УНИВЕРЗИТЕТ У БЕОГРАДУ

ФАКУЛТЕТ ОРГАНИЗАЦИОНИХ НАУКА

ЗАВРШНИ РАД

Развој мобилне игрице у Јава окружењу

Ментор

Проф. др Душан Бараћ

Студент

Јован Митрашиновић, 2018/0340

Београд, 2023. године

САДРЖАЈ

АПСТРАКТ 2

1. УВОД 3

2. ОПИС ТЕХНОЛОГИЈЕ И ОСНОВНИХ ПРИНЦИПА 5

2.1 LibGDX ФРЕЈМВОРК И МОДУЛИ 5

2.2 ЖИВОТНИ ЦИКЛУС АПЛИКАЦИЈЕ 7

2.3 УПРАВЉАЊЕ ДАТОТЕКАМА 8

2.4 УПРАВЉАЊЕ УНОСИМА 9

2.5 ИНТЕРНАЦИОНАЛИЗАЦИЈА И ЛОКАЛИЗАЦИЈА 10

2.6 УПРАВЉАЊЕ МЕМОРИЈОМ 10

2.7 MATH UTILITIES 11

2.8 КООРДИНАТНИ СИСТЕМ 12

2.9 ИГРИЦЕ ЗА ДЕЦУ 14

3. ОПИС ПРОБЛЕМА 17

3.1 ОПИС ИГРЕ 17

3.1.1 Типови објеката 17

3.1.2 Циљ и механика игре 17

3.1.3 Избор тежине 17

4. РЕАЛИЗАЦИЈА ИГРЕ 18

4.1 ПРЕГЛЕД АРХИТЕКТУРЕ 18

4.2 ИМПЛЕМЕНТАЦИЈА 18

4.2.1 Рад са средствима 19

4.2.2 Дефинисање конфигурације 21

4.2.3 Пакет објеката 22

4.2.4 Контролер 23

4.2.5 Класа за рендеровање 26

4.2.6 Прављење екрана 29

4.2.7 Помоћни пакет за дибаговање 31

5. ОПИС РАДА СИСТЕМА 32

6. ЗАКЉУЧАК 35

ЛИТЕРАТУРА 36

АПСТРАКТ

Предмет истраживања овог рада је ***LibGDX*** фрејмворл који омогућава развијање могилних игрица за компијутере, као и за мобилне уређаје. Конкретно у овом случају коришћен је за развијање мобилне игрице за андроид уређаје.

Циљ рада је истраживање функционалности овог релативно новог оквира, као и упознавање са основним концептима дводимензионалног програмирања.

У овом раду примењене су основне методе објектно орјентисаног програмирања као и неки од основних дизајн патерна.

Резултат овог рада је потпуно функционална мобилна игрица са графички-корисничким интерфејсом која има неколико нивоа тежине који се могу подесити у зависности уиграности играча.

1. УВОД

Тема овог дипломског рада представља развијање дводимензионалне мобилне игрице користећи ***LibGDX*** фрејморк. Сам фрејмворк је направљен у Јава програмском језику и користиће се помоћу програма **Android studio,** који се примарно користи за развијање видео игара за рачунаре као и за мобилне уређаје.

Велики проблем у данашње време, код развоја игрица генерално, представља који фрејмворк је најбољи и најпоузданији? На то питање свако има другачији одговор јер сваки фрејмворк за себе има своје предности и мане. Самим тим диволопер треба сам да процени у зависности на каквом пројекту ради који фрејмворк ће му обезбедити све потребене функционалности. Тако је и сам аутор овог рада одлучио да најбољи почетак за даље развијање у области мобилних игара лежи баш у овом фрејмворку.

LibGDX фрејмворк није први избор компанија које се баве развијањем видео игара, али је веома практичан и користан за некога ко тек улази у овај део софтверске индустрије. Поменут фрејмворк омогућава дивелоперу да развија, дебагује и тестира апликације на сопственом рачунару, док истовремено тај исти код може да користи на андроид уређајима. Основни циљ је да се успостави потпуна компатибилност између рачунара и мобилних уређаја.

Генерално, овакав пројекат је права ствар за студенте који хоће даље да напредују у правцу развијања мобилних игрица. Фокус није само на томе да се изучи неки фрејмворк и неки специфичан програмски језик, него и да се примене основни принципи програмирања са којима ћемо се сусрести у будућности на својим првим пословима.

Рад је подељен на 6 нумерисаних поглавља, након којих је дат приказ коришћене литературе.

У другом поглављу је детаљно описан фрејмворк који је коришћен за идраду рада као и неки основни принципи програмирања који су коришћени приликом израде истог.

У трећем поглављу је описан циљ датог пројекта као и основна механика игре.

У четвртом поглављу је прво објашњена аритектура пројекта, а затим детаљно описана његова имплеметација.

У петом поглављу је описано како систем финкционише у реалном времену.

У шестом поглављу је представљен закључак као и рекапитулација читавог рада. Такође је предложено неколико могућих надоградњи овог система у некој даљој будућности.

2. ОПИС ТЕХНОЛОГИЈЕ И ОСНОВНИХ ПРИНЦИПА

2.1 LibGDX ФРЕЈМВОРК И МОДУЛИ

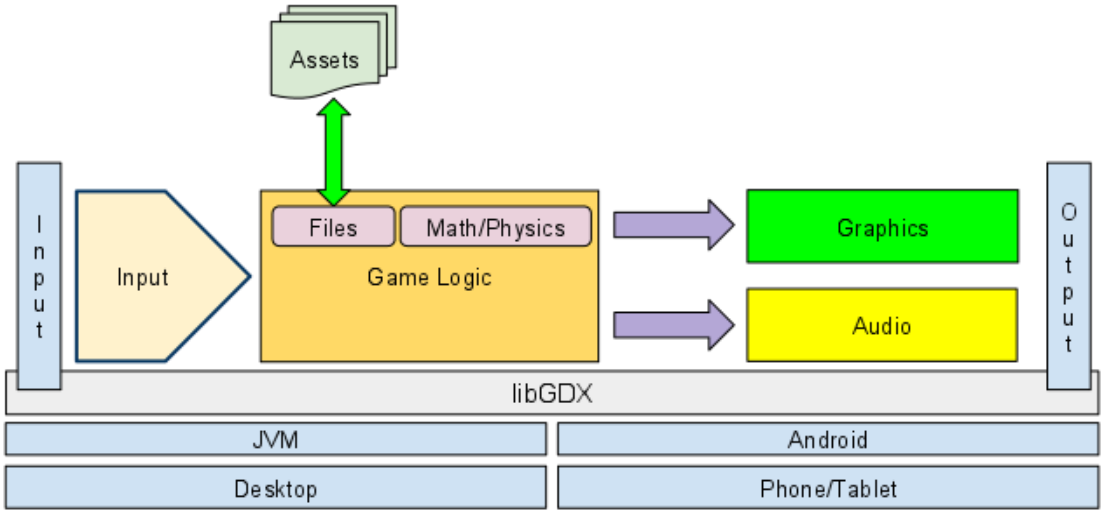
Најбитнија ствар на почетку је да објаснимо како сам фрејмворк функционише, које су његове предности и мане, као и да се детаљно упознамо са његовом архитектуром. Касније ћемо причати о неким основним принципима објектно орјентисаног програмирања који су кључ за издару и функционисање једног реалног и квалитетног пројекта.

LibGDX је бесплатан фрејмворк који може било ко да користи, такође је и *open-sorce* што значи да је код у коме је писан фрејмворк доступан свим дивелоперима да га користе и модификују.

Фрејмворк транспарентно користи код специфичан за платформу кроз различите бекенде да би приступио могућностима платформе кроз коју се користи. Већину времена дивелопер не мора да пише никакав код специфичан за платформу, осим приликом прављења почетних класа које захтевају другачија подешавања у зависности од бекенда. Већ је поменуто да је фрејмворк написан у *Java* програмском језику, али компајлирани бајткод је језички-независан.

У својој основи, ***LibGDX*** се састоји од 6 модула у форми интерфејса који пружају средства за интеракцију са оперативним системом:

1. Апликациони - покреће апликацију и информише апи клијент о свим догађајима на нивоу апликације.
2. Модул фајлова – излаже основни фајл ситем платформе.
3. Улазни модул – информише апи клијент о улазима корисника као што су миш, тстатура итд.
4. Интернет модул – пружа средства за приступ ресурсима преко HTTP/HTTPSa.
5. Аудио модул – омогућава средства за пуштање звучних ефеката као и за директан приступ аудио уређајима.
6. Графички модул – омогућава испитивање и подешавање видео режима.

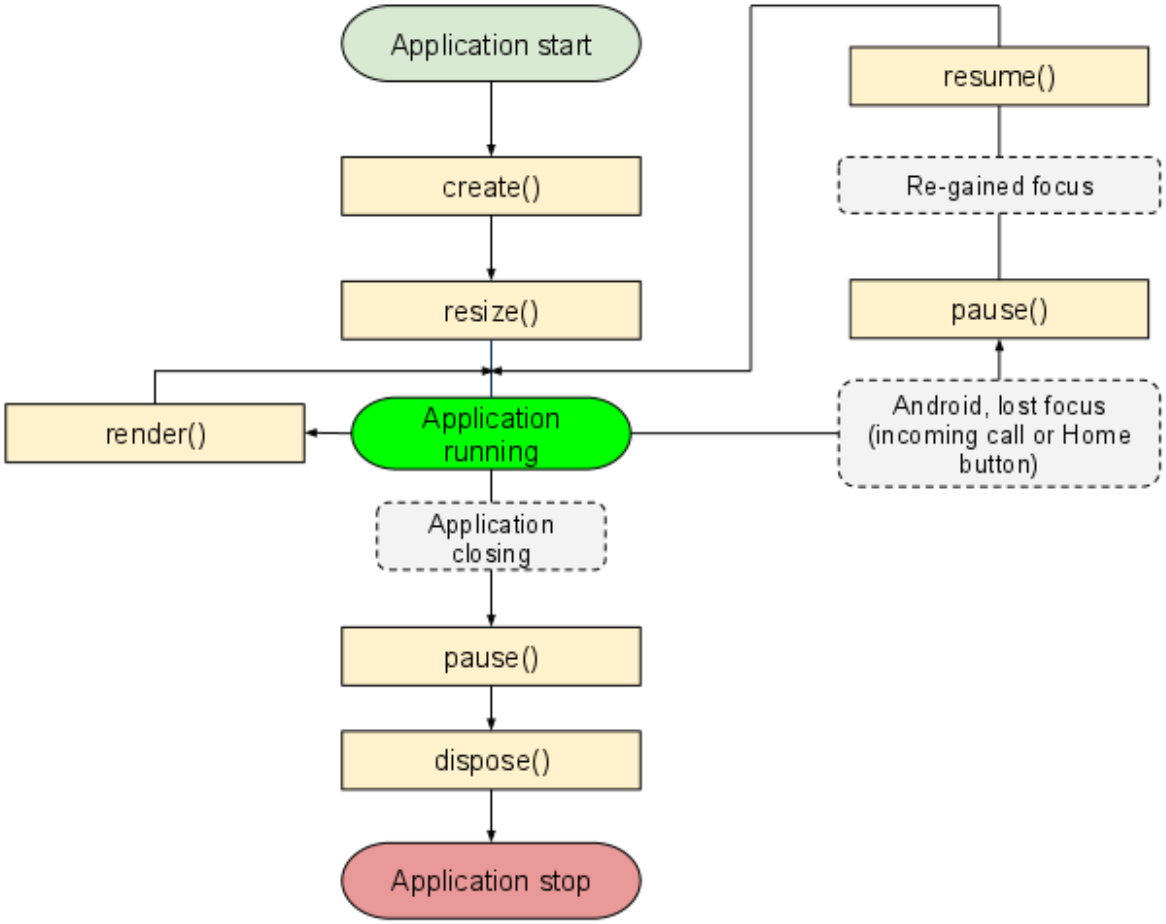


Слика 1 - Архитектура фрејмворка(Преузето из литературе бр. 4)

Поменутим модулима може се приступити преко статичких поља *GDX* класе. Ово представља скуп глобалних варијабли које омогућавају лак приступ било ком модулу. Овкав принцип генерално се сматра за веома лошу праксу писања кода, али његова сврха је да се избегну компликације које настају услед слања референци стварима које се често користе у коду.

2.2 ЖИВОТНИ ЦИКЛУС АПЛИКАЦИЈЕ

Поред модула веома битна ствар датог фрејмворка је животни циклус апликације. Он управља стањима апликације, као што је креирање, паузирање и наставак, приказивање и одлагање апликације. Програмер се повезује са овим догађајима животног циклуса тако што имплементира *ApplicationListener* интерфејс и прослеђивањем инстанце те имплементације до апликационе имплементације одређеног позадинског дела. Одатле апликација зове *ApplicationListener* сваки пут када се догоди неки догађај на нивоу апликације.



Слика 2 - Животни циклус апликације(Преузето из литературе бр. 4)

Све методе *ApplicationListenera* се позивају на истој нити. Ова нит је нит за рендеровање на коју се могу упутити *OpenGL* позиви. Било која графичка операција која директно укључује *OpenGL* мора бити извршена на нити за рендеровање. Ако се изврши на било којој другој нити, резултираће недефинисаним понашањем. Ово је због тога што је *OpenGL* контекст активан само у нити за рендеровање. Постављање контекста актуелним на другим нитима има проблема на многим андроид уређајима, стога није подржано. Ниједна класа у ***LibGDX***-у није безбедна за нит ако није експлицитно означена као таква у документацији класе. Посебно, никада не би требало да се изводе операције са више нити на било чему што је повезано са графиком или звуком.

2.3 УПРАВЉАЊЕ ДАТОТЕКАМА

Једна веома битна ствар коју треба споменути је управљање датотекама. Апликације већ поменутог фрејмворка се могу покренути на 4 различите платформе и то су: десктоп системи, андриоид, iOS и JavaScript/WebGL подржани веб браузери. Свака од ових платформи управља фајловима на другачији начин. **LibGDX-**ов модул датотека пружа заједнички интерфејс за све поменуте платформе са могућношћу да читају, пишу, копирају, премештају, бришу и излиставају фајлове.

Датотека у ***LibGDX-***у је представљена инстанцом *FIleHandle* класе. Ова класа има свој тип који дефинише где је датотека лоцирана. На датој слици је приказана доступност и локација свих типова датотека за сваку платформу.

Type	Description, file path and features	Desktop	Android	HTML5	iOS
Classpath	Classpath files are directly stored in your source folders. These get packaged with your jars and are always *read-only*. They have their purpose, but should be avoided if possible.	Yes	Yes	No	Yes
Internal	Internal files are relative to the application's *root* or *working* directory on desktops, relative to the *assets* directory on Android, and relative to the `core/assets/` directory of your GWT project. These files are *read-only*. If a file can't be found on the internal storage, the file module falls back to searching the file on the classpath. This is necessary if one uses the asset folder linking mechanism of Eclipse, see Project Setup. Relative paths (`./` or `../`) are not always supported and thus shouldn't be used.	Yes	Yes	Yes	Yes
Local	Local files are stored relative to the application's *root* or *working* directory on desktops and relative to the internal (private) storage of the application on Android. Note that Local and internal are mostly the same on the desktop.	Yes	Yes	No	Yes
External	External files paths are relative to the home directory of the current user on desktop systems. On Android, the app-specific external storage is used.	Yes	Yes	No	Yes
Absolute	Absolute files need to have their fully qualified paths specified. *Note*: For the sake of portability, this option must be used only when absolutely necessary	Yes	Yes	No	Yes

Слика 3 - Приказ типова датотека(Преузето из литературе бр. 4)

Absolute и *classpath* датотеке се углавном користе за алате као што су десктоп едитори, који имају доста комплексе улазно/излазне захтеве за датотеке.

Internal датотеке су сва средства(слике, аудио фајлови итд.) која су упакована у нашу апликацију.

Local датотеке су су веома мале датотеке које се користе за чувања стања игре. Она су генерално приватна за нашу апликацију.

External датотке су веома велике датотеке које садрже скриншотове или неке скинуте датотеке са интернета. Екстерна меморија је несабилна, што значи да корисник може да обрише написане датотеке.

2.4 УПРАВЉАЊЕ УНОСИМА

Поред управљања дататотекама потребно је и споменути управљање уносима. Различите платформе имају различите улазне могућности. На десктопу корисници могу да разговарају да апликацијом користећи тастатуру и миш, док идентично важи за игре које су засноване на веб претраживачу. На андроиду миш и тастатура су замењени екраном осетљивим на додир.

LibGDX апстрахује све ове различите улазне уређаје. Миш и екран осетљив на додир се третирају као исти са мишевима који немају мулти-додирну подршку и екранима

који немају подршку везану за дугме. У зависности од улазног уређаја, може се периодично испитивати стање уређаја или регистровати слушаоца који ће примати улазне догађаје хронолошким редоследом. Први је довољан за многе аркадне игре, нпр. аналогне контроле штапића. Ово последње је неопходно ако су укључени елементи корисничког интерфејса као што су дугмад, јер се они ослањају на секвенце догађаја као што су додиривање. Свим улазним објектима се приступа преко *input* модула.

2.5 ИНТЕРНАЦИОНАЛИЗАЦИЈА И ЛОКАЛИЗАЦИЈА

Технички гледано, интернационализација је процес дизајнирања софтвера тако да се потенцијално може прилагодити различитим језицима и регионима без инжењерских промена. Локализација је процес прилагођавања интернационализованог софтвера за одређени регион или језик додавањем компоненти специфичних за локализацију и превођењем текста. Због дужине термина, и интернационализација и локализација се често скраћују на i18n односно l10n, где 18 и 10 означавају број слова између почетног и крајњег слова одговарајућих речи.

У ***libGDX*-у**, класа *I18Bundle* се користи за складиштење и преузимање стрингова који су осетљиви на локализацију. Тај пакет конкретно омогућава да се лако обезбеди различит превод за апликацију.

2.6 УПРАВЉАЊЕ МЕМОРИЈОМ

Игре су апликације са великим ресурсима. Слике и звучни ефекти могу заузети знатну количину РАМ-а. Такође, већином ових ресурса не управља *Java garbage collector*. Уместо тога, њима управљају локални драјвери.

Постоји више класа у ***LibGDX*-у** које представљају такве ресурсе. Сви они имплементирају заједнички интерфејс за једнократну употребу који указује да се инстанце ове класе морају ручно одлагати на крају животног века. Неуспех у одлагању ресурса ће довести до озбиљног цурења меморије!

Ресурсе треба одложити чим више нису потребни, ослобађајући меморију повезану са њима. Приступ одбаченом ресурсу ће довести до недефинисаних грешака, па се треба побринути да се избришу све референце које постоје према одложеном ресурсу.

Када се треба одлучити да ли одређена класа треба да се уклони, прво се провери да ли има *dispose()* методу која је имплементирана из *Disposable* интерфејса. Ако јесте, онда је безбедно радити са изворним ресурсом**.**

Највећа примена свега до сада поменутог о управљању меморијом лежи у *Object pooling* дизајн патерну који је детаљније описан у поглављу 4.2.4.

2.7 MATH UTILITIES

Математички пакет садржи различите корисне класе за решавање проблема у геометрији, линеарној алгебри, детекцији колизија, интерполацији и уобичајеним конверзијама јединица.

Класа *MathUtils* покрива бројне корисне делове. Постоји статички насумични члан који служи да би се избегло инстанцирање једног у сопственом коду. Коришћење исте насумичне инстанце у целом коду такође може да обезбеди доследно детерминистичко понашање све док се чува коришћена почетна вредност. Постоје константе за конверзију између радијана и степени, као и табеле за тражење синусних и косинусних функција. Постоје и *float* верзије уобичајених функција *java.lang.Math* како би се избегло кастовање *double* променљиве.

2.8 КООРДИНАТНИ СИСТЕМ

Када се ради са ***LibGDX*** (или било којим другим системом заснованим на *OpenGL-y*), постоји велика шанса да се бавите различитим координатним системима. То је зато што *OpenGL* апстрахује јединице зависне од уређаја, чинећи га погоднијим за циљање више уређаја и фокусирање на логику игре. Понекад ће можда морати да се изврши конверзија између координатних система за које ***LibGDX*** нуди различите методе.

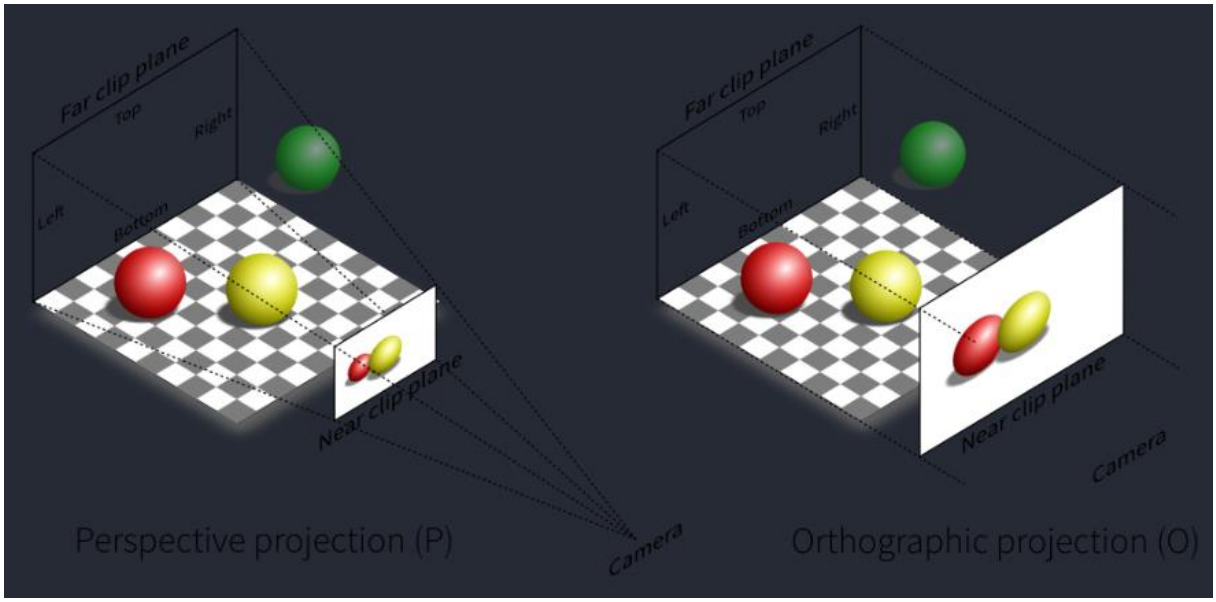
Кључно је да се разуме у ком координатном систему ради апликација. У супротном, могу се лако направити претпоставке које нису тачне.

Постоји неколико кординатних система, и то су:

- Touch coordinates
- Screen or image coordinates
- Pixmap and texture coordinates
- Normalized render coordinates
- Normalized texture (UV) coordinates
- World coordinates
- GUI/HUD coordinates
- Game coordinates

Најбитније ствар коју је потребно објаснити кориднатни ситем света(world cordinates). Кординате света су конвертоване, у *vertex shader-y*, у нормализоване кординате за рендеровање. *Vertex shader* представља графичке функције процесовања које се користе да би се додали специјални ефекти неком објекту. Камера или оквир се користе за дефинисање стратегије како то учинити. На пример, да би се одржала размера, могу се додати црне траке. Камера се користи за израчунавање матрице приказа, која преводи кординате света у координате у односу на камеру, узимајући у обзир локацију и

ротацију камере. Такође израчунава матрицу пројекције, која конвертује кординате света у нормализоване координате рендеровања у опсегу [-1,-1] до [+1,+1]. У 2Д играма углавном није потребна разлика између ове две матрице, већ је потребна само комбинована матрица трансформације. Може постојати више камера или оквира за приказ, а такође може имати више светских координатних система. Типична игра има најмање два од њих, и то: *GUI/HUD coordinates* и *Game coordinate*.



Слика 4 - Разлика у пројекцијама(Преузето из литературе бр. 4)

2.9 ИГРИЦЕ ЗА ДЕЦУ

У данашње време свако мало дете има приступ мобилним уређајима својих родитеља, тако да је све више нових игрица намењених баш за најмлађе. Аутор овог дипломског рада такође има малу сестру, која је у једну руку такође била инспирација за израду овог рада. Неке од дечијих омиљених игрица које су такође прављене помоћу ***LibGDX***-а су:

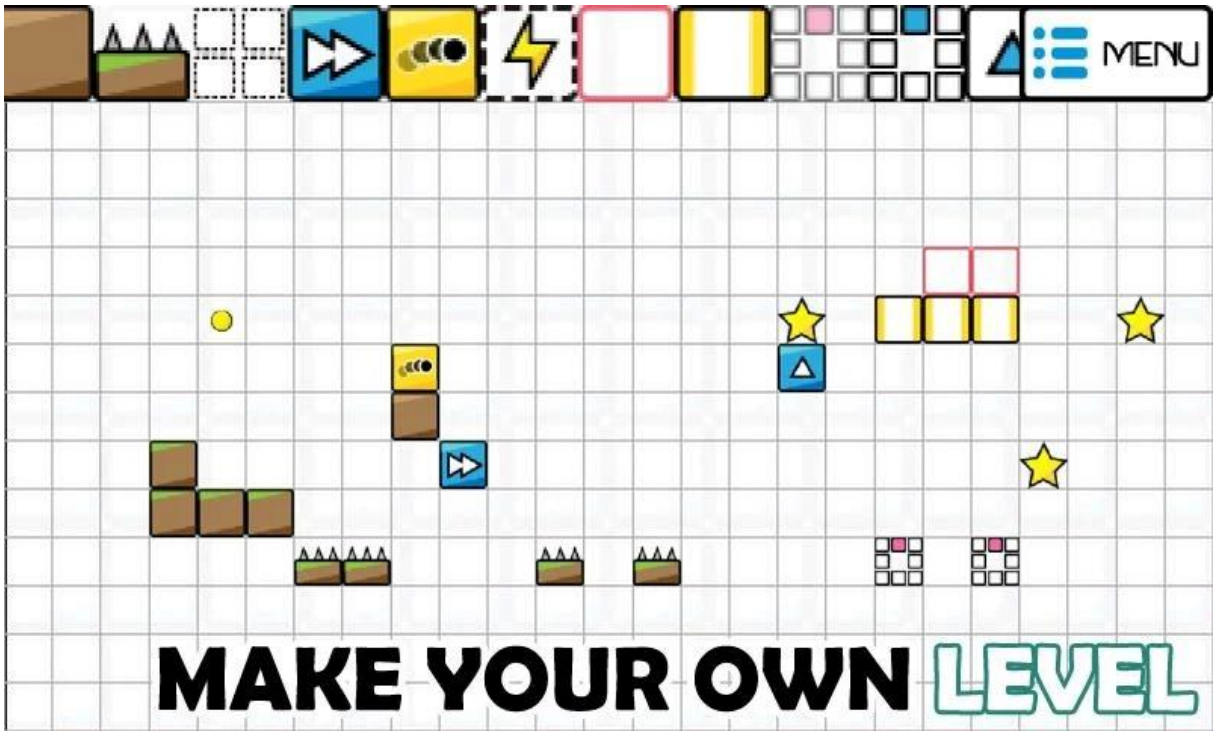
1. Stone Pillar

Слика 5- Приказ игрице Stone Pillar

Слика 6 - Приказ игрице Stone Pillar

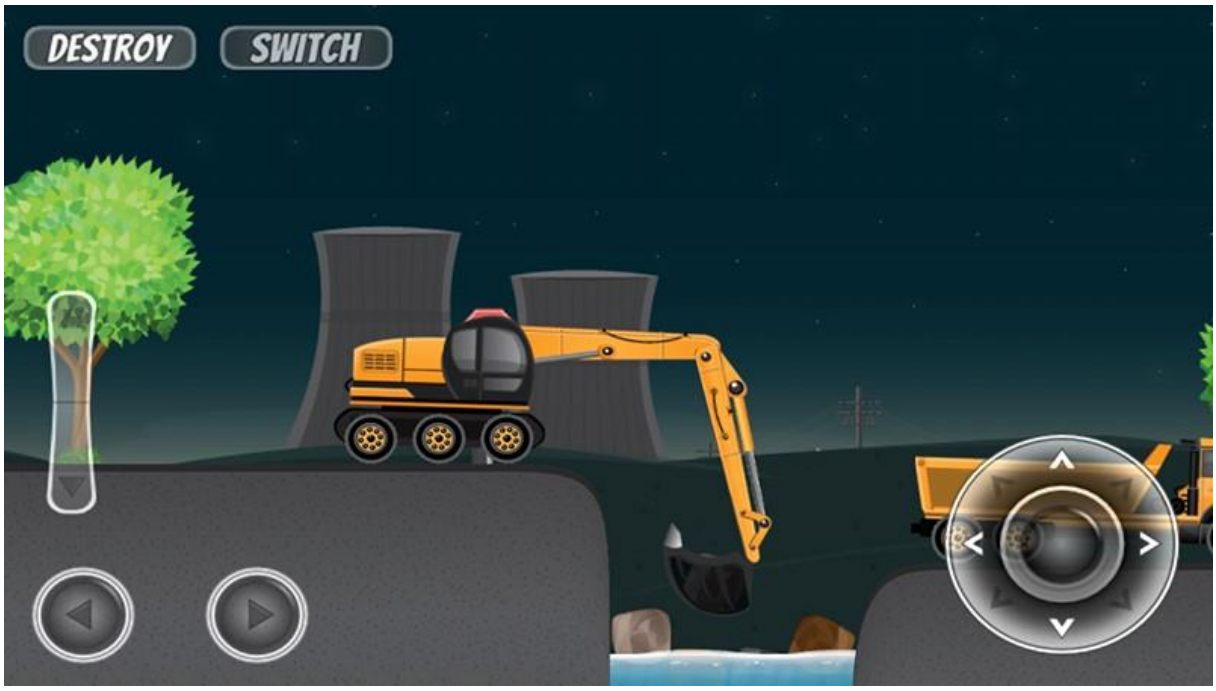
2. Bouncy Ball

Слика 7 - Приказ игрице Bouncy Ball

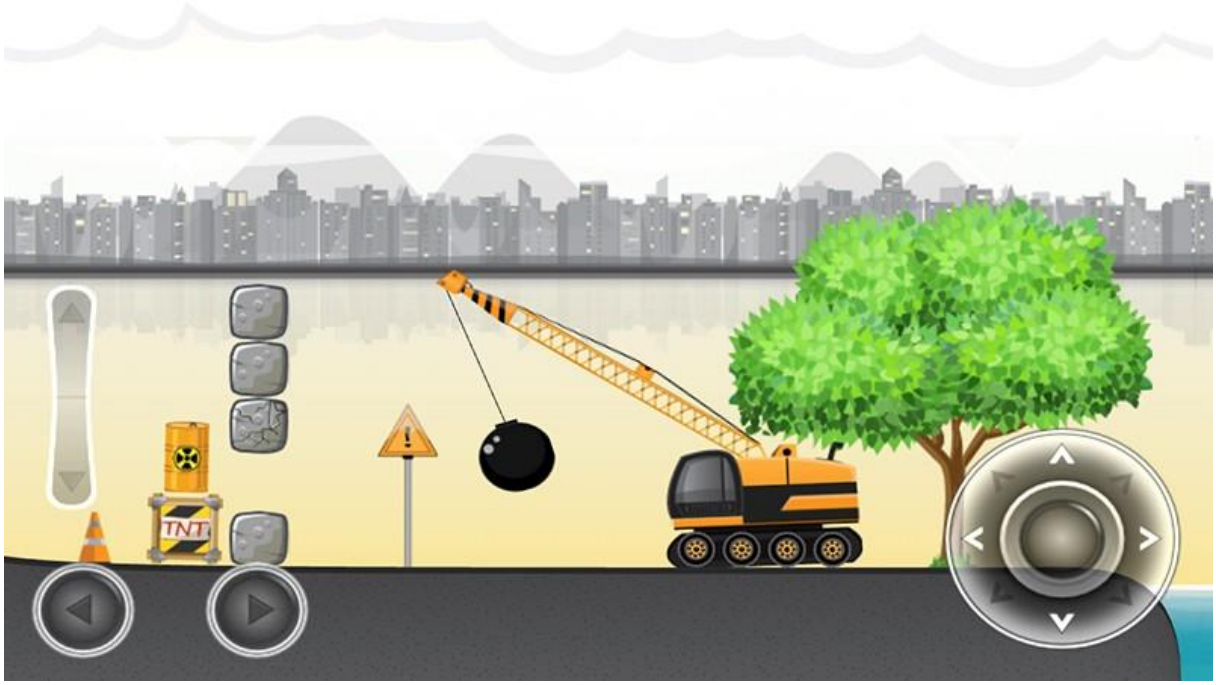


Слика 8 - Приказ игрице Bouncy Ball

3. Counstruction City



Слика 9 - Приказ игрице Construction City



Слика 10 - Приказ игрице Construction City

3. ОПИС ПРОБЛЕМА

Пробелм који се решава у овом дипломском раду је креирање креирање игрча и објеката кој се требају избегавати у дводимензионалном кординатном систему са фокусом на коришћењу што је мање меморије протебно за производњу објеката, као и на оптимизацију кода и слика које се требају рендеровати. Поред механике саме игре потребно је направити мени у којем ће играч имати понуђено неколико опција везаних за саму игру.

3.1 ОПИС ИГРЕ

3.1.1 Типови објеката

У игри постоје 2 типа објеката и то су актер и препреке. Оба типа су су кругови у кординатном систему са сликама Сунђер Боба и Лигњослава.

3.1.2 Циљ и механика игре

Циљ игре је да галвни актер који је представљен као Сунђер Боб избегава објекте који падају са горњег дела екрана у облику Лигњослава. Избегавање се врши тако што играч прстом на телефону помера карактера у свим правцима, док препреке падају линерано по Y оси. Препреке се потупоно случајно формирају било где на екрају, чинећи игру веома непредвидивом. Циљ је да играч направи што већи скор тако што ће што дуже остати жив избегавајући препреке.

3.1.3 Избор тежине

У почетном менију, пре старта игре кориснику је понуђено дугме за промену тежине игре. Кликом на дугме на екрану појављују се 3 опције које су: лак, средњи и тежак. Кориским може у било ком трениутку да напусти игру и промени ниво тежине ако му неодговара.

4. РЕАЛИЗАЦИЈА ИГРЕ

4.1 ПРЕГЛЕД АРХИТЕКТУРЕ

Захваљујући ***LibGDX*** фрејмворку могуће је креирати више екрана које можемо да користимо унутар наше игре. Такође поменути фрејмворк нам омогуће да користимо небројано сцена, тј. мини апликација које су доступне, да би имали приступ различитим функционалностима као жто су на пример *shapeRenderer* и *viewport*, о којима ћемо причати касније. Корен наше архитектуре почиње из класе *Game* из које се даље позивају све остале неопходе класе. Игрица је подељена на 5 основних пакета и то су:

1. assets - комуницира и прикупља средства као што су слике и формати.
2. config - конфигутација основних објеката и ипроменљивих који су неопходи за фунционисање целе игрице.
3. entity – обухвата класе свих објеката са којима игрица располаже
4. util – служи као дибагер, односно за исцртавање кординатног ситема и линја које нам служе за лакши увид прликом креиранња објеката игре.
5. screen – у њему су садржане класе свих екрана које угрица поседује као и најбитнији под пакет *game* у којем су најбитније класе везане за креиранје и рендеровање објеката.

Сви ови пакети спадају унутар *core* пакета. Игрица поседује још *android* и *desktop* пакете. У првом се налазе сва наша средства са којима комуницира *assets* пакет, док у другом се налазе класа за покретање игрице као и класа за пакованње свих средстава у неки атлас о коме ће бити речи касније.

4.2 ИМПЛЕМЕНТАЦИЈА

Иглавни изазов имплементације ове игре био је не само наравити игрицу која је функционална, него и да цео код буде у складу са паринципима објектно орјентисаног програмирања или такозваним **SOLID** принципима. Поред наведених принципа коришћени су и 2 дизајн патерна под именом **Singleton** и **Object pool** патерн о којима ће бити више речи касније. Још један изазов је био направити непробојни код, што значи

да је систем потпуно енкапсулиран, не захтева интернет везу и може да ради у затвореним просторима или поверљивим окружењима. Такође то представља код који хвата или открива грешке у уносу, пријављује грешку кориснику и „грациозно“ се носи са грешком. Грациозно решавање грешке обично значи да након откривања грешке и штампања дијагностике, програм или прекида или дозвољава кориснику да поново унесе податке.

4.2.1 Рад са средствима

Средства или *assets* представљу спољне датотеке које корисе у датој апликацији, конкретно у нашем случају су фонтови за слова и бројеве, као и слике објеката, позадине, дугмади итд. Све слике које користимо у пројекту можемо на класичан начин убацивати једну по јенду али то би одузимало много процесорског вермена. Разлог томе је да игрица сваки пут када хоће да рендерује неки објекат, *batch* о коме ћемо причати касније, мора да шаље геометрије графичкој карти која затим исцртава. Увек би се учитавао објекат по објекат из посебних фајлова, што одузима доста времена. Зато смо дошли до решења да све слике које су потребне спојимо у једну слику која се зове атлас. Предност атласа је то што када желимо да исцртамо неки објекат, увек учитавамо један те исти фајл(атлас) и само из тог фајла узимамо делове слике које су нам потребни. Фрејмворк то у позадини обавља слањем кордината одређеног сегмента у атласу. У пакету средстава налазе се две главне класе о којима ћемо причати појединачно.

AssetPaths је класа која у облику глобалних променљивих садржи релативне путање до наших атласа и фонтова које користимо.

```java
package com.igrica.assets;

public class AssetPaths {

    public static final String UI_FONT = "ui/fonts/ui_font_32.fnt";

    public static final String GAME_PLAY = "gameplay/gameplay.atlas";

    public static final String UI = "ui/ui.atlas";

    public static final String UI_SKIN = "ui/uiskin.json";

    private AssetPaths() {}
}
```

Слика 11 Дефинисање путањи до атласа

AssetDescriptors је заправо конфигурациона класа која регулише које се средство одакле учитава и ког је типа.

```java
package com.igrica.assets;

import ...

public class AssetDescriptors {

    public static final AssetDescriptor<BitmapFont> FONT = new AssetDescriptor<~>(AssetPaths.UI_FONT, BitmapFont.class);

    public static final AssetDescriptor<TextureAtlas> GAME_PLAY =
            new AssetDescriptor<~>(AssetPaths.GAME_PLAY, TextureAtlas.class);

    public static final AssetDescriptor<TextureAtlas> UI =
            new AssetDescriptor<~>(AssetPaths.UI, TextureAtlas.class);

    public static final AssetDescriptor<Skin> UI_SKIN =
            new AssetDescriptor<~>(AssetPaths.UI_SKIN, Skin.class);

    private AssetDescriptors() {}
}
```

Слика 12 Дефинисање AssetDescriptora

4.2.2 Дефинисање конфигурације

Једна од најбитнијих класа је *GameConfig* која се налази у пакету *config* и она нам служи за декларисање свих глобалних променљивих које су неопхонде да за функционисање целе игрице. То су оцновни подаци као нпр: висине и ширени екрана за приказ, као и света у којем се одвија игрица, брзина препрека, број живота итд. То се може видети на следећој слици.

```
package com.igrica.config;

public class GameConfig {

    public static final float WIDTH = 480f;
    public static final float HEIGHT = 800f;

    public static final float HUD_WIDHT = 480f;
    public static final float HUD_HEIGHT = 800f;

    public static final float WORLD_WIDTH = 6.0f;
    public static final float WORLD_HEIGHT = 10.0f;

    public static final float WORLD_CENTER_X = WORLD_WIDTH / 2f;
    public static final float WORLD_CENTER_Y = WORLD_HEIGHT / 2f;

    public static final float MAX_PLAYER_X_SPEED = 0.25f;
    public static final float OBSTACLE_SPAWN_TIME = 0.25f;
    public static final float SCORE_MAX_TIME = 1.25f;

    public static final int LIVES_START = 3;

    public static final float EASY_OBSTACLE_SPEED = 0.1f;
    public static final float MEDIUM_OBSTACLE_SPEED = 0.14f;
    public static final float HARD_OBSTACLE_SPEED = 0.18f;

    public static final float PLAYER_BOUNDS_RADIUS = 0.4f; //jedinice igrice
    public static final float PLAYER_SIZE = 2 * PLAYER_BOUNDS_RADIUS;

    public static final float OBSTACLE_BOUNDS_RADIUS = 0.3f; //jedinice igrice
    public static final float OBSTACLE_SIZE = 2 * OBSTACLE_BOUNDS_RADIUS;
```

Слика 13 Дефинисање конфигурационих променљивих

4.2.3 Пакет објеката

У нашем систему постоје 2 главна објекта и то су играч и препрека, док још један споредан објекат који смо убацили је позадина игрица. Разлог зашто је позадина објекат сам за себе лежи у томе што смо је подешавали по кординатам различитим од *default* кордината. Тиме смо је издвојили у посебан објекат тако да увек можемо да извучемо њене кординате.

Друга два објекта, играч и препрека, наслеђују абстракну класу *GameObjectBase*, која садржи све заједничке пропертије објеката, да се исти код не би понављао више пута. То се назива *dry principle*. У њих спадају кординате, висина и ширина, гетери и сетери поља и још неколико потребних метода.

Класа играч само наслеђује дату класу са свим њеним методама док класа препрека има једну своју допунску методу која проверава да ли је објекат препреке дотакао објекат играча. Она заправо сваки пут кад објекат препреке буде дотакнут прслеђује *boolean* вредност која се даље проверава и у зависности од њене вредности играч губи живот.

```
public boolean isPlayerColliding(Player player) {
    Circle playerBounds = player.getBounds();

    //intersektor je bibliotek sa metodama za proveravanje da li
    //se dva presecaju
    boolean overlaps =  Intersector.overlaps(playerBounds, this.getBounds());

    //kao if gde je hit = true
    hit = overlaps;
    return overlaps;
}
```

Слика 14 Провера додира овјеката

Као што видимо са слике интерсектор је библиотека која узима кординате оквира два објекта и проверава да ли је дошло до пресека.

4.2.4 Контролер

У нашем пројекту контролер је једна од две најбитније класе за функионисање целог пројекта. У њему се врши креација свих објеката, брисање и ажурирање истих, као и још много других.

Једана од веома битних ствари коју смо применили у овом пројекту је *object pool design patern* који је кључна ствар за чување меморије. Циљ овог патерна је да се не прави бесконачно много објеката препрека које играч избегава, што би пунило меморију до краја. Главна ствар је да се користи базен(pool) који ће имати неки коначно мали капацитет и сваки пут када се базен напун објектима најстарије креиран објекат се ослобађа из базена и поново се убацује у употребу. Тиме имамо коначно мало заузете меморије која се попуњава производњом објеката. У нашој игрици објекти заузимају свакако превише мало меморије тако да овај патерн не прави велику разлику, али је много битна ствар када се ради на много већим пројектима.

Прво морамо креирати базен коме постављамо 2 параметра, коју класу објеката прима и које је величине.

```
private Pool<Obstacle> obstaclePool;
```

```
obstaclePool = Pools.get(Obstacle.class, max: 40);
```

Слика 15 Дефинисање базена

Галвна метода у овој класи је *update* метода која позива 4 под методе које ажурирају или креирају нове овјекте, скор итд.

```
public void update(float delta){
    if(isGameOver()){
        log.debug("Game Over");
        return;
    }

    updatePlayer();
    updateObstacles(delta);
    updateScore(delta);
    updateDisplayScore(delta);

    if(isPlayerCollidingWithObstacle()){
        log.debug("Collision detected");
        lives--;

        if(isGameOver()){
            log.debug("GAME OVER!!!");
            GameManager.instance.updateHighScore(score);
        }else{
            restart();
        }
    }
}
```

Слика 16 Update() метода

Позивом методе *updateObstacle()* проверава се да ли постоји већ неки креиран овјекат. Ако има он се ажурира, ал ако нема креирарају се нови и на крају се купе објекти који су испали из света игрице и стављају се у базен.

```
private void updateObstacles(float delta){

    for(Obstacle obstacle : obstacles){
        obstacle.update();
    }

    createNewObstacle(delta);

    removePassedObstacles();

}
```

Слика 17 Ажурирање препрека

У креирању препрека 2 битне ствари су минимум и максимум, које представљају вреднос на X оси где ће се појавити препрека.

```
private void createNewObstacle(float delta){
    obstacleTimer += delta;

    if(obstacleTimer > GameConfig.OBSTACLE_SPAWN_TIME){
        float min = 0;
        float max = GameConfig.WORLD_WIDTH - GameConfig.OBSTACLE_SIZE ;

        float obstacleX = MathUtils.random(min, max);
        float obstacleY = GameConfig.WORLD_HEIGHT;

        Obstacle obstacle = obstaclePool.obtain();
        DifficultyLevel difficultyLevel = GameManager.instance.getDifficultyLevel();
        obstacle.setYSpeed(difficultyLevel.getObstacleSpreed());
        obstacle.setPositions(obstacleX, obstacleY);

        obstacles.add(obstacle);
        obstacleTimer = 0f;
    }
}
```

Слика 18 Креирање препрека

Такође у методи која купи објекте који испадну проверава вредност на Y оси.

```
private void removePassedObstacles(){
    if(obstacles.size > 0){
        Obstacle first = obstacles.first();

        float minObstacleY = -GameConfig.OBSTACLE_SIZE;

        if(first.getY() < minObstacleY){
            obstacles.removeValue(first, identity: true);
            obstaclePool.free(first);
        }
    }
}
```

Слика 19 Елиминисање објеката који испадну

4.2.5 Класа за рендеровање

Као што смо већ поменули класа за рендеровање је друга најбитнија ствар овог пројекта, али пре него што кренемо у објашњење њене функционалности, треба објаснити неке појмове.

ViewPort је објекат екрана, тј. оквира на моитору где ће се игрица приказивати, такође он конторлише наш објекат камере, који ће бити детаљно објашњен после. Наша игрица користи више различитих оквира других димензија за исцртавање појединачних компоненти. Оквир позадине и оквир екрана нису исти, него су подешеи по различитим димензијама али се комбинују и налепљују један на други, чији је ефекат приказан у игрици. Битна ствар је да у зависности од тога коју компоненту рендерујемо, треба да

користимо њен оквир који увек морамо да сетујемо функцијом apply(), да би рендерер знао тачно на ком оквиру шта и како приказује.

SpriteBatch је задужен за прикупљање кордината тј. геометрија сваког објекта који се исцртава. Битна ствар коју треба нагласити је да он не исцртава објекте из центра него креће од тачке (0,0) док рендерер ради то из центра објекта.

Renderer има улогу сличну као *spriteBatch* само што рендерер користимо више за исцртавање објката док спрајт за исцртавање јутилитија. Главна метода ове класе је *init()* ко служи да се инстанцирају сви објекти и вредности који су му потребни као и за извлачење средстава из атласа о којима смо већ причали. Друга битна метода је *render()* која позива методе за појединачно исцртавање компонената. Свака рендер функција прво мора да постави оквир који користи функицојм *apply()* и затим да се постави пројекциона матрица на комбиновану камеру. И трећа ствар коју свака функција има је *begin()* којом се започиње цртање.

```
viewport.apply();
batch.setProjectionMatrix(camera.combined);
batch.begin();
```

Слика 20 Основне операције рендеровања

Послења ствар о коју треба објаснити је пројекциона матрица и комбинована камера. Размислите о снимању слике камером. На пример. користећи камеру свог паметног телефона сликајући клупу у парку. Када то урадите, видећете клупу у парку на екрану свог телефона.

Локација клупе на слици је у односу на место на коме сте стајали приликом снимања фотографије. Другим речима, то је релативно у односу на камеру. У типичној игри, не постављате објекат у односу на објекат. Уместо тога, ви их постављате у свој свет игре. Превођење између вашег света игре и ваше камере се врши помоћу матрице (што је једноставно математички начин за трансформацију координата). На пример. када померите камеру удесно, онда се клупа помери улево на фотографији. Ово се зове матрица погледа.

Тачна локација клупе на слици зависи и од удаљености између клупе и камере. Бар је тако у 3Д а и 2Д је веома сличан. Када је даље мањи је, када је близу камере већи. Ово се зове пројекција перспективе. Можете имати и ортографску пројекцију, у ком случају се величина објекта не мења у зависности од удаљености до камере. У сваком случају, локација и величина клупе у парку се преводи на локацију и величину у пикселима на екрану. На пример. клупа је широка два метра у парку, док је на фотографији 380 пиксела. Ово се зове пројекцијска матрица.

Camera.combined() представља комбиновану матрицу погледа и пројекције. Другим речима описује где ствари у вашем свету игре треба да буду приказане на екрану.

Позивање *batch.setProjectionMatrix(camera.combined)* упућује *batch* да користи ту комбиновану матрицу. Требало би да се позове кад год се вредност промени. Ово се обично дешава када се позива промена величине и такође кад год се помери објекат или на други начин промени камера. Ако нисте сигурни, можете то позвати на почетку вашег метода рендеровања.

Углавном у рендер методи посоји још и додатна функционалност која проверава да ли се игрица игра на тач скрину и у зависости од покрета прста она прави векторе који се конвертују у кординате на које се помера објекат.

```
public void render(float delta){

    debugCameraControler.handleDebugInput(delta);
    debugCameraControler.applyTo(camera);

    //prveravanje da li je dotaknut ekran telefon tj implementacija
    //da radi i na tac skrin
    if(Gdx.input.isTouched() && !controller.isGameOver()){
        Vector2 screenTouch = new Vector2(Gdx.input.getX(), Gdx.input.getY());
        Vector2 worldTouch = viewport.unproject(new Vector2(screenTouch));

        Player player = controller.getPlayer();
        worldTouch.x = MathUtils.clamp(worldTouch.x,  min: 0,
                max: GameConfig.WORLD_WIDTH - player.getWidth()
        );
        player.setX(worldTouch.x);
    }

    GdxUtils.clearScreen();

    renderGamePlay();

    renderUi();

    renderDebug();

}
```

Слика 21 Метода за рендеровање

4.2.6 Прављење екрана

У датом пројекту постоји 5 екрана који се користе и то су: екран за за учитавање, опције, за најбољи постигнут резултат, мени екран и екран игрие. Екран учитавања иницијализује све протребне вредноси и учитава атласе. Када то зврши он позива објекат класе *MenuScreen* која својом методом *initUi()* исцртава се што је потребно.

Први део методе прави дугмиће за одређене радње који приликом клика на њих позивају објекат класе неког другог екрана који позива своју *initUi()* методу за исцртавање.

```java
@Override
protected Actor initUi() {
    Table table = new Table();

    TextureAtlas gamePlayAtlas = assetManager.get(AssetDescriptors.GAME_PLAY);
    Skin uiskin = assetManager.get(AssetDescriptors.UI_SKIN);
    TextureRegion backgroundRegion = gamePlayAtlas.findRegion(RegionNames.BACKGROUND);
    table.setBackground(new TextureRegionDrawable(backgroundRegion));

    //---BUTTONS---
    TextButton playButton = new TextButton( text: "PLAY", uiskin);
    playButton.addListener((ChangeListener) (event, actor) -> { play(); });

    TextButton highScore = new TextButton( text: "HIGHSCORE", uiskin);
    highScore.addListener((ChangeListener) (event, actor) -> {
            showHighScore();
    });

    TextButton optionsButton = new TextButton( text: "OPTIONS", uiskin);
    optionsButton.addListener((ChangeListener) (event, actor) -> {
            showOptions();
    });

    TextButton quitButton = new TextButton( text: "QUIT", uiskin);
    quitButton.addListener((ChangeListener) (event, actor) -> { quit(); });
```

Слика 22 Рендеровање корисничког интерфејса

Други део методе прави табелу у коју додаје све дугмиће као редове, а затим прави још једну табелу чији је једини елемент претходна табела која служи за центрирање. Користе се две табеле из разлога што свака табела може да подешава пропертије своје деце.

```
//setup table
Table buttonTable = new Table(uiskin);
buttonTable.defaults().pad(20);
buttonTable.setBackground(RegionNames.PANEL);

buttonTable.add(playButton).row();
buttonTable.add(highScore).row();
buttonTable.add(optionsButton).row();
buttonTable.add(quitButton);

table.add(buttonTable);
table.center();
table.setFillParent(true);
table.pack();

return table;
```

Слика 23 Креирање табела

4.2.7 Помоћни пакет за дибаговање

Последњи пакет у нашем пројекту је *debug* чија улога је исцртавање кординатног система, кретање по њему и додатно цртање линија и објеката како би нам било лаше да подесмо оснофне функционалности динамике иггре као што је кретња објеката. Такође пакет је универзалан, што значи да ако се импортује у боло коју други пројекат радиће без проблема. Он садржи 4 класе:

1. DebugCameraConfig – дефинисање конфигуртације, тј. свих глобалних варијабли и објеката неопходих за функционисање целог пакета.
2. DebugCameraControler – чија је главна улога постављање позиција у коридантном систему и функције померања објеката по истом.
3. GdxUtils – једина улога је чишћење екрана, тј. да се постави „бело платно“ по коме ће се цртати.
4. ViewPortUtils – служи за исцртавање кординатног система.

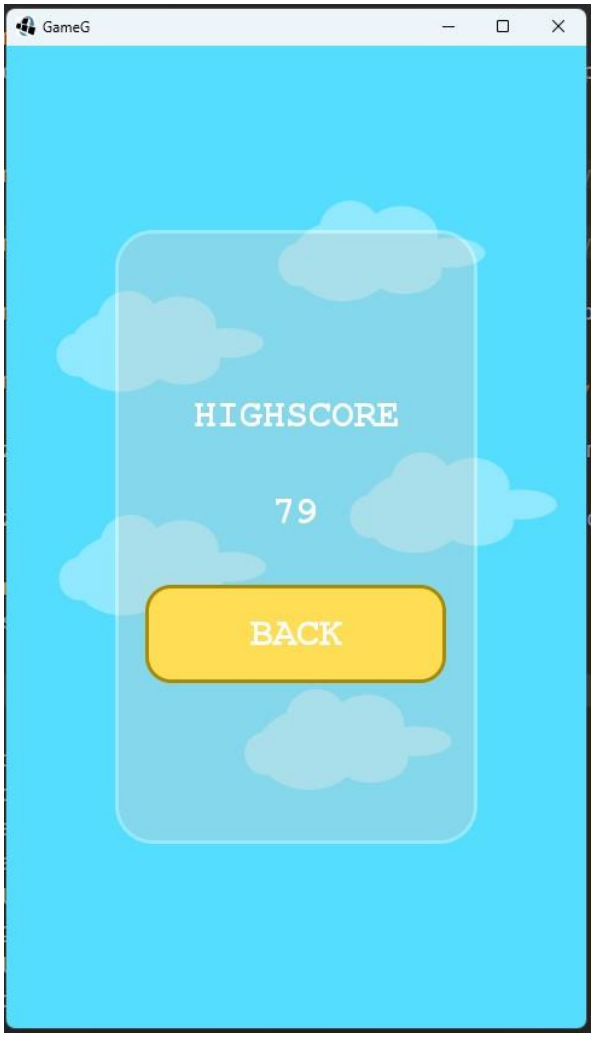
5. ОПИС РАДА СИСТЕМА

Приликом покретања игре корискину се приказује главни мени на којем постоје опција за покретање игре, да се види најбољи остварен резултат, да се подеси тежина игре и дугме за гашење.

Слика 24 Главни мени

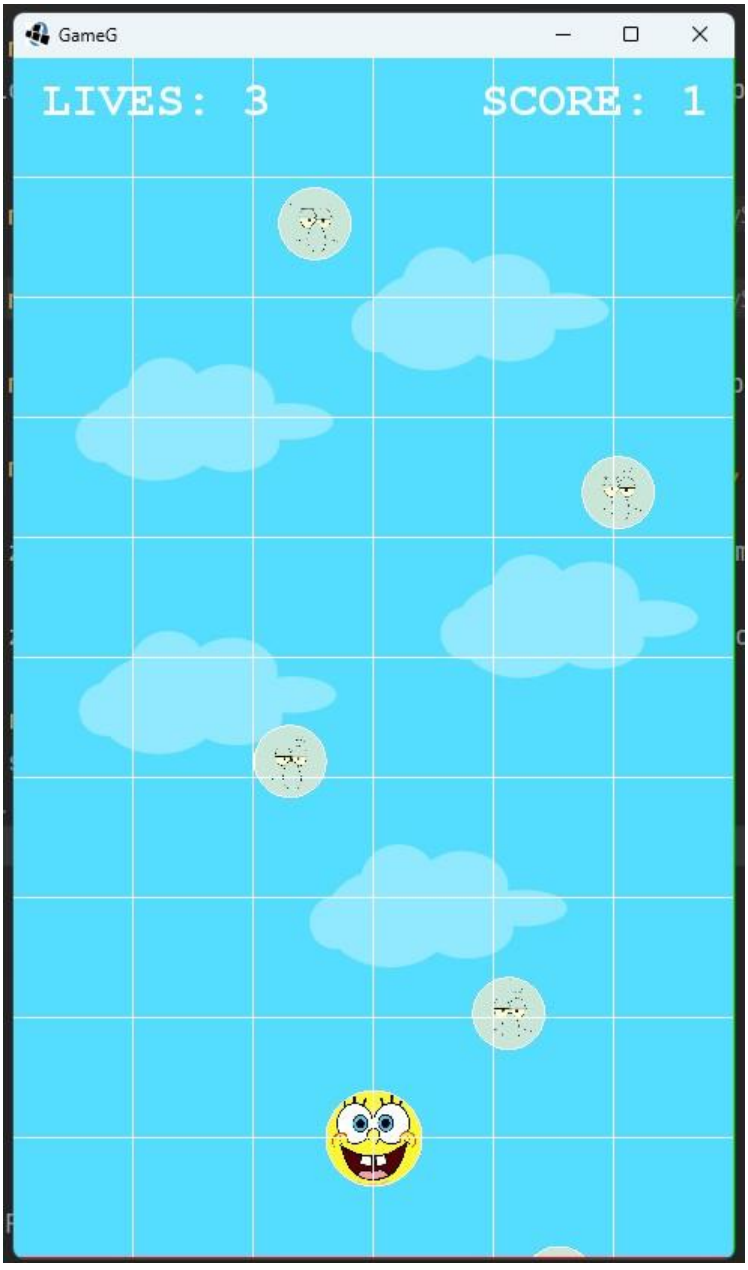
Кликом на дугме *options* отвара се мени за бирање нивоа тежине. Након бирање притиском на дугме *back* играч се враћа у почетни мени. Приликом притиска на *highscore* дугме играч може да погледа најбољи остварен резултат.

Слика - 25 Мени за мерење тежине



Слика 26 - Highscore мени

Приликом притискања дугмета *play* започиње се игра. Играч је представљен у форми сунђер Боба и циљ игре је да се избегне што више препрека које падају са неба у форми Лигњослава. Играч може да се помера у свим правцима да би избегао објекте.



Слика 27 Приказ игрице

6. ЗАКЉУЧАК

Циљ овог рада је била примена различитих принципа програмирања и дизајн патерна како би се произвела потпуно функционална апликација која минимизује коришћење меморије. Игрица је имплементирана у *Android studiu* коришћењем ***LibGDX*** фрејмворка. Један од успеха овог рада лежи у томе што је аутор имао прилику за упознавање са до сада непознатм комцептима и технологијама, који су и били један од многих мотива за избор теме овог рада.

Програмска логика игре и њених функционалности је имплементирана у овом раду у објектно орјентисаном програмском језику **Java**. Захваљујући њој, омогућено је коришћење неких познатих концепата као што су полиморфизам и енкапсулација.

Иако је игра имплементирана успешно, постоје и даље места за унапређивање. Пре свега, могће је убацити још различитих типова објеката дугих геометриских облика и величина. Такође могуће је увести неку врсту муниције која се скупља и испаљује како би се уништили објекти који имају некилико живота. Већ наведене функционалости су доступне у библиотекама фрејмворка који је коришћен за израду.

ЛИТЕРАТУРА

1. “LibGDX wikipedia” [Веб сајт] Доступно на: (https://en.wikipedia.org/wiki/LibGDX, сајт је посећен 22.07.2023. године)
2. Сајт на коме се налази документација libGDX фрејмворка [Веб сајт] Доступно на: (https://javadoc.io/doc/com.badlogicgames.gdx , сајт је посећен 23.07.2023. године)
3. “Understanding the libGDX Projection Matrix” [Веб сајт] Доступно на: (https://stackoverflow.com/questions/33703663/understanding-the-libgdx-projection-matrix , сајт је посећен 19.07.2023. године)
4. “LibGDX official website” [Веб сајт] Доступно на: (https://libgdx.com/wiki/ , сајт је посећен 29.07.2023. године)
5. “Java Tutorial for Complete Beginners” Туторијал који објашњава освовне принципе објектно орјентисаног програмирања у Java програмском језику [Веб сајт] Доступно на: (https://www.udemy.com/course/java-tutorial/ , сајт је посећен 29.07.2023. године)
6. Веб сајт на коме се налазе различити туторијали везани за различите функционалности libGDX фрејмворка [Веб сајт] Доступно на: (https://happycoding.io/tutorials/libgdx/ , сајт је посећиван у периоду од 12.06 до 15.07.2023. године)
7. “Polling computer science wikipedia” [Веб сајт] Доступно на: (https://en.wikipedia.org/wiki/Polling_(computer_science) , сајт је посећен 25.08.2023. године)
8. “Interface disposable документација интерфејса” [Веб сајт] Доступно на: (https://javadoc.io/doc/com.badlogicgames.gdx/gdx/latest/com/badlogic/gdx/utils/Disposable.html , сајт је посећен 25.08.2023. године)
9. “Class I18NBundle документација класе” [Веб сајт] Доступно на: (https://javadoc.io/doc/com.badlogicgames.gdx/gdx/latest/com/badlogic/gdx/utils/I18NBundle.html , сајт је посећен 25.08.2023. године)

10. Веб сајт на коме се налази детаљан опис архитектуре libGDX фрејмворка [Веб сајт] Доступно на: (https://2021.desosa.nl/projects/libgdx/posts/architecture/ , сајт је посећен 25.08.2023. године)
11. Веб сајт на коме се налази детаљан опис животних циклуса libGDX фрејмворка [Веб сајт] Доступно на: (https://subscription.packtpub.com/book/game-development/9781785289361/1/ch01lvl1sec12/understanding-the-application-lifecycle-of-libgdx , сајт је посећен 25.08.2023. године)
12. "Cartesian coordinate system wikipedia" [Веб сајт] Доступно на: (https://en.wikipedia.org/wiki/Cartesian_coordinate_system , сајт је посећен 25.08.2023. године)