## Ղեկավար՝ Ա. Դարբասյան

## Խումբ՝ Հարությունյան Ռուբեն, Մարտիրոսյան Տիգրան

# Գիտահետազոտական աշխատանք

## Խնդիր

Ներկայիս տեխնոլոգիապես գերզարգացած աշխարհում, մարդիկ միշտ իրենց հետ են փորձում պահել իրենց ամբողջ ինֆորմացիան, առաջին անհրաժեշտության ծրագրային ապահովումը, ցանցային ծրագրերից օգտվելու հնարավորությունը: Այսպես կոչվող <<խելացի հեռախոսները>> մեծ մասամբ տալիս են այդ հնարավորությունը, բացի այդ դառնում են 21-րդ դարի ժամանցի հիմնական միջոցներից մեկը: Հեռախոսները և պլանշետները տալիս են մարդուն իր ընկերների հետ հախորդակցվելու, կարևոր տեղեկություններով կիսվելու, ժամանակ անցկացնելու մեծ հնարավորություններ:

Համակարգչային խաղերը` արդեն 30 տարի է, ինչ հայտնի են իրենց պահանջվածությամբ ցանկացած երկրի ծրագրային շուկայում, ինչի շնորհիվ նրանք շատ կարճ ժամանակահատվածում հայտնվեցին նաև հեռախոսային պլատֆորմների համար:

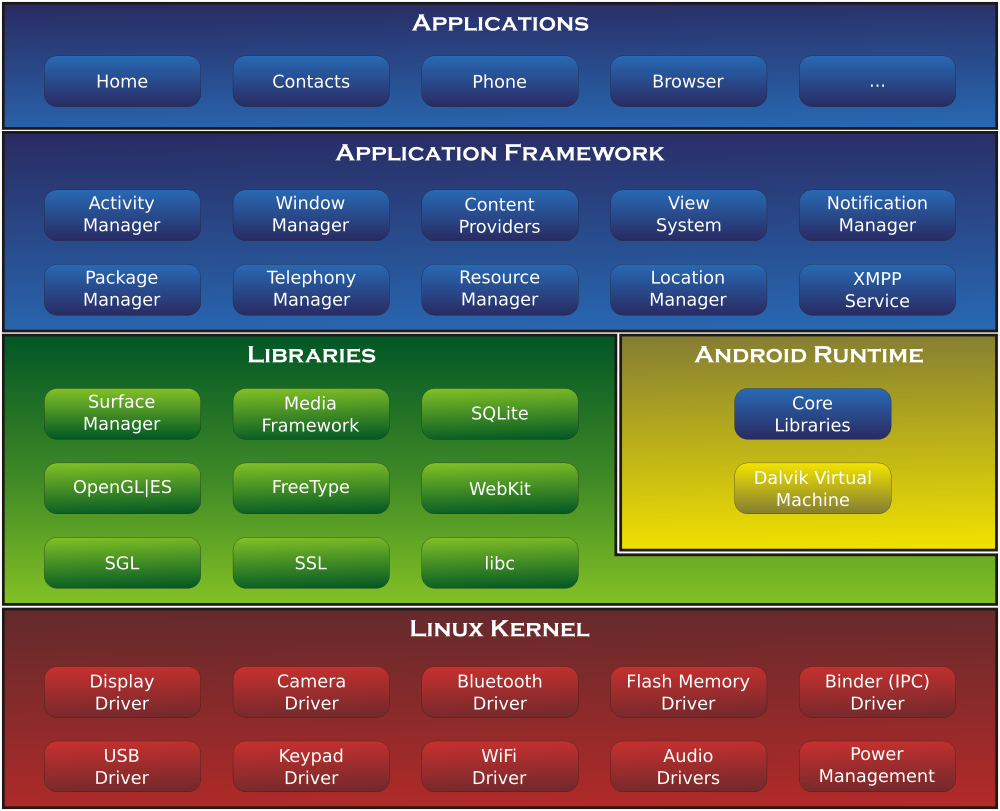
Սակայն պետք չէ թերագնահատել անձնական համակարգիչների պլատֆորմը, քանի որ մոտ 20 տարի է, ինչ խաղերի ճնշող մեծամասնությունը գրվել է Windows/Mac OS օպերացիոն համակարգերի համար: Բացի մեծ կորպորացիոն կազմակերպություններից, ինչպիսիք են՝ EA Games-ը, RockStar North-ը, Nintendo-ն, խաղերի արտադրությամբ են զբաղվում մեծ քանակությամբ անկախ ծրագրավորողներ: Արաջացել են նաև խաղերի online շուկաներ, որոնց վառ օրինակ են՝ Steam-ը և GOG.com-ը: Valve-ն (Steam-ի ստեղծող կազմակերպությունը) նաև ստեղծել է օպերացիոն համակարգ հատուկ համակարգչային խաղերի համար՝ Steam OS անունով (Debian (Linux Kernel)-ի հիմքով):



Այս, և բազմաթիվ այլ պատճառներով, որոշվեց ընտրել որպես գիտահետազոտական աշխատանքի խնդիր՝ բազմապլատֆորմ խաղի ստեղծումը, որը պետք է Windows-ից տեղափոխվեր Android (սմարթֆոնների շուկայում ամենատարածվածը լինելու պատճառով սկզբնական թիրախը ընտրվեց Android-ը):

## Լուծումը

Հաշվի առնելով որ Android-ը հիմնական թիրախն էր, սկսեցինք ուսումնասիրել Android Application-ների ստեղծման գործընթացը: Քանի որ Android-ը Linux Kernel-ի հիմքով օպերացիոն համակարգ է, ապա այն ունի հիմնական C/C++ գրադարանները իր մեջ, սակայն այդ փաթեթը չի կարելի համեմատել օրիգինալ GNU Linux-ի հիմքով օպերացիոն համակարգերի գրադարանների հետ: Android Application-ների հիմքում ընկած է Java-ն, որի Runtime տվյալ դեպքում հանդիսանում է Dalvik Virtual machine-ը իր հիմքային գրադարանների հետ միասին, որոնց միջոցով ստեղծվել է Android Application Framework-ը:



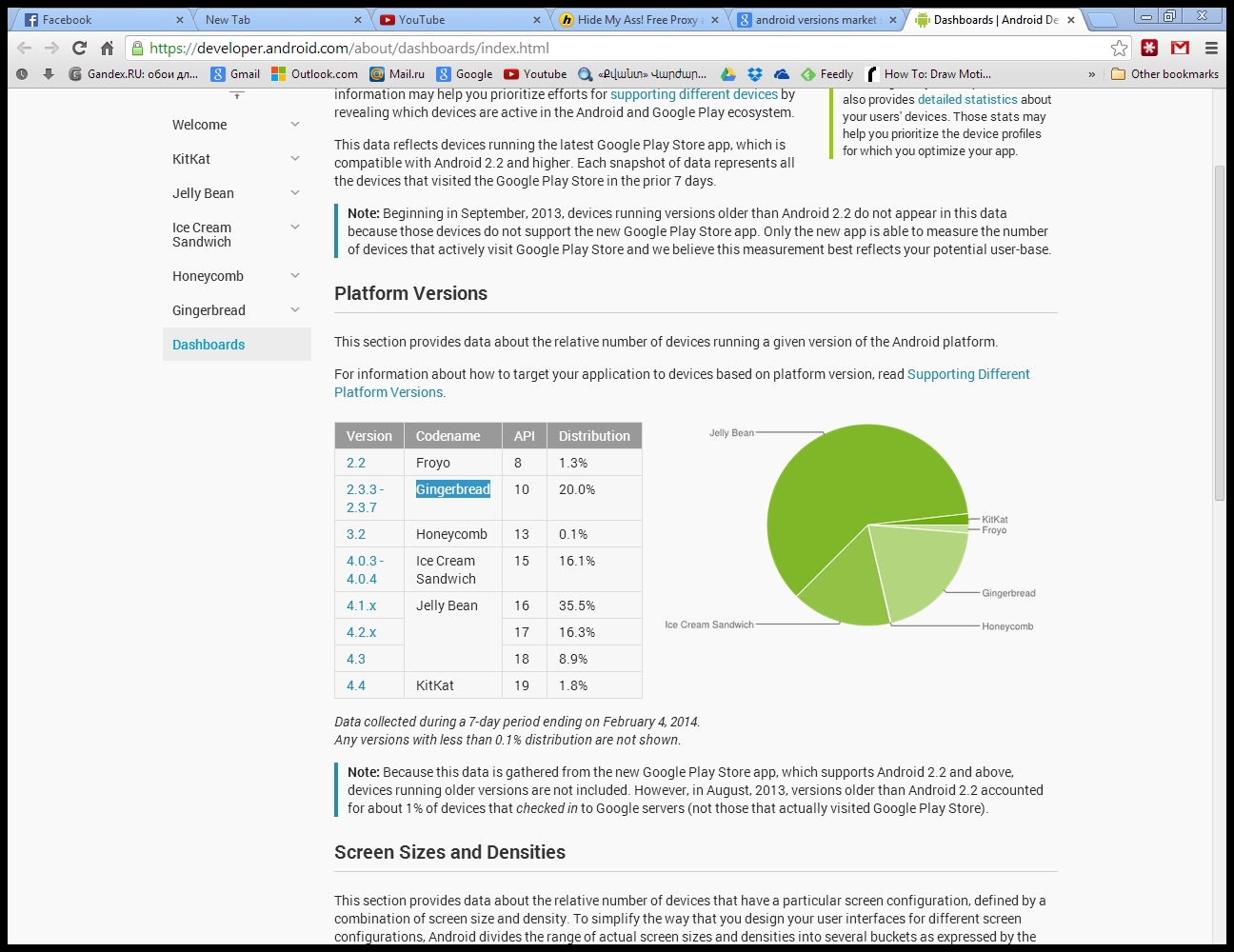
Հետևանքում սովորական Android պրոյեկտը ստեղծվում է Android Studio/Eclipse (Android Plugin-ով) ինտեգրացված ծրագրավորման միջավայրերում: Երկու դեպքերում ել վերջնական .apk ֆայլի ստեղծումը լինում է ծրագիրը բնութագրող .xml և Java-ի .class ֆայլերից Ant-Build-ի միջոցով: Բացի այդ` տարբերվում են Android-ի տարբերակից կախված առաջարկվող հնարավորությունների փաթեթները:

Windows-ում մինիմալ փորձարկումներից հետո պարզվեց, որ Java-ի գրադարանների միջոցով գրաֆիկայի պատկերումը շատ դանդաղ է (15-20 FPS, 4 պատկեր): Միակ լուծումը տվյալ խնդրին գրաֆիկական քարտի հնարավորությունների օգտագործումն է, ավելի ցածր մակարդակի ծրագրավորման ինտերֆեյսի միջոցով: Android-ում միակ հնարավոր տարբերակը OpenGL ES 1.0/2.0-ն է: OpenGL-ը կարելի է օգտագործել՝ կամ Java-ի և Android-ի գրադարանների միջոցով (մասնավորապես ՞import android.opengl՞-ի միջոցով), կամ Android NDK-ի միջոցով (C/C++ ծրագրավորման լեզուներով):

Առաջին տարբերակի դեպքում առաջանում էր շատ մեծ տարբերություն Windows և Android տարբերակների միջև, քանի որ չկար երկու պլատֆորմների համար ընդհանուր գրադարան: Բնականաբար կարելի էր Android Emulator-ի միջոցով ստուգել Android տարբերակը, տեղում ուղղումներ կատարելով, սակայն փորձը ցույց տվեց, որ Android Emulator-ի միանալու միջին ժամանակը 30 րոպեն էր, այն ել 5-7 FPS արագությամբ, ինչը այդքան ել հարմար չէ ծրագրի թեստավորման համար:

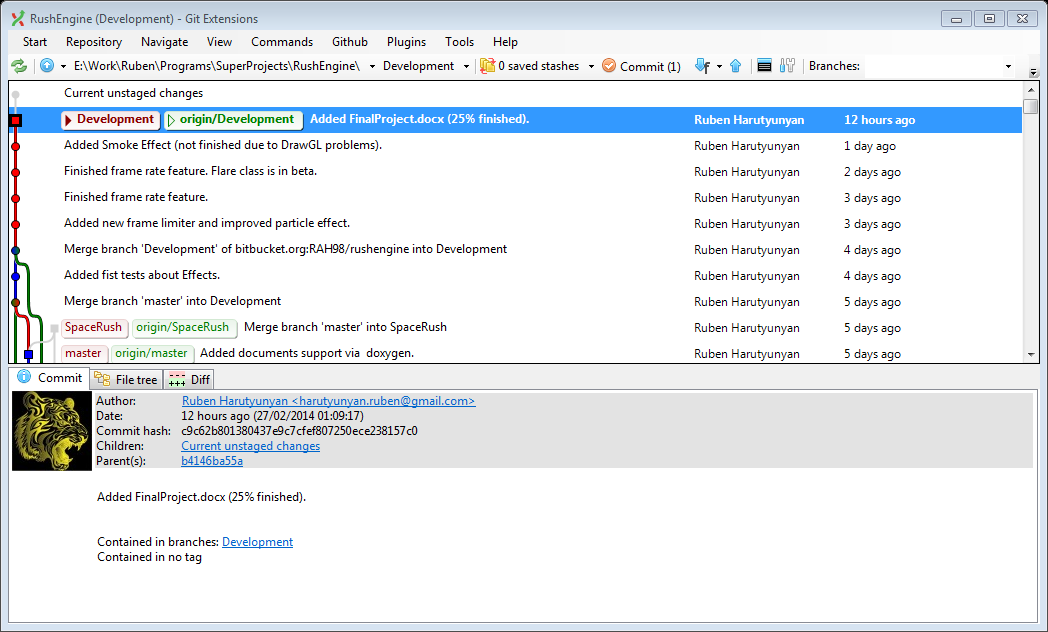
Մյուս տարբերակը տալիս էր հնարավորություն գրել կոդ, որը մի քանի պրեպրոցեսսորային դիրեկտիվների միջոցով կկարգավորեր տարբերությունները Windows/Mac OSX/Linux/Android/iOS համակարգերի միջև, քանի որ բոլորը ունեն C/C++ ծրագրավորման ինտերֆեյս (բացի Windows Phone-ից, որը թույլատրում է միայն C# .NET ծրագրեր, բայց մենք այդքան էլ հետաքրքրված չէինք նրանով): Բացի այդ, C/C++ տարբերակը անհամեմատ արագ է Java տարբերակից: Ստացվում է, որ մենք օգտագործելով Android NDK (Native Development Kit), որը աշխատում է Java JNI-ի միջոցով, ազատվում ենք պլատֆորմների տարբերությունների կախվածությունից և կստանանք անհամեմատ ավելի լավ ծրագրի արագություն: Սակայն առաջանում էր խնդիր աշխատանքի ծավալի հետ կապված, քանի որ ավելանում էր .png նկարների ներբեռնման, ձայնային էֆֆեկտների, ժամանակի չափման, ֆոնտերի ներբեռնման և ֆիզիկական չափումների կատարման պլատֆոորմից կախված ծրագրերի ստեղծումը: Նկարների ներբեռնման, ձայնային էֆֆեկտների, ժամանակի չափման, ֆոնտերի ներբեռնման համար օգտագործեցինք SDL 2.0 գրադարանը, որը չնայած իր պարզունակությանը, ընդանուր առմամբ բավարարում էր մեր պահանջները: Ֆիզկական միջավայրի սիմուլացիայի համար օգտագործվեց Box2D, որը իրեն լավ էր ցուցաբերել դեռ Angry Birds խաղերի շարքում: Box2D-ն հիմնականում գրված է եղել Desktop պլատֆորմների համար, սակայն իր C++-ի օգտագործման, և մինիմալ cstdlib-ից կախվածության պատճառով, նրա տեղափոխումը Android NDK-ի պլատֆորմ խնդիր չէ:

Երկրորդ կարևոր խնդիրը Android-ի տարբերակի ընտրությունն էր, ինչից կախված որոշվում էին մեր հնարավորությունները: Հաշվի առնելով Android-ի տարբերակների տարածվածության մասին տեղեկությունները, մինիմալ ամենատարածվաց տարբերակի Android 2.3.3 (Gingerbread)-ն է. որը սկսած 2.3 տարբերակից թույլ է տալիս ստեղծել ծրագրեր միայն Android NDK-ի միջոցով, առանց Java-ի օգտագործման (բացառությամբ Application Avtivity Java class-ի, որով կզբաղվի SDL-ը):



Երրորդ կարևոր խնդիրը վերաբերվում էր ծրագրավորման միջավայրի ստեղծման մեջ: Քանի որ թիմը երկու հոգուց է կազմված, ապա կարիք առաջացավ օգտագործելու կենտրոնական սերվեր՝ կոդը և այլ ֆայլերը պահելու համար, բացի այդ պահել փոփոխությունների ցուցակ, ցանկալի է Desktop client-ի հնարավորությամբ, և պարտադիր offline աշխատելու հնարավորությամբ: Ամենահարմար ծրագրային կոդի կառավարման համակարգը այս դեպքում Git-ն էր, որը ստեղծվել է Linux-ի թիմի կողմից և հենց Git-ն էլ օգտագործվել է Linux-ի ծրագրավորման համար: Git սերվերների առաջարկությունը բավականին մեծ է, սակայն կոդը ազատ այլ մարդկանց տրամադրելու պայմանով (անվճար տարբերակում): Բացառություն էր կազմում BitBucket.org-ը, որը տրամադրում է անսահմանափակ քանակությամբ և ծավալով անձնական պրոյեկտներ, մինչև 5 անդամի սահմանափակումով, ինչից էլ մենք օգտվեցինք: Որպես Git Desktop Client օգտագործվեց GitExtensions-ը:

Կոդի անալիզի և վիզուալիզացիայի համար օգտագործել ենք Doxygen/Visual Studio Dependency Graph գործիքները:

  
C/C++ կոդ գրելու համար մենք բնականաբար օգտագործեցինք Visual Studio 2010/2013 Ultimate, որը հնարավորություն տվեց ստուգել կոդի արագությունը և գտնել թույլ տեղերը, իսկ Visual Studio Debugger-ը միշտ համարվել է ամենալավը խաղային պրոյեկտների համար:

Մենք օգտագործեցինք C++11-ի որոշ հնարավորություններ (օրինակ՝ std::to\_string ֆունկցիան), որոնք, ի դեպ, հասանելի են Visual Studio 2010 (VC++10) և բարձր տարբերակներում, g++ 4.8.1-ում և ավելի բարձր տարբերակներում (Android NDK-ի համար):

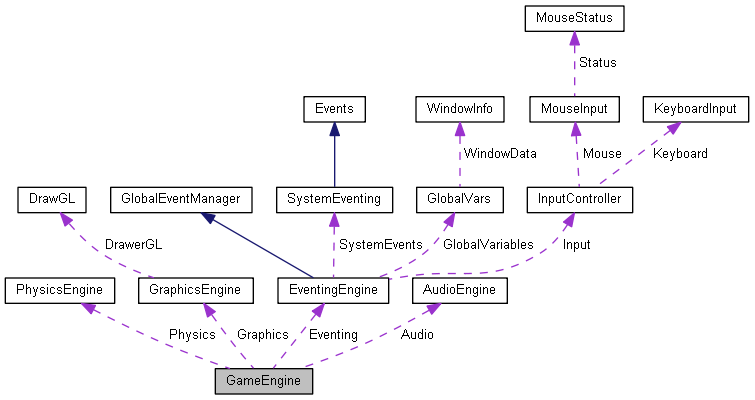
Չորրորդ խնդիրը հենց ծրագրի ստեղծումն էր: Կարևորը սկզբնական framework-ի ստեղծումն էր, որը հիմնվելով C++, SDL, Box2D, OpenGL 1.0/ES 1.0 հնարավորությունների վրա, տրամադրեր հարմար ֆունկցիաներ խաղի տրամաբանությունը ստեղծելու համար:

Այդպես ստեղծվեց RushEngine պրոյեկտը, որը իրենից ներկայացնում է C++ Static Library Project, և ում ներառելով հիմնական խաղի պրոյեկտի մեջ կարելի է հեշտությամբ ստեղծել արդեն հիմնական խաղը:

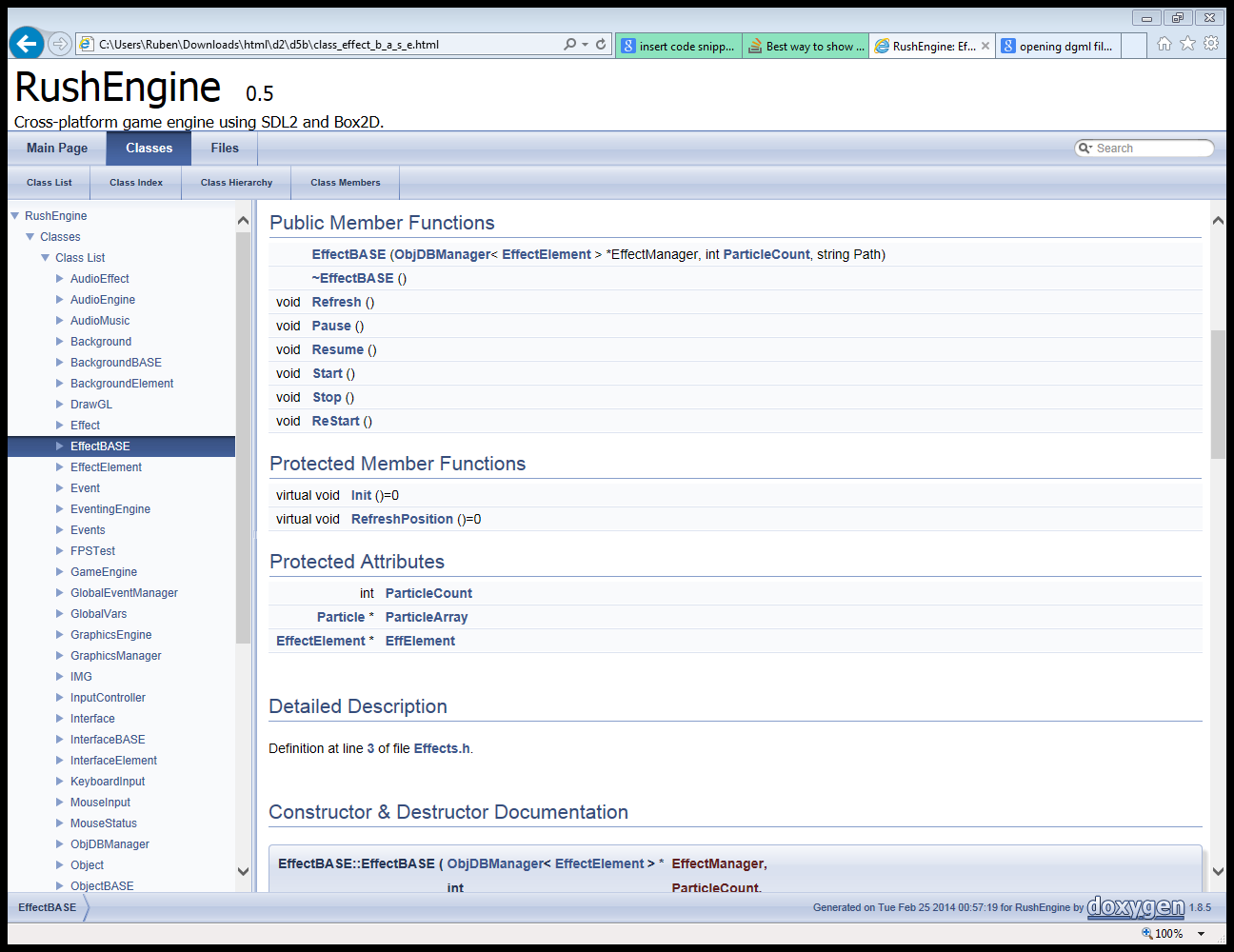
Սա GameEngine.h ֆայլի պարունակությունն է, որը միանգամից ներառում է խաղի պրոյեկտի մեջ բոլոր պահանջված գրադարանները (SDL 2.0 գրադարանը բաժանված է փոքր հատվածների, ինչի պատճառով էլ ներառվում են բազմաթիվ .lib ֆայլեր):

RushEngine-ը ստեղծվել է առավելագույն կերպով մոդուլար լինելու, մոդուլների միջև հաղորդակցությունը պահպանելու, առավելագույն հնարավոր ծրագրի արագությունը ապահովելու համար և վերջնական օգտագործողին հարմար լինելու համար:

Այդ նպատակով ստեղծվել է մի GameEngine class որը իր մեջ պարունակում է ՝ GraphicsEngine (գրաֆիկական գործողությունների և պատուհանների մենեջմենտի համար), PhysicsEngine (ֆիզիկական հաշվարկների կարգավորման համար), ObjectsEngine (չնայած ավելի շատ ներքին RushEngine-ի կարգավորման համար նախատեսված լինելուն, կարող է օգտագործվել վերջնական պրոյեկտներում, հիմնական կիրառությունը ֆիզիկայի, ինտերֆեյսի, գրաֆիկական էֆֆեկտների և գրաֆիկայի միջև արագ ինֆորմացիայի փոխանցումն է և պահպանումը), EventingEngine (կարգավորում է նեքին իրադարձությունների և մոդուլների միջև հաղորդակցությունը, նաև ստանում է օպերացիոն համակարգից տեղեկություններ, այդ թվում՝ mouse-ի դիրքը էկրանին, կոճակների սեղմումների մասին տեղեկությունները), AudioEngine (զբաղվում է աուդիո էֆֆեկտների կարգավորմամբ) class-երը:

C++ compilation-ի ժամանակ class redefinition, function redefinition, և այլ սխալներից խուսափելու համար օգտագործել է “#pragma once” պրեպրոցեսսորակին դիրեկտիվը, որը, չնայած իր ոչ պաշտոնական լինելուն, առկա է բոլոր նոր C++ compiler-ներում, և ի տարբերություն կլասիկ “#ifndef \_\_FILENAME\_\_ #define \_\_FILENAME\_\_ #endif” include guard-երի, ավելի արագ է:

Մյուս հարցը կայանում էր շատ էֆֆեկտիվ API տրամադրելու մեջ: Այդ նապատակով օգտագործվեց C++-ի polymorphism-ը, ընդ որում, այս մեթոդը օգտագործվում է նաև Microsoft Foundation Classes-ի հիմքով C++ պրոյեկտներում: Գոյություն ունեն Effect, Object, Interface, Bsckground class-եր:

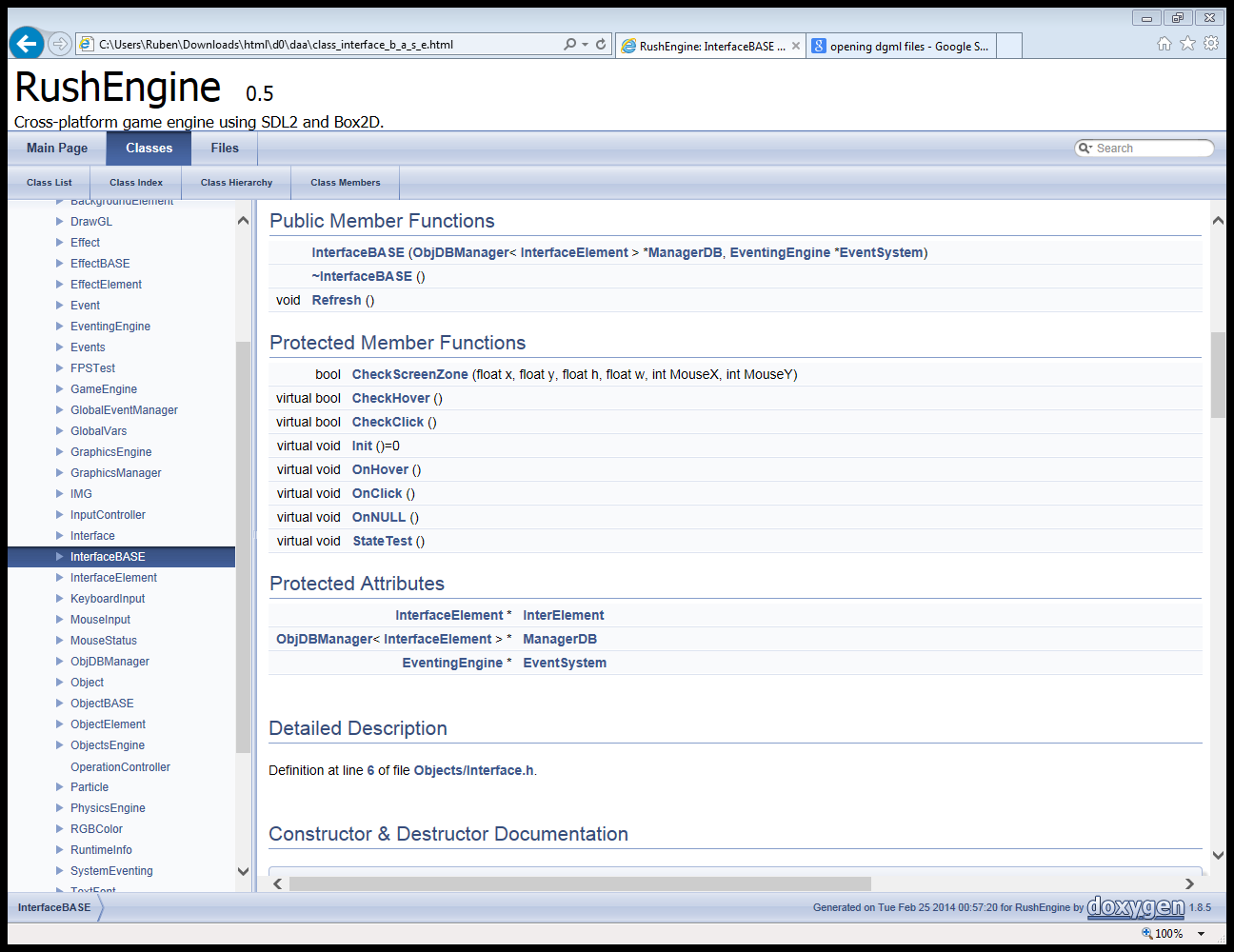


Վերևի նկարից երևում է, որ Effect class-ը ունի գռաֆիկական էֆֆեկտի ցուցադրվելու համար անհրաժեշտ Refresh(), Pause(), Resume(), Start(), Stop(), ReStart() ֆունկցիաները, բացի այդ Init() և RefreshPosition() pure virtual ֆունկցիաներ, որոնք պետք է վերաբեռնվեն օգտագործողի կողմից: Init()-ը՝ կանչվելու է class-ի constructor-ի կողմից, RefreshPosition()-ը՝ ամեն անգամ նոր frame նկարելուց (ֆիքսված հաճախականությամբ, հիմնականում 120 FPS):

Օրինակ՝  


(Բնականաբար վերևում կարող են հանդիպել կոդի մասեր որոնց մասին չի նշվել, սակայն քանի որ այս էֆֆեկտի կոդը մոտ 200 տողից է կազմված, ամբողջովին copy/paste անելը այդքան էլ հարմար չէ)

Նույնը վերաբերվում է Interface class –ին, որի միջոցով կարելի է պատկերել էկրանին տարբեր տեսակների կոճակներ, input box-եր, combo box-եր, և այլ input էլէմենտներ: CheckHover() և CheckClick() ֆունկցիաներով կարելի է ստուգել mouse-ի մասին ինֆորմացիան, և դրանից ելնելով տարբեր գործողություններ կատարել: Ինչպես միշտ, պետք է լրացնել Init() ֆունկցիան, իսկ Refresh() ֆունկցիան կանչվելու է գրաֆիկայի կողմից՝ ֆիքսված հաճախականությամբ:

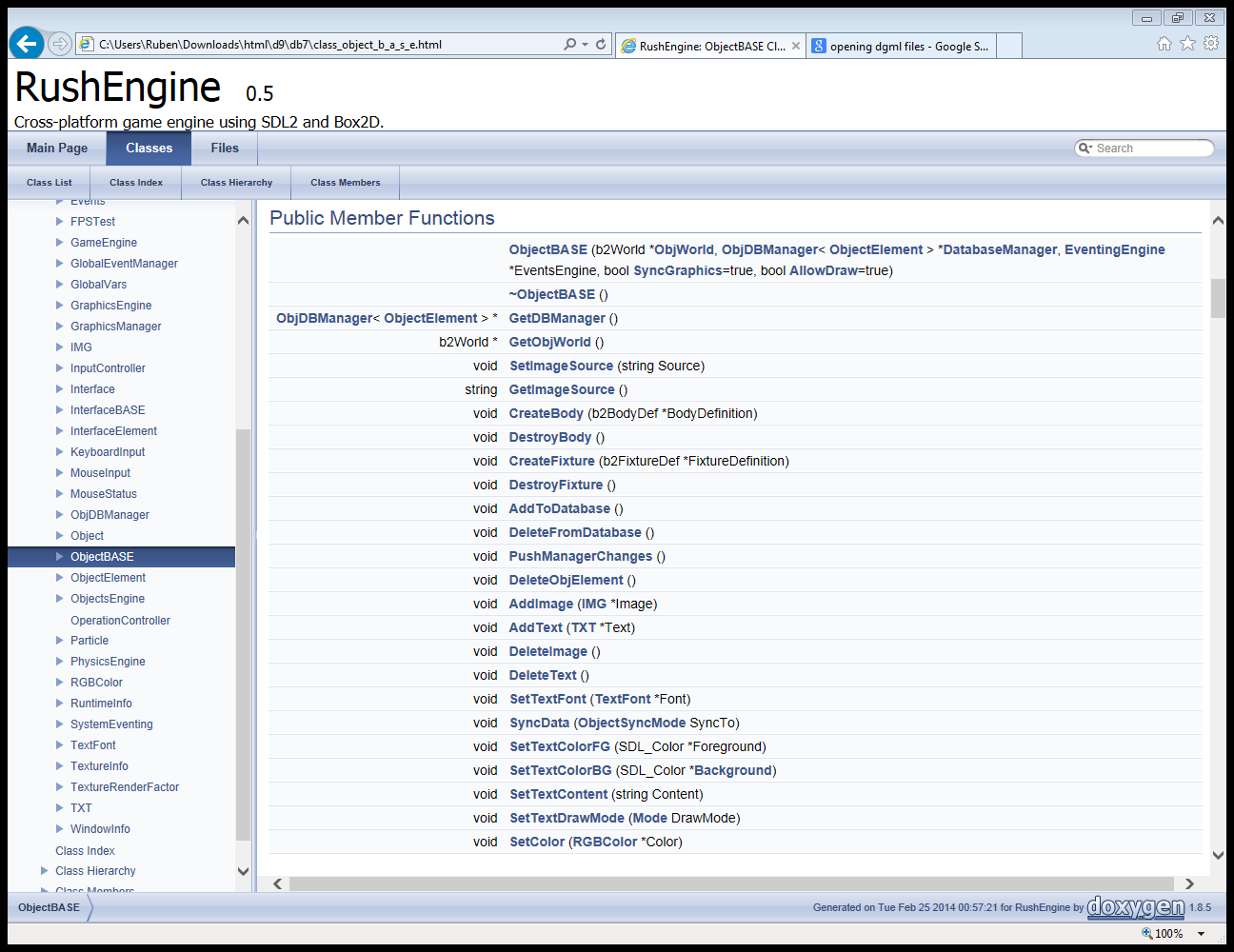


Օրինակ՝



Object class-ը օգտագործում է Box2D-ի ֆիզիկական հաշվարկները, հանդիսանում է RushEngine-ի և Box2D-ի միացման կետը, իր մեջ պարունակում է Box2D-ի փոքրագույն մասնիկ b2Body, b2Fixture class-երի փոփոխականները:

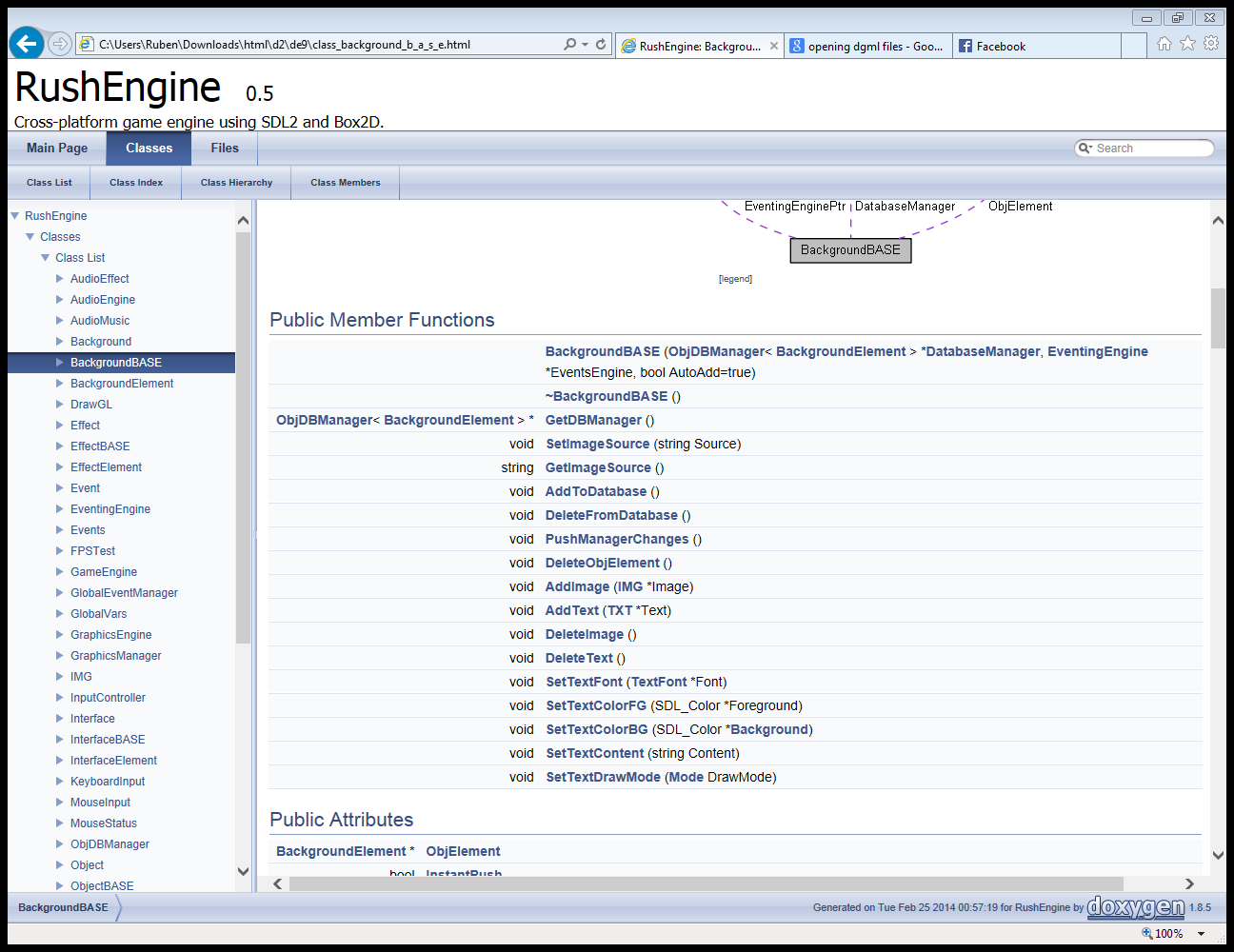
Ի տարբերություն Effect և Interface class-երի, Object class-ը չի պահանջում ժառանգ, այն արդեն պատրաստ է փոփոխականների և ֆունկցիաների միջոցով կատարել բոլոր պահանջվող գործողությունները: Գրաֆիկայի կողմից կանչերի ժամանակ այն ուղղակիորեն փոխանցում է b2Body-ի ինֆորմացիան, ցանկացած թարմացում կատարվում է Box2D-ի կողմից՝ b2World.Step() ֆունկցիայի միջոցով:



Օրինակ՝



Background class-ը չի պահանջում ժառանգ, այն արդեն պատրաստ է փոփոխականների և ֆունկցիաների միջոցով կատարել բոլոր պահանջվող գործողությունները: Գրաֆիկայի կողմից կանչերի ժամանակ այն ուղղակիորեն փոխանցում է մի IMG class-ի pointer, որը իր մեջ պարունակում է հետին պլանի նկարի մասին տեղեկությունները, չի պահանջում թարմացումներ:



Օրինակ՝



Այս ամբողջ համակարգը աշխատեցնելու համար պահանջվում է GameEngine class-ի փոփոխականի ստեղծում: Բացի այդ պետք է կանչել որոշ, համակարգի աշխատանքը կարգավորող, ֆունկցիաներ:



Գրաֆիկան կազմված է յուրաքանչյուր տեսակի կլասսերի, մարդու կողմից որոշվող, ծրագրի ամբողջ աշխատանքի ընթացքում հաստատուն թվով շերտերից, որոնք պատկերվում են ըստ՝

1. Background

Layer1

…

LayerN

1. Object

Layer1

…

LayerN

1. Effect

Layer1

…

LayerN

1. Interface

Layer1

…

LayerN

հերթականության: Հետևանքում միշտ երևում են Interface class-ի ներկայացուցիչները, ինչն էլ մեզ անհրաժեշտ է:

Պետք է ուշադրություն դարձնել նաև այս համակարգի հիմքը հանդիսացող vector-ներին: Նրանց շատ լայնածավալ օգտագործման պատճառով առաջացավ չափումներ կատարելու կարիք, որոնցից եզրակացնում ենք, որ vector-ի արագությունը իր անդամին ջնջելուց շատ ցածր է, բացառությամբ վերջին անդամին ջնջելը, նույնը վերաբերվում է անդամ ավելացնելուն: Այդ պաճառով ստեղծվեց ObjDBManager<template> class-ը: Այն ոչ թե միանգամից է կատարում փոփոխությունները, այլ հավաքում է ավելացվող և ջնջվող անդամների ցուցակը, և ObjDBManager.PushChanges() ֆունկցիայի կանչից հետո, հնարավորության դեպքում ջնջվող անդամներին փոխարինում է ավելացվողներով (հետևանքում ցանկացած ջնջելու կամ ավելացնելու փոփոխություն կատարվում է վերջում, առավելագույն արագությամբ), հետևաբար պահանջվում է ևս մի տող կոդ, այն է՝



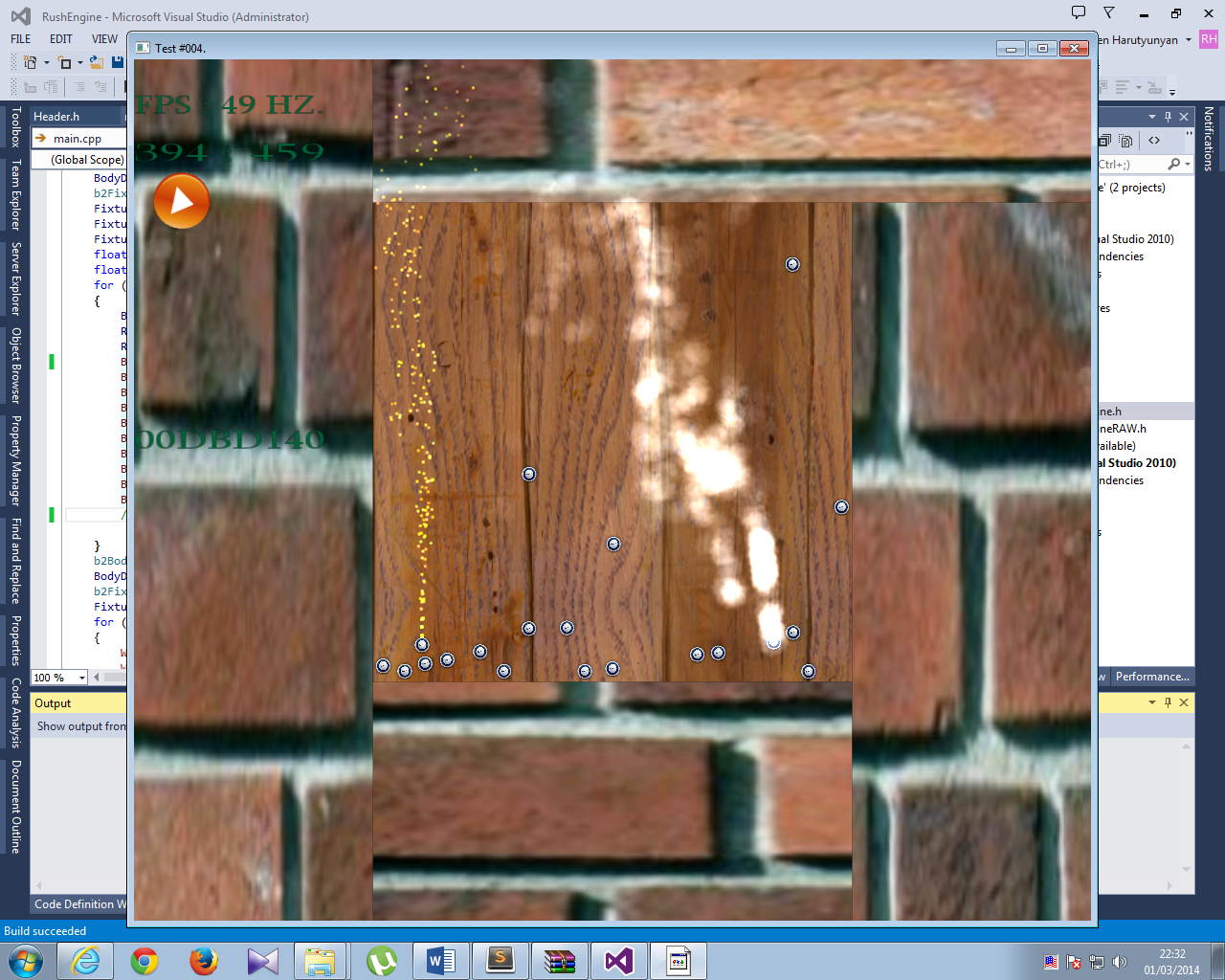
main() ֆունկցիան իրենից ներկայացնում է ընդամենը կարգավորված ֆունկցիաների կանչերի կարգ, ինչը միշտ կիրառվել է խաղերի մեջ՝ համարվելով ամենալավ կազմակերպված, էֆֆեկտիվ համակարգը:

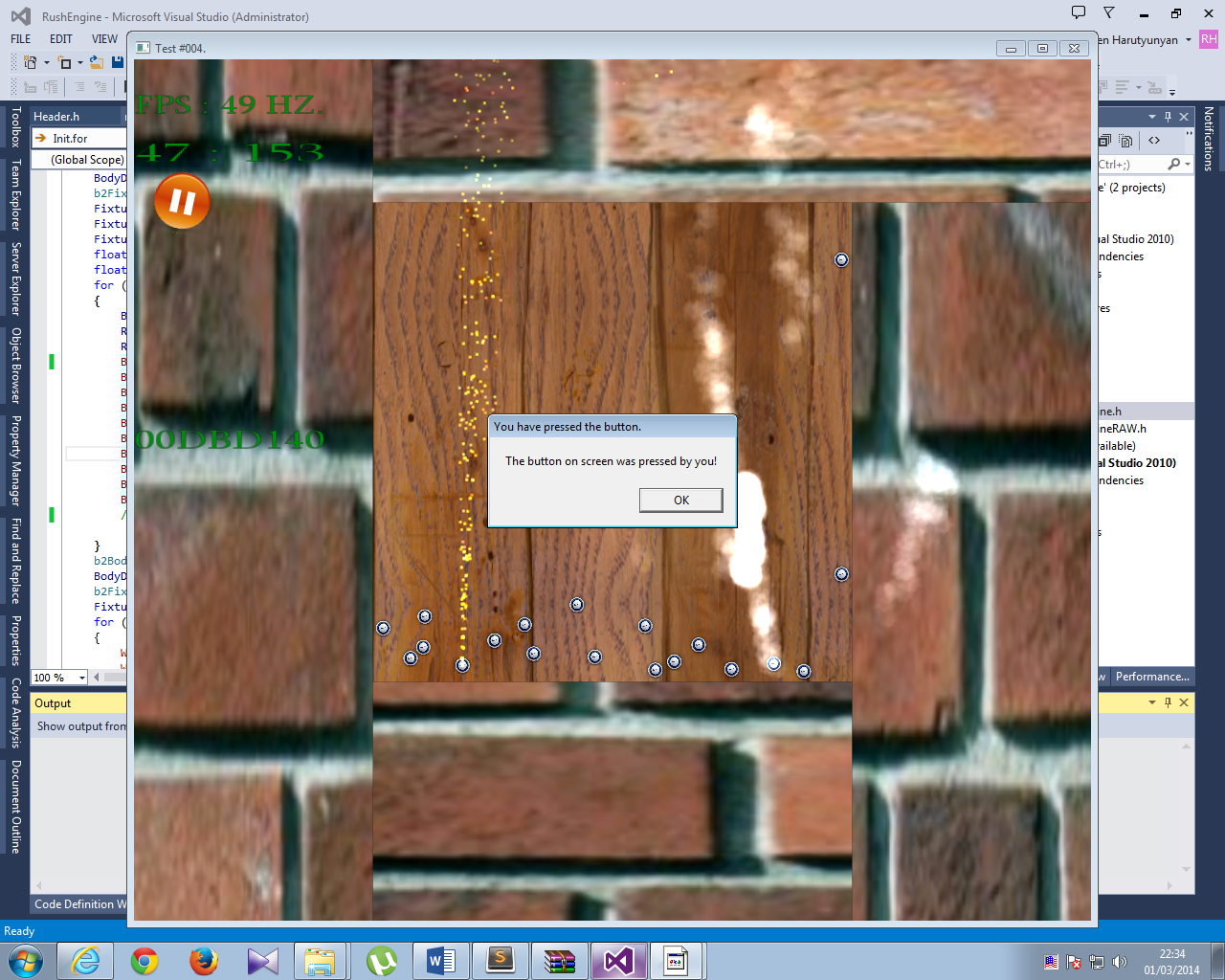
Այսպես մենք ստանում ենք մեր առաջին թեսթավթրման միջավայրը՝ ծխի, կայծերի էֆֆեկտներով, 4 պատերից և 20 գնդերից կազմված (Object class) ֆիզիկական միջավայրով, մի կոճակով, որը սեղմվելուց MessageBox-է ստեղծում, և մի հետին պլանի նկարից կազմված:

ՈՒշադրություն դարձրեք այն փաստին, որ կոճակների սեղմման մասին տեղեկությունները հավաքվում են մի while ցիկլի ընթացքում, ինչը բացառում է մի event-ի մի քանի անգամ կրկնվալու դեպքը (սեղմման ժամանակի ավելի երկար լինելու, քան մի ցիկլի ժամանկն է պատճառով):



Թեստի screenshoot-երը՝





1. Վերջաբան

Ի տարբերություն սկզբնական նպատակի, որը պահանջում էր հասնել ավելի գլոբալ և շուկայի կողմից պահանջված նպատակի, աշխատանքի ընթացքում առաջացավ կարիք՝ ստեղծել ծրագծավորման միջավայր, և ոչ թե խաղ: Մեր ստեղծած միջավայրում գրաֆիկայի, ինտերֆեյսի և էֆֆեկտների հետ աշխատելը շատ հեշտ է և ոչ ժամանակատար: Մեր ստեղծած միջավայրի միջոցով մենք կարող ենք ծրագիրը գործածել ինչպես Android-ի, և iOS-ի, այլ նաև մյուս օպերացիոն համակարգերի միջավայրերում (ներառելով Windows/Mac/Linux desktop միջավայրերը), կոդում նվազագույն ձևափոխություններ կատարելով (միայն խաղի կառավարման կոդում, քանի որ touch/mouse/keyboard/accelerometer կառավարման համակարգերի միջև տարբերությունները շատ մեծ են և խիստ կախված խաղի պահանջներից): Ինչպես տեսաք՝ վերը նշված Test-ում ընդհամենը մի քանի տասնյակ տող պարզ կոդով, հնարավորություն ենք ունենում գրաֆիկայի հետ կապված խնդիրների և դժվարությունների մեծ մասը լուծել: Մեզ միայն մնում է ստեղծել խաղի համար պահանջվող asset-ները (.png պատկերներ, .ttf ֆոնտեր և այլն) և գրել նրա աշխատանքի տրամաբանությունը: Մենք հնարավորություն ենք ստանում ստեղծել խաղ, որը կբավարարի ներկայիս շուկայի պահանջներին: Այս աշխատանքի շնորհիվ մենք էլ ավելի զարգացրեցինք մեր հմտությունները ծրագրավորման մեջ և ծանոթացանք ծրագրավորման լեզուներին, նրանց առանձնահատկություններին, ծրագրավորման միջավայրերին և նրանց տարատեսակներին, ձեռք բերելով ապագայի համար բավականաչափ հմտություններ` ավելի բարձր որակի ծրագրեր ստեղծելու համար: Ավարտեմ միտքս նրանով, որ մենք այս գիտահետազոտական աշխատանքի շնորհիվ շատ բան ենք սովորել, սկսած ծրագրավորման լեզուներից, վերջացրած ծրագրեր գրելու մեր ձեռք բերած փորձից, որը մեզ շատ է պետք գալու ապագայում:

1. Հղումներ

SDL ( [www.libsdl.org](http://www.libsdl.org/))

Box2D ([www.box2d.org](http://www.box2d.org/))

Android Development Kit (developer.android.com/sdk/index.html)

Android NDK Tutorial (<http://www.ntu.edu.sg/home/ehchua/programming/android/Android_NDK.html>)

Cygwin ([www.cygwin.com](http://www.cygwin.com/))

Cygwin Tutorial (<http://www.ntu.edu.sg/home/ehchua/programming/howto/Cygwin_HowTo.html>)

Git (<http://git-scm.com/>)

GitExtensions (<https://code.google.com/p/gitextensions/>)

Git Server ([www.bitbucket.org](http://www.bitbucket.org/))

Doxygen ([www.doxygen.org/](http://www.doxygen.org/))