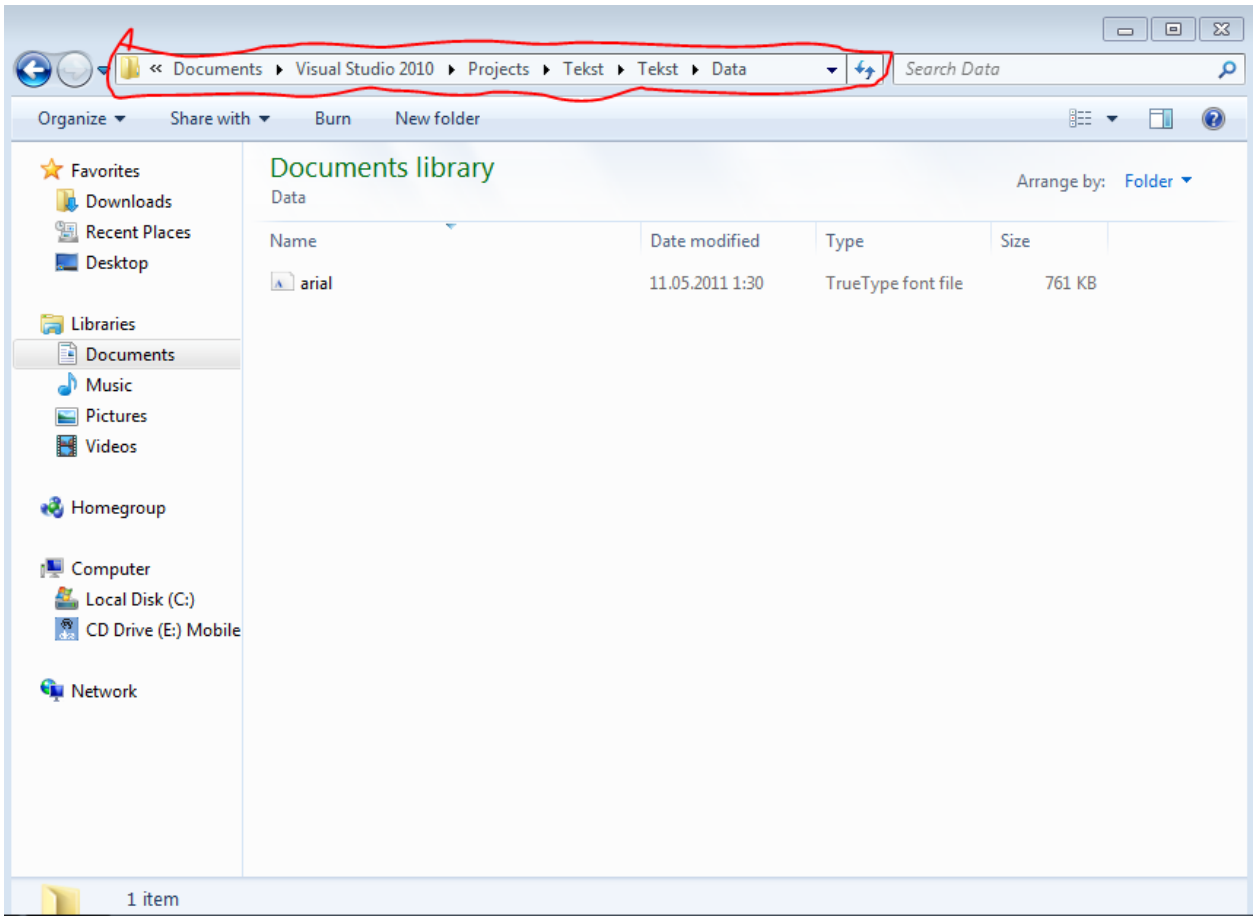
Lisad



Lisa 1. Kataloogipuu minu Data kausta.



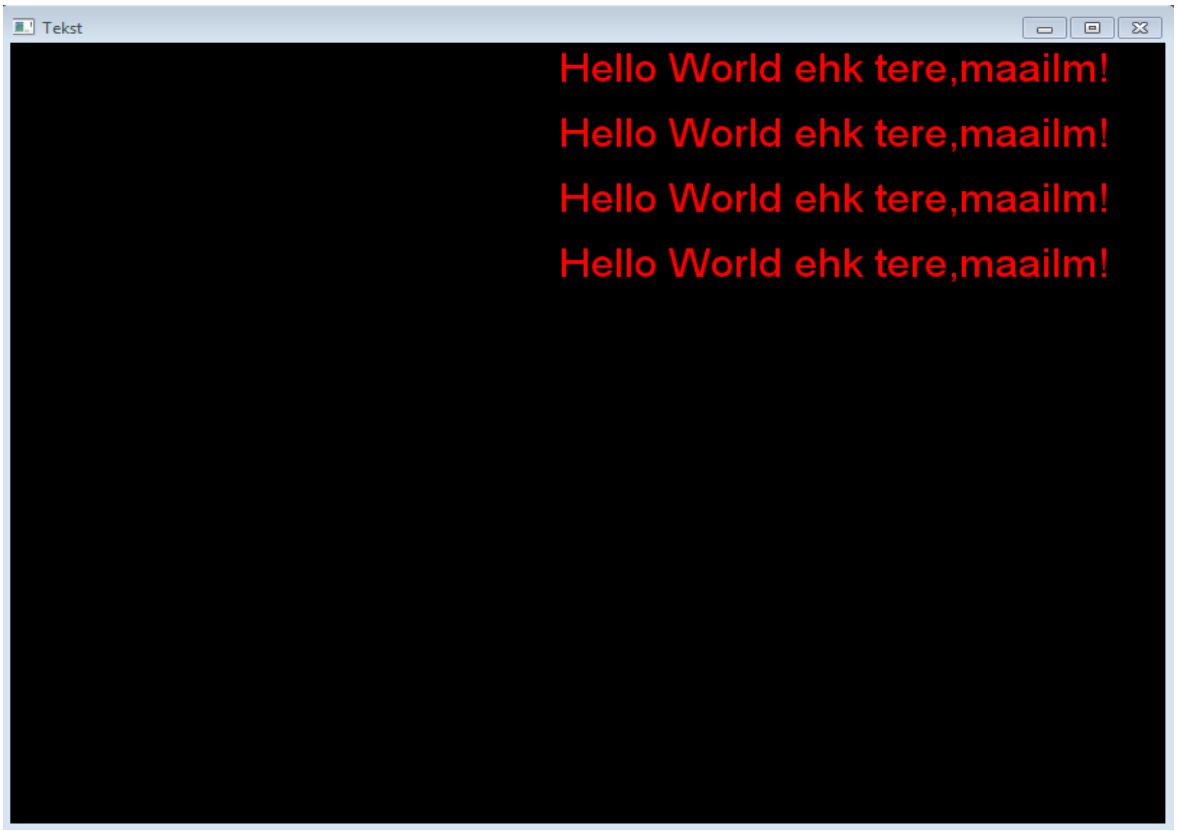
Lisa 2. Koht, kus valisin endale fondi.



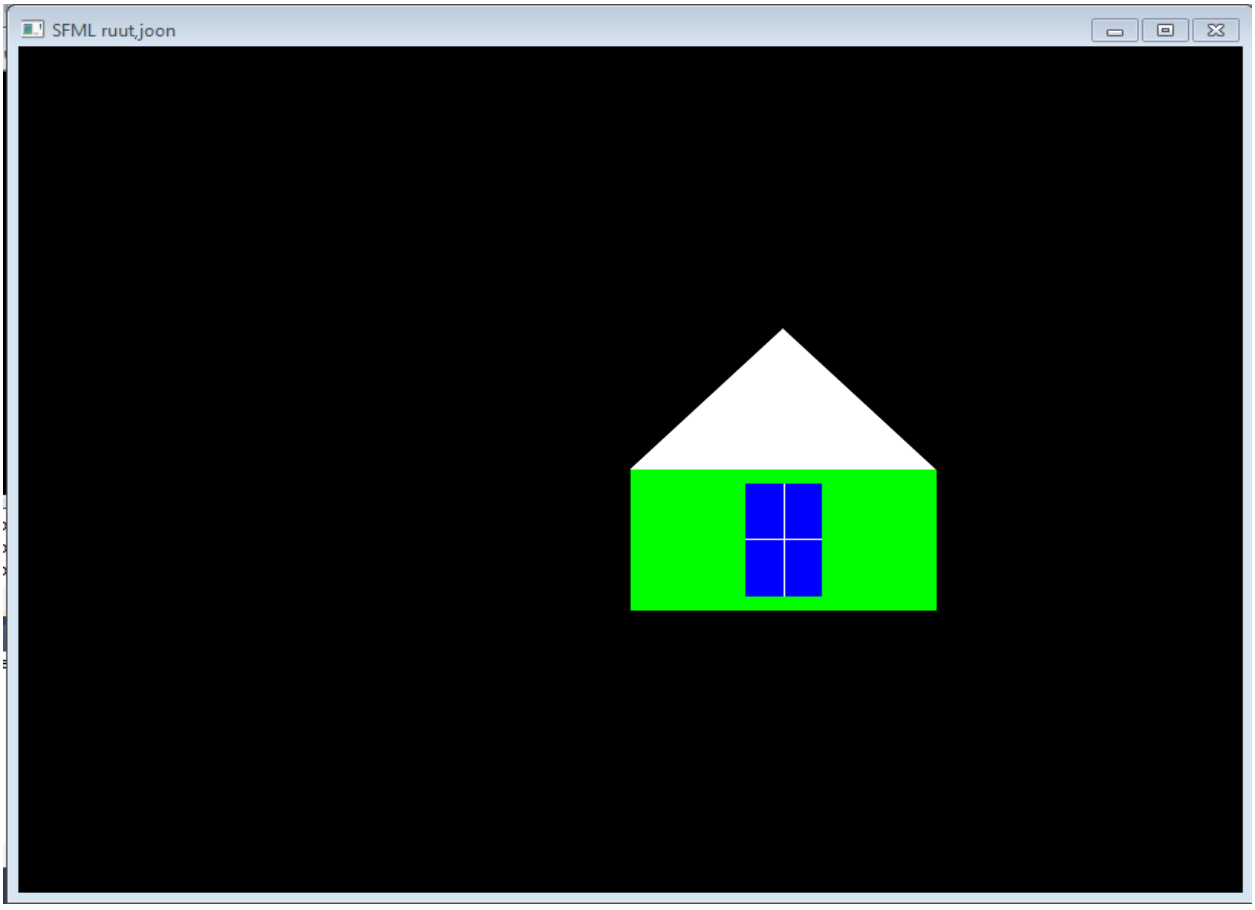
Lisa 3. Näidis minu Data kausta asuvast teest.

```
#include <SFML\Graphics.hpp>

int main(){

sf::RenderWindow aken(sf::VideoMode(800, 600), "Tekst");

     sf::Font minuFont;

     // Laeb kirjatüübi faili kettalt.
     if (!minuFont.loadFromFile("Data/arial.ttf"))
     {
          return false;
     }

     sf::Text tekst;
     tekst.setCharacterSize(30);//Teksti suurus
     tekst.setFont(minuFont);//Kasutatav kirjatüüp
     tekst.setColor(sf::Color::Red);//Teksti värvus
     tekst.setPosition(380, 10);  //Teksti asukoht

     //Teksti sisu
     tekst.setString("Hello World ehk tere,maailm!");
     //Teksti stiil
     tekst.setStyle(tekst.Underlined|tekst.Bold|tekst.Italic);

while (aken.isOpen())
    {

        sf::Event event;

        while (aken.pollEvent(event))
        {

            if (event.type == sf::Event::Closed)
                aken.close();
        }

          aken.clear();
          aken.draw(tekst);
          aken.display();
    }

     return 0;

}
```

Lisa 4. Teksti kasutamise näide koos kirjatüübi laadimisega.



Lisa 5 .Ülesande tulemus.



Lisa 6. Peatükis 6 asuva ülesande nr. 4 tulemus

```
#include <SFML/Graphics.hpp>


int main (){
	sf::RenderWindow window(sf::VideoMode(800,600,32),"SFML
ruut,joon");
	window.setFramerateLimit(60);
	//Muudab hiire kursori nähtamatuks ekraanil
	window.setMouseCursorVisible(false);

	sf::Event event;

	sf::RectangleShape ruut;
	ruut.setPosition(100,100);//Positsioon
	ruut.setSize(sf::Vector2f(50,50));//Ruudu suurus
	ruut.setFillColor(sf::Color::Green);//Värv

	while (window.isOpen())
		{
			//Sündmused
		while (window.pollEvent(event))
				{

				//Kuular akna sulgemiseks.
				if (event.type == sf::Event::Closed)
				{
					window.close() ;
				}


				}
		ruut.move(10,0);
		//Joonistame ja kuvame objektid aknale.
		window.clear();
		window.draw(ruut);
		window.display();
		}
		return 0;
}
```

Lisa 7. Objekti liigutamine kaadrisagedusel.

```
//Teegid
#include <SFML/Graphics.hpp>

int main (){

	sf::RenderWindow window(sf::VideoMode(800,600,32),"Nuppu
loomine");
	window.setFramerateLimit(60);
	sf::Event event;
	sf::RectangleShape nupp;
	nupp.setPosition(350,300);//Positsioon
	nupp.setSize(sf::Vector2f(100,50));//Objekti suurus
	nupp.setFillColor(sf::Color::Red);//Värv
	sf::Mouse hiir;

	while (window.isOpen())
		{
		while (window.pollEvent(event))
			{
			//Kuular akna sulgemiseks.
			if (event.type == sf::Event::Closed)
			{
				window.close() ;
			}
if(hiir.getPosition(window).x> ristkylik.getPosition().x &&
hiir.getPosition(window).x < ristkylik.getPosition().x+
ristkylik.getSize().x && hiir.getPosition(window).y>
ristkylik.getPosition().y && hiir.getPosition(window).y<
ristkylik.getPosition().y+ ristkylik.getSize().y &&
hiir.isButtonPressed(sf::Mouse::Left) ){

			ristkylik.setFillColor(sf::Color::Green);

			}else{

			ristkylik.setFillColor(sf::Color::Red);

			}
		}

		//Joonistame ja kuvame objektid aknale.
		window.clear();
		window.draw(nupp);
		window.display();
	}
	return 0;
}
```

Lisa 8. Nupu loomise näide.