Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Filip Samotorčan

Igra Breakout za Android

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Borut Batagelj

Ljubljana 2012

Besedilo je oblikovano z urejevalnikom besedil 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: 00287/2012
Datum: 11.04.2012

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

Kandidat: **FILIP SAMOTORČAN**

Naslov: **IGRA BREAKOUT ZA ANDROID**
BREAKOUT GAME FOR ANDROID

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Kandidat naj izdela izobraževalno igro za mobilne naprave z operacijskim sistemom Android. Igra naj vsebuje izobraževalne elemente. Pri igranju naj izkorišča senzorje mobilne naprave. V ta namen naj izdela splošno uporabno programsko knjižnico za razvoj računalniških iger za platformo Android ter preuči in uporabi ustrezen fizikalni pogon.

Mentor:

viš. pred. dr. Borut Batagelj

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Filip Samotorčan, z vpisno številko **63080317**, sem avtor diplomskega dela z naslovom:

Igra Breakout za Android

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Borut Batagelj,
- 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 4. avgust 2012 Podpis avtorja:

Rad bi se zahvalil staršem, ki so me spodbujali in podpirali med študijem. Posebna zahvala tudi mentorju viš. pred. dr. Borut Batagelj, ki mi je pomagal in usmerjal.

Kazalo

Povzetek

V diplomski nalogi je opisan razvoj igre Breakout za operacijski sistem Android. Najprej so opisane osnove igre ter njena zgodovina. Sledi opis tehnologije in orodij uporabljenih v izdelavi. Izdelava aplikacije je razdeljena na dva dela. V prvem delu je opisana izdelava 2D knjižnice za delo z knjižnico OpenGL. Tukaj se nahajajo vse OpenGL operacije, ki so se uporabile pri izdelavi aplikacije. V tem delu se tudi nahaja opis dela z zvokom, dotiki in nagibi na Android platformi. Celotna knjižnica je napisana v programskem jeziku Java. V drugem delu se nahaja opis implementacije aplikacije s pomočjo te knjižnice. V tem delu je tudi opisana uporaba knjižnice Box2D za uporabo pri računanju fizike.

Ključne besede: Android, Java, Eclipse, OpenGL, Box2D, Breakout

Abstract

The thesis describes the development of an Breakout game for Android operating system. First are described the basics and the game's history. Next come the tools and technology used in the development. Production of the application is divided into two parts. The first part explains how the 2D library was created for work with the OpenGL. Here are described all the OpenGL operations, which were used in the making of the game. This section also describes work with sound, touch and tilts on the Android platform. The entire library is written in Java programming language. The second part describes the implementation of the game using this library. This section also describes how to use Box2D library for calculation of physics.

Poglavje 1

Uvod

Mobilne tehnologije so v zadnjih letih zelo napredovale v zmogljivosti, kot tudi razširjenosti med uporabniki. Skoraj vsak od nas ima in uporablja vsaj eno mobilno napravo. Zmogljivost mobilnih naprav se tudi hitro povečuje in tudi aplikacije so vedno bolj zmogljive. Današnjo zmogljivost mobilnih naprav lahko primerjamo že z osebnimi računalniki.

Trenutno najbolj razširjen operacijski sistem za mobilne naprave je Android, ki nam s pomočjo Google Play omogoča namestitev in uporabo veliko število aplikacij. Aplikacije za Android platformo so večinoma razvite v programskem jeziku Java, vendar nam Android omogoča tudi pisanje aplikacij v programskem jeziku C++. Tudi sam uporabljam mobilni telefon z operacijski sistemom Android. Eden glavnih razlogov za razvoj aplikacije za Android je, da lahko aplikacijo razviješ v programskem jeziku Java in tudi testiranje na dejanski napravi je zelo preprosto. Drugi razlog je tudi, da smo z Android platformo bili že seznanjeni in tudi programski jezik Java smo veliko obdelali že na fakulteti. Zato je bila odločitev mobilne platforme in programskega jezika zelo lahka.

Ker smo hoteli aplikacijo narediti čimbolj hitro in odzivno smo se odločili, da bomo za izrisovanje uporabili OpenGL. Android že v celoti podpira OpenGL

v programskem jeziku Java tako, da ni bilo potrebnih nobenih implementacij v programskem jeziku C++. Tudi z OpenGL smo bili že seznanjeni pri izdelavi igre za operacijski sistem iOS [33] tako, da je bila implementacija v OpenGL preprosta in lahka. Vse OpenGL ukaze, ki smo jih uporabili za izris, smo implementirali v posebni knjižnici. Odločitev za to je bila neodvisnost naše igre do OpenGL in uporabnost knjižnice v nadaljnjih projektih.

Izdelavo celotne aplikacije smo naredili z orodjem Eclipse. Poleg orodja Eclipse smo uporabili tudi Android SDK, ki nam omogoča namestitev testne aplikacije na dejansko mobilno napravo, kot tudi razhroščevanje. Za testiranje smo uporabili dejansko mobilno napravo in ne simulator, ki se dodatno nahaja v Android SDK. Odločitev za to je bila zelo slaba podpora OpenGL v simulatorju. Če pogledamo celotno igro, se na mobilni napravi izvaja v povprečju 55 sličic na sekundo, če pa poženemo isto igro na simulatorju se izvaja v povprečju 3 sličice na sekundo tako, da testiranje, zaradi počasnosti, ni bilo mogoče na simulatorju.

Poglavje 2

Igra Breakout

V tem poglavju so opisane osnove igre Breakout in njena zgodovina.

2.1 Zgodovina

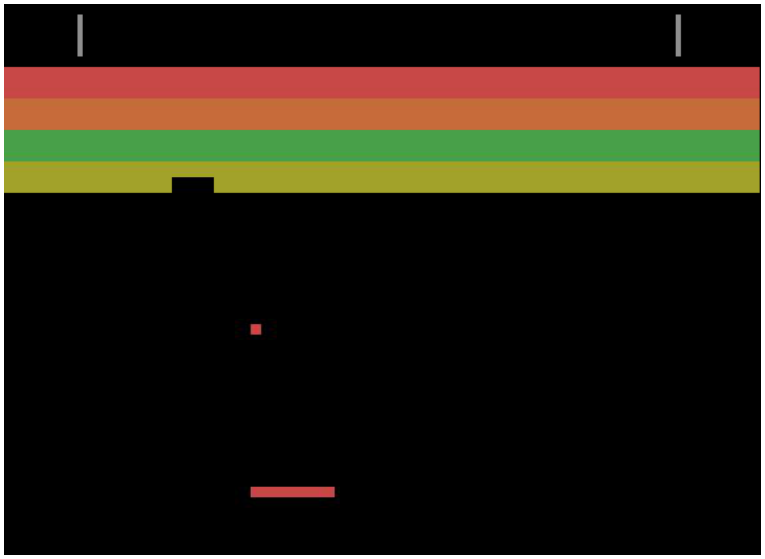
Igro Breakout sta zasnovala Nolan Bushnell in Steve Bristow potem, ko se je slednji pridružil podjetju Atari. Imela sta idejo spremeniti obstoječo igro Pong za igranje z enim igralcem, kjer bi igralec usmerjal žogo s ploščkom, da podre vse opeke. Bila sta prepričana, da bo igra uspela in začeli so izdelovati koncept igre. Alcorn je bil dodeljen kot vodja projekta in razvoj s Cyan Engineering se je pričel leta 1975. Istega leta je Alcorn dodelil Steve Jobsu izdelavo prototipa s ponudbo US $750. Steve Jobs je obljubil izdelavo prototipa v štirih dneh.

Jobs je opazil njegovega prijatelja Steve Wozniak zaposlenega v Hewlett-Packard, ki je bil zmožen izdelave modela z manjšim številom elektronskih čipov in ga povabil v sodelovanje z delitvijo dobička $750. Za prihranek na komponentah je Wozniak imel majhen trik, ki ga ostali inženirji niso razumeli. Prvotni rok štirih dni je bil izpolnjen in Jobs ter Wozniak sta si razdelila dobiček $750.
Atari ni mogel uporabiti prototipa, ki ga je sestavil Steve Wozniak. Z obli-

kovanjem plošče z manjšim število elektronskih čipov je proizvodnja postala prezapletena. Atari je končal svojo različico, ki je bila lažja za proizvodnjo, vendar je bila sestavljena iz več elektronskih čipov in zato tudi dražja. [1]

2.2 Opis igre Breakout

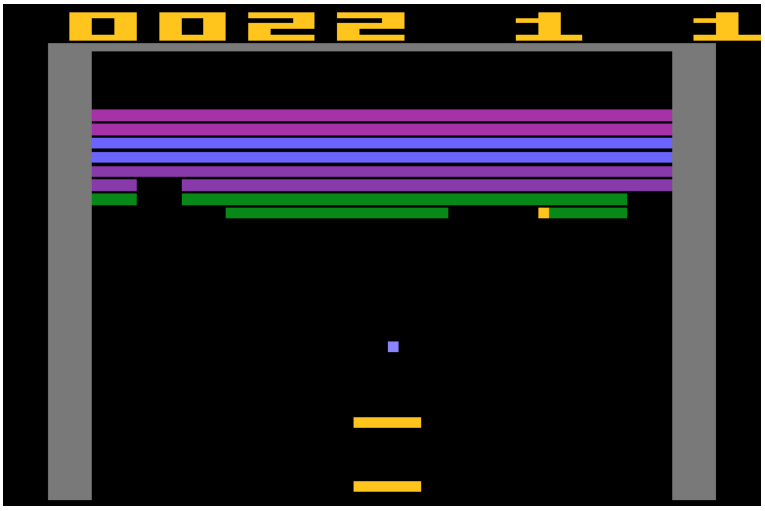
Igra Breakout [2], vidna na sliki 2.1, se začne z osmimi vrstami opek in vsaka druga vrsta opek je različne barve. Barve od spodaj navzgor so rumena, zelena, oranžna in rdeča. Z eno samo žogo mora igralec podreti čim več opek z uporabo stene in ploščka, ki ga na dnu premika igralec. Če igralec zgreši žogo s ploščkom izgubi eno življenje. Igralec ima tri življenja, da podre vse opeke. Rumene opeke prinesejo eno točko, zelene tri točke, oranžne pet točk in rdeče sedem točk. Vsakič, ko žoga zadane rdečo opeko ali strop se plošček zmanjša za polovico dolžine. Poleg tega se tudi hitrost žoge povečuje vsakih štirih zadetkov opek in vsakič, ko zadane oranžno ali rdečo opeko. Največje število točk, ki ga igralec lahko doseže je 896 z eliminacijo dveh zaslonov opek. Ko igralec podre vse opeke na drugem zaslonu je igra končana.



Slika 2.1: Prvotna igra Breakout.

Igra je postala zelo razširjena in zato je bilo narejeno tudi veliko podobnih iger. Ena izmed njih je igra **Super Breakout** [3] na sliki 2.2. Pri tej igri obstajajo tri napredne vrste igre med katerimi lahko igralec izbira.

- **Double**, da igralcu na razpolago dva ploščka, eden nad drugim, in dve žogi. Igralec izgubi življenje takrat, ko izpusti obe žogi izven igre in točke so podvojene takrat, ko sta obe žogi v igri.
- **Cavity** ohranja en plošček in eno žogo. Dve dodatni žogi sta zaprti na vsaki strani zidu, ki jih igralec lahko reši tako, da jih zadane z obstoječo žogo. Število točk, ki jih igralec dobi za zadetek opeke je odvisno od število žog v igri. Igralec izgubi življenje samo takrat, ko izpusti vse žoge izven igre.
- **Progressive** ima tudi samo eden plošček in eno žogo, vendar vsakič, ko žoga zadane plošček se strop zmanjša za število, ki je odvisno od koliko časa je žoga v igri.



Slika 2.2: Igra Super Breakout.

Druga zanimiva izpeljanka igre Breakout je tudi igra **Breakout 2000** [4] na sliki 2.3. Igra Breakout 2000 je narejena v 3D različici originalne igre Bre-

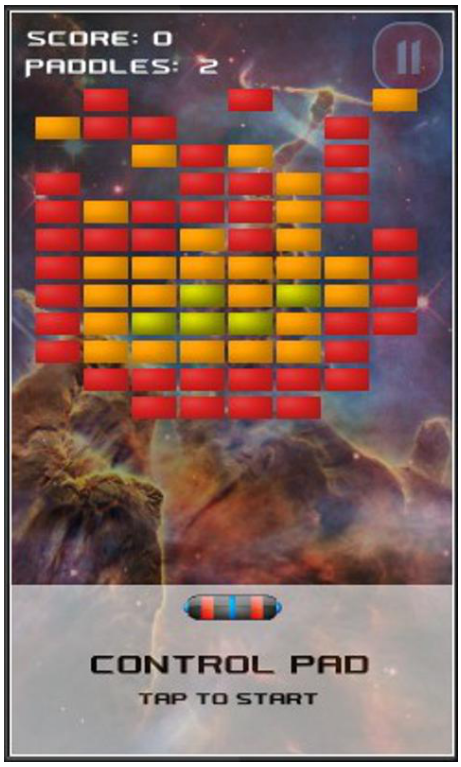
akout. Zasnovana je bila za enega ali dva igralca in cilj igre je ostal enak, vendar v 3D načinu. Skupaj je bilo deset različnih stopenj za preživeti in vsaka stopnja je bila sestavljena iz petih faz. Z vsakim končanjem stopnje ali faze je igra postala težja.

Igra vsebuje dobre in slabe moči, ki jih igralec lahko pridobi med igranjem. V igri so neuničljive opeke, opeke ki jih je potrebno zadeti večkrat in običajne opeke. Gibanje žoge je bilo omejeno na nižji raven igralnega polja in zadetek nižjih opek je povzročilo padec višjih opek v nižje igralno polje. Igra vsebuje tudi način igranja proti računalniku. V tem načinu lahko igralec usmeri žogo na računalnikovo stran igre, kjer zadetek opek povzroči dvojne točke.

Slika 2.3: Igra Breakout 2000.

Za mobilno platformo Android je zanimiva igra **Best Breakout** [5], vidna na sliki 2.4, ki je zelo podobna originalni igri Breakout s posodobljenim izgledom. Enako igra vsebuje opeke, žogo in plošček s katerim usmerjamo žogo. Igra vsebuje tri stopnje, kjer je v vsaki sobi drugačna postavitev opek in ozadje, kjer se odvija igra. Za premikanje ploščka uporabimo prst, kje z drsenjem po zaslonu premikamo plošček levo in desno. Na žalost igra ne podpira premikanje plošča z nagibi mobilne naprave. Igra uporablja zelo podobne

zvoke originalne igre Breakout zato je vzdušje zelo podobno.

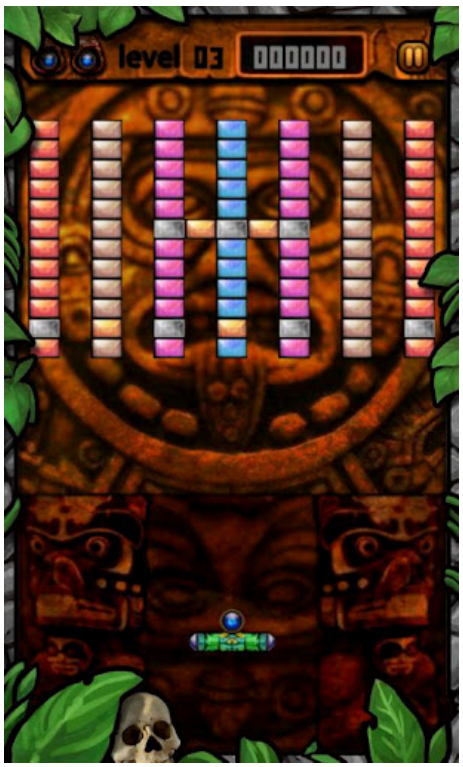


Slika 2.4: Igra Best Breakout za mobilno platformo Android.

Igra Breakout je zelo popularna zato imamo na Android platformi veliko iger, ki vsebujejo igralne elemente igre Breakout. Med te igre spadajo:

- **Break the Bricks**, na sliki 2.5, vsebuje več kot 60 stopenj, 3 vrste hitrosti igranja in spletno beleženje število doseženih točk;
- **Hitting Ball** vsebuje igranje z več kot eno žogo, različne širine ploščka, žogo, ki gre lahko skozi opeke in spletno beleženje število doseženih točk;
- **BloxBreaker** je zelo podobna igri Super Breakout, uporablja knjižnico Box2D za računanje fizike in vsebuje nadzor ploščka z nagibi mobilne naprave;

- **Brick Breaker** je zelo preprosta izvedba igre Breakout, ki vsebuje tudi spletno beleženje število doseženih točk;



Slika 2.5: Igra Break the Bricks za mobilno platformo Android.

2.3 Opis razvite igre Breakout

Naša razvita igra je zelo podobna osnovni različici igre Breakout. Imamo en plošček in eno žogo ter opeke in like, ki jih je potrebno zadeti z žogo. Imamo različne barve opek, likov, črk in številk. Ker je igra narejena za Android platformo, lahko plošček nadzorujemo tako, da nagibamo napravo ali pa če nam je lažje, lahko plošček nadzorujemo z dotikom prsta na zaslonu. Igra ima tudi nekaj poučnih stvari, kot je računanje, prepoznavanje barv, likov in besed.

Igra vsebuje šest vrst map:

- **Klasična soba** je narejena tako, da je čim bolj podobna osnovni igri Breakout. Vsebuje barvne opeke, ki jih je potrebno zadeti z žogo. Barve opek, ki jih je možno srečati so rdeča, oranžna, rumena, zelena, modra, vijolična, črna in bela. Za usmerjanje žoge igralec uporablja plošček, ki se nahaja na dnu zaslona. Vsak zadetek opeke nam doda točko in, ko podremo vse opeke se igra ponovno začne z žogo, ki je 40% hitrejša, kot prejšnja žoga. Igre ni mogoče nikoli končati, zato je cilj doseči čim več točk, preden izgubimo vsa življenja.

- **Barvna soba** na sliki 2.6 ima vse komponente prejšnje sobe z dvema dodatnima spremembama. Prva sprememba je, da za vsako barvno opeko, ki jo zadenemo, nam igra pove katere barve je bila. Drugo spremembo pa je mogoče opaziti v levem in desnem spodnjem kotu, kjer je izrisana ena od opek. Namen je zadeti tisto opeko, ki je izrisana levo ali desno spodaj za dodatne točke. Ko zadenemo tako opeko dobimo levo ali desno spodaj novo opeko za zadeti.

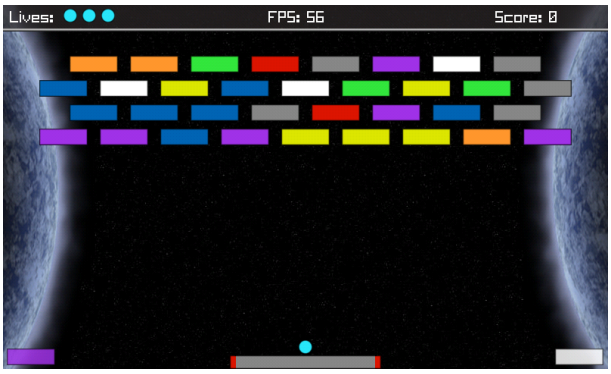
- **Soba z liki** je podobna prejšnji sobi s to razliko, da imamo namesto barvnih opek barve like. Like, ki lahko srečamo so pravokotnik, kvadrat, krog in trikotnik. Ko zadenemo lik nam igra izgovori kateri lik smo zadeli, namesto katere barve je bil lik in tudi levo in desno spodaj imamo like namesto opek.

- **Soba s številkami** na sliki 2.7 ima namesto opek, kroge s številkami. Tudi ko zadenemo številko nam igra pove katero številko smo zadeli. Namen igre je še zmeraj zadeti čim večje število številk z žogo. Dodatno imamo levo in desno spodaj račun, sestavljen iz dveh številk in rezultata. Ko zadenemo vse tri številke v računu, dobimo dodatne točke in nov račun.

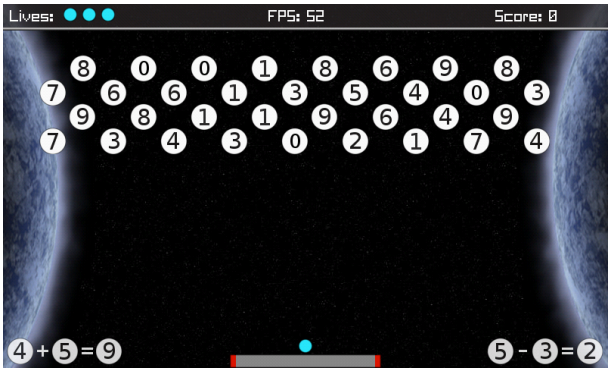
- **Soba s črkami** je sestavljena iz črk v krogu namesto opek. Podobno nam tukaj igra izgovori, katero črko smo zadeli in ne njeno barvo ali

obliko. Levo in desno spodaj se nahaja tričrkovna beseda, katero je potrebno sestaviti z zadetki črk.

- **Naključna soba** je naključno narejena vsakič, ko začnemo sobo. V tej sobi se lahko nahajajo vse barvne opeke, liki, številke in črke. Cilj igre ostaja enak, zadeti čim več opek, likov, številk in črk dokler nam ne zmanjka življenj.



Slika 2.6: Barvna soba.



Slika 2.7: Sobo s številkami.

Poglavje 3

Razvojna orodja

Najprej bomo opisali programski jezik Java v katerem je bila igra razvita. Sledi opis aplikacijskega programskega vmesnika (ang. Application programming interface) OpenGL in razvojno okolje Eclipse z Android SDK, Gimp ter opis operacijskega sistema Android.

3.1 Programski jeziki

3.1.1 Java

Java [6] je programski jezik, ki ga je leta 1995 razvil James Gosling, kot sestavni del Sun Microsystems. Od aprila 2009 je Sun Microsystems postal podružnica podjetja Oracle. Jezik izvira iz predhodnih jezikov, kot sta C in C++, vendar je veliko poenostavljen in ima manj nizko zmogljivih ukazov. Javanski programi se večinoma prevedejo v binarno obliko, ki jo potem izvaja Java Virtual Machine (JVM), neodvisno od operacijskega sistema na katerem je nameščen. Namen tega jezika je razbremeniti razvijalce, da za vsak operacijski sistem ne napišejo nove aplikacije, ampak aplikacijo napišejo samo enkrat, ki se bo izvajala na veliko različnih operacijskih sistemih. Java je od leta 2012 dalje eden najbolj priljubljenih programerskih jezikov v uporabi, še zlasti pri internetnih aplikacijah z več kot 10 milijonov uporabnikov.

3.1.2 OpenGL

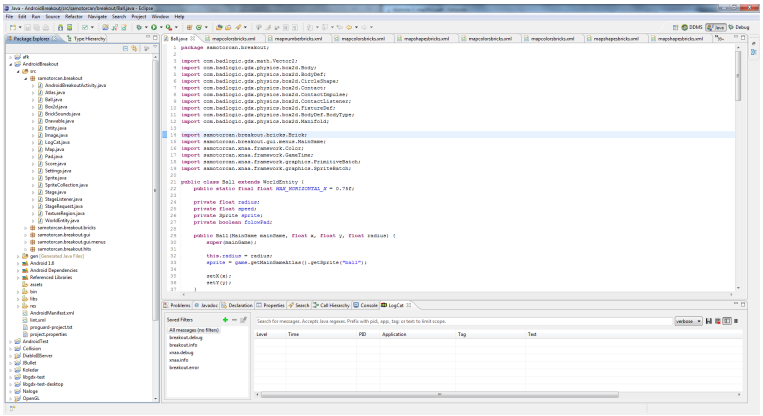
OpenGL (ang. Open Graphics Library) [7] je vmesnik, ki definira kako se izriše 2D in 3D računalniška grafika. Vsebuje več kot 250 različnih funkcij, ki se uporabljajo za izris zahtevnih 3D objektov (slika 3.1) ali za izris preprostih primitivov. OpenGL je razvilo podjetje Silicon Graphics Inc. v letu 1992 in se pogosto uporablja v navidezni resničnosti, znanstveni vizualizaciji, vizualizaciji podatkov, simulaciji letenja in v video igrah. OpenGL je oskrbovan od neprofitne organizacije Khronos Group. Trenutno je zadnja različica 4.2, vendar se za mobilne naprave uporablja prirejena različica OpenGL, ki se imenuje OpenGL ES. Zadnja različica OpenGL ES je 2.0, ki je na voljo v Android operacijskem sistemu od različice 2.2 naprej.

Slika 3.1: Čajnik izrisan z OpenGL.

3.2 Eclipse

Za razvoj naše igre smo uporabili razvojno orodje Eclipse Helios [8] na sliki 3.2. Orodje je odprtokodno in brezplačno in nam zelo olajša razvoj programske opreme skozi razvojni cikel. Primarno podpira razvoj za programski

jezik Java vendar z raznimi dodatki, ki ji je možno dodatno namestiti, Eclipse podpira Ada, C, C++, COBOL, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy in Scheme. Eclipse je možno namestitev na veliko različnih operacijskih sistemov, kar je še ena prednost pred ostalimi razvojnimi orodij.



Slika 3.2: Razvojno orodje Eclipse.

3.3 Android SDK

Poleg orodja Eclipse smo uporabljali tudi Android SDK [9], še posebej vtičnik ADT (ang. Android Development Toolkit). ADT, vtičnik za Eclipse, nam ponuja velik nabor orodij, ki so združena znotraj razvojnega orodja Eclipse. Glavna prednost vtičnika ADT je njegov grafični vmesnik, ki nam olajša delo tako, da nam ni potrebno delati v konzolni aplikaciji preko ukazne vrstice. Glavne funkcionalnosti, ki nam jih vtičnik omogoča, so:

- **Izgradnjo novih projektov za Android aplikacije.** S tem dobimo dostop do vseh dodatnih razredov in metod, ki jih lahko uporabljamo v Android aplikaciji.
- **Integracija Android SDK z orodjem Eclipse.** V orodju Eclipse se pojavijo dodatni gumbi, nastavitve, funkcionalnosti, ki jih lahko uporabljamo med razvojem aplikacije za Android.

- **Android emulator** nam omogoča, da lahko našo aplikacijo poženemo in preverimo, če pravilno deluje in kako se izvaja.

- **Preverjanje na napravi** lahko našo aplikacijo namestimo na dejansko mobilno napravo in tako preverimo razne nepravilnosti, ki se pojavijo pri emulatorju.

- **Razhroščevanje aplikacije** v Android SDK nam omogoča razhroščevanje, tako na emulatorju, kot tudi na napravi. Razhroščevalnik ima tudi dodatni nadzor nad porabljenim pomnilnikom, tako da lahko hitro ugotovimo, če napačno uporabljamo pomnilnik. Še posebej lahko to vidimo pri OpenGL, kjer je potrebno določen pomnilnik sprostiti po uporabi.

- **Urejevalnik kode** nam omogoča sprotno preverjanje sintakse programske kode, namigi za odpravljanje napaka in gradnje grafičnega uporabniškega vmesnika, za našo aplikacijo, preko klikov in potegov. Tako nam ni potrebno pisati kode za postavitev in izgled grafičnega uporabniškega vmesnika. Omogoča tudi možnost pisanja grafičnega uporabniškega vmesnika v jeziku XML.

- **LogCat** nam omogoča pisanje v določeno okno v Eclipsu tako, da v kodi pokličemo metodo z besedilom za izpis. Predvsem se uporablja poleg razhroščevanja aplikacije, za lažje sledenje in izpis določenih spremenljivk.

- **Z zajemom slike** lahko iz naprave ali emulatorja shranimo sliko zaslona. To je še posebej uporabno, ko hočemo dobiti trenutno sliko zaslona na napravi medtem, ko na emulatorju lahko preprosto zajamemo sliko celotnega zaslona na računalniku.

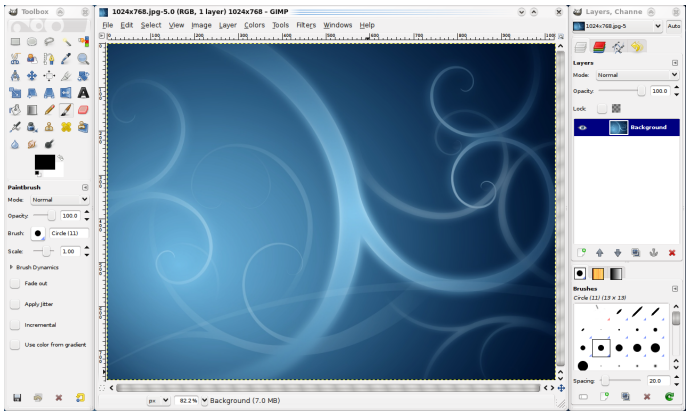
- **Z integrirano dokumentacijo za Android** lahko z miško označimo metodo ali razred in pojavi se nam opis te metode ali razreda. V podrobnostih si lahko preberemo tudi daljši opis in primer uporabe.

- **Digitalni certifikati** se uporabljajo pri podpisu naše aplikacije, ko jo izvozimo v paket, za namestitev na Android napravo.

3.4 Gimp

GIMP [10], na sliki 3.3, je prosto dostopen odprto kodni program za urejanje grafike. Projekt sta začela leta 1995 Spencer Kimball in Peter Mattis, zdaj pa ga vzdržuje skupina prostovoljcev. GIMP je razširjen za obdelavo digitalne grafike in fotografij s spleta. Za izbiro orodja Gimp smo se odločili zato, ker je orodje preprosto za uporabo in, ker smo se srečali z njim že v prejšnjih projektih. Orodje smo uporabljali predvsem za:

- uporabniški vmesnik, ki vsebuje: gumbe, ozadja, napise in podobno;
- glavni del igre za opeke, like, črke, številke, plošček, žoge;
- izdelava črk za napise v igri,
- ter za združevanje slik v večje slike, da smo pridobili na hitrosti celotne igre.



Slika 3.3: Aplikacija GIMP za urejanje grafike.

3.5 Operacijski sistem Android

Android operacijski sistem [11] (logotip na sliki 3.4) je odprtokodni sistem, ki temelji na Linux operacijskem sistemu. Android je trenutno eden najbolj razširjenih operacijskih sistemov s tržnim deležem že več kot 50%. Najdemo ga nameščenega na veliko vrst mobilnih napravo, kot so Samsung, HTC, Sony Ericsson itd., pri čemer vsak od proizvajalcev prilagodi izgled operacijskega sistema, da najbolj ustreza njihovi napravi. Ker je Android odprtokodni sistem nam omogoča cenejše in lažje razvijanje aplikacij. Android je zasnovan tako, da ni omejen glede velikosti zaslona, resolucije, zmogljivosti procesorja itd., zato je tudi zelo razširjen med različnimi proizvajalci. Prednost imajo tudi uporabniki, saj je aplikacij za Android veliko in so večinoma brezplačne.

Slika 3.4: Logotip operacijskega sistema Android.

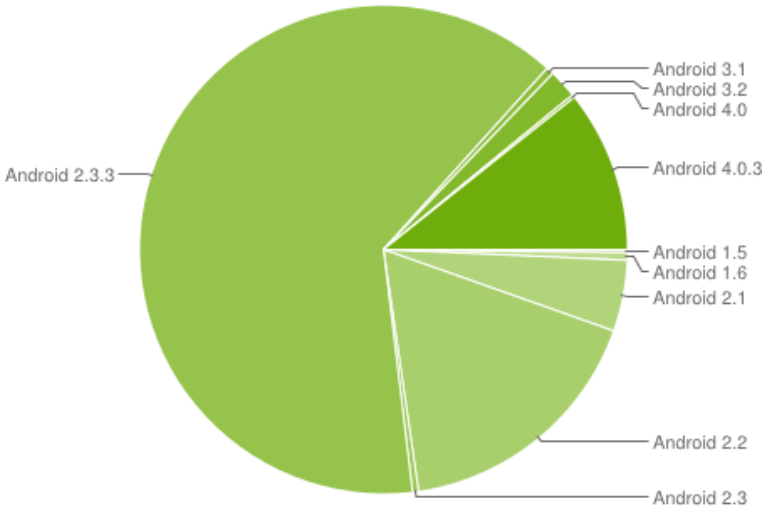
Glavne prednosti operacijskega sistema Android so:

- je brezplačen;
- je odprtokoden, kar omogoča lažje razvijanje aplikacij;
- za proizvajalce omogoča lažje in cenejše razvijanje mobilnih naprav;

- je enostaven, hiter in omogoča večopravilnost;
- vmesnik je prilagodljiv na posamezno mobilno napravo s strani proizvajalca;
- vmesnik je zelo enostaven za uporabo;
- omogoča namestitev veliko različnih aplikacij s pomočjo Google Play [12];
- enostavna posodobitev operacijskega sistema, kot tudi samih aplikacij;
- itd.

Za razvoj androida je najzaslužnejši Google, ki je ravno v ta namen ustanovil poslovno združenje več podjetij, imenovano Open Handset alliance (OHA), ter pod svoje okrilje vzel hitro rastoče podjetje Android Inc. Prva beta različica operacijskega sistema Android je bila izdana novembra 2007, naslednje leto pa je bil izdan prvi SDK. Prva različica operacijskega sistema Android je prišla na trg septembra 2008. Operacijski sistem je vseboval funkcije, kot so budilka, pregledovalnik za internet, kamero, itd. Od prve izdaje naprej je Android prejel mnogo posodobitev. Posodobitve so bile večinoma usmerjenje v odpravljanje napak in uvedbe novih lastnosti. Od aprila 2009 se vsaka nova različica Androida poimenuje po sladici. Do sedaj so bile razvite naslednje različice operacijskega sistema Android (slika 3.5 prikazuje razširjenost različic operacijskega sistema Android):

- Cupcake 1.5, API 3
- Donut 1.6, API 4
- Eclair 2.0 in 2.1, API 7
- Froyo 2.2, API 8
- Gingerbread 2.3.x, API 9-10
- Honeycomb 3.x.x, API 11-13
- Ice Cream Sandwich 4.0.x, API 14-15



Slika 3.5: Razširjenost različic operacijskega sistema Android.

3.5.1 Zgradba operacijskega sistema

Operacijski sistem Android je sestavljen iz petih elementov vidno na sliki 3.6. Ti elementi so aplikacije, njihovo ogrodje, knjižnice, prevajalnik, in Linux jedro.

Aplikacije

Vse aplikacije so napisane v programskem jeziku Java. Poleg tega se za izdelavo grafičnega uporabniškega vmesnika uporablja tudi XML [13]. Aplikacije so sestavljene v Android paket s končnico .apk. Vsaka aplikacija se požene v svojem Linux procesu. Operacijski sistem požene proces takrat, ko mu je poslana zahteva za izvajanje aplikacije. Ko z izvajanjem aplikacije zaključimo, se proces zapre. To omogoča rabo pomnilnika tudi drugim aplikacijam.

Aplikacijsko orodje

V aplikacijskem ogrodju se nahajajo vse sistemske aplikacije, ki so:

- upravljavec aktivnosti,

- upravljavec pomnilnika,
- upravljavec lokacij,
- upravljavec obvestil.

Knjižnice

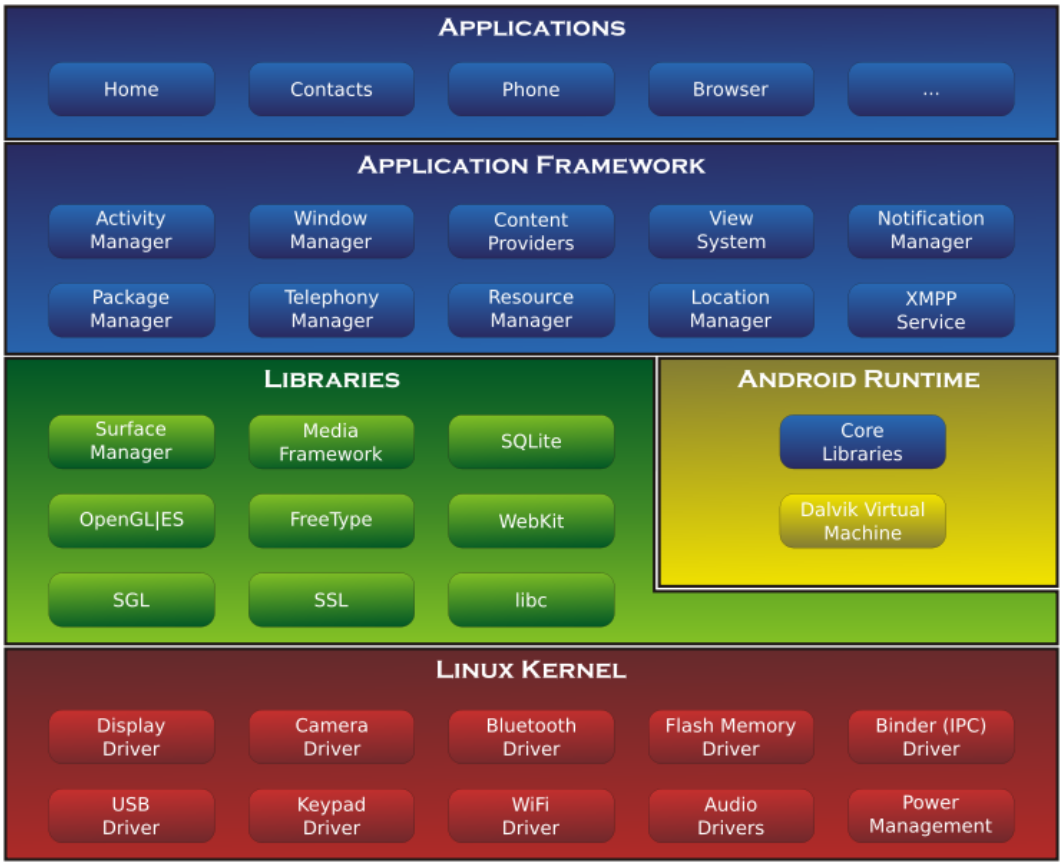
So temeljni del sistema Android. Do njih navaden uporabnik operacijskega sistema nima dostopa. Uporabljajo jih razvijalci za dostop do strojnih komponent naprave. Knjižnica, ki se v naši igri uporablja za izris je OpenGL.

Prevajalnik

Operacijski sistem Android za prevajanje kode uporablja prevajalnik JIT (ang. Just in time compiler). To omogoča prenos aplikacij na več različnih prenosnih naprav, brez ponovnega pisanja izvorne kode.

Linux jedro

Glavne naloge jedra so upravljanje s pomnilnikom, zagotavljanje varnosti, omrežne povezave, upravljanje s porabo energije itd. Prav tako jedro vsebuje gonilnike strojne opreme in predstavlja mejo med strojno opremo ter ostalimi deli operacijskega sistema.



Slika 3.6: Elementi operacijskega sistema Android.

Poglavje 4

Programske knjižnice za igre

V tem poglavju bomo opisali programske knjižnice za razvoj iger, ki nam zelo olajšajo in pohitrijo razvoj igre. Najprej bomo opisali zelo popularno knjižnico za razvoj iger imenovano XNA [15], sledi opis knjižnice XNI [14], ki je prilagoditev knjižnice XNA za iOS operacijski sistem in nazadnje še opis knjižnice libGDX [16], ki nam omogoča hkratno razvijanje za operacijske sisteme Windows [17], Linux [18], MacOS [19], Android in podpora HTML5 [20].

4.1 XNA

Knjižnico je izdelalo podjetje Microsoft za .NET programski okvir [23] (ang. Framework), ki se uporablja na operacijskih sistemih Windows in na igralni konzoli Xbox 360 [24]. Za delo z knjižnico se uporablja programski jezik C# [34], ki se prevede in izvaja s pomočjo .NET programskega okvirja. Za delo z knjižnico nam podjetje Microsoft tudi ponuja razvojno okolje Visual Studio [25] v katerem programiramo z jezikom C#. Knjižnica zna delati tudi nizko zahtevnimi stvarmi, kot so izrisovanje, delo z zvokom in podobno. Knjižnica ponuja podporo izrisa v 2D in 3D načinu s pomočjo vmesnika DirectX [35], ki komunicira z grafično kartico. Za specifične in bolj zahtevne operacije nam knjižnica ponuja tudi dostop do DirectX funkcij. Za razvoj

iger na igralni konzoli Xbox 360 imamo tudi možnost izrabe XNA Creators Club [27], ki nam omogoča objavo izdelanih iger za igralno konzolo Xbox 360. Razvoj z knjižnico XNA je tudi dokaj preprost, ker obstaja veliko primerov kako izdelati igro s pomočjo XNA knjižnice, kot tudi zelo dobra dokumentacija knjižnice in njenih funkcij. Spodnja slika 4.1 prikazuje logotip knjižnice.

Slika 4.1: Logotip knjižnice XNA.

Različice knjižnice XNA:

- **XNA Game Studio Express** je bila prva različica knjižnice, ki je bila izdana 11. decembra 2006. Primarno je bila namenjena študentom in ljudem, ki jim je ustvarjanje iger hobi. Kmalu je postala zelo popularna in lahko najdemo tudi bolj zahtevne igre, ki so narejene z knjižnico XNA.

- **XNA Game Studio 2.0** je bila izdana 13 decembra 2007. Glavna dodatna novost je bila mogoč razvoja v vseh različicah Visual Studia 2005 z vključno zastonjske različice Visual C# 2005 Express Edition [26]. Knjižnica je tudi dobila podporo za delo z Xbox Live, kjer lahko igra beleži vse dosežke in vzpone.

- **XNA Game Studio 3.0** je bila izdana 30. oktobra 2008. Dobila je podporo za Visual Studio 2008 in Visual C# 2005 Express Edition. Glavna novost je bila podpora za programski jezik C# 3.0 in pisanja LINQ [28] stavkov, ki pohitrijo in olajšajo pisanje programske kode.

- **XNA Game Studio 3.1** je bila izdana 11. junija 2009. Nagraditve, ki jih je knjižnica prejela so: podpora predvajanja videa v igrah, izboljšano delo z zvokom in podpora za delo z Xbox 360 avatarji.

- **XNA Game Studio 4.0** je bila izdana 16. septembra 2010. Glavna dodana vrednota je bila razvoj za operacijski sistem Windows Phone [21] z vključno pospešenim izrisom v 3D načinu. Dodana je bila tudi možnost razvoja na Visual Studio 2010. Posodobitve, ki so tudi vključene v tej različici so: dotiki na zaslonu, delo z mikrofonom, izboljšano delo z zvokom in podpora programskemu jeziku Visual Basic [22].

4.2 XNI

XNI knjižnica je predelava knjižnice XNA v jezik Objective-C [29] za delo z operacijskim sistemom iOS, ki se izvaja na iPhone [31] in iPad [32] napravah. Knjižnica je še v zgodnji fazi izdelovanja in vsebuje samo glavne lastnosti knjižnice XNA. Uradno je bila tudi že izdana igra Monkey Labour [30], ki uporablja knjižnico XNI, kar dokazuje, da je knjižnica zrela za uporabo. Knjižnica se je tudi uporabljala na Fakulteti za računalništvo in informatiko, kjer je bilo izdelanih vrsto iger [36] za operacijski sistem iOS. Tako kot z knjižnico XNA je delo z knjižnico XNI lahko in preprosto. Knjižnica je razvita tako, da je čim bolj podobna knjižnici XNA, kar nam zelo olajša delo z knjižnico XNI, če že poznamo in uporabljamo knjižnico XNA. Za delo z knjižnico moramo poznati programski jezik Objective-C in imeti razvojno okolje za operacijski sistem iOS kot je Xcode. Spodnja slika 4.2 prikazuje logotip knjižnice.

4.3 libGDX

Knjižnica libGDX je napisana v programskem jeziku Java in jo lahko uporabljamo v operacijskih sistemih, ki podpirajo programski jezik Java, kot so Window, Linux, MacOS, Android in podpora spletnemu jeziku HTML5. Za

Slika 4.2: Logotip knjižnice XNI.

delo z izrisom v 2D in 3D načinu uporablja OpenGL in WebGL za izris v spletu. Knjižnica tudi nudi nativno podporo knjižnici Box2D, kar se zelo pozna pri hitrosti izvajanja na Android operacijskem sistemu.

Glavna prednost knjižnice je razvijanje igre za poljubni operacijski sistem kjer, ko končamo, imamo narejeno igro, ki se lahko izvaja na prej opisanih operacijskih sistemih. Druga prednost je tudi razvijanje igre na enem operacijskem sistemu za drugi operacijski sistem. Primer tega je celotno razvijanje igre na operacijskem sistemu Windows za operacijski sistem Android. S tem nimamo nobenih problemov z Android simulatorjem, ker lahko igro v celoti testiramo v operacijskem sistemu Windows. V času pisanja tudi posodabljajo knjižnico za podporo operacijskemu sistemu iOS za naprave iPhone in iPad. Spodnja slika 4.3 prikazuje logotip knjižnice.

Slika 4.3: Logotip knjižnice libGDX.

Poglavje 5

Razvoj 2D knjižnice

V tem poglavju bomo opisali razvoj 2D knjižnice, ki se potem uporabi pri razvoju igre. Glavni namen knjižnice je risanje v OpenGL in delo z zvokom, nagibom in pritiski na zaslon mobilne naprave. Knjižnica je narejena po zgledu knjižnice XNI in XNA, vendar za operacijski sistem Android. Vsebuje samo izris 2D objektov in ne tudi 3D, ker ni bilo potrebe za razvoj igre. Knjižnica je narejena tako, da se jo zelo lahko uporabi tudi pri razvoju drugih 2D iger za operacijski sistem Android. Knjižnica je narejena za Android različico 1.6, kar pomeni, da deluje na mobilnih napravah, ki imajo nameščeno različico Android 1.6 ali večjo. Slaba stran tega je, da ne vsebuje razredov in funkcionalnosti, ki so bile dodane po različici 1.6 (primer uporabnega razreda je XPath, za delo z XML, ki je bil dodan šele v različici 2.2 [37])

5.1 Uporaba

Za uporabo knjižnice je potrebno narediti nekaj sprememb osnovnemu projektu Android aplikacije. V osnovnem razredu projekta je potrebno podedovani razred *Activity* [39] spremeniti v razred *Game*. S tem knjižnica poskrbi, da se pravilno ustvarijo vse potrebne stvari, kot so OpenGL, zvok, pritiski itd. (v kodi 5.1 se nahajajo spremembe, ki jih je potrebno narediti za upo-

rabo knjižnice). Sedaj je vse pripravljeno za uporabo knjižnice.

Koda 5.1: Glavni razred pripravljen za delo z knjižnico.

```
1  import samotorcan.xnaa.framework.Game;
2  import samotorcan.xnaa.framework.GameTime;
3
4  public class TestAndroidActivity extends Game {
5      @Override
6      protected void loadContent() {
7          super.loadContent();
8          // nalaganje slik, zvokov, pisav
9      }
10
11     @Override
12     protected void update(GameTime gameTime) {
13         super.update(gameTime);
14         // update koda
15     }
16
17     @Override
18     protected void draw(GameTime gameTime) {
19         super.draw(gameTime);
20         // draw koda
21     }
22 }
```

Glavne dve metodi, ki jih je potrebno implementirati sta *update* in *draw*. V metodi *update* se nahajajo vse kalkulacije, ki jih igra izvaja. V metodi *draw* pa se nahaja izris igre. V metodi *loadContent* se nahajajo vsi ukazi, ki nalagajo slike, pisave in zvok. Knjižnica tudi vsebuje veliko razredov. Glavni med njimi so:

- **SpriteBatch** se uporablja za izris 2D objektov v OpenGL;
- **ContentManager** je narejen za nalaganje slik, zvokov, pisav;
- **Texture** vsebuje sliko, ki je bila naložena s pomočjo razreda *ContentManagerja*;
- **SpriteFont** vsebuje naloženo pisavo;
- **SoundEffect** vsebuje naložen zvok;

- **SoundCollection** hrani in upravlja objekte razreda *SoundEffect*;
- **TouchPanel** hrani pritiske na mobilno napravo;
- **AccelerometerPanel** vsebuje nagibe mobilne naprave;
- **GraphicsDeviceManager** se uporablja za nastavitve in branje vrednosti mobilne naprave, kot so: širina, višina, celotni zaslon, zatemnitev mobilne naprave in podobno.

5.2 SpriteBatch

Razred *SpriteBatch* se uporablja za izris slik in napisov v OpenGL. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.2.1 Uporaba

Prvo je potrebno narediti nov objekt razreda *SpriteBatch*. Razred vsebuje en konstruktor, ki sprejme objekt razreda *GraphicsDevice*. Objekt dobimo tako, da na glavnem razredu Android aplikacije pokličemo metodo *getGraphicsDevice*, ki nam vrne objekt. Ko imamo ustvarjen *SpriteBatch* moramo za izris najprej poklicati metodo *begin*, potem en ali več klicev metod za izris in na koncu metodo *end*. Ko pokličemo metodo *end* se podane slike izrišejo na zaslon.

Izris slike

Razred *SpriteBatch* vsebuje veliko metod za izris slike, ena najbolj preprostih je ***draw(texture, position, sourceRectangle, color)***, ki vsebuje parametre:

- **texture** je slika, ki jo naložimo s pomočjo razreda *ContentManager*;
- **position** zahteva, da podamo objekt razreda *Vector*, ki opisuje, kje na zaslonu se naj izriše slika;

- **sourceRectangle** obreže sliko tako, da se na zaslonu prikaže samo tisto, kar se nahaja v podanem pravokotniku;
- **color** je barva s katero bo slika obarvana (če hočemo sliko izrisati v naravni barvi podamo belo barvo).

Ostale metode *draw* vsebujejo opcijske parametre, kot so rotacija, skalacija in podobno. Paziti je potrebno tudi, da se vsi klici za izris nahajajo v metodi draw glavnega razreda Android. Primer inicializacije razreda *SpriteBatch* in izris slike se nahaja v kodi 5.2.

Koda 5.2: Primer uporabe razreda *SpriteBatch* za izris slike.

```
1  ...
2  @Override
3  protected void draw(GameTime gameTime) {
4      super.draw(gameTime);
5
6      SpriteBatch spriteBatch = new SpriteBatch(getGraphicsDevice());
7      spriteBatch.begin();
8          spriteBatch.draw(
9              getContent().loadTexture2D(R.drawable.ic_launcher),
10             new Vector2(50, 50),
11             new Rectangle(0, 0, 100, 100),
12             Color.white()
13         );
14     spriteBatch.end();
15 }
16 ...
```

Izris napisa

Za izris napisa je potrebno poklicati metodo ***public void drawString(spriteFont, text, position, color, rotation, origin, scale, effects, layerDepth)***, ki vsebuje parametre:

- **spriteFont** je pisava, ki se uporabi za izris napisa;
- **text** vsebuje besedilo, ki se izriše;
- **position** vsebuje koordinate, kjer se izriše napis;

- **color** je barva s katero obarvamo napis;
- **rotation** pove za koliko naj bo napis zavrten;
- **origin** predstavlja točko, kjer se vrši rotacija;
- **scale** vsebuje skalacijo po x in y osi;
- **effects** so dodatne modifikacije, kot je zrcaljenje;
- **layerDepth** določa kateri napis prekriva drugi napis, če se napisa prekrivata.

Primer izris napisa "OpenGL"prikazuje koda 5.3.

Koda 5.3: Primer uporabe razreda *SpriteBatch* za izris napisa "OpenGL".

```
...
@Override
protected void draw(GameTime gameTime) {
    super.draw(gameTime);

    SpriteBatch spriteBatch = new SpriteBatch(getGraphicsDevice());
    spriteBatch.begin();
        spriteBatch.drawString(
            getContent().loadSpriteFont(R.drawable.ic_launcher),
            "OpenGL",
            new Vector2(50, 50),
            Color.white(),
            0,
            Vector2.zero(),
            Vector2.one(),
            SpriteEffects.NONE,
            0
        );
    spriteBatch.end();
}
...
```

5.2.2 Razvoj

Razred *SpriteBatch* je sestavljen iz treh glavnih metod: *begin*, *end* in variacije metode *draw* za izris slike. Za izris napisa ima razred *SpriteBatch* metodo *drawString*.

Begin

Metoda *begin* ne zahteva nobenih parametrov in vse kar naredi je, da pripravi tabelo za hranjenje ukazov, ki se bodo izvršili v metodi *end*. V tej metodi se tudi preveri, da se metoda *begin* pokliče samo enkrat v paru z metodo *end*.

Draw

Metoda *draw* in vse njene variacije, shranijo podane parametre v tabelo, ki jo je pripravila metoda *begin*. Preveri se tudi, da se je pred klicem metode *draw* poklicala tudi metoda *begin*.

End

V metodi *end* se zgodi izris vseh podanih ukazov, ki so shranjeni v prej omenjeni tabeli. Izvajanje metode *end* se vrši v podanem vrstnem redu:

1. **preverjanje klica** preveri, da se je metoda *end* poklicala po klicu metode *begin*;
2. **OpenGL enable** zagotovi vse potrebne funkcionalnosti, ki se nadaljnje uporabljajo za izris v OpenGL;
3. **sortiranje po sliki** sortira slike tako, da se enake slike nahajajo ena za drugo;
4. **v pripravi za izris** se izračunajo koordinate objekta, barve slike, koordinate slike, rotacija in skalacija;
5. **izris** izriše sliko s pomočjo metode *glDrawElements*.
6. **OpenGL disable** onemogoči funkcionalnosti OpenGL, ki so bile omogočene v koraku OpenGL enable.

Glavni del izrisa se zgodi z metodo ***glDrawElements (mode, count, type, indices)*** [38], ki zahteva:

- **mode** predstavlja kakšen objekt se bo izrisal;

- **count** je število objektov, ki jih bomo izrisali;
- **type** pove kakšnega tipa so koordinate tabele;
- **indices** so koordinate za tri tabele v katerih se nahajajo koordinate objekta, barve in koordinate slike;

Objekt izrišemo tako, da ga sestavimo iz dveh trikotnikov, kjer v treh tabelah podamo koordinate objekta, barve in koordinate slike. Ker metoda *glDrawElements* lahko izriše več enakih objektov z isto sliko, to metodo pokličemo samo enkrat za vsako različno sliko. Tukaj se lepo vidi zakaj je pametno slike združiti v eno veliko sliko, saj se bo metoda *glDrawElements* poklicala samo enkrat in tako bomo veliko pridobili na hitrosti izvajanja. Spodnja koda 5.4 prikazuje glavni del metode *end*, kjer se zgodi izris z OpenGL.

Koda 5.4: Izris z OpenGL.

```
1  // se premakne v sredisce koordinatnega sistema
2  gl.glLoadIdentity();
3
4  // podamo tabelo koordinat objekta
5  gl.glVertexPointer(2, GL10.GL_FLOAT, 0, floatBuffer(vertexArray));
6
7  // podamo tabelo barv
8  gl.glColorPointer(4, GL10.GL_FLOAT, 0, floatBuffer(colorArray));
9
10 // podamo tabelo koordinat slike
11 gl.glTexCoordPointer(2, GL10.GL_FLOAT, 0, floatBuffer(texArray));
12
13 // povemo katero sliko bomo uporabili
14 gl.glBindTexture(GL10.GL_TEXTURE_2D, array.get(0).texture.getTextureId());
15
16 // izrisemo n trikotnikov s koordinatami objekta, barvami in koordinatami slike
17 gl.glDrawElements(GL10.GL_TRIANGLES, 6 * array.size(),
18     GL10.GL_UNSIGNED_SHORT, shortBuffer(indexArray));
```

DrawString

Metoda *drawString* deluje tako, da preprosto za vsak znak izriše sliko s pomočjo metode *draw*. Vse kar se dodatno računa v metodi *drawString* je odmik med znaki, ki je podan v pisavi, ter rotacija, ki zavrti celoten napis in ne vsak znak posebej.

5.3 ContentManager

Razred *ContentManager* se uporablja za nalaganje slik, zvokov in pisav. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.3.1 Uporaba

Za uporabo razreda *ContentMangarja* nam ni potrebno narediti novega objekta, ker za to že poskrbi glavni razred Android aplikacije, ki deduje razred *Game*. Za pridobitev razreda *ContentManager* preprosto pokličemo metodo *getContent* na glavnem razredu Android aplikacije. Razred *ContentManager* vsebuje tri glavne metode:

- **loadTexture2D**, ki naloži sliko za uporabo pri izrisu s pomočjo razreda *SpriteBatch*;
- **loadSoundEffect** naloži zvok, ki ga lahko predvajamo z metodo *play*;
- **loadSpriteFont** naloži pisavo, ki jo uporabimo pri izrisu napisa s pomočjo razreda *SpriteBatch*.

Za pridobitev slike preprosto pokličemo metodo *loadTexture2D* in podamo ime slike, ki jo želimo naložiti. Ime slike je ključ, ki ga dobimo s pomočjo Android ADT vtičnika, ki nam, vsakič ko dodamo sliko v mapo z imenom "res", ustvari enoličen ključ v statičnem razredu *R*. Potrebno je tudi paziti, da se vsa nalaganja primarno zgodijo v metodi *loadContent* glavnega razreda Android zato, ker nalaganje med izvajanjem igre veliko vpliva na hitrost. Primer nalaganja slike se nahaja v kodi 5.5.

Koda 5.5: Nalaganje slike.

```
1 @Override
2 protected void loadContent() {
3     super.loadContent();
4
5     Texture2D texture = getContent().loadTexture2D(R.drawable.ic_launcher);
6 }
```

Za sprostitev pomnilnika, ko ne rabimo več slike, preprosto pokličemo metodo *dispose* na objektu slike.

5.3.2 Razvoj

Razred *ContentManager* vsebuje shranjevanje že naloženih slik, zvokov in pisav. Če pokličemo nalaganje enake slike dvakrat, se v resnici slika naloži samo v prvem klicu v drugem klicu pa se vrne shranjena slika iz prvega klica. Tako preprečimo možne napake, kjer bi naložili že naloženo sliko in tako po nepotrebnem zasedli pomnilnik z enako sliko.

Metodi *loadNewTexture2D* in *loadSpriteFont* naložita sliko s pomočjo metode *BitmapFactory.decodeResource* [40], kjer podamo primarni ključ metode, ki nam ga ustvari Android ADT vtičnik. Metoda *loadSoundEffect* pa dejansko ne nalaga zvoka ampak to prepusti konstruktorju razreda *SoundEffect*. Implementacija metode *loadSpriteFont* je prikazano v kodi 5.6.

Koda 5.6: Implementacija metode *loadSpriteFont*.

```
1  public SpriteFont loadSpriteFont(int resourceId) {
2      if (!fonts.containsKey(resourceId)) {
3          BitmapFactory.Options options = new BitmapFactory.Options();
4          options.inScaled = false;
5          Bitmap bitmap = BitmapFactory.decodeResource(
6              game.getResources(),
7              resourceId,
8              options
9          );
10
11         SpriteFont font = new SpriteFont(bitmap, resourceId, game);
12         fonts.put(resourceId, font);
13     }
14     return fonts.get(resourceId);
15 }
```

OpenGL ima tudi omejitev maksimalne velikosti slik, kot tudi omejitev velikosti slike. Največja velikost slike je odvisna od naprave zato ima OpenGL posebno metodo, ki nam pove maksimalno velikost slike. Velikost slike pa je omejena na velikost dva na potenco celega števila. Primer veljavne velikosti

slike je 512, ker 2 na potenco 9 je 512. Primer neveljavne velikosti je 800, ker ne obstaja celo število, ki bi ustrezalo 2 na potenco nekaj je 800. Zaradi tega ima razred *ContentManager* tudi možnost razširitve slike, ki razširi sliko s transparentno barvo do ustrezne velikosti. Tako v primeru velikosti 800, sliko razširi s transparentno barvo na velikost 1024. Del algoritma za razširitev slike je prikazan v kodi 5.7.

Koda 5.7: Del algoritma za razširitev slike.

```
int newWidth = roundUpToPowerOfTwo(bitmap.getWidth());
int newHeight = roundUpToPowerOfTwo(bitmap.getHeight());

int[] pixels = new int[bitmap.getWidth() * bitmap.getHeight()];
bitmap.getPixels(
    pixels,
    0,
    bitmap.getWidth(),
    0,
    0,
    bitmap.getWidth(),
    bitmap.getHeight()
);

int[] newPixels = resizeBitmap(
    pixels, bitmap.getWidth(),
    bitmap.getHeight(),
    newWidth,
    newHeight
);
Bitmap powerOfTwoBitmap = Bitmap.createBitmap(
    newPixels,
    newWidth,
    newHeight,
    bitmap.getConfig() == null ? Bitmap.Config.ARGB_4444 : bitmap.getConfig()
);

Texture2D texture = new Texture2D(
    powerOfTwoBitmap,
    resourceId,
    bitmap.getWidth(),
    bitmap.getHeight(),
    game
);

bitmap.recycle();
```

5.4 Texture

Razred *Texture* se uporablja za dejansko nalaganje slike v OpenGL, ter shranjevanje slike. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.4.1 Uporaba

Večinoma objekt razreda *Texture* nikoli ne bomo ročno ustvarili, ker zato poskrbi razred *ContentManager* in prepreči nalaganje enakih slik. Razred ima veliko metod, med njimi so najbolj uporabne metode:

- **dispose** sprosti pomnilnik, ki ga zaseda slika (slika je neuporabna po tem klicu);
- **isDisposed** preveri če je pomnilnik, ki ga slika zaseda, sproščen;
- **getResourceId** vrne primarni ključ, ki ga je dodelil Android ADT vtičnik;
- **getTextureId** vrne primarni ključ, ki ga je dodelil OpenGL;
- **getWidth** je širina slike brez razširjanja, če velikost slike ni bila ustrezna;
- **getHeight** je višina slike brez razširjanja, če velikost slike ni bila ustrezna;
- **getBitmapWidth** je dejanska širina slike, ki se nahaja v pomnilniku;
- **getBitmapHeight** je dejanska višina slike, ki se nahaja v pomnilniku;

5.4.2 Razvoj

Razred *Texture* preprosto naloži že naloženo sliko v pomnilniku v OpenGL. To naredi s pomočjo metode *GLUtils.texImage2D* [41], ki sprejme štiri parametre:

- **target** vedno GL10.GL_TEXTURE_2D za običajne slike;
- **level** vedno 0 za nalaganje slik;
- **bitmap** slika, ki je že naložena v pomnilniku;
- **border** vedno 0 za slike.

Pred klicem te metode je potrebno ustvariti enolični ključ, ki je vezan na sliko. To naredimo s pomočjo metode *glGenTextures* [42]. Ko imamo enolični ključ povemo, z metodo *glBindTexture* [43], da bomo od sedaj naprej delali z sliko, ki je vezana na ta ključ. Za dodajanje slike na izbrani ključ pokličemo metodo *GLUtils.texImage2D*, ki veže podano sliko s trenutnim ključem, ki je bil izbran z metodo *glBindTexture*. Sliko je potrebno tudi sprostiti v pomnilniku, to storimo z metodo *recycle*, ker je sedaj shranjena v OpenGL. Spodnja koda 5.8 prikazuje ustvarjanje slike v OpenGL.

Koda 5.8: Inicializacija slike v OpenGL.

```
1 int[] texId = new int[1];
2 gl.glGenTextures(1, texId, 0);
3 this.textureId = texId[0];
4
5 gl.glBindTexture(GL10.GL_TEXTURE_2D, textureId);
6
7 GLUtils.texImage2D(GL10.GL_TEXTURE_2D, 0, bitmap, 0);
8
9 bitmap.recycle();
```

Ko slike več ne rabimo jo je potrebno tudi sprostiti iz OpenGL pomnilnika. To storimo s pomočjo metode *glDeleteTextures* [44], kjer podamo enolični ključ, ki smo ga dobili ob ustvarjanju slike v OpenGL. To stori metoda *dispose* v razredu *Texture*.

5.5 SpriteFont

Razred *SpriteFont* se uporablja za hranjenje slik znakov za izbrano pisavo. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.5.1 Uporaba

Glavni namen objekta razreda *SpriteFont* je hranjenje slik znakov, ki jih uporabi razred *SpriteBatch* za izris napisa. Objekt razreda *SpriteBatch* nam ni potrebno nikoli ročno ustvariti, ker nam ga ustvari razred *ContentManager* (vidno v kodi 5.9) po potrebi. Najbolj uporabne metode v razredu so:

- **getCharacters**, ki nam vrne vse znake, ki jih pisava zna izrisati;
- **getSpriteCharacter** vrne sliko za zahtevan znak;
- **getSpacing** nam pove razmik med znaki;
- **setSpacing** nastavi razmik na želeno dolžino;
- **measureString** izmeri dolžino besedila, ki ga podamo.

Ko objekt razreda *SpriteFont* ne potrebujemo več je potrebno pomnilnik tudi sprostiti z metodo *dispose*.

Koda 5.9: Pridobitev pisave za izris besedila.

```
1 @Override
2 protected void loadContent() {
3     super.loadContent();
4
5     SpriteFont font = getContent().loadSpriteFont(R.drawable.ic_launcher);
6 }
```

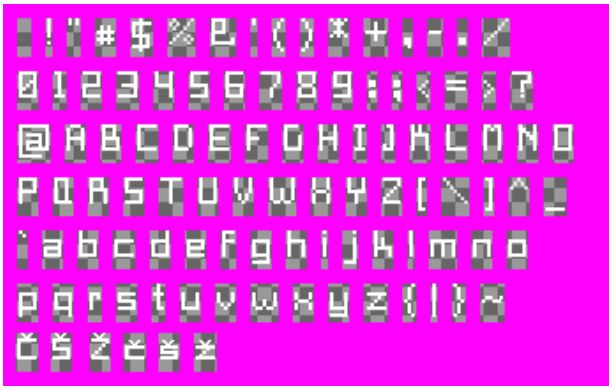
5.5.2 Razvoj

Glavni del implementacije razreda *SpriteFont* se nahaja v konstruktorju. Konstruktor sprejme tri parametre: sliko pisave, enolični ključ in primarni objekt razreda *Game*. Slika pisave mora biti v pravilnem formatu, da jo razred *SpriteFont* lahko uporabi, drugače jo preprosto zavrže.

Format slike mora biti sestavljen tako, da je ozadje znaka transparentno, ozadje slike pa roza barve. Barva znaka je lahko poljubne barve. Primer pravilne slike pisave je vidno na sliki 5.1. Razred *SpriteBatch* iz vsakega

znaka na sliki pisave ustvari novo sliko za vsak posamezen znak. Te slike znakov potem uporabi razred *SpriteBatch* za izris napisa. Enolični ključ in objekt razreda *Game* se uporabi pri ustvarjanju objekta razreda *Texture* za hranjenje slike pisave.

Razred *SpriteFont* hrani tudi zaporedje znakov, ki se nahajajo na sliki pisave. Tako lahko ugotovi kateri znak na sliki pripada črkovnemu znaku. Privzeto je zaporedje znakov nastavljeno na zaporedje, ki je vidno na sliki 5.1, vendar lahko to zaporedje spremenimo s pomočjo metode *setCharacters*. S tem smo zagotovili, da je zaporedje znakov na sliki pisave neodvisno od razreda *SpriteFont*.



Slika 5.1: Primer slike pisave.

5.6 SoundEffect in SoundCollection

Razreda *SoundEffect* in *SoundCollection* se uporabljata za hranjenje in upravljanje zvokov. V nadaljevanju bomo opisali uporabo in zgradbo razredov.

5.6.1 Uporaba

Objekt razreda *SoundEffect*, podobno kot objekt razreda *SpriteFont*, nam ni potrebno ustvariti, ker ga ustvari razred *ContentManager*, ki poskrbi, da se

enaki zvoki naložijo samo enkrat. Razred *SoundCollection* se uporablja za hranjenje vseh zvokov razreda *SoundEffect*, medtem ko razred *SoundEffect* hrani samo en zvok. Za predvajanje zvoka preprosto pokličemo metodo *play* objekta razreda *SoundEffect*. Primer nalaganje in predvajanje zvoka je vidno v kodi 5.10. Druge uporabne metode v razredu *SoundEffect* so tudi:

- **duration**, ki nam pove dolžino zvoka v sekundah;
- **getMasterVolume** nam vrne glasnost vseh zvokov;
- **setMasterVolume** nastavi glasnost vsem zvokov na podano vrednost.

Koda 5.10: Nalaganje in predvajanje zvoka.

```
1 @Override
2 protected void loadContent() {
3     super.loadContent();
4
5     SoundEffect sound = getContent().loadSoundEffect(R.raw.soundBoom);
6     sound.play();
7 }
```

5.6.2 Razvoj

Za predvajanje zvokov razred *SoundCollection* uporablja Android razred *SoundPool* [45], ki ima tri glavne metode:

- **load** naloži zvok s podanim enoličnim ključem;
- **play** predvaja zvok s podanim enoličnim ključem, ki nam ga je ustvarila metoda *load*, ter glasnost zvoka;
- **unload** sprosti pomnilnik, ki ga zaseda podani zvok.

Konstruktorju razreda *SoundPool* moramo tudi podati parameter maxStreams, ki pove koliko zvokov se lahko istočasno predvaja. S pomočjo razreda *SoundPool*, razred *SoundCollection* naloži zvok, ga predvaja in na koncu zavrže, ko ga več ne potrebuje.

Razred *SoundEffect* podpira tudi ugotavljanje dolžine zvoka. Za ugotavljanje dolžine si pomaga z Android razredom *MediaPlayer* [46], ki ima metodo *getDuration*, ki vrne dolžino zvoka v milisekundah. Ker je ugotavljanje zvoka z razredom *MediaPlayer* zelo počasno ima razred *SoundEffect* v konstruktorju parameter, ki pove ali naj ugotovi dolžino zvoka s pomočjo razreda *MediaPlayer*. Drugi parameter konstruktorja je tudi, da podamo dolžino sami, ko nalagamo zvok, kar je veliko bolj uporabno, saj dolžino zvoka lahko ugotovimo zelo lahko. V kodi 5.11 se nahaja konstruktor, kjer povemo ali naj razred *SoundEffect* sam ugotovi dolžino zvoka. V kodi 5.11 je tudi vidna uporaba razreda *SoundCollection* za nalaganje zvoka.

Koda 5.11: Eden od konstruktorjev razreda *SoundEffect*.

```
1  public SoundEffect(int resourceId, Game game, boolean loadDuration) {
2      this.resourceId = resourceId;
3      this.game = game;
4
5      this.game.getSoundCollection().add(this.resourceId);
6
7      if (loadDuration) {
8          MediaPlayer mp = MediaPlayer.create(game, this.resourceId);
9          this.duration = (double)mp.getDuration() / 1000d;
10         mp.release();
11     }
12 }
```

5.7 TouchPanel

Razred *TouchPanel* se uporablja za hranjenje pritiskov na napravi. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.7.1 Uporaba

Razred *TouchPanel* se uporablja tako, da pokličemo statično metodo *getState*, ki nam vrne objekt razreda *TouchCollection*, ki dejansko hrani pritiske na napravi. Objekt razreda *TouchCollection* hrani pritiske naprave, ki so se zgodili med enim ciklom metode *update* in *draw* razreda *Game*. Ker je lahko

dotikov na napravo več kot eden, se je potrebno odločiti katerega vzeti. V večini primerov vzamemo zadnji dotik, ki se je zgodil na zaslonu, saj nas prejšnji dotiki ne zanimajo. Ko imamo dotik razreda *TouchLocation* lahko uporabimo metode:

- **getPosition** nam vrne objekt razreda Vektor, ki vsebuje koordinate pritiska na napravi;
- **getState** nam pove kateri dotik se je zgodil (pritisk, spust, premik);
- **getId** nam vrne enolični ključ dotika, da jih lahko ločimo med seboj.

V kodi 5.12 se nahajajo ukazi za pridobitev koordinat zadnjega dotika, ki so se zgodili med ciklom metod *update* in *draw* razreda *Game*. Potrebno je tudi paziti, da nam metoda *getLastItem* lahko vrne vrednost *null*, kar pomeni, da se dotik ni zgodil v zadnjem ciklu.

Koda 5.12: Zadnji dotik na napravi.

```
1  @Override
2  protected void update(GameTime gameTime) {
3      super.update(gameTime);
4
5      TouchLocation location = TouchPanel.getState().getLastItem();
6      if (location != null) {
7          float x = location.getPosition().getX();
8          float y = location.getPosition().getY();
9      }
10 }
```

5.7.2 Razvoj

Celotni pritiski naprave so narejeni iz treh razredov:

- **TouchPanel** je statičen razred, ki se uporablja za pridobitev objekta razreda *TouchCollection*;
- **TouchCollection** hrani vse dotike v objektu razreda *TouchLocation*;
- **TouchLocation** hrani dejanski dotik, ki se je zgodil na napravi.

Da v Android aplikaciji ujamemo dotik je potrebno implementirati vmesnik *OnTouchListener*, ki ga dodamo objektu razreda *Activity* z metodo *setOnTouchListener*. V vmesniku *OnTouchListener* je potrebno implementirati metodo *onTouch*, ki nam vrne objekt razreda *MotionEvent*, ko se zgodi dotik. Iz objekta razreda *MotionEvent* ugotovimo tip in koordinate pritiska in ga shranimo v *TouchCollection*. Sedaj lahko pridemo do dotika preko statičnega razreda *TouchPanel*. Spodnja koda 5.13 prikazuje prestrezanje in shranjevanje dotika.

Koda 5.13: Prestrezanje in shranjevanje dotika.

```
@Override
public boolean onTouch(View v, MotionEvent event) {
    if (isInitialized) {
        TouchLocationState tls;
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                tls = TouchLocationState.PRESSED;
                break;
            case MotionEvent.ACTION_MOVE:
                tls = TouchLocationState.MOVED;
                break;
            case MotionEvent.ACTION_UP:
                tls = TouchLocationState.RELEASED;
                break;
            default:
                tls = TouchLocationState.INVALID;
                break;
        }
        tc.add(new TouchLocation(
            tc.count(),
            tls,
            new Vector2(event.getX(), event.getY())));
    }
}
```

5.8 AccelerometerPanel

Razred *AccelerometerPanel* se uporablja za hranjenje nagibov naprave. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.8.1 Uporaba

Razred *AccelerometerPanel* je zelo podoben razredu *TouchPanel* vendar, da namesto pritiskov hrani nagibe naprave. Nagib naprave je sestavljen iz koordinat v 3D prostoru, ki prikazujejo mobilno napravo. Iz primerjave prejšnjih in sedanjih koordinat mobilne naprave lahko ugotovimo v katero stran se je nagib zgodil. Spodnja koda 5.14 prikazuje ukaze za pridobitev koordinat mobilne naprave, če se je nagib zgodil.

Koda 5.14: Nagibi mobilne naprave.

```
1  @Override
2  protected void update(GameTime gameTime) {
3      super.update(gameTime);
4
5      AccelerometerMovement movement = AccelerometerPanel.getState().getLastItem();
6      if (movement != null) {
7          float x = movement.getX();
8          float y = movement.getY();
9          float z = movement.getZ();
10     }
11 }
```

5.8.2 Razvoj

Podobno kot dotiki, so nagibi narejeni iz treh razredov:

- **AccelerometerPanel** je statičen razred, ki se uporablja za pridobitev objekta razreda *AccelerometerCollection*;
- **AccelerometerCollection** hrani vse nagibe v objektu razreda *AccelerometerMovement*;
- **AccelerometerMovement** hrani dejanski nagib, ki se je zgodil na napravi.

Za zajem nagiba je v Android aplikaciji potrebno implementira vmesnik *SensorEventListener*, ki ga dodamo objektu razreda *SensorManager* z metodo *registerListener*. Za pridobitev objekta razreda *Sensor* je potrebno preveriti,

če naprava podpira zajem nagibov (koda 5.15 prikazuje dodajanje in preverjanje vmesnika za nagibe). V vmesniku *SensorEventListener* je potrebno implementirati metodo *onSensorChanged*, ki nam vrne objekt razreda *SensorEvent*, ko se nagib zgodi. Iz objekta razreda *SensorEvent* ugotovimo koordinate nagiba in ga shranimo v *AccelerometerCollection*. Sedaj lahko pridemo do nagiba preko statičnega razreda *AccelerometerPanel*. Spodnja koda 5.16 prikazuje prestrezanje in shranjevanje nagiba.

Koda 5.15: Dodajanje in preverjanje vmesnika za nagibe.

```
1  sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
2  accelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
3
4  if (accelerometer != null) {
5      sensorManager.registerListener(
6          this,
7          accelerometer,
8          SensorManager.SENSOR_DELAY_NORMAL
9      );
10 }
```

Koda 5.16: Prestrezanje in shranjevanje nagiba.

```
1  @Override
2  public void onSensorChanged(SensorEvent event) {
3
4      AccelerometerMovement am = new AccelerometerMovement(
5          ac.count(),
6          AccelerometerMovementState.UNDEFINED,
7          event.values[0],
8          event.values[1],
9          event.values[2]
10     );
11 }
```

5.9 GraphicsDeviceManager

Razred *GraphicsDeviceManager* se uporablja za nastavitve in branje vrednosti mobilne naprave, kot so: širina, višina, celotni zaslon, zatemnitev mobilne naprave in podobno. V nadaljevanju bomo opisali njegovo uporabo in kako je razred sestavljen.

5.9.1 Uporaba

Objekt razreda *GraphicsDeviceManager* je potrebno najprej ustvariti s konstruktorjem, ki sprejme objekt razreda *Game*. Ko imamo objekt narejen, lahko pokličemo podane metode:

- **isFullScreen** nam pove, če je aplikacija v celozaslonskem načinu;
- **setFullScreen** omogoči ali onemogoči celozaslonski način;
- **isDimScreenOn** nam pove, če ima aplikacija onemogočeno zatemnjevanje zaslona;
- **setDimScreen** omogoči ali onemogoči zatemnjevanje zaslona;
- **setPreferredBackBufferWidth** nastavi širino zaslona na podano vrednost;
- **getPreferredBackBufferWidth** vrne nastavljeno širino zaslona;
- **setPreferredBackBufferHeight** nastavi višino zaslona na podano vrednost;
- **getPreferredBackBufferHeight** vrne nastavljeno višino zaslona.

Spodnja koda 5.17 prikazuje kako ustvarimo objekt razreda *GraphicsDeviceManager* in nastavimo širino zaslona na 480 pik in višino zaslona na 800 pik.

Koda 5.17: Nastavitev velikosti zaslona na 480x800.

```
1 @Override
2 protected void initialize() {
3     super.initialize();
4
5     GraphicsDeviceManager gdm = new GraphicsDeviceManager(this);
6     gdm.setPreferredBackBufferWidth(480);
7     gdm.setPreferredBackBufferHeight(800);
8 }
```

5.9.2 Razvoj

Za celozaslonski način ima razred *GraphicsDeviceManager* metodo *setFullScreen*. Metoda s pomočjo objekta razreda *Activity* in metodi *addFlags* in *clearFlags* nastavi aplikacijo v celozaslonski način. Spodnja koda 5.18 prikazuje implementacijo metode *setFullScreen*.

Koda 5.18: Implementacija metode *setFullScreen*.

```
1  public void setFullScreen(boolean fullScreen) {
2      this.fullScreen = fullScreen;
3
4      if (fullScreen) {
5          game.getWindow().addFlags(
6              WindowManager.LayoutParams.FLAG_FULLSCREEN
7          );
8          game.getWindow().clearFlags(
9              WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN
10         );
11     } else {
12         game.getWindow().addFlags(
13             WindowManager.LayoutParams.FLAG_FORCE_NOT_FULLSCREEN
14         );
15         game.getWindow().clearFlags(
16             WindowManager.LayoutParams.FLAG_FULLSCREEN
17         );
18     }
19 }
```

Za preprečitev zatemnjevanja zaslona metoda *setDimScreen* uporablja Android razred *PowerManager* [47] in metodo *newWakeLock*. Za dostop do razreda *PowerManager* je potrebno dodati dovoljenje WAKE_LOCK v Android AndroidManifest.xml datoteko (5.19 prikazuje primer datoteke AndroidManifest.xml) . Spodnja koda 5.20 prikazuje implementacijo metode *setDimScreen*.

Koda 5.19: Primer AndroidManifest.xml datoteke z WAKE_LOCK dovoljenjem.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="samotorcan.breakout"
4      android:versionCode="1"
5      android:versionName="1.0" >
```

```
6
7       ...
8       <uses-permission android:name="android.permission.WAKE_LOCK" />
9       ...
10  </manifest>
```

Koda 5.20: Implementacija metode setDimScreen.

```
1   public void setDimScreen(boolean dimScreen) {
2       if (this.dimScreen != dimScreen) {
3           this.dimScreen = dimScreen;
4
5           if (dimScreen) {
6               if (game.wl != null && game.wl.isHeld())
7                   game.wl.release();
8
9               game.wl = null;
10          } else {
11              PowerManager pm = (PowerManager)game.getSystemService(Context.POWER_SERVICE);
12              game.wl = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "DoNotDimScreen");
13          }
14      }
15  }
```

Poglavje 6

Razvoj igre

V tem poglavju bomo opisali razvoj igre Breakout. V prvem delu se nahaja opis knjižnice Box2D, ki je bila uporabljena za računanje fizike, v drugem delu pa sledijo opisi razredov in metod, ki so bili implementirani v naši igri.

6.1 Box2D

Knjižnico Box2D [48] smo uporabili za računanje fizike pri žogi, ploščku in opekah. Na začetku je potrebno ustvariti svet, kjer bomo dodajali naše elemente. To storimo s konstruktorjem razreda *World*, kjer moramo tudi podati moč gravitacije. Ko imamo svet narejen je potrebno narediti vsak objekt posebej in ga dodati v svet. Paziti je potrebno tudi, da opeko odstranimo iz sveta, ko jo zadane žoga. To storimo z ukazom *destroyBody*, kje podamo naš objekt, ki ga želimo odstraniti iz sveta. Objekt razreda *World* je potrebno tudi simulirati vsak obhod metode *update*. To storimo z metodo *step*, kjer podamo pretečen čas, ki ga naj svet simulira. V kodi 6.1 se nahaja ustvarjanje sveta in opeke, simuliranje, ter odstranitev opeke iz sveta.

Koda 6.1: Box2D primer.

```
1 // ustvari svet brez gravitacije
2 World world = new World(Vector2.Zero, false);
3
4 // ustvari nepremicno opeko
```

```
5  BodyDef bodyDef = new BodyDef();
6  bodyDef.type = BodyType.StaticBody;
7  bodyDef.position.set(Box2d.pixelToWorld(50), Box2d.pixelToWorld(50));
8
9  PolygonShape shape = new PolygonShape();
10 Box2d.setAsBox(
11     shape,
12     Box2d.pixelToWorld(50),
13     Box2d.pixelToWorld(25)
14 );
15
16 FixtureDef fixtureDef = new FixtureDef();
17 fixtureDef.shape = shape;
18
19 Body body = world.createBody(bodyDef);
20 body.createFixture(fixtureDef);
21
22 shape.dispose();
23
24 // simuliraj fiziko za pretecen cas 15 milisekund
25 world.step(0.015, 8, 3);
26
27 // odstrani opeko
28 world.destroyBody(body);
```

Pri knjižnici Box2D je potrebno tudi paziti na velikost objektov. Knjižnica ima omejeno maksimalno hitrost objekta na 2 enoti za vsako simulacijo. Zato smo se odločili za pomanjšanje objektov iz koordinat zaslona v koordinate sveta ter s tem dosegli večjo maksimalno hitrost naših objektov. Implementirali smo nove metode za pretvarjanje koordinate sveta v koordinate zaslona in obratno. V kodi 6.1, vrstici 12 in 13, lahko vidimo uporabo metode *pixelToWorld*, ki pretvori koordinate zaslona v koordinate sveta.

Knjižnica Box2D pozna dva glavna tipa objektov:

- **nepremični objekti**, ki jih ni možno premakniti s simulacijo fizike;
- **običajni objekti**, ki se pravilno obnašajo v svetu fizike.

Primer nepremičnih objektov so stranice naše igre, ki delujejo kot zidovi za odbijanje žoge. Običajni objekt pa je žoga, ki se odbija od zidov in opek. Iz

kode 6.1, vrstica 6, je razvidno, da je opeka nepremični objekt in zadetek z žogo jo ne premakne, ker na njo ne deluje fizika.

6.2 Entity

Namen razreda *Entity* je predstavitev vseh fizičnih objektov, ki se lahko pojavijo v naši igri. Razred vsebuje koordinate objekta, glede na zaslon naprave, ter enolični ključ, ki identificira objekt. Razred *Entity* je abstraktni razred, kar pomeni, da ga ni mogoče ustvariti, kot objekt, vendar ga dedujejo drugi razredi. Razred, ki podeduje razred *Entity* mora implementirati naslednje metode:

- **update** vsebuje kodo za nadzor objekta;
- **draw** vsebuje kodo za izris objekta;
- **added** metoda se pokliče takrat, ko je objekt dodan v svet;
- **removed** metoda se pokliče takrat, ko je objekt odstranjen iz sveta.

Najbolj pomembne metode v razredu *Entity* so:

- **setX** nastavi x koordinato objekta;
- **setY** nastavi y koordinato objekta;
- **getX** vrne x koordinato objekta;
- **getY** vrne y koordinato objekta;
- **setId** nastavi enolični ključ objekta;
- **getId** vrne enolični ključ objekta;
- **setUpdate** pove ali se naj izvede metoda *update*;
- **isUpdate** pove ali se bo izvedla metoda *update*;

- **setDraw** pove ali se naj izriše objekt z metodo *draw*.
- **isDraw** pove ali se bo izvedla metoda *draw*.

Koda 6.2 prikazuje implementacijo testnega razreda *TestObject*, ki podeduje razred *Entity*. Razred *TestObject* je sedaj pripravljen za uporabo v drugih razredih, kot je razred *Stage* za hranjenje objektov razreda *Entity*.

Koda 6.2: Podedovanje razreda *Entity*.

```
1  public class TestObject extends Entity {
2      @Override
3      public void update(GameTime time) {
4          // koda za nadzor objekta
5      }
6
7      @Override
8      public void draw(GameTime time,
9              SpriteBatch spriteBatch,
10             PrimitiveBatch primitiveBatch) {
11         // koda za izris objekta
12     }
13
14     @Override
15     public void added(Stage stage, GameTime gameTime) {
16         // se izvede, ko je objekt dodan v stage
17     }
18
19     @Override
20     public void removed(Stage stage, GameTime gameTime) {
21         // se izvede, ko je objekt odstranjen iz stage
22     }
23 }
```

6.3 WorldEntity

Razred *WorldEntity* deduje razred *Entity* in doda nove metode za delo s svetom. Glavna razlika med razredom *WorldEntity* in *Entity* je pripravljenost razreda *WorldEntity* za delo z Box2D in razredom *World*. Pomembne metode v razredu *WorldEntity* so:

- **setX** dodano nastavi tudi x koordinato objekta v svetu;

- **setY** dodano nastavi tudi y koordinato objekta v svetu;
- **update** nastavi koordinate objekta, če so bile spremenjene s simulacijo sveta;
- **createBody** je metoda, kje je potrebno implementirati sestavo objekta;
- **added** metoda pokliče metodo *createBody*, ki zgradi objekt za svet;
- **removed** metoda odstrani objekt iz sveta;
- **addContactListener** doda poslušalca za trke med objekti;
- **removeContactListener** odstrani podanega poslušalca.

Koda 6.3 prikazuje implementacijo testnega razreda *TestWorldObject*, ki podeduje razred *WorldEntity*. Razred *TestWorldObject* je sedaj pripravljen za uporabo v svetu fizike s pomočjo knjižnice Box2D.

Koda 6.3: Podedovanje razreda *WorldEntity*.

```
1  public class TestWorldObject extends WorldEntity {
2      public TestWorldObject(MainGame game) {
3          super(game);
4      }
5
6      @Override
7      protected void createBody() {
8          // koda za ustvarjanje objekta sveta
9      }
10
11     @Override
12     public void update(GameTime time) {
13         // koda za nadzor objekta
14     }
15
16     @Override
17     public void draw(GameTime time,
18             SpriteBatch spriteBatch,
19             PrimitiveBatch primitiveBatch) {
20         // koda za izris objekta
21     }
22
23     @Override
```

```
24      public void added(Stage stage, GameTime gameTime) {
25          // dodatna koda, ki se izvede, ko je objekt dodan v stage
26      }
27
28      @Override
29      public void removed(Stage stage, GameTime gameTime) {
30          // dodatna koda, ki se izvede, ko je  objekt odstranjen iz stage
31      }
32  }
```

6.4 Stage

Razred *Stage* se uporablja za shranjevanje objektov razreda *Entity* in vseh drugih objektov razreda, ki dedujejo *Entity*. Razred *Stage* ima dve pomembni metodi:

- **update**, ki pokliče metodo *update* vsakega objekta, ki ga hrani;
- **draw**, ki izriše vse objekte s pomočjo *draw* metode vsakega objekta.

Razred *Stage* ima tudi odloženo dodajanje in odstranjevanje objektov. To zagotovi, da se dodajanje in odstranjevanje objektov zgodi pred klicem metod *update*, ki bi povzročile problem dodajanja ali odstranjevanja elementov med izvajanjem zanke (koda 6.4).

Koda 6.4: Metoda za odlozeno dodajanje elementov v *Stage*.

```
1  public void addItem(Entity item, boolean immediately) {
2      if (immediately) {
3          items.add(item);
4          item.added(this, updateGameTime);
5          callAddListeners(item);
6      } else {
7          requests.add(new StageRequest(item, StageRequest.Request.ADD));
8          callAddListeners(item);
9      }
10 }
```

Na objekt razreda *Stage* lahko dodamo tudi poslušalec, ki nas bo obvestil, ko se objekt doda ali odstrani. To storimo s pomočjo metod *addListener* in *removeListener*. Implementacija poslušalca je zelo preprosta, kjer je potrebno

implementirati metodi *itemAdded* in *itemRemoved*. Koda 6.5 prikazuje implementacijo poslušalca, ki igralcu doda 10 točk, ko se odstrani določena opeka iz objekta razreda *Stage*.

Koda 6.5: Poslušalec za dodajanje točk.

```
1  private class BrickRemovedListener implements StageListener {
2      @Override
3      public void itemAdded(Entity item) { }
4
5      @Override
6      public void itemRemoved(Entity item) {
7          if ((selectedShape == EnumShape.RECTANGLE && item instanceof RectangleBrick) ||
8              (selectedShape == EnumShape.SQUARE && item instanceof SquareBrick) ||
9              (selectedShape == EnumShape.CIRCLE && item instanceof CircleBrick) ||
10             (selectedShape == EnumShape.TRIANGLE && item instanceof TriangleBrick)) {
11             shapeHit = true;
12             mainGame.addScore(10);
13         }
14     }
15 }
```

Poleg metodi *update* in *draw* se v razredu *Stage* nahajajo še metode:

- **getItems** vrne vse objekte, ki jih vsebuje *Stage*;
- **addItem** doda objekt v *Stage*;
- **addItems** doda poljubno število objekt v *Stage*;
- **getItem** nam vrne objekt po podanem enoličnem ključem objekta;
- **removeItem** odstrani podani objekt iz *Stage*;
- **removeItems** odstrani poljubno število podanih objektov iz *Stage*;
- **addListener** doda poslušalca za odstranjevanje in dodajanje objektov v *Stage*;
- **removeListener** odstrani podanega poslušalca.

6.5 Sprite

Razred *Sprite* je zelo preprosti razred. Glavni namen razreda je hranjenje dela slike, ki jo je možno izrisati s pomočjo metode *draw*. Sedaj lahko naredimo poljubno število različnih objektov razreda Sprite, ki vsebujejo del slike. Tako imamo lahko samo eno veliko sliko, kjer je sestavljeno veliko manjših slik. Slika 6.1 prikazuje sestavljeno sliko, ki se v igri uporablja za izris gumbov.

Slika 6.1: Primer sestavljene slike.

Pomembne metode v razredu *Sprite* so:

- **setTextureRegion** nastavi novi del slike za izbrani objekt;
- **setId** nastavi enolični ključ za izbrani objekt;

- **getId** vrne enolični ključ objekta;
- **getWidth** vrne širino izbranega dela slike;
- **getHeight** vrne višino izbranega dela slike;
- **draw** izriše del slike.

6.6 Map

Razred *Map* se uporablja za nalaganje sobe iz XML zapisa. Za branje XML datoteke uporabljamo Android razred *SAXParser*. Razred bere XML datoteko vsak znak posebej in kliče določene metode. Sestavljen je iz treh glavnih metod, ki jih je potrebno implementirati.

StartElement

Metoda *startElement* se pokliče takrat, ko naletimo na začetno značko v XML datoteki. V tej metodi smo implementirali sledenje vseh odprtih značk in ustvarjanje opek, ko pridemo do pravilne značke. Vsa imena značk se nahajajo v enumeraciji zaradi lažjega spreminjanja imena značke. V kodi 6.6 se nahaja del implementacije metode *startElement*, kjer naredimo novo opeko, ko pridemo do značke z enumeracijo *RECTANGLE_BRICK*, ki vsebuje ime *rectangleBrick*.

Koda 6.6: Del implementacije metode *startElement*.

```
1  @Override
2  public void startElement(String uri, String localName,
3      String qName, Attributes attributes) throws SAXException {
4
5      if (qName.equalsIgnoreCase(XmlElements.RECTANGLE_BRICK.toString())
6          && currentElements.size() == 2
7          && currentElements.get(0) == XmlElements.MAP
8          && currentElements.get(1) == XmlElements.BRICKS) {
9
10         currentElements.add(XmlElements.RECTANGLE_BRICK);
11         foundElement = XmlElements.RECTANGLE_BRICK;
12
```

```
13          currentBrick = new RectangleBrick(game);
14          bricks.add(currentBrick);
15      }
16  }
```

Characters

Metoda *characters* se pokliče takrat, ko v XML datoteki naletimo na tekstovne znake. Vse kar smo implementirali v tej metodi je shranjevanje tekstovnih znakov v objekt razreda *String* za uporabo v metodi *endElement*.

EndElement

Podobno kot metoda *startElement* se metoda *endElement* pokliče takrat, ko naletimo na konec značke v XML datoteki. V tem delu smo implementirali vse dele, kjer je potrebno vedeti tekstovno vrednost, ki se nahaja med začetkom in koncem določene značke. Koda 6.7 prikazuje del implementacije metode *endElement*, kjer preberemo koordinate opeke.

Koda 6.7: Del implementacije metode *endElement*.

```
1  @Override
2  public void endElement(String uri, String localName,
3      String qName) throws SAXException {
4
5      if (foundElement == XmlElements.X) {
6          currentBrick.setX(MathHelper.parseFloatWithDefault(elementValue, 0));
7      } else if (foundElement == XmlElements.Y) {
8          currentBrick.setY(MathHelper.parseFloatWithDefault(elementValue, 0));
9      }
10 }
```

V kodi 6.7 je vidna tudi uporaba razreda *MathHelper*, ki nam pomaga pri pretvarjanju niza znakov v določena števila. Metoda *parseFloatWithDefault* pretvori niz znakov v spremenljivko tipa float. Metoda ima tudi parameter za privzeto vrednost, če niza znakov ni mogoče pretvoriti v spremenljivko tipa float.

V XML opisu sobe se nahajajo posamezne opeke, kot tudi nastavitve sobe. Primer teh so začetna življenja in začetna hitrost žoge. Koda 6.8 predstavlja del XML datoteke, ki se uporablja za predstavitev sobe. Na sliki so tudi vidne koordinate opeke, ki se preberejo in nastavijo v metodi *endElement*.

Koda 6.8: XML za predstavitev sobe.

```
<map>
    <lives>3</lives>
    <defaultBallSpeed>250</defaultBallSpeed>
    <bricks>
        <rectangleBrick>
            <position>
                <x>88</x>
                <y>32</y>
            </position>
            <color>red</color>
        </rectangleBrick>
        <rectangleBrick>
            <position>
                <x>168</x>
                <y>32</y>
            </position>
            <color>red</color>
        </rectangleBrick>
        ...
    </bricks>
</map>
```

Pomembne metode, ki se nahajajo v razredu Map so:

- **setDefaultValues** nastavi objekt razreda *Map* na privzeto vrednost;
- **reloadMap** ponovno prebere XML datoteko;
- **getBricks** vrne vse opeke, ki so opisane v XML datoteki;
- **getEntities** vrne dodatne objekte, ki so opisani v XML datoteki;
- **getNumberOfLives** vrne število življenj, ki je zapisano v XML datoteki;
- **getDefaultBallSpeed** vrne začetno hitrost žoge, ki je zapisano v XML datoteki;

- **getMapResourceId** vrne enolični ključ datoteke, ki je bila naložena;
- **getMapId** vrne enolični ključ objekta razreda *Map*.

6.7 Settings

Razred *Settings* je statičen razred, ki se uporablja za branje in shranjevanje nastavitev. Dve glavni metodi razreda *Settings* sta *addSetting* in *getSetting*. Metoda *addSetting* prebere določeno vrednost, metoda *addSetting* pa shrani določeno vrednost v nastavitve. Da shranimo nastavitve v datoteko je potrebno poklicati metodo *save*, ki uporabi razred *ObjectOutputStream* za shranjevanje. Za branje nastavitev iz datoteke uporabimo metodo *load*, ki s pomočjo razreda *ObjectInputStream* prebere nastavitve iz datoteke.

Koda 6.9 prikazuje implementacijo metod *load* in *save*, kjer je tudi vidna uporaba Android metod *openFileInput* in *openFileOutput* za branje in pisanje datotek.

Koda 6.9: Implementacija metod *load* in *save*.

```
1  public void load(Game game) {
2      try {
3          ObjectInputStream ois =
4              new ObjectInputStream(game.openFileInput(FILENAME));
5
6          settings = (HashMap<String, String>)ois.readObject();
7          ois.close();
8      } catch (FileNotFoundException e) {
9          createDefaultFile(game);
10     } catch (Exception e) {
11         Log.v(LogCat.ERROR, e.toString(), e);
12     }
13 }
14
15 public void save(Game game) {
16     try {
17         ObjectOutputStream oos =
18         new ObjectOutputStream(game.openFileOutput(FILENAME, Context.MODE_PRIVATE));
19
20         oos.writeObject(settings);
```

```
21          oos.close();
22      } catch (Exception e) {
23          Log.v(LogCat.ERROR, e.toString(), e);
24      }
25  }
```

Razred *Settings* omogoča tudi shranjevanje privzetih nastavitev, ki se uporabijo ob prvem zagonu igre. Za branje privzetih nastavitev uporabimo Android razred *SAXParser*, podobno kot pri branju sobe razreda *Map*. Koda 6.10 prikazuje privzeto XML datoteko za nastavitve.

Koda 6.10: Privzete nastavitve.

```
1  <defaultSettings>
2      <add key="muteSound" value="false" />
3      <add key="useAccelerometer" value="true" />
4      <add key="showFPS" value="false" />
5  </defaultSettings>
```

6.8 Score

Razred *Score* je zelo podoben razredu *Settings*, vendar namesto shranjevanje nastavitev, razred *Score* shranjuje točke, ki jih igralec doseže v igri. Metode, ki se nahajajo v razredu *Score*, so:

- **addScore** doda podane točke za določeno sobo;
- **getMapScores** vrne vse točke za podano sobo;
- **load** prebere vse točke iz datoteke;
- **save** shrani točke v datoteko.

Spodnja koda 6.11 prikazuje primer uporabe, kjer preberemo točke iz datoteke v razred *Score*, preberemo točke za drugo sobo, shranimo točke za prvo sobo in zapišemo točke v datoteko.

Koda 6.11: Delo z razredom Score.

```
1  Score.getInstance().load(this);
2
```

```
3 ArrayList<Integer> map2Scores = Score.getInstance().getMapScores("map2");
4 Score.getInstance().addScore("map1", 125);
5
6 Score.getInstance().save(this);
```

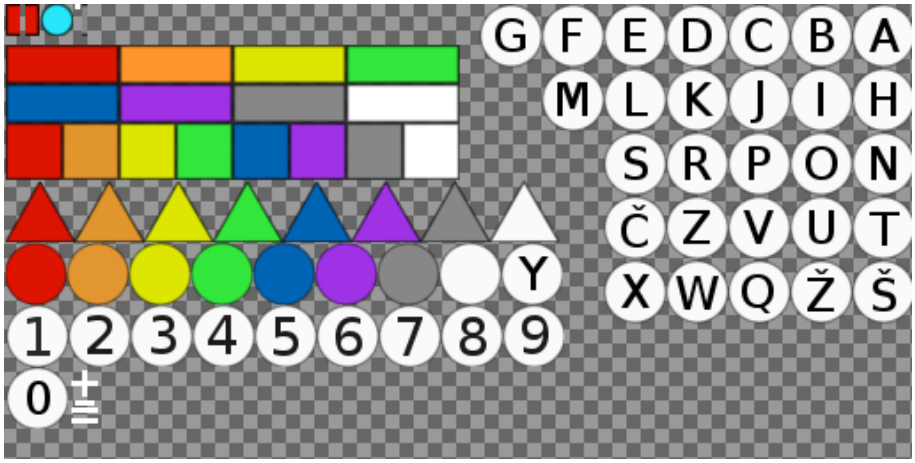
6.9 Atlas

Razred *Atlas* se uporablja za nalaganje vseh objektov razreda *Sprite*. Za uporabo razreda *Atlas* moramo narediti tudi XML datoteko, kjer so opisane koordinate objektov razreda *Sprite*. Koda 6.12 prikazuje primer XML datoteke, ki jo uporabi razred *Atlas* za nalaganje objektov razreda *Sprite*. Za branje XML datoteke se uporablja Android razred *SAXParser*, podobno kot pri razredu *Map*. XML datoteka mora vsebovati tudi ime slike (primer slike je viden na sliki 6.2), ki se uporabi za izdelovanje objektov razreda *Sprite*.

Koda 6.12: Primer XML datoteke za ustvarjanje objektov razreda *Sprite*.

```
1  <atlas>
2      <atlasImage>R.drawable.maingame</atlasImage>
3      <sprites>
4          <sprite>
5              <id>leftPad</id>
6              <x>0</x>
7              <y>0</y>
8              <width>8</width>
9              <height>18</height>
10         </sprite>
11         <sprite>
12             <id>middlePad</id>
13             <x>44</x>
14             <y>0</y>
15             <width>1</width>
16             <height>18</height>
17         </sprite>
18         <sprite>
19             <id>rightPad</id>
20             <x>11</x>
21             <y>0</y>
22             <width>8</width>
23             <height>18</height>
24         </sprite>
25         <sprite>
```

```
26              <id>ball</id>
27              <x>20</x>
28              <y>0</y>
29              <width>18</width>
30              <height>18</height>
31          </sprite>
32      </sprites>
33  </atlas>
```



Slika 6.2: Primer sestavljene slike za uporabo v razredu *Atlas*.

Spodnja koda 6.13 prikazuje primer uporabe razreda *Atlas*, kjer naredimo objekt razreda *Sprite* in zahtevamo *Sprite* z enoličnim ključem "ball". Dobljeni *Sprite* tu izrišemo v metodi *draw* razreda *Game* s pomočjo metode *draw* razreda *Sprite*.

Koda 6.13: Primer uporabe razreda *Atlas*.

```
1  private Atlas atlas;
2
3  @Override
4  protected void loadContent() {
5      super.loadContent();
6
7      atlas = new Atlas(this, R.raw.spriteLocations);
8  }
9
10 @Override
11 protected void draw(GameTime gameTime) {
```

```
12      super.draw(gameTime);
13
14      SpriteBatch spriteBatch = new SpriteBatch(getGraphicsDevice());
15      spriteBatch.begin();
16          atlas.getSprite("ball").draw(
17              gameTime,
18              spriteBatch,
19              50,
20              50,
21              Vector2.One(),
22              Color.white(),
23              0
24          );
25      spriteBatch.end();
26  }
```

6.10 Opeke

V igro smo implementirali 6 vrst opek. Prve štiri opeke so zelo preproste, kjer vsaka predstavlja enega izmed likov. Liki, ki jih lahko srečamo v igri so: pravokotnik, kvadrat, krog in trikotnik. Te opeke so lahko tudi različnih barv. Peta opeka predstavlja opeko kroga, kjer krog vsebuje eno od števil med 0 in 9. Šesta opeka tudi predstavlja opeko kroga, vendar namesto števila vsebuje črko. Slika 6.3 prikazuje vsako izmed šestih možnih opek.

Slika 6.3: Vseh 6 opek, ki jih lahko vidimo v igri.

Vsak razred šestih opek podeduje abstraktni razred Brick. Metode, ki so implementirane v abstraktnem razredu Brick so:

- **getScore** nam vrne število točk, ki jih igralec dobi, ko zadane opeko;
- **setScore** nastavi število točk, ki jih igralec dobi, ko zadane opeko;
- **addDestroySound** doda zvok, ki se predvaja, ko igralec zadane opeko;

- **removeDestroySound** odstrani podani zvok, ki se predvaja, ko igralec zadane opeko;
- **setColor** nastavi barvo opeke;
- **getColor** vrne nastavljeno barvo opeke;
- **removed** dodatno predvaja nastavljene zvoke.

Vsaka opeka mora tudi implementirati metodo *createBody*, ki vsebuje objekt sveta za uporabo pri računanju fizike. Koda 6.14 prikazuje implementacijo metode *createBody* za opeko, ki prikazuje krog.

Koda 6.14: Implementacija metode *createBody* za opeko krog.

```
1  @Override
2  protected void createBody() {
3      BodyDef bodyDef = new BodyDef();
4      bodyDef.type = BodyType.StaticBody;
5      bodyDef.position.set(Box2d.pixelToWorld(getX()), Box2d.pixelToWorld(getY()));
6
7      CircleShape shape = new CircleShape();
8      shape.setRadius(Box2d.pixelToWorld(radius));
9
10     FixtureDef fixtureDef = new FixtureDef();
11     fixtureDef.shape = shape;
12
13     Body body = world.createBody(bodyDef);
14     body.createFixture(fixtureDef);
15
16     setBody(body);
17     shape.dispose();
18 }
```

Poglavje 7

Zaključek

Skozi diplomsko nalogo smo spoznali, da nam je implementacija igre vzela več časa kot pričakovani čas enega meseca. Glavni del povečanje časa za implementacijo igre nam je vzela implementacija 2D knjižnice. Dobra stran implementacije 2D knjižnice je hitrejše izdelovanje naslednjih projektov v operacijske sistemu Android. Dodatni čas nam je tudi vzelo testiranje na mobilnih napravah z različnimi velikosti zaslona. Iz pridobljenih testov smo implementirali avtomatsko razširjanje/zmanjševanje slike na mobilnih napravah različne velikosti zaslona. Zaradi povečanega časa smo se bili primorani odreči tudi nekaterih funkcionalnosti, na primer večje število sob, daljše sobe, shranjevanje igre in podobno.

Razvito igro lahko uporabljamo na kateri koli mobilni napravi, ki ima nameščen operacijski sistem Android z različico 1.6 ali več. Igro lahko uporabljamo kjerkoli in kadarkoli, saj za njeno uporabo ni nikakršnih omejitev (npr. internetna povezava). Igra je tudi preprosta za uporabo, kot tudi uporabniški vmesnik, ki je intuitiven in enostaven. Igra Breakout je tudi zelo popularna, kot tudi preprosta za razumeti zato igralec hitro osvoji pomen igralnih elementov. Igra je tudi zelo hitra in uporablja zelo malo pomnilnika, kar je pomembno za mobilne naprave, ki imajo omejeno zmogljivost proti sodobnemu računalniku.

Igra vsebuje tudi poučne elemente, kot je prepoznavanje števil, oblik, barv in črk. Spodbuja tudi učenje matematike, kjer je potrebno seštevati in odštevati, da lahko uničimo pravilne opeke in tako dobimo dodatne točke. Pri prepoznavanju črk je potrebno tudi sestaviti besede, ki se pojavijo na zaslonu za dodatne točke. Za lažje igranje igra ponuja nadzor z nagibi mobilne naprave, kot tudi klasični način z potegi prstov na zaslonu.

Razvita 2D knjižnica za operacijski sistem Android je bila implementirana v celoti, kot je bilo zamišljeno na začetku diplome. Vsebuje veliko elementov, ki jih lahko uporabimo tudi v drugih aplikacijah, ki želijo hitri izris 2D objektov s pomočjo OpenGL. Knjižnica je razdeljena v elemente tako, da lahko uporabimo samo tiste elemente, ki jih aplikacija potrebuje. Knjižnica je tudi preprosta za uporabo, ker se zgleduje po ostalih knjižnicah, kot sta XNI in XNA.

7.1 Izboljšave

Prva možna velika izboljšava je implementacija več jezikov. Pod to izboljšavo spada zvok, kot tudi napisi v igri. Potrebno bi bilo narediti vse zvoke v drugem jeziku, kot tudi vse slike, ki vsebujejo napise. Pri tem bi se velikost igre veliko povečala, kar lahko povzroči določene probleme, kot so prevelik začetni nalagalni čas, velika poraba pomnilnika in podobno. Pri tem bi bilo tudi pametno premisliti za implementacijo sprotnega nalaganja zvokov in slik za zmanjšanje začetnega nalagalnega časa. Veliko časa bi tudi vzelo testiranje odzivnosti in hitrosti na počasnejših mobilnih napravah in kako to vpliva pri igranju igre.

Druga veliko izboljšava je shranjevanje igre za poznejše igranje. Potrebno bi bilo implementirati shranjevanje trenutnega stanja igre, nalaganje stanja v igro in implementacija grafičnega vmesnika za delo s shranjenimi igrami.

Potrebno bi bilo tudi premisliti koliko je maksimalno število stanj igre, ki jih lahko shranimo, koliko prostora zasedajo naša shranjena stanja igre v mobilni napravi, kdaj jih izbrišemo zaradi prevelikega števila in podobno.

Pod tretjo izboljšavo spadajo preproste izboljšave, kot so večje število sob in opek, daljše sobe, implementacija moči in podobno. Te izboljšave so dokaj preproste in vse, kar bi vzelo za implementacijo je čas.

7.2 Odzivi

Igro smo tudi razdelili med ljudi in jih povprašali o mnenjih. Odzivi na igro so bili pozitivni. Med njimi so omenili odzivnost igre, uporaba nagiba mobilne naprave za usmerjanje ploščka, lep izgled igre, hitro razumevanje igre in poučnost pri igranju igre. Še posebej so pohvalili preprostost igre za igranje in velika podobnost originalni igri Breakout. Izražena je bila tudi želja po dodatnih in različnih sobah v prihodnjih različicah igre, kot tudi prevod igre v slovenski jezik. Ena izmed želj je bila tudi predstavitev igre na Google Play za lažje nalaganje igre na mobilno napravo, kot tudi podajanje igre med prijatelji.

Literatura

[1] (2012) Wikipedia, "Zgodovina igre Breakout". Dostopno na:
http://en.wikipedia.org/wiki/Breakout_(video_game)#History_and_development

[2] (2012) Wikipedia, "Igra Breakout". Dostopno na:
http://en.wikipedia.org/wiki/Breakout_(video_game)#Gameplay

[3] (2012) Wikipedia, "Igra Super Breakout". Dostopno na:
http://en.wikipedia.org/wiki/Breakout_(video_game)#Super_Breakout

[4] (2012) Wikipedia, "Igra Breakout 2000". Dostopno na:
http://en.wikipedia.org/wiki/Breakout_(video_game)#Breakout_2000

[5] (2012) playandroid, "Igra Best Breakout". Dostopno na:
http://www.playandroid.com/blog/android-game-review-best-breakout

[6] (2012) Wikipedia, "Programski jezik Java". Dostopno na:
http://en.wikipedia.org/wiki/Java_(programming_language)

[7] (2012) Wikipedia, "OpenGL". Dostopno na:
http://en.wikipedia.org/wiki/OpenGL

[8] (2012) Wikipedia, "Eclipse". Dostopno na:
http://en.wikipedia.org/wiki/Eclipse_(software)

[9] (2012) Wikipedia, "Android SDK". Dostopno na:
http://en.wikipedia.org/wiki/Android_software_development

[10] (2012) Wikipedia, "GIMP". Dostopno na:
http://en.wikipedia.org/wiki/GIMP

[11] (2012) Wikipedia, "Android operacijski sistem". Dostopno na:
http://en.wikipedia.org/wiki/Android_(operating_system)

[12] (2012) Wikipedia, "Google Play". Dostopno na:
http://en.wikipedia.org/wiki/Google_Play

[13] (2012) Wikipedia, "XML". Dostopno na:
http://en.wikipedia.org/wiki/XML

[14] (2012) Retronator, "XNI". Dostopno na:
http://xni.retronator.com/

[15] (2012) Wikipedia, "XNA". Dostopno na:
http://en.wikipedia.org/wiki/Microsoft_XNA

[16] (2012) game development framework, "libGDX". Dostopno na:
http://libgdx.badlogicgames.com

[17] (2012) Wikipedia, "Microsoft Windows". Dostopno na:
http://en.wikipedia.org/wiki/Microsoft_Windows

[18] (2012) Wikipedia, "Linux". Dostopno na:
http://en.wikipedia.org/wiki/Linux

[19] (2012) Wikipedia, "Mac OS". Dostopno na:
http://en.wikipedia.org/wiki/Mac_OS

[20] (2012) Wikipedia, "HTML5". Dostopno na:
http://en.wikipedia.org/wiki/HTML5

[21] (2012) Wikipedia, "Windows Phone". Dostopno na:
http://en.wikipedia.org/wiki/Windows_Phone

[22] (2012) Wikipedia, "Visual Basic". Dostopno na:
`http://en.wikipedia.org/wiki/Visual_Basic`

[23] (2012) Wikipedia, ".NET okvir". Dostopno na:
`http://en.wikipedia.org/wiki/.NET_Framework`

[24] (2012) Wikipedia, "Xbox 360". Dostopno na:
`http://en.wikipedia.org/wiki/Xbox_360`

[25] (2012) Wikipedia, "Microsoft Visual Studio". Dostopno na:
`http://en.wikipedia.org/wiki/Microsoft_Visual_Studio`

[26] (2012) Wikipedia, "Microsoft Visual Studio Express". Dostopno na:
`http://en.wikipedia.org/wiki/Microsoft_Visual_Studio_Express`

[27] (2012) Microsoft, "XNA Creators Club". Dostopno na:
`http://create.msdn.com`

[28] (2012) Wikipedia, "LINQ". Dostopno na:
`http://en.wikipedia.org/wiki/Language_Integrated_Query`

[29] (2012) Wikipedia, "Objective-C". Dostopno na:
`http://en.wikipedia.org/wiki/Objective-C`

[30] (2012) Retronator, "Monkey Labour". Dostopno na:
`http://dawnofplay.com/monkeylabour`

[31] (2012) Wikipedia, "iPhone". Dostopno na:
`http://en.wikipedia.org/wiki/IPhone`

[32] (2012) Wikipedia, "iPad". Dostopno na:
`http://en.wikipedia.org/wiki/IPad`

[33] (2012) Wikipedia, "iOS". Dostopno na:
`http://en.wikipedia.org/wiki/IOS`

[34] (2012) Wikipedia, "C#". Dostopno na:
http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[35] (2012) Wikipedia, "DirectX". Dostopno na:
http://en.wikipedia.org/wiki/DirectX

[36] (2012) Fakulteta za računalništvo in informatiko, "XNI". Dostopno na:
http://gameteam.fri.uni-lj.si

[37] (2012) Android developers, "XPath". Dostopno na:
http://developer.android.com/reference/javax/xml/xpath/
package-summary.html

[38] (2012) OpenGL, "glDrawElements". Dostopno na:
http://www.khronos.org/opengles/sdk/docs/man/xhtml/
glDrawElements.xml

[39] (2012) Android developers, "Activity". Dostopno na:
http://developer.android.com/reference/android/app/
Activity.html

[40] (2012) Android developers, "decodeResource". Dostopno na:
http://developer.android.com/reference/android/graphics/
BitmapFactory.html#decodeResource(android.content.res.
Resources,%20int,%20android.graphics.BitmapFactory.Options)

[41] (2012) Android developers, "texImage". Dostopno na:
http://developer.android.com/reference/android/opengl/
GLUtils.html#texImage2D(int,%20int,%20android.graphics.
Bitmap,%20int)

[42] (2012) OpenGL, "glGenTextures". Dostopno na:
http://www.khronos.org/opengles/documentation/opengles1_0/
html/glGenTextures.html

[43] (2012) OpenGL, "glBindTexture". Dostopno na:
`http://www.khronos.org/opengles/sdk/docs/man/xhtml/glBindTexture.xml`

[44] (2012) OpenGL, "glDeleteTextures". Dostopno na:
`http://www.khronos.org/opengles/sdk/1.1/docs/man/glDeleteTextures.xml`

[45] (2012) Android developers, "SoundPool". Dostopno na:
`http://developer.android.com/reference/android/media/SoundPool.html`

[46] (2012) Android developers, "MediaPlayer". Dostopno na:
`http://developer.android.com/reference/android/media/MediaPlayer.html`

[47] (2012) Android developers, "PowerManager". Dostopno na:
`http://developer.android.com/reference/android/os/PowerManager.html`

[48] (2012) Box2D, "Box2D priročnik". Dostopno na:
`http://box2d.org/manual.pdf`

Slike

Koda