Факултет инжењерских наука Универзитета у Крагујевцу

Милош Р. Ристић

Софтвер за графички приказ симулације кретања плочице

завршни рад

Крагујевац, 2015.

Факултет инжењерских наука Универзитета у Крагујевцу

Назив студијског програма: Машинско Инжењерство

Ниво студија: Основне академске студије

Модул: Информатика у Инжењерству

Предмет: Софтверски инжењеринг

Број индекса: 711/2014

Милош Р. Ристић

Софтвер за графички приказ симулације кретања плочице

завршни рад

Комисија за преглед и одбрану:

1. **Др Ненад Филиповић, проф. - ментор**
2. _____
3. _____

Датум одбране:_____

Оцена:_____

У оквиру овог завршног рада кандидат треба да

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

Препоручена литература:

[1] .....

[2] .....

[3] .....

...

Крагујевац, датум

ментор:

Др Ненад Филиповић, проф.

_____

Резиме рада: У овом раду је приказан развој задате апликације коришћењем Visual Studio 2015 развојног окружења и програмског језика c++ . Представљен је поступак креирања апликације која симулира игрицу, разбијање цигала лоптицом одбијањем од плочице коју корисник покреће лево десно у доњем делу екрана. Циљ је да се све цигле униште а да не падне лоптица на под. Рад је употпуњен графичким приказом корака од почетка израде, па све до стартовања и тестирања апликације.

Кључне речи: Visual studio 2015, c++, CImg, Windows, Објектно орјентисано програмирање.

Summary: This paper presents the development of the given application using Visual Studio 2015 development enviroment an programming laguage c++. Presented is a method of creating applications which simulate game, brick breaking whit ball refusal of tiles that can be moved left and right in the lower part of the screen. The goal is to destroy all the bricks but ball that does not fall on the floor. The work Is complemated by a graphical representation of the steps from th beginning of creation, to the start-up and testing application.

Keywords : Visual studio 2015, c++, CImg, Windows, Object-Oriented Programming.

САДРЖАЈ

УВОД

Основно средство комуникације између учесника у процесу комуникације јесте језик. Програмски језици омогућавају креирање апликативног софтвера. Они омогућавају корисницима да саопште рачунарима шта треба да ураде и представљају основу за развој система. Програмски језици представљају скуп симбола и правила за писање програмског кода.

Програмирање је начин управљања помоћу дефинисања програма рада. За решавање комплексних математичких проблема постоје веома развијени програмски језици. Ови програми омогућују руковање елементарним операцијама, скуповима унапред програмираних операција.

Сваки језик користи другачији скуп правила и синтаксу која одређује правило за грађење одређене наредбе. Успех комуникације између учесника у процесу комуникације зависи од језика који мора бити разумљив. Начин комуникације између човека и компјутера је специфичан, тако да директна комуникација између човека и компјутера није могућа. Језик компјутера састоји се из симбола (0 и 1), а језик човека је богат и разноврстан.

Таквим језиком је било потребно одрадити апликацију која симулирати игрицу разбијање цигала. У времену где компјутери постају свакодневница, и где је непојмљиво више било шта радити без икакве помоћи рачунара, овакав вид апликације није новост. Овако нешто је већ развијено, али у наредном раду ће бити приказана манипулација унапред одређеним садржајем, и ауторов лични печат што се тиче развоја поменуте апликације у развојном окружењу Visual Studio 2015, и његовој алатки C++.

1
ПРОГРАМСКИ ЈЕЗИЦИ

Програмски језик је вештачки језик који се може користити за контролу понашања машине, нарочито рачунара. Програмски језици су дефинисани преко синтаксних и семантичких правила која респективно описују њихову структуру и значење. Многи програмски језици имају неку форму писаних спецификација њихове синтаксе и семантике а неки су дефинисани једино преко званичне имплементације. Програмски језици се користе да олакшају комуникацију са рачунаром приликом организовања и манипулације информација, али и да прецизно изразе алгоритме. Неки аутори ограничавају израз „програмски језик“ само на језике којима се могу изразити сви могући алгоритми, а понекад се користи израз „рачунарски језик“ који се односи на више ограничене вештачке језике. У међувремену је створено више хиљада програмских језика, и нови се стварају сваке године.[2]

Машински и асемблерски језици су језици ниског нивоа, који захтевају од програмера да се посвети управљању свим стварима везаним за чување података и операције над њима. На другом крају налазе се језици високог нивоа, који су ближи природном језику и ослобађају програмера бриге о тим стварима, такође читљивији и далеко лакши за писање програма.

Програмски језици се, према начину описивања рада програма, деле на функцијске (Lisp, Skim), процедуларне (C, Paskal, Basic), секвенцијалне и објектно-оријентисане (Java, Ada), структуралне (SQL) и многе друге. Програмски језици по овој подели могу бити мешовити, тј. да дозвољавају различите парадигме у оквиру истог програма, те нпр. C++ дозвољава и објектно-оријентисани и процедурални приступ, штавише процедурални приступ је неопходана при дефиницији почетне тачке програма у функцији main.

Машински језик се састоји од нумеричког кода за операције који одређени рачунар може директно извршити. Тај код је алфанумеричка серија 0 и 1, или бинарни код (бајт), који се често претвара у хексадецимални код (на бази броја 16), ради лакше читљивости и модификације. Инструкције машинских језика обично користе један број бајтова за представљање операција, сабирање на пример, а други за представљање операнда (бројева са којима се врши операција) и/или локације за следећу инструкцију. Асемблерски језик је један ниво изнад машинског језика.

Користи кратки мнемонички код за инструкције и омогућава програмеру да уноси имена за блокове меморије која садржи податке. Дизајниран је да омогући лако превођење у машински језик. Иако се блокови података у асемблерском језику позивају преко имена, а не преко адресе у меморији, ипак не постоји могућност софистикованог организовања сложених информација. Као и машински језик, асемблерски језик подразумева да програмер има одређено знање које ће му омогућити детаљно разрађивање рачунарске архитектуре. Корисно је пре свега због свих тих детаља који су важни, тј. приликом програмирања одређеног рачунара за међусобну интерацију са другим улазним и излазним уређајима, као нпр. штампач, скенер, одређени

уређаји за складиштење информација и важних података и друге пре свега оптичке и чврсте дискове. Помоћу њих се могу изражавати алгебарске операције на исти начин као и у математици, и где омогућавају коришћење потпрограма у којима се пакују најчешће коришћене операције, где се омогућава и њихово поновно искоришћавање. Машински језик као језик је тежак за писање и читање, јер не личи на уобичајено математичко представљање нити личи на природни језик, а његов сопствени код варира од рачунара до рачунара.[2]

1.1 Објектно-Оријентисано Програмирање

Објектно оријентисано програмирање (ООР) у рачунарству је једна од програмских парадигми. Од раних 1980-тих па до данас ова парадигма је постала најутицајнија парадигма у комерцијалном развоју софтвера. Програмски језици попут C++ и Јава програмски језик су својом популарношћу утврдили водећи статус ООП-а као de facto обавезног приступа при развоју софтвера данас, те и језици који традиционално нису били објектно оријентирани него су накнадно додали ООП концепте (нпр. Visual Basic, Pascal програмски језик итд.). Такође треба нагласити да се заснива на различитим техникама као што су наслеђивање, полиморфизам, енкапсулација, и модуларност.

Решавање проблема парадигмом објектног-оријентисаног програмирања је може се рећи исто као и начин на који људи размишљају и решавају одређене проблеме. Састоји се пре свега од индентификације објекта, а након тога и постављања објекта, који ће се користити, у одговарајућу секвенцу за решење одређеног проблема. Ту се пре свега ради о дизајну одређеног облика чија ће понашања као јединица у одређеној међусобној интеракцији решити одређени проблем. Та интеракција између објеката се састоји у једној размени порука, где свака одређена порука покреће енкапсулиране операције у поменутом објекту и самим тим решава део обично већег и ширег, али и сложенијег проблема. Гледано са аспекта решавања таквих проблема, објектно-оријентисано програмирање то обавља у четири корака:

1. индентификовање проблема
2. индентификовање објеката који су потребни за његово решење
3. индентификовање порука које ће објекти међусобно слати и примати
4. креирање секвенце порука објектима, које ће решавати проблем или проблеме.

У парадигми објектно оријентисаног програмирања, сами објекти су структуре података које представљају одређено и добро, тј. јасно дефинисано знање о спољашњем свету, тј стварности. Класична организација је у хијерархијској класи, причему свака класа објекта поседује одређене информације о особинама објекта које се чувају у инстанцама променљивих, где су све повезане (концептом асоцијације) са сваком инстанцом понаособ у одређеној класи. Сваки објекат у стању је да може да препозна други објекат, само преко његовог интерфејса. Подаци и сва остала логика сваког објекта су скривени од других објеката, јер се тиме омогућава раздвајање имплементације од одређеног понашања објекта у интеракцији са осталим објектима. Основне

особине објекта су: идентитет, стање и понашање. Идентитет представља назив објекта којим се одређени објекат разликује од свих осталих. Понашање објекта је дефинисано операцијама које је могуће извршити над објектом, а активирање одређене операције се извршава поруком. И стање објекта, које је део прошлости и садашњости и које одређује понашање објекта у будућности.

Свака класа се састоји од:

- података-чланова
- објекат-члан
- функција чланица-метода

Податак члан и објекат члан су атрибути класе, и користећи њих описујемо стање објеката. Објекат је модел ентитета где су атрибути есенцијалне особине ентитета које га описују. Методе дефинишу понашање објеката. Темељи објектно-оријентисаног програмирања су:

- Апстракција
- Енкапсулација
- Модуларност
- Полиморфизам
- Наслеђивање

Апстракција је поступак раздвајања битног од небитног. У току овог поступка неопходно је уочити који подаци и везе су битне за дати домен проблема, и оне битне податке имплементирати у класи. Постоје три групе апстракције: апстракција предмета, апстракција процеса и синтетска апстракција (апстракција виртуелне машине).

Енкапсулација подразумева поступке којима се обједињавају стања и понашања у једну целину. Излазни резултат свега овога је класа. Поред тога, енкапсулација такође подразумева и обезбеђивање контроле приступа у циљу поштовања принципа скривања информација.

Модуларност Модуларност је поступак разбијања програма на мање делове, где се добија могућност да они могу сами, и аутономно да функционишу. Модуларност се примењује у циљу омогућења вишеструке употребе софтверских компоненти, односно тзв. поновна употреба кода (code reuse).

Полиморфизам У директној вези са наслеђивањем је и полиморфизам, који представља вероватно једно од моћнијих својстава објектно оријентисаног програмирања. То подразумева да објекат извршава операцију на начин својствен изведеној класи којој припада, иако му се приступа као објекту основне класе.

Наслеђивање Један од фундаменталних аспеката ООР-а је наслеђивање. Што се тиче наслеђивања, ту постоји више врста, као што је наслеђивање имплементације, наслеђивање интерфејса, итд. Такође поред тога, зависно од програмског језика и његове ООР имплементације, наслеђивање се може поделити на једноструко и вишеструко.

Суштинска дефиниција наслеђивања поред свега је то да оно представља везу између класа у којој једна класа наслеђује структуру и понашање друге класе-једноструко наслеђивање, или више класа-вишеструко наслеђивање. У наредном случају се ради о програмском језику C++, који подржава само једноструко наслеђивање. Класа која је наслеђује назива се базном или основном класом, а класа која је наслеђује, назива се изведеном

класом. За наслеђивање се може рећи да ке хијерархијска организација класа, где једна класа наслеђује другу и задржава комплетан садржај класе коју наслеђује, а тај садржај може проширити, или редефинисати.[3]

1.2 Visual Studio 2015 развојно окружење

Visual Studio 2015 је јединствен скуп алатки које омогућава програмерима и развојним тимовима да направе квалитетне апликације на Microsoft-овим платформама. Подршка за развој оперативног система Windows Vista и система Microsoft Office помаже програмерима да креирају ефектне и богате клијентске апликације. Сада, уз укључену подршку ASP.NET AJAX и програмски додатак Silverlight за Visual Studio 2015, програмери могу такође да граде велики број богатих интерактивних апликација за Web.

VS представља једну од моћнијих програмских алатки-тј визуелно развојно окружење које омогућава лако коришћење многих технолошких погодности које пружа .NET Framework и C++. Веома просто руковање у ствари подразумева да VS у прилично великом делу помаже кориснику, и мноштво других сложених послова обавља уместо њега. Преведено то подразумева да уместо корисника прави целокупан костур апликације и тиме ослобађа корисника од тог сложеног посла и учења самог начина на који би то морао да ради. Како се Microsoft платформа буде и даље развијала у својим могућностима кроз производе система Windows Server 2015 и SQL Server 2015, Visual Studio 2015 ће наставити да буде јединствено окружење које је програмерима и развојним тимовима неопходно за успешан рад. Прво издање програма Visual Studio појавило се на тржишту 1997 године и садржао је одвојене IDE-ове (који су захтевали посебну инсталацију) за програме Visual C++, Visual Basic, J++, и алатку познату као InterDev. Програм Visual Studio 6.0 имао је драстична побољшања која су означила рођење програма Visual Basic 6 и која су осликавала идеју скупа јединствених услуга за све језике. Visual Studio .NET 2002 i Visual Studio .NET 2003 су ту идеју спровели у дело са апликацијом .NET Framework. По први пут један програмер је могао да напише апликацију на језику по свом избору коришћењем могућности општег скупа алатки укључујући дизајнере, контроле „превуци и отпусти“ и IntelliSense. Заједно са порастом продуктивности програмера, дошло је и до пораста у величини и комплексности развојних пројеката и тимова. Visual Studio 2005 је настао да би помогао програмерима у тимовима различитих величина да повећају међусобну сарадњу и смање комплексност развоја. Са сваким новим издањем, Microsoft је поново потврдио своју посвећеност оспособљавању програмера тако што је оставарио комуникацију са заједницом како би дала повратне информације ради усавршавања производа. Ни Visual Studio 2015 није изузетак. Фокус сваког новог издања програма Visual Studio увек је настојао да буде што бољи и успешнији. У Visual Studio 2015 продуктивност програмера се не завршава са уређивачем кода и чаробњацима.

Проширивањем фокуса на продуктивност, архитектуре за апликације и основну платформу, Visual Studio 2015 вам омогућава да решавате нове пословне проблеме док истовремено смањујете укупне прошкове креирања апликација.[1]

1.3 C++

C++ је виши програмски језик који је првобитно развијен у Bell Labs (лабораторији телекомуникационе компаније Bell) за објектно оријентисано програмирање у пројекту под руководством Bjarnea Stroustrupa током 1980их као проширење програмском језикуC, па му је оригинално име било „C са класама“ (енг. *C with classes*). Због велике потражње за објектно оријентисаним језицима и способностима, стандард за програмски језик C++ ратификован је 1998. у стандарду ISO/IEC 14882. [4]

Programski jezik C++ je nasledio celokupnu sintaksu od programskog jezika C, a dodata su sledeća proširenja:

- klase
  - definisanje funkcija članica i ograničavanja njihovog nivoa pristupa (javne, zaštićene i privatne)
  - nasleđivanje klasa
  - virtuelne i apstraktne metode klasa
  - konstruktori i destruktori
  - definisanje funkcija za operatore nad klasnih tipovima (tzv. „operatorske funkcije“)
- reference (tip podataka uveden radi pojednostavljivanja rada sa pokazivačima)
- imenski prostori (eng. *namespace*)
- šabloni (eng. *Templates*)
- novi operatori za manipulaciju dinamičkom memorijom, `new` i `delete`.
- niz novih biblioteka objekata i funkcija:
  - novi pristup problemu datoteka, standardnog ulaza i izlaza itd. u obliku *tokova* (`iostream`, `fstream`, `sstream` itd.)
  - nova standardna biblioteka, po nazivu „STL“, koja obuhvata rad sa vektorima (`vector`), mapama (`map`), skupovima (`set`), niskama (`string`), re dovima (`queue`), itd.
  - biblioteka koja sadrži vrlo širok skup algoritama za rešavanje čestih problema, poput efikasnog pronalaženja elementa u određenoj strukturi podataka, sortiranja, itd. (`algorithm`).

Sve promene uvedene u C++ su izgrađene na C-ovim funkcijama, odnosno sav kod napisan na C++ se interno prevodi u C-ov kod. C++ je zamišljen tako da se svaki dotadašnji program napisan u C-u može pokrenuti i pomoću C++-ovog kompajlera.

Кључне речи c++ су: asm, float, sizeof, auto, float, static, bool, for, break, friend, struct, case, goto, switch, catch, if, template, char, inline, this, class, int, trow, const, long, true, mutable, try, continue, namespase, typeidef, default, new, typename, typied, delete, do, double, else, enum, explicit, export, extern, operator, private,

protected, public, register, return, short, signed, union, usigned, virtual, void, volatile, wchar_t, while.

Кључне речи представљају резервисане индентификаторе који имају одговарајуће значење и које користи језик C++. Индентификатори представљају имена именских простора, класа, метода, променљивих, итд.

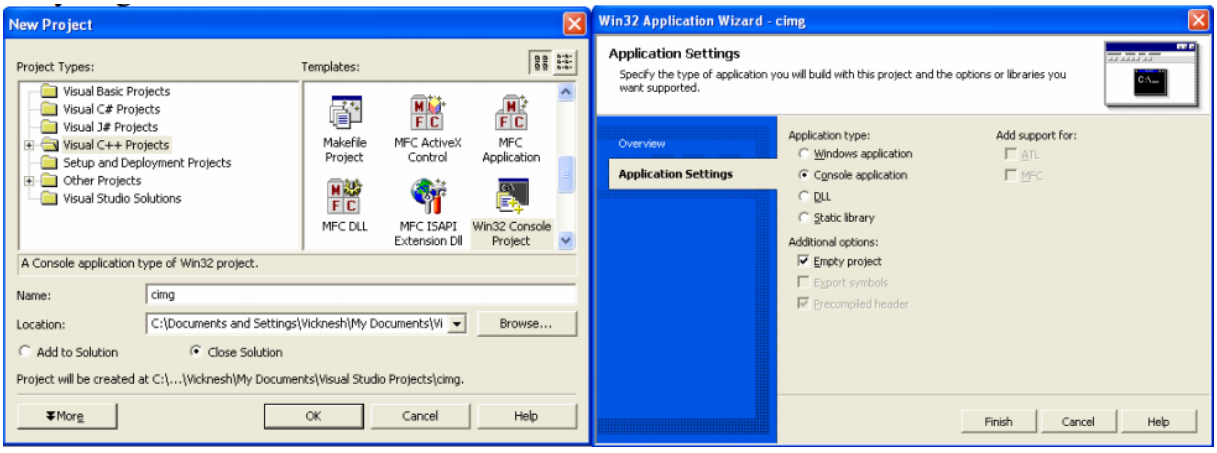
2
CImg БИБЛИОТЕКА

У наредном пројекту је коришћен је додатни алат, Cimg библиотека која је бесплатна, због визуализације и приказивања саме симулације. Cimg библиотека дефинише класе и методе за управљање слика и графике у C++ коду. Може се користити за учитавање и чување различитих формата, вредности приступа пиксела, филтера, цртања, текстова, 3D објеката, компјутерске статистике, управљање корисничким интеракцијама на сликама итд. Cimg дефинише једну класу слике која мозе да има скупове података до 4 димензије (1D скаларни сигнал до 3D слике). Такође може да управља колекцијама слика и секвенцама.

Cimg алат је самосталан, лако преносив и у потпуности ради на различитим оперативним системима (Unix, Windows, MacOS X, *BSD) и компатабилан је са различитим C++ компајлерима (Visual c++, g++, clang++, icc,..) Cimg.h фајл треба бити у истом директоријуму где је и пројектни сорс фајл.[5]

2.1 Отварање пројекта:

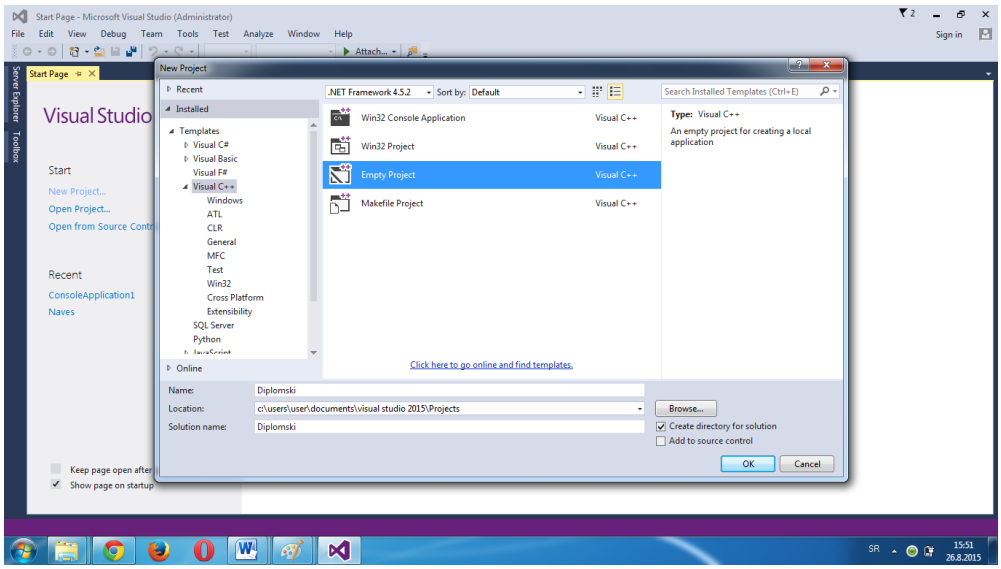
CImg библиотека се билдује на Win32 Console Project . Неопходно је креирати нови пројекат , Visual Studio Почетна страна , Клик на New Project , затим уписати одговарајуће име датотеке . Клик на подешавања апликације, чекирано је „Console Applicaton“, чекирати “Empty Project " затим клик на "Finish". Додат је Cimg.h на хеадер фајл у креиран CPP фајл. Што је приказано на слици 1. [5]



Слика 1.

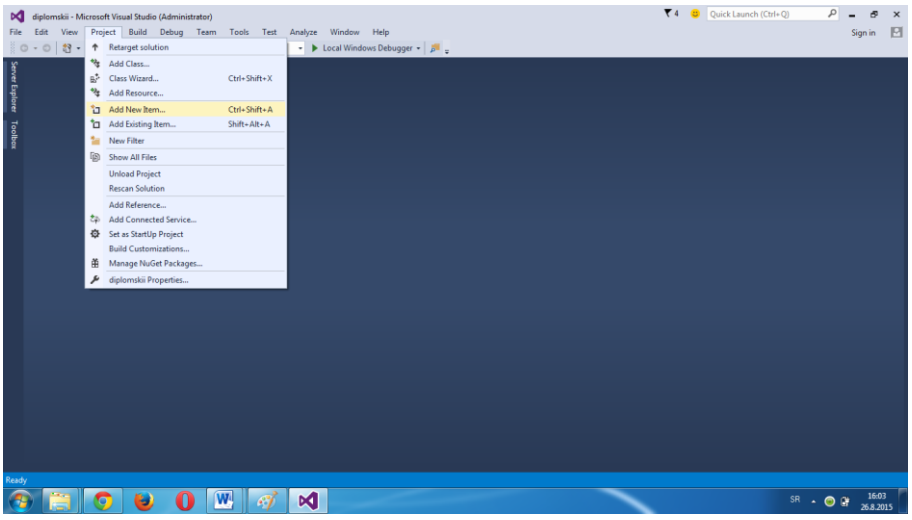
3
ИЗРАДА АПЛИКАЦИЈЕ

Покренут је Visual Studio 2015, кликом на New Project, отвара се нов прозор и одабира се одговарајућа врста пројекта. Empty Project треба одабрати. Затим се врши упис имена апликације. Слика 2.



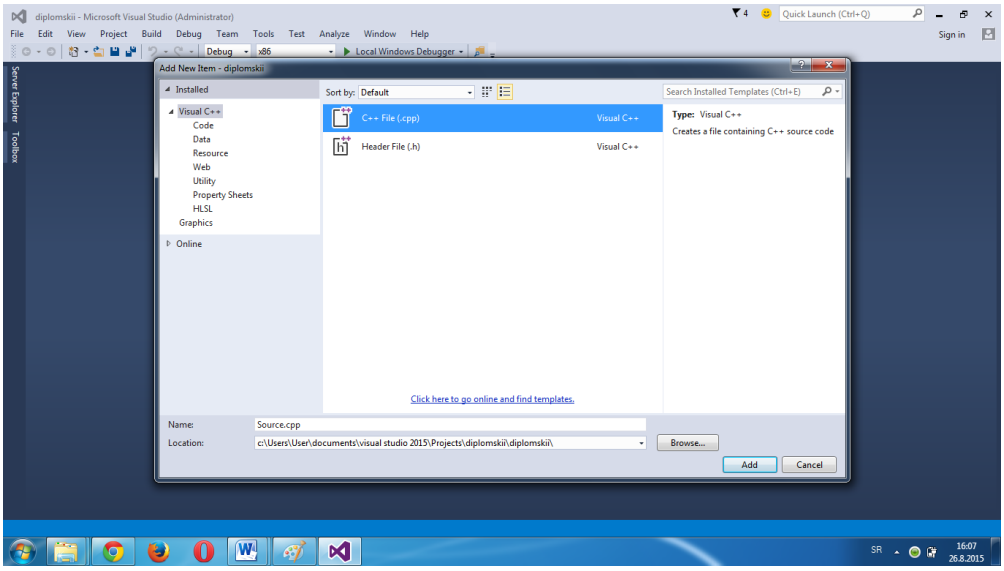
Слика 2.

Затим кликнути на иконицу Project. Изаћиће нам падајући мени и ту кликнути на Add new item.



Слика 3.

Одабрати C++ file (.cpp), затим задати име sourse коду. У овом случају задао сам име „Razbijac“. Затим кликнути на дугме Add и отвара се нова радна површина где можемо почети са кодовима. Слика 4.



Слика 4.

3.1 Код Апликације

Позива се Cimg библиотека „ `#include "CImg.h"“` у којој се дефинису класе и методе, позадина симулиране игрице, дефиниција позиције и облика самог прозора игрице па и позиција лоптице одакле треба кренути са покретањем и положаја цигле као и осталих компоненти.

```
#include "CImg.h"    // позивање библиотеке
using namespace cimg_library;


int main() {
	CImg<double> background("grb.bmp"); // Убацање произвољне позадине у овом случају грб ФИНА

	CImg<unsigned char> board(8,10,1,1,0),
		visu0(background/2),
		visu(visu0),
		brick(16,16,1,1,200), // позиција цигле
		racket(64,8,1,3,0), // позиција плоче
		ball(8,8,1,3,0); // позиција лоптице
	const unsigned char white[3] = { 255,255,255 }, green1[3] = { 60,150,30 }, green2[3] = { 130,255,130 };

	{
		cimg_for_borderXY(brick,x,y,1) brick(x,y) = x>y?255:128;
	}
	{
		cimg_for_insideXY(brick,x,y,1) brick(x,y) = cimg::min(255,64+8*(x+y));
	}
	brick.resize(31,15,1,1,1).resize(32,16,1,1,0);
	ball.draw_circle(4,4,3,white); ball-=ball.get_erode(3)/1.5f; // исцртава лоптицу
	racket.draw_circle(4,3,3.7f,green1).draw_circle(3,2,1.8f,green2); // исцртава плочу
```

```
	{
		cimg_forY(racket,y) racket.draw_rectangle(4,y,racket.dimx()-
7,y,CImg<unsigned char>::vector(y*4,255-y*32,255-y*25).ptr());
	}

// Креирање зида, рекета задате апликације
	racket.draw_image(racket.get_crop(0,0,racket.dimx()/2-1,racket.dimy()-
1).mirror('x'),racket.dimx()/2,0);

	const int w = visu.dimx(),
		h = visu.dimy(),
		w2 = w/2,
		h2 = h/2,
		Wb = ball.dimx(),
		Hb = ball.dimy(),
		polaWb = Wb/2,
		polaHb = Hb/2,
		Rw = racket.dimx(),
		Rh = racket.dimy(),
		polaRw = Rw/2;

	float xr = (float)(w-polaRw),
		oxr = (float)xr,
		Xb = 0,
		Yb = 0,
		staroXb = 0,
		staroYb = 0,
		brzinaXb = 0,
		brzinaYb = 0;
```

Следећи део кода креира дисплеј, и цео прозор који излази приликом клика за старт игрице.

```
CImgDisplay disp(visu,"Diplomski",0); // креирање диспеја
	disp.move((CImgDisplay::screen_dimx()-w)/2,(CImgDisplay::screen_dimy()-h)/2);
	// Креира и приказује број бодова
	for (unsigned int Bodova=0, Ukupno=0; !disp.is_closed && disp.key!=cimg::keyESC
&& disp.key!=cimg::keyQ; )
	{
		if (Ukupno) {
			int X = (int)xr;

			if (disp.mouse_x>=0)
				X = (int)(w2+((disp.mouse_x<0?w2:disp.mouse_x)-w2)*2);
			else
				disp.set_mouse(xr>w2?w-81:80,h2);
```

После целокупног кодирања дисплеја, облика, и осталих компоненти написан је код за ограничавање компонената, да плочица се креће до зида, понашање лоптице приликом ударања у зид.

```
if (X<polaRw) { // да плоча не излази из екрана лево
				X = polaRw; disp.set_mouse(80,h2);
			}
			if (X>=w-polaRw) { // да плоча не излази из екрана десно
				X = w-polaRw-1;
				disp.set_mouse(w-81,h2);
			}

			oxr = xr;
```

```
                        xr = (float)X;
                        staroXb = Xb;
                        staroYb = Yb;
                        Xb+=brzinaXb;
                        Yb+=brzinaYb;
                        if ((Xb>=w-polaWb) || (Xb<polaWb)) {
                                Xb-=brzinaXb;
                                Yb-=brzinaYb;
                                brzinaXb=-brzinaXb;
                                }

if (Yb<polaHb) { // ако лопта удари у плафон, да се одбије
                                Yb = (float)polaHb;
                                brzinaYb=-brzinaYb;
                        }
                        // ако лопта удари у плочу, да се одбије
                        if (Yb>=h-Rh-8-polaHb && Yb<h-8-polaHb && xr-polaRw<=Xb &&
xr+polaRw>=Xb) {
                                Xb = staroXb;
                                Yb = h-Rh-8.0f-polaHb;
                                brzinaYb=-brzinaYb;
                                brzinaXb+=(xr-oxr)/4;

                                if (cimg::abs(brzinaXb)>8)
                                        brzinaXb*=8/cimg::abs(brzinaXb);
                        }
                        if (Yb<board.dimy()*16) {
                                        const int X = (int)Xb/32, Y = (int)Yb/16;

// ако лопта удари у циглу, цигла нестаје а додаје се један бод за сваку циглу
                                if (board(X,Y)) {
                                        Bodova++;
                                        board(X,Y) = 0;
                                        const unsigned int x0 = X*brick.dimx(),
                                                y0 = Y*brick.dimy(),
                                                x1 = (X+1)*brick.dimx()-1,
                                                y1 = (Y+1)*brick.dimy()-1;


        visu0.draw_image(background.get_crop(x0,y0,x1,y1),x0,y0);

                                if (staroXb<(X<<5) || staroXb>=((X+1)<<5))
                                        brzinaXb=-brzinaXb;
                                else if (staroYb<(Y<<4) || staroYb>=((Y+1)<<4))
                                        brzinaYb=-brzinaYb;
                        }}

                disp.set_title("Score : %u/%u", Bodova, Ukupno); // исписује број бодова
                        }
```

Код који показује шта када лоптица падне на под, у том случају крај игре, а уколико све цигле се разбију онда је постигнут циљ игре. Код за крај игре или победу је следећи:

```
                //Ако је крај или почетак игре
                if (Yb>h || Bodova==Ukupno) {
                        disp.show_mouse();
                        while (!disp.key && !disp.is_closed && !disp.button) {
```

```
                        ((visu=visu0)/=2).draw_text(Ukupno ? ((Bodova != Ukupno)
?"Game Over" : "You Win") :"Get Ready ?",50,visu.dimy()/2-10,white,0,25).
                        display(disp);
                    disp.wait();
        if (disp.is_resized) disp.resize(disp); // у случају мењања величине дисплеја
        }
```

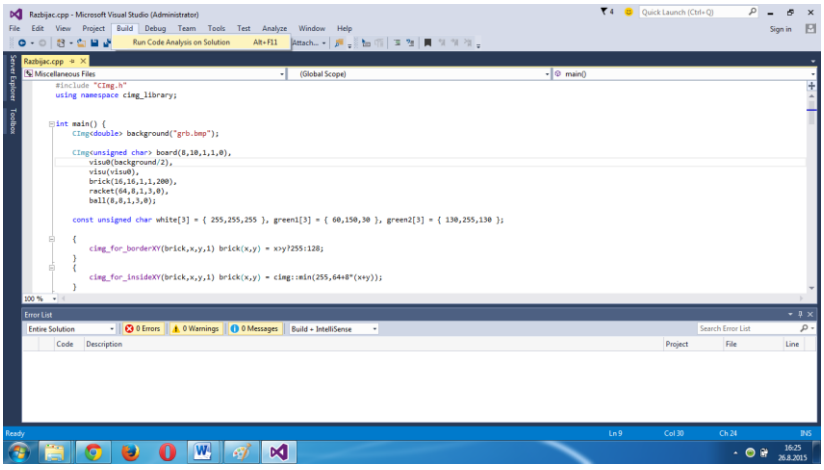
Када дође до краја игре, написана је форма за насумично попуњавање или сместање цигли на панел при сваком новом почетку игрице.

```
board.fill(0); visu0 = background;
                    cimg_forXY(board,x,y) if (0.2f+cimg::crand()>=0) {
                        CImg<> cbrick =
CImg<double>::vector(100+cimg::rand()*155,100+cimg::rand()*155,100+cimg::rand()*155).
                                unroll('v').resize(brick.dimx(),brick.dimy());
                        cimg_forV(cbrick,k)
(cbrick.get_shared_channel(k).mul(brick))/=255;
                        visu0.draw_image(cbrick,x*32,y*16);
                        board(x,y) = 1;
                    }
                    Ukupno = (int)board.sum();
                    Bodova = 0;
                    staroXb = Xb = (float)w2;
                    staroYb = Yb = board.dimy()*16.0f+Hb;
                    brzinaXb = 2.0f;
                    brzinaYb = 3.0f;
                    disp.hide_mouse();
            } else disp.display((visu=visu0).draw_image(racket,(int)(xr-polaRw),h-Rh-
8).draw_image(ball,(int)(Xb-polaWb),(int)(Yb-polaHb)));
            if (disp.wait(15).is_resized) disp.resize(disp);
        }

  return 0;
        }
```

3.2 Build апликације

После израде кода апликације, мора се сачувати код. Затим клик на дугме Build (Слика 5). Уколико нема грешака иде се на опцију Debug а затим Start Debug. Тако покрећемо наш куцани програм и прави се иконица „Diplomski.exe“ за покретање апликације.



Слика 5.

4
ТЕСТИРАЊЕ И ВАЛИДАЦИЈА АПЛИКАЦИЈЕ

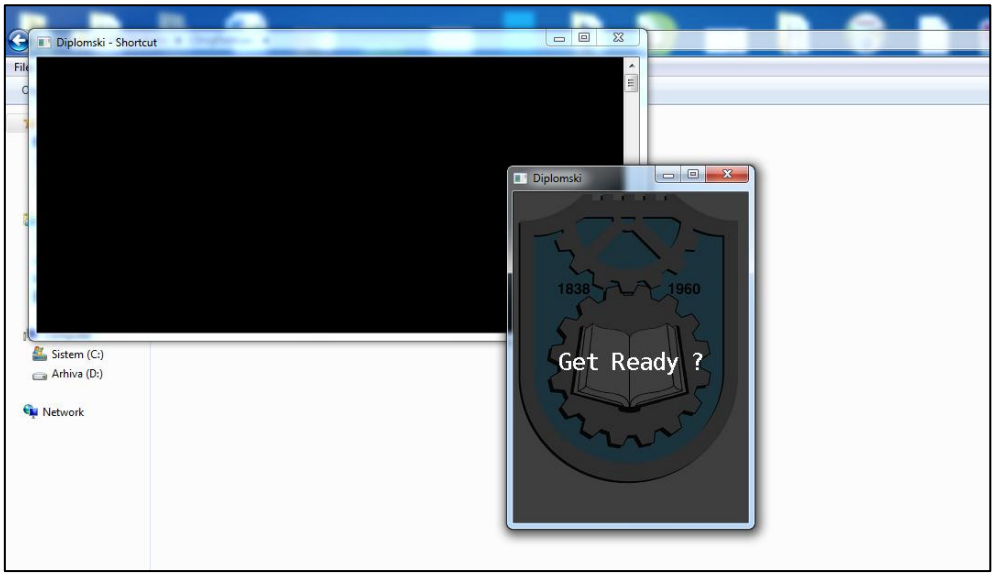
Сваки програмски производ па и ова симулација игрице која разбија цигле треба да прође и кроз фазу тестирања и валидације. Намена тестирања је да покаже да апликација ради онозаште је намењена и да утврди апликационе дефекте пре него што се стави у употребу. Када се тестира програмирани производ, тада се извршава тест, где се тестирање врши вештачким подацима. Тестирање је генералнији део свих процеса валидације, који укључују и статичке технике валидације. Резултати теста треба да покажу грешке, аномалије или друге информације о апликацији које указују на непредвиђењо понашање. [6]

Апликација се покреће коришћењем тест података где се помно прати њено понашање и анализирају сви детаљи који могу да покажу одређено неприкладно понашање, и на крају теста систем треба да покаже да може да обезбеди одређене функционалности, перформансе и да је стабилан у току нормалне употребе.

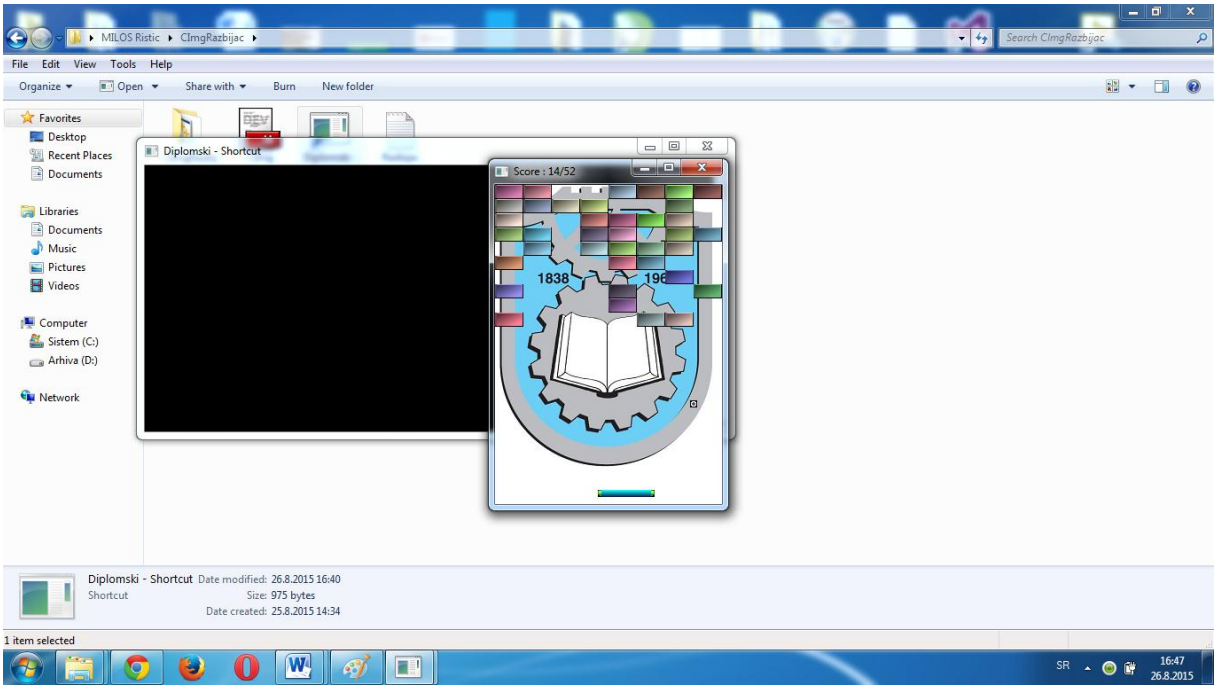
Тестирање апликације у овом случају би требало да открије грешке или отказе, и прикаже да ли постоји неко неприкладно понашање које није дефинисано спецификацијом.Постоје неколико врста тестирања:

- Развојно тестирање-тестира се у току развоја производа да би се открили багови или други дефекти**.** Врсте развојног тестирања:
  - Тестирање јединица, тестирају се индивидуалне програмске јединице и класе.
  - Тестирање система, систем се тестира као целина.
  - Тестирање компоненти, неколико посебних јединица се интегришу како би креирале сложене компоненте. Фокус тестирања је тестирање итерфејса.
- Тестирање производа-одређен тим тестира комплетну верзију апликације пре него што се она испоручи у одређеном окружењу за тестирање.
- Корисничко тестирање-корисници, или потенцијални корисници апликације тестирају производ, и овај начин је најбитнији, разлог је тај што корисниково радно окружење има велике ефекте на поузданост, перформансе, употребљивост, итд., а ово се не може рефлектовати у окружењу за тестирање. Врсте корисничког тестирања:
  - Алфа тестирање, корисници раде са развојним тимом да би тестирали софтвер на развојним страницама.
  - Бета тестирање, корисницима се испоручује производ како би могли да експериментишу са њим и установе проблеме које развојни тим није пронашао.
  - Тестирање прихватљивости, корисници тестирају систем да би одлучили да ли је спреман да буде прихваћен у корисничко окружење за употребу. [6]

На самом почетку тестирања прво отворити фаил где је сачуван целокупан рад. Ту се налази иконица „Diplomski.exe“. Двокликом на њу покрецемо нашу апликацију „Razbijac“. Отвара се прозор са почетком игрице. Слика 6.

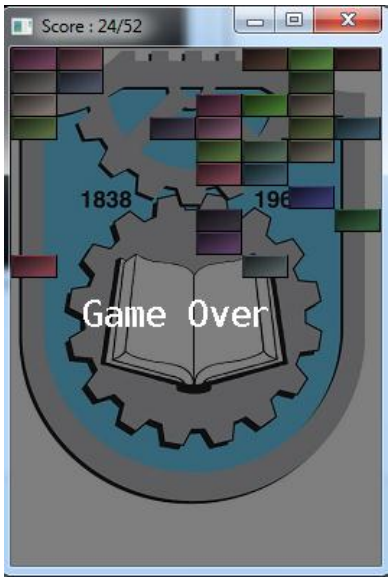


Слика 6.



Слика 7.Тестирање игре, померањем миша и разбијањем цигли

Пропуштањем лоптице да падне на под ћемо проверити крај игре у случају када нису све цигле разбијене и да ли нас обавештава да је крај игре. Слика 8. А разбијањем свих цигли достиже се циљ игре. (Слика 9)



Слика 8.

Слика 9.

ЗАКЉУЧАК

Програмирање је и компликовано и једноставно-одвија се у глави и зависи највише од тога да ли знамо шта желимо добити на крају. Уз познавање програмских језика, неопходна је и логика, да би то што желимо спровели у дело, тј. да би могли то да представимо преко кода.

C++ програмски језик је најбоље решење када је потребан брз развој desktop апликација за Windows оперативни систем. Брзина имплементације апликација, добра техничка документација као и распрострањеност Windows оперативног система, чине C++ можда и најпопуларнијим програмским језиком данашњице.

Комбиновањем постојећих алатки као на пример Cimg библиотека, и логиком, добијен је код који савршено функционише, и испуњава захтеве крајњем кориснику, а највише млађим корисницима.

ЛИТЕРАТУРА

[1]http://sr.wikipedia.org/sr-ec/C

[2] https://sr.wikipedia.org/sr/Програмски_језик

[3] http://www.it-modul.rs/07/2012/uvod-u-objektno-orijentisano-programiranje/

[4] др Ирина Брановић , *Објектно орјентисано програмирање c++*, Универзитет Сингидунум, Београд 2011.

[5] Vicknesh Selvam, *An Introduction to the CImg Library.*

[6] др Зоран Јеремић, *Тестирање софтвера и управљање квалитетом*.