UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Jan

OGRODJE ZA RAZVOJ IGER ZA iOS

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: doc. dr. Peter Peer

Ljubljana, 2011

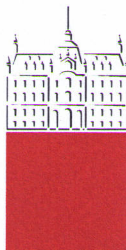
Rezultati diplomskega dela so intelektualna lastnina Fakultete za računalništvo in informatiko Univerze v Ljubljani. Za objavljanje ali izkoriščanje rezultatov diplomskega dela je potrebno pisno soglasje Fakultete za računalništvo in informatiko ter mentorja.

Izvorna koda izdelanega ogrodja XNI do različice 0.4.4 je ponujena odprtokodno pod licencno MIT.

Besedilo je oblikovano z urejevalnikom besedil LyX in sistemom LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si



Št. naloge: 01765/2011

Datum: 01.09.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ JAN**

Naslov: **OGRODJE ZA RAZVOJ IGER ZA IOS**
GAME DEVELOPMENT FRAMEWORK FOR IOS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Predstavite izdelavo programske knjižnice za razvoj računalniških iger za platformo iOS, čimbolj primerno za učenje razvoja iger. Opišite smernice in utemeljite izbor arhitekture. Razložite potek izdelave in uporabljene rešitve za dosego zadanih ciljev. Zaključite s pregledom ustvarjene funkcionalnosti, najpomembnejšimi izkušnjami iz dela za okolje iOS, rezultati uporabe ogrodja v praksi in smernicami za nadaljnji razvoj.

Mentor:

doc. dr. Peter Peer

Dekan:

prof. dr. Nikolaj Zimic

IZJAVA O AVTORSTVU
diplomskega dela

Spodaj podpisani Matej Jan,

z vpisno številko 63030086,

sem avtor diplomskega dela z naslovom:

Ogrodje za razvoj iger za iOS

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Peter Peera
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 27.9.2011 Podpis avtorja:

Zahvala

Pisanje diplomskega dela je potekalo precej hitro, saj ni težko pisati o stvari, ki si ji z veseljem posvetil večji del preteklega leta. Za tekoče zapisovanje misli se zahvaljujem reviji Joker in Sergeju Hvali, ki je z jedijskim potrpljenjem skrbel za urjenje mojega sloga v petih letih dela tam.

Preostale zahvale ležijo v času pred pisanjem, od prvih korakov v Objective-Cju do prve prave izdane igre.

Za prvi stik z razvojem za iOS se zahvaljujem mentorju Petru Peeru in ekipi GameTeam za odlično iPhone poletno šolo. Ker vem, da so kvalitetne pedagoške dejavnosti bolj dobra volja proaktivnih učiteljev, kot s strani fakultete primerno nagrajeno poslanstvo, si zaslužijo posebno spoštovanje.

Naslednji korak do ogrodja iz teme diplomske naloge se nanaša na razvoj igre za inovativni projekt Moject. Zahvala gre Davidu Slocombu, Adrianu Ashleyu in ekipi Razuma, da so mi zaupali razvoj in pustili eksperimentirati s 3D grafiko iz česar je nastal prototip ogrodja.

Sledilo je pedagoško obdobje, ko sem skupaj z mentorjem Petrom Peerom in asistentom Bojanom Klemencem pripravljal ogrodje za uporabo na vajah predmeta Tehnologija iger in navidezna resničnost. Zahvaljujem se jima za priložnost, da sem lahko pustil svoj pečat pri pripravi učnega programa ter seveda za sodelovanje pri izvedbi, kar je privedlo do iskrenih pohval študentov na koncu semestra. Njim še posebej hvala, saj sem se od njih naučil vsaj toliko, kolikor so se oni od mene. Preveč jih je, da bi jih našteval, kot svoj iskren poklon pa sem jim pripravil video montažo ustvarjenih iger.

Medtem ko sem se izgubljal med šolskimi klopmi, so Razumovci pripravljali naskok na mobilni trg iger. Za prvi izdan naslov so mi zopet zaupali programersko svobodo in moje ogrodje je postalo podlaga za prvo izdano igro pod novim imenom ekipe Dawn of Play. Čeprav sem se kdaj izgubil na naši skupni poti, gre vsem sodelavcem velika zahvala, da sploh lahko počnem, kar me veseli. Posebno mesto zaslužita Rok Jamnik in Žiga Hajdukovič, ki kljub vsemu verjameta vame, ter Jan Hadžić, katerega ilustracije vedno navdušujejo.

Da bi uspešno končal izpite in v miru spisal diplomo sem se zatekel na študijski dopust k staršema. Kot ponavadi sta mi nesebično ponudila svojo podporo in gostoljubje. Hvala! Verjetno bosta od vseh prav onadva najbolj zadovoljna, ko bo diploma končno za mano.

Svoje mesto si zaslužijo tudi vsi, ki so me v tem času spremljali v premnogih dvigih in spustih. Najbližja prijatelja v iskanju neštetih trenutkov odraščanja Petra in Jan. Sopotnik v študiju, ponočevanjih in stanovanjskih dogodivščinah Goran Gligorin. Edini, ki zares verjame v svoje sanje, Iztok Levac. Marko, ki naju z Iztokom drži na tleh. Superwoman Daphne in celotna ekipa Urban Roof. Plesalci, ki razumejo mojo potrebo po gibanju, člani Sweet With Style in Roman Urek. Neustavljivi ambiciozneži Gregor, Katja, Nuša in Andrej de Reggi. Svetli točki, zaradi katerih po vsakem dežju posije sonce, Mojca in Kristina. Sotrpnika iz asistentskih vrst fakultete Ciril Bohak in Luka Čehovin. Vzpodbud polni razvijalci iger, bolje znani kot Lamoot, BlodyAvenger in Aryes. Marc Dingena, ki s projektom Tjoonz skrbi, da programiranje vedno poteka v pravem ritmu. Klemen, Sofija, Mateja in Manca, za vse sprehode in neskončne pogovore. Brez vseh vas zadnja leta študentskega življenja ne bi bila nič več kot kup praznih listov.

Čisto na koncu, a ne nazadnje, se še enkrat zahvaljujem mentorju Petru Peeru. Pri izbiri in pisanju diplome mi je dal vse možnosti in izkazal popolno zaupanje. Kljub temu, da sta si najini mnenji glede izvedbe predmeta včasih prišli navzkriž, je vedno ostal v pomoč. Za mentorja si ne bi mogel želeti več.

V spomin na 3D-Level in Artificial Engines

Kazalo

Povzetek 1

Abstract 2

1 Uvod 3

1.1 Razvoj iger 3
1.2 Oblike programskih knjižnic za razvoj iger 4
1.2.1 Obseg 4
1.2.2 Programski jezik in arhitekturna paradigma 7
1.3 Razvoj za iOS 7
1.3.1 iOS in XNA 8
1.4 Učenje razvoja iger 13
1.5 Pregled naloge 15

2 Načrt prenosa ogrodja XNA v okolje iOS 17

2.1 Smernice za razvoj ogrodja XNI 17
2.2 Pregled arhitekture ogrodja XNA 19
2.3 Pregled arhitekture iOS 22
2.4 Razlike med programskima jezikoma C# in Objective-C 23
2.4.1 Imenski prostori 23
2.4.2 Zasebne in notranje metode ter lastnosti 26
2.4.3 Strukture 27
2.4.4 Zbirke s statičnim tipom 28
2.4.5 Dogodki 30
2.5 Razlike med okoljema Direct3D in OpenGL 30
2.5.1 Programabilni senčilniki 30
2.5.2 Urejenost matrik in transformacija med koordinatnimi sistemi 31

3 Razvoj ogrodja XNI **33**
3.1 Kreiranje projekta v Xcode 33
3.2 Razvoj osnovne arhitekture 34
3.2.1 Game in GameHost 34
3.2.2 Izvedba igrine zanke 35
3.2.3 GameWindow, GameView in GameViewController . . . 36
3.3 Razvoj cevovoda vsebine 37
3.4 Uporaba ogrodja med razvojem 37
3.5 Izdaja ogrodja 38

4 Uporaba **40**
4.1 Končni izdelek 40
4.2 Uporaba XNI v izobraževanju 41
4.3 Uporaba XNI v profesionalnem razvoju 43
4.3.1 Hitrost razvoja 44
4.3.2 Hitrost izvajanja 47
4.3.3 Razširljivost 48

5 Zaključek **50**
5.1 Napake in popravki 51
5.2 Nadaljnji razvoj 53
5.2.1 Tehnične izboljšave 53
5.2.2 Tržna strategija 55

Seznam slik **57**
Seznam tabel **58**

Literatura **59**

Seznam uporabljenih kratic in simbolov

ARM — Advanced RISC Machine

CIL — Common Intermediate Language

DLL — Dynamic Link Library

GUI — Graphical User Interface

FPS — Frames Per Second

GLSL — OpenGL Shading Language

GPS — Global Positioning System

HLSL — High Level Shader Language

LCD — Liquid Crystal Display

LLVM — Low Level Virtual Machine

MVC — Model-View-Controller

OpenGL — Open Graphics Library

OpenGL ES — Open Graphics Library for Embedded Systems

SDL — Simple DirectMedia Layer

TINR — Tehnologija iger in navidezna resničnost

XML — Extensible Markup Language

XNA — XNA is Not an Acronym

XNI — XNA for iOS

Povzetek

Med razvojem iger se izoblikujejo deli programske kode, ki so neodvisni od igralnosti. Da jih lahko uporabljamo med več projekti in delimo z drugimi uporabniki jih ločimo v programske knjižnice. Te v svetu iger obstajajo na več nivojih, od nizkonivojskih knjižnic, preko srednjenivojskih ogrodij, do visokonivojskih pogonov iger. V diplomskem delu predstavljamo razvoj srednjenivojskega ogrodja XNI za razvoj iger za mobilne naprave iOS. Zaradi preteklih izkušenj z Microsoftovim ogrodjem XNA, ki je zelo primerno za uporabo v izobraževanju, smo naše ogrodje zasnovali kot arhitekturno kopijo XNA iz programskega jezika C# v Objective-C. Po analizi obeh okolij smo pripravili uporabne tehnike za prenos kode med njima, med izvedbo pa še konkretne rešitve za programiranje iger s knjižnicami iOS. Skozi iterativni razvoj smo dodatno izpopolnili celoten postopek izdelave, uporabe in izdaje statične knjižnice v razvojnem okolju Xcode. Ogrodje se je dobro obneslo tako pri izvajanju praktičnega dela poučevanja razvoja iger na visokošolskem študiju, kot v profesionalnem razvoju. Skozi uporabo so se pojavile tudi pomanjkljivosti, za katere smo v zaključku predvideli ustrezne popravke. Na koncu ponujamo še možni dolgoročnejši načrt razvoja, tako s tehničnega kot tržnega stališča.

Ključne besede:

razvoj iger, izobraževanje, ogrodje XNA, iOS, iPhone, iPad, Objective-C, C#, OpenGL, UIKit, Apple

Abstract

Many parts of source code in game development stay independent of gameplay. They are separated into program libraries for reuse on multiple projects and sharing between users. Game development identifies them on different levels, from lower level libraries and frameworks to higher level middleware and game engines. The thesis focuses on the development of XNI Framework, a mid-level static library for developing games for iOS devices. It is designed as a one-on-one class copy of Microsoft's XNA Framework due to our positive experience and high regard of its use in education. Our analysis of the differences between C# and Objective-C, as well as the respective environments of the source and target platforms, enabled us to develop useful techniques for translating code and reimplement the framework with iOS libraries. Throughout the iterative development, we perfected the workflow for producing, using and publishing a static library with Xcode. XNI was successfully used in practice, both in education and professional game development. We also identified many potential areas for improvement and provided solutions for their utilization. Our work is concluded with possible long-term direction of the framework with insights into both technical and economical aspects of its future.

Key words:

game development, education, XNA Framework, iOS, iPhone, iPad, Objective-C, C#, OpenGL, UIKit, Apple

Poglavje 1

Uvod

1.1 Razvoj iger

Razvoj računalniških iger je multidisciplinarna dejavnost, ki združuje inženirsko stran računalništva z oblikovalskimi področji ustvarjanja vizualnih, literarnih in zvočnih del. Pomembna so tudi organizacijska znanja ravnateljevanja, vodenja projektov in nadzora kakovosti ter širše povezava s trženjem in prodajo [1]. Na področju računalništva spada med razvoj programske opreme, a črpa znanja iz veliko ostalih področij: razvoja algoritmov in podatkovnih struktur, računalniške grafike, umetne inteligence, modeliranja in simulacije, mrežne komunikacije in ostalih [2].

Razvoj računalniških iger je od garažnih začetkov v sedemdesetih letih prejšnjega stoletja, ko je lahko posameznik sam naredil celotno igro, zraslo v veliko gospodarsko panogo z izdelki naraščajoče kompleksnosti [2, 3]. Tako kot pri razvoju ostale programske opreme, se tudi pri razvoju iger s časom izoblikujejo deli izvorne kode, ki med različnimi izdelki ostajajo bolj ali manj enaki. Da bi lahko spisano funkcionalnost porabili v več izdelkih, se jo izloči iz končnega izdelka in iz nje izoblikuje programsko knjižnico. Programerji si s tem olajšajo in pohitrijo izdelovanje [4].

V diplomski nalogi bomo predstavili nastanek take programske knjižnice, ogrodja XNI, ki služi razvoju iger za platformo iOS. Naš cilj je pojasniti razloge za njen nastanek in predstaviti sam razvoj ogrodja od načrtovanja do uporabe v praksi. Pred tem je potrebno razumeti posamezne tematike, v sklopu katerih je nastalo ogrodje XNI. Še posebej to velja ob dejstvu, da je XNI zasnovan kot čimbolj direktna kopija že obstoječega ogrodja XNA, Microsoftove programske knjižnice za razvoj iger za operacijski sistem Windows, Windows Phone 7 in igralno konzolo Xbox 360.

1.2 Oblike programskih knjižnic za razvoj iger

Programske knjižnice se razlikujejo v mnogo dimenzijah, od katerih je za področje razvoja iger pomembnih kar nekaj. Naštejmo ključne lastnosti direktno na primeru XNI, za lažjo umestitev v natančnejšem pregledu, ki sledi:

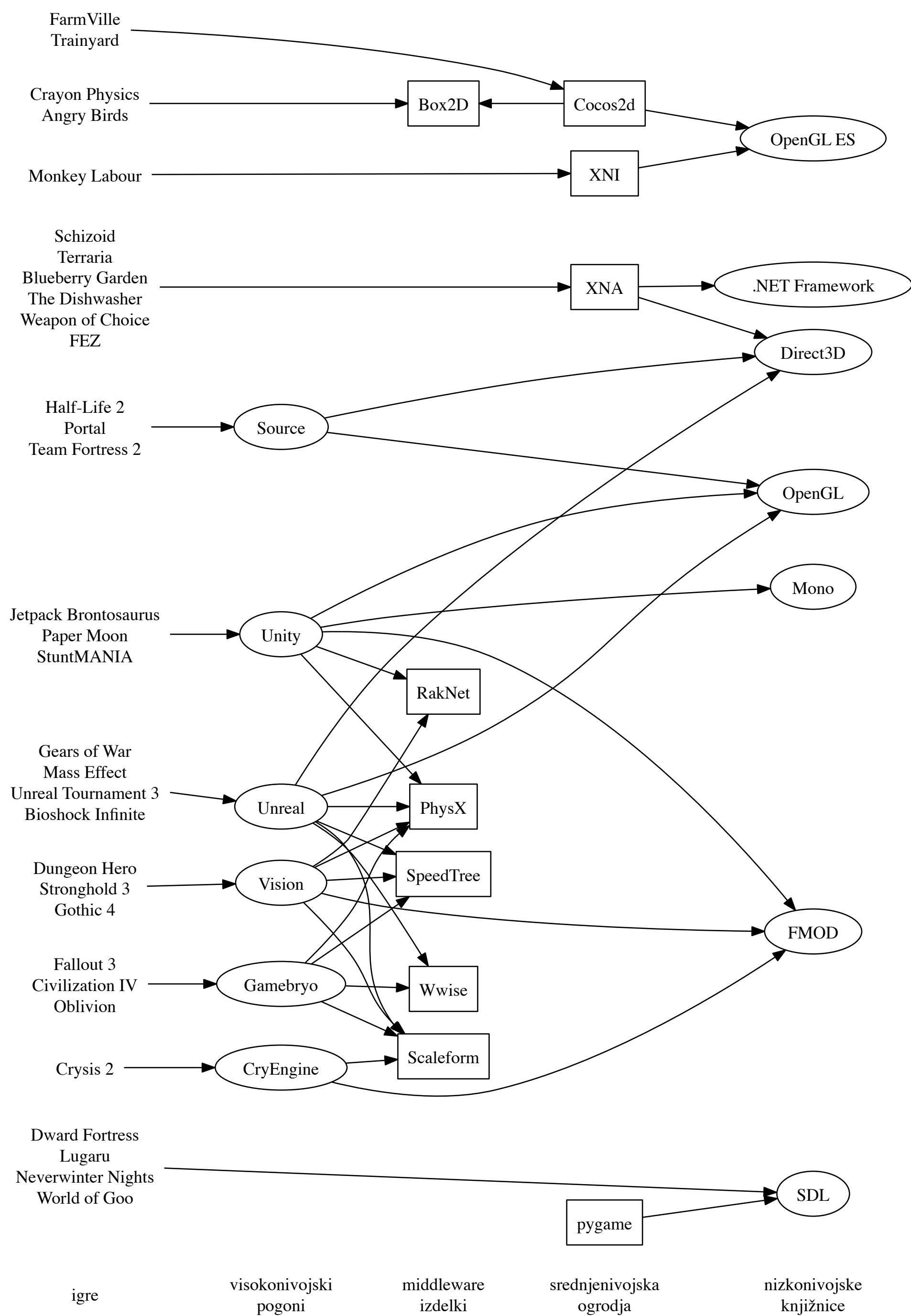
- *Obseg*: Obseg določa vsebino knjižnice. Funkcionalnost ogrodja XNI je zelo široka in pokriva večino področij razvoja, od izrisa, zvoka in vhodnih naprav, do povezovanja več igralcev in spletnih storitev. Po višini leži na srednjem nivoju in je splošno namenjeno razvoju vsakovrstnih iger.

- *Programski jezik*: Programski jezik skupaj z odvisnimi knjižnicami (tistimi, ki jih knjižnica uporablja za svoje delovanje) določa, v katerih programskih jezikih in okoljih je možno knjižnico uporabiti. XNI je spisan v programskem jeziku Objective-C, na podlagi knjižnic operacijskega sistema iOS. Tako se tudi igre z XNIjem piše v Objective-Cju in izvaja na napravah z iOS.

- *Arhitekturna paradigma*: Izbira obsega in programskega jezika še vedno dopušča svobodo pri izvedbi s stališča stila programiranja. Arhitektura XNI močno odraža načela objektno-orientiranega programiranja. Kot ogrodje se poslužuje načela prevzema nadzora (angl. inversion of control) in od uporabnika zahteva, da svojo igro zgradi znotraj že pripravljene strukture.

1.2.1 Obseg

Med razvojem programske opreme se zaradi želje po boljši organiziranosti oziroma, samo po sebi, zaradi uporabe ločenih delov kode v obliki knjižnic, ustvarja določena hierarhija, ki jo je mogoče opisati z usmerjenim acikličnim grafom. Ta ureditev nam omogoča, da posamezne dele označimo za višje ali nižje [3]. Tudi v praksi pogosto srečamo oznake knjižnic, da so nizke ali visoke stopnje (angl. low, high-level). Ker gre za neformalno delitev brez določenih meja, kje se končajo ene in začnejo druge, lahko zgolj opišemo, kakšne vrste knjižnic najdemo na določeni višini. Ena od splošnih lastnosti, ki se pri tem izrazi, je predvsem specifičnost. Primere nekaterih knjižnic iz domene razvoja iger smo za ilustracijo višine skupaj z njihovimi medsebojnimi odvisnostmi zajeli na sliki 1.1.



Slika 1.1: Višina programskih knjižnic za razvoj iger

Za obseg je pomembna tudi širina knjižnice. Odvisna je od števila različnih področij, ki jih pokriva funkcionalnost, pri tem pa skupaj z višino določa, kako specifično rešuje probleme teh področij.

Knjižnice, ki se običajno označujejo z nizko stopnjo, se ukvarjajo s primitivnimi operacijami. Grafična vmesnika OpenGL in Direct3D na primer skrbita za izvajanje osnovnih grafičnih ukazov v grafični procesni enoti. Na podobno stopnjo spadajo večplatformske knjižnice, ki poskrbijo za enovit vmesnik do vhodnih in izhodnih naprav (primer na sliki 1.1 je SDL), podporo računskim operacijam linearne algebre in branje podatkov iz standardnih formatov. Na tej stopnji knjižnice niso osredotočene zgolj na igre; njihovo uporabo dosledno srečamo tudi v znanstvenih vizualizacijah in navidezni resničnosti. Hkrati se večja ali manjša specializiranost za razvoj iger vseeno kaže v njihovih lastnostih, recimo osredotočenosti na hitrost izvedbe pred ostalimi faktorji, saj igre v prvi vrsti zahtevajo interaktivnost v realnem času.

Nivo višje stojijo knjižnice, ki že rešujejo probleme specifične za razvoj iger. Ukaze iz grafičnih knjižnic spodnje stopnje se uporabi za izgradnjo splošno uporabnih operacij, kot je izris slik v 2D prostoru in izris statičnih ter animiranih 3D modelov. Matematične zmogljivosti so nadgrajene s podporo reševanju običajnih geometrijskih problemov, kot je računanje obsegajočih krogel 3D modela (angl. bounding sphere) in sekanje le teh z vidnim poljem kamere v obliki prisekane piramide (angl. frustum). Druga področja vključujejo mrežno komunikacijo za izdelavo večigralskih iger, sistem za gradnjo grafičnega uporabniškega vmesnika in splošne fizikalne knjižnice. Samo oznako srednjega nivoja pri opisu splošnih (širokih) ogrodij sicer srečamo redko; uporabili smo ga, ker je spekter višine dovolj globok za trinivojsko delitev. Se pa z ozkim pomenom vmesne programske opreme (angl. middleware) v razvoju iger označuje ozke knjižnice, specializirane za eno samo področje [5]. Glede na našo delitev je kljub oznaki marsikatera knjižnica te vrste na relativno visokem nivoju.

Na najvišjem nivoju se knjižnice osredotočajo na specifične probleme enega izmed žanrov iger in predstavitvene paradigme značilne zanj (2D ali 3D okolje, prvoosebna ali tretjeosebna perspektiva). Govorimo večinoma o pogonih iger (angl. game engines), ki razvijalcem olajšajo veliko dela z v celoti izdelano podlago za razvoj igre v izbranem žanru. Hkrati so zaradi specializiranosti lahko tudi omejujoči in prilagoditev njihovega delovanja svojim potrebam ni trivialna [6].

1.2.2 Programski jezik in arhitekturna paradigma

Višje kot se nahaja programska knjižnica, več odločitev v smislu arhitekture so morali sprejeti njeni razvijalci. Na ta način knjižnice programerju svojo arhitekturo bolj ali manj tudi narekujejo. Knjižnice, ki delujejo na način, da programerju ponudijo okostje, znotraj katerega ta le še spiše za njegov izdelek specifično kodo, pravimo ogrodja (angl. framework). Ogrodja prevzamejo nadzor nad izvajanjem in ga vrne končnemu programu le na mestih, kjer je to potrebno (angl. inversion of control) [7]. Običajne knjižnice na drugi strani zgolj združujejo pogosto uporabljene rutine in v primeru objektne orientiranosti postrežejo z relativno nepovezanimi razredi. Nad njimi ima, za razliko od ogrodij, programer vedno nadzor in od njih zahteva naj zanj izvedejo neko operacijo.

Podobno kot razlika med ogrodji in običajnimi knjižnicami se arhitektura odraža s tem, ali je zasnovana proceduralno ali objektno. Programski jezik v katerem je napisana v večini že nosi to informacijo, saj v Cju recimo ni mogoče spisati objektno orientirane knjižnice. Žal tudi pri knjižnicah spisanih v objektno naravnanih programskih jezikih dejanska arhitektura nujno ne odraža načel objektno orientiranega načrtovanja. Dobra arhitektura zahteva izkušene programske načrtovalce, skupaj z dobrim poznavanjem in izkoriščanjem lastnosti ciljnega programskega jezika. Dodatno je v programiranju mogoče doseči enako funkcionalnost z radikalno različnimi pristopi, kar naredi oznako dobre arhitekture precej subjektivno. V praksi najdemo primere za vse omenjene razlike, kar postavi izbiro programskega jezika (C, C++, Objective-C, Java, C# (C sharp), Python ...) proceduralne ali objektne paradigme (OpenGL proti Direct3D) ter običajne knjižnice napram ogrodju (SDL proti XNA) na mejo okusa. Če obseg knjižnice določa njeno funkcionalnost, potem izbira programskega jezika in arhitekturne paradigme določa stil, s katerim knjižnica to funkcionalnost ponuja programerju. Različne potrebe in okusi tako omogočajo veliko število konkurenčnih knjižnic.

1.3 Razvoj za iOS

Omenili smo, da je XNI zgrajen s pomočjo knjižnic operacijskega sistema iOS. Gre za Applov operacijski sistem, nameščen na njihovih prenosnih napravah: pametnemu telefonu iPhone, tabličnemu računalniku iPad in multimedijskemu predvajalniku iPod touch [8]. Kljub različnim primarnim namembnostim so vse iOS naprave računalniki s skupnim uporabniškim vmesnikom.

Hkrati iOS pogosto razumemo kot platformo, saj poleg operacijskega sistema prinaša dobro definirane strojne naprave in podporno okolje, predvsem prodajno pot App Store, preko katerega uporabniki dostopajo do nove programske opreme. Kombinacija vsega trojega je omogočila, da je na trgu zasedla pomembno mesto. Ne le na področju mobilnih telefonov [8], temveč tudi kot prenosna igralna naprava [9].

Pomembna lastnost naprav je, da njihovi procesorji uporabljajo nabor strojnih ukazov ARM, medtem ko razvoj v veliki meri opravljamo še v simulatorju iOS okolja znotraj operacijskega sistema Mac OS X na osebnih računalnikih z naborom strojnih ukazov x86 [10]. To je pomembno pri razvoju programskih knjižnic, saj je zaželeno uporabniku ponuditi obe verziji: eno za razvoj na simulatorju in drugo, ki teče na končni strojni opremi. Knjižnici, ki vsebuje prevod v več strojnih arhitektur rečemo univerzalna (pogovorno tudi debela, angl. fat) knjižnica [11, 12].

V prejšnjem podpoglavju o oblikah programskih knjižnic smo govorili o vsebinski in programsko-arhitekturni obliki. Pri delu s platformo iOS je potrebno omeniti še povsem tehnično ozadje v zvezi s povezovanjem programske kode knjižnice v ciljni program. Poznamo statične in dinamične knjižnice. Prve poveže s ciljnim programom že povezovalnik (angl. linker), medtem ko dinamične knjižnice omogočajo boljše deljenje z drugimi programi s povezovanjem pred zagonom ali med izvajanjem [13]. Apple v svojem razvojnem okolju Xcode omogoča le izdelavo statičnih knjižnic za iOS [14].

Še ena pomembna lastnost razvoja za iOS je izbira programskih jezikov. Apple ponuja prevajalnik za C, C++ in Objective-C [15]. Slednji je popolna nadgradnja (angl. strict superset) Cja, tako da lahko v Objective-Cju prosto uporabljamo klasični C. Tudi, če se odločimo za C ali C++, se moramo vsaj pri končni aplikaciji Objective-Cja vsaj dotakniti. Programske knjižnice nižjega nivoja delujejo kot ogrodja in zahtevajo, da našo aplikacijo izdelano z dedovanjem iz osnovnih razredov UIKit knjižnice, spisanih v Objective-Cju.

1.3.1 iOS in XNA

Glavni razlog za nastanek ogrodja XNI leži v želji po razvoju iger za iOS z ogrodjem XNA. Zametki ležijo v januarju 2010, ko smo se na Razumu prvič začeli ukvarjati s platformo iOS. Poznavanje zgodovine in takratnega stanja razvoja za iOS je pomembno, saj so bile odločitve glede knjižnice odvisne tudi od političnih odločitev Appla glede možnosti razvoja za iOS.

Pred razvojem za iOS smo pridobili večletne izkušnje z Microsoftovim objektno orientiranim ogrodjem srednjega sloja XNA. V tistem času je bilo na-

knjižnica	cena tržne uporabe	arhitekturna paradigma	programski jezik	vrsta knjižnice (višina)	3D grafika
Bord3D	49 $	objektno-orientirana	C++	srednjenivojski pogon	da
Cocos2d	zastonj (MIT)	objektno-orientirana	Objective-C	srednjenivojsko ogrodje	da, s Cocos3d
Corona	199 $/leto	skriptni jezik	Lua	srednjenivojska knjižnica	ne
GameSalad	zastonj	vizualni urejevalnik	/	visokonivojski urejevalnik	ne
Galaxy Engine	zastonj (New BSD)	objektno-orientirana	Objective-C	srednjenivojsko ogrodje z urejevalniki	da
Game Editor	zastonj (GPL) ali 99 $/leto	vizualni urejevalnik, skriptni jezik	C	visokonivojski pogon z urejevalnikom	ne
GLBasic	80 €	proceduralna	Basic	srednjenivojska knjižnica	da
Impact	99 $	objektno-orientirana	JavaScript	srednjenivojsko ogrodje	ne
iptk	zastonj (MIT)	objektno-orientirana	C++	nizkonivojska knjižnica	da
iSGL3D	zastonj (MIT)	objektno-orientirana	Objective-C	srednjenivojsko ogrodje	da
iTorque 2D	99 $	vizualni urejevalnik, skriptni jezik	TorqueScript	visokonivojski pogon z urejevalnikom	ne
Marmalade	149 $	proceduralna, skriptni jezik	C++, Lua	visokonivojski pogon	da
Oolong Engine	zastonj (MIT)	objektno-orientirana, proceduralna	C, C++	nizkonivojska knjižnica	da, direktno OpenGL
Sparrow	zastonj (BSD)	objektno-orientirana	Objective-C	srednjenivojsko ogrodje	ne
SIO2	399,99 $	proceduralna, skriptni jezik	C++, Lua	visokonivojski pogon	da
ShiVa3D	169 €	vizualni urejevalnik, skriptni jezik	C++, Lua, Objective-C	visokonivojski pogon	da
Unity	399 $	vizualni urejevalnik, skriptni jezik	JavaScript, C#, Boo	visokonivojski pogon z urejevalnikom	da
Unreal	99 $	objektno-orientirana, skriptni jezik	C++, UnrealScript	visokonivojski pogon z urejevalniki	da
XNI	zastonj (MIT)	objektno-orientirana	Objective-C	srednjenivojsko ogrodje	da

Tabela 1.1: Knjižnice za razvoj iger za iOS

menjeno razvoju iger za PCje z operacijskim sistemom Windows ter igralno konzolo Xbox 360. Zahteve ob iskanju primerne knjižnice za platformo iOS so bile podobne: možnost brezplačne uporabe, objektna orientiranost in funkcionalnost na nivoju srednjega sloja s podporo 3D grafiki. Po pregledu najpopularnejših knjižnic tistega časa, nobena od možnosti ni povsem ustrezala. Še najbližje kriterijem je bil Cocos2d. Danes bi skupaj z razširitveno knjižnico Cocos3d relativno ustrezal, vendar je bila prva javna objava razširitve šele maja 2011. Podobno danes našim kriterijem ustrezata še Galaxy Engine in iSGL3D, vendar noben od njiju ni bil na voljo v začetku 2010. Natančen pregled trenutne ponudbe knjižnic [16, 17, 18] glede na naše kriterije je podan v tabeli 1.1.

Preverili smo tudi možnost pisanja iger za iOS direktno z ogrodjem XNA. XNA deluje na podlagi ogrodja .NET, ki je zasnovano za prenosljivost med različnimi platformami. Ker je Microsoft poskrbel le za Windows in Xbox 360, so drugi avtorji ustvarili svojo implementacijo izvršilnega okolja za ostale platforme, kot so Mac in Linux, z imenom Mono [19]. Pri prenosu na iOS so se morali spopasti z Applovo omejitvijo razvoja za iOS, ki ni dovoljevala poganjanja interpretirane kode. Ker je na tem principu delovalo izvajanje v vmesno kodo prevedenih .NET programov, so posebej za razvoj za iOS pod imenom MonoTouch izdelali prevajalnik, ki vmesno kodo vnaprej prevede v strojne ukaze ARM procesorja [20].

Prenos .NET okolja na iOS je bil šele prvi korak do iger spisanih z ogrodjem XNA na iOSu. Poleg lastne izvedbe knjižnic ogrodja .NET je bilo potrebno isto storiti z ogrodjem XNA. To nalogo poskušata opraviti projekta MonoGame (predhodno XNATouch) [21] in novejši ExEn [22]. Slednji je nastal kasneje kot neodvisna vejitev prvega, z bolj osredotočenimi cilji in konsistentno izvedbo. V času našega odločanja je bil na voljo le XNATouch, pregled obeh rešitev skupaj z našim XNI pa je na voljo v tabeli 1.2.

Obstaja več razlogov, zakaj se nismo odločili za razvoj iger z ogrodjem XNATouch:

- MonoTouch, na katerem je zgrajen XNATouch, ni na voljo zastonj. S preizkusno različico je možno razvijati v okolju simulatorja, za izdajo na napravah z iOS (tržna uporaba) pa je trenutno najnižja cena 399 $ [23].

- XNATouch ni imel in še vedno nima implementirane podpore za 3D grafiko. Projekt, ki smo ga takrat načrtovali za razvoj, je to zahteval.

- XNATouch je bil v času odločanja zasnovan glede na XNA verzijo 3.1. Med tem je Microsoft izdal XNA 4.0, ki je v veliki meri spremenil in

knjižnica	cena tržne uporabe	vrsta rešitve	programski jezik	verzija ogrodja XNA	3D grafika
ExEn	zastonj, 399 $ za MonoTouch	.NET knjižnica na podlagi MonoTouch	C#	3.1	ne
MonoGame	zastonj (Ms-PL), 399 $ za MonoTouch	.NET knjižnica na podlagi MonoTouch	C#	4.0	ne
XNI	zastonj (MIT)	statična knjižnica za iOS	Objective-C	4.0	da

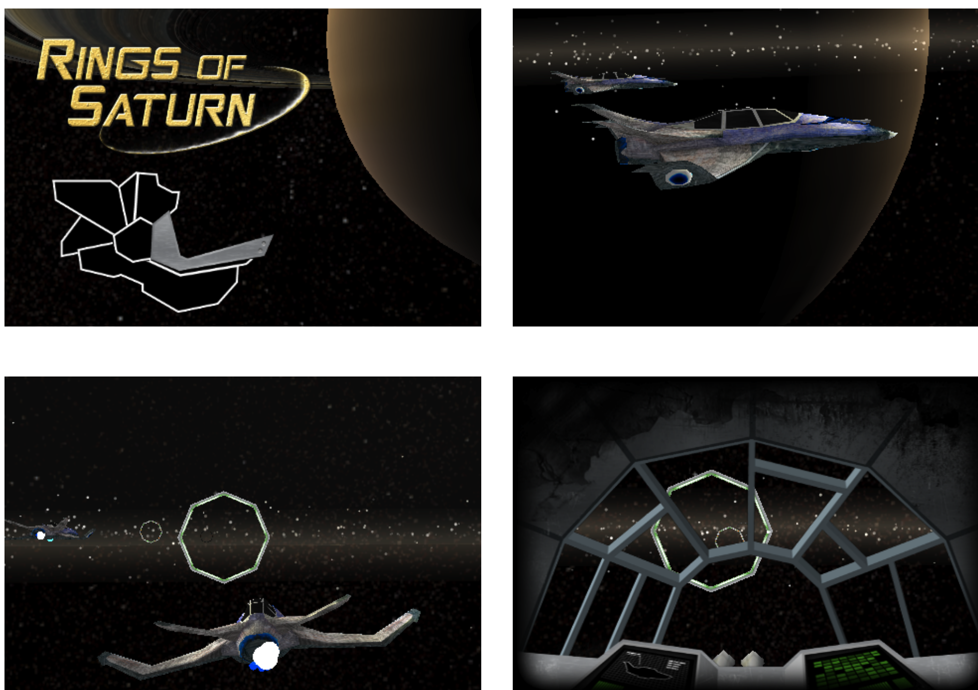
Tabela 1.2: Rešitve za uporabo ogrodja XNA v razvoju za iOS

izboljšal arhitekturo knjižnice. Prestrukturiranje po zgledu nove verzije so na projektu začeli šele marca 2011.

- Možnost uporabe orodja, s katerim je mogoče razvijati za več platform, je bila zaradi Applove politike precej vprašljiva. Strogo so zavrnili možnost delovanja okolij Java in Flash. Kljub prilagojenemu prevajalniku za .NET taka uporaba nekaj časa ni bila jasno dovoljena.

Glede zadnje točke se je kasneje najprej izkazalo, da je bila naša odločitev povsem pravilna. Aprila 2010 je Apple popravil enega izmed členov pogojev razvoja za iOS tako, da je dovoljeval programe razvite zgolj v jezikih C, C++, Objective-C in JavaScript [24]. To je prepovedalo uporabo orodij, ki so iz drugih programskih jezikov ustvarjali sicer neinterpretirano programsko kodo. Sledilo je obdobje močnega pritiska na Apple in septembra 2010 se je situacija končno obrnila v prid projekta MonoTouch in podobnih orodij. Apple je z novimi pogoji dovolil vse načine prevajanja in interpretiranja iz poljubnih jezikov z edino omejitvijo, da noben del kode ni naknadno prenesen preko spleta [25]. Za nastanek ogrodja XNI je bilo to sicer prepozno, saj je bil njegov razvoj že močno v delu, vendar tudi brez tretje in četrte točke, prvi dve še vedno držita. MonoTouch ni na voljo zastonj, preimenovani MonoGame, kakor tudi nova konkurenčna izvedba ExEn, še vedno nimata podpore za 3D grafiko.

Na podlagi vseh omenjenih faktorjev smo začeli s prototipno implementacijo lastne objektno orientirane knjižnice za razvoj iger srednjega nivoja v jeziku Objective-C. Ta se je zaradi našega preteklega dela močno zgledovala po arhitekturi ogrodja XNA in je združevala priporočeno kodo za razvoj OpenGL aplikacij v iOSu [27] s preostalimi razredi zasnovanimi po funkcio-



Slika 1.2: Posnetki zaslona igre Rings of Saturn

Slika 1.3: Igra Rings of Saturn v prototipu projekta Moject [26]

nalnosti razredov v ogrodju XNA. S to prototipno kodo, ki še ni bila ločena v svojo knjižnico, je bila marca 2010 razvita eksperimentalna 3D igra Rings of Saturn (sliki 1.2 in 1.3). Kasneje je ta prototip služil za izhodišče nastanka ogrodja XNI v sklopu izdelave primerne knjižnice za učenje razvoja iger za iOS, o čemer govorimo v naslednjem podpoglavju.

1.4 Učenje razvoja iger

Kmalu po izdelavi igre Rings of Saturn smo se ločeno začeli pogovarjati s Fakulteto za računalništvo in informatiko o pripravi gradiva za predmet Tehnologija iger in navidezna resničnost (TINR). XNI je v veliki meri posledica potrebe po knjižnici za izvajanje vaj pri tem predmetu, zato je pomembno razumeti tematiko učenja razvoja iger.

Glede na vrsto področij, ki jih pokriva razvoj iger, je potrebno povedati, da učni načrt predmeta TINR obsega razvoj iger s stališča programerjev. Drugi glavni pristop k učenju razvoja iger daje poudarek na oblikovanje igralnosti in spoznavanje celotne produkcije igre z razumevanjem povezovanja različnih področij [3]. Za ta način je primerno začeti poučevanje na zelo visokem nivoju, kjer se uporabi obstoječ pogon, razvojno okolje z urejevalniki ali kar eno od iger z dobro podporo za spreminjanje njene vsebine (angl. modding) [28, 29].

Sistem od zgoraj navzdol omogoča hitre rezultate in je primeren za tiste brez programerskih izkušenj, osredotočene zgolj na razvoj iger. Možno je tudi nadaljevanje v programiranje; skriptanje dogodkov v igri služi kot uvod v programiranje, vedno globlje poseganje v pogon počasi daje razumevanje delovanja pogona. Vseeno je v izobraževanju običajnejši pristop od spodaj navzgor. V okviru splošnega študija računalništva študentje predhodno pridobijo izkušnje iz večine od naslednjih tem: programiranja, objektnega razvoja aplikacij, računalniške grafike, linearne algebre, mrežnega programiranja in umetne inteligence [30]. Tako je tudi za učenje razvoja iger primerno, da učenje gradi na tem splošnejšem znanju.

Tu se nam postavi vprašanje: kako visoko začeti učenje razvoja iger s stališča programerja oziroma spoznavanje tehnologije iger? Končano igro sestavlja izvedba igralnosti, ki je specifična za izdelek in splošnejši deli, katere smo v treh nivojih (nizki, srednji, visoki) obravnavali v podpoglavju 1.2.1 o obsegu knjižnic za razvoj iger. Možno se je osredotočiti na učenje katerega-koli od teh delov, kar povzemamo v tabeli 1.3. Pri izbiri moramo upoštevati želeno zahtevnost učnega programa, njegovo dolžino in dejansko kvaliteto

sestavni deli igre	pristopi k učenju razvoja in teme
igralnost	izdelava igralnosti z vizualnim urejevalnikom, kombinacija uporabe urejevalnika in skript, klasično programiranje
visokonivojski pogon	programiranje grafičnega in fizikalnega pogona, organizacija scene, procesiranje vhodnih ukazov, iskanje poti, umetnointeligenčni agenti
srednjenivojska arhitektura	glavna zanka igre, izris 2D slik in 3D modelov z grafičnimi vmesniki, geometrijske strukture in matematične operacije nad njimi
nizkonivojske rutine	izdelava matematične knjižnice za linearno algebro, branje vhodnih slikovnih in 3D formatov, lasten programski izrisovalnik 3D prostora

Tabela 1.3: Učenje razvoja na različnih sestavnih delih igre

predhodnega znanja študentov. V našem primeru smo se za relativno kratek enosemestrski predmet na praktični smeri odločili za učenje visokega nivoja, torej izgradnje pogona igre in same igralnosti. Funkcionalnost nižjih stopenj mora biti zato študentom dana na voljo v obliki knjižnice srednjega nivoja.

Pri iskanju take knjižnice za učenje smo se odločali glede na naslednje kriterije:

- *Širina funkcionalnosti*: Želimo si eno samo knjižnico, ki pokriva vse osnovne zahteve iger (izris, zvok, vhod, uvoz vsebin). Možno je sicer uporabiti več ločenih knjižnic, da pokrijemo vsa potrebna področja, vendar enovita knjižnica predstavlja prednost s celovito zasnovano arhitekturo in stilom. To omogoča hitrejše in lažje delo, saj se ni potrebno toliko ukvarjati s samo zgradbo in načinom delovanja knjižnice.

- *Objektna orientiranost*: Čeprav se ravno razvoj iger zaradi stremenja k čim hitrejšemu izvajanju marsikje v svetu drži proceduralne paradigme, govorijo splošni trendi zaradi strmo naraščajoče kompleksnosti močno v prid objektno-orientiranemu programiranju [31]. Različni akterji v igri se zelo naravno preslikajo v objekte, kar predstavlja zelo primerno izbiro za arhitekturo same igre. S stališča izobraževanja je zato ugodno, da tudi knjižnica, ki jo uporabljamo za podlago, sledi načelom objektne orientiranosti.

- *Cena*: Prednost imajo projekti, ki so na voljo zastonj, saj je potrebno zagotoviti možnost dela za veliko število študentov, tako v učilnici, kot pri individualnem delu študentov.

- *Enostavnost razhroščevanja*: Pri učenju pogosto prihaja do napak, zato jih je potrebno čim enostavneje reševati. Še boljše je seveda, da se jih z

dobro zasnovo knjižnice čim več onemogoči. Dobra knjižnica mora na nepravilno uporabo jasno opozoriti in izpostaviti konkreten problem.

- *Dobra dokumentacija*: Da lahko študentje sami raziskujejo delovanje knjižnice in niso omejeni na prikazane primere, mora knjižnica nuditi dobro dokumentacijo funkcionalnosti razredov in metod v njej. Na ta način vzpodbujamo radovednost in omogočamo samostojno učenje.

Vsem omenjenim kriterijem zelo ustreza ogrodje XNA, zato ni čudno, da se pojavlja v vedno več izobraževalnih programih [32, 33, 34]. Ima pa XNA to slabost, da smo omejeni na razvoj za operacijski sistem Windows, Windows Phone 7 in konzolo Xbox 360. Sami smo potrebovali knjižnico za razvoj iger za okolje iOS, tako da je direktno XNA žal odpadel. Ker takratni edini način za uporabo XNAja na platformi iOS, XNATouch (današnji MonoGame), ni ustrezal našim kriterijem, smo se zopet zatekli k razvoju lastne knjižnice. Nastal je projekt XNI, zgrajen na podlagi programske kode spisane za igro Rings of Saturn, a ločen v lastno knjižnico, ki naj bi čimbolj direktno preslikal ogrodje XNA v Objective-C.

1.5 Pregled naloge

Skozi uvod smo spoznali razloge za nastanek ogrodja XNI, kot tudi vsa področja, ki se jih izdelava ogrodja za razvoj iger za iOS dotika. Glavni del naloge je posvečen konkretno razvoju ogrodja XNI in tehnologijam, s katerimi smo se srečali med razvojem.

Drugo poglavje obravnava načrt razvoja ogrodja. V veliki meri zaobjema analizo razlik med okoljem XNA in platformo iOS ter smernicami, ki so pri prenosu ogrodja XNA na iOS skrbele za sprejemanje netrivialnih odločitev.

V tretjem poglavju opisujemo konkretno izvedbo načrta, pri čemer smo naleteli na nove, konkretnejše probleme pri doseganju želene funkcionalnosti v novem okolju.

Četrto poglavje je namenjeno uporabi ogrodja XNI v praksi. V največji meri se je izkazalo pri izvajanju vaj predmeta TINR, omenjamo pa tudi primernost za profesionalni razvoj. Tako igre študentov, kot komercialna igra Monkey Labour so bili uspešno izdani v spletni trgovini App Store.

V zadnjem, petem poglavju zaključujemo s pridobljenimi izkušnjami, ugotovljenimi napakami in načrtovanimi popravki. V enem letu razvoja so se na področju razvoja iger za iOS pojavile nove spremembe, tako da je možnih

več smeri nadaljevanja. Za konec predstavljamo možnosti za nadaljnji, tudi tržni razvoj ogrodja.

Poglavje 2

Načrt prenosa ogrodja XNA v okolje iOS

Po sprejeti odločitvi, da se izdela novo knjižnico srednjega nivoja za okolje iOS tako, da se ogrodje XNA čimbolj dosledno prenese v Objective-C, je najprej nastopila faza raziskovanja značilnosti in razlik med obema okoljema skupaj z načrtovanjem arhitekture nove knjižnice ter postavljanjem smernic za prenos.

Na podlagi prvih ugotovitev se je ustvaril projekt in vzpostavilo razvojno okolje v katerem je bilo knjižnico mogoče razvijati, uporabljati v testnih projektih in izdajati za uporabo javnosti. Ko je bil osnovni razvojni proces vzpostavljen, smo se vrnili k raziskovanju in načrtovanju in ustvarili najmanjši možni nabor funkcionalnosti, da je že omogočal poganjanje iger z uporabo ogrodja XNI. Razvoj se je nadaljeval z novo iteracijo raziskovanja, načrtovanja, razvoja in izdaje. Podobno prepletanje je sledilo vse do konca razvoja.

Razvoj lahko označimo z iterativnim pristopom [35], glede na uporabo prototipne izvedbe v igri Rings of Saturn pa tudi z evolucijskim prototipiranjem [36]. Na ta način smo se predhodno srečali z večino problemov razvoja v iOS okolju, za katere smo skozi raziskovanje pred začetkom razvoja že našli ustrezne rešitve in zgradili načrt za precej stabilno arhitekturo, ki je kasneje ni bilo potrebno veliko popravljati.

2.1 Smernice za razvoj ogrodja XNI

Pred samim začetkom razvoja kakršnekoli programske opreme je potrebno imeti jasne cilje, kaj želimo z njo doseči [37]. Povzemimo glavne cilje in

zahteve naše knjižnice opisane v uvodu:

- omogoča razvoj iger za naprave z operacijskim sistemom iOS,
- pokriva ustrezna področja nižjih in srednjih nivojev funkcionalnosti, ki naj bi jih študentje že osvojili, oziroma so zelo tehnične narave, predvsem:
  - branje slikovnih datotek in izris 2D ter 3D računalniške grafike,
  - branje in predvajanje zvočnih in glasbenih datotek,
  - dostop do podatkov iz vhodnih naprav,
  - operacije linearne algebre nad vektorji in matrikami ter metode za njihovo grajenje,
- sledi načelom objektne orientiranosti,
- je zastonj in prosto dostopno,
- čimbolj preprečuje nepravilno rabo in omogoča enostavno razhroščevanje ter
- je dobro dokumentirana.

Ker je vsem, razen prvi točki, odlično odgovarjalo ogrodje XNA smo postavili glavno smernico, da bo ogrodje XNI čimbolj veren prenos ogrodja XNA. To pomeni, da ne bo enaka le funkcionalnost, temveč naj gre za čimbolj popolno enakost same arhitekture razredov obeh knjižnic. V praksi gre za 1-na-1 preslikovanje razredov, njihovih lastnosti in metod ogrodja XNA v razrede ogrodja XNI, razen tam, kjer razlike med jezikoma C# in Objective-C tega ne dovoljujejo.

To nas privede do druge glavne smernice, ki zahteva, da se pri preslikovanju upošteva standarde za razvoj v Objective-C [38]. V prvi meri se to kaže pri poimenovanju metod, ki so zaradi sintakse pošiljanja sporočil po vzoru SmallTalka razširjene z imeni parametrov metod v C#u.

Kasneje se je izkazalo, da bi bilo pri poimenovanju razredov bolje, da bi imela smernica standardov prednost pred čim večjo enakosti XNA. V Objective-C se zaradi pomanjkanja imenskih prostorov (angl. namespace) za eksterne knjižnice uporablja dodajanje predpon imenom razredov, da ne prihaja do prekrivanja z uporabnikovimi pa tudi z obstoječimi razredi. Potreba po uporabi tega pravila se je pokazala šele pozno v razvoju, tako da smo le pri

nekaterih imenih uporabili predpono Xni, medtem ko so imena večine razredov enaka tistim v XNA. To je za uporabnika moteče, saj po prvem pravilu pričakuje, da mora uporabiti razred iz istim imenom, a mora biti pozoren še na izjeme s predpono. Tako bi bilo bolje, da bi vsi razredi nosili predpono, kar bi dodatno imelo smisel za uporabnike knjižnice, ki izvirajo direktno iz Objective-C okolja.

Tretja glavna smernica, ki pokriva preostale netrivialne probleme pri prenosu, je stremenje k čim lažji in za učenje primerni uporabi knjižnice. Smernica služi izobraževalnim ciljem knjižnice in je vzrok za marsikatero odločitev, ki bi bila drugačna, če bi recimo dali prednost čim hitrejšemu izvajanju. Enostavna uporaba je dodatno primernejša za hiter, prototipni razvoj, ki predstavlja pomemben pristop k razvoju iger [39, 40].

2.2 Pregled arhitekture ogrodja XNA

Microsoftova knjižnica XNA je namenjena razvoju iger za operacijska sistema Windows, Windows Phone 7 in konzolo Xbox 360. Spisana je v programskem jeziku C# z uporabo ogrodja .NET in grafične knjižnice DirectX.

Trenutno je ogrodje XNA v verziji 4.0, ki je prineslo precejšnjo izboljšavo arhitekture razredov. Ogrodje se deli v več ločenih dinamičnih knjižnic (datotek DLL) opisanih v tabeli 2.1. Pri tem velja, da so knjižnice s Content.Pipeline v imenu na voljo le med razvojem v Windows okolju. Uporabijo se v koraku predprocesiranja vhodnih datotek in niso na voljo med izvajanjem na ciljni platformi.

Z drugega vidika lahko arhitekturo knjižnice XNA vidimo skozi imenske prostore. Ti so za izvršilni del zbrani v tabeli 2.2. Cevovod vsebine (angl. content pipeline) za predprocesiranje ima dodatne imenske prostore, ki so razvidni iz tabele 2.3. Skozi oba pregleda postane jasno, da se knjižnica deli na dva velika dela: izvršni del in cevovod vsebine. Cevovod vsebine je nabor procesov, ki se izvedejo na podatkovnih datotekah vsebine že med prevajanjem igre. Ti procesi naložijo izvirne podatke iz vhodnih datotek in jih spremenijo v obliko, kot jo potrebuje ogrodje XNA kasneje med izvajanjem igre. Glavni razlog za to je hitrost nalaganja igre med izvajanjem, saj bi sicer vse te postopke igra morala izvajati medtem, ko igralec čaka na zagon igre. Obstajajo pa tudi druge prednosti [41].

knjižnica	vsebina
Microsoft.Xna.Framework.dll	glavno jedro knjižnice
Microsoft.Xna.Framework.Avatar.dll	prikaz avatarjev storitve Xbox Live
Microsoft.Xna.Framework.Game.dll	ogrodje arhitekture igre in komponent
Microsoft.Xna.Framework.Graphics.dll	del povezan z grafiko preko DirectXa
Microsoft.Xna.Framework.Input.Touch.dll	dotikovni vhodni vmesnik
Microsoft.Xna.Framework.Net.dll	povezovanje več igralcev preko omrežja
Microsoft.Xna.Framework.Storage.dll	shranjevanje podatkov
Microsoft.Xna.Framework.Video.dll	predvajanje video datotek
Microsoft.Xna.Framework.Xact.dll	predvajanje z orodjem Xact pripravljenih zvočnih datotek
Microsoft.Xna.Framework.Content.Pipeline.dll	ogrodje za predprocesiranje vhodnih datotek
Microsoft.Xna.Framework.Content.Pipeline. AudioImporters.dll	branje zvočnih datotek
Microsoft.Xna.Framework.Content.Pipeline. EffectImporter.dll	branje senčilnikov
Microsoft.Xna.Framework.Content.Pipeline. FBXImporter.dll	branje 3D modelov v formatu FBX
Microsoft.Xna.Framework.Content.Pipeline. TextureImporter.dll	branje slikovnih datotek
Microsoft.Xna.Framework.Content.Pipeline. VideoImporter.dll	branje video datotek
Microsoft.Xna.Framework.Content.Pipeline. XImporter.dll	branje 3D modelov tipa v formatu X

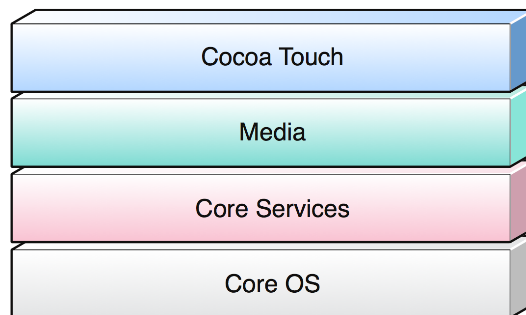
Tabela 2.1: Dinamične knjižnice ogrodja XNA

imenski prostor	vsebina
Microsoft.Xna.Framework	glavni nabor razredov
Microsoft.Xna.Framework.Audio	programski vmesnik za delo z zvokom na nizkem nivoju, pripravljenim z orodjem Xact
Microsoft.Xna.Framework.Content	nalaganje predprocesiranih vhodnih datotek med izvajanjem
Microsoft.Xna.Framework.Design	razredi za pomoč pretvarjanju med programskimi tipi
Microsoft.Xna.Framework.GamerServices	delo s storitvijo Xbox Live
Microsoft.Xna.Framework.Graphics	API za delo z grafiko na nizkem nivoju
Microsoft.Xna.Framework.Graphics.PackedVector	posebni grafični podatkovni tipi
Microsoft.Xna.Framework.Input	branje podatkov iz klasičnih vhodnih naprav
Microsoft.Xna.Framework.Input.Touch	branje podatkov dotikovnega vmesnika
Microsoft.Xna.Framework.Media	predvajanje večpredstavnih datotek
Microsoft.Xna.Framework.Net	razredi za povezovanje več igralcev prek omrežja
Microsoft.Xna.Framework.Storage	razredi za branje in shranjevanje med igro ustvarjenih podatkov

Tabela 2.2: Imenski prostori izvršnega dela ogrodja XNA

imenski prostor	vsebina
Microsoft.Xna.Framework.Content.Pipeline	glavni nabor razredov
Microsoft.Xna.Framework.Content.Pipeline.Audio	podatkovni tipi za zvočne podatke
Microsoft.Xna.Framework.Content.Pipeline. Graphics	grafični podatkovni tipi
Microsoft.Xna.Framework.Content.Pipeline. Processors	razredi za predprocesiranje podatkov
Microsoft.Xna.Framework.Content.Pipeline. Serialization.Compiler	razredi za prevajanje in zapisovanje podatkov
Microsoft.Xna.Framework.Content.Pipeline. Serialization.Intermediate	razredi za zapisovanje vmesnih podatkov v XML
Microsoft.Xna.Framework.Content.Pipeline.Tasks	razredi za izvajanje zapisovanja v končni binarni format

Tabela 2.3: Imenski prostori cevovoda vsebine XNA



Slika 2.1: Sloji operacijskega sistema iOS

2.3 Pregled arhitekture iOS

Na drugi strani imamo Applov operacijski sistem iOS. Jedro sistema je spisano v jeziku C, knjižnice na višjih nivojih večinoma v Objective-C. Celotna arhitektura se deli na štiri nivoje, kot je prikazano na sliki 2.1 [15].

Na najvišjem nivoju Cocoa Touch najdemo vrsto uporabnih ogrodij. Med njimi sta za igre pomembni predvsem:

- UIKit, glavno ogrodje za izdelavo aplikacij v iOSu s podporo za uporabniški vmesnik, večopravilnost, dotikovni vmesnik in druge vhodne naprave (pospeškomer, kompas, GPS ...) ter
- GameKit, razredi za delo z večigralstvom in storitvijo Game Center.

Nivo nižje leži Media z ustreznimi programskimi vmesniki za delo z grafiko, zvokom in videom. Za nas so pomembne naslednji deli:

- AV Foundation, predvajanje multimedijskih vsebin,
- OpenAL, delo s prostorskim zvokom in
- OpenGL ES, hiter izris 2D in 3D grafike.

Iz nivoja Core Services sta pomembnejša le vmesnika Foundation in Core Foundation (Objective-C vmesnik do Foundation storitev), ki prinašata osnovni nabor podatkovnih struktur (nizi, zaporedja, slovarji) in druge osnovne funkcionalnosti s katerimi gradimo aplikacije.

Z vsemi omenjenimi knjižnicami skupaj smo v Objective-Cju lahko reproducirali funkcionalnost knjižnice XNA. Potrebno je bilo zgraditi tudi nekaj razredov samega ogrodja .NET, katerim v arhitekturi iOS nismo našli direktne ustreznice, kot tudi lastnosti samega jezika C#, predvsem sistem dogodkov (angl. events). Podrobna analiza razlik sledi v naslednjih podpoglavjih.

2.4 Razlike med programskima jezikoma C# in Objective-C

Do zdaj smo pregledali želene funkcionalnosti ogrodja XNA in nabor knjižnic v sistemu iOS, s katerimi smo to funkcionalnost ponovno spisali. Ogledati si moramo še razlike med izvornim programskim jezikom C#, ciljnim Objective-C in njunima osnovnima knjižnicama (imenski prostor System iz ogrodja .NET in Core Foundation iz iOS), saj so te najbolj vplivale na potek prenosa.

2.4.1 Imenski prostori

Prva razlika med organizacijo razredov med jezikoma je uporaba imenskih prostorov v C#u in uvažanje zaglavnih datotek (angl. header file) v Objective-Cju. Medtem ko v C#u zgolj uporabimo

```
1 using Microsoft.Xna.Framework;
```

da se lahko sklicujemo na vse razrede iz tega imenskega prostora, je v Objective-Cju potrebno posamično uvoziti zaglavne datoteke vseh drugih razredov, ki jih želimo uporabljati. Za uporabo vektorjev in matrik, bi morali recimo uporabiti:

```
1 #import "Vector3.h"
2 #import "Matrix.h"
```

Na srečo se da te stavke združiti v svojo zaglavno datoteko, tako da imamo na enem mestu smiselno združene vse razrede, ki jih uporabljamo skupaj. XNI za vsak izvirni imenski prostor XNAja ponuja pripravljeno datoteko, ki uvaža vse razrede tega imenskega prostora. Uporaba v ciljni igri je tako enako enostavna kot `using` stavki v C#u, saj uvozimo zgolj datoteko, ki uvozi vse razrede iz želenega imenskega prostora (njena vsebina je v izpisu 2.1):

```
1 #import "Retronator.Xni.Framework.h"
```

Če tudi pri samih razredih znotraj imenskega prostora uporabljamo te zaglavne datoteke naletimo na problem. Ker pri definiciji vektorja uporabljamo matrike in pri metodah matrike vektorje, bi načeloma morala vsaka zaglavna datoteka uvoziti drugo. To bi ustvarilo krožno odvisnost, katere prevajalnik

```
#import "FrameworkEnums.h"
#import "Protocols.h"

// Data structures
#import "XniPoint.h"
#import "Rectangle.h"
#import "Color.h"

// Game
#import "IGraphicsDeviceManager.h"
#import "Game.h"
#import "GameTime.h"
#import "GameServiceContainer.h"
#import "GraphicsDeviceManager.h"

// Game host
#import "GameHost.h"
#import "GameWindow.h"

// Game components
#import "IGameComponent.h"
#import "IUpdatable.h"
#import "IDrawable.h"
#import "GameComponent.h"
#import "DrawableGameComponent.h"
#import "GameComponentCollection.h"
#import "GameComponentCollectionEventArgs.h"

// Linear algebra
#import "Vector2.h"
#import "Vector3.h"
#import "Vector4.h"
#import "Quaternion.h"
#import "Matrix.h"
```

Izpis 2.1: Vsebina zaglavne datoteke Retronator.Xni.Framework.h, ki uvozi razrede iz imenskega prostora

```
1  #import "FrameworkEnums.h"
2  @class Protocols;
3  
4  // Data structures
5  #import "PointStruct.h"
6  #import "RectangleStruct.h"
7  @class XniPoint, Rectangle, Color;
8  
9  // Game
10 @protocol IGraphicsDeviceManager;
11 @class Game, GameTime, GameServiceContainer, ↩
       GraphicsDeviceManager;
12 
13 // Game host
14 @class GameHost, GameWindow;
15 
16 // Game components
17 @protocol IGameComponent, IUpdatable, IDrawable;
18 @class GameComponent, DrawableGameComponent, ↩
       GameComponentCollection, GameComponentCollectionEventArgs↩
       ;
19 
20 // Linear algebra
21 #import "Vector2Struct.h"
22 #import "Vector3Struct.h"
23 #import "Vector4Struct.h"
24 #import "MatrixStruct.h"
25 @class Vector2, Vector3, Vector4, Quaternion, Matrix;
```

Izpis 2.2: Vsebina zaglavne datoteke Retronator.Xni.Framework.classes.h, ki napove razrede iz imenskega prostora

ne bi mogel razrešiti [42]. Za ta namen se v zaglavni datoteki razreda le napove uporabo drugih razredov s `@class` direktivo, recimo:

```
1  @class Vector3, Matrix;
```

Poleg datoteke z združenimi `#import` stavki smo pripravili še datoteko z združenimi `@class` direktivami. Te uvažamo v zaglavnih datotekah znotraj ogrodja XNI, podobno kot v prejšnjem primeru:

```
1  #import "Retronator.Xni.Framework.classes.h"
```

V izpisu 2.2 opazimo, da podobno kot `@class` obstaja direktiva `@protocol` za napoved vmesnika (angl. interface). Razne Cjevske ukaze, recimo naštevanja (`enum`), podatkovne strukture (`struct`) in definicije (`define`), je potrebno uvoziti v obeh vrstah združenih zaglavnih datotek.

Zadnja izjema pri našem sistemu so dedovani razredi. V tem primeru `@class` direktiva v zaglavni datoteki ne zadostuje, saj mora definicija otroka vsebovati točno definicijo starša [42]. Pri dedovanju moramo zato ločeno uporabiti uvoz zaglavne datoteke starševskega razreda. Zaradi enosmernosti dedovanja v tem primeru ne more priti do problema krožnih uvažanj.

Na ta način smo pri gradnji ogrodja XNI povsem nadomestili imenske prostore.

2.4.2 Zasebne in notranje metode ter lastnosti

Eno od pomembnih načel objektno-orientiranega razvoja je skrivanje informacij (angl. information hiding) o delovanju objekta pred zunanjim uporabnikom [43]. V C#u ga dosežemo z zasebnimi (angl. private) metodami in z notranjim modifikatorjem (angl. internal). Ta metodo ali razred naredi nevidnega za zunanje programske komponente (angl. assembly), ne glede na siceršnjo vidnost znotraj komponente. To je pomemben mehanizem, kako skrijemo javne in zaščitene (angl. protected) metode ter lastnosti razredov programske knjižnice, ki so potrebne za njeno delovanje, ne pa za uporabnika knjižnice [44].

Objective-C sicer pozna koncept javnih, zaščitenih in zasebnih spremenljivk, ne pa tudi metod [45]. Pri tem so zasebne spremenljivke vidne končnemu uporabniku v javni zaglavni datoteki. Za skrivanje informacij moramo tako poseči po različnih tehnikah, ki lahko zahtevajo preveč dodatnega truda, da bi opravičile uporabo. Skrivanje zasebnih spremenljivk iz javnih zaglavnih datotek se sicer da doseči z dvojnimi izvedbenimi razredi [46], a uporabnik nima prave koristi od tega. Zasebne in notranje metode ter lastnosti se nam po drugi strani splača skrivati vsaj zato, da jih sistemi za lajšanje programiranja (angl. code completion) ne ponujajo uporabniku.

Skrivanje zasebnih in notranjih metod dosežemo z uporabo razširitev razredov (angl. categories). Zasebne metode navedemo v nepoimenovani razširitvi (angl. anonymous category), ki jo dodamo v izvedbeno (angl. implementation) datoteko razreda [47]. Ker so uporabniku dostopne samo zaglavne datoteke, ne bo dobil definicij na ta način skritih metod.

Za notranje metode smo to tehniko nekoliko spremenili. Ker gre za javne metode, ki morajo biti dostopne iz izvedbenih datotek drugih razredov, jih

ne smemo skriti v izvedbeno datoteko. Namesto tega izdelamo dodatno zaglavno datoteko, kjer z istim postopkom razširitve osnovnemu razredu dodamo notranje metode in lastnosti. To datoteko lahko zdaj vključujemo iz drugih izvedbenih datotek, ne dodamo pa je med zaglavne datoteke ciljnega produkta (angl. public headers). Tako se ne prenesejo končnim uporabnikom, katerim je vidnost metod in lastnosti ponovno skrita.

2.4.3 Strukture

Pomembna prednost C#a, ki je v Objective-Cju žal ne moremo enakovredno nadomestiti, so strukture. Te so v C#u precej enakovredne razredom. Tako strukturam kot razredom lahko definiramo polja, lastnosti in metode, razlika pa je, da se razredi prenašajo po referenci, strukture pa po vrednosti. Dodatno se razredi ustvarijo na pomnilniški kopici, strukture pa na skladu [48]. Tako so le-te enakovredne primitivnim tipom in ne potrebujejo ločene dodelitve in sprostitve pomnilnika. Seveda ima uporaba struktur tudi svoje slabosti. Ker se prenašajo v funkcije po vrednosti, se mora na sklad vedno znova kopirati celotna vsebina strukture [49]. Na drug problem naletimo pri shranjevanju struktur v spremenljivke, ki pričakujejo objekt. Potreben je dodatni korak shranjevanja v začasni objekt (angl. boxing), kar ustvari objekt na kopici in obremenjuje čiščenje pomnilnika (angl. garbage collection) [50]. Ker vse to vpliva na hitrost izvajanja, je pravilna uporaba struktur in razredov zelo pomembna pri programiranju iger.

V Objective-Cju lahko potegnemo vzporednico s C#om le glede objektov. Strukture pozna iz Cja, kjer služijo zgolj združevanju podatkov, ne moremo pa jim na objektno-orientirani način dodajati metod. Največ kar lahko naredimo je, da za vsako metodo strukture v C#u naredimo Cjevsko funkcijo, ki ji kot enega izmed parametrov podamo podatkovno strukturo, na kateri naj izvede operacijo.

Na začetku načrtovanja XNI smo zastavili smernico, da zagotovimo čimbolj veren arhitekturni prenos ogrodja in da damo zaradi izobraževalne narave prednost enostavnosti uporabe pred hitrostjo izvajanja. Zaradi tega smo v ogrodju XNI skoraj vse strukture iz C#a nadomestili z razredi v Objective-Cju. Pri tem smo v ozadju vseeno najprej izdelali Cjevske strukture, ki nosijo vse potrebne podatke in Cjevske funkcije, za osnovne operacije nad temi podatki. Razredi nosijo podatke v teh strukturah in pri večini operacijah nad podatki kličejo funkcije v Cju.

Uporabnik ogrodja XNI na ta način ne vidi razlik med strukturami in razredi, enako, kot lahko v XNAju z obema dela na objektno-orientiran način.

Če kasneje pride do performančnih problemov, lahko z direktno uporabo Cjevskih struktur in funkcij še vedno zaobide strošek dela z razredi in pohitri izvajanje.

2.4.4 Zbirke s statičnim tipom

Še ena razlika med jezikoma je v statični zasnovi C#a napram dinamičnosti Objective-Cja. V standardnih knjižnicah obeh okolij najdemo zbirke (angl. collections) za delo z objekti poljubnega tipa. V Objective-Cju to zadostuje, saj ni potrebe po eksplicitnemu spreminjanju tipa (angl. casting), če splošni kazalec (id, skrajšano iz angl. identifier) priredimo spremenljivki z določenim tipom. C# je bil po drugi strani striktno statičen. Pri delu z zbirkami je bilo potrebno vedno ročno dodajati spremembo tipa. S časom so razvili možnost splošnih razredov (angl. generics), ki se avtomatsko prevedejo v verzijo za želeni podatkovni tip. Zbirke so lahko nadomestili s splošnimi verzijami, pri katerih se ob definiranju spremenljivke določi, kakšen tip objektov bo nosila [51]. XNA ta način v veliki meri uporablja in gre celo en korak dlje. Iz splošnih zbirk z dedovanjem ustvarja razrede, ki še semantično določajo namen zbirke. Na ta način je napačna raba ogrodja dvojno zavarovana: najprej s statično določenim tipom podatkov v zbirki in nato še s statično določenim tipom zbirke.

Ker je tudi pri izdelavi ogrodja XNI preventiva napačne rabe ena od smernic, smo za vse take zbirke izdelali razrede, ki statično omejujejo tip hranjenih podatkov. V notranjosti sicer še vedno uporabljajo za hranjenje običajne zbirke iz knjižnice Core Foundation z uporabo vzorca poverjanja (angl. delegation pattern).

Da ne bi bilo potrebno vedno znova programirati preslikave, smo izrabili možnost uvoza (angl. import) datotek in nadomestili pomanjkanje splošnih razredov z izdelavo predlog (angl. templates). Za splošni zbirki (običajno in samo-za-branje verzijo) smo pripravili okostje implementacije, ki podobno kot splošni tipi uporablja spremenljivko za podatkovni tip. V ciljnem razredu ji z definicijo določimo vrednost želenega tipa in uvozimo datoteko z implementacijo. Enostavni primer uporabe predloge je prikazan v izpisu 2.3.

Ta način žal ne more povsem nadomesti splošnih razredov. Primeren je pri dedovanju iz splošnih razredov, ne pa tudi pri njihovi uporabi kot statično določenemu, a dinamično sprogramiranemu podatkovnemu tipu. XNA se na srečo večinoma poslužuje prvega načina, tako da smo s to rešitvijo na več mestih ustregli našim smernicam.

```
// Izvirna definicija razreda v XNA (C#)

public sealed class ModelEffectCollection :
                        ReadOnlyCollection<Effect>

// In njegovega starša

public class ReadOnlyCollection<T> {
    private IList<T> items;

    public ReadOnlyCollection(IList<T> list)

    public int Count { get; }

    public T this[int index] { get; }
}


// Definicija razreda v XNI
// ModelEffectCollection.h

#define ReadOnlyCollection ModelEffectCollection
#define T Effect*

#include "ReadOnlyCollection.h"

#undef ReadOnlyCollection
#undef T


// Definicija predloge
// ReadOnlyCollection.h

@interface ReadOnlyCollection : NSObject {
    NSArray *items;
}

- (id) initWithList:(NSArray*)list;

@property (nonatomic, readonly) int count;

- (T)itemAt:(int)index;

@end
```

Izpis 2.3: Nadomestitev dedovanja iz splošnih razredov na posplošenem primeru razreda ModelEffectCollection

2.4.5 Dogodki

C# za komunikacijo med razredi eksplicitno določa sistem dogodkov (angl. events) po vzorcu opazovalca (angl. observer pattern). Na dogodke se objekti naročijo, da jih opazovani objekt obvesti, da je na določeni točki izvajanja. Sistem dogodkov v C#u je zgrajen na osnovi delegatov (angl. delegates), ki so v praksi kazalci na funkcije [52]. Podobno vlogo imajo v Objective-Cju izbiralci (angl. selectors) [42] in z njimi lahko ročno sami dodamo podoben sistem dogodkov. XNI razpolaga z razredoma Delegate in Event, s katerima so izvedeni dogodki iz izvirnih razredov.

2.5 Razlike med okoljema Direct3D in OpenGL

Nekaj pomembnih razlik med ogrodjema XNA in XNI nastane tudi zaradi dveh konkurenčnih grafičnih knjižnic, na katerih sta zgrajeni: XNA deluje z Microsoftovo knjižnico Direct3D, XNI z odprtokodnim standardom OpenGL ES.

2.5.1 Programabilni senčilniki

Pod iOSom sta na voljo dve verziji knjižnice OpenGL ES. Verzija 1.1 omogoča starejši pristop nespremenljivega grafičnega cevovoda (angl. fixed shader pipeline). Verzija 2.0 prinaša programabilne senčilnike (angl. programmable shader pipeline) s podporo jeziku GLSL [27]. Na drugi strani XNA že od začetka zahteva grafično kartico z možnostjo programabilnih senčilnikov [53]. Tako je celotna grafična arhitektura v XNA zgrajena na modelu senčilnikov. Da bi razvijalcem olajšali prestop z nespremenljivega načina, so poskrbeli za predpripravljene senčilnike in razrede za delo z njimi. Glavni med njimi je razred BasicEffect, ki ustreza osnovnim možnostim, ki so na voljo v nespremenljivem grafičnem cevovodu [54, 55].

XNI smo spisali z OpenGL ES verzijo 1.1. Razlogi so bili naslednji:

- delo z knjižnico OpenGL ES med verzijama 1.1 in 2.0 je precej različno, naš prototip pa je temeljil le na verziji 1.1,
- nismo imeli predhodnih izkušenj s pisanjem senčilnikov GLSL, da bi lahko hitro spisali nadomestek potrebnih HLSL senčilnikov in
- za začetnike v 3D grafiki povsem zadostuje razred BasicEffect, ki ga je mogoče v celoti spisati v nespremenljivem grafičnem cevovodu.

Zaradi te odločitve pri prenašanju razredov grafičnega področja nismo prenesli funkcionalnosti, ki se kakorkoli tiče senčilnikov.

2.5.2 Urejenost matrik in transformacija med koordinatnimi sistemi

Da pri izrisu 3D grafike dobimo pravilen rezultat, se moramo zavedati, na kakšen način so urejene matrike na eni in drugi strani, ter kako grafična knjižnica z matrikami transformira vektorje med različnimi koordinatnimi sistemi v grafičnem cevovodu.

OpenGL ES verzije 1.1 je spisan glede na specifikacijo običajnega OpenGL verzije 1.5. Iz odgovorov na pogosta vprašanja izvemo, da OpenGL pričakuje podatke o premiku (angl. translation) na 13. do 15. mestu 16-mestne tabele, kot je označena v specifikaciji [56]. Ta pravi, da operacija nalaganja matrik pričakuje podatke urejene po zaporednih stolpcih (angl. column-major order). Iz obojega izvemo, da OpenGL pričakuje vektorje matrike shranjene v stolpce in tudi zapis 2D matrike v 1D tabelo po zaporednih stolpcih. Dodatno sta iz specifikacije pomembni informaciji, da se pri transformacijah med prostori matrike množijo z leve, pri operaciji množenja pa z desne strani [57].

XNA na drugi strani shranjuje matrike z vektorji urejenimi v vrstice [58], matrike pa se vedno množijo z desne [59]. Da bo vsebina matrik med ogrodjema XNA in XNI povsem enaka, tudi v XNIju vektorje shranjujemo v vrstice matrike (vektor za premik ohranimo zapisan v četrti vrstici matrike). Hkrati želimo, da OpenGL prejme podatke v pričakovanem vrstnem redu. Pomnilniško strukturo moramo zato definirati v zaporedju vrstic (angl. row-major order). Tako se bodo omenjeni podatki o premiku iz četrte vrstice zopet znašli na 13. do 15. mestu 16-mestne tabele podatkov matrike, kot to pričakuje OpenGL. Enak rezultat bi dobili tudi, če bi uporabili zapis v zaporedju stolpcev in bi pri nalaganju matrik v OpenGL uporabljali ukaz `LoadTransposeMatrix` namesto `LoadMatrix`.

Zaradi množenja z leve pri transformacijah v OpenGLu, moramo biti na to pozorni pri sestavljanju matrike $\mathbf{ModelView}$, ki združuje postavitev objekta v svetu ($\mathbf{World}$) s pogledom ($\mathbf{View}$). Z matriko $\mathbf{ModelView}$ OpenGL iz koordinat v prostoru objekta (angl. object coordinates) izračuna koordinate v prostoru pogleda (angl. eye coordinates).

V XNAju se kot rečeno pri transformacijah vektorje množi z desne (zato morajo ti biti zapisani v vrsticah (angl. row vectors)), tako da opisana trans-

formacija zgleda takole:

$$\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{ModelView}_{\mathbf{XNA}} = \vec{\mathbf{v}}_{\mathbf{eye}} \tag{2.1}$$

Matriko $\mathbf{ModelView}_{\mathbf{XNA}}$ pripravimo enostavno z množenjem matrik $\mathbf{World}$ in $\mathbf{View}$ saj:

$$\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{World} = \vec{\mathbf{v}}_{\mathbf{world}} \tag{2.2}$$

$$\vec{\mathbf{v}}_{\mathbf{world}} \cdot \mathbf{View} = \vec{\mathbf{v}}_{\mathbf{eye}} \tag{2.3}$$

$$\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{World} \cdot \mathbf{View} = \vec{\mathbf{v}}_{\mathbf{eye}} \tag{2.4}$$

OpenGL bo izvedel opisano transformacijo nasprotno, z množenjem z leve, pri čemer so vektorji zapisani v stolpcih (angl. column vectors), torej transponirano glede na XNA:

$$\mathbf{ModelView}_{\mathbf{GL}} \cdot \vec{\mathbf{v}}^{\mathbf{T}}_{\mathbf{object}} = \vec{\mathbf{v}}^{\mathbf{T}}_{\mathbf{eye}} \tag{2.5}$$

V OpenGLu moramo tako matriko $\mathbf{ModelView}_{\mathbf{GL}}$ pripraviti z zmnožkom $\mathbf{View}^{\mathbf{T}} \cdot \mathbf{World}^{\mathbf{T}}$, kar sledi iz izpeljave:

$$\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{ModelView}_{\mathbf{XNA}} = \vec{\mathbf{v}}_{\mathbf{eye}} \tag{2.6}$$

$$\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{World} \cdot \mathbf{View} = \vec{\mathbf{v}}_{\mathbf{eye}} \tag{2.7}$$

$$(\vec{\mathbf{v}}_{\mathbf{object}} \cdot \mathbf{World} \cdot \mathbf{View})^{\mathbf{T}} = (\vec{\mathbf{v}}_{\mathbf{eye}})^{\mathbf{T}} \tag{2.8}$$

$$\mathbf{View}^{\mathbf{T}} \cdot \mathbf{World}^{\mathbf{T}} \cdot \vec{\mathbf{v}}^{\mathbf{T}}_{\mathbf{object}} = \vec{\mathbf{v}}^{\mathbf{T}}_{\mathbf{eye}} \tag{2.9}$$

XNI tako upošteva, da od uporabnika dobi matriki $\mathbf{World}$ in $\mathbf{View}$ z vektorji v vrsticah, kot je to običajno za XNA. Ko jih naloži skozi klice OpenGL, jih ta reinterpretira v matrike z vektorji v stolpcih (transponirane glede na XNA). Izračun matrike $\mathbf{ModelView}$ se tako pravilno izvede z naslednjim množenjem:

```
1 glMatrixMode(GL_MODELVIEW);
2 glLoadMatrixf((float*)basicEffect.view.data);
3 glMultMatrixf((float*)basicEffect.world.data);
```

Poglavje 3

Razvoj ogrodja XNI

S postavljenimi smernicami in analiziranimi glavnimi razlikami med izvornim in ciljnim okoljem, smo začeli z iterativnim razvojem ogrodja. V tem poglavju opisujemo rešitve problemov pri izdelavi ogrodja, ki ne izvirajo iz vnaprej predvidenih razlik med okoljema, temveč se nanašajo predvsem na tehnično stran izdelave.

3.1 Kreiranje projekta v Xcode

Za izdelavo aplikacij (in knjižnic) za iOS smo potrebovali [14]:

- Mac računalnik z Intel procesorjem in operacijskim sistemom Mac OS X ter
- iOS SDK, ki vključuje [15]:
    - ogrodja in standardne knjižnice,
    - razvojno okolje Xcode z urejevalnikom grafičnega uporabniškega vmesnika Interface Builder,
    - orodje za testiranje Instruments,
    - iOS Simulator, simulator naprav iPhone in iPad, ter
    - iOS Developer Library, zbirko dokumentacije za razvoj.

V programu Xcode smo projekt XNI ustvarili s predlogo statične knjižnice. Njeno programiranje sicer ne zahteva posebnih razlik od dela v izvršilnih projektih. V projekt smo skozi celoten razvoj dodajali potrebne datoteke za

izvedbo Objective-C razredov in ostalo programsko kodo v Cju. Pomembno je le, da smo zaglavnim datotekam, ki morajo biti na voljo končnemu uporabniku, nastavili vidnost na javno (public). Tako se v koraku kopiranja javnih zaglavnih datotek med grajenjem statične knjižnice prenesejo v ciljno mapo poleg zgrajene knjižnjice.

3.2 Razvoj osnovne arhitekture

V pregledu arhitekture knjižnic okolja iOS smo videli, da bomo morali osnovni del ogrodja XNI zgraditi z ogrodjem UIKit, namenjenim izdelavi aplikacij v iOSu. Poglejmo si tehnične detajle izvedbe osnovne arhitekture.

Glavni razred za izdelavo igre v ogrodju XNA je razred `Game` iz katerega z dedovanjem zgradimo svojo igro [60]. Po vstopni točki v program v proceduri `Main` se pripravi instanco končnega razreda igre in na njem pokliče metodo `Run`, ki se ne vrne, dokler uporabnik ne zapre aplikacije. Od osnovne arhitekture poleg razreda `Game` obstaja le še razred `GameWindow`, ki predstavlja abstrakcijo okna v katerem teče igra znotraj gostujočega operacijskega sistema. Metoda `Run` poskrbi, da operacijski sistem pripravi okno, v katerem bo igra tekla, in požene igrino zanko. Zanka skrbi za sprejemanje sporočil operacijskega sistema (angl. message pumping) in klicanje procesiranja ter izrisa ciljne igre v želenih časovnih intervalih.

3.2.1 Game in GameHost

Za izvedbo opisane arhitekture smo morali določene detajle izvedbe prirediti, saj UIKit deluje nekoliko drugače. Tudi tu imamo vstopno točko v proceduri `main`, ki preko funkcije `UIApplicationMain` ustvari glavni razred aplikacije (angl. principal class) in posrednika (angl. delegate class). Glavni razred mora dedovati iz razreda `UIApplication` (lahko se uporabi tudi direktno tega, kar je tudi priporočeno), posrednik pa mora uporabiti vmesnik `UIApplicationDelegate`. Uporaba posrednikov je na splošno Applov način, kako se izogniti dedovanju iz kompleksnih razredov kot je `UIApplication` . Namesto tega spišemo svoj razred, ki uporabi dani vmesnik posrednika, preko katerega bo `UIApplication` obveščal naš razred o pomembnih dogodkih ali zahteval določeno delovanje [61].

Ogrodje XNA v ozadju prav tako loči logiko poganjanja igre (razred `Game`) od priprave sistema (razred `GameHost`). Da bi čimbolj poustvarili to dvonivojsko delitev, smo se odločili da `GameHost` v XNI deduje iz `UIApplication`,

`Game` pa uporabi vmesnik `UIApplicationDelegate`.

3.2.2 Izvedba igrine zanke

Najbolj značilni mehanizem v vsaki igri je igrina zanka, ki skrbi za interaktivnost igrinega sveta z igralcem. Izmenično sprejema uporabnikove ukaze, posodablja svet in mu novo stanje vrne predvsem preko izrisa na zaslon in z zvokom ter to zaporedje ponavlja do zaustavitve [3].

XNAjev razred `Game` razpolaga z dvema načinoma delovanja te zanke [60]. V privzetem načinu enakomernih časovnih korakov (angl. fixed-step game loop) uporabnik določi, v kakšnih intervalih naj igra kliče uporabnikove metode za posodabljanje (angl. `update`) in izris (angl. `draw`) igre. Zanka tako preda uporabniku nadzor, da izvede ti dve metodi, nakar primerja dejanski porabljen čas z želenim intervalom in proces zaustavi za manjkajočo razliko. Če je porabljen čas večji od želenega intervala je ta čas negativen in zaustavitev ni potrebna. Takrat rečemo, da igra teče počasi, kar razred `Game` tudi sporoča preko lastnosti `isRunningSlow`.

Drugi način delovanja zanke je izvajanje brez prekinitev (angl. variable-step game loop). Enako kot v prejšnjem primeru igra kliče metode za posodabljanje in izris, a na koncu ne doda spanja procesa, temveč takoj začne z novim obhodom zanke. Igra v tem primeru v celoti zaposli procesorsko jedro in teče z najvišjim možnim številom slik na sekundo (angl. FPS — frames per second).

Zaradi razumevanja možnosti izvedbe igrine zanke v okolju iOS moramo omeniti še eno od lastnosti, ki je v XNAju povezana z njenim izvajanjem. Razred `GraphicsDeviceManager` razpolaga z nastavitvijo `SynchronizeWithVerticalRetrace`, kar pomeni, da igra prikaz izrisane slike počne istočasno s strojnim intervalom osveževanja zaslona [62].

Sinhronizacija z izrisom je tudi priporočljiv način za izvedbo igrine zanke v iOSu. Ker `UIApplication` že ima svojo zanko za izvajanje sporočil operacijskega sistema (angl. run loop), moramo uporabiti le mehanizem, ki v to zanko doda klic naše metode. Sprva se je za to uporabljal razred `NSTimer`, ki je lahko izvedel določeno metodo v želenem časovnem intervalu. Njegova slabost je (ne)natančnost, ki je med 50 in 100 ms [63], medtem ko igrina zanka običajno zahteva interval dolg samo 17 ms (60 FPS). V iOS verziji 3.1 so zato dodali boljši razred `CADisplayLink`, ki kliče želeno metodo ob osvežitvi zaslona [64], kar običajno sovpada z želenimi 60 FPS.

Vseeno se pri XNI nismo odločili za uporabo tega postopka. `CADisplayLink` nudi stabilno klicanje naše metode, a smo omejeni na ponujeni interval (ali

njegove večkratnike). Da bi se držali funkcionalne enakosti XNAja, je bilo potrebno izbrati eno od dveh alternativ. Ena od dodatnih možnosti za izvedbo igrine zanke v iOSu je uporaba dodatne niti, ki posodablja in izrisuje igre vzporedno z izvajanjem sporočil operacijskega sistema. To je sicer primerno za igre, ki tudi sicer uporabljajo večnitno arhitekturo, a se ne sklada z izvirnim delovanjem ogrodja XNA. Zato smo uporabili drugo alternativo, pri kateri vskočimo v zanko razreda `UIApplication`, prevzamemo nadzor in sami izdelamo običajno igrino zanko. Ta v razredu `GameHost` preko funkcije `CFRunLoopRunInMode` izvaja sporočila operacijskega sistema, nakar pokliče povezani objekt tipa `Game`, da izvede en korak zanke. V njem, glede na uporabnikovo izbiro, ustavi proces za potreben čas in pokliče posodobitev in izris igre.

Možnosti izbire sinhronizacije z izrisom v ogrodju XNI zaenkrat nismo izvedli. Ker se v okolju iOS sinhronizaciji niti ni mogoče izogniti, naša igra v načinu brez prekinitev teče v simulatorju z najvišjo možno vrednostjo FPS, v ciljnem okolju na iOS napravah pa smo omejeni na osveževanje s frekvenco 60 FPS.

Omenimo še, da trenutna izvedba s funkcijo `CFRunLoopRunInMode` v določenih robnih primerih, ko operacijski sistem prikazuje dodatne poglede (angl. views), ne izvaja vseh potrebnih sporočil operacijskega sistema. Ob operacijah v sistemu Game Center smo tako naleteli na neodzivnost elementov grafičnega vmesnika. Delno smo rešili problem tako, da se pred dodajanjem dodatnih pogledov prevzem zanke sprosti in vrne izvajanju razreda `UIApplication`. Po razpustitvi se nadzor zopet prevzame za popolno obvladovanje izvajanja zanke in glavnih igrinih metod. Problem še vedno ostaja, če dodaten pogled brez naše vednosti doda operacijski sistem. Rešitev predlagamo med opisom napak in popravkov v zaključku.

3.2.3 GameWindow, GameView in GameViewController

Drugi detajl glede razvoja osnovne arhitekture je priprava okna in pogleda za izris slike z OpenGLjem (`CAEAGLLayer`).

Ogrodje UIKit narekuje izdelavo grafičnega uporabniškega vmesnika (angl. GUI — graphical user interface) po metodologiji model-pogled-krmilnik (angl. MVC — model-view-controller). GUI se sestavi tako, da se v okno (`UIWindow`) doda različne poglede (otroke razreda `UIView`), katere obnašanje upravljajo njihovi kontrolniki (otroki razreda `UIViewController`) [61]. Tako smo tudi mi z dedovanjem iz teh razredov sestavili glavni razred `GameWindow` (z dedovanjem iz `UIWindow`), kot to zahteva XNA. Dodali smo mu še notranja

`GameView` in `GameViewController` za ugoditev iOS arhitekturi.

Pri izvedbi smo morali biti pozorni na različne ločljivosti zaslonov in vrednosti gostote slikovnih elementov, ki so na napravah z visokoločljivostnimi zasloni Retina podvojili ločljivost pri enaki fizičnih dimenzijah zaslona [65]. Druga glavna skrb okna in krmilnika pogleda je spreminjanje orientacije zaslona. Zaradi prenosnosti naprav je mogoče hitro obrniti orientacijo zaslona med pokončno in ležečo, na kar se mora igra ustrezno odzivati, seveda v skladu z željami programerja.

3.3 Razvoj cevovoda vsebine

Kot smo omenili v pregledu arhitekture ogrodja XNA je njen pomemben del cevovod vsebine. Cevovod sestoji iz ustreznih programskih knjižnic in integracije v razvojno okolje Visual Studio ter njegov sistem za gradnjo aplikacij MSBuild [66, 67].

Če bi želeli povsem poustvariti naravo delovanja cevovoda vsebine, bi morali podobno integracijo doseči z razvojnim okoljem Xcode. Ker bi za to potrebovali dobro poznavanje razvojnega okolja in možnosti za njegovo razširjanje, smo presodili, da v začetni fazi razvoja ogrodja to ne bi bila smiselna odločitev. Osredotočili smo se na samo izgradnjo funkcionalnosti programske knjižnice in vse razrede vključili v ciljno knjižnico ogrodja XNI.

Cevovod vsebine v XNI se sproži šele med izvajanjem ob nalaganju vsebine skozi razred `ContentManager`. Na ta način sicer izgubimo glavno prednost cevovoda vsebine, ki v XNA optimizira nalaganje s predpripravo podatkov iz vhodnih datotek v ciljno obliko. Čeprav je to glavna prednost take arhitekture, je pomembno opozoriti, da tudi brez nje ogrodje XNI ohranja vso funkcionalnost cevovoda vsebine. Izvedba same arhitekture cevovoda vsebine je programsko torej enaka, le brez vmesnih korakov shranjevanja binarnih datotek s predelano vsebino in njihovim nalaganjem med izvajanjem.

3.4 Uporaba ogrodja med razvojem

Izdelano ogrodje je mogoče iz ciljnih projektov uporabljati na dva načina. Pri prvem v Xcode povežemo direktno projekt XNI in izvorno kodo v njem. S tem lahko med razvojem ciljne aplikacije popravljamo tudi samo ogrodje. Razvojno okolje ga sproti prevaja z novimi spremembami. Drugi način uporablja vključitev vnaprej prevedene statične knjižnice v projekt, kar je pri-

merneje za končne uporabnike. Ta način je opisan v podpoglavju o izdaji ogrodja.

Za povezavo z ogrodjem XNI v obliki izvorne kode so potrebni naslednji koraki [68]:

1. V ciljni projekt dodamo projekt XNI.

2. Korakom gradnje določimo odvisnost, naj pred grajenjem igre predhodno prevede knjižnico XNI.

3. V povezovalnem koraku gradnje dodamo, naj igro poveže s knjižnico XNI.

4. Med ogrodja dodamo še Applova ogrodja, ki jih XNI potrebuje za delovanje.

5. Glavni razred igre mora dedovati iz razreda `Game`.

6. Spremenimo proceduro `main`, da ustvari glavni objekt aplikacije tipa `GameHost`, za posrednik pa uporabi glavni razred igre.

Na tem mestu omenimo, da se v kontekstu okolja Xcode uporablja drugo ločevanje med knjižnicami in ogrodji, kot smo ga opisali v uvodu. Ogrodja v Xcodu se nanašajo na poseben način izdajanja knjižnic skupaj z vsemi drugimi potrebnimi datotekami, predvsem zaglavnimi [69]. To bi teoretično lahko naredili tudi mi, vendar za ustvarjanje statičnih ogrodij za iOS Xcode žal ne ponuja predlog.

3.5 Izdaja ogrodja

Ogrodje smo na opisani način med uporabo v lastnih projektih razvili do te mere, da je bilo pripravljeno za izdajo. Za shranjevanje verzij programske kode smo uporabili sistem Subversion v Googlovem javnem repozitoriju Google Project Hosting in s tem samo programsko kodo dali javno na voljo z odprtokodno licenco. Vsakdo si lahko iz repozitorija prenese celoten Xcode projekt in ga na način opisan v prejšnjem delu poveže s svojo igro. To je primerno za tiste, ki jih zanima, kako je XNI zgrajen ali v želji po spreminjanju knjižnice. Za končne uporabnike, ki jih zgradba ne zanima, je primernejši način povezovanja z že prevedeno knjižnico.

Med razvojem iOS aplikacij jih skupaj z ogrodjem izvajamo v dveh glavnih okoljih: znotraj operacijskega sistema Mac OS X v aplikaciji iOS simulator in na ciljnih napravah z operacijskim sistemom iOS. V prvem poteka izvajanje na Intelovih procesorjih z naborom ukazov x86, v drugem jih poganjajo procesorji arhitekture ARM [10]. Če ogrodje XNI uporabljamo v obliki izvorne kode, Xcode poskrbi, da se prevaja ogrodje skupaj z igro v strojno kodo potrebne oblike.

Na ta način se zgradita dve verziji knjižnice libXni.a. Končnim uporabnikom moramo dostaviti obe, da lahko polnopravno razvijajo aplikacije v obeh okoljih. Ciljni uporabnik mora v tem primeru stalno menjati, s katero knjižnico naj se poveže igra, kar je nezaželeno. Rešitev tega problema je izdelava univerzalne knjižnice, ki združuje obe ciljni arhitekturi.

Da bi čimbolj avtomatizirali izdelavo take knjižnice smo v ogrodje XNI dodali nov ciljni izdelek (angl. target), ki s pomočjo skripte v zadnjem koraku grajenja knjižnice poskrbi, da se zgradita obe verziji arhitekture in izdelka poveže skupaj [70]. Univerzalno knjižnico in javne zaglavne datoteke smo na koncu združili v arhiv in datoteko ponudili za prenos s strani projekta [71].

Za končne uporabnike sama izdaja knjižnice ni dovolj. Potrebno je bilo izdelati še navodila za uporabo. Prva so bila izdelana v slovenščini v skripti Razvoj računalniških iger za iPhone in iPad z uporabo ogrodja XNI [72], druga pa v angleščini na strani projekta [73].

Poglavje 4

Uporaba

4.1 Končni izdelek

Glavne iteracije načrtovanja in razvoja ogrodja XNI so potekale med julijem 2010 in februarjem 2011. V tem času so izšle 4 večje različice ogrodja:

- *0.1*: najosnovnejši deli ogrodja za izris 2D slik in interakcijo preko dotikovnega vmesnika,
- *0.2*: stabilnostne posodobitve in dokončan sistem komponent (angl. game components) in storitev (angl. game services),
- *0.3*: podpora za 3D grafiko in nalaganje modelov v tekstovnem formatu .x ter
- *0.4*: podpora za zvok, predvajanje glasbenih datotek in pisanje tekstov.

Popoln pregled razvite funkcionalnosti po imenskih prostorih je objavljen na strani projekta [74].

Funkcionalnost ogrodja zadostuje za izdelavo 2D in 3D iger z zvokom, glasbo ter interakcijo preko dotikovnega vmesnika ter pospeškomera. Mogoče je uporabiti tudi vse druge zmogljivosti iOS naprav, saj ogrodje ne preprečuje direktne uporabe ostalih knjižnic operacijskega sistema.

Ogrodje je bilo uporabljeno tako v izobraževalnem procesu, kot za profesionalni, komercialni razvoj iger. V obeh primerih se je izkazalo kot primerno za delo in ustrezno za objavo v Applovi trgovini App Store.

4.2 Uporaba XNI v izobraževanju

Ogrodje XNI je primarno nastalo za uporabo v praktičnem delu predmeta Tehnologija iger in navidezna resničnost visokošolskega študija Fakultete za računalništvo in informatiko Univerze v Ljubljani v zimskem semestru šolskega leta 2010/11. Glavni del načrta in razvoja ogrodja se je zgodil poleti v pripravah na šolsko leto, rezultat česar je bila osnovna različica 0.1. Nato se je med poučevanjem ogrodje dogradilo še s potrebnimi manjkajočimi deli za izvedbo predmeta.

Skupaj s profesorjem in asistentom smo sodelovali pri izvajanju vaj in tako direktno videli uporabo XNIja v izobraževalnem procesu. Glede na to, da noben od udeležencev ni imel predhodnih izkušenj z ogrodjem XNA, smo lahko z uporabo XNIja ugotovili tudi širšo kvaliteto učenja z uporabo objektno orientiranega ogrodja srednje stopnje, ki velja tako za XNI kot XNA. Študentje so si pri vajah pomagali z omenjeno skripto Razvoj računalniških iger za iPhone in iPad z uporabo ogrodja XNI [72], ki je v praksi razložila teoretično osnovo pridobljeno na predavanjih [75]. Dodatno so jim bili v veliko pomoč videoposnetki vaj, skozi katere so si lahko ponovili poljubne dele praktičnih prikazov uporabe ogrodja.

Izkazalo se je, da s samo uporabo razredov in funkcionalnosti v ogrodju XNI študentje niso imeli problemov. Skozi vrsto primerov smo jim demonstrirali izgradnjo vseh osnovnih gradnikov iger, kar so z uspehom prenesli na lastne igre. Celo brez direktne dokumentacije (pomagali so si lahko posredno z opisi razredov v XNAju) z uporabo samih razredov ni bilo večjih problemov. To po naše potrjuje intuitivnost objektne paradigme pri razvoju iger, kjer medsebojno sodeluje veliko različnih podsistemov.

Največ problemov glede arhitekture ogrodij XNA in XNI leži v prevzemu nadzora. Razred `Game` povsem skriva ozadje delovanja igrine zanke in komponent igre (angl. game components). To ustvari nekakšno mističnost glede delovanja celotne igre (podoben problem je pri razvoju uporabniškega vmesnika z urejevalnikom Interface Builder). Ker uporabniku ni direktno jasno, kako deluje igra, mu sprva ni enostavno razmišljati o izgradnji njene arhitekture s pomočjo danih komponent. Zna le slediti primerom, sam pa težko načrtuje nadaljnji razvoj. Zato je zelo pomembno, da se pri učenju dobro razloži, kako deluje razred `Game` in sistem komponent. Za osnovno razumevanje samega ogrodja drugih pomembnejših točk nismo odkrili.

Študentje so imeli veliko problemov predvsem s pripravo statične knjižnice za delovanje v razvojnem okolju Xcode. Ker namestitev zahteva izvedbo relativno veliko korakov, je pri praktično vsakem od njih nekdo spre-

Slika 4.1: Posnetki zaslona iger študentov izdanih v trgovini App Store

gledal katero od navodil. Na podlagi pogosto ponavljajočih težav smo tako zbrali rešitve teh problemov in jih novim uporabnikom dali na voljo v delu dokumentacije [76]. V prihodnosti bi bilo primerno z uporabo predlog poenostaviti proces namestitve in integrirati razvoj v Xcode.

Ostali problemi uporabe so odpadli še na jezik Objective-C. Glavni razlog sta bila neznačilna sintaksa jezika po vzoru SmallTalka in ročno upravljanje s pomnilnikom ter delovanje kazalcev. Kar se tiče učenja novih jezikov smo mnenja, da je čim širše poznavanje sintaks koristno: programerje uči novih načinov razmišljanja in razkriva dejstvo, da je pridobljeno semantično znanje prenosljivo.

Čisto na koncu je morda en od večjih problemov učenja razvoja iger za iOS v tem, da je platformsko zaklenjen na operacijski sistem Mac OS X in razvojno okolje Xcode. Ker OS X nima prevladujočega tržnega deleža je večina študentov omejena na delo v učilnicah z Apple opremo ali na legalno in praktično sporne namestitve Mac OS Xa na poljuben računalnik (lahko tudi v obliki virtualizacije). Ta problem ogrodje XNI posredno naslavlja v samem bistvu kopije ogrodja XNA. Ker se arhitekturno skoraj povsem sklada z izvirnikom, je mogoče vse naučeno znanje v enem okolju zelo enostavno uporabiti v drugem. Čeprav so predavanja potekala v Objective-Cju, smo mnenja, da bi jim bilo mogoče brez problemov vzporedno slediti z reševanjem nalog v C#u.

V vsakem primeru se je ogrodje XNI izkazalo za primerno. Študentje so to potrdili z nadpovprečnim uspehom pri izdelavi lastnih iger. Od 32 študentov, ki so opravili svoje obveznosti pri predmetu, je bilo 8 njihovih iger uspešno izdanih na storitvi App Store (slika 4.1). Raznolikost razvitih iger, ki pokrivajo žanre od raznih arkadnih iger pa do dirkanja in celo platformerjev [77], kaže na primerno izbiro knjižnice srednjega nivoja.

4.3 Uporaba XNI v profesionalnem razvoju

Vzporedno z delom v izobraževanju se je ogrodje XNI razvijalo za potrebe profesionalnega razvoja iger za naprave iOS. V tem primeru uporabe pridejo bolj do izraza zahteve po hitrosti razvoja, hitrost izvajanja in razširljivosti. V nadaljevanju opisujemo subjektivno oceno vseh treh, glede na izkušnje z razvojem igre Monkey Labour, ki je bila z ogrodjem XNI razvita med novembrom in decembrom 2010.

4.3.1 Hitrost razvoja

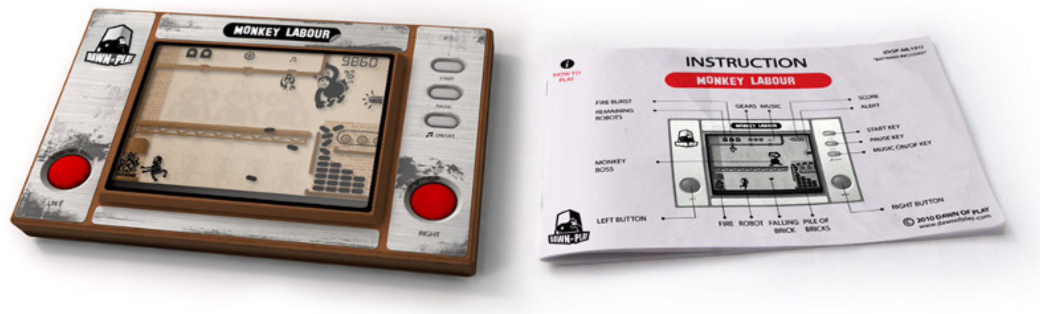
Monkey Labour je arkadna igra za iOS platformo (sliki 4.2 in 4.3), razvita v dobrem mesecu časa z enim glavnim programerjem. Izdelali smo jo z ogrodjem XNI in objavili na App Storu 23. decembra 2010 [78].

Razvoj je potekal zelo hitro predvsem zaradi uporabe ogrodja XNI, saj smo s predhodnimi dolgoletnimi izkušnjami z ogrodjem XNA lahko izdelali točno enako arhitekturo, kot bi jo sicer. Seveda je to specifična prednost za razvijalce, ki s platforme XNA prehajajo na razvoj iger za iOS. V nasprotnem primeru se morajo naučiti nove arhitekture, podobno kot pri katerikoli drugi knjižnici.

Načeloma lahko pri hitrosti razvoja z ogrodjem XNI računamo na prednosti arhitekture in funkcionalnosti ogrodja XNA, vendar s pomembnima razlikama:

- *Upravljanje s pomnilnikom*: Ogrodje XNA deluje na ogrodju .NET, ki razpolaga z avtomatskim sproščanjem pomnilnika (angl. garbage collection) [79]. Razvoj pod iOS zahteva ročno dodeljevanje in sproščanje s pomočjo štetja referenc (angl. reference counting) [80]. Krivulja učenja je zaradi tega za začetnike bolj strma in samo pisanje kode zahteva več dela. Nasprotno avtomatsko sproščanje pomnilnika zahteva več učenja v koraku optimizacije, saj prinaša slabosti, ki so zelo občutne ravno v domeni razvoja iger [81]. Če se ne zavedamo implicitnega dodeljevanja in se mu že preventivno izognemo, nam učinkovitost razvoja prav tako pade zaradi potrebnih korakov optimizacije.

- *Operatorji*: C# omogoča definiranje operatorjev (angl. operator overloading), kar XNA s pridom izkorišča za pregleden zapis računskih izrazov nad vektorji, matrikami, točkami in barvami. Objective-C definiranja operatorjev ne podpira, tako da smo morali vse operacije negacije, seštevanja, odštevanja, množenja in transformacije vektorjev (`vektor↩ * matrika`) nadomestiti s klici metod. Da bi omili situacijo, smo poleg metod, ki po vzoru XNAja vračajo novo ustvarjen objekt, dodali še klice, ki spremenijo direktno strukturo na kateri kličemo operacijo. Poleg tega vsi ti klici vračajo kazalec na samega sebe, tako da lahko na objektih verižno izvedemo več zaporednih operacij (angl. method chaining) [82]. Primer izboljšane sintakse dela z vektorji in matrikami, ki občutno pohitri delo in izboljša berljivost, je v izpisu 4.1.



Slika 4.2: Izgled izmišljene prenosne igralne konzole LCD z igro Monkey Labour

Slika 4.3: Vizualni učinki v igri Monkey Labour: prikaz grafičnih elementov pri pogledu pod kotom, simulacija pritiskanja na zaslon, sledovi elementov zaradi počasnega osveževanja in sence pod elementi

```
1  // Projekcija vektorja
2  // XNA
3  f = Vector2.Dot(new Vector2(1, 2), Vector2.Normalize(new Vector2(3, 1)));
4
5  // XNI (brez povezovanja metod)
6  f = [Vector2 dotProductOf:[Vector2 vectorWithX:1 y:2] with:[Vector2 normalize↩
       :[Vector2 vectorWithX:3 y:1]]];
7
8  // XNI (s povezovanjem metod)
9  f = [Vector2 dotProductOf:[Vector2 vectorWithX:1 y:2] with:[[Vector2 ↩
       vectorWithX:3 y:1] normalize]];
10
11
12 // Množenje matrik
13 // XNA
14 m = Matrix.CreateTranslation(15.6f, 10.5f, 0) * Matrix.CreateRotationX(↩
       MathHelper.PiOver2) * Matrix.CreateRotationZ(MathHelper.Pi) * Matrix.↩
       CreateScale(0.1f);
15
16 // XNI (brez povezovanja metod)
17 m = [Matrix multiply:[Matrix multiply:[Matrix multiply:[Matrix ↩
       createTranslationX:15.6 y:10.5 z:0] by:[Matrix createRotationX:M_PI_2]] ↩
       by:[Matrix createRotationZ:M_PI]] by:[Matrix createScaleUniform:0.1]];
18
19 // XNI (s povezovanjem metod)
20 m = [[[[Matrix createTranslationX:15.6 y:10.5 z:0] multiplyBy:[Matrix ↩
       createRotationX:M_PI_2]] multiplyBy:[Matrix createRotationZ:M_PI]] ↩
       multiplyBy:[Matrix createScaleUniform:0.1]];
21
22
23 // Transformacija vektorja
24 // XNA
25 v = Vector3.Transform(Vector3.UnitX * 5, Matrix.CreateRotationZ(angle) * ↩
       Matrix.CreateTranslation(offset));
26
27 // XNI (brez povezovanja metod)
28 v = [Vector3 transform:[Vector3 multiply:[Vector3 unitX] by:5] with:[Matrix ↩
       multiply:[Matrix createRotationZ:angle] by:[Matrix createTranslation:↩
       offset]]];
29
30 // XNI (s povezovanjem metod)
31 v = [[[Vector3 unitX] multiplyBy:5] transformWith:[[Matrix createRotationZ:↩
       angle] multiplyBy:[Matrix createTranslation:offset]]];
```

Izpis 4.1: Delo z vektorji in matrikami s povezovanjem metod

4.3.2 Hitrost izvajanja

Pri načrtovanju ogrodja XNI smo povedali, da hitrost izvajanja ne bo ena izmed naših smernic za reševanje arhitekturnih problemov pri prenosu iz C#a v Objective-C. Vseeno se moramo razlik pri performancah zavedati, saj lahko le tako pravilno načrtujemo obseg naših iger pri trenutnem stanju optimizacije ogrodja. Glavne razlike v hitrosti izvajanja glede na ogrodje XNA so naslednje:

- *Hitrost nalaganja vsebine igre*: Ker cevovod vsebine ni ločen korak predprocesiranja, temveč se nalaganje iz izvirnih datotek in transformacije podatkov dogajajo direktno ob sprožitvi nalaganja med izvajanjem, se moramo zavedati, da bodo časi nalaganja vsebine zaradi tega občutno daljši. Lastniki prenosnih naprav seveda zahtevajo hitro odzivnost in pričakujejo, da lahko takoj začnejo uporabljati pognano aplikacijo. Zato kaže ureditev arhitekture cevovoda vsebine izbrati za enega primarnih ciljev izboljšave ogrodja, o čemer več pišemo v zaključku.

- *Delo s strukturami*: Omenili smo že, da Objective-C razpolaga zgolj s strukturami iz Cja, ki le združujejo več primitivnejših podatkovnih tipov in niso izrazno enake razredom, kot je to v C#u. Zato smo strukture, ki predstavljajo vektorje, matrike, točke, pravokotnike in barve nadomestili z razredi. To za sabo potegne ceno priprave in sproščanja prostora objekta na kopici, medtem ko se strukture v C#u dodelijo na skladu. Po drugi strani ima uporaba razredov namesto struktur tudi svojo prednost. Ko pošiljamo objekt v druge metode, se mora vanje prenesti zgolj kazalec namesto izdelave kopije celotne strukture, kar je v primeru matrike s 16 števili s plavajočo vejico kar precejšnja velikost. Ogrodje XNA za rešitev tega celo namenja posebne klice metod, pri katerih se tudi strukture prenašajo po referenci namesto po vrednosti. Ta način pri strukturah v XNIju se uporabi sam po sebi.

- *Avtomatsko sproščanje pomnilnika*: Na iOS platformi ni omogočen avtomatski sistem sproščanja pomnilnika, temveč moramo sami skrbeti za lastništvo nad objekti. Objective-C ponuja svojevrstno avtomatsko sproščanje objektov z zamikom (angl. autorelease). Zaradi pogoste uporabe tega načina, predvsem zaradi ustvarjanja množice začasnih objektov iz prejšnje točke, naletimo na podobne probleme kot pri običajnem avtomatskem sproščanju pomnilnika. Posledica velikega število ustvarjenih objektov med tekom igre so v XNAju občasne kratke pavze ob sprožitvi avtomatskega sproščanja. Podobno se v XNIju to pozna,

ko se sproščajo avtomatsko sproščeni objekti. Da ne pride do podobnih občasnih daljših zaustavljanj, smo izvedli avtomatsko sproščanje ob koncu vsakega obhoda igrine zanke, kot priporoča Apple v navodilih za delo s pomnilnikom [80].

4.3.3 Razširljivost

Tretja od pomembnejših lastnosti, po kateri lahko ocenjujemo programske knjižnice, je razširljivost. Zanima nas, kako enostavno je izdelati stvari, ki jih sama knjižnica ne podpira. Zaradi močne integracije in povezanosti med razredi, še posebej pri ogrodjih, smo lahko precej odvisni od sposobnosti knjižnice. Zaradi skrivanja podatkov zna biti težko priti do potrebnih spremenljivk s katerimi lahko sami nadgradimo dano funkcionalnost.

Ogrodje XNI je dobro razširljivo zaradi dveh lastnosti:

- *Odprtokodnost*: Ogrodje XNI je objavljeno v javnem repozitoriju izvorne kode Google Code, kjer so dostopne vse verzije izvorne kode skozi sitem Subversion. To omogoča vsakomur, da točno analizira delovanje razredov in po potrebi spiše ustrezne popravke ali razširitve.

- *Objective-C*: Ostali prenosi ogrodja XNA na iOS, zgrajeni s platformo MonoTouch, nam sicer omogočajo, da naše programe pišemo še vedno v jeziku C#, a poleg očitne prednosti ima to tudi svojo slabost. Oteženo je namreč sodelovanje ogrodja XNA z ostalimi ogrodji, ki sestavljajo iOS in so spisana predvsem v Objective-Cju. Z XNIjem lahko brez omejitev uporabljamo vse ostale knjižnice operacijskega sistema, kot tudi izkoriščamo prednosti razvoja z okoljem Xcode.

Da je ogrodje XNI razširljivo, smo v praksi pokazali večkrat. Ogrodje XNA na primer omogoča zgolj izris črt debeline 1 slikovni element. Če poznamo OpenGL, lahko pri izrisovanju črt pred klicem izrisa grafičnih primitivov brez problema uporabimo ukaz, ki spremeni debelino pri izrisu črt.

Podobno lahko namesto črt ukažemo izris točk, kar je bilo v zadnji verziji ogrodja XNA odstranjeno. Pogled v izvorno kodo naštevalnega tipa `PrimitiveType` nam razkrije, da so vrednosti tipa vezane na OpenGL konstante za črte in trikotnike. Tako lahko pri klicu izrisa primitivov namesto vrednosti iz tipa `PrimitiveType` uporabimo direktno OpenGL konstanto za izris pik.

Še en primer razširljivosti je uporaba Applovega ogrodja Game Kit. Ker v XNI še niso sprogramirani razredi imenskih prostorov `Net` in `GamerServices`, z njimi ne moremo izdelati večigralstva ter uporabe spletnih storitev

lestvic (angl. leaderboards) in dosežkov (angl. achievements). V igri Monkey Labour smo zato izdelali potrebno funkcionalnost direktno s temu namenjenim ogrodjem GameKit. V prihodnosti bo ta funkcionalnost v samem XNI, dokler pa temu ni tako, nas nič ne ovira, da ne bi uporabili knjižnice operacijskega sistema. To je zelo dobro za sledenje najnovejšim trendom in zmožnostim novih verzij operacijskega sistema, kar omogoča konkurenčnost drugim knjižnicam.

Poglavje 5

Zaključek

Prvi del razvoja ogrodja XNI je končan. Produkt je nastal kot posledica nekaterih ključnih dejavnikov:

- potrebe po razvoju za platformo iOS,
- veliko predhodnih izkušenj z ogrodjem XNA,
- Applove politike omejevanja načinov ustvarjanja aplikacij in
- potrebe po učenju razvoja iger za iOS.

Čeprav je Apple v času razvoja ogrodja omilil svoje nasprotovanje večplatformskemu razvoju in ga ponovno dovolil, še vedno obstaja več razlogov za nadaljnji obstoj ogrodja XNI:

- Rešitve, ki omogočajo pisanje iOS aplikacij z ogrodjem XNA v programskem jeziku C# so plačljive in ne podpirajo 3D grafike.
- Arhitektura in stil ogrodij XNI (in XNA) lahko nekomu osebno bolj odgovarja kot ostale konkurenčne knjižnice.
- Možnost za hitrejši prenos iger s platforme iOS na Windows, Windows Phone 7 in Xbox ter obratno.
- Trenutna verzija ogrodja XNI že zadošča za večino osnovnih potreb razvoja iger. Nadaljnja razširljivost je enostavna, ogrodje pa je v praksi preverjeno v vseh korakih razvoja in izdaje iger za iOS.
- Primernost za učenje programiranja iger, ki sledi iz dovolj nizkega nivoja same funkcionalnosti XNA. Prenos naučenega znanja med XNI in XNA je prav tako pomemben faktor.

Skozi razvoj ogrodja in uporabo v praksi so se hkrati pokazale pomanjkljivosti dosedanjih odločitev, ki smo jih opisali med rezultati razvoja. Dodatno razvoj operacijskega sistema iOS prinaša nove poti za nadaljevanje. Za zaključek bomo tako opisali ugotovljene napake in popravke, ki bi jih bilo v prihodnosti smiselno uvesti. Na koncu sledi še obravnava dolgoročnejše prihodnosti razvoja ogrodja, tako z inženirskega kot tržnega stališča.

5.1 Napake in popravki

Predlagani popravki, ki sledijo iz dosedanjega dela z ogrodjem so naslednji:

- *Enotna predpona*: Že pri postavljanju smernic za razvoj ogrodja smo omenili pomanjkljivost glede predpon. Ker smo želeli ohraniti imena razredov, se nismo držali ustaljene norme, da se imenom v knjižnicah doda enotno predpono, s čimer se izogne možnemu prekrivanju imen. Ker se je potreba po tem pokazala šele pozno v razvoju, nosi predpono zgolj en razred (`XniPoint`). Bolje bi bilo, da bi se za držanje tega pravila odločili že takoj na začetku, saj nas zdaj čaka postopek preimenovanja več kot 150 že zgrajenih razredov. Da ne bi bilo preimenovanje potrebno še enkrat, je to spremembo dobro izvesti šele, ko bo končno odločeno, kakšna naj bo enotna predpona.

- *Enotna uporaba zbirk z določenim tipom*: Tudi v tem primeru smo rešitev problema pomanjkanja splošnih zbirk začeli uporabljati šele sredi razvoja. Tako se na nekaterih mestih še vedno pojavlja podvojena koda, ki bi jo lahko odpravili s postopkom vključevanja predlog, opisanim v podpoglavju 1.2.1. Trenutno arhitekturo je potrebno pregledati in ustrezno nadomestiti z novim načinom izvedbe.

- *Imena inicializatorjev*: Pri določanju imen inicializacijskih metod smo odkrili pomanjkljivost dinamične narave Objective-Cja. Ker v XNAju strukturi `Vector` in `Point` obe uporabljata konstruktor s parametri poimenovanimi `x` in `y`, smo inicializatorje v Objective-Cju v obeh primerih poimenovali `initWithX:y:`. Ker `Vector` uporablja parametre tipa `float`, `Point` pa tipa `int`, gre za enako poimenovano metodo s parametri različne pomnilniške velikosti. Objective-C se pri klicu metod na objektih z dinamičnim tipom `id` odloča le glede na ime metode, tako da iz konteksta ne zna izbrati ali naj pokliče metodo s parametri tipa `float` ali `int`. V primerih, ko izbere napačno, se v metodo ne prenesejo prave vrednosti, kar privede do napačnega nadaljevanja izvajanja.

Za rešitev tega problema je potrebno zagotoviti edinstvena imena metod (v primeru `XniPoint` recimo z inicializatorjem `initWithIntX:y:`). Popravke za to napako smo uvedli že takoj po odkritju.

- *Zanka igre*: Trenutna verzija glavne zanke v razredih `Game` in `GameHost` ni povsem prilagojena na sodelovanje s sistemom. Kot smo omenili na koncu podpoglavja 3.2.2 o izvedbi zanke, se je v primeru dodajanja drugih pogledov iz ogrodja UIKit izkazalo, da se, kljub našemu procesiranju sporočil, vsa ne izvedejo. Posledica so problemi z odzivnostjo in nedelovanjem takih pogledov. Trenutno smo problem zaobšli tako, da pred prikazom drugih pogledov prepustimo nadzor nad izvajanjem nazaj sami aplikaciji in ga po koncu prikaza nazaj prevzamemo. Žal lahko v določenih primerih, recimo med registracijo uporabnika v storitev Game Center, pride do prikaza drugih pogledov, brez, da mi za to izvemo in ustrezno reagiramo. Potrebno je preučiti, če lahko glavno zanko igre popravimo, da bo izvedla tudi manjkajoča sistemska sporočila. Dodatna pomanjkljivost zanke igre je manjkajoča podpora za sinhronizacijo z osveževanjem zaslona. V tem primeru bi morali uporabiti alternativni način izvedbe zanke z uporabo razreda `CADisplayLink`. Za pravilno izvedbo je potrebno preučiti, kako točno se obnaša igrina zanka v ogrodju XNA pri različnih kombinacijah nastavitev lastnosti `SyncWithVerticalRetrace` in `IsFixedTimeStep`.

- *Uporaba ogrodja GLKit*: En od osnovnih delov ogrodij XNA in XNI je podpora računanju z vektorji in matrikami. Ker iOS podobne funkcionalnosti do zdaj ni ponujal, smo potrebne razrede in operacije morali v XNIju spisati sami. V novi različici operacijskega sistema iOS 5 je Apple pripravil svojo izvedbo te funkcionalnosti z ogrodjem GLKit [83]. V prihodnosti bi bilo dobro za interno strukturo podatkov v razredih za delo z linearno algebro uporabiti strukture iz ogrodja GLKit. Nato lahko za izvedbo operacij nad njimi uporabimo že pripravljene metode in se izognemo pisanju lastne programske kode na mestih, kjer do zdaj še nismo utegnili prenesti funkcionalnosti. Pri tem je potrebno biti pozoren na način zapisa matrik (vektorji v stolpcih ali vrsticah), kot smo to zapisali v podpoglavju 2.5.2.

- *Enotsko testiranje*: Novost razvojnega okolja Xcode 4 je podpora enotskim testom [84]. Glede na to, da mora ogrodje XNI vračati povsem enake rezultate kot XNA, je uporaba enotskih testov za preverjanje pravilnosti izvedbe zelo na mestu. Z ogrodjem XNA je potrebno izdelati

nabor rezultatov glavnih operacij nad razredi in jih nato v obliki enotskih testov uporabiti za testiranje ogrodja XNI.

5.2 Nadaljnji razvoj

Omenili smo razloge za nadaljevanje obstoja ogrodja, a prihodnost razvoja vseeno ni samoumevna. Vprašati se je potrebno tudi, kako nadaljevanje projekta ekonomsko upravičiti.

Na eni strani lahko ogrodje uporabljamo interno, brez da bi poleg odprtokodnosti posebej skrbeli za druge uporabnike. V tem primeru nadaljnji razvoj upravičuje hitrejša izdelava iger s poznano arhitekturo in popoln nadzor nad njenim razvojem. Nadaljnji razvoj ogrodja je lahko povsem odvisen od funkcionalnosti, ki jo potrebujemo pri trenutnih projektih. Vsako novo fazo načrtovanja in razvoja izvedemo tako, da analiziramo arhitekturo in delovanje razredov s ciljnega področja ogrodja XNA ter opravimo prenos v Objective-C s pomočjo ogrodij v iOSu. Čas za razvoj moramo obravnavati v sklopu projekta, za katerega je potrebno nadgraditi XNI, naložba pa se povrne pri ponovni uporabi v prihodnjih projektih. Nadaljnji razvoj ogrodja XNI bi se tako financiral posredno preko razvoja iger.

Druga možnost je, da ogrodje postavimo v ospredje in ga poskušamo financirati vsaj delno neposredno. V tem primeru mora dovolj dobro zadovoljevati potrebe ciljne skupine, da so pripravljeni za uporabo tudi plačati. Zaradi tržne uspešnosti platforme iOS [9, 85, 86], slabega ugleda storitve Xbox Live Indie Games [87] in možnostjo razvoja za Windows Phone 7 z ogrodjem XNA [88], se ponuja vrsta razlogov za prehajanje razvijalcev z ogrodja XNA na platformo iOS in vzporedni razvoj za obe okolji. Tu leži priložnost za ogrodje XNI kot samostojno rešitev.

Slabost zgolj interne uporabe je visoka odvisnost od ciljnih projektov, zaradi katerih se nadgrajuje sposobnosti ogrodja. V praksi je potrebno čim hitreje končati projekte, tako da se pogosto izdela le najnujnejše spremembe, čas za preoblikovanje kode (angl. refactoring) in splošno nadgradnjo pa je težko najti. V skrbi za neposreden razvoj ogrodja se temu izognemo, a moramo računati, da zaradi dodatnega časa vloženega v trženje XNIja ostane obratno manj časa za razvoj samih iger.

5.2.1 Tehnične izboljšave

Večji posegi v ogrodje, ki bi izboljšali njegovo splošno delovanje, so naslednji:

- *Cevovod vsebine*: Kot smo zapisali pri oceni hitrosti izvajanja je cevovod vsebine glavno mesto za izboljšanje tako hitrosti nalaganja, kot poenotenja načina delovanja z ogrodjem XNA. Hkrati gre za najtežavnejšo spremembo, saj dobra tehnična rešitev v okolju Xcode ni jasna. Potreben je obširen raziskovalni proces, kako najbolj elegantno povezati predprocesiranje vsebine v postopek grajenja izvršilne datoteke igre.

- *Game Center*: Ena tržno najpomembnejših manjkajočih funkcionalnosti ogrodja XNI je podpora spletnim storitvam in večigralstvu skozi sistem Game Center. Z ogrodjem GameKit je potrebno izdelati funkcionalnost imenskih prostorov `GamerServices` in `Net`, kar predstavlja dolgotrajnejši razvojni cikel.

- *iOS 5 in OpenGL ES 2.0*: Trenutno ogrodje XNI podpira iOS različice 3.2 (ali novejše) in širši nabor naprav, ki delujejo z OpenGL ES 1.1. Prihajajoči operacijski sistem iOS 5 predstavlja smiselno točko za prekinitev združljivosti z nespremenljivim grafičnim cevovodom v OpenGL ES 1.1 in preiti izključno na novejši OpenGL ES 2.0. iOS 5 bo namreč na voljo samo za naprave, ki podpirajo programabilni grafični cevovod novega OpenGL ES standarda. S tem bi sistem senčilnikov lahko polno prenesli z ogrodja XNA na XNI in zagotovili identičnost platform, razen na nivoju samega programskega jezika senčilnikov (HLSL v XNA in GLSL v XNI).

- *Optimizacija*: Za hitrejše delovanje ogrodja je potrebno izdelati sistem za testiranje hitrosti osnovnih operacij knjižnice in primerjati različne optimizacijske rešitve ključnih ozkih grl. Smiselno je izdelati splošno specifikacijo za testiranje hitrosti izvajanja, s katero se lahko primerja ogrodje XNI z drugimi izvedbami ogrodja XNA na platformi iOS ali z drugimi srednjenivojskimi knjižnicami za izdelavo iger.

Izboljšava uporabe ogrodja z ločenim cevovodom vsebine predstavlja tudi dobro priložnost, da se strukturo projekta spremeni tako, da bo mogoče Objective-C kodo uporabiti še za izdelavo različice ogrodja za operacijski sistem Mac OS X. Ker imajo ogrodja obeh operacijskih sistemov veliko skupnega občutnih sprememb v samem ogrodju ne bi smelo biti. Večji poudarek bi bil na pripravi projekta za prevajanje z dvema ciljnima skupinama ogrodij.

Naslednji korak, poleg podpore Mac OS Xa, je lahko podpora za izvajanje v brskalnikih. Dobra priložnost se odpira za sodelovanju s projektom JSIL, ki avtomatsko prevaja kodo CIL programov spisanih z ogrodjem .NET v JavaScript [89]. Ker zaradi pravnih omejitev ni dovoljeno avtomatsko prevesti

tudi knjižnic ogrodij .NET in XNA, mora manjkajočo izvedbo razredov teh ogrodij nekdo spisati ročno. Glede na to, da gre za zelo podoben projekt, kot je XNI, se lahko uporabi vse naučeno znanje in v semantičnem smislu tudi spisano kodo.

Še več, če bi se izdelalo pravi prevajalnik med vsemi tremi jeziki (CIL, Objective-C in JavaScript), bi se lahko marsikje uporabilo direktno izvorno kodo. Tako orodje bi bilo tudi sicer uporabno za olajšanje ročnega prenosa iger z ogrodja XNI v XNA in obratno. Podrobneje se splača pregledati prevajalniško infrastrukturo LLVM, ki jo Apple uporablja za prevajanje Objective-C kode, a izvira iz raziskovalnega projekta in do danes podpira vmesnike za razne programske jezike, tudi CIL [90].

5.2.2 Tržna strategija

Poleg tehničnih izboljšav, ki bi ogrodje XNI naredile konkurenčnejše s programerskega stališča, je potrebno razmisliti tudi o tržnem načrtu, ki bi omogočil finančno smotrnejši nadaljnji razvoj. Ker konkurenčna platforma za enostaven prehod iz okolja .NET v iOS, MonoTouch, postavlja začetno ceno relativno visoko (399 $ [23]), pušča precej manevrskega prostora za profesionalne uporabnike. Na drugi strani imamo zastonj na voljo zelo razširjeno in dobro podprto knjižnico srednjega nivoja Cocos2D [91].

Tudi sami imamo željo vsaj osnovno funkcionalnost ponuditi zastonj. Tako lahko lažje konkuriramo kot prva izbira ogrodja za razvoj za iOS in ne le kot možnost za razvijalce, ki so pred tem že poznali XNA. Prav tako je ogrodje na ta način primernejše za uporabo v izobraževanju in omogoča lažjo pridobitev uporabnikov.

Ena od možnih strategij za izvedbo takega prodajnega modela bi bilo razbitje trenutne statične knjižnice libXNI.a na več knjižnic po vzoru ločenih delov ogrodja XNA, naštetih že v pregledu v tabeli 2.1. Tako bi še bolj poenotili ciljno strukturo knjižnice, hkrati pa dobimo možnost, da nekatere dele izdamo odprtokodno, druge pa proti plačilu. Trenutno funkcionalnost (osnovni razredi ogrodja, infrastruktura igre, vhod in zvok) bi ponudili zastonj, da lahko z njimi predvsem amaterji in študentje prosto razvijejo vse glavne dele svojih iger. Nove dele spletne podpore in večigralstva, ki so pomembni predvsem za profesionalne razvijalce, bi po drugi strani prodajali. Če bi se razvil še prevajalnik med izvornimi kodami različnih programskih jezikov, bi se to orodje lahko tržilo po višjih cenah, saj omogoča velike prihranke razvijalcem.

S tako tržno strategijo se ustvari privlačna vstopna točka za nove uporab-

nike, ki lahko z ogrodjem zastonj razvijajo igre v Objective-Cju na način, ki so ga vajeni iz ogrodja XNA. Naše pretekle izkušnje iz uporabe v izobraževanju lahko izkoristimo za izdelavo učnega gradiva, tako za lažji prehod z jezika C# na Objective-C, kot tudi za tiste, ki se z razvojem iger ali celo programiranjem srečujejo prvič. S takimi vsebinami se izvaja trženje produkta in pridobivanje uporabnikov.

V naslednjem koraku, ko imajo uporabniki spisano igro z ogrodjem XNI in jo hočejo nadgraditi z naprednimi možnostmi, se jim splača uporabiti plačljive dele knjižnice, s katerimi to enostavno dosežejo na že vajen način. Če je prodaja njihove igre uspešna, se v zadnjem koraku lahko poslužijo tudi dražjih orodij za avtomatski prenos na druge platforme.

Slike

1.1 Višina programskih knjižnic za razvoj iger 5
1.2 Posnetki zaslona igre Rings of Saturn 12
1.3 Igra Rings of Saturn v prototipu projekta Moject 12

2.1 Sloji operacijskega sistema iOS 22

4.1 Posnetki zaslona iger študentov izdanih v trgovini App Store . 42
4.2 Izgled izmišljene prenosne igralne konzole LCD z igro Monkey Labour 45
4.3 Vizualni učinki v igri Monkey Labour 45

Tabele

1.1 Knjižnice za razvoj iger za iOS 9
1.2 Rešitve za uporabo ogrodja XNA v razvoju za iOS 11
1.3 Učenje razvoja na različnih sestavnih delih igre 14

2.1 Dinamične knjižnice ogrodja XNA 20
2.2 Imenski prostori izvršnega dela ogrodja XNA 21
2.3 Imenski prostori cevovoda vsebine XNA 21

Literatura

[1] E. Bethke, *Game development and production*, Wordware Game Developer's Library, Wordware Publishing, Texas, 2003, pogl. 5.

[2] J. Blow, "Game Development: Harder Than You Think," *Queue*, let. 1, št. 10, str. 28–37, 2004.

[3] S. Rabin (ured.), *Introduction to game development*, Game development series, Course Technology, Boston, 2. izd., 2010.

[4] M. L. Griss, "Software reuse: from library to factory," *IBM Systems Journal*, let. 32, št. 4, str. 548–566, 1993.

[5] (2011) Game engine — Wikipedia, The Free Encyclopedia. Dostopno na:
`http://en.wikipedia.org/w/index.php?title=Game_engine&oldid=445488089`

[6] M. DeLoura, "Game engine survey 2011," *Game Developer Magazine*, let. 18, št. 5, str. 7–12, 2011.

[7] R. E. Johnson, B. Foote, "Designing Reusable Classes," *Object-Oriented Programming*, let. 1, št. 2, str. 22–35, 1988.

[8] R. Povše, *Razvoj iger za iOS*, diplomsko delo, Fakulteta za elektrotehniko, računalništvo in informatiko, Univerza v Mariboru, 2011.

[9] (2010) Apple iPhone and iPod touch Capture U.S. Video Game Market Share. Dostopno na:
`http://blog.flurry.com/bid/31566/Apple-iPhone-and-iPod-touch-Capture-U-S-Video-Game-Market-Share`

[10] (2010) A few things iOS developers ought to know about the ARM architecture. Dostopno na:
`http://wanderingcoder.net/2010/07/19/ought-arm/`

[11] (2006) lipo(1) Mac OS X Manual Page. Dostopno na:
http://developer.apple.com/library/mac/#documentation/Darwin/Reference/ManPages/man1/lipo.1.html

[12] (2011) Build fat static library (device + simulator) using Xcode and SDK 4+ – Stack Overflow. Dostopno na:
http://stackoverflow.com/questions/3520977/build-fat-static-library-device-simulator-using-xcode-and-sdk-4

[13] (2011) Library (computing) — Wikipedia, The Free Encyclopedia. Dostopno na:
http://en.wikipedia.org/w/index.php?title=Library_(computing)&oldid=443884416

[14] Apple Inc., *iOS Development Guide*, nov. 2010. Dostopno na:
http://developer.apple.com/library/ios/documentation/xcode/conceptual/iphone_development/iOS_Development_Guide.pdf

[15] Apple Inc., *iOS Technology Overview*, nov. 2010. Dostopno na:
http://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSTechOverview.pdf

[16] (2011) iOS Game Programming Overview. Dostopno na:
http://www.idevgames.com/articles/ios-overview

[17] (2009) The Commercial iPhone Game Engine Comparison (3D and 2D). Dostopno na:
http://maniacdev.com/2009/09/the-commercial-iphone-game-engine-comparison-3d-and-2d/

[18] (2009) iPhone Game Engine Comparison – Open Source. Dostopno na:
http://maniacdev.com/2009/08/the-open-source-iphone-game-engine-comparison/

[19] (2011) Microsoft XNA — Wikipedia, The Free Encyclopedia. Dostopno na:
http://en.wikipedia.org/w/index.php?title=Microsoft_XNA&oldid=439726797

[20] B. Costanich, *Developing C# Apps for iPhone and iPad Using MonoTouch: IPhone OS Apps and Games Development for .NET Developers*, Apress, New York, 2011, pogl. 1.

[21] (2011) MonoGame. Dostopno na:
`https://github.com/mono/MonoGame#readme`

[22] (2011) ExEn: The power of C# and XNA comes to iOS, Silverlight, Android – Andrew Russell. Dostopno na:
`http://andrewrussell.net/exen/`

[23] (2011) Store / Xamarin. Dostopno na:
`https://store.xamarin.com/`

[24] (2010) Daring Fireball: New iPhone Developer Agreement Bans the Use of Adobe's Flash-to-iPhone Compiler. Dostopno na:
`http://daringfireball.net/2010/04/iphone_agreement_bans_flash_compiler`

[25] (2010) Statement by Apple on App Store Review Guidelines. Dostopno na:
`http://www.apple.com/pr/library/2010/09/09Statement-by-Apple-on-App-Store-Review-Guidelines.html`

[26] (2010) Moject prototype combines pico projector, smartphone and motion gaming (video) — Engadget. Dostopno na:
`http://www.engadget.com/2010/04/15/moject-prototype-combines-pico-projector-smartphone-and-motion/`

[27] Apple Inc., *OpenGL ES Programming Guide for iOS*, feb. 2011. Dostopno na:
`http://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLES_ProgrammingGuide.pdf`

[28] M.-H. Tsai, C.-H. Huang, J.-Y. Zeng, "Game programming courses for non programmers," *International conference on game research and development*, str. 219–223, Perth, Australia, dec. 2006.

[29] M. Jan, "F1 za garažno izdelovanje iger," *Joker*, let. 15, št. 4, str. 58–65, 2007.

[30] Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, *Predstavitveni zbornik prvostopenjskega visokošolskega študija*, 2008. Dostopno na:
`http://www.fri.uni-lj.si/file/113229/predstavitveni-zbornik-fri-vss.pdf`

[31] J. Gold, *Object-oriented game development*, Addison Wesley, Essex, 2004, pogl. 1.

[32] J. Linhoff, A. Settle, "Teaching game programming using XNA," *Conference on innovation and technology in computer science education*, str. 250–254, Madrid, Spain, jul. 2008.

[33] B. Wu, A. I. Wang, J.-E. Strøm, K. T. Blomholm, "An Evaluation of Using a Game Development Framework in Higher Education," *Conference on software engineering education and training*, str. 41–44, Washington, DC, USA, feb. 2009.

[34] O. Denninger, J. Schimmel, "Game Programming and XNA in Software Engineering Education," *Computer Games & Allied Technology*, Singapur, apr. 2008.

[35] C. Larman, *Agile and iterative development: a manager's guide*, Agile software development series, Addison-Wesley, Boston, 2004, pogl. 1.

[36] A. M. Davis, "Operational Prototyping: A New Development Approach," *IEEE Software*, let. 9, št. 5, str. 70–78, 1992.

[37] W. W. Royce, "Managing the development of large software systems: concepts and techniques," *International conference on software engineering*, str. 328–338, Monterey, CA, USA, apr. 1987.

[38] Apple Inc., *Coding Guidelines for Cocoa*, apr. 2010. Dostopno na:
`http://www.apple.com.cn/developer/mac/library/documentation/Cocoa/Conceptual/CodingGuidelines/CodingGuidelines.pdf`

[39] T. Fullerton, C. Swain, S. Hoffman, *Game design workshop: a playcentric approach to creating innovative games*, Gama Network Series, Elsevier Morgan Kaufmann, Burlington, 2008, pogl. 8.

[40] J. Schell, *The art of game design: a book of lenses*, Elsevier Morgan Kaufmann, Burlington, 2008.

[41] (2011) What Is Content? Dostopno na:
http://msdn.microsoft.com/en-us/library/bb447756.aspx

[42] Apple Inc., *The Objective-C Programming Language*, dec. 2010. Dostopno na:
http://developer.apple.com/library/mac/documentation/cocoa/conceptual/objectivec/objc.pdf

[43] G. A. Pascoe, “Elements of object-oriented programming,” *Byte*, let. 11, št. 8, str. 139–144, 1986.

[44] (2011) internal (C# Reference). Dostopno na:
http://msdn.microsoft.com/en-us/library/7c5ka91b.aspx

[45] (2011) Why doesn’t Objective-C support private methods? – Stack Overflow. Dostopno na:
http://stackoverflow.com/questions/2158660/why-doesnt-objective-c-support-private-methods

[46] (2011) Hide instance variable from header file in Objective C – Stack Overflow. Dostopno na:
http://stackoverflow.com/questions/2103858/hide-instance-variable-from-header-file-in-objective-c

[47] (2008) Objective-C: Private Methods – Mac Developer Tips. Dostopno na:
http://macdevelopertips.com/objective-c/private-methods.html

[48] (2011) Objects (C# Programming Guide). Dostopno na:
http://msdn.microsoft.com/en-us/library/ms173110.aspx

[49] (2006) End users shouldn’t have to rewrite XNA math functions in order to improve performance. Dostopno na:
http://social.msdn.microsoft.com/forums/en-US/xnaframework/thread/e715c16b-6b02-454a-b836-d402093bd36f/

[50] N. Leischner, O. Liebe, O. Denninger, *Optimizing performance of XNA on Xbox 360*, ZFS Karlsruhe, FZI Forschungszentrum Informatik, 2008.

[51] (2011) Generics (C# Programming Guide). Dostopno na:
http://msdn.microsoft.com/en-us/library/512aeb7t.aspx

[52] J. Liberty, B. MacDonald, *Learning C# 3.0*, Learning Series, O'Reilly, Sebastopol, 2008, pogl. 17.

[53] (2011) Supported Operating Systems and Hardware – XNA Game Studio. Dostopno na:
http://msdn.microsoft.com/en-us/library/bb203925(v=XNAGameStudio.10).aspx

[54] A. Lobão, B. Evangelista, J. Farias, R. Grootjans, *Beginning XNA 3.0 game programming: from novice to professional*, Apress, New York, 2009.

[55] A. Reed, *Learning XNA 3.0*, O'Reilly, Sebastopol, 2008, pogl. 13.

[56] (2001) OpenGL FAQ / 9 Transformations. Dostopno na:
http://www.opengl.org/resources/faq/technical/transformations.htm

[57] M. Segal, K. Akeley, *The OpenGL Graphics System: A Specification (Version 2.1 - December 1, 2006)*, Silicon Graphics, Inc., dec. 2006. Dostopno na:
http://www.opengl.org/registry/doc/glspec21.20061201.pdf

[58] (2011) Matrix Structure. Dostopno na:
http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.matrix.aspx

[59] (2011) Transforms (Direct3D 9). Dostopno na:
http://msdn.microsoft.com/en-us/library/bb206269(v=vs.85).aspx

[60] (2011) What Is a Game Loop? Dostopno na:
http://msdn.microsoft.com/en-us/library/bb203873.aspx

[61] Apple Inc., *iOS Application Programming Guide*, feb. 2011. Dostopno na:
http://developer.apple.com/library/ios/documentation/iphone/conceptual/iphoneosprogrammingguide/iphoneappprogrammingguide.pdf

[62] (2011) GraphicsDeviceManager.SynchronizeWithVerticalRetrace Property. Dostopno na:

http://msdn.microsoft.com/en-us/library/microsoft.xna.framework.graphicsdevicemanager.synchronizewithverticalretrace.aspx

[63] (2009) Timer Programming Topics: Timers. Dostopno na: http://developer.apple.com/library/ios/#documentation/Cocoa/Conceptual/Timers/Articles/timerConcepts.html

[64] Apple Inc., *CADisplayLink Class Reference*, avg. 2009. Dostopno na: http://developer.apple.com/library/ios/documentation/QuartzCore/Reference/CADisplayLink_ClassRef/CADisplayLink_ClassRef.pdf

[65] (2011) Apple – iPhone 4 – Learn about the high-resolution Retina display. Dostopno na: http://www.apple.com/iphone/features/retina-display.html

[66] (2011) What is the Content Pipeline? Dostopno na: http://msdn.microsoft.com/en-us/library/bb447745.aspx

[67] (2006) Build it ahead of time – Shawn Hargreaves Blog. Dostopno na: http://blogs.msdn.com/b/shawnhar/archive/2006/11/07/build-it-ahead-of-time.aspx

[68] (2011) Instructions on using Xni framework for advanced coders. XNI Framework, XNA for iOS – Google Project Hosting. Dostopno na: http://code.google.com/p/xni/wiki/NoNonsenseInstructions

[69] (2006) Framework Programming Guide: What are Frameworks? Dostopno na: http://developer.apple.com/library/mac/#documentation/MacOSX/Conceptual/BPFrameworks/Concepts/WhatAreFrameworks.html

[70] (2011) Xcode 4: make a library in one file that works on BOTH device AND simulator – Red Glasses. Dostopno na: http://red-glasses.com/index.php/tutorials/xcode4-make-a-library-in-one-file-that-works-on-both-device-and-simulator/

[71] (2011) Downloads, XNI Framework. XNA for iOS – Google Project Hosting. Dostopno na: http://code.google.com/p/xni/downloads/list

[72] M. Jan, *Razvoj računalniških iger za iPhone in iPad z uporabo ogrodja XNI*, učno gradivo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2011.

[73] (2011) How to use XNI Framework. XNI Framework, XNA for iOS – Google Project Hosting. Dostopno na:
`http://code.google.com/p/xni/wiki/Instructions`

[74] (2011) Index for the overview of supported classes in XNI. XNI Framework, XNA for iOS – Google Project Hosting. Dostopno na:
`http://code.google.com/p/xni/wiki/Reference`

[75] P. Peer, B. Klemenc, M. Jan, *Tehnologija iger*, učno gradivo, Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2011.

[76] (2011) The common pitfalls of setting up your project to use Xni. XNI Framework, XNA for iOS – Google Project Hosting. Dostopno na:
`http://code.google.com/p/xni/wiki/FAQ`

[77] (2011) XNI Class 2010/11 Student Showcase – YouTube. Dostopno na:
`http://www.youtube.com/watch?v=c86nz-i0hj0`

[78] (2011) Monkey Labour – Press Page. Dostopno na:
`http://dawnofplay.com/monkeylabour/press/`

[79] J. Richter, "Garbage Collection: Automatic Memory Management in the Microsoft .NET Framework," *MSDN Magazine*, let. 1, št. 9, 2000.

[80] Apple Inc., *Memory Management Programming Guide*, mar. 2011. Dostopno na:
`http://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/MemoryMgmt/MemoryMgmt.pdf`

[81] (2007) Twin paths to garbage collector nirvana – Shawn Hargreaves Blog. Dostopno na:
`http://blogs.msdn.com/b/shawnhar/archive/2007/07/02/twin-paths-to-garbage-collector-nirvana.aspx`

[82] (2008) Method chaining. Dostopno na:
`http://firstclassthoughts.co.uk/java/method_chaining.html`

[83] (2011) iOS 5 for Developers – Apple Developer. Dostopno na:
`http://developer.apple.com/technologies/ios5/`

[84] (2011) Xcode 4 User Guide: Building and Running Your Code. Dostopno na:
http://developer.apple.com/library/ios/#documentation/ToolsLanguages/Conceptual/Xcode4UserGuide/Building/Building.html

[85] (2011) Apple's iPhone is the Most Popular Phone in U.S. – Yahoo! Finance. Dostopno na:
http://finance.yahoo.com/news/Apple-iPhone-Most-Popular-wscheats-2452949058.html?x=0&.v=1

[86] (2011) Daring Fireball: The iPad's Dominance of the Tablet Market. Dostopno na:
http://daringfireball.net/2011/07/ipad_dominance

[87] (2011) Xbox Live Indie Games: no way to make a living. Dostopno na:
http://arstechnica.com/gaming/news/2011/07/xblig-feature.ars

[88] (2010) XNA Game Studio 4.0 Available for Download! – XNA Game Studio Team Blog. Dostopno na:
http://blogs.msdn.com/b/xna/archive/2010/09/16/xna-game-studio-4-0-available-for-download.aspx

[89] (2011) JSIL. Dostopno na:
http://jsil.org/

[90] (2011) Low Level Virtual Machine — Wikipedia, The Free Encyclopedia. Dostopno na:
http://en.wikipedia.org/w/index.php?title=Low_Level_Virtual_Machine&oldid=444400153

[91] (2011) cocos2d for iPhone. Dostopno na:
http://www.cocos2d-iphone.org/