

УНИВЕРЗИТЕТ У БЕОГРАДУ

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

ДИПЛОМСКИ РАД

ИЗ ПРОГРАМИРАЊА МОБИЛНИХ УРЕЂАЈА

Android апликација - Игра
балансирања лоптице

Аутор

Ђорђе Живановић, 0033/2013

Ментор

Др Саша Стојановић

Београд, 4. јул 2017.

Садржај

1 Увод	2
1.1 Структура	3
1.2 Постојећа решења	3
2 Преглед решења и предлог игре	4
3 Функционалности	6
3.1 Почетни екран	6
3.1.1 Основни	6
3.1.2 Мени	7
3.1.3 Опције полигона	7
3.2 Игра	8
3.2.1 Игра побеђена	9
3.2.2 Игра изгубљена	10
3.3 Прављење полигона	10
3.3.1 Опције за уређивање полигона	10
3.3.2 Чување полигона	11
3.4 Резултати	12
3.5 Подешавања	13
4 Android	14
4.1 GUI код	14
4.2 Java код	15
4.3 Чување података	16
4.4 Unit Test	17
5 Модел физике	18
6 Графика	21
6.1 Општа идеја	21
6.2 Оптимизације код играња игре	21
7 Архитектура	23
7.1 Организација пакета	23
7.2 Пакет game	24
7.2.1 Пакет acceleration.filter	24

7.2.2	Пакет coefficient	25
7.2.3	Пакет collision	25
7.2.4	Пакет utility	25
7.3	Пакет statistics.database	26
7.3.1	Пакет table	27
7.4	Пакет shape	27
7.4.1	Пакет constants	27
7.4.2	Пакет coordinate	28
7.4.3	Пакет draw	28
7.4.4	Пакет factory	29
7.4.5	Пакет figure	30
7.4.6	Пакет movement.collision.detection	30
7.4.7	Пакет movement.collision.handling	31
7.4.8	Пакет parser	31
7.4.9	Пакет scale	32
7.4.10	Пакет sound	32
8	Закључак	33
	Литература и линкови	34

Глава 1

Увод

Основни циљ пројекта био је истраживање Android API¹-а који се користи за прављење апликација за Android уређаје. Да би се у потпуности разумела његова употреба и организација кода у апликацијама намењеним Android-у, морало се приступити изазову прављења игрице попут ове. Поред Android API-а, због одабира игре у којој је потребна симулација лопта, постојали су проблеми моделирања физике и рачунарске графике.

Први у низу проблема који је решаван био је делимично позавање API-а. Први корак ка његовом превазилажењу био је предмет програмирање мобилних уређаја ([2]), на којем су научене основне ствари које он нуди. Други корак је био читање документације на AndroidDeveloper сајта ([1]) на ком се осим описа пакета, класа, интерфејса, метода налазе савети и упутства за коришћење Android API-а. Последњи корак на који се прибегавало кад проблем није могао бити решен је StackOverflow ([3]). Наведена секвенца у процесу решавања омогућила је превазилажење препрека и давала мотивацију за даљи рад у борби са непознатим.

Пошто је постојала жеља за што реалнију симулацију лопте, и њене физика приликом њеног динамичког, а и статичког кретања, појавила су се два нова проблема. Први од њих био је рачунарска графика, чија мотивација потиче из жеља аутора да наликује графици у познатим играма попут *GTA* (линк [4]). Први корак ка томе било је моделирање нечега што се може назвати модел физике. Он ће омогућити кориснику видно савршену симулацију физичких односа између објеката у игри. Он је уједно и био други проблем. Међутим дати модел је требало поткрепити осећајем да не постоји освежавање екрана. Нити да корисник има могућност да разликује лопту у стварности од оне у игрици. Стога је графика оптимизована у виду паралелизације. А физика моделирана по угледу на физику у стварном свету.

Последњи изазов, ако не и најважнији била је организација кода у великому пројекту као што је овај која омогућава брузу проширивост и лагано додавање нових функционалности, мењање постојећих модела судара... Овај проблем, иако на први поглед наиван, како се повећавала количина кода био је све већи. Стога се код у одређеним фазама рефакторисао и разбијао на модуле. Крајњи код је последица жеље аутора да се добије модуларан и проширив код.

¹Application programming interface

1.1 Структура

У глави 4 биће објашњено који делови Android API-а су коришћени, као и њихове мане и врлине, и зашто су баш они одабрани. У глави 5 биће причано о моделу који судара који је коришћен и зашто су биране његове компоненте. У глави 6 биће описан систем за исцртавање екрана. У глави 7 биће описана софтверска архитектура (пројектни обрасци, организација кода). У глави 3 биће наведени неки случајеви коришћења система и како се систем понаша у одређеним тренуцима. У глави 8 биће наведени закључци, да ли систем може да се побоља, да ли могу да се додају нове функционалности и сл.

1.2 Постојећа решења

Битна особина коју програмер мора да поседује је способност да чита туђи код и да прилагођава својој имплементацији. Књига кномогла да направим модел судара какав је у игрици је [5]. Она ми је омогућила да добијем увид у организацију кода и она је само била увод у моје решење. Поред тога модификовани су разни модели и тестирали за потребе. За доста реалнији модел постоји пуно више радова на интернету, али за почетну идеју користио сам упрошћену имплементацију једног модела из књиге [5], која се налази у линку: [7].

Глава 2

Преглед решења и предлог игре

У наредних неколико табела налази се преглед могућности већ постојећих игара сличних рађеној у дипломском раду (нису све исте по логици, али све имају лопту која се креће), као и жеља за функционалности коју би лопта у дипломском раду подржала.

Игра	Нови полигон	Преглед свих полигона	Рангирање	Звук судара
Rolling Sky ¹	Не	Да	Да	Да
Physics Drop ²	Не	Да	Не	Не
Bouncing Ball ³	Не	Да	Не	Не
Bounce Classic ⁴	Не	Да	Да	Не
Rapid Roll ⁵	Не	Не	Не	Не
Balance Ball ⁶	Не	Не	Да	Не
Crazy Balancing Ball ⁷	Не	Да	Да	Не
Balance Ball Game	Да	Да	Да	Да

Табела 2.1: Постојећа решења.

Игра	Трење	Гравитација рупе	Убрзање уређаја	Параметри	Звук игре
Rolling Sky	Не	Не	Не	Не	Да
Physics Drop	Да	Не	Не	Не	Да
Bouncing Ball	Не	Не	Не	Не	Да
Bounce Classic	Не	Не	Не	Не	Да
Rapid Roll	Не	Не	Не	Не	Не
Balance Ball	Не	Не	Да	Не	Не
Crazy Balancing Ball	Да	Не	Да	Не	Да
Balance Ball Game	Да	Да	Да	Да	Не

Табела 2.2: Постојећа решења.

Игра	Круж. препр.	Прав. преп.	Рот. прав. препр.	Вртл. препр.	Одбиј.
Rolling Sky	Не	Не	Да	Не	Не
Physics Drop	Да	Да	Да	Не	Да
Bouncing Ball	Не	Да	Не	Не	Не
Bounce Classic	Не	Да	Да	Не	Не
Rapid Roll	Не	Да	Не	Не	Не
Balance Ball	Да	Да	Не	Не	Не
Crazy Balancing Ball	Да	Да	Да	Не	Не
Balance Ball Game	Да	Да	Да	Да	Да

Табела 2.3: Постојећа решења.

Као што се може да види из табела 2.1, 2.2, 2.3 оно што би одвајало ову игру од постојећих игара било би постојање трења између лопте и подлоге, као и могућност модификовања полигона (уређивања по својој воли). Поред тога подржавања свих добрих карактеристика које оне имају, редом излистаних , као и додатне могућности физике судара.

Биће коришћен програмски језик Java искључиво јер омогућава лакшу израду пројекта и лакше дебаговање. У комбинацији са Java биће коришћен xml за израду GUI⁸. Биће коришћен Android Studio⁹ који ће давати лагодност у погледу дебаговања, build-овања и праћења промена. Он омогућава интегрисани рад са системима за праћење ревизија попут Git (линк [6]). У наставку се налази списак свих алата који ће бити коришћени:

Алат	Сврха
AndroidStudio 2.3.3	IDE
compileSdkVersion 25.0.1	систем за build-овање уграђен у AndroidStudio
SourceTree 2.1.2.5.	Систем за ревизију
TeXMaker 4.5	L <small>A</small> T <small>E</small> X уређивач

Табела 2.4: Алти коришћени при развоју пројекта.

Машина на којој ће бити писан и компајлиран код и на којој ће радити Android Studio је HP Omen са 12GB RAM, 512GB SSD, i7-6700HQ 2,7GHz, оперативни систем Windows 10.0.15063. Машина на којој ће бити покретана апликација је NVIDIA Shield Tablet K1, који је у тренутку првог инсталирања имао Android 5.0 верзију инсталiranу.¹⁰. Минимални Андроид ОС¹¹ који подржава апликацију је 5.0.

⁸Graphical User Interface

⁹<https://developer.android.com/studio/index.html>

¹⁰У тренутку писања последњих измена ажуриран је на Android 7.0

¹¹Оперативни систем

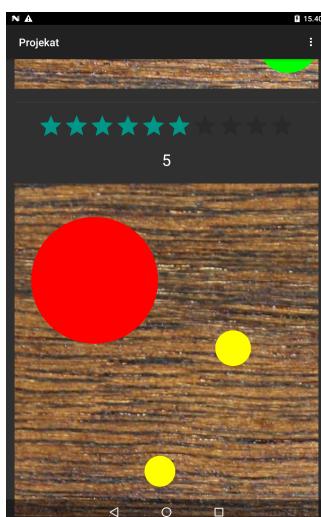
Глава 3

Функционалности

Ма колико изгледало наивно и просто направити дизајн који ће привући корисника, док не кренете да правите апликацију, не схватите колико немате појма шта просечан корисник хоће. Стога након пажљивог разматрања направљено је пет "екрана" из којих корисник приступа одговарајућим функционалностима које ће бити изложене у даљем тексту

3.1 Почетни экран

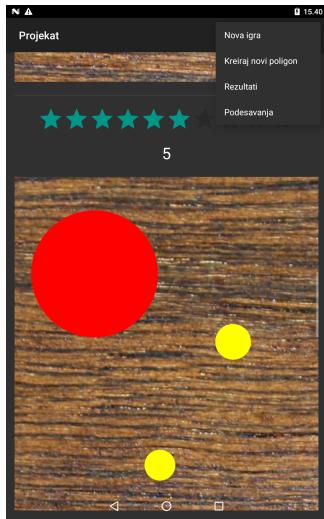
3.1.1 Основни



Слика 3.1: Почетни экран

Кад корисник покрене апликацију појављује му се листа полигона које може да игра, са изгледом полигона (слика 3.1). Такође изнад изгледа полигона налазе се његово име и тежина. Ово омогућава да кориснику почетнику одабере ниво прикладан за њега, или експерту да се окуша са нечим тежим. Изглед полигона је скалиран да одговара оном који ће бити у игрици. Кликом на било који од полигона у листи покреће му се игра за тај полигон.

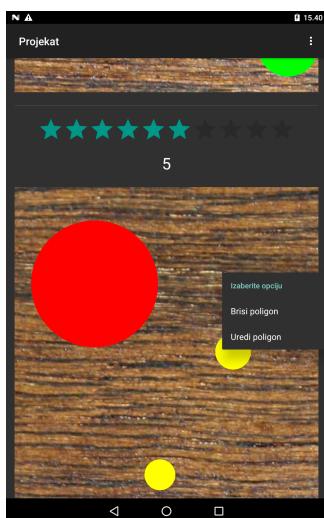
3.1.2 Мени



Слика 3.2: Мени

Корисник све време на врху прозора има мени који отвара ако прстом кликне на три тачке. Из менија који се појави (слика 3.2) корисник може да одабере једну од 4 опције. Прва опција започиње нову игру, али у режиму од више нивоа који су насумично генерисани. Друга опција отвара нови екран у ком корисник прави свој полигон. Трећа опција отвара нови екран резултати, где корисник може да види све резултате претходних игри, као и других играча. Четврта опција отвара подешавања за игру, која корисник може да мења.

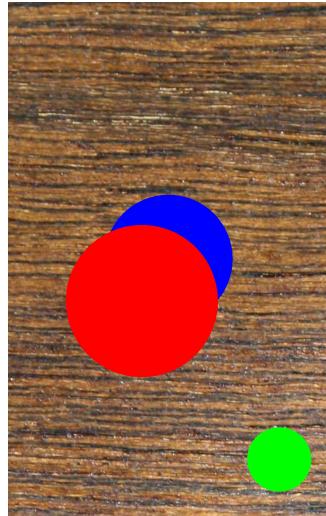
3.1.3 Опције полигона



Слика 3.3: Падајући мени везан за полигон

Корисник задржавањем прста на полигон добија падајући мени (слика 3.3) са две опције. Прва опција брише полигон из целе игре. А друга омогућава његово уређивање, тако што отвара нови прозор са датим полигоном.

3.2 Игра



Слика 3.4: Тренутак у игри

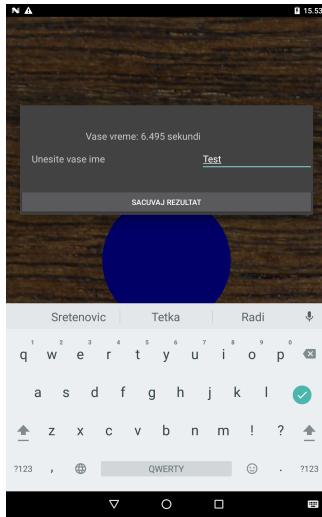
Постоје два мода играња. Први мод је обичан у ком је циљ да црвену лопту корисник убаци у зелену рупу за што краће време (слика 3.4). Што му је време краће, биће боље рангиран на листи за тај полигон. Други мод је авантуристчки који омогућава кориснику да игра десет насумичних нивоа заредом који су поређани по растућој тежини и биће рангиран на посебној листи за тај мод. Корисник има неколико типова препрека на које може да наиђе (слика 3.8). Типови препрека:

1. Зид (ивица екрана и жути правоугаоник) - лопта се одбија од зида под истим углом којим улази. Губитак енергије током судара зависи од коефицијента подешеног у подешавањима.
2. Стуб (жути круг) - исто као зид, са тим да је кружног облика.
3. Амбис (црни круг) - лопта кад упадне у њега играч губи игру.
4. Вртлог (плави круг) - покушава да увуче лопту у њега, и потом у зависности од брзине баца из њега или га врти.
5. Крај (зелени круг) - кад лопта упадне у њега корисник је победио дати полигон.¹

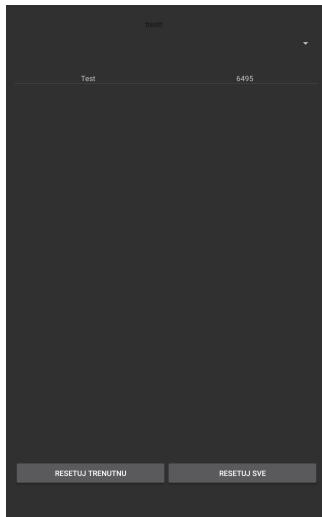
¹Амбис, вртлог и крај кад лопта их дотакне вуку је ка себи одређеном силом

Лопта све време добија брзину у зависности од силе која делује на Android уређаја. На лопту у сваком тренутку делује трење од стране подлоге које покушава да је врати у стање мировања. У сваком тренутку корисник може да изађе из одговарајућег полигона, али неће му бити сачувано ништа што је играо.

3.2.1 Игра побеђена



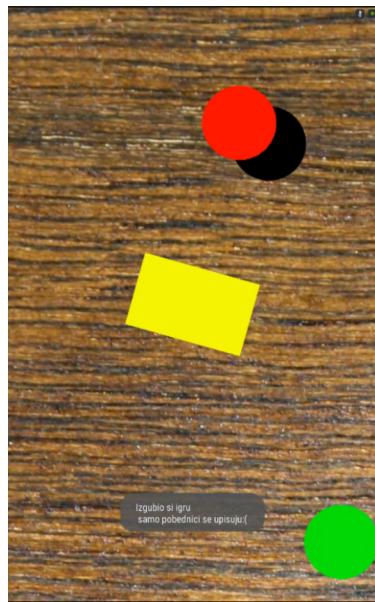
Слика 3.5: Дијалог кад победи



Слика 3.6: Листа резултата за полигон који је победио

Кад корисник убаци лопту у зелену рупу, искаче му дијалог (слика 3.5). На дијалогу пише време, као и нуди кориснику да унесе своје име. Кад кликне на дугме *SACUVAJ REZULTAT* корисников резултат се чува у листи резултата за тај плигон и отвара листа резултата за исти (слика 3.6).

3.2.2 Игра изгубљена



Слика 3.7: Обавештење кад корисник изгуби

Када корисник упадне у амбис изађе му обавештење да је изгубио игру (слика 3.7).

3.3 Прављење полигона

Постоје два мода у која корисник може да уђе кад уређује полигон.

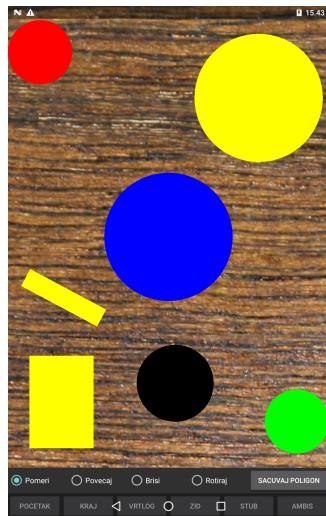
Један је мод уређивања, у ком се налази тако што или је кликнуо *Uredi poligon* на почетном екрану или је сачувао полигон под неким именом. У овом моду кад корисник кликне назад аутоматски ће се чувати поруке и бити избацивана упозорења ако нешто није како треба (фали крајња позиција, почетна...).

Други мод је обични мод и то је док корисник није сачувао полигон. У овом моду кад се кликне дугме назад, ништа неће бити сачувано.

3.3.1 Опције за уређивање полигона

Корисник има опције генерисања препрека за лопту наведеним у секцији 3.2 кликтањем одговарајућег дугмета са именом. Такође може и да генерише почетну позицију лопте. Поред тога са свим фигурама корисник може да ради једну од четири опције:

1. *Pomeri* - Помера одговарајућу фигуру на полигону тако да корисник мора да задржава прст на фигури док је помера, Кад пусти, ту ће фигура бити померена.
2. *Povečaj*- Повећава одговарајућу фигуру, тако да ако корисник одабере фигуру и помера прст, фигура ће се повећавати у односу на то где је његов прст (за

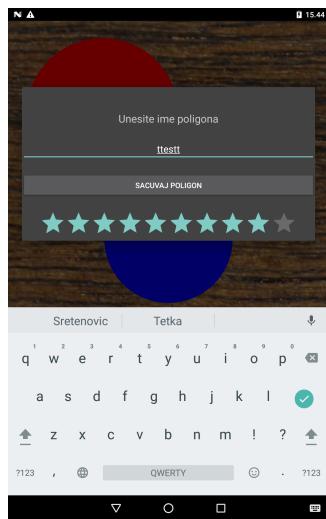


Слика 3.8: Екран који корисник види кад уређује полигон

круг ће крајња позиција прста означавати позицију тачке са кружнице, за правоугаоник позицију одговарајућег темена).

3. *Brisi* - Брише одабрану фигуру са полигона.
4. *Rotiraj* - Ротира правоугаоник према позицији прста тако да одабрана права која је одређена центром правоугаоника и прстом ротира око центра пратећи прст, са том правом ротира и цео правоугаоник.

3.3.2 Чување полигона

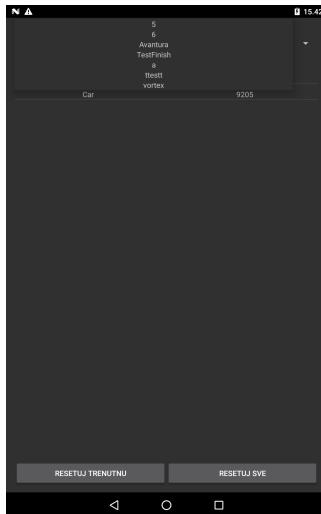


Слика 3.9: Дијалог који се појави кад корисник хоће да сачува полигон

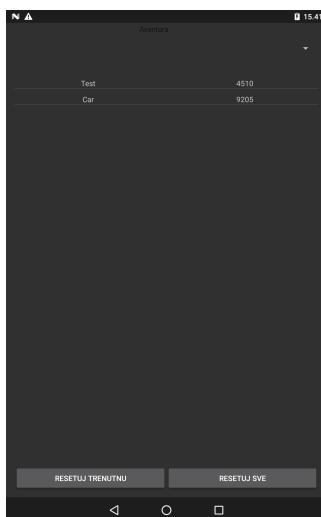
На крају корисник кад заврши генерисање правоугаоника има дугме *SACUVAJ POLIGON*. Тад се кориснику појављује дијалог на коме уноси тежину полигона као

и име тог полигона. Када кликне *SACUVAJ POLIGON* на новом дијалогу, сачуваће полигон. Ако је постојао стари пребрисаће га.

3.4 Резултати



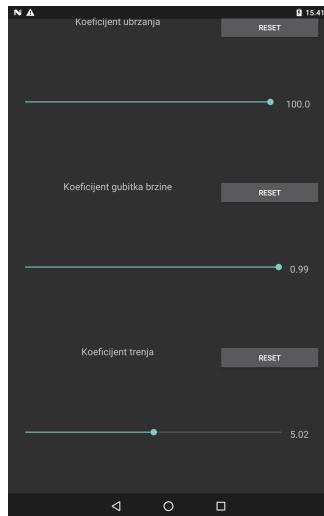
Слика 3.10: Избор полигона за који хоће да се виде резултати



Слика 3.11: Листа резултата за одговарајући полигон

Корисник има опцију да види статистику за све полигоне које је играо, и који постоје . Такође има могућност да види и за авантура мод (под именом *Avantura*). Корисник кликом на спинер бира одговарајући ниво за који хоће да прикаже резултате (слика 3.10) и потом, му се приказују резултати сортирани од најбољег ка најлошијем (3.11) у погледу времена. Корисник има могућност ресетовања листе резултата за један изабрани полигон притискањем дугмета *RESETUJ TRENU* или ресетовањем резултата свих листи са притискањем дугмета *RESETUJ SVE*.

3.5 Подешавања



Слика 3.12: Подешавања за игру

Корисник има опцију да подеси одговарајуће коефицијенте које се користе током симулације игре (слика 3.12). То се постиже померањем одговарајуће тачке на дужи. Редом су наведени коефицијенти:

1. Коефицијент убрзања - Колико брзо ће лоптица да убрзава, што већи коефицијент брже убрзава.
2. Коефицијент губитка брзине - Колико енергије остаје у лоптици приликом судара, што већи коефицијент, више енергије остаје.
3. Коефицијент трења - Колико лоптица успорава приликом кретања по подози, што већи коефицијент више успорава.

Корисник има могућност да врати подешавања свих коефицијената на подразумевана притиском на дугме *RESET* из одговарајућег реда за коефицијенте.

Глава 4

Android

Још од избацувања iPhone-а као првог комерцијалног "паметног" телефона који ќе објединити у једном уређају све функционалности које има PC (линк [8]), ако не и више, било је јасно у ком смеру се запутило IT¹ тржиште. Телефони или ти компјутери у уепу су у том тренутку постали будућност, а наша садашњост. Како ће се развијати IT технологија, није познато, али зна се да ће огроман удео имати телефони. Одатле и потиче моја жеља да овладам вештином Android програмирања, оперативним системом који је преузео примат на тржиштут паметних телефона. (линк [9])

4.1 GUI код

Сав GUI који је коришћен, писан је у xml-у који подржва одговарајуће Android Studio IDE. Сва имена GUI компоненти давана су тако да прво иде тип компоненте и затим се надовезује одговарајући опис који је карактеристичан за употребу дате компоненте. Типа *seekBarCoefficientAcceleration* означава *seekBar* GUI компоненту, а *CoefficientAcceleration* каже да се користи да представи коефицијент убрзаша. Имена сва су писана CamelCase-ом². При избору компоненти биране су тако да што пријатније изгледа кориснику и да што лагодније буде за рад (на основу узорка од неколико корисника који су пробали различите верзије GUI-а).

Где је било неопходно да позиционирање компоненти буде независно од типа од екрана коришћен је *LinearLayout*³, док је за неке ствари где је битна само позиција компоненти, коришћен *RelativeLayout*⁴.

Коришћена је Android Dark Material Theme као основна тема.

¹Internet Technology

²<http://wiki.c2.com/?CamelCase>

³<https://developer.android.com/reference/android/widget/LinearLayout.html>

⁴<https://developer.android.com/guide/topics/ui/layout/relative.html>

4.2 Java код

GUI компоненте на одговарајућим екранима референциране су тако што се из одговарајућег прозора нађе компонента уз помоћ `findViewById`⁵

Постоји пет активности (*MainActivity*, *CreatePolygonActivity*, *GameActivity*, *SettingsActivity*, *StatisticsActivity*). Свака подржавајући одговарајућу функционалност из главе 3. Дакле *MainActivity* почетни је экран (3.1), *GameActivity* саму игру (3.2), *SettingsActivity* подешавања (3.5), *StatisticsActivity* резултате (3.4) и *CreatePolygonActivity* уређивање полигона (3.3).

Свака активност је прављена тако да је изведена из активности *CommonActivity*. Свака активност која наслеђује ову класу има подешен прозор тако да је навигациона трака скривена док корисник не превуче прстом са дна уређаја на горе. Поред тога оријентација је увек вертикално, да корисник не би губио време ако случајно окрене уређај. Такође је остато доступан кориснику да би могао у сваком тренутку да сазна више о обавештењу које му стигне (али тек након што превуче прстом је экран са врха ка дну). Дата имплементације је по узору на већину данашњих ексклузивних играчких назлова за Android уређаје попут Hill Climb Racing (линк за преузимање [10]). Даље коришћен је MVC⁶ пројектни узорак прилагођен за Android. При чему имамо активност која прослеђује своје догађаје контролеру, и он у зависности од њих обавља акције и у моделу се то чува. Постоји и имплементација где у моделу постоје методе које обрађују податке, али изабраним је раздвојена имплементација кода, од приступа подацима, и олакшана читкоћа кода. Тамо где није био велики обим потребних метода (*MainActivity*, *StatisticsActivity* и *SettingsActivity*) обједињени су Controller и View.

Постојање класе *CommonModel* за циљ има омогућавање заједничког модела свим активностима које су за потребу имали рендериовање направљеног/који се прави по-лигона.

Тамо где је разумно било да се појављују дијалози (као за чување резултата по успешној игри, или за чување направљеног полигона) прављене су класе (*SaveDialog* и *GameOverDialog*) које проширују класу *Dialog*⁷ и праве одговарајући потребан GUI. Ово је омогућило финије подешавање дијалога и њиховог изгледа од унапред направљених класа попут *AlertDialog*⁸.

Тамо где су била потребна исцртавања на једном екрану (*CreatePolygonActivity* и *GameActivity*) коришћене су одговарајуће класе које су проширивале *SurfaceView*⁹ (биће касније објашњено како ово ради у глави 6).

За *SettingsActivity* било је потребно унапредити постојећу GUI компоненту *SeekBar*. Због тога је додата класа *SeekBarUpgrade* која додаје још неки низ особина и омогућава лакше додавање више *SeekBar*-ова у *SettingsActivity*.

Пошто је било неопходно да лопта реагује на силу која делује на Android

⁵<https://developer.android.com/reference/android/view/View.html> и <https://developer.android.com/reference/android/app/Activity.html>

⁶Model View Controller

⁷<https://developer.android.com/reference/android/app/Dialog.html>

⁸<https://developer.android.com/reference/android/app/AlertDialog.html>

⁹<https://developer.android.com/reference/android/view/SurfaceView.html>

уређај у *GameActivity* активности, коришћен је уграђени Android сензор *TYPE_ACCELEROMETER*¹⁰. Он сваких 20ms прослеђује активности детектоване вредности убрзања уређаја по *x*, *y* и *z* оси. Њихово коришћење је описано даље у глави 5. Ова вредност од 20ms емпириским утврђивањем се показала као довољна да корисник стекне осећај реалистичности кретања куглице и реаговања исте на силе које делују на уређај.

Да би корисник стекао што реалистичнији осећај кретања лоптице и тренутка њеног судара са препрекама или уласка у њих неопходно је било подржати звук. Изабрана је класа *SoundPool*¹¹ која омогућава пуштање звукова. Имплементиран је омотач за њу у виду класе *SoundPlayer* који додаје још неке функционалности. Звучни ефекти су одабрани емпириски да што краће трају и да корисник не обрађа превелику пажњу на њих. Детаљније у методама, интерфејсу и свим класама за звукове у глави 7.

4.3 Чување података

Постојале су потребе за три начина чувања података у апликацији.

Први потребан начин је било перзистирање коефицијената потребних за симулацију кретања лопте. Пошто су они били потребни да перзистирају дуж активности *GameActivity* и *SettingsActivity* и није их било пуно, одбачене су опције да се чувају у бази и посебном фајлу. Стога је одабрана опција *SharedPreferences*¹² која чува податке у облику паре key/value. Ово омогућава згодно додавање нове константе без гломазних мењања база и без мењања шеме фајла. Овај систем је имплементиран у класи *Coefficient*.

Други потребан начин било је перзистирање података везаних за име нивоа, тежину, као и рангирање нивоа. То је омогућено коришћењем SQLite језика који је ништа друго него лакша варијанта SQL¹³ језика за рад са базом. Прво је имплементиран *SQLiteOpenHelper* у виду класе *GameDatabaseHelper* који омогућава мењање шеме базе (ако се врши додавање нове функционалности у игрици везане за ниво), као и приступ истој. Даље је *GameDatabaseHelper* омотан у класи *GameDatabase* која имплементира читав низ функционалности који је био неопходан у апликацији.

Последњи потребан начин било је перзистирање нивоа (њиховог изгледа). Одабран је начин чувања у фајловима. То омогућава згодно додавање нових фигура само додавањем новог типа фигуре, и не изискује гломазно мењање базе података и API-а функционалности базе. Пошто је било неопходно да се омогући перзистирање димензија фигура на полигону за сваки тип уређаја, то је урађено скалирањем фигура у односу на величину екрана. Скалирање фигура се обавља уз помоћ класа *UtilScale* и *UtilScaleNormal* (апстрактна и имплементација). Класа која чува полигоне у облику фајлова у овом формату је *ShapeParser*.

¹⁰https://developer.android.com/guide/topics/sensors/sensors_overview.html

¹¹<https://developer.android.com/reference/android/media/SoundPool.html>

¹²<https://developer.android.com/reference/android/content/SharedPreferences.html>

¹³Standard Query Language

4.4 Unit Test

За потребе тестирања коришћени су корисници који су играли ову игрицу, као и *JUnit* пакет који је омогућио тестирање одређених метода класа и функционалности. Да би ова апликација ишла у продукцију, неопходна су бројна унапређења у тестирању као и броју тестова.

Глава 5

Модел физике

У овом поглављу биће изложен рад са класом *CollisionModel* и како она ради. Иако на први поглед изгледа просто направити модел судара, кад кренете да правите појављује се све више проблем. Наивно решење једног, створи још бар један проблем.

Прво се са сензора детектује ново убрзање које делује на уређај (гравитационо + убрзање уређаја) и оно се филтрира да би се спречило превелике осцилације у убрзању, ако сензор нешто погрешно прочита. Даље филтрирано убрзање као и време кад је детектовано се прослеђују моделу са листом фигура и лоптом (позиција, полу-пречник). Ово је опис система до "предаје" управљања лопте класи *CollisionModel*.

Даље се убрзање скалира (множи са одговарајућим коефицијентом из подешавања апликације) што омогућава да лопта пуно брже реагује на промене убрзања уређаја. Даље, лопта у сваком тренутку има вектор кретања брзине по све три осе v_x, v_y, v_z . Да би се израчунале нове брзине v_x, v_y потребно је "потпуно" убрзање (сила) која делује на лопту. "Потпуно" убрзање се рачуна тако што се скалирано убрзање потом дода на трење (боге речено одузме). Трење се рачуна као убрзање по z оси скалирано са коефицијентом трења добијеним из подешавања апликације, али тако да има знак супротан од брзине. Овај модел трења је реалистичан јер трење и у стварности је μN , где је N нормална сила (на уређају је убрзање у суштини та нормална сила). Потом се нова брзина рачуна

$$v_i = v_{previousi} + a_{full}\Delta T, i \in \{x, y\}$$

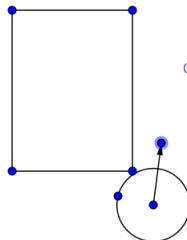
При чему је a_{full} "потпуно" убрзање, $v_{previousi}$ претходна брзина лопте, а v_i нова и ΔT разлика између времена претходне детекције сензора и тренутне детекције (који се обрађује). Ако је брзина променила смер по некој од оса и трење је утицало на то, брзина по тој оси постаје 0.

Сад се рачунају потенцијалне нове позиције лопте по формули:

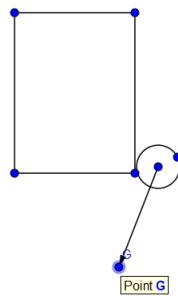
$$i = i_{old} + v_i \Delta T, i \in x, y$$

Где је i нова координата центра лопте по одговарајућој оси, а i_{old} стара координата. Потом се рачунају судари са свим фигурама на екрану И за оне фигуре са којима се

сударила рачуна се како утичу на промену брзине. Све те фигуре утичу на промену брзине. Постоје два типа фигура.



Слика 5.1: Судар лопте и правоугаоника кад се лопта креће нагоре



Слика 5.2: Судар лопте и правоугаоника кад се лопта креће надоле

Први су класичне препреке од које лопта одбија. Могу бити правоугаоне (квадрасте) или кружне (лоптасте). За правоугаонке је коришћен упрошћен модел, где су странице правоугаоника паралелне осама. Промена брзине рачуна тако да под оним углом којим је ушла мора да се и одбије (за случај кад удара само страницу правоугаоника, без ћошка). То је одрађено простом провером гледа се у коју страницу удари и онда ако је ударила у страницу чија је оса паралелна рецимо оси x онда се мења брзина y (враћа се негативна вредност по тој оси). Аналогно и за осу y . Док кад удари ћошак врши се промена брзине по оси као у случајевима да није ћошак, али само под условом да се "приближава" тој ивици. Кад се каже приближава, мисли се у случају кад рецимо лопта прилази са доњој страници правоугаоника са доње стране, онда ће се "одбити" по y оси (слика 5.1). Али рецимо у случају кад би прилазила са горње стране не би се одбила по y оси (слика 5.2). Ово спречава да се лопта "заглави" на ћошку, а и даје релативно реалистичан судар. Кад се ради са правоугаоницама којима странице нису паралелне осама, проблем се сведе на претходни. Извршимо просту ротацију читавог координатног система (уједно и брзине лопте) за угао $-\alpha$ при чему је α угао под којим се налази правоугаоник у односу на x осу. Ротација се врши множењем сваке координате коју смо користили у старом систему ротационом матрицом ¹. Тако да је нови проблем сведен на већ

¹<http://mathworld.wolfram.com/RotationMatrix.html>

решен. Проблем се реши као у претходном случају и онда се промена брзине само врати у почетни систем ротацијом промена брзине за α . Пошто ротација захтева рачунање \sin и \cos коришћена је оптимизације по којој су они израчунати унапред (кад се учитава полигон) и онда нема губитка времна при њиховом евалуирању.

За кружне препреке тражена брзина је :

$$v_{new} = v_i - 2 \frac{\langle v_{old}, c_1 - c_2 \rangle (c_1 - c_2)}{\|c_1 - c_2\|^2}$$

При чему су c_1 стара координата центра лопте (пре потенцијалног померања), c_2 координата центра препреке са којом се лопта судара, v_{old} брзина лопте као и v_{new} брзина која треба да се добије. При чему је $\|nesto\|$ ознака за дужину вектора, док је $\langle n_1, n_2 \rangle$ ознака за скаларни производ два вектора $nesto_1, nesto_2$. Промена брзине се рачуна:

$$v_{change} = \frac{v_{new} - v_{old}}{2}$$

Овај начин рачунања избегава било какво коришћење тригонометрије и због тога је брз.

Друга врста препрека су рупе. Има их три типа, али у суштини се све три понашају на исти начин. Промена брзине се рачуна по формулама:

$$v_{change} = \frac{(c_2 - c_1) \cdot GRAVITY_CON}{\|c_2 - c_1\|^2}$$

Овде су c_2 нова координата лопте, c_1 координата препреке. Док је $GRAVITY_CON$ константа којом се лопта привлачи.

Након што се израчунају промене брзине за све сударе у том тренутку лопте, оне се множе са 2 и све се сабирају. Након тога ако нема промене брзине по x координати или је лопти промењена брзина под утицајем рупа лопта може да се помери на новоизрачунату координату по тој оси. Иначе по x оси не може. Аналогно је и за y осу. На крају брзина се мења додавањем збира промена брзине по тој оси на брзину генерисану након рачунања "потпуног" убрзања. Оод условом да је брзина промене различита од 0 или под условом да се лопта "сударила" са рупом претходно израчуната брзина губи одређен проценат који је подешен у подешавањима апликације.

Глава 6

Графика

Иако рачунарска графика на први поглед изгледа једноставна област, што се више задубљујеш у њу, откриваш каква озбиљна наука стоји.

6.1 Општа идеја

Док су за активности у којима није потребно учстало исцртавања екрана коришћена GUI нит, тамо где је потребно (*CreatePolygonActivity*, *GameActivity*) морало је бити пронађено другачије решење. За површ уместо стандардног *Canvas-а* који припада *ImageView*, који захтева исцртавање целог екрана¹ користи се *SurfaceView* који се само освежи (остатак екрана се не мења) кад је неопходно. И то исцртавање се ради у посебној нити, која кад заврши посао, само замени *Canvas* од *SurfaceView* са новим *Canvas-ом*. Што умањује заузетост GUI нити непотребним исцртавањем, и омогућава да се користи у рачунању и ажурирању *SurfaceView*. Ради смањења загревања уређаја, нова нит која исцртава *Canvas* то ради само кад је затражено од ње (кад се десила промена позиције), остатак времена спава.

Цео модел је оптимизован тако да се тежило максималној паралелизацији и минимализацији броја lock-ова. У обе активности се на почетку иницијализације *SurfaceView* праве класе *ShapeFactory* и *ShapeDraw*, од којих прва служи за парсирање полигона из фајлова (и њихово скалирање), док друга служи за цртање фигура по *Canvas-у* *SurfaceView*. Да би фигура била исцртана помоћу класе *ShapeDraw* неопходно је да подржава *ShapeDrawInterface*, тј. да може да се кликне на њу, ротира, промени величина, помери, израчуна угао нагиба. Такође при иницијализацији *SurfaceView* прави се и посебна нит која ће да ради исцртавање. При уништавању *SurfaceView* нит се уништава.

6.2 Оптимизације код играња игре

Код играња игре, нема потребе за непотребно рендеровање и исцртавање других фигура по *Canvas-у* осим на почетку. Стога се направи спрајт целог полигона без

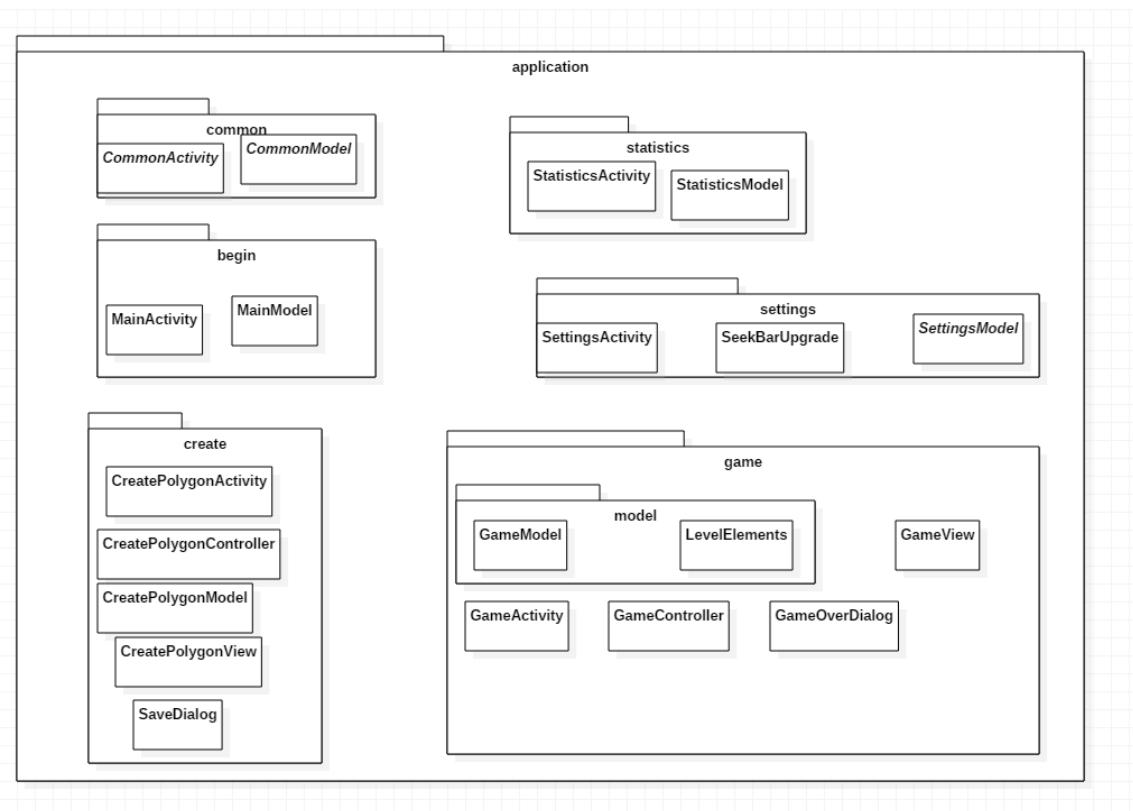
¹<https://developer.android.com/reference/android/widget/ImageView.html>

лопте, и лопта се лепи касније на спрајт како се мења њена позиција. Ово омогућава убрзано ажурирање екрана.

Глава 7

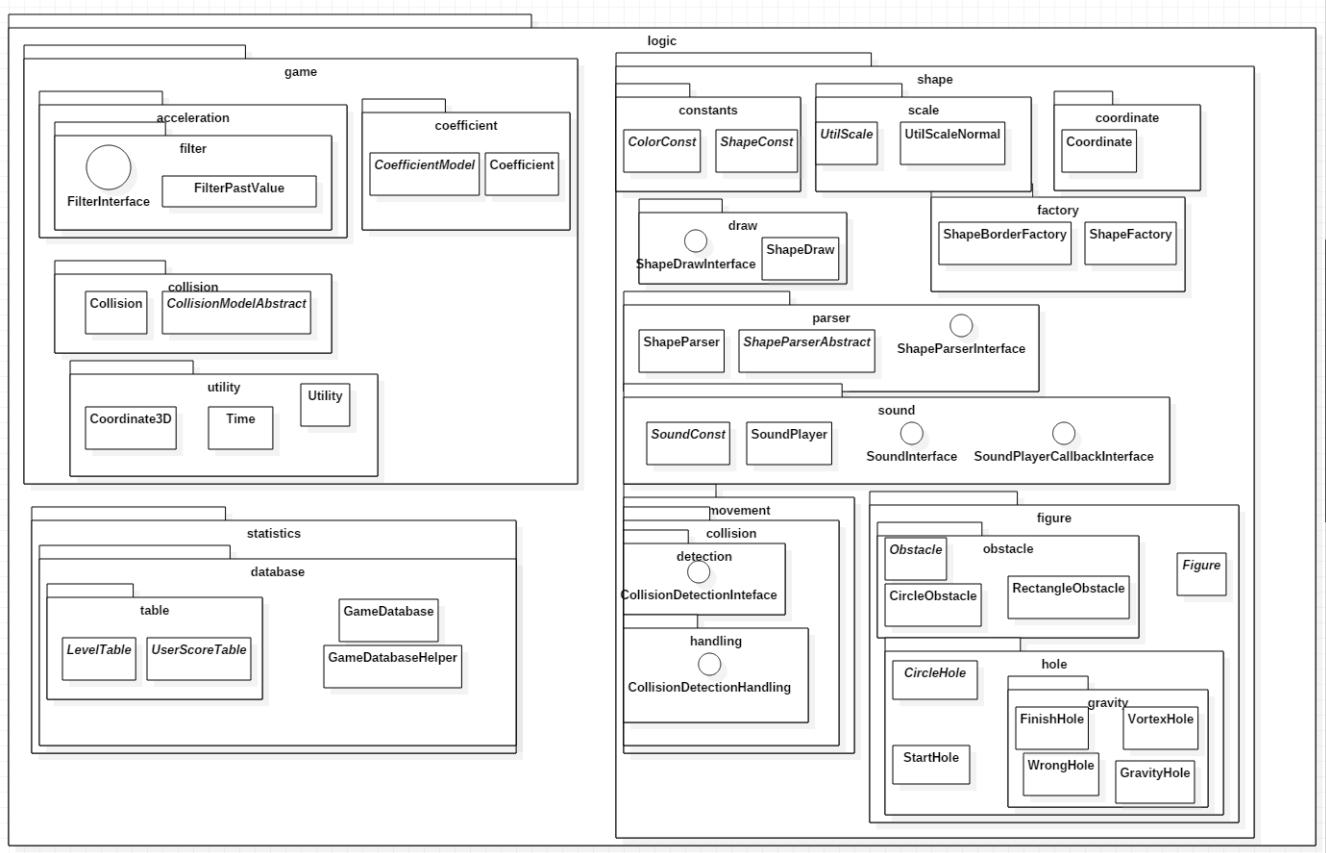
Архитектура

7.1 Организација пакета



Слика 7.1: Организација класа по пакетима (application пакет)

Читав java код је организован тако да се налази унутар пакета `com.example.popina.projekat` и то у два потпакета. Код који је везан за MVC преглед и Android део налази се у `application` потпакету (слика 7.1). Код који је везан за логику игре (база података, како се праве фигуре, парсира...) налази се у потпакету `logic` (слика 7.2). Логика класа из потпакета `application` је објашњена у глави 4, као и у глави 6. Такође логика пакета `coefficient` је објашњена у глави 4, као

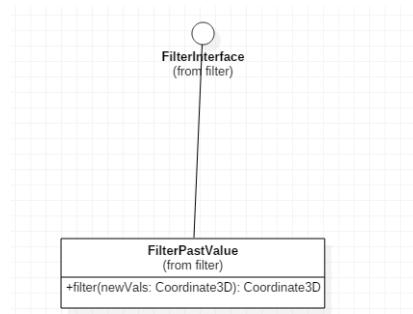


Слика 7.2: Организација класа по пакетима (logic пакет)

и логика *collision* пакета унутар кога је *collisionModel*. Стога овде ће бити објашњена организација кода у потпакету *logic* који описује логику из архитектурног угла тј. како ради апликација.

7.2 Пакет game

7.2.1 Пакет acceleration.filter



Слика 7.3: Класа filter

Унутар овог пакета се налази интерфејс *FilterInterface* чија метода *filter* прима очитане вредности убрзања са улаза и враћа филтриране вредности. Ово спречава да се дешавају нагле промене убрзања услед случајно лошег очитавања сензора. У апликацији је имплементирана у виду *FilterPastValue* класе (слика 7.3). Ова класа филтрира тако што последњу филтрирану вредност и ону детектовану скалира тако да $\alpha (0 \leq \alpha \leq 1)$ се множи са новом вредношћу, а са $1 - \alpha$ са старом и то се сабира. Ова класа се користи код филтрирања вредности убрзања у *GameActivity*.

7.2.2 Пакет coefficient

У овом пакету класа *Coefficient* са методом *updateValues* чува коефицијенте унутар *SharedPreferences* (чије име се налази у *CoefficientModel*).

7.2.3 Пакет collision

Класа *CollisionModelAbstract* има методу *updateSystem* која прима за аргументе филтрирано убрзање, време детекције сензора, листу *Figure* које представљају препреке, или циљ за лопту, као и саму лопту. Након позива ове методе треба да буде ажурирана позиција лопте, као и пуштен одговарајући звук player-ом који је примила класа у конструктору. Враћа једну од 4 вредности које кажу да ли је игра побеђена, изгубљена, да ли има колизије или нема колизије. Метода *setLastTime* служи за иницијализацију референтног времна од кога ће мерити промена брзине лопте.

7.2.4 Пакет utility

Овде се садрже како само име каже Utility ствари као што су *Coordinate3D* која представља 3D координату тачке у простору, *Time* која представља временски интервал која има почетак и крај и чија дужина може да се рачуна преко методе *timeInt*.

Класа *Utility* обухвата методе које служе за рад са координатама, конверзијама, насумичним бројем. Редом су наведени потиси и описи:

- *static double radianToDeg(float rad)* - претвара угао *rad* из радијана у степене и враћа као повратну вредност.
- *static double degToRadian(float deg)* - претвара угао *deg* у степенима у радијане и враћа као повратну вредност.
- *static float convertMsToS(float ms)* - претвара из милисекунди *ms* у вредност у секундама.
- *static float convertNsToS(float ns)* - претвара из наносекунди *ns* у вредност у секундама.
- *static float opositeSign(float val)* - враћа супротан знак од броја *val*.

- *static float convertRadianAngleTo2PiRange(float angle)* - пребацује угао *angle* у радијанима у $[0, 2\pi]$ интервал.
- *static double randomNumberInInterval(int startInterval, int endInterval)* / враћа број у задатом интервалу $[startInterval, endinterval]$.
- *static Coordinate rotatePointAroundCenter(...)*-ротира тачку око центра за одређени угао (тамо где нема центра узима се $(0, 0)$ за центар, тамо где нема угла користе се израчунате вредности синуса и косинуса које су прослеђене за брже рачунање ротације).
- *static float calculateAngle(Coordinate center, Coordinate point)* - рачуна угао између *x* осе и праве одређене тачком *point*и центром *center*.
- *static boolean doesSegmentIntersectsCircle(Coordinate beginSegment, Coordinate endSegment, Coordinate center, float radius, boolean isXLine)* - одређује да ли круг (*Coordinate* центар и *radius* полуупречник) сече прослеђени сегмент (почетак сегмента је тачка *beginSegment*, крај *endSegment*), са тим да су сегменти увек паралелни *x* или *y* оси што се прослеђује параметром *isXLine* (да ли је *x* оса).
- *static boolean isDimBetweenDims(float dimBegin, float dimEnd, float dim)* - одређује да ли је вредност *dim* између две вредности *dimBegin* и *dimEnd* на реалној правој, при чему се користи одступање од 0,01.
- *static float distanceSquared(Coordinate point1, Coordinate point2)* - враћа растојање између две координате *point1* и *point2*квадрирано.
- *public static boolean isDistanceBetweenCoordLessThan(Coordinate coordinate1, Coordinate coordinate2, float dist, boolean isSquared)* - Одређује да ли је растојање између две координате *coordinate1* и *coordinate2* мање од прослеђеног *dist*, при чему се користи тачност од 0,01. Ако је растојање већ квадрирано нема потребе да се квадрира (што се може проследити као параметар *isSquared*).

7.3 Пакет statistics.database

Овај пакет садржи две класе *GameDatabaseHelper* и *GameDatabase*. *GameDatabaseHelper* проширује класу *SQLiteOpenHelper*¹ и служи да омогући згодније мењање шема база (уништавање старе базе и ажурирање на нову).

GameDatabase је једна огромна фасада за приступ бази податка. Редом су наведене њене методе као и њени описи:

- *String getFirstLevel()* - враћа име првог полигона из табеле *LevelTable*.

¹<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper.html>

- *int insertUser(String user, String levelName, long time)* - убацује време *time* корисника *user* за полигон *levelName* у табелу *UserScoreTable*.
- *int insertLevel(String levelName, int levelDifficulty)* - убацује полигон *levelName* у табелу *LevelTable* при чему му је тежина *levelDifficulty*. У случају да постоји полигон са истим именом стари полигон ће бити обрисан.
- *Cursor queryHighScore(String levelName)* - враћа *Cursor*² који кад се итерира садржи сортирану опадајуће листу времена са одговарајућим корисницима за полигон *levelName*.
- *int deleteHighScore(String level)* - брише листу времена за полигон *level* из табеле *UserScoreTable*.
- *int deleteLevel(String level)* - брише полигон (а самим тим и резултате везане за њега) из базе података-
- *int getDifficulty(String levelName)* - враћа тежину за одговарајући полигон *levelName*
- *LinkedList<String> getLevels(int difficulty)* - враћа листу нивоа са тежином *difficulty*.

7.3.1 Пакет table

Овај пакет садржи две класе које у суштини представљају табеле, тј. садрже имена колона која им припадају као и одговарајући SQL који служи за њихово генерирање и уништавање.

LevelTable у себи садржи поред *_ID*-а и колоне *TABLE_COLUMN_LEVEL_NAME* (представља име полигона које корисник сачува) као и *TABLE_COLUMN_LEVEL_DIFFICULTY* што представља тежину нивоа. Стављен је *Unique constraint* ограничење на име полигона, да случајно се не дода више редова са истим именом полигона, него да се увек ажурира један.

emphUserScoreTable у себи садржи поред *_ID*-а и колоне *TABLE_COLUMN_USER_NAME* што представља име играча који је на ранг листи, Ту су и колона *TABLE_COLUMN_TIME* која представља време за које је пређен тај полигон. Кao и колона *TABLE_COLUMN_FK_LEVEL* која садржи *_ID* реда из табеле *LevelTable* који референцира (тј. полигон ком припада тај резултат).

7.4 Пакет shape

7.4.1 Пакет constants

Унутар овог пакета се налазе две класе *ColorConst* и *ShapeConst*. *ColorConst* чува константе везане за боје одређених фигура. Док *ShapeConst* чува константе

²<https://developer.android.com/reference/android/database/Cursor.html>

везане за почетне позиције одређених фигура и величину. Као и податке неопходне при чувању полигона у фајл (имена фигура, којим редом се стављају параметри...)

7.4.2 Пакет coordinate

Унутар овог пакета налази се класа *Coordinate* која представља једну тачку или вектор (у зависности од тога за шта је потребна). Стога подрава скларани производ два вектора (*scalarProduct*), одузимање (*subCoordinate*) и сабирање вектора (*addCoordinate* - враћа као нови вектор, док *addToThisCoordinate* мења вектор за који је позвана), величину (*magnitudeSquared*) и дохватање и мењање одговарајућих координата. Такође подржава методу *toString* јер се користи код чувања полигона.

7.4.3 Пакет draw

Има једну класу и један интерфејс. Интерфејс *ShapeDrawInterface* има следеће уговоре које класе које га имплементирају морају имати :

- *void drawOnCanvas(Canvas canvas)* - мора да црта себе на *canvas* -у.
- *void moveTo(Coordinate coordinate)* - мора да помери своју позицију (центар или како је везано) на координату *coordinate*.
- *void resize(Coordinate c)* - мора да промени величину ако се зна да је кликнута координата *c*.
- *boolean isCoordinateInside(Coordinate c)*- враћа *true* ако је *c* унутар дате фигуре.
- *void rotate(Coordinate c, float angle)* - ротира фигуру на кликнуту тачку *c*, ако се зна почетни угао нагиба фигуре *angle* кад је кренуло ротирање фигуре.
- *float calculateAngle(Coordinate point)* - рачуна угао између угла ротиране фигуре и оног одређеног центром те фигуре и тачком *point*

Класа *ShapeDraw* служи за испртавање *ShapeDrawInterface* по *Canvas*-у. Њена употреба може се прочитати у глави 6. Полье типа *CommonModel* служи за синхронизацију међу нитима. Методе које подржава:

- *void spriteOnBackground(LinkedList<? extends ShapeDrawInterface> listFigures)* - црта листу фигура *listFigures* на *Canvas* који већ садржи..
- *void drawOnCanvas(LinkedList<? extends ShapeDrawInterface> listFigures, Canvas canvas)* - црта листу фигура *listFigures* на *canvas*, при чему се прво постави подразумевана позадина која је учитана као позадина (по њој ће се цртати).
- *public void drawOnCanvas(ShapeDrawInterface shapeDrawInterface, Canvas canvas)* - црта фигуру *shapeDrawInterface* на *canvas*.

7.4.4 Пакет factory

Овај пакет представља модификацију пројектног обрасца Апстрактна фабрика. При чему не постоји више фабрика, него једна која прави све објекте и која прима *UtilScale* за скалирање фигура (кад се прочитају из фајла у процентима, да би направила онакве какви одговарају екрану корисника), као и за обрнуто скалирање (кад треба да се сачувавају). Њена основна намена је прављење фигура одговарајућег типа. Клас којом је она подржана је *ShapeFactory*. Следе битне методе:

- *StartHole createStartHole()*-прави почетну позицију лопте, скалирану за екран уређаја.
- *FinishHole createFinishHole()*- прави рупу у коју лопта треба да уђе, скалирану за екран уређаја.
- *WrongHole createWrongHole()*- прави амбис у који лопта не сме да упадне, скалиран за уређај екрана.
- *RectangleObstacle createObstacleRectangle()*- прави правоугаону препреку од које се лопта одбија, скалирану за уређај екрана.
- *CircleObstacle createObstacleCircle()*- прави кружну препреку од које се одбија лопта, скалирану за уређај екрана.
- *VortexHole createVortexHole()*-прави вртлог у који лопта ако упадне врти се и избацује из ње, скалиран за уређај екрана.
- *Figure scaleFigure(Figure f)*- скалира фигуру *f* за екран користећи *UtilScale* који је прослеђен у конструктору .
- *Figure scaleReverse(UtilScale utilScale)*- скалира фигуру *f* за фајл користећи *UtilScale* који је прослеђен у конструктору .
- *LinkedList<Figure> scaleFigures(LinkedList<Figure> listFigures)*-скалира листу фигура *listFigures*за екран користећи *UtilScale* који је прослеђен у конструктору.
- *inkedList<Figure> scaleReverseFigures(LinkedList<Figure> listFigures)*-скалира листу фигура *listFigures*за фајл користећи *UtilScale* који је прослеђен у конструктору.

Класа *ShapeBorderFactory* садржи методу чији потпис је *LinkedList<RectangleObstacle> createBorders()* и која генерише листу зидова (четири) као листу правоугаоних препрека скалираних за екран. Ово омогућава избегавање посебних провера за зидове (јер се зидови посматрају као фигуре).

7.4.5 Пакет figure

Унутар овог пакета, а и његових подпакета налазе се класе које имплементирају лопту, и препреке.

Figure класа имплементира интерфејсе *ShapeParserInterface*, *ShapeDrawInterface*, *SoundInterface*, *CollisionDetectionInterface* који се користе код чувања у фајлове, пуштања звука, испртавања, и детектовања и обрађивања колизије. Поред тога ако фигура дође у стање мировања омогућава да се звук не пушта више. Звук се такође не пушта ако лопта има учесталу колизију са препракама. Такође *toString* метода је неопходна због чувања у фајловима. Има свој центар који ће се користити код померања, ротирања... Центар може да се дохвати и постави.

Пакет hole

Постоји класа *CircleHole* која имплементира лопту и лопасте препреке у 2D. Изведена је из класе *Figure*. Самим тим имплементира методе из *ShapeDrawInterface* и *CollisionDetectionInterface*, уз додато поље полуупречник, које може да се дохвати и постави. Постоји и класа *StartHole* која служи за приказивање лопте, додатно још имплементира методе из *SoundInterface* као и промењену *toString* методу.

Садржи пакет *gravity* који служи за класе које представљају рупе (тј. има неки облик гравитације који вуче лопту ка њима). Стога *GravityHole* имплементира још *CollisionHandlingInterface* који омогућава промену брзине након судара (у суштини мења брзину тако да иде ка центру лопте). У случају *FinishHole* и *WrongHole* које проширују *GravityHole* долази до краја игре кад лопта упадне у њих (с тим да је у једном случају позитиван, а у другом негативан крај). Док *VortexHole* такође проширује, али игра се не губи кад лопта упадне у њу, нити побеђује. Такође *toString* и звук су другачији за сваки тип препреке.

Пакет obstacle

У овом пакету постоје класе *Obstacle*, *CircleObstacle* и *RectangleObstacle*. Прва проширује класу *Figure*, и поставља одговарајући звук који ће бити пуштан за препреке. Такође имплементира *CollisionHandlingInterface* који се користи након судара да се промени брзина објекта, тј. није крај игре кад се лопта судари са њима. Друге две класе проширују класу *Obstacle*. И код *CircleObstacle* и *RectangleObstacle* све методе класе *Figure* су override-оване и самим тим ту су имплементације за фигуре типа правоугаоник и круг у 2D. Поред тога *RectangleObstacle* садржи и поља *width* и *height* која могу да се дохвате и поставе и представљају одговарајућу ширину и висину. Може да се дохвате и координате одговарајућих темена. Слично за лопту може да се дохвати полуупречник.

7.4.6 Пакет movement.collision.detection

Налази се интерфејсе *CollisionDetectionInterface*. Класе које га имплементирају морају да имају следеће методе:

- *boolean doesCollide(CircleHole ball)* - да ли постоји судар између лопте *ball* и објекта класе која имплементира интерфејс .
- *boolean isGameOver()* - да ли је игра по судару лопте са објектом класе која имплементира интерфејс готова.
- *boolean isWon()* - да ли је игра по судару лопте са објектом класе која имплементира интерфејс побеђена или изгубљена (мора прво претходна метода да врати *true*).

7.4.7 Пакет movement.collision.handling

Налази се интерфејсе *CollisionHandlingInterface*. Класе које га имплементирају морају да имају следеће методе:

- *Coordinate getSpeedChangeAfterCollision(StartHole ballOld, StartHole ballNew, Coordinate3D speed)* - враћа вектор који треба додати вектору брзине *speed* тренутног кретања лопте *ballOld*, при чему је потенцијална нова позиција *ballNew*.

7.4.8 Пакет parser

Садржи класе и интерфејсе који омогућавају чување полигона у фајл и његово учитавање ради даљег приказивања на екран.

ShapeParserInterface је интерфејс који омогућава читање фигура из фајлова и њихово скалирање. Следеће методе класе које имплементирају интерфејс морају имати:

- *ShapeParserInterface scale(UtilScale utilScale)* - враћа објекат класе која имплементира *ShapeParserInterface* скалирану за екран уз помоћ *utilScale*.
- *ShapeParserInterface scaleReverse(UtilScale utilScale)* - враћа објекат класе која имплементира *ShapeParserInterface* скалирану за фајл (у процентима) уз помоћ *utilScale*.

ShapeParserAbstract је апстрактна класа која чита *ShapeParserInterface* из фајла и уз помоћ објекта класе *ShapeFactory* прави фигуре, и уз помоћ објекта класе *ShapeDraw* их исцртава на екрану. Класе које је изводе морају да подрже следеће методе:

- *ShapeParserInterface parseLine(String line)*- изведена класа парсира линију фајла и враћа објекат интерфејса *ShapeParserInterface*.
- *LinkedList<? extends ShapeParserInterface> parseFile(String fileName)*- парсира фајл *fileName* при чему позива *parseLine* методу, и генерише листу фигура спремних за цртање на дати екран.

- `void drawImageFromFile(Canvas canvas, String fileName)`- црта фигуре из фајла `fileName` на платно `canvas`.

Класа `ShapeParser` је изведена из `ShapeParserAbstract` и подржава све методе из њеног интерфејса, с тим да свуда генерише објекте изведених класа из `Figure` уместо `ShapeParserInterface`

7.4.9 Пакет scale

Поседује две класе. Прва је апстрактна `UtilScale` и кад се иницијализује прима величину екрана и на основу тога скалира дужине по висини (`scaleHeight` и `scaleReverseHeight`) ширини (`scaleWidth` и `scaleReverseWidth`) као и координате (`scaleReverseCoordinate` и `scaleCoordinate`). `scalexxx` скалира вредности за екран уређаја, док `scaleReverse` скалира за фајл (претвара у процене). Класа `UtilScaleNormal` имплементира претходно наведене методе.

7.4.10 Пакет sound

Садржи класе и интерфејсе за звук током судара лопте са препреком. Класа `SoundConst` садржи константе везане за имплементацију музичког player-а. `SoundPlayerCallback` је интерфејс чије имплементације омогућавају пуштање звука са редним бројем дефинисаним у `SoundConst` (као `idSound` се прослеђује) преко методе чији је потпис

`void playSound(int idSound).`

Имплементиран је у виду класе `SoundPlayer`.

`SoundInterface` интерфејс условљава фигуре којега имплементирају да пуштају одређени звук приликом судара преко методе

`void playSound(SoundPlayerCallback soundPlayerCallback).`

При чему је `soundPlayerCallback` player преко кога се пушта звук.

Глава 8

Закључак

Игрица је за циљ имала да научим Android API. Али проблеми попут рачунарске графике и моделирања физике пребацили су тежиште пројекта.

Стога игрица је постигла првобитни циљ, а то је Android апликација чији код је читак, добро организован, лак за проширивање. Додавање нових препрека у игри је веома једноставно извођењем из постојећих класа. Додавање нових опција у модел физике је веома једноставно мењањем једне методе. Механика играња тј. исцртавања фигура апликације је оптимизована до границе кад корисник не осећа да се ради о Android уређају. У суштини, читав апликација је добро организована и проширива у свим погледима.

Међутим проблем савршене физике остаје отворено питање. Иако ми физика није била омиљени предмет током школовања, модел који је урађен, довољан је да корисник не примети несавршеност физике у неким деловима. Требало да се среди даљим изучавањем моделирања физике у оваквим типовима апликације.

Следеће отворено питање да ли користити OpenGL и да ли ће он дати још већа побољшања у погледу исцртавања фигура. И са тим питање, да ли треба да се поред лопте која се креће треба да дода неколико покретних препрека. Да ли би оне кориснику дале још више уживања? Све ово захтева темељно испитивање OpenGL API-а за Anrdoid и његово темељно тестирање на одговарајућим моделима физике који буду коришћени.

Треће отворено питање је изглед апликације. Иако сам се трудио у сваком погледу да постигнем да апликација буде што привлачнија за кориснике, видно је да неки делови захтевају побољшање (попут боја препрека, лопте, могућност постављања позадине нивоа, ...). Стога треба наћи добро GUI дизајнера за Android који би предложио неке промене.

Последње отворено питање је сама логика апликације, тј. како се ради adventure мод игрице, и да ли треба дозволити кориснику да прави сам нивое. Стога треба извршити истраживање код корисника да се види шта они очекују од овакве апликације по питању нивоа.

И последње питање је, да ли апликација треба да иде у продукцију, тј. на Google Play Store (линк [11]). Ако буде постојала жеља, неопходно је систем тестирања подићи на веома озбиљан ниво, да се и најмањи багови уклоне.

Литература и линкови

- [1] Google. *Android Developers*.
<https://developer.android.com/index.html>
- [2] Саша Стојановић, Захарије Радивојевић, Милош Џветановић. *Програмирање мобилних уређаја*.
<http://rti.etf.bg.ac.rs/rti/si4pmu/>
- [3] Stack Exchange Inc. *stack overflow*.
<https://stackoverflow.com/>
- [4] RockStar Games. *GrandTheftAuto*. <http://www.rockstargames.com/grandtheftauto/>
- [5] Ian Millington. *Game Physics Engine Development*. CRC Press. 2nd Edition. 2010.
- [6] Ђорђе Живановић. *Ball Game*. <https://bitbucket.org/popina1994/ball-game>
- [7] Ian Millington. *Cyclone physics system*. <https://github.com/idmillington/cyclone-physics/tree/master/src>
- [8] *A brief history of smartphones*. <http://www.thesnugg.com/a-brief-history-of-smartphones.aspx>
- [9] *Mobile/Tablet Operating System Market Share*. <https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=8&qpcustomd=1>
- [10] *Hill Climb Racing*. <https://play.google.com/store/apps/details?id=com.fingersoft.hillclimb&hl=sr>
- [11] Google. *Google Play*. <https://play.google.com/store>