

ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

ДИПЛОМСКИ РАД

ИЗ ПРОГРАМИРАЊА МОБИЛНИХ УРЕЂАЈА

---

Android апликација - лопта на плочи

---

*Аутор*

Ђорђе Живановић, 0033/2013

*Ментор*

Др Саша Стојановић

2. јул 2017.

# Садржај

<b>1 Увод</b>	<b>2</b>
1.1 Мотивација . . . . .	2
1.2 Постојећа решења . . . . .	3
1.3 Коришћени алати и машине . . . . .	3
1.4 Структура . . . . .	3
<b>2 Функционалности</b>	<b>5</b>
2.1 Почетни екран . . . . .	5
2.1.1 Основни . . . . .	5
2.1.2 Мени . . . . .	6
2.1.3 Опције полигона . . . . .	6
2.2 Игра . . . . .	7
2.2.1 Игра побеђена . . . . .	8
2.2.2 Игра изгубљена . . . . .	9
2.3 Прављење полигона . . . . .	9
2.3.1 Опције за уређивање полигона . . . . .	9
2.3.2 Чување полигона . . . . .	10
2.4 Резултати . . . . .	11
2.5 Подешавања . . . . .	12
<b>3 Android</b>	<b>13</b>
3.1 GUI код . . . . .	13
3.2 Java код . . . . .	14
3.3 Чување података . . . . .	15
3.4 Unit Test . . . . .	16
<b>4 Модел судара</b>	<b>17</b>
<b>5 Графика</b>	<b>18</b>
5.1 Општа идеја . . . . .	18
5.2 Оптимизације код играња игре . . . . .	18
<b>6 Архитектура</b>	<b>20</b>
6.1 Организација пакета . . . . .	20
6.2 Пакет game . . . . .	21
6.2.1 Пакет acceleration.filter . . . . .	21

6.2.2	Пакет coefficient . . . . .	22
6.2.3	Пакет collision . . . . .	22
6.2.4	Пакет utility . . . . .	22
<b>7</b>	<b>Заключак</b>	<b>24</b>
	<b>Литература и линкови</b>	<b>25</b>

# Глава 1

## Увод

Основни циљ пројекта био је да истражим *Android API*<sup>1</sup> који се користи за прављење апликација за *Android* уређаје. Да бих у потпуности разумео његову употребу и организацију кода у апликацијама за *Android*, морао сам да приступим изазову прављења игрице попут ове. Поред *Android API*-а први пут сам се сусрео са проблемима моделирања физике у рачунарској графици. У овом поглављу сам се осврнуо на мотивацију за имплементацију на овај начин, као и изазове са којима сам се сусрео у изради пројекта.

### 1.1 Мотивација

Вештина коју програмер мора да има је прилагођавање новонасталим ситуацијама и решавање проблема који дођу у њима.

Прављење великог пројекта са непознатим API-ем један од великих изазова на које сам наишао. Први корак у томе био је предмет програмирање мобилних уређаја ([2]), који сам слушао и на ком сам научио основне ствари које нуди *Android API*. Други корак је било злостављање *AndroidDeveloper* сајта ([1]) на ком смо налазили на предмету одговарајуће класе, методе, константе, чланке који су нам омогућавали прављење жељених апликација. Последњи корак на који се прибегавало кад проблем није могао бити решен је *StackOverflow* ([3]). Наведени кораци су ми омогућавали и давали мотивацију за даљи рад у борби са непознатим.

Други велики мотив био је рачунарска графика. Пошто као сваки мали дечак у великом телу сам имао жељу да научим како раде игрице попут *GTA* ([4]). Први корак ка томе било је моделирање нечега што се може назвати модел судара. Дати пројекат је био изазован у погледу организације кода за модел судара, јер су сви делови уско спретнути.

Последњи мотив, а можда и најважнији била је организација кода у великом пројекту као што је овај, која омогућава брзу проширивост и лагано додавање нових функционалности, мењање постојећих модела судара... Ово је врхунац знања сваког програмера.

---

<sup>1</sup>Application programming interface

## 1.2 Постојећа решења

Битна особина коју програмер мора да поседује је способност да чита туђи код и да прилагођава својој имплементацији. Књига која ми је помогла 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 дosta 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].

## 1.3 Коришћени алати и машине

Користио сам програмски језик Java искључиво јер омогућава лакшу израду пројекта и лакше дебаговање. У комбинацији са Java коришћен је xml за израду GUI<sup>2</sup>. Коришћен је Android Studio<sup>3</sup> који ми је дао 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 коришћених алатова:

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

Табела 1.1: Алати коришћени при развоју пројекта.

Машина на којој је писан и компајлиран код и на којој је радио Android Studio је HP Omen са 12GB RAM, 512GB SSD, i7-6700HQ 2,7GHz, оперативни систем Windows 10.0.15063 (у време писања последњих измена на коду). Машина на којој је покретана апликација је NVIDIA Shield Tablet K1, који је у тренутку првог инсталирања имао Android 4.4 верзију инсталирани. У тренутку писања ажуриран је на Android 7.0. Минимални Андроид ОС<sup>4</sup> који подржава апликацију је 5.0.

## 1.4 Структура

У глави 3 биће објашњено који делови Android API-а су коришћени, као и њихове мање и врлине, и зашто су баш они одабрани. У глави 4 биће причано о моделу који судара који је коришћен и зашто су биране његове компоненте. У глави 5 биће описан систем за исцртавање екрана. У глави 6 биће описана софтверска архитектура

<sup>2</sup>Graphical User Interface

<sup>3</sup><https://developer.android.com/studio/index.html>

<sup>4</sup>Оперативни систем

(пројектни обрасци, организација кода). У глави 2 биће наведени неки случајеви коришћења система и како се систем понаша у одређеним тренуцима. У глави 7 биће наведени закључци, да ли систем може да се побоља, да ли могу да се додају нове функционалности и сл.

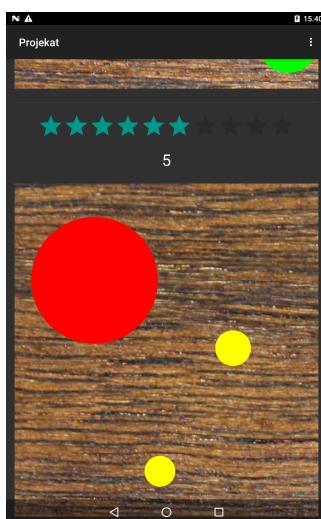
# Глава 2

## 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

### 2.1 Почетни экран

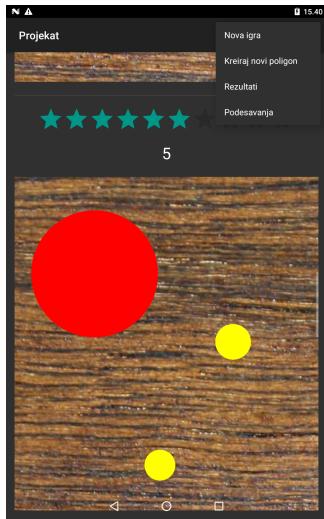
#### 2.1.1 Основни



Слика 2.1: Почетни экран

Кад корисник покрене апликацију појављује му се листа полигона које може да игра, са изгледом полигона (слика 2.1). 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.

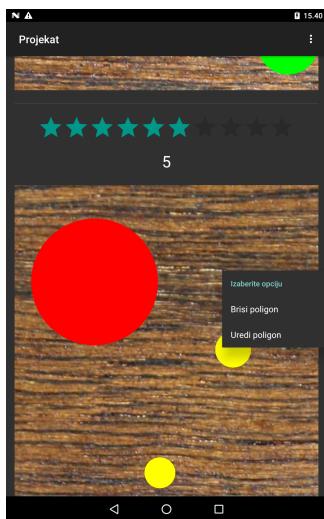
## 2.1.2 Мени



Слика 2.2: Мени

Корисник све време на врху прозора има мени који отвара ако прстом кликне на три тачке. Из менија који се појави (слика 2.2) 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.

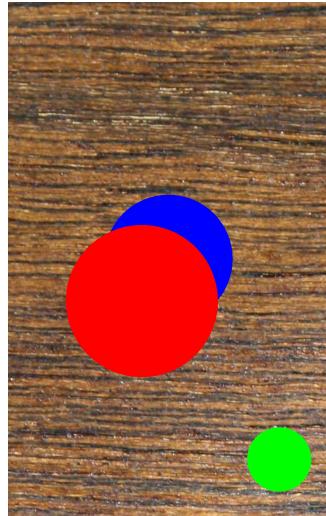
## 2.1.3 Опције полигона



Слика 2.3: Падајући мени везан за полигон

Корисник задржавањем прста на полигон добија падајући мени (слика 2.3) са две опције. Прва опција брише полигон из целе игре. А друга омогућава његово уређивање, тако што отвара нови прозор са датим полигоном.

## 2.2 Игра



Слика 2.4: 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 2.4). 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 2.8). Типови препрека:

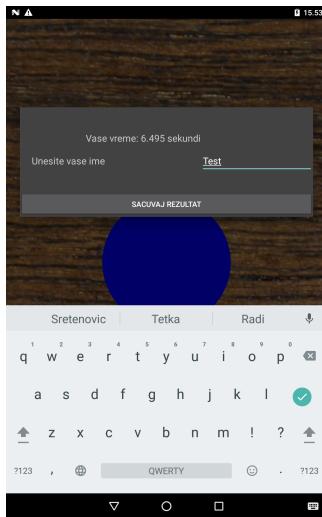
1. Зид (ивица екрана и жути правоугаоник) - лопта се одбија од зида под истим углом којим улази. Губитак енергије током судара зависи од коефицијента подешеног у подешавањима.
2. Стуб (жути круг) - исто као зид, са тим да је кружног облика.
3. Амбис (црни круг) - лопта кад упадне у њега играч губи игру.
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.<sup>1</sup>

---

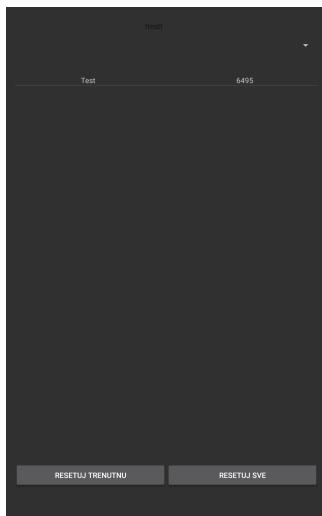
<sup>1</sup>Амбис, вртлог и крај кад лопта их дотакне вуку је ка себи одређеном силом

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.

### 2.2.1 Игра побеђена



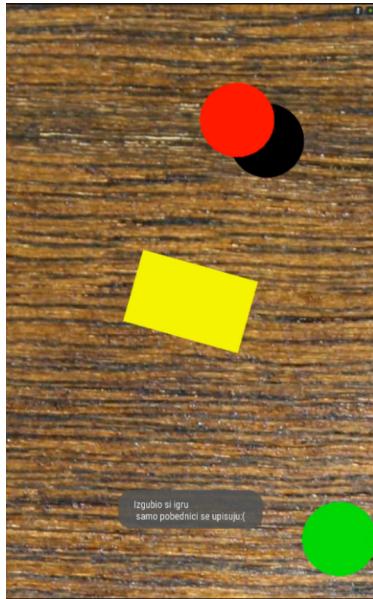
Слика 2.5: Дијалог кад победи



Слика 2.6: Листа резултата за полигон који је победио

Кад корисник убаци лопту у зелену рупу, искаче му дијалог (слика 2.5). 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 2.6).

### 2.2.2 Игра изгубљена



Слика 2.7: Обавештење кад корисник изгуби

Када корисник упадне у амбис изађе му обавештење да је изгубио игру (слика 2.7).

## 2.3 Прављење полигона

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

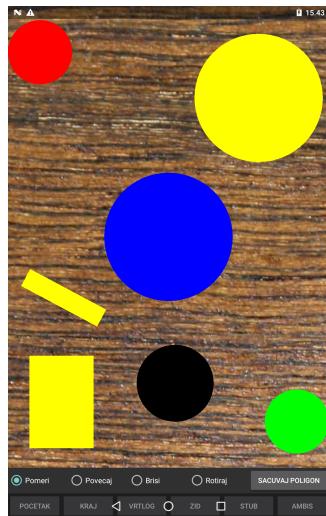
Један је мод уређивања, у ком се налази тако што или је кликнуо *Uredi poligon* на почетном екрану или је сачувао полигон под неким именом. У овом моду кад корисник кликне назад аутоматски ће се чувати поруке и бити избацивана упозорења ако нешто није како треба (фали крајња позиција, почетна...).

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.

### 2.3.1 Опције за уређивање полигона

Корисник има опције генерисања препрека за лопту наведеним у секцији 2.2 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

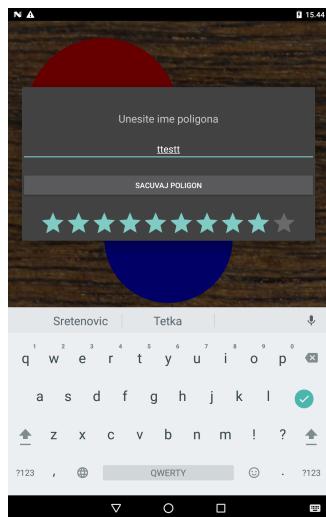


Слика 2.8: Екран који корисник види кад уређује полигон

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.

### 2.3.2 Чување полигона

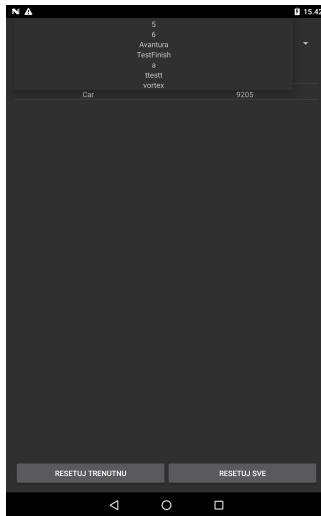


Слика 2.9: Дијалог који се појави кад корисник хоће да сачува полигон

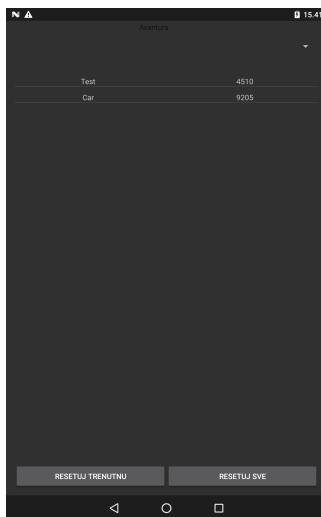
На крају корисник кад заврши генерисање правоугаоника има дугме *SACUVAJ POLIGON*. Тад се кориснику појављује дијалог на коме уноси тежину полигона као

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

## 2.4 Резултати



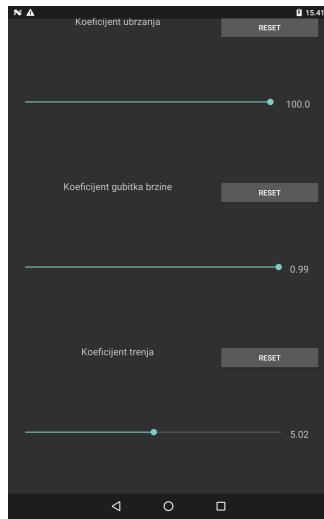
Слика 2.10: Избор полигона за који хоће да се виде резултати



Слика 2.11: 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 2.10) и потом, му се приказују резултати сортирани од најбољег ка најлошијем (2.11) у погледу времена. Корисник има могућност ресетовања листе резултата за један изабрани полигон притискањем дугмета *RESETUJ TRENUINU* или ресетовањем резултата свих листи са притискањем дугмета *RESETUJ SVE*.

## 2.5 Подешавања



Слика 2.12: Подешавања за игру

Корисник има опцију да подеси одговарајуће коефицијенте које се користе током симулације игре (слика 2.12). То се постиже померањем одговарајуће тачке на дужи. Редом су наведени коефицијенти:

1. Коефицијент убрзања - Колико брзо ће лоптица да убрзана, што већи коефицијент брже убрзана.
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 3

## 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 IT<sup>1</sup> 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])

### 3.1 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 убрзања. Имена сва су писана CamelCase-ом<sup>2</sup>. 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*<sup>3</sup>, док је за неке ствари где је битна само позиција компоненти, коришћен *RelativeLayout*<sup>4</sup>.

Коришћена је Android Dark Material Theme као основна тема.

---

<sup>1</sup>Internet Technology

<sup>2</sup><http://wiki.c2.com/?CamelCase>

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

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

## 3.2 Java код

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

Постоји пет активности (*MainActivity*, *CreatePolygonActivity*, *GameActivity*, *SettingsActivity*, *StatisticsActivity*). Свака подржавајући одговарајућу функционалност из главе 2. Дакле *MainActivity* почетни је экран (2.1), *GameActivity* саму игру (2.2), *SettingsActivity* подешавања (2.5), *StatisticsActivity* резултате (2.4) и *CreatePolygonActivity* уређивање полигона (2.3).

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 MVC<sup>6</sup> 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*<sup>7</sup> и праве одговарајући потребан GUI. Ово је омогућило финије подешавање дијалога и њиховог изгледа од унапред направљених класа попут *AlertDialog*<sup>8</sup>.

Тамо где су била потребна исцртавања на екрану (*CreatePolygonActivity* и *GameActivity*) коришћене су одговарајуће класе које су проширивале *SurfaceView*<sup>9</sup> (биће касније објашњено како ово ради у глави 5).

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

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

<sup>6</sup>Model View Controller

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

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

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

уређај у *GameActivity* активности, коришћен је уграђени Android сензор *TYPE\_ACCELEROMETER*<sup>10</sup>. Он сваких 20ms прослеђује активности детектоване вредности убрзања уређаја по *x*, *y* и *z* оси. Њихово коришћење је описано даље у глави 4. 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*<sup>11</sup> 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 6.

### 3.3 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*<sup>12</sup> 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 SQL<sup>13</sup> 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*.

<sup>10</sup>[https://developer.android.com/guide/topics/sensors/sensors\\_overview.html](https://developer.android.com/guide/topics/sensors/sensors_overview.html)

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

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

<sup>13</sup>Standard Query Language

## 3.4 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 4

### Модел судара

У овом поглављу биће изложен рад са класом

# Глава 5

## Графика

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

### 5.1 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 екрана<sup>1</sup> 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.

### 5.2 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

---

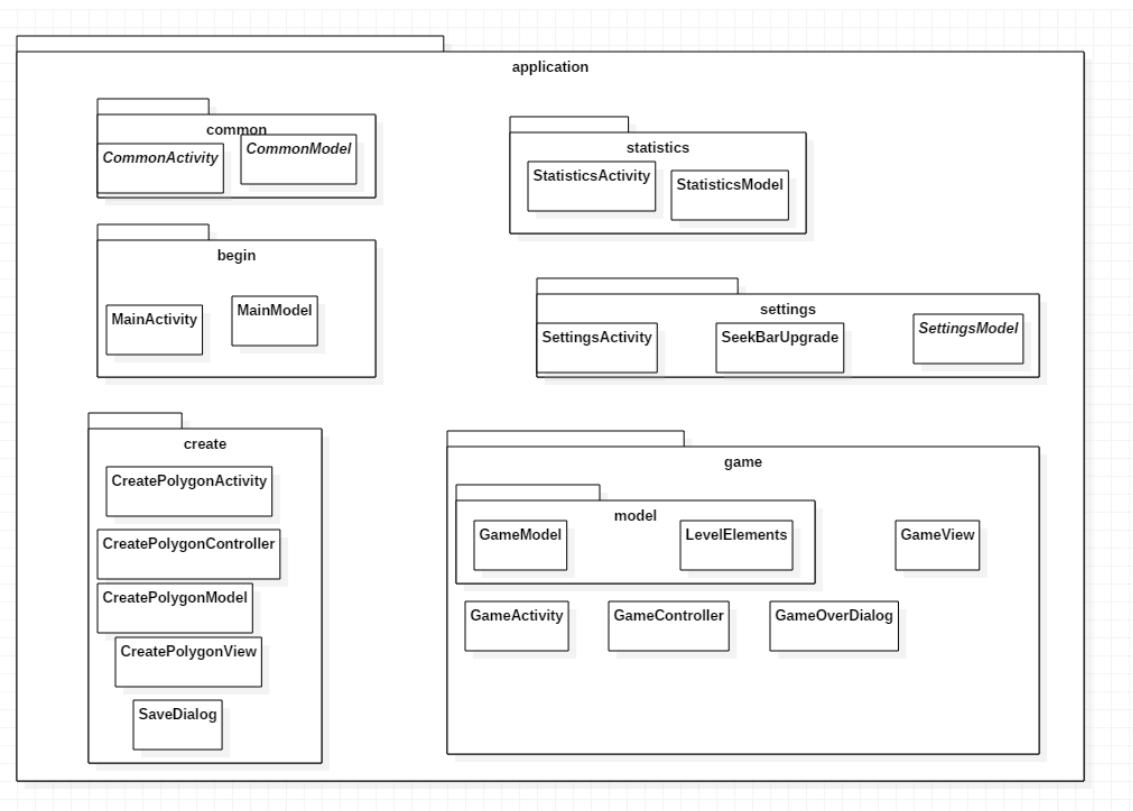
<sup>1</sup><https://developer.android.com/reference/android/widget/ImageView.html>

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

# Глава 6

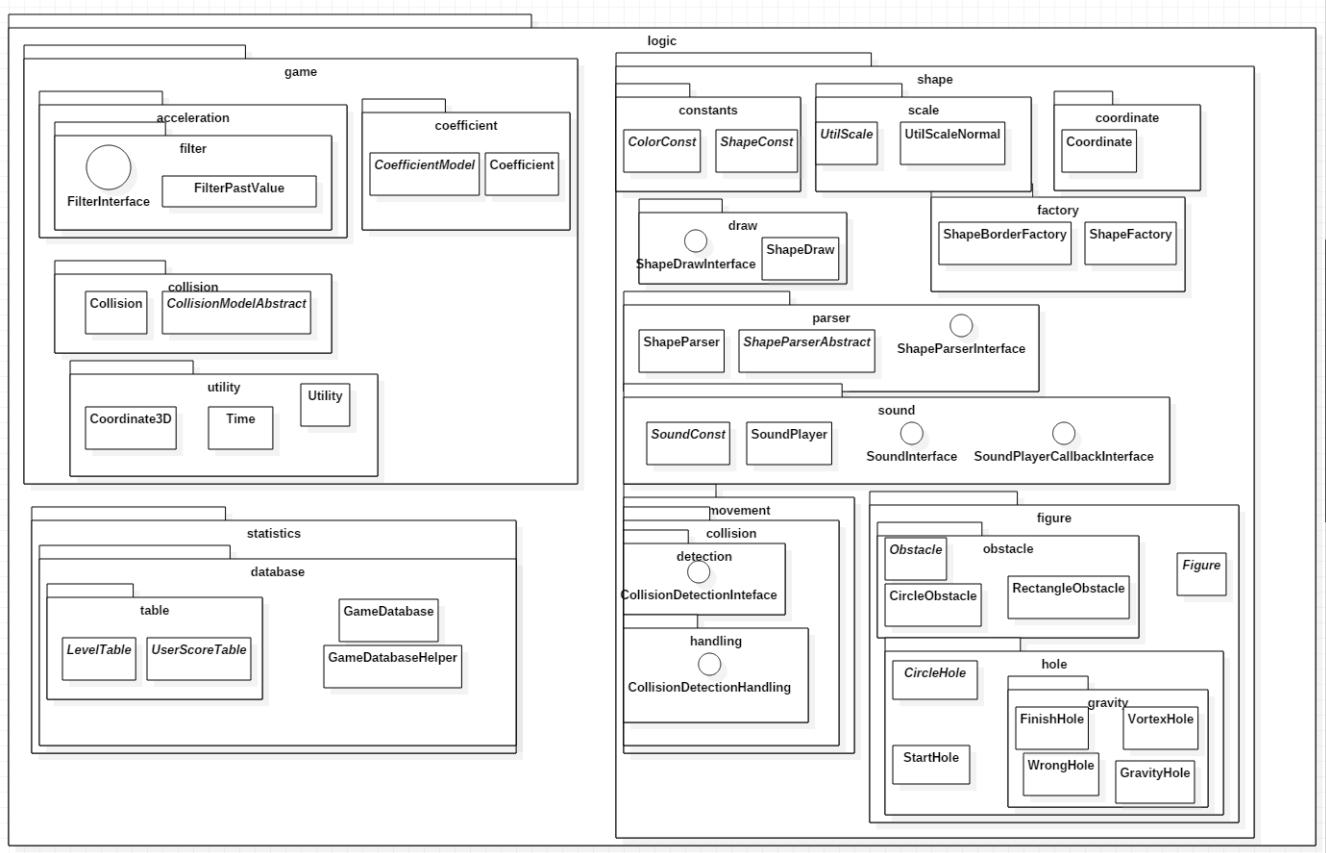
## Архитектура

### 6.1 Организација пакета



Слика 6.1: 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 6.1). Код који је везан за логику игре (база података, како се праве фигуре, парсира...) налази се у потпакету `logic` (6.2). Логика класа из потпакета `application` је објашњена у глави 3, као и у глави 5. Такође логика пакета `coefficient` је објашњена у глави 3, као и

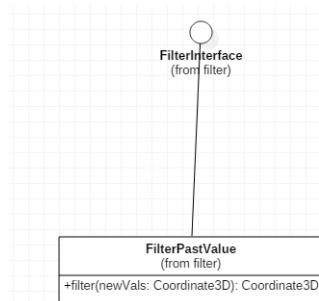


Слика 6.2: Организација класа по пакетима (logic пакет)

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

## 6.2 Пакет game

### 6.2.1 Пакет acceleration.filter



Слика 6.3: 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 6.3). 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*.

### 6.2.2 Пакет coefficient

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

### 6.2.3 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.

### 6.2.4 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. Метода *radianToDeg* претвара угао из радијана у степене. Метода *degToRadian* претвара угао у степенима у радијане. Метода *convertMsToS* претвара из милисекунди у секунде, док *convertNsToS* претвара из наносекунди у секунде. Метода *oppositeSign* враћа супротан знак од броја који је прослеђен. Метода *convertRadianAngleTo2PiRange* пребацује угао у радијанима у  $[0, 2\pi]$  интервал. Метода *randomNumberInInterval* враћа број у задатом интервалу. Метода *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). *calculateAngle* рачуна угао између  $x$  осе и праве одређене тачком и центром. Метода *doesSegmentIntersectsCircle* одређује да ли круг сече прослеђени сегмент, са тим да су сегменти увек паралелни  $x$  или  $y$  оси што се прослеђује последњи параметром. Метода *isDimBetweenDims* одређује да ли је вредност између две вредности на реалној

правој, при чему се користи одступање од 0,01. *distanceSquared* враћа растојање између две координате квадрирано. *isDistanceBetweenCoordLessThan* Одређује да ли је растојање између две координате мање од прослеђеног, при чему се користи тачност од 0,01. Ако је растојање већ квадрирано нема потребе да се квадрира (што се може проследити као параметар).

# Глава 7

## Заключак

# 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>