Tallinna Ülikool

Digitehnoloogiate Instituut

Marmalade raamistiku õppematerjal C++ keeles

Seminaritöö

Autor: Toomas Häide

Juhendaja: Jaagup Kippar

Autor:....., ...,, 2015

Juhendaja:....., ...,, 2015

Instituudi direktor:....., ...,, 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

....

(kuupäev) (autor)

Sisukord

Sissejuhatus

Käesoleva seminaritöö eesmärgiks on luua lühike õppematerjal Marmalade raamistikust, mis annaks ülevaade sellest, et mida kujutab endast Marmalade raamistik ja kuidas seda kasutada. Milline näeb välja töökorraldus kasutades Marmalade raamistikku. Selgitada välja mõned tugevused ja murekohad mis võivad tekkida antud raamistikku kasutades.

Käesolev töö on jaotatud viieks peatükiks. Esimeses peatükis annab autor lühiülevaate sellest, mis on Marmalade raamistik ja mille jaoks ta kasulik on.

Teises peatükis annab autor ülevaate sellest, kuidas seadistada Marmalade raamistikku ja mida täpsemalt on vaja et tööd alustada.

Kolmandas peatükis annab autor ülevaate sellest kuidas luua esimene rakendus ja proovida kas antud raamistiku paigaldamine on olnud edukas.

Neljandas peatükis tutvustab autor Marmalade raamistiku töökorraldust. Kuidas lisatakse projekti faile, kuidas püütakse kasutaja sisendit ja kuidas luua keerukamat rakendust.

Viiendas peatükis näitab autor kuidas töö vältel loodud rakendust jooksutada android tüüpi telefonil.

1. Mis on Marmalade raamistik

Marmalade SDK on võimekas mitmeplatvormiline tööriist rakenduste, eelkõige mängude, loomiseks. Marmalade võimaldab arendajatel kirjutatud koodi jooksutada erinevatel operatsioonisüsteemidel ilma platvormispetsiifiliste nõueteta. Marmalade ei ole täismahus mängumootor vaid pigem nagu nimigi seda ütleb, raamistik mille põhjal näiteks enda mängumootor ehitada.

Marmalade tuumaks on C++ tarvkaraarenduskomplekt mis annab arendajatele suurima võimekuse ja paindlikuse. Marmalade Quick, Web Marmalade ja Marmalade Juice baseeruvad C++ Marmalade raamistikul, aga pakuvad alternatiivseid lahendusi arendajatele kes eelistavad programmeerimiskeeltena kasutada näiteks Lua-d, HTML-i või Objective-C-d.

Et tagada mitmikplatvormi tugi, jagab Marmalade loodud rakendused kaheks osaks. Kogu kood, mis programmeerija kirjutab, kompileeritakse ja lingitakse programmiteegiks mille omakorda jooksutab laadurprogramm. Laadurprogramm on platvormipõhine abstraktsioonikiht mis suhtleb operatsioonisüsteemiga ja jooksutab programmeerija kirjutatud koodi operatsioonisüsteemile arusaadaval moel.

1.1. Marmalade võimaluste tutvustamine

Marmelade pole veel laialdast kasutuspinda leidnud, arvan, et osaliselt on see tingitud antud raamistiku keerukusest. Üheks eelduseks, et marmaladega töötada, on programeerimiskeele C++ tundmine, kuna väga paljud loomulikuna tunduvad asjad on tarvis ise implementeerida, näiteks helide taasesitusmootor. Sellest keerukusest tingituna kaasnevad ka suuremad arendamis- ja ajakulud. Marmalade pole ka vaba tarkvara vaid hoopis tasuline ning viie erineva litsentsitüübiga, millest üks neist on täiesti tasuta. Leidub ka mängustuudioid kes pole kõigest sellest end hirmutada lasknud, näiteks PopCap Games kes meisterdas Marmaladega valmis menuka Plants Vs Zombies (Joonis 1)

Joonis 1. Taimed vs Elavad surnud (PopCap Games)

1.2. Marmalade poolt toetatud arenduskeskkonnad ja platvormid

Enne, kui marmaladega tööd alustada, tasub ülevaadata missuguseid arenduskeskkondi marmalade üldse toetab ja kas arendajana on sul üldse võimalik neid kasutada. Eriti tähelepanelik tasub olla

Windows-i kasutajatel, kuna arenduskeskkonnana sobib ka Visual Studio Express for Windows desktop variant. Aga kindlasti peavad klappima aastanumbrid. Näiteks 2015 versiooniga antud raamistik ei pruugi töötada aga 2012 aasta variandiga ta töötab.

Marmalade poolt toetatud operatsioonisüsteemid arendamiseks on:

- Microsoft windows 7 või hilisem
- Apple Mac OS X 10.8 või hilisem

Marmalade poolt toetatud arenduskeskkonnad on:

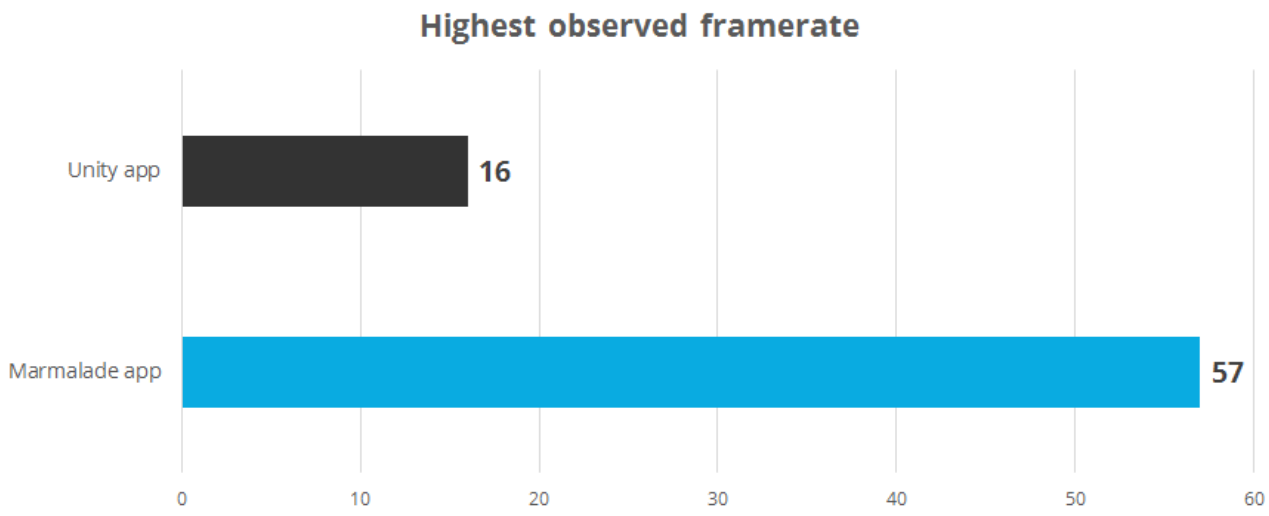
- Microsoft Visual Studio 2008/2010/2012
- Xcode 6

Lisaks sellele tasub veel ära mainida, et Marmalade C++ praegusel hetkel toetab ainult C++03 standardiga kooskõlas olevaid päisefaile. See tähendab seda, et mitte ühtegi C++11 teekidest pole võimalik kasutada, sealhulgas näiteks regex, algorith, chrono, map, jne. Marmalade see-eest toetab mõningaid C++11 keele võimalusi, mis sõltub jällegi kompilaatorist mida mingi sihtplatvorm kasutab. Teisisõnu, kui on soov kasutada C++11 keele võimalusi, siis on vajalik uurida kas sihtplatvormi kompilaator üldse seda üldse toetab.

1.3. Miks Marmalade

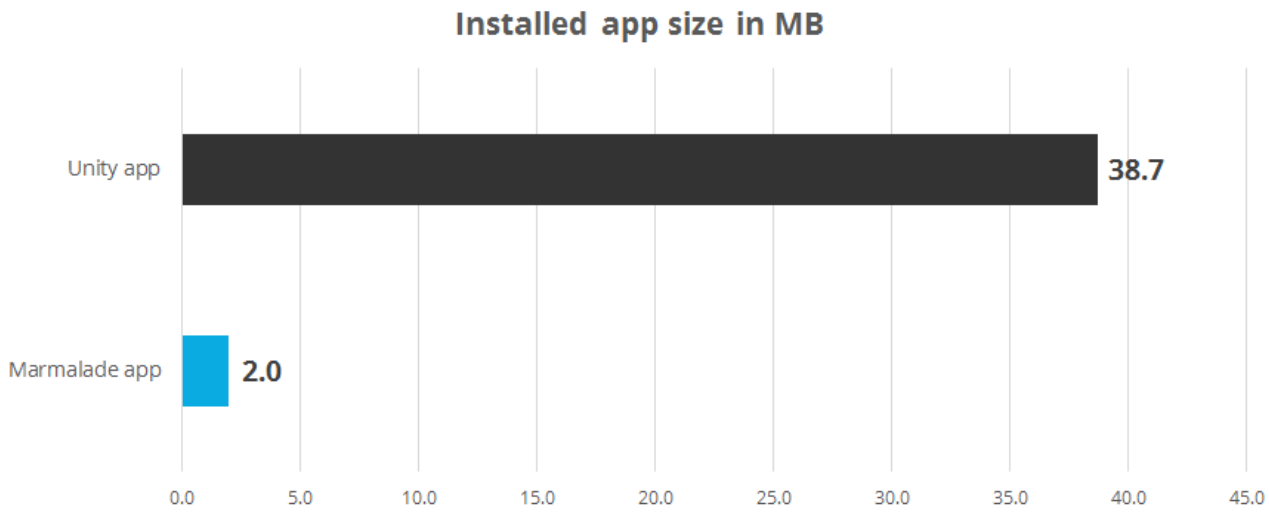
Marmalade põhikonkurendiks praegusel hetkel on Unity. Selle tõttu korraldas Marmalade meeskond Unity ja Marmaladega loodud rakenduste võrdluse. Võrreldi paketi suurust, installeritud rakenduse suurust, rakenduse poolt kuvatavat kaadrite arvu sekundis ja projekti suurust.

Uuringu kohaselt, saavutas uuringu käigus loodud Marmalade rakendus kaadrisageduse 57 kaadrit sekundis versus 16 kaadrit sekundis samaväärse rakendusega mis loodud kasutades Unity 3D mootorit. Uuuringu kohaselt oli Marmalade rakendus ligi 300% kiirem Unity 3D rakendusest (Joonis 2).



Joonis 2. Marmalade FPS vs Unity 3d FPS (Rudman, 2015)

Uuringu kohaselt oli ka Marmaladega loodud rakenduse installeeritud lõppsuurus 85 % väiksem Unity 3D-ga loodud rakenduse lõppsuurusest (Joonis 3).



Joonis 3. Installeeritud rakenduse suurus (Rudman, 2015)

Uuringu tulemuste kohaselt on Marmalade neist kahest parem valik kui eesmärgiks on võimalikult suure jõudlusega ja väiksema mälumahuga rakenduse loomine.

1.4. Platvormid millele on võimalik marmaladega rakendusi luua

Marmalade poolt toetatud platvormid on nutitelefonid(Tabel 1), nutitelerid(Tabel 2), laua- ja sülearvutid(Tabel 3).

Tabel 1. Nutitelefonid (Marmalade Technologies Ltd)

	iOS	Android	Windows phone	BlackBerry	Tizen
Operatsioonisüsteemi versioon	5.1 ja uuem	2.1 ja uuem	8 ja uuem	Playbook OS ja BB10	2.2 ja uuem
Nutitelefonid	iPhone 3GS ja uuemad, iPod Touch 4 generatsioon ja uuemad	Kõik	Kõik	Kõik	Kõik
Tahvelarvutid	Kõik	Kõik	Kõik	Kõik	Kõik
Arhitektuur	ARM, AArch64	ARM, x86, MIPS	ARM	ARM, x86	ARM

Tabel 2. Nutitelerid (Marmalade Technologies Ltd)

	LG Smart TV	Roku
Operatsioonisüsteemi versioon	Netcast 3.0, 4.0	Roku 2 & 3 Roku Streaming Stick
Arhitektuur	ARM	ARM

Tabel 3. Laua- ja sülearvutid (Marmalade Technologies Ltd)

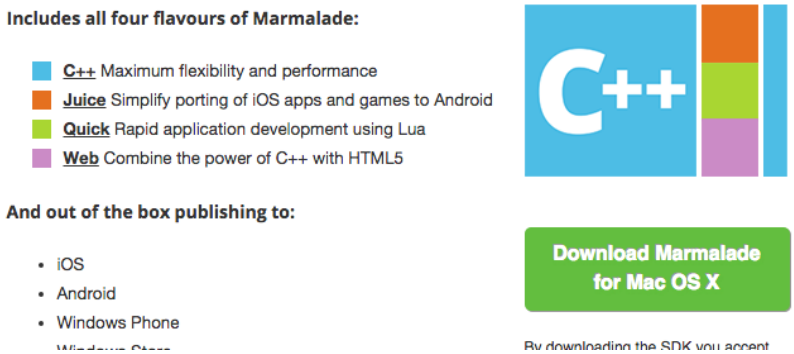
	Windows	Mac
Operatsioonisüsteemi versioon	Windows XP ja uuemad	OS X 10.8 ja uuemad
Arhitektuur	X86	X86

2. Marmalade seadistamine

Enne kui saab esmast rakendust programmeerima hakata kontrolli kas sul on olemas toetatud arenduskeskkond, kas siis Xcode 6 OSX platvormil või Microsoft Visual Studio. Järgmine samm oleks Marmalade tarkvaraarenduskomplekti ja litsentsi hankimine. Tarkvaraarenduskomplekt on saadaval aadressilt https://www.madewithmarmalade.com vajutades nupule *download free SDK*.(Joonis 4)

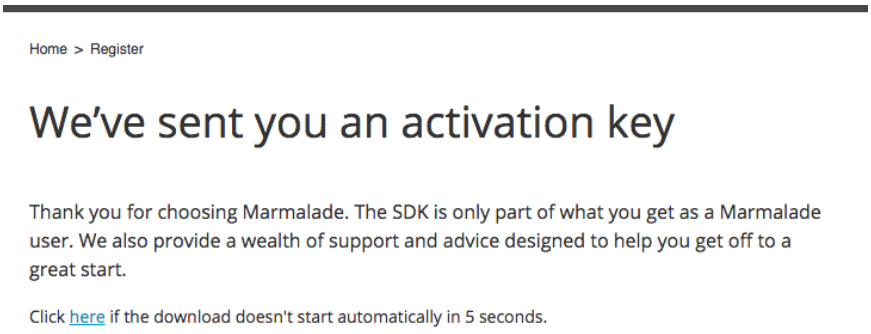
Joonis 4. Marmalade allalaadimine

Järgnevalt lehelt valida *download Marmelade for* kas siis Windows või Mac OS X.(Joonis 5)



Joonis 5. Marmalade versiooni valik

Täida ära registreerimisvorm ja sulle saadetakse e-kirjana Marmalade aktiveerimislink. Ava see link ja sulle saadetakse Marmalade aktiveerimisvõti ja Marmalade tarkvaraarenduskomplekt peaks samuti allalaadimist alustama.(Joonis 6)

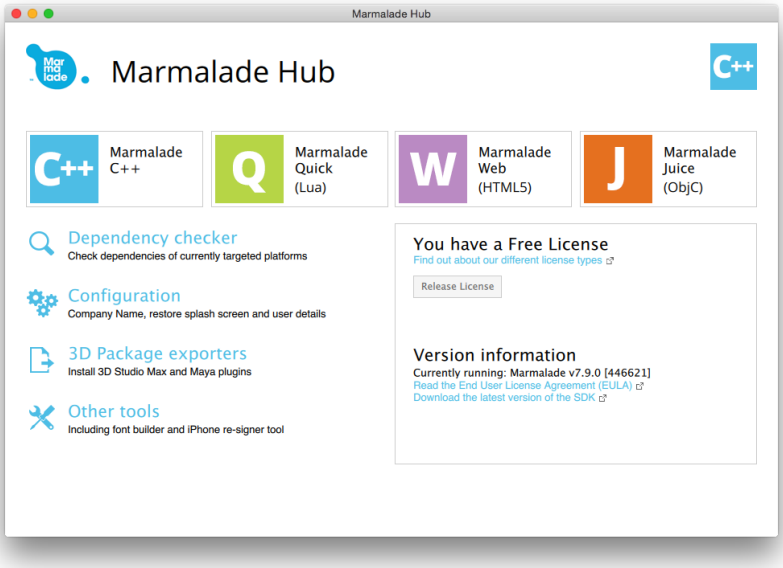


Joonis 6. Aktiveerimisvõtme muretsemine

Pärast allalaadimise lõpetamist installi Marmalade raamistik, ava Marmalade Hub ja aktiveeri see e-kirjale saadetud võtmega.

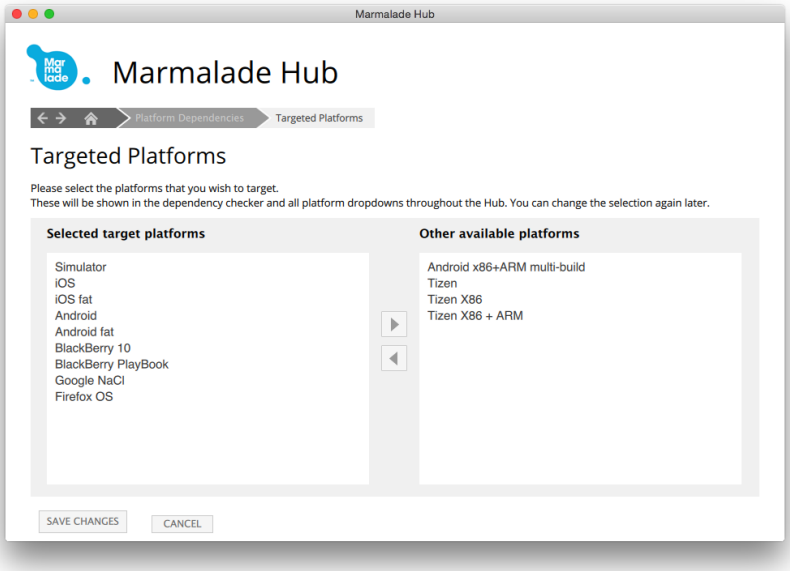
2.1. Marmalade Hub

Marmalade hub(Joonis 7) on peamine kasutajasõbralik graafiline kasutajaliides Windows ja OSX platvormile mille eesmärgiks on hõlbustada Marmalade raamistiku erinevate projektide haldamist ja loomist.



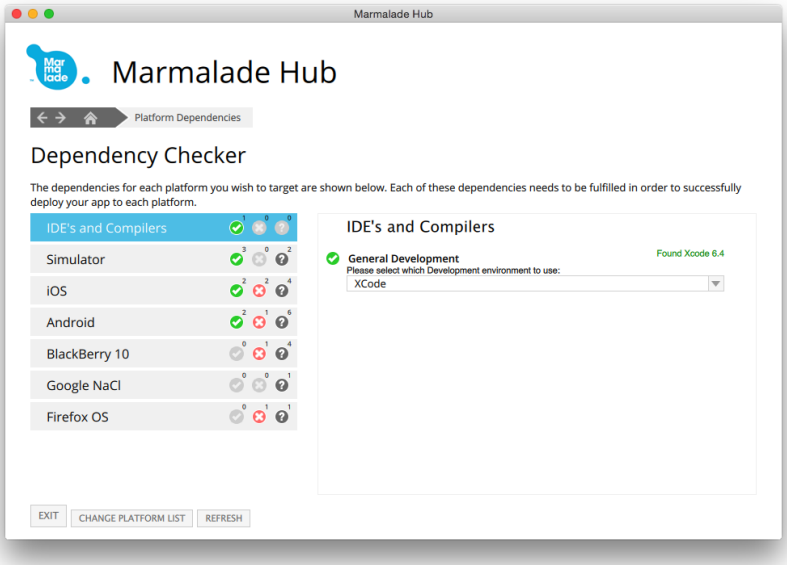
Joonis 7. Marmalade Hub

Nüüd tuleb veel üle kontrollida kas Marmalade on õigesti tuvastanud ka arenduskeskkonna. Selleks valime menüüst *dependency checker*. Veendume, et Selected target platformsi all oleks olemas valik simulator ja vajutame save changes (Joonis 8).



Joonis 8. Marmalade arendatavate platvormide nimekiri

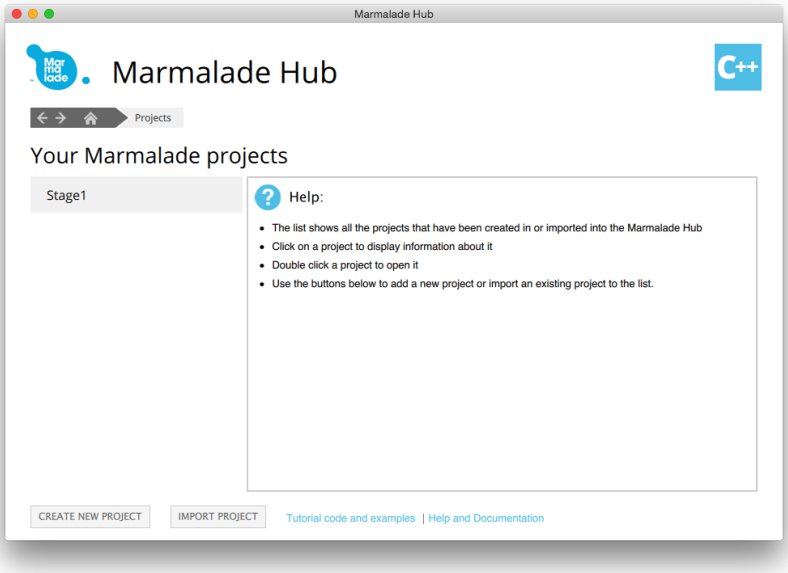
Vasakpool paiknevast menüüst märgistame ära IDE's and Compilers ja veendume et oleks valitud õige arenduskeskkond. Kuna see õppematerjal on kirjutatud kasutades Maci, on ka General Development IDE XCode (Joonis 9). Windowsil peaks see olema siis kas Microsoft Visual Studio 2012 Express või varasem ja vajutame Exit.



Joonis 9. Marmalade sõltuvuste haldur.

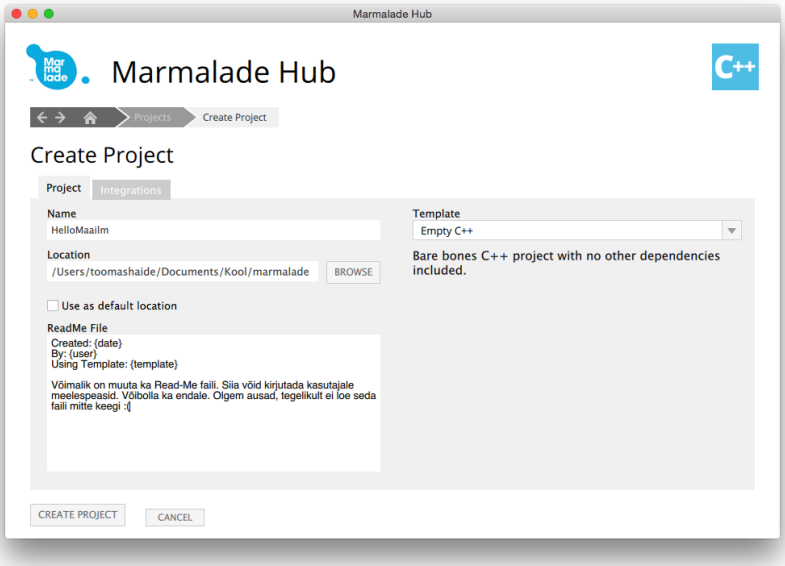
3. Esmase rakenduse loomine

Marmalade projekti on põhimõtteliselt võimalik luua kahel viisil. Üks viisidest on seda teha käsurea kaudu ja teine võimalus on kasutada Marmalade Hub-i. Antud juhul kasutame me Hub-i. Projektitüübiks valime Marmalade C++ vajutades ikoonil. Avaneb projektide nimistu. Vajutame nupule "CREATE NEW PROJECT" (Joonis 10).



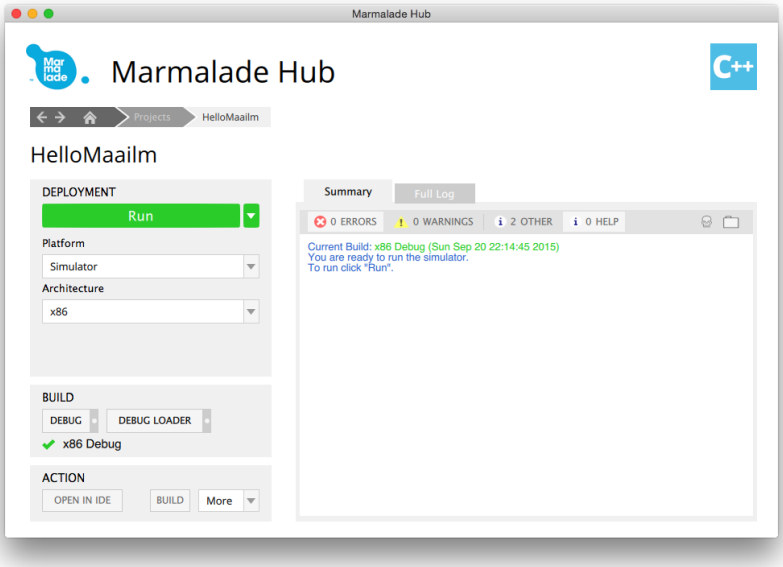
Joonis 10. Projektide nimekiri

Järgmisena avaneb Projekti loomisvaade (Joonis 11). Anna enda projektile sobiv pealkiri. Võid lisada ka ReadMe faili, kui selleks tõesti soov ja vajadus on. Templateks sobib juba valitud Empty C++.



Joonis 11. Projekti loomisvaade

Vajutades nupule "CREATE PROJECT" luuakse uus projekt ja avatakse projekti haldusvaade(Joonis 12).



Joonis 12. Projekti haldusvaade

Nüüd on esialgne projekt valmis ja võid seda vabalt käivitada, testimise mõttes, vajutades nuppu *Run*. Kuigi see rakendus veel midagi peale antud logo kuvamise teha ei oska, siis ilus vaadata on teda ikka(Joonis 13). Samuti on tore teada kas Marmalade tarkvaraarenduskomplekti paigaldamine on olnud edukas või mitte.

Joonis 13. Marmalade logo simulaatoris

Et muuta antud rakendus pisut kasulikumaks, siis tasub avada antud projekt arenduskeskkonnas ja muuta selle lähtekoodi. Selleks tuleb vajutada projekti haldusvaates nupule *OPEN IN IDE*. Pärast mõningaid muudatusi võiks `main()` funktsiooni sisaldav fail näha välja selline (Koodinäide 1).

```
#include "s3e.h"
#include "IwDebug.h"
int main()
{
    while (!s3eDeviceCheckQuitRequest())
    {
        s3eSurfaceClear(0, 0, 255);
        s3eDebugPrint(120, 150, "`xaedaedTere, maailm!", 0);
        s3eSurfaceShow();
        s3eDeviceYield(0);
    }
    return 0;
```

```
}
```

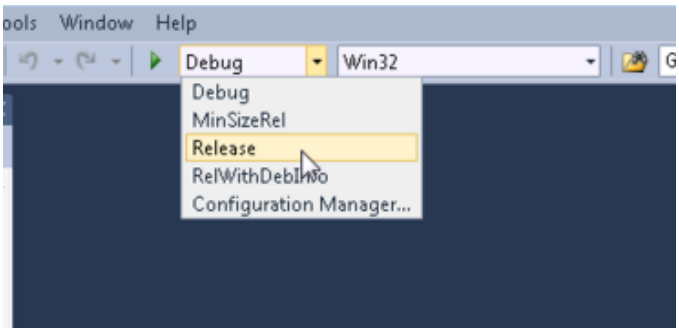
Koodinäide 1. HelloMaailm.cpp

Veidi lähemalt mida iga rida koodi antud rakenduses teeb:

- Käsklus `#include "s3e.h"` lisab `main()` funktsiooni sisaldavasse faili S3E rakenduste programmeerimisliidese. S3E täispikk nimi on *Segundo Embedded Execution Environment* ja kujutab endast programmeerimiskeeles C kirjutatud rakenduste programmeerimisliidest.

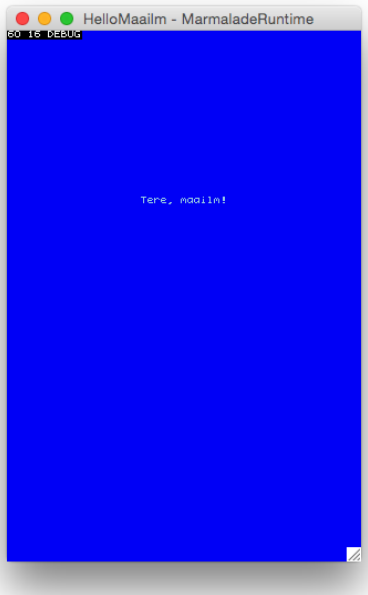
- Käsklus `#include "IwDebug.h"` sarnaselt eelnevale, lisab `main()` funktsiooni sisaldavasse faili vigade silumisteegi.
- `while (!s3eDeviceCheckQuitRequest())` kontrollib kas programmi jooksutav seade on saatnud programmile töö lõpetamiskäsu.
- `s3eSurfaceClear(0, 0, 255);` Puhastab ekraani kirjutades kõik pikslid üle sinise värviga.
- `s3eDebugPrint(120, 150, "Tere, maailm!", 0);` Trükib stringi, mis on kolmas parameeter, koordinaatide alguspunktist 120 pikslit paremale ja 150 pikslit alla. Koordinaatide alguspunkt on 0,0. Neljas parameeter on *wrap* mis võtab sisendina tõeväärtusfunktsiooni parameetri 0 või 1. Kas toimub sõnade ülekanne järgmisele reale kui tekst peaks minema üle ekraani parema serva.
- `s3eSurfaceShow();` Kuvab vastloodud kuvapinna ekraanile.
- `s3eDeviceYield(0);` Tagastab operatsioonisüsteemile kontrolli, et täita esmavajalike funktsioone, näiteks sisendi kuulamine, sündmuste saatmine programmile. Kui väärtus on 0, tähendab see, et operatsioonisüsteemile antakse nii vähe aega kui võimalik et ta saaks täita enda ülesandeid.

Nüüd peaks valima arenduskeskkonas *build configuration* menüüst konfiguratsiooni nimega *Debug x86*. Ära lase end häirida sellest, et pilt ei kajasta meie projekti konfiguratsioone (Joonis 14).



Joonis 14. Build configuration (The University of Warwick)

Pärast seda sammu võib antud koodijuppi jooksutada ka kasutades arenduskeskonna Run käsklust. Kuna mina olen kasutanud antud juhendi loomisel arenduskeskkonda Xcode, siis võib pilt olla mõningal viisil erinev aga põhimõte on sama (Joonis 15).

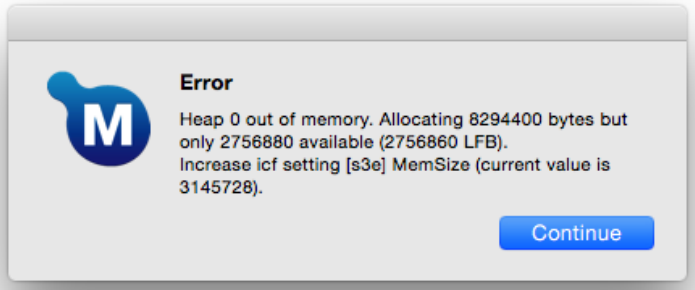


Joonis 15. Tere, Maailm

4. Edasijõudnud rakenduse loomine

Kui esimesed katsetused on olnud edukad, võib edasi liikuda mahukamate ülesannete kallale.

Ja kõige esimene neist ongi seotud just mahutavusega. Nimelt kui Marmalade rakendus seadmes käivitatakse, luuakse mälukuhi, millele eraldatakse mingi teatud hulk mälu mida kahjuks või õnneks dünaamiliselt enam juurde ei eraldata. Vaikeväärtusena on seda mälu üsna vähe - 4Mb nutiseadmetel ja 10 Mb Windows/Linux seadmetel. Kui mälukuhjale antud mälu on otsakorral, annab Marmalade veateate (Joonis 16).



Joonis 16. Heap out of memory error

Et Marmaladele eraldatud mälu hulka suurendada, tuleb projektist üles otsida fail nimega app.icf. Sellesse faili tuleb pärast kommentaare lisada 2 rida koodi (Koodinäide 2. Memsize 50mb).

```
[S3E]
MemSize = 50000000
```

Koodinäide 2. Memsize 50mb

Sellega anname märku, et mälukuhjale 0 tuleb eraldada 50 Mb mälu. Maksimaalselt on võimalik luua kuni 8 erinevat mälukuhja.

4.1. Simulaatori akna suuruse muutmine

Simulaatori akna suuruse võiks seada natukene suuremaks, kuna vaikeväärtusena on selle suurus 320x480 pikslit, mis tänapäeva standardite järgi on liiga pisike. Windowsil käib see üsna hõlpsasti, tuleb lihtsalt käivitada simulaator, valida ülevalt menüüst configuration ja valida avanenud nimekirjast surface, ja muuta parameetri Predefined resolution väärtus iPhone5/5C/5S Portrait. Kasutades aga OS X arvutit, tuleb projektist üles otsida fail nimega development.icf ja muuta parameetreid SurfaceHeight ja SurfaceWidth. Käsitsi peab muutma just nende parameetrite `value` välja. Pärast seda tuleb arenduskeskkonna XCode puhul projekt uuesti käivitada.

4.2. MKB Süsteem

Iga Marmalade projekti südameks on selle projekti MKB fail. MKB fail on tekstifail mis kirjeldab kõiki projekti sätteid. MKB fail sealhulgas kirjeldab ka missugused koodifailid kuuluvad projekti ja samas ka missugune on selle projekti struktuur. Niiet kui on soov faile projekti lisada, siis tuleb need ka MKB failis ära kirjeldada. Avame meie praeguse projekti MKB faili, et teha mõningad muudatused. Meie praegune projektifail peaks välja nägema selline (Koodinäide 3), kui ei näe siis pole hullu.

```
#!/usr/bin/env mkb
# Kommentaaride lisamiseks, alusta rida # sümboliga
files
{
    HelloMaailm.cpp
}

subprojects
{
    iwutil
}
```

Koodinäide 3. HelloMaailm.mkb

- Grupp `files{}` ütleb Marmalade raamistikule missugustest failidest meie projekt koosneb.
- Grupp `subprojects{}`, ütleb Marmaladele missugustele alamprojektidele viidata. Alamprojekt on nagu platvormi laiendusteek mida on võimalik rakenduses kasutada kui

selleks soov on, aga vaikimisi neid projekti ei lisata, et säästa mälu ja ruumi. Oma laiendusi on võimalik ka ise luua.

Defineerime mõned lisagrupid ja lisame ka projekti mõned failid, ning nüüd võiks meie projektifail välja näha selline (Koodinäide 4).

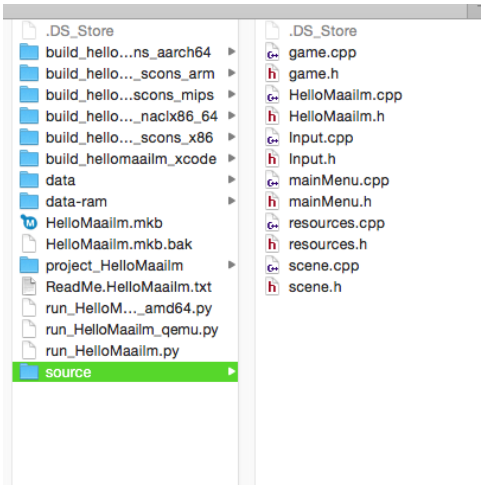
```
#!/usr/bin/env mkb
files
{
    [source]
    (source)
    HelloMaailm.cpp
    Input.cpp
    scene.cpp
    resources.cpp
    mainMenu.cpp
    game.cpp

    HelloMaailm.h
    Input.h
    scene.h
    resources.h
    mainMenu.h
    game.h
}

subprojects
{

    iw2d
    iw2dscenegraph
    iw2dscenegraphcore
    iwtween

}

assets
{

    (data)
    textures

    (data-ram/data-gles1, data)
    .

}
```

Koodinäide 4. HelloMaailm.mkb pärast muudatusi

Natuke põhjalikumat juttu antud faili süntaksist:

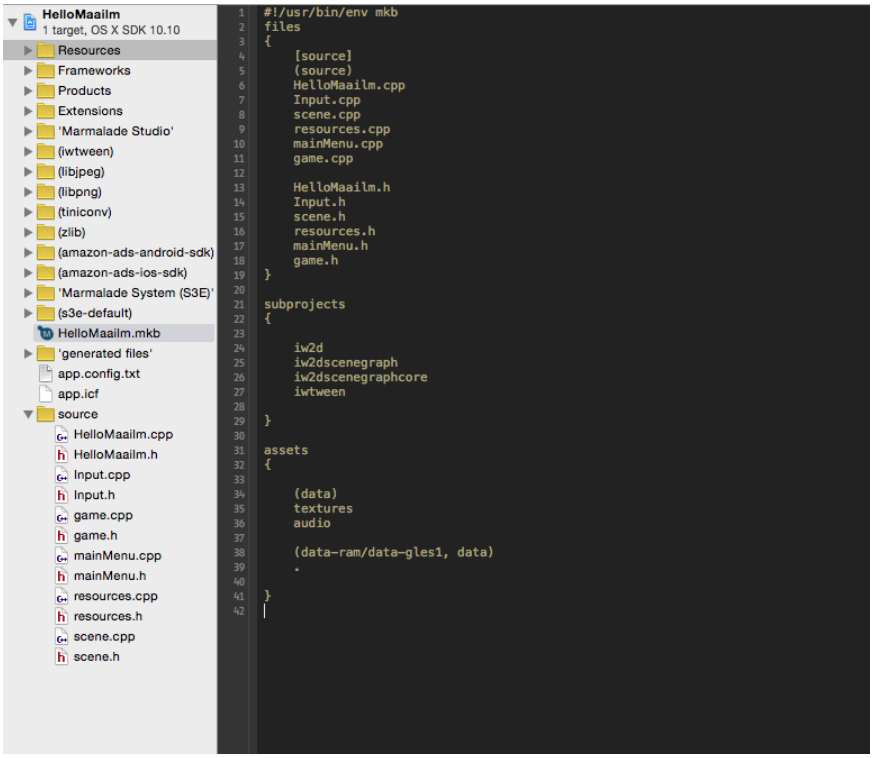
- `[source]` tähistab seda, et failidele mis projekti lisatakse, luuakse ka arenduskeskkonnas oma virtuaalne kaust nimega source.
- `(source)` tähtistab seda, et antud koodifailid paiknevad projektifaili sisaldavas kataloogis omakorda kataloogis nimega source.
- `iw2d` on lihtne, kahemõõtmelist joonistuspinda sisaldav graafikateek.
- `iw2dscenegraph` ja `iw2dscenegraphcore` on mängustseene sisaldav graaf. Näiteks antud näites võtame kasutusele 2 stseeni, esimene neist on mängu menüü ja teine mängustseen. Neid stseene hoitakse graafipuus kuna sellisel viisil on võimalik hierarhiliselt stseene struktureerida.
- `iwtween` on teek mis võimaldab sprite objekte liigutada mingisuguse animatsiooniga.
- `assets{}` grupp ütleb, missuguseid andmefaile projekt endaga kaasa peaks võtma, sinna alla kuuluvad näiteks tekstuurid, helid, kirjad.

Nüüd kui on MKB failis projekti struktuur ja failid mida lisada defineeritud, tuleb lihtsalt need failid meil luua. Selleks, peame kahjuks arenduskeskkonna sulgema ja need failid käsitsi looma. Selleks navigeerime projektifaili sisaldavasse kausta ja loome sinna uue kausta nimega source. Tõstame sinna kausta ka `main()` funktsiooni sisaldava faili ja loome uued failid, mis me MKB failis kirjeldasime (Joonis 17).



Joonis 17. Projekti struktuur

Avame uuesti nüüd enda projekti, kas siis Marmalade Hub-ist või topeltklõpsuga HelloMaailm.mkb failil ja näeme, et kui kõik on õigesti tehtud, peaks meie projektistruktuur nägema välja selline (Joonis 18).

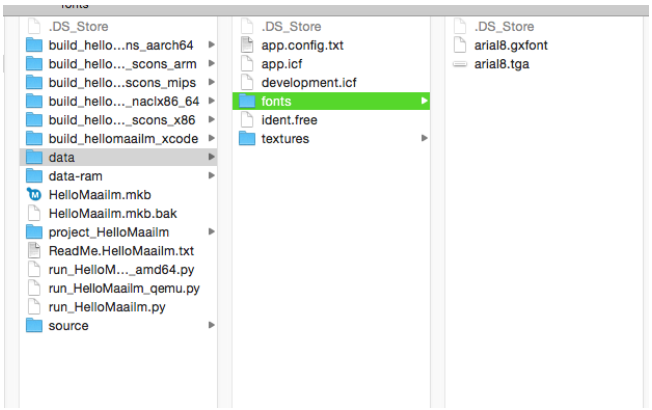


Joonis 18. XCode projektistruktuur

Pane tähele, et ka `main()` funktsiooni sisaldavale implementatsioonifailile loodi samanimeline päisefail, ehk siis antud juhul HelloMaailm.h.

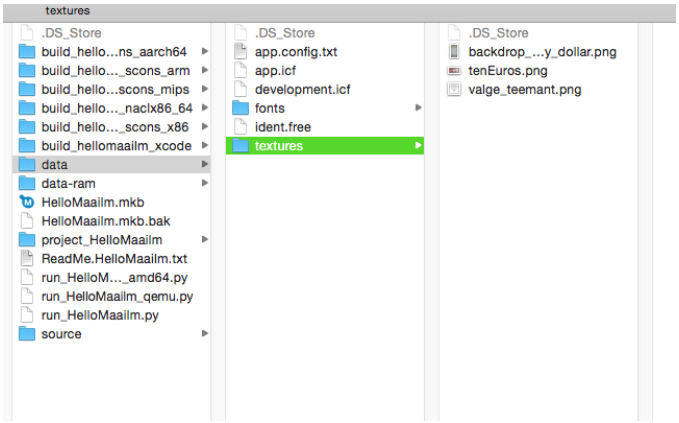
4.3. Mängu ressursside hankimine

Kõik antud projektiga seotud ressursid on saadaval aadressilt www.toomas.it või antud tööga kaasasolevalt cd plaadilt. Meil on vaja luua projekti data kausta alla omakorda kaust nimega fonts, sinna paigutame 2 faili nimega arial8.gxfont ja arial8.tga (Joonis 19).



Joonis 19. Kirjastiile sisaldav kaust

ja lisaks on veel meil tarvis hankida kolm tekstuuri: tenEuros.png, backdrop_flappy_dollar.png ja valge_teemant.png ja luua uus kaust textures ja paigutada need sinna (Joonis 20).



Joonis 20. Tekstuure sisaldav kaust

4.4. Interaktiivsuse lisamine

Et muuta antud interaktiivsemaks ja mängulisemaks, tuleks sellele lisada mingisugune sisendi tuvastus. Marmalade SDK kasutab sisendi registreerimiseks tagasihelistusfunktsioone(edaspidi *callback* funktsioon). Et kuulata puutetundlikut sisendit, peame me registreerima mitu *callback* funktsiooni, mis kutsutakse välja, kui mingisugune sündmus aset leiab. Marmalade toetab mitmeid puutetundlikke sündmusi, sealhulgas:

- S3E_POINTER_BUTTON_EVENT - Sündmus mis kutsutakse välja seadmetel, mis toetavad ainult ühte puudutust korraga, siis kui kasutaja puudutab, või lõpetab ekraani puudutamise.
- S3E_POINTER_MOTION_EVENT - Sündmus mis kutsutakse välja seadmetel, mis toetavad ainult ühte puudutust korraga, siis kui kasutaja hoiab all ja liigutab oma sõrme ekraanil.
- S3E_POINTER_TOUCH_EVENT - Sündmus mis kutsutakse välja seadmetel, mis toetavad mitut puudutust korraga, siis kui kasutaja puudutab, või lõpetab ekraani puudutamise.
- S3E_POINTER_TOUCH_MOTION_EVENT - Sündmus mis kutsutakse välja seadmetel, mis toetavad mitut puudutust korraga, siis kui kasutaja hoiab all ja liigutab oma sõrme ekraanil.

Lisaks tasub meeles pidada, et:

- Mitmikpuudutust toetavaid sündmusi saab välja kutsuda mitu korda kaadri jooksul, kuna mitu puudutust on võimalik tuvastada ühe kaadri jooksul.
- Kui on soov lisada tugi mitut- ja üksikut puudutust tuvastavale platvormile, peab kõigile neljale sündmusele lisama *callback* funktsioonid.
- Et püüda sisendit on kasulik hoida kõik sisendi püüdmisega seotud funktsioonid eraldi klassis. Selle tarbeks lõimegi Input klassi.

4.5. Input klass

Input klass antud näites on kergekaaluline ümbris, suuremale klasside kogumikule. Input klass registreerib kasutaja puudutusi ja tema käest on võimalik küsida puudutuse täpseid koordinaate ja eelmisi registreeritud puudutusi ja seisundeid. Avame Input.h päisefaili ja muudame seda (Koodinäide 5).

```
#ifndef __INPUT_H_INCLUDED__
#define __INPUT_H_INCLUDED__

#include "s3ePointer.h"

#define MAX_TOUCHES         10
```

```
class Input
{

//Loome muutujad, et pidada meeles puudutusi
public:
    int     m_X, m_Y;
    bool    m_Touched;
    bool    m_PrevTouched;

public:
    Input();

    void Update();
    void Reset();

    //Defineerime erinevad callback funktsiooni mustrid, mida marmalade vajab
sisendi püüdmiseks.
    static void TouchButtonCB(s3ePointerEvent* event);
    static void TouchMotionCB(s3ePointerMotionEvent* event);
    static void MultiTouchButtonCB(s3ePointerTouchEvent* event);
    static void MultiTouchMotionCB(s3ePointerTouchMotionEvent* event);

};

extern Input* g_pInput;

#endif // _INPUT_H
```

Koodinäide 5. Input.h (Hopwood, Stage 2 - Input and audio)

Antud päisefail defineerib *callback* funktsioonid mida on vaja sisendi püüdmiseks ja defineerib globaalse muutuja `g_pInput`, kuna sisendi püüdjat peab rakenduses olema idee poolest ainult üks. Järgmisena avame Input.cpp implementatsioonifaili ja muudame seda (Koodinäide 6).

```
#include "Input.h"
//Defineerime globaalse viida Input klassile, mida saab kasutada ühe klassi
eksemplari hoidmiseks.
Input* g_pInput = 0;

//Nüüd on aeg defineerida tagasihelistus funktsioonid,
//millega katame mõned sisendi juhud mida Marmalade genereerib ekraani
puudutusel.
void Input::TouchButtonCB(s3ePointerEvent* event)
{
    g_pInput->m_PrevTouched = g_pInput->m_Touched;
    g_pInput->m_Touched     = event->m_Pressed != 0;
    g_pInput->m_X           = event->m_x;
    g_pInput->m_Y           = event->m_y;
}

void Input::TouchMotionCB(s3ePointerMotionEvent *event)
```

```
{
    g_pInput->m_X = event->m_x;
    g_pInput->m_Y = event->m_y;
}

void Input::MultiTouchButtonCB(s3ePointerTouchEvent *event)
{
    g_pInput->m_PrevTouched     = g_pInput->m_Touched;
    g_pInput->m_Touched         = event->m_Pressed != 0;
    g_pInput->m_X               = event->m_x;
    g_pInput->m_Y               = event->m_y;
}

void Input::MultiTouchMotionCB(s3ePointerTouchMotionEvent *event)
{
    g_pInput->m_X = event->m_x;
    g_pInput->m_Y = event->m_y;
}

//Input klassi konstruktor, kus kontrollime, kas antud seadeldis suudab
tuvastada mitu puudutust korraga
//või ainult ühe puudutuse.
Input::Input() : m_Touched(false), m_PrevTouched(false)
{
    if (s3ePointerGetInt(S3E_POINTER_MULTI_TOUCH_AVAILABLE) != 0) {
        s3ePointerRegister(S3E_POINTER_TOUCH_EVENT,
(s3eCallback)MultiTouchButtonCB, 0);
        s3ePointerRegister(S3E_POINTER_TOUCH_MOTION_EVENT,
(s3eCallback)MultiTouchButtonCB, 0);
    }
    else
    {
        s3ePointerRegister(S3E_POINTER_BUTTON_EVENT,  (s3eCallback)TouchButtonCB,
0);
        s3ePointerRegister(S3E_POINTER_MOTION_EVENT,  (s3eCallback)TouchMotionCB,
0);
    }
}

//Uuendamis funktsioon, mida tavaliselt kutsub välja main() funktsioon.
Uuendatakse üks kord kaadri jooksul ja funktsioon ise kuulab s3ePointerUpdate
funktsiooni.
void Input::Update()
{
    s3ePointerUpdate();
}

//Reset funktsioon, mis nullib ära puudutused
void Input::Reset()
{
    m_PrevTouched   = false;
    m_Touched       = false;
```

```
}
```

Koodinäide 6. Input.cpp (Hopwood, Stage 2 - Input and audio)

4.6. Stseenid

Kui Input klass on loodud, oleks tore seda testida. Aga enne veel kui seda on võimalik testida peaks looma mõne mängustseeni kus seda oleks võimalik katsetada. Stseeni defineerimine on juba veidi mahukam ülesanne ja nõuab kahe alamklassi laiendamist. Avame enda scene.h päisefaili ja kirjutame sinna (Koodinäide 7).

```
#ifndef __HelloMaailm__scene__
#define __HelloMaailm__scene__

#include <list>
#include "Iw2DSceneGraph.h"
#include "IwTween.h"

using namespace Iw2DSceneGraph;
using namespace Iw2DSceneGraphCore;
using namespace IwTween;
class SceneManager;

class Scene : public CNode
{

protected:
    unsigned int        m_NameHash;
    bool                m_IsActive;
    bool                m_IsInputActive;
    SceneManager*       m_Manager;
    CTweenManager       m_Tweener;

public:
    bool                IsActive() const                    { return m_IsActive;
}
    void                   SetActive(bool active)                { m_IsActive =
active; }
    void                SetName(const char* name);
    unsigned int        GetNameHash() const                 { return m_NameHash;
}
    void                     SetManager(SceneManager* manager)    { m_Manager =
manager;}
    void                SetInputActive(bool active)          { m_IsInputActive =
active;}
    CTweenManager&      GetTweener()                         { return m_Tweener;
}

public:
```

```
    Scene();
    virtual ~Scene();
    virtual void    Init();
    virtual void    Update(float deltaTime = 0.0f, float alphaMul = 1.0f);
    virtual void    Render();
};

class SceneManager
{
protected:
    Scene*              m_Current;
    Scene*              m_Next;
    std::list<Scene*>   m_Scenes;

public:
    Scene* GetCurrent()         { return m_Current; }
public:
    SceneManager();
    ~SceneManager();
    void    SwitchTo(Scene *scene);
    void    Update(float deltaTime = 0.0f);
    void    Render();
    void    Add(Scene* scene);
    void    Remove(Scene* scene);
    Scene*  Find(const char* name);
};

extern SceneManager* g_pSceneManager;

#endif /* defined(__HelloMaailm__Scene__) */
```

Koodinäide 7. scene.h (Hopwood, Stage 3 - Scenes, sprites and labels)

Samm-sammult seletan ma mida mõni siiamaani tundmatu koodirida tähistab:

- `class Scene` : public tähistab seda, et klass Scene laiendab klassi `CNode`, mis on baasklass kõikidele objektidele kes peaksid kuuluma scene graph klassi. Scene graph klass on siis, nagu eelpool mainitud, stseenide graaf kus on võimalik defineerida hierarhiaid.
- `unsigned int m_NameHash;` defineerib meie loodud stseeni objektile nimeräsi muutuja. Kõikide stseenide nimed lastakse läbi paiskefunktsiooni, et saavutada parem jõudlus stseenide otsingul.
- `bool m_IsActive;` tähistab seda, kas antud stseen on aktiivne või mitte.
- `SceneManager* m_Manager;` on stseenihaldur, mis igal stseenil peab olema.
- `CTweenManager m_Tweener;` on animatsiooni haldur mida meie klassil vaja on.

- `bool IsActive() const { return m_IsActive; }` on funktsioon mis objektorienteeritud programmeerimise heade tavade kohaselt tagastab klassi väärtuseid.
- `virtual void Init();` defineerib virtuaalse funktsiooni `Init()` mis initsialiseerib klassi luues tema algväärtused. `Virtual` tähistab seda, et see meetod kasutab dünaamilist sidumist.
- `virtual void Update(float deltaTime = 0.0f, float alphaMul = 1.0f);` Virtuaalne meetod mida hakkab välja kutsuma `SceneManager` üks kord kaadri jooksul. Igal stseenil on enda `Update()` meetod mis uuendab praegusel hetkel aktiivset stseeni.
- `Scene* m_Current;` on praegusel hetkel aktiivne stseen
- `std::list<Scene*>  m_Scenes;` on stseenide list

Järgmisena tuleb meil implementeerida eelnevalt defineeritud meetodid, aveme scene.cpp (Koodinäide 8).

```
#include "scene.h"
#include "IwGx.h"
#include "Input.h"

SceneManager* g_pSceneManager = 0;

Scene::Scene() : m_NameHash(0), m_IsActive(true), m_IsInputActive(false)
{
    m_IsVisible = false;
}

Scene::~Scene()
{
}

void Scene::SetName(const char* name)
{
    m_NameHash  = IwHashString(name);
}

void Scene::Init()
{
}

void Scene::Update(float deltaTime, float alphaMul)
{

    if (!m_IsActive)
        return;

    m_Tweener.Update(deltaTime);
    CNode::Update(deltaTime, alphaMul);
```

```
}

void Scene::Render()
{

    CNode::Render();

}
//
// SceneManager
//
SceneManager::SceneManager() : m_Current(0), m_Next(0)
{
}

SceneManager::~SceneManager()
{

    for    (std::list<Scene*>::iterator    it    =    m_Scenes.begin();    it    !=
m_Scenes.end(); it++) {
        delete *it;
    }
}

void SceneManager::Add(Scene* scene)
{
    m_Scenes.push_back(scene);
    scene->SetManager(this);
}

void SceneManager::Remove(Scene* scene)
{
    m_Scenes.remove(scene);
}

Scene* SceneManager::Find(const char *name)
{
    unsigned int name_hash = IwHashString(name);
    for    (std::list<Scene*>::iterator    it    =    m_Scenes.begin();    it    !=
m_Scenes.end(); it++) {
        if ((*it)->GetNameHash() == name_hash) {
            return *it;
        }
    }
    return 0;
}

void SceneManager::Update(float deltaTime)
{

    for(std::list<Scene*>::iterator it = m_Scenes.begin(); it != m_Scenes.end();
it++)
        (*it)->Update(deltaTime);
```

```
}

void SceneManager::Render()
{

    for(std::list<Scene*>::iterator it = m_Scenes.begin(); it != m_Scenes.end();
it++)
        (*it)->Render();

}

void SceneManager::SwitchTo(Scene *scene)
{

    m_Next = scene;

    if(m_Current == 0)
    {
        m_Current = m_Next;
        m_Current->SetActive(true);
        m_Current->m_IsVisible = true;
        m_Current->SetInputActive(true);
        m_Next = 0;
    }else{
        m_Next->SetActive(true);
        m_Next->SetInputActive(true);
        m_Next->SetActive(true);
        m_Next->m_IsVisible = true;

        m_Current->SetActive(false);
        m_Current->m_IsVisible = false;
        m_Current->SetInputActive(false);
        m_Current = m_Next;
        m_Next = 0;

    }

}
```

Koodinäide 8. scene.cpp (Hopwood, Stage 3 - Scenes, sprites and labels)

`SceneManager::Find(const char *name)` kasutab C++ listi iteraatorit, käies läbi kogu listi ja tagastades stseeni mille nime räsikood vastab otsitava stseeni nime räsikoodiga.

4.7. Ressurssidehalduri loomine

Avame faili resources.h ja modifitseerime seda (Koodinäide 9).

```
#ifndef __HelloMaailm__resources__
#define __HelloMaailm__resources__
```

```
#include "Iw2D.h"
#include "Iw2DSceneGraph.h"

using namespace Iw2DSceneGraph;

class Resources
{

protected:
    CIw2DImage*         Dollar;
    CIw2DImage*         MenuBg;
    CIw2DImage*         GameBg;
    CIw2DFont*              Font;

public:
    CIw2DImage*         getDollar()            { return Dollar; }
    CIw2DImage*         getMenuBg()        { return MenuBg; }
    CIw2DImage*         getGameBg()       { return GameBg; }
    CIw2DFont*              getFont()              { return Font; }

public:
    Resources();
    ~Resources();
};

extern Resources* g_pResources;

#endif /* defined(__HelloMaailm__resources__) */
```

Koodinäide 9. resources.h (Hopwood, Stage 3 - Scenes, sprites and labels)

`CIw2DImage*  Dollar;` Sellega defineerime protected muutuja heade objektorienteeritud programmeerimise tavade kohaselt. Public meetoditena kirjutame jällegi protected muutujate ligipääsu meetodid.

Ja vastav implementatsioonifail, resources.cpp, hakkab välja nägema selline (Koodinäide 10).

```
#include "resources.h"
#include "Iw2D.h"

Resources::Resources()
{

    Dollar  = Iw2DCreateImage("textures/tenEuros.png");
    MenuBg  = Iw2DCreateImage("textures/press_to_start.png");
    GameBg  = Iw2DCreateImage("textures/backdrop_flappy_dollar.png");
    Font    = Iw2DCreateFont("fonts/arial8.gxfont");

}
Resources::~Resources()
```

```
{

    delete Dollar;
    delete MenuBg;
    delete GameBg;
    delete Font;
}

//Globaalne muutuja resursside halduseks
Resources* g_pResources = 0;
```

Koodinäide 10. resources.cpp (Hopwood, Stage 3 - Scenes, sprites and labels)

4.8. Mängustseeni loomine

Jäänud on veel eelviimane samm, enne kui saab antud rakendust käivitada, nimelt mängustseeni loomine. Selleks peame avama game.h faili ja sinna kirjutama (Koodinäide 11).

```
#ifndef __HelloMaailm__game__
#define __HelloMaailm__game__

#include "scene.h"

class Game : public Scene
{

protected:
    float       currentRoundScore;
    float       lastClickPosition;
    float       nextPosition;
    float       zeroEurPos;

    CLabel*     scoreLabel;
    CSprite*    background;
    CSprite*    dollarSprite;

private:
    void initUI();

public:
    Game() {}
    ~Game();

    void Init();

    void Update(float deltaTime = 0.0f, float alphaMul = 1.0f);

    void Render();

    void switchToScene(const char* scene_name);
    void addToRoundScore(float score);
```

```
    void newGame();

};
```

Koodinäide 11. game.h

protected meetodiga defineerime jälle vaid klassisisesed muutujad, public meetoditena defineerime scene.h meetodite ülekirjutused ja ka mõned uued meetodid, mida meie lihtne mäng vajab.

Ja nüüd kõige mahukam koodifail, game.cpp, mis sisaldab ka veidi mänguloogikat (Koodinäide 12).

```
#include "IwGx.h"
#include "IwHashString.h"
#include "Input.h"
#include "resources.h"
#include "game.h"
using namespace IwTween;
Game::~Game()
{
}

void Game::addToRoundScore(float score){
    currentRoundScore += score;
    char str[16];
    snprintf(str, 16, "%.2f", currentRoundScore);
    scoreLabel->m_Text = str;
}

void Game::newGame()
{
    currentRoundScore = 0;
    zeroEurPos      =      (float)IwGxGetScreenHeight()-dollarSprite->GetImage()-
>GetHeight();
    dollarSprite->m_X = (float)IwGxGetScreenWidth() / 2;
    dollarSprite->m_Y = zeroEurPos;

}

void Game::Update(float deltaTime, float alphaMul)
{
    if (!IsActive())
        return;

    Scene::Update(deltaTime, alphaMul);

    lastClickPosition = dollarSprite->m_Y - 500;
    nextPosition = dollarSprite->m_Y + 300;

    if  (m_IsInputActive  &&  m_Manager->GetCurrent()  ==  this  &&  !g_pInput-
>m_Touched && g_pInput->m_PrevTouched) {

        g_pInput->Reset();
```

```
        if (dollarSprite->HitTest(g_pInput->m_X, g_pInput->m_Y)) {

            m_Tweener.Tween(2.0f,
                            FLOAT, &dollarSprite->m_Y, lastClickPosition,
                            EASING, Ease::powOut, END);

            addToRoundScore(1);

        }else{

        }
    }else{
        if  (dollarSprite->m_Y    >=  ((float)IwGxGetScreenHeight()-dollarSprite-
>GetImage()->GetHeight())) {

            zeroEurPos  =  (float)IwGxGetScreenHeight()-dollarSprite->GetImage()-
>GetHeight();
            m_Tweener.Tween(4.0f,
                            FLOAT, &dollarSprite->m_Y, zeroEurPos,
                            EASING, Ease::linear, END);

        }else{

            m_Tweener.Tween(4.0f,
                            FLOAT, &dollarSprite->m_Y, nextPosition,
                            EASING, Ease::linear, END);

        }
    }
}

void Game::Render()
{

    Scene::Render();

}

void Game::initUI()
{

    background = new CSprite();
    background->m_X = (float)IwGxGetScreenWidth() / 2;
    background->m_Y = (float)IwGxGetScreenHeight() / 2;
    background->SetImage(g_pResources->getGameBg());
    background->m_W = background->GetImage()->GetWidth();
    background->m_H = background->GetImage()->GetHeight();
    background->m_AnchorX = 0.5;
    background->m_AnchorY = 0.5;

    background->m_ScaleX = (float)IwGxGetScreenWidth() / background->GetImage()-
>GetWidth();
```

```
    background->m_ScaleY    =    (float)IwGxGetScreenHeight()    /    background-
>GetImage()->GetHeight();
    AddChild(background);


    CLabel* scoreLabelText = new CLabel();
    scoreLabelText->m_X = 10;
    scoreLabelText->m_Y = 0;
    scoreLabelText->m_W = (float)IwGxGetScreenWidth();
    scoreLabelText->m_H = 30;
    scoreLabelText->m_Text = "Clicks: ";
    scoreLabelText->m_AlignHor = IW_2D_FONT_ALIGN_LEFT;
    scoreLabelText->m_AlignVer = IW_2D_FONT_ALIGN_TOP;
    scoreLabelText->m_Font = g_pResources->getFont();
    scoreLabelText->m_Color = CColor(0xff, 0xff, 0x30, 0xff);
    AddChild(scoreLabelText);

    scoreLabel = new CLabel();
    scoreLabel->m_X = 80;
    scoreLabel->m_Y = 0;
    scoreLabel->m_W = (float)IwGxGetScreenWidth();
    scoreLabel->m_H = 30;
    scoreLabel->m_Text = "0";
    scoreLabel->m_AlignHor = IW_2D_FONT_ALIGN_LEFT;
    scoreLabel->m_AlignVer = IW_2D_FONT_ALIGN_TOP;
    scoreLabel->m_Font = g_pResources->getFont();
    scoreLabelText->m_Color = CColor(0xff, 0xff, 0xff, 0xff);
    AddChild(scoreLabel);

}


void Game::Init()
{

    Scene::Init();

    currentRoundScore = 0;

    initUI();

    dollarSprite = new CSprite();
    dollarSprite->SetImage(g_pResources->getDollar());
    dollarSprite->m_X = (float)IwGxGetScreenWidth() / 2;
    dollarSprite->m_Y = (float)IwGxGetScreenHeight() / 2;
    dollarSprite->m_W = dollarSprite->GetImage()->GetWidth();
    dollarSprite->m_H = dollarSprite->GetImage()->GetHeight();
    dollarSprite->m_AnchorX = 0.5;
    AddChild(dollarSprite);

}
```

Koodinäide 12. game.cpp

- `void Game::addToRoundScore(float score);` on meetod mis lisab igal rahatähe puudutamisel skoorile ühe punkti.
- `void Game::newGame();` initsialiseerib mängu algoleku.
- `void Game::Update(float deltaTime, float alphaMul);` on meetod mida kutsub välja stseenihaldur üks kord kaadri jooksul. Kui kaua üks kaader kestab on ka ära defineeritud funktsiooni parameetri, deltaTime, poolt. Selles meetodis samuti jälgime meie DollarSprite asukohta ja uuendame seda vastavalt sisendile.
- `m_Tweener.Tween()` on funktsioon mis oskab vastavalt sisendile arvutada matemaatiliste funktsioonide väärtusi ja neid järk järgult tagastada. Näiteks antud juhul kasutame me tween objekti et liigutada DollarSprite objekti. Funktsioon võtab varieeruva arvu parameetreid, antud juhul parameetrid tähistavad seda, et funktsiooni väärtusi peaks muutma 4 sekundi jooksul, FLOAT väärtuseid kasutades, &dollarSprite->m_Y väärtusi muutes, zeroEurPos poole aeglaselt libisedes ja lineaarselt sisendväärtuseid suurendades. END token tähistab funktsiooni parameetrite lõppu.
- `void Game::initUI()` initsialiseerib graafilise kasutajaliidese, luues uued sprite objektid kõigile eelnevalt defineeritud muutujatele.
- `void Game::Init()` initsialiseerib mängustseeni üldiselt

4.9. Main loop

Nüüd on jäänud veel viimane samm enne kui saab projekti käivitada. Avame projektinimega päisefaili, antud juhul HelloMaailm.h, ja kirjutame sinna (Koodinäide 13).

```
#if !defined(__HELLO_MAAILM__)
#define __HELLO_MAAILM__

#include "IwTween.h"
using namespace IwTween;

extern CTweenManager*       g_pTweener;


#endif  // __MAIN_H__
```

Koodinäide 13. HelloMaailm.h

Ja nüüd avame HelloMaailm.cpp ja kirjutame omakorda sinna (Koodinäide 14).

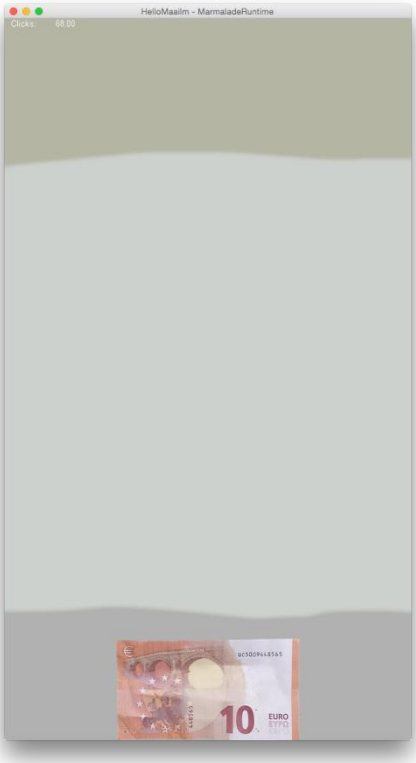
```
#include "HelloMaailm.h"
#include "IwTween.h"
#include "Iw2D.h"
#include "Input.h"
#include "scene.h"
#include "mainMenu.h"
#include "resources.h"
#include "game.h"

using namespace IwTween;

#define FRAME_TIME (1.0f / 60.0f)

CTweenManager* g_pTweener = 0;

int main()
{
    //Initsialiseerib Iw2D mooduli. Kohustuslik samm, enne kui Iw2d käsklusi
kasutama saab hakata
    Iw2DInit();

    g_pResources    = new Resources();
    g_pInput        = new Input();
    g_pSceneManager = new SceneManager();
    g_pTweener      = new CTweenManager();


    Game* game = new Game();
    game->SetName("game");
    game->Init();
    g_pSceneManager->Add(game);

    g_pSceneManager->SwitchTo(game);


    while (!s3eDeviceCheckQuitRequest()) {


        uint64 new_time = s3eTimerGetMs();

        g_pInput->Update();

        g_pTweener->Update(FRAME_TIME);

        g_pSceneManager->Update(FRAME_TIME);

        Iw2DSurfaceClear(0xff000000);

        //Joonistab Iw2DSurface objektile antud pildi.

        g_pSceneManager->Render();
```

```
        //Kuvab Iw2DSurface objekti.
        Iw2DSurfaceShow();

        int yield = (int)(FRAME_TIME * 1000 - (s3eTimerGetMs() - new_time));
        if (yield < 0)
            yield = 0;

        s3eDeviceYield(yield);
    }

    //Enne kui rakendus suletakse, peab mälust kustutama tekitatud objektid!

    delete g_pInput;
    delete g_pSceneManager;
    delete g_pResources;
    delete g_pTweener;
    //Sarnaselt initsialiseerivale funktsioonile, on see kohustuslik samm, enne
rakenduse sulgemist.
    Iw2DTerminate();

    return 0;
}
```

Koodinäide 14. HelloMaailm.cpp lõplik

- `#define FRAME_TIME (1.0f / 60.0f)` defineerime aja, kui kaua peaks üks kaader kestma.
- `g_pResources = new Resources();` Muretseme endale uue isendi globaalsest ressursside klassist.
- `Game* game = new Game();` Loome uue mängustseeni objekti.
- `game->SetName("game");` anname mängustseenile identifikaatori.
- `game->Init();` kutsume välja mängustseeni `Init()` meetodi.
- `g_pSceneManager->Add(game);` Lisame stseenihaldurile uue stseeni.
- `g_pSceneManager->SwitchTo(game);` ütleme stseenihaldurile, et antud stseen kuvataks.

Lõpuks ometi võib vajutada nuppu run, et testida kas antud rakendus käivitub. Kui kõik on vastavalt juhendile tehtud ei tohiks probleeme esineda. Lõpptulemus võiks välja näha midagi sellist (Joonis 21).



Joonis 21. Lihtne mänguke

Klõpsates rahatähel, suureneb Clicks väärtus, rahatäht tõuseb kõrgemale ja alustab jälle langemist. Mängu mõte on vaadata, kes rohkem klõpse koguda jõuab (Joonis 22).

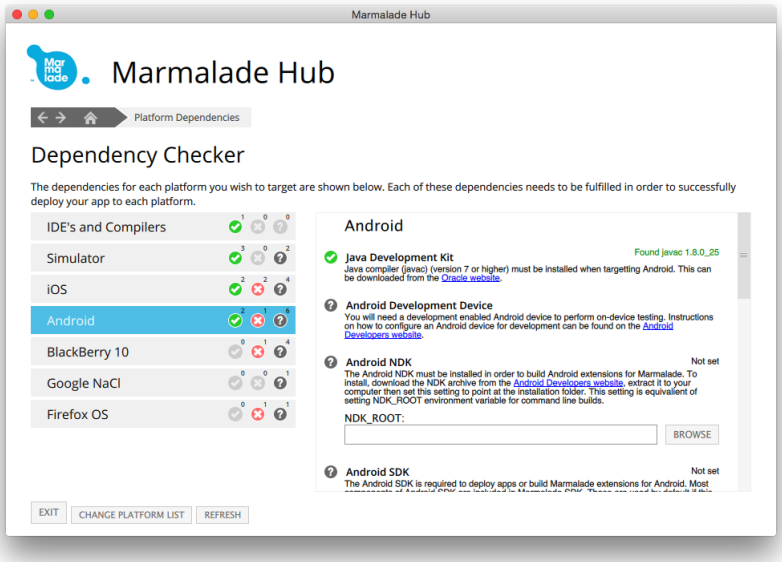
Joonis 22. Lihtne mänguke kaks

5. Mobiiltelefonis rakenduse käivitamine

Oleks kurb kui jätaks kasutamata võimaluse antud rakendust mobiiltelefonis proovida. Antud juhul lihtsuse mõttes jooksutame antud rakendust Androidi telefonis. Mida on selleks vaja:

- Androidi seadeldis mille operatsioonisüsteemi versioon on vähemalt 2.2 (*Gingerbread*) või uuem.
- *Java SE SDK (JDK)* on kohustuslik, kui on soov androidi platvormile kompileerida.
- Windowsi kasutajatele oleks tore kui oleks installitud ka *ADB (Android Debug Bridge)* draiver, et Marmalade Hub saaks otse uusimat projekti versiooni installida ühendatud seadmele.

Esimene samm oleksgi siis *Java SDK* installimine. Kõigepealt võiks kontrolli mõttes avada *Marmalade Hub* > *Dependency checker* ja vaadata kas Androidi menüüs on olemas linnuke kohal kus on kirjas *Java Development Kit* (Joonis 23).



Joonis 23. Androidi sõltuvuste kontroll

Kui Marmalade Java Development Kiti ei leia, peab minema aadressile http://www.oracle.com/technetwork/java/javase/downloads/index.html ja hankima endale *Java platform (JDK)*. Marmalade soovitab kasutada *JDK 1.7* versioone. Võib juhtuda ka seda, et rakendus töötab ka uuema *JDK* versiooniga. Kui peaks muresid tekkima oleks esimene koht see mida kontrollida. Vanemad versioonid *JDK*-st on saadaval, kui kerida eelpool mainitud veebilehel täiesti lõppu ja avada *Java Archive*. Kui *JDK* on installitud, tuleb windowsi kasutajatel lisada installitud *JDK* bin kaust ka PATH variable alla Süsteemi muutujate all, et saaks antud teegi käskluseid jooksutada käsurealt, sõltumata kaustast kus viibitakse.

Windowsi kasutajatel tuleb installida ka spetsiifilise Androidi seadme draiverid mille peal antud rakendust üritatakse jooksutada ja *ADB* paigaldamine on samuti lausa hädavajalik.

5.1. Package, Install, Run

Kui kõik eelnevad punktid on täidetud saab asuda lõbusama osa kallale, milleks on rakenduse jooksutamine telefonis. Avame enda projekti Marmalade Hub-is (Joonis 12). valime Deployment menüült Platformiks Android, architecture arm ja configuration menüü alt valime android(default).

Konfiguratsioone on võimalik ka ise juurde luua vastavalt vajadusele, aga antud olukorras peaks vaikeseadetest piisama. Vajutame *Package, Install and Run*. Tükk aega ragistab ja mõtleb, näitab mõned reklaamid ja lõpptulemus võiks olla selline (Joonis 24).

Joonis 24. Rakendus mobiilis

Kokkuvõte

Siinse seminaritöö käigus valmis õppematerjal Marmalade raamistiku kohta.

Seminaritöö eesmärk, luua lühike kuid ülevaatlik õppematerjal sellest kuidas kasutada Marmalade raamistikku rakenduste loomiseks, sai täidetud. Samuti võib lugeda täidetuks eesmärgi leida Marmalade raamistiku mõned murekohad ja tugevused.

Negatiivsetest aspektidest tooks autor välja selle, et Marmalade raamistik ei ole väga algajasõbralik. Raamistiku kasutajaid ei ole väga palju, niiet ka õppematerjali antud raamistiku kohta ei eksisteeri nii suures hulgas kui näiteks Unity 3D kohta. Ka on tasuta litsentsi puhul kasutajatugi peaaegu et olematu.

Positiivsete aspektidena tooks autor välja võimaluse, kasutada endale harjumuspäraseks saanud arenduskeskkonnana ka Xcode. Samuti oli autori arvates rakenduse käivitamine telefonis selge ja kiire protsess. Dokumentatsioon mida Marmalade pakub on samuti kõrgel tasemel.

Kui tööd edasi arendada oleks üks võimalikest variantidest kolmanda osapoolena võrrelda kahte rakendust, üks neist mis on loodud kasutades Unity 3D mängumootorit ja teine Marmalade raamistikku.

Kasutatud allikad

Hopwood, M. (kuupäev puudub). *Stage 2 - Input and audio.* Kasutamise kuupäev: 14. Oktoober 2015. a., allikas Made with Marmalade: http://docs.madewithmarmalade.com/display/MD/Input+and+Audio

Hopwood, M. (kuupäev puudub). *Stage 3 - Scenes, sprites and labels.* Kasutamise kuupäev: 19. oktoober 2015. a., allikas Made with Marmalade: http://docs.madewithmarmalade.com/display/MD/Scenes%2C+Sprites+and+Labels

Marmalade Technologies Ltd. (kuupäev puudub). *What do we support.* Kasutamise kuupäev: 24. Oktoober 2015. a., allikas Marmalade Documentation: http://betadocs.madewithmarmalade.com/display/MD/What+do+we+support#Whatdowesupport-developmentplatformsupport

PopCap Games. (kuupäev puudub). *PopCap games.* Kasutamise kuupäev: 23. September 2015. a., allikas PvZ_Title_1024_600.jpg: http://static-www.ec.popcap.com/www.popcap.com/sites/all/themes/popcap_2012/games/plants_vs_zombies/screenshots/nook/PvZ_Title_1024_600.jpg

Rudman, M. (2. Märts 2015. a.). *Cross-platform comparisons: Performance and package size of Marmalade and Unity applications.* Kasutamise kuupäev: 28. Oktoober 2015. a., allikas Made with marmalade: https://www.madewithmarmalade.com/blog/marmalade-unity-app-performance-comparison

The University of Warwick. (kuupäev puudub). *Installing Geant4 with Visual Studio Windows.* Kasutamise kuupäev: 27. Oktoober 2015. a., allikas http://www2.warwick.ac.uk/fac/sci/physics/staff/research/bmorgan/geant4/installingonwindows/step_10_set_build_configuration.png