Razvoj 2D Java igre u libGDX i Overlap2D okruženju

Šajina, Robert

Undergraduate thesis / Završni rad

2017

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Pula / Sveučilište Jurja Dobrile u Puli**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:137:805283

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

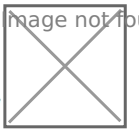
Download date / Datum preuzimanja: **2024-03-28**



Repository / Repozitorij:

Digital Repository Juraj Dobrila University of Pula





Sveučilište Jurja Dobrile u Puli

Odjel za informacijsko-komunikacijske tehnologije

ROBERT ŠAJINA

RAZVOJ 2D JAVA IGRE U LIBGDX I OVERLAP2D OKRUŽENJU

Završni rad

Pula, rujan, 2017. godine

Sveučilište Jurja Dobrile u Puli
Odjel za informacijsko-komunikacijske tehnologije

ROBERT ŠAJINA

RAZVOJ 2D JAVA IGRE U LIBGDX I OVERLAP2D OKRUŽENJU

Završni rad

JMBAG: 0303054675, redoviti student
Studijski smjer: Sveučilišni preddiplomski studij informatika
Kolegij: Napredne tehnike programiranja
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: doc.dr.sc. Tihomir Orehovački

Pula, rujan, 2017. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisan, Robert Šajina, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljen način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

Robert Šajina

U Puli, rujan, 2017. godine



IZJAVA

o korištenju autorskog djela

Ja, Robert Šajina, dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj završni rad pod nazivom „Razvoj 2D Java igre u LibGDX i Overlap2D okruženju“ koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, rujan, 2017. godine

Potpis

Robert Šajina

Sažetak

Ovaj rad opisuje koncept razvoja 2D Java igre koristeći LibGDX i Overlap2D razvojna okruženja. Igra se sastoji od dva prozora od kojih jedan prikazuje početni zaslon s opcijama kao što su odabir razina i kupnja dodatnih mogućnosti igrača dok drugi prikazuje trenutnu razinu na kojoj se igrač nalazi. Igra ima bočni pogled na igrača koji pomicanjem prema gore ili dolje izbjegava prepreke kako bi uspješno prošao razinu. U radu se opisuje rad s alatom Overlap2D te njegova integracija s LibGDX razvojnim okruženjem kako bi se razvila cjelokupna igra. Koncept realizacije koji igra slijedi jest korištenje alata Overlap2D za izradu prezentacijskog sloja odnosno korisničkog sučelja dok se LibGDX okruženje koristi za definiranje logike igre kao što su otkrivanje kolizija i pomicanje igrača čime se jasno odvaja prezentacijski dio od logičkog djela. Cilj igre je da bude korisnički orijentirana, kako bi korisniku pružila maksimalnu zabavu. Igra zato koristi jednostavno upravljanje igračem dodirom na ekran.

Ključne riječi: *Java, Overlap2D, LibGDX, 2D, kolizija, Eclipse, Box2D, Jetpack Rush, Android, igra.*

Abstract

This bachelor thesis describes concept of developing 2D Java game using LibGDX and Overlap2D frameworks. The game consists of two screens, one of which displays a startup screen with options such as level selection and additional player features purchases while the second shows the current level at which the player is located. The game has a side view of the player, who is by moving up or down avoiding obstacles to successfully pass the level. This bachelor thesis describes the development with the Overlap2D tool and its integration with the LibGDX framework to develop the overall game. The implementation concept that the game follows is using Overlap2D tool to create a presentation layer respectively user interface, whilst using

LibGDX framework to define game logic such as collision detection and player movement, thus clearly separating the presentation part of the logical part. The aim of the game is to be user-oriented, to give the user maximum fun. The game therefore uses a simple player control by touching the screen.

Keywords: *Java, Overlap2D, LibGDX, 2D, collision, Eclipse, Box2D, Jetpack Rush, Android, game.*

Sadržaj

1. Uvod 1
2. Video igre 3
   2.1 Povijest video igra 3
   2.2 Industrija video igra 8
   2.3 Glavne vrste vizualne prezentacije video igara 9
      2.3.1 Izometrijska 9
      2.3.2 Filmska kamera 10
      2.3.3 Fiksni / okretni zaslon 10
      2.3.4 Slobodno-lutajuća kamera 11
      2.3.5 2D pomicanje 11
3. Indie igre vs AAA igre 13
4. Pregled korištenih tehnologija 14
   4.1 Java programski jezik 14
   4.2 Eclipse 16
   4.3 LibGDX 18
   4.4 Overlap2D 20
5. Izrada 2D igre 24
   5.1 Uvod u igru 24
   5.2 Kreiranje grafičkog sučelja 28
   5.3 Kretanje igrača 34
   5.4 Otkrivanje kolizija 37
      5.4.1 Otkrivanje kolizija s objektima scene 37
      5.4.2 Otkrivanje kolizija s objektima kreiranih u vrijeme izvođenja igre 40
6. Zaključak 42
7. Literatura 44
8. Popis slika 47

1. Uvod

Tema ovog završnog rada je izrada više platformske 2D igre. Zadatak je bio razviti igru s grafičkim sučeljem u programskom jeziku Java koristeći LibGDX i Overlap2D okruženja. Igra je primarno rađena za Android platformu pa se prema tome na zaslonu nalaze i gumbi koji služe za određene akcije igrača. Također dodana je i podrška za desktop računala te se može igrati s tipkovnicom. Igra veoma sliči poznatoj Android igri Jetpack Joyride izdane 2011. godine od strane Halfbrick Studios. Jetpack Joyride koristi jednostavno upravljanje likom dodirom na ekran, odnosno dodirom na bilo kojem mjestu na ekranu pomiče igrača prema gore. Ova igra izrađena je na sličan način uz dodatne mogućnosti kao što su ubrzavanje igrača i primjenjivanje štita. Štit kao i mogućnost ubrzavanja igrača se može kupiti s virtualnim novcem sakupljenim nakon uspješnog prolaska razine. Igra se sastoji od deset razina. Težina razina postepeno se povećava na svakoj sljedećoj razini. Sukladno tome korisnik je primoran kupovati mogućnost zaštite i ubrzavanja kako bi mogao uspješno proći razinu. Kada korisnik prvi puta instalira igru, ona započinje od prve razine. Sve su ostale razine zaključane te ih korisnik postepeno otključava nakon svakog uspješnog prolaska novo otključane razine. Igra koristi princip 2D pomicanja koji će biti objašnjen u prvom poglavlju.

Video igre su postale svakodnevica u današnjem svijetu te se više niti ne smatraju štetnima već se smatra da igre potiču motoriku korisnika. Industrija video igara nikada nije bila veća nego sada te svatko tko posjeduje neki uređaj ima ili je imao neku igru instaliranu na tom uređaju. Mobilne igre postale su hit zbog mobilnosti mobilnog uređaja. To znači da svatko može igrati svoju omiljenu igru na bilo kojem mjestu u bilo kojem trenutku. Najpoznatije mobilne igre u pravilu su manje, grafički jednostavnije i sadržajno kompliciranе što izaziva ovisnost kod korisnika. Najčešće se radi o arkadnim igrama koje igra jako velik broj igrača u periodu od nekoliko mjeseci. Manje poznate mobilne igre su u pravilu veće strateške igre koje igra manji broj korisnika, ali je period igranja znatno veći, do nekoliko godina.

U prvom poglavlju ovog rada govori se o povijesti, industriji i kategorijama video igara. Također se i detaljnije opisuje tip igre koji se koristio za izradu ove igre. Drugo poglavlje opisuje korištene tehnologije, odnosno opisuje Javu kao programski jezik te LibGDX, Overlap2D i Eclipse kao razvojna okruženja. U posljednjem poglavlju opisuje se sama igra. Poglavlje započinje pregledom korisničinih ekrana te kratkim objašnjenjima implementacije korisničkog sučelja. Nadalje opisuje se način kretanja igrača kao i otkrivanje kolizija između igrača i objekata.

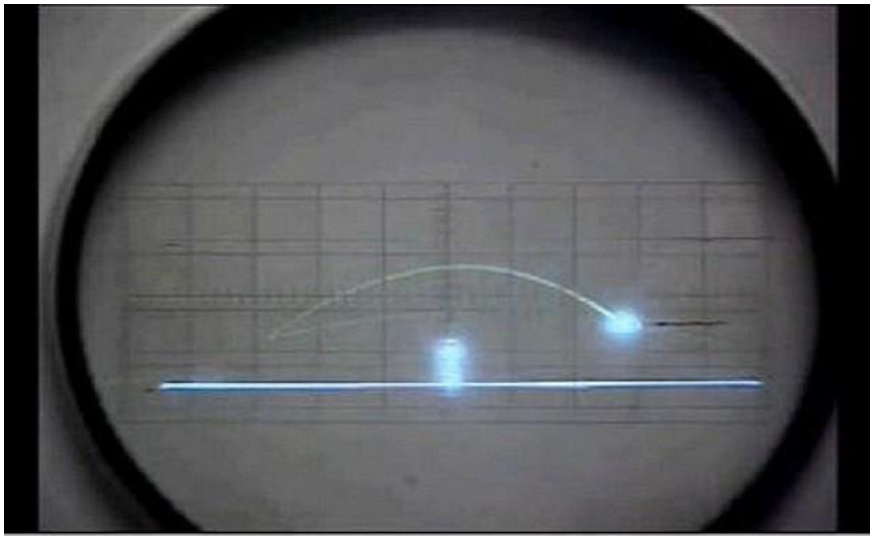
2. Video igre

U ovom poglavlju govori se o povijesti video igra počevši od pedesetih godina prošlog stoljeća pa sve do danas. Također se navode najznačajnije igre koje su obilježile određena desetljeća. Nadalje, govori se o industriji video igra i vizualnoj prezentaciji video igre.

2.1 Povijest video igra

Godine 1948. u SAD-u je odobren patent za uređaj pod nazivom Zabavna sprava od katodnih cijevi (eng. *Cathode-Ray Tube Amusement Device*) čime započinje povijest video igara [37]. Njegovi autori su Thomas T. Goldsmith i Estle Ray Mannu. Koristio je osam katodnih cijevi kako bi simulirao ispucavanje projektila na cilj, te je imao i ručice pomoću kojih se je mogla kontrolirati brzina i kut ispucavanja. U ono vrijeme se još nije moglo ništa iscrtavati po ekranu već su mete bile crtane na prozirne folije koje su bile postavljene ispred ekrana. Sljedeći izum nastao je vrlo brzo. Već je 1952. godine, A.S. Duglas napravio grafičku verziju igre križić kružić koja je sadržavala prvi oblik umjetne inteligencije.

Prva video igra dostupna javnosti razvijena je 1958. godine. Autor igre je William Higinbotham, a sama igra zove se Tenis za dvoje (eng. *Tennis for Two*). Igra je imala bočni pogled na teniski teren te je zadatak igrača bio prebaciti loptu preko mreže. U to je vrijeme izrada neke igre bila hobi kojim su se najviše bavili studenti, pa je tako 1961. godine na MIT-u nastala igra Rat svemira (eng. *Spacewar*) u kojoj su dva igrača upravljali svaki svojim svemirskim brodom koji je mogao pucati. Cilj igre bio je ubiti protivnika i ne upasti u crnu rupu koja se je kretala sredinom ekrana.



SLIKA 1: TENIS ZA DVOJE [6]

Šezdesete godine obilježila je igra Potjera (eng. *Chase*). Bila je to prva igra koju je bilo moguće igrati na televizoru, a autor iste je Ralph Baer. U sljedećih nekoliko godina, nastao je i pištolj kao prvi oblik analogne komandne palice. Tijekom sedamdesetih godina dolazi do procvata arkadnih video igara, ponajprije zbog pojave igara koje su se igrale na strojevima na žetone. Predvodnici su bili Nolan Bushnell i Ted Dabney koji su 1972. godine osnovali tvrtku Atari i na tržište izbacili igru PONG.



SLIKA 2: IGRA PONG [31]

Osamdesetih godina počele su se proizvoditi video konzole koje su mogle pokretati više od jedne igre što je uzrokovalo porast broja igara i njihovih proizvođača, što je naravno povećalo i tržište računalnih igara. Zbog brzog rasta industrije, 1983. godine došlo je do raspada tržišta za igre te je mnogo proizvođača i izdavača igara bankrotiralo. Nakon godine dana tržište se je polako počelo oporavljati te su u to vrijeme nastale neke od popularnih igra poput Super Mario Bros-a, Final Fantasy-a itd.



SLIKA 3: SUPER MARIO BROS [2]

Devedesetih godina javljaju se nove mogućnosti za igre u vidu 3D grafike. Godine 1996. na tržištu se pojavljuje Voodoo grafička kartica koja preuzima na sebe teret iscrtavanja 3D scena, a među prvim igrama koje su iskoristile tu tehnologiju bio je Quake. Nakon toga polako su sve igre počele prelaziti na 3D grafiku. Sve su igre počele dobivati i višekorisničke komponente za igranje putem interneta, a pojavile su se i mogućnosti modifikacije grafike igre koju je mogao napraviti svatko.



SLIKA 4: 3D IGRA QUAKE [5]

U novom tisućljeću počele su se pojavljivati MMORPG (*Massively Multiplayer Online Role Playing Game*) i druge vrste ogromnih online igri, a izuzev njih najpopularniji su postale FPS (*First Person Shooting*) i slične akcijske igre.



SLIKA 5: BATTLEFIELD 4 [29]



SLIKA 6: GRAND THEFT AUTO 5 [17]

2.2 Industrija video igra

Prema procjenama Klastera hrvatskih proizvođača računalnih igara [9], samo u 2014. godini, industrija računalnih igara ostvarila je u Hrvatskoj ukupne prihode iznad 50 milijuna kuna, zapošljavala je 250 osoba, a još ih se bar 200 time bave u slobodno vrijeme ili kao *freelanceri* [1].

U Hrvatskoj trenutno posluje 20-ak tvrtki koje stvaraju računalne igre. Prema podacima domaći proizvođači računalnih igara izvoze 99% zabavnog softvera, ističe tajnik Klastera, Ante Vrdelja [9], što znači da je Hrvatska kao igrač u tom segmentu prepoznata na svjetskom tržištu, koje po ukupnoj vrijednosti i prometu nadmašuje filmsku ili glazbenu industriju.

Među vodećim tvrtki koje razvijaju računalne igre u Hrvatskoj nabrajaju se: Nanobit, Lion Game Lion, Dreamatrix, 2x2 Games, Binx Interactive, Cateia Games, Exordium Games i Croteam. U posljednjih 20 godina objavljeno je gotovo 30 većih računalnih igara od kojih je desetak zabilježilo veliki uspjeh na globalnom tržištu, kao što su: Serious Sam, The Talos Principle, Startpoint Gemini, Captain Brawe, Gas Guzzlers, Unity of Command i druge.

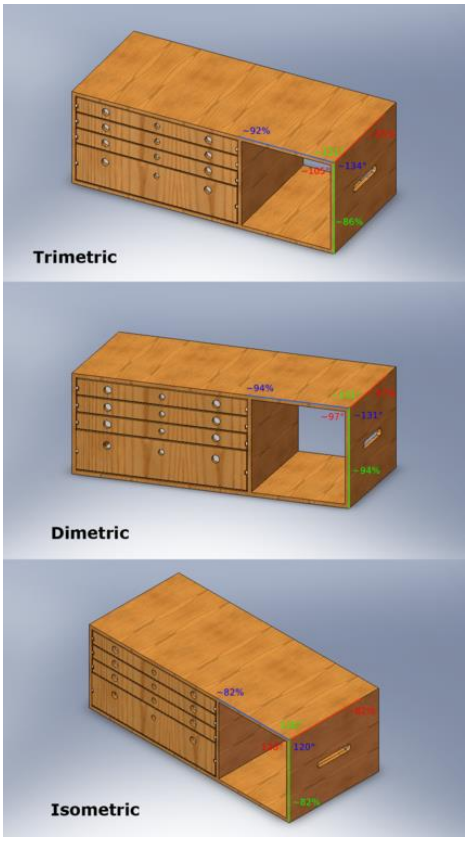
[1] Osoba koja je samozaposlena te nije nužno u radnom odnosu.

2.3 Glavne vrste vizualne prezentacije video igara

Prezentacija video igre je zapravo način na koji korisnik gleda na akciju. To je način pozicioniranja i pomicanja kamere. Sukladno tome postoji pet glavnih vrsta [32] vizualne prezentacije: 2D pomicanje (eng. 2D scrolling), filmska kamera (eng. *Cinematic camera*), fiksni / okretni zaslon (eng. *Fixed / Flip-screen*), slobodno-lutajuća kamera (eng. *Free-roaming camera*) i izometrijska (eng. *Isometric*).

2.3.1 Izometrijska

Ovakve igre koriste izometrijsku, trimetričnu ili dimetrijsku dvodimenzionalnu perspektivu kako bi stvorili 3D efekt. Iako se često koristi za zastupanje određene točke gledišta koja obuhvaća pogled iz ptičje perspektive, fokus je isključivo u korištenju 2D-a da bi se dobio 3D efekt. Na slici 7. prikazane su navedene perspektive.



SLIKA 7: IZOMETRIJSKA, TRIMETRIJSKA I DIMETRIJSKA PERSPEKTIVA [33]

2.3.2 Filmska kamera

U ovoj se perspektivi fiksne pozicije kamere postavljaju tijekom stvaranje igre za postizanje maksimalnog filmskog efekta. Korisnik ne može kontrolirati pozicije kamera. Na slici 8. plavom linijom prikazana je putanja kretanja kamere. Kamera prati tu putanje, a korisnik ne može ni na koji način promijeniti pogled kamere.



SLIKA 8: PRIKAZ PUTANJE FILMSKE KAMERE [3]

2.3.3 Fiksni / okretni zaslon

Igre koje koriste ovakvu perspektivu koriste jedan fiksni zaslon za prikaz određene razine ili scene. Veličina razine može biti i veća od jednog nepokretnog zaslona, te s u ovom slučaju, kada igrač dođe do kraja zaslona, cijeli se zaslon preklopi, točnije zamijeni s novim fiksnim zaslonom.

2.3.4 Slobodno-lutajuća kamera

U ovoj perspektivi kamera je pod kontrolom igrača i ne slijedi određeni cilj, tj. nije unaprijed određena putanja i kut kamere. Igrač ima slobodu pomicanja i rotiranja u okolini. Zumiranje je također dio ove perspektive.



SLIKA 9: SLOBODNO-LUTAJUĆA KAMERA U IGRI GTA SAN ANDREAS [23]

2.3.5 2D pomicanje

2D pomicanje [25] koriste video igre gdje je kamera postavljena pod kutom kojim može gledati akciju sa strane. Takve igre su generalno u 2D s likovima koji se pomiču s lijeve strane prema desnoj strani ekrana. Većina ovakvih igara zahtjeva korisnike da se kreću samo u jednom smjeru, najčešće udesno. Međutim, mnoge igre dopuštaju povratak, odnosno kretanje u lijevo kao i kretanja prema gore i prema dolje. Ovakav se tip prezentacije igre uobičajeno koristi za akcijske igre s likovima koji trče, penju se i preskaču kroz niz sekvencijalno naprednih razina. Jedan od primjera je izvorna igra Contra (1988) za igraću konzolu Nintendo Entertainment System (NES). U Contri su igrači morali propucati svoj put kroz svaku razinu pomicanjem udesno, bez mogućnosti povratka. Kako su se grafičke i procesorske brzine povećale i unaprijedile, video igre su također evoluirale iznad perspektive 2D pomicanja. Međutim, neke igre i danas koriste ovakav način dizajna igre.



SLIKA 10: 2D POMICANJA U IGRI JETPACK JOYRIDE [8]

3. *Indie* igre vs AAA igre

Samostalni razvojni programeri postoje oduvijek, ali se sredinom 2000. godine javlja nova ideja o nezavisnom razvoju igara. Tada se javlja i pojam Indie. Indie razvojni programeri su programeri koji razvijaju igre samostalno ili u manjim razvojnim kompanijama. Pri tome oni su neovisni, odnosno nisu u suradnji s većim tvrtkama te polažu sva prava na svoje igre. Također nisu ovisni o rokovima i nisu ograničeni nikakvim pravilima.

AAA igre su suprotnost Indie igara, odnosno to su visoko budžetne igre razvijane od strane većih razvojnih kompanija [22]. To su igre koje imaju visoke marketinške budžete koji mogu iznositi i do nekoliko desetaka milijuna dolara, te se za njih planira prodaja milijunskih primjeraka. Da bi takve igre postigle što veći uspjeh, izdavači izdaju verzije za sve veće platforme kao što su: PC, Xbox 360 i PS3.

Razlika između Indie i AAA igra je najvećim djelom budžet [20]. AAA igre, kao što je prethodno navedeno, razvijaju velike i visoko utemeljene kompanije koje su u mogućnosti izdati desetke milijuna dolara za razvoj igre, dok Indie igre razvijaju male kompanije čija je financijska moć ograničena. To znači da Indie igre imaju nešto slabiju grafiku, a najveća razlika je u samoj prodaji postignutoj od reklamiranja. Ali to ne znači da su Indie igre lošije od AAA igara. Razvojni programeri Indie igara pomiču granice dosadašnjih koncepata igara, dok se velike kompanije drže sigurnog i utvrđenog koncepta. Indie igre nisu previše usmjerene na grafiku koliko su na samu priču igre, dok se AAA usmjeravaju na realizam i jaku interakciju. Najvažnija stvar koju treba spomenuti jest da jedna AAA igra nikada neće biti besplatna, osim u slučajevima kada se loše igre prenapuhuju prije samog izdavanja što onda uzrokuje veliki pad igrača nakon samo nekoliko mjeseci nakon izdavanja igre. Indie igre mogu dolaziti kao proizvod koji se kupuje, ali istodobno postoji velik broj igara koje su potpuno besplatne. Najčešće takve igre razvijaju Indie programeri koji su razvijaju svoje prve igre ili su u fazama učenja razvijanja igara.

4. Pregled korištenih tehnologija

U ovom poglavlju govori se o tehnologijama korištenim pri izradi ove igre. Tehnologije koje su korištene su: Java programski jezik, Eclipse, LibGDX i Overlap2D razvojna okruženja.

4.1 Java programski jezik

Java je široko korišten programski jezik [15] izričito dizajniran za korištenje u distribuiranom okruženju interneta. Jedan je od najpopularnijih programskih jezika za Android mobilne aplikacije. Java je dizajnirana da ima izgled i dojam C++ programskog jezika, ali je jednostavnija za korištenje provodeći objektno-orijentirani model programiranja. Java se može koristiti za stvaranje kompletnih aplikacija koje se mogu izvoditi na jednom računalu, ili se mogu distribuirati između servera i klijenta u mreži. Do nedavno se je mogla koristiti i za izgradnju manjih aplikacijskih modula ili apleta za korištenje u dijelu web stranice.

Java programski jezik [18] je jezik na visokoj razini koji može biti obilježen sljedećim pojmovima:

- ✓ jednostavan,
- ✓ objektno-orijentiran,
- ✓ distribuiran,
- ✓ više-dretveni,
- ✓ dinamički,
- ✓ neutralne arhitekture,
- ✓ prenosiv,
- ✓ visoke performanse,
- ✓ robustan,
- ✓ siguran.

Početkom devedesetih godina prošlog stoljeća, Javu [12] je stvorio tim kojeg je predvodio James Gosling za Sun Microsystems. Izvorno je dizajnirana za upotrebu na digitalnim mobilnim uređajima, kao što su danas pametni telefoni. Međutim, kada je Java 1.0 objavljena javnosti 1996. godine njen glavni fokus je pomaknut na uporabu na internetu. Omogućavala je interakciju s korisnicima tako što je razvojnim programerima omogućavala način izrade animiranih *web* stranica. Tijekom godina evoluirala je u uspješan jezik za upotrebu i na internetu i izvan njega.

4.2 Eclipse

Eclipse je nevjerojatna zajednica alata otvorenog koda [16], projekata i kolaborativnih radnih skupina. To je platforma koja je dizajnirana od temelja kao alat za razvoj aplikacija. Dizajnirana je za rad na više operacijskih sustava pružajući robusnu integraciju sa svakim operativnim sustavom na kojim se koristi. Platforma je bazirana na Javi koja dopušta programerima kreiranje vlastitih razvojnih okruženja iz plug-in komponenta koje su ugrađene u Eclipse od strane njegovih članova.

Projekt Eclipse je započeo 2001. godine kada je IBM donirao tri milijuna linija koda iz svojih Java alata. Izvorni cilj Eclipse-a bio je stvoriti i poticati zajednicu otvorenog koda koja bi nadopunjavala zajednicu koja okružuje Apache. Smatra se da je ime dobio po svom sekundarnom cilju, a to je da zasjene (eng. eclipse) Microsoftovo razvojno okruženje Microsoft Visual Studio.

U poduzeću, velika prednost razvojne platforme otvorenog koda jest ta da dopušta IT sektoru miješanje i usklađivanje razvojnih alata, a ne predanost jednom proizvodu pojedinog dobavljača. Iako je Eclipse platforma pisana u Javi, ona podržava dodatke koji razvojnim programerima omogućuju razvoj i testiranje koda pisanog u drugim jezicima.

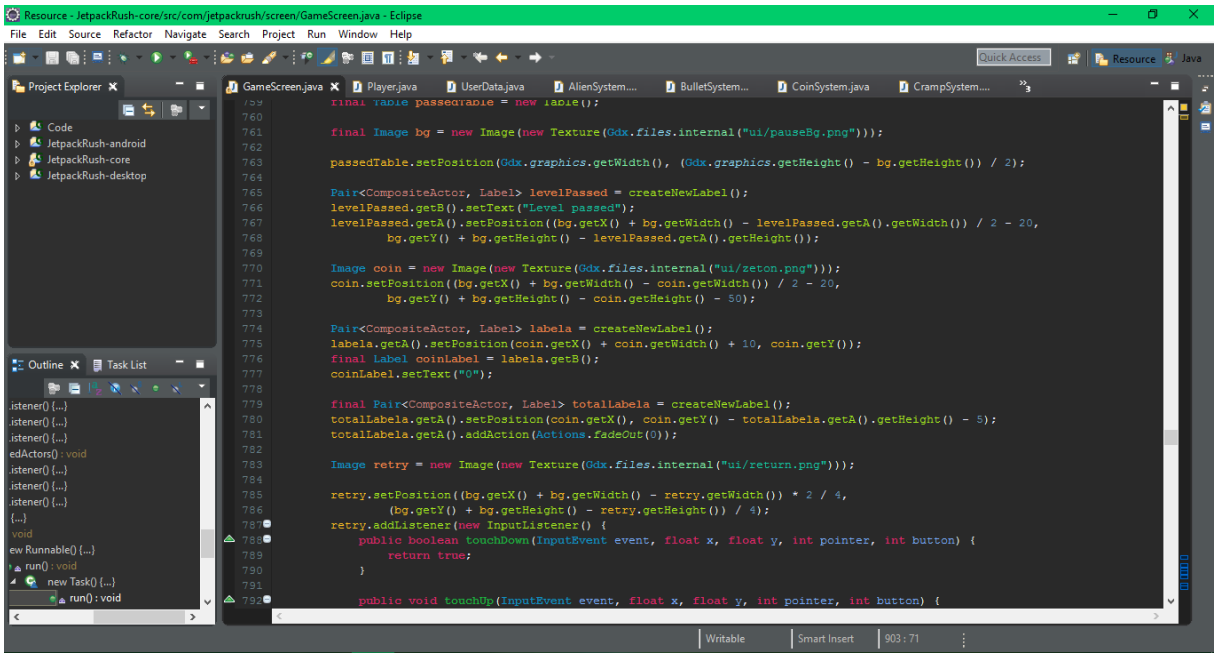
Dakle, Eclipse [26] podržava Javu kao jezik u prvom planu, ali uz to podržava i druge jezike kao što su: C, C++, JavaScript, PHP, Python, R, Ruby itd. Platforma Eclipse dizajnirana je tako da se odabire samo pojedina podrška za programski jezik te se tada samo dio platforme „instalira“ na računalo. Dodatne podrške za ostale programske jezike mogu se jednostavno dodati preko priključaka (eng. *plug-in*).

Eclipse trenutno ima 83 projekta s 287 članova koji održavaju i kreiraju nove projekte, s preko 71 milijun linija koda i 664 suradnika.

Tu se javlja i pojam Eclipse Foundation. To je neprofitna organizacija, podržana od strane članova koja je domaćin Eclipse projektima te koja pomaže zajednici otvorenog koda i ekosustavu komplementarnih proizvoda i usluga. Oni pružaju četiri usluge Eclipse zajednici:

1) IT infrastrukturu – kontrolira IT strukturu Eclipse zajednice otvorenog koda, uključujući repozitorije koda, baze podataka, razvojno orijentirane forume, stranicu za preuzimanje i glavnu stranicu.
2) Upravljanje intelektualnim vlasništvom – konstantno promoviranje i ohrabrivanje korisnika softvera na korištenje Eclipse tehnologija za izgradnju njihovih softverskih proizvoda.
3) Podrška razvojnoj zajednici – Eclipse zajednica ima zasluženu reputaciju za pružanje kvalitetnog softvera na pouzdan i predvidljiv način.
4) Razvoj ekosustava – promocija Eclipse projekata i širenje Eclipse ekosustava.

Za potrebe ovog rada korišten je Eclipse Mars 2 s paketom „Eclipse IDE for Java EE Developers“.



SLIKA 11: PRIKAZ RADNOG OKRUŽENJA ECLIPSE MARS 2

4.3 LibGDX

LibGDX je razvojno okruženje za igre i vizualizaciju [7] , koje radi na više platformi. Trenutno podržane platforme su: Windows, Linux, Mac OS X, Android, Blackberry, iOS i HTML 5. Ovo razvojno okruženje dopušta razvojnim programerima pisanje jednog koda koji će kasnije moći izvršavati na različitim platformama bez modifikacija koda. Umjesto čekanja da se najnovije promjene prebace na neki uređaj ili da se prevedu u HTML5, programer može imati koristi od ekstremno brzog integracijskog ciklusa kodiranja aplikacije uglavnom u desktop okruženju, znajući da će aplikacija imati isto ponašanje i na ostalim platformama. U razvoju se mogu koristiti i ostali programski jezici koji koriste JVM kao što su: Kotlin, Scala, Clojure itd.

LibGDX omogućava programiranje na niskoj razini, pružajući direktan pristup datotečnim sustavima, ulaznim uređajima, audio uređajima i OpenGL-u putem jedinstvenog OpenGL ES 2.0 i 3.0 sučelja.

Iznad te niske razine, LibGDX sadrži snažan skup API-a (eng. *Application programming interface*) koji pomažu programeru sa svakodnevnim zadaćama razvoja igara poput iscrtavanja animacija i teksta, izgradnje korisničkih sučelja, reprodukcije zvučnih efekata i glazbenih datoteka, izračuna linearnih algebra i trigonometrijskih izračuna, čitanja JSON-a i XML-a itd.

Tamo gdje je to potrebno, LibGDX napušta Javu i cilja na izvorni kod kako bi se nastojale postići najbolje moguće performanse. Programeri se ne moraju brinuti o radu koda na različitim platformama zbog tih funkcionalnosti Java API-ja. Mnogi dijelovi LibGDX-a zaobilaze poznate probleme pojedinih platforma što je dodatna prednost.

LibGDX nastoji biti okvir (eng. *framework*) za izradu igara, a ne jezgra (eng. *engine*), priznajući da ne postoji jedinstveno rješenje za oboje. Umjesto toga LibGDX pruža apstrakcije koje dopuštaju programerima da programiraju igru ili aplikaciju na način na koji žele.

LibGDX također povezuje mnoge biblioteke trećih strana kako bi osigurao njezinu funkcionalnost:

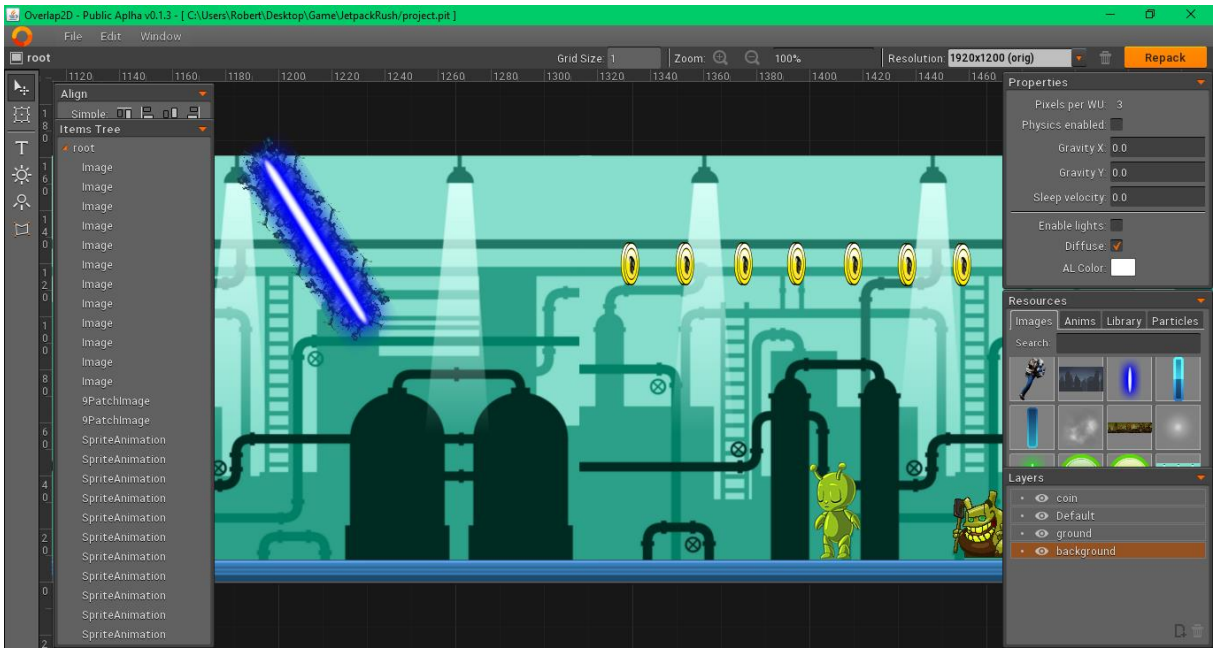
- ✓ OpenGL – premijerno okruženje za razvoj prijenosnih, interaktivnih 2D i 3D grafičkih aplikacija [11],
- ✓ FreeType – besplatno dostupna biblioteka za prikaz fontova [13],
- ✓ mpg123 – mpg123 distribucija sadrži MPEG 1.0/2.0/2.5 audio svirač / dekoder, kao i upotrebljive biblioteke za dekodiranje i izlaz [10],
- ✓ xiph.org - otvoren, besplatan, sažet audio format za srednju i visoku kvalitetu [27],
- ✓ soundtouch – biblioteka otvorenog koda za procesiranje audio zapisa, tj. za mijenjanje tempa, ugođaja i stope reprodukcije audio datoteka [19],
- ✓ Box2D – jezgra otvorenog koda pisan u C++ za simuliranje krutih tijela u 2D [30].

  Neke od značajki Box2D su:
  - neprestano otkrivanje kolizija,
  - konveksni poligoni i krugovi,
  - višestruki oblici po pojedinom tijelu,
  - učinkovito upravljanje parova,
  - kontinuirana fizika s vremenom kolizije,
  - kontakt, trenje i restitucija.
- ✓ LWJGL - Java biblioteka [14] koja omogućava različitim platformama pristup popularnim izvornim API-ima korisnim za razvoj grafičkih (OpenGL), audio (OpenAL) i paralelno računalnih (OpenCL) aplikacija.
- ✓ OpenAL – 3D audio API prikladan za korištenje s igrama i mnogim drugim vrstama audio aplikacija s podrškom za različite platforme [34].
- ✓ KissFFT – vrlo mala, razumno učinkovita, mixed-radix FFT (eng. Fast Fourier Transform) biblioteka koja može koristiti fiksne ili decimalne vrste podataka [4].
- ✓ Ashley – mali okvir entiteta napisan u Javi [21]. Nadahnut je okvirima kao što su Ash i Artemis. Ashley nastoji biti okvir entiteta bez upotrebe crne magije i tako čineći AMP jednostavnim i transparentnim za upotrebu.

Tehnologije korištene za izradu ovog rada su: FreeType, Box2D, Ashley te ekstenziju treće strane Overlap2D koje će biti objašnjena u nastavku.

4.4 Overlap2D

Overlap2D je *editor* otvorenog koda za korisničko sučelje, sadržaj i razine igre, bez jezgre, odnosno s beskonačnim brojem jezgra [36]. To je platforma koja opisuje igru programera. Izrađen je kako bi odvojio programiranje od sadržaja igre. On omogućava programeru da izradi kompleksne sadržaje korištenjem slika, animacija, efekata čestica, svjetlosnog sustava, fizike i složenih grupiranih komponenata.



SLIKA 12: PRIKAZ RADNOG OKRUŽENJA OVERLAP2D

Editor ima mnoštvo mogućnosti kao što su kreiranje komponenata koje će se koristiti kao gumb te onda te komponente mijenjaju prikaz ovisno o tome da li su pritisnute ili ne. Za svaku komponentu dodanu u *editor* se može navesti njen „Identifier“ odnosno njen jedinstveni naziv, te se onda kasnije tu komponentu može dohvatiti iz koda po tom jedinstvenom nazivu. Također se za svaku komponentu mogu navesti neka prilagođena svojstva tako da se navede naziv za to svojstvo (ključ) i njezinu vrijednost. Ta se svojstva također mogu dohvatiti iz koda. Svaka komponenta može imati i oznake (eng. *tags*). Oni služe za slučajeve kada se u kodu želi dohvatiti više

elemenata s istom funkcijom te s njima nešto raditi ili im dodjeljivati određene akcije i slično.

Elemente se na platno dodaje jednostavnim dovlačenjem iz palete slika, animacija, efekata ili iz spremljenih komponenata. Pri dovlačenju komponenata na platno potrebno je najprije označiti na koji sloj će se ta komponenta postaviti (ako je kreirano više slojeva).

Komponente se mogu i grupirati u jednu komponentu te tu komponentu spremiti kako bi se ubrzao razvoj igre i povećala upotrebljivost. To će biti objašnjeno na sljedećem primjeru komponente igrača.

Na slici 13. je prikazana figura komponente igrača koja se koristi u igri.



SLIKA 13: SLIKA KOMPONENTE IGRAČA

Ta komponenta je izrađena od više komponenata, prikazanih na slici 14.



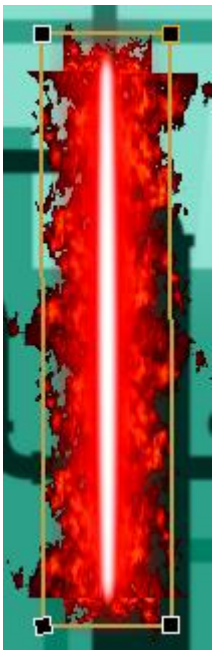
SLIKA 14: SASTAV KOMPONENTE IGRAČA

Slika s vatrom je zapravo animacija, inače bi se isti efekt mogao postići u nekom programu za slike. Animacija vatre je dva puta dodana iza slike igrača te namještena da izgleda kao da izlazi vatra iz ruksaka na mlazni pogon. Nakon što se komponente rasporede na željen način za dobivanje željenog rezultata, odaberu se sve komponente i njih se pretvori u jednu kompozitnu komponentu, koju se tada može spremiti u knjižnicu pod jedinstvenim nazivom. Iz knjižnice se kasnije mogu dovlačiti na platno ili se mogu učitavati iz koda tijekom rada igre.

Na svaku komponentu se mogu dodati tri dodatne komponente:

- ✓ *shader* komponenta,
- ✓ *polygon* komponenta,
- ✓ *physics* komponenta.

U ovom su projektu, na komponente s kojima se igrač može sudariti (osim za leteće komponente koje ispaljuju protivnici) dodane *polygon* i *physics* komponente. Dodavanjem *polygon* komponente kreira se pravokutnik oko komponente, koji se može prilagođavati po potrebi, što za ovu igru nije potrebno. *Physics* komponenta pomaže pri otkrivanju kolizija.



SLIKA 15: SLIKA ELEMENTA S POLYGON KOMPONENTOM

Pri otkrivanju kolizija elementa i igrača koristi se ovakav pristup, odnosno na svaki element s kojim se igrač može sudariti dodane su obje navedene komponente.

5. Izrada 2D igre

U ovom poglavlju prikazan je kratki pregled korisničkog sučelja, popraćen pod poglavljem u kojem se govori o kreiranju korisničkog sučelja. Nadalje, govori se o principu kretanja igrača i otkrivanju kolizija igrača sa objektima scene.

5.1 Uvod u igru

U sljedećim poglavljima govori se o korisničkim ekranima i samoj realizaciji igre i njenih glavnih funkcionalnosti.

Prvi prozor koji se otvara pri pokretanju igre sadrži mogućnosti pokretanja igre (ako je prije igrano nastavlja se na razini na kojoj je igrač bio zadnji put), pregleda i odabira razina (ako nisu još prijeđene, razine su zaključane), kupovanja nekih od dodataka u igri te mogućnosti paljenja i gašenja zvuka.



SLIKA 16: POČETNI ZASLON IGRE

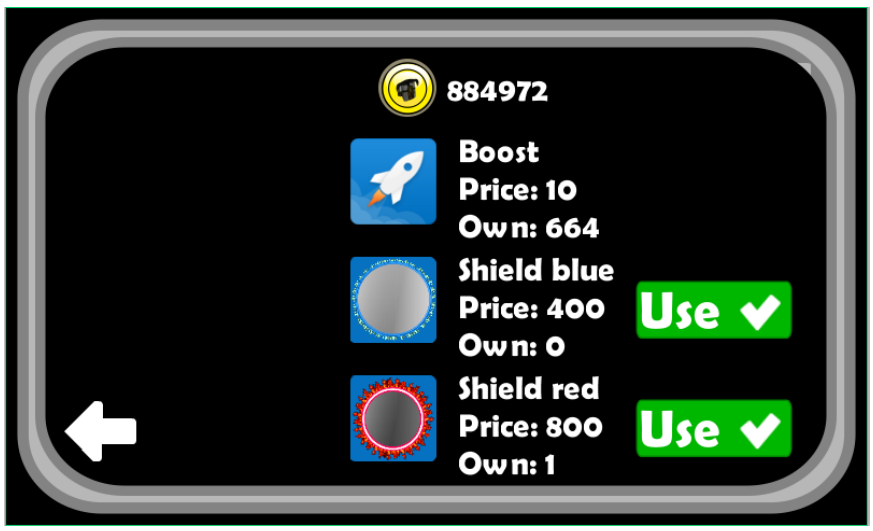
Odabirom prikaza razina („Levels“) dobivamo sljedeći ekran animacijom dolaska s desne strane ekrana do sredine ekrana.



SLIKA 17: ZASLON S PRIKAZOM RAZINA

Na ovom ekranu može se odabrati neku razinu (ako nije zaključana) te tada igra vodi na tu razinu. Strelicom u lijevom donjem kutu trenutni ekran se skriva animacijom koja pomiče ekran u desno.

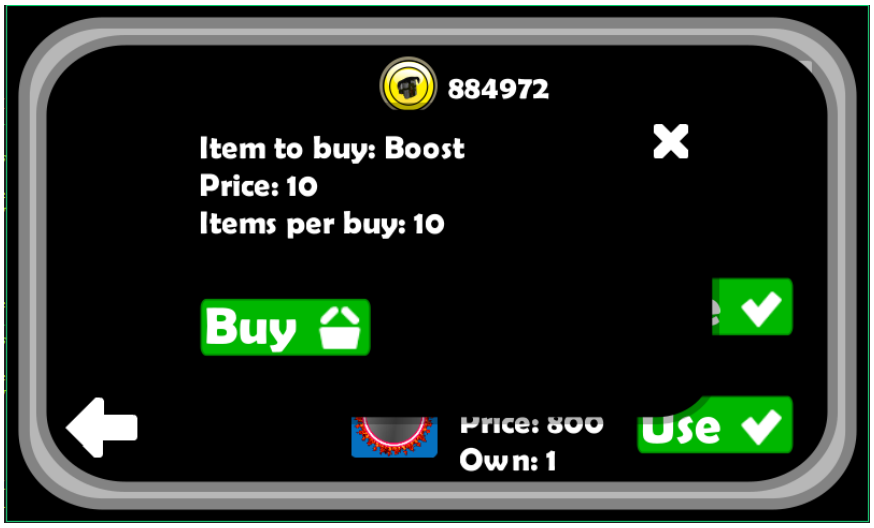
Odabirom prikaza trgovine („Store“) dobiva se sljedeći ekran animacijom dolaska s desne strane ekrana do sredine ekrana.



SLIKA 18: ZASLON S PRIKAZOM TRGOVINE

Na ovom ekranu stoji prikaz artikala koje igrač more kupiti ako ima dovoljno zlatnika. Za svaki artikl naveden je naziv, cijena i količina koju igrač trenutno posjeduje. Također je na vrhu ekrana prikazana trenutna količina zlatnika koji ima igrač. Štitovi imaju i gumb „Use“ kako bi igrač mogao odabrati štit koji želi koristiti.

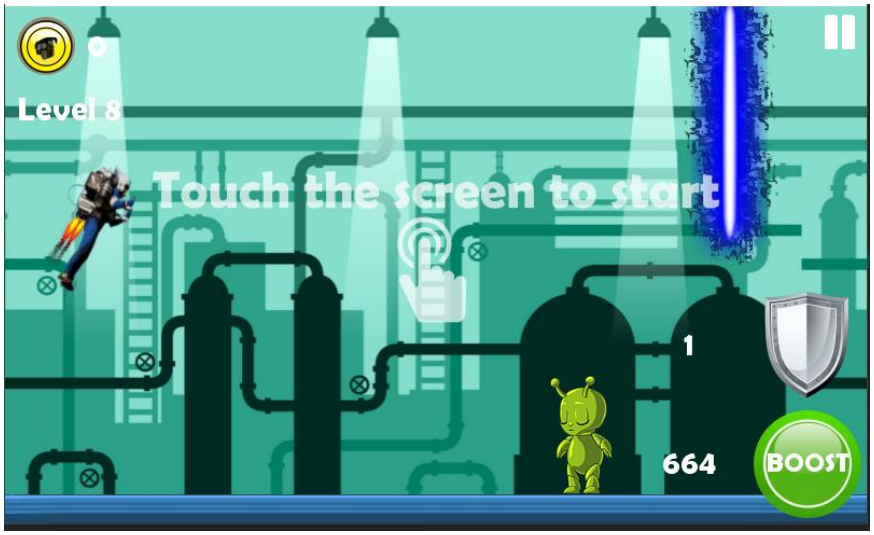
Odabirom neke od slika artikala za kupnju prikazuje se mali ekran za kupnju:



SLIKA 19: PROZOR ZA KUPNJU PROIZVODA

Na ekranu za kupnju prikazuje se gumb za kupnju te je naveden artikl koji se kupuje, njegova cijena, te količina po kupnji. Strelicom u lijevom donjem kutu trenutni ekran se skriva animacijom koja pomiče ekran u desno, dok se mali prozorčić za kupnju pomiče prema gore.

Odabirom opcije igranja („Start“) prikazuje se novi ekran koji prikazuje razinu koji je igrač odabrao ili na koju je došao sekvencijalno.



SLIKA 20: ZASLON ZA VRIJEME IGRANJA

U gornjem lijevom kutu je prikazano trenutno stanje prikupljenih zlatnika na trenutno razini te naziv razine na kojoj se igrač trenutno nalazi. U desnom donjem kutu se nalaze gumbi koji ubrzavaju igrača ili mu pružaju zaštitu, te njihova preostala količina. Dodirom na ekran počinje igra, odnosno igrač se počinje pomicati u desno te on treba izbjegavati zapreke ili metke koje ispaljuju neprijatelji.

5.2 Kreiranje grafičkog sučelja

Za prikaz početnog ekrana koristi se klasa *MainMenu* te su ti ekrani napravljeni isključivo kroz LibGDX (ne koristi se Overlap2D).

Za prikaz se koristi jedna pozornica koja je dužine 800 piksela i visine 480 piksela, koja se rasteže ovisno o veličini ekrana.

```
stage = new Stage(new StretchViewport(800, 480));
```

SLIKA 21: INICIJALIZACIJA POZORNICE

Na slici 17. i slici 18. prikazana su dva prikaza koja nisu inicijalno vidljiva pri otvaranju. Te su komponente nakon kreiranja postavljene izvan ekrana tako da ne bi bile vidljive igraču. One sadrže sve komponente koje su prikazane u njima te su one dodane na pozornicu (eng. *Stage*). Time se postiže ponašanje da kada igrač odabere pojedini izbor („Levels“, „Store“), poziva se potrebna komponenta kojoj je dodana animacija tako da ona „doleti“ s desne strane ekrana na sredinu. Kako se ta komponenta pomiče, pomiču se i sve komponente unutar nje.

Za takvu realizaciju korištena je LibGDX klasa Table na koju su dodane slike.

```
levelsTable = new Table();
```

SLIKA 22: INICIJALIZACIJA TABLICE

```
Image back = new Image(new Texture(Gdx.files.internal("ui/arrowLeft.png")));
back.setPosition(10, 10);
levelsTable.addActor(back);
```

SLIKA 23: DODAVANJE SLIKE U TABLICU

Objekt klase Table se naposljetku dodaje na pozornicu:

```
stage.addActor(levelsTable);
```

SLIKA 24: DODAVANJE GLUMCA NA POZORNICU

Odabirom opcije „Levels“ iz izbornika poziva se akcija koja premješta objekt zajedno s objektima koje on sadrži na centru ekrana dodavanjem akcije *moveTo(x,y)* objektu *levelsTable*:

```
levelsTable.addAction(Actions.moveTo((800- levelsBg.getWidth()) / 2,
        (480 - levelsBg.getHeight()) / 2, transitionSpeed));
```

SLIKA 25: DODAVANJE AKCIJA NA GLUMCA

Metoda *moveTo(x,y)* koristi se za sve animacije pomicanja objekata po ekranu.

Na slike su dodane klase koje osluškuju njihov dodir te se prilikom dodira pozivaju određene akcije.

```
levels.addListener(new ClickListener() {
    public void clicked(InputEvent event, float x, float y) {
        MyGdxGame.buttonClickSound();
        MyGdxGame.transitionSound();

        levelsTable.addAction(Actions.moveTo((800- levelsBg.getWidth()) / 2,
                (480 - levelsBg.getHeight()) / 2, transitionSpeed));

    }

});
```

SLIKA 26: DODAVANJE OSLUŠKIVANJA DODIRA

Na objekt *levels* dodan je novi objekt *ClickListener* koji pri dodiru na sliku poziva metodu *clicked(event,x,y)*, koja zatim izvršava kod koji je unutar nje.

Za prikaz ekrana razina, odnosno prikaza same igre služi klasa *GameScreen*. U ovoj se klasi učitava scena iz Overlap2D projekta te se inicijaliziraju svi potrebni objekti za prikaz igre.

Scena se učitava pozivom metode *loadScene(scene, view):*

```
sceneLoader = new SceneLoader(myResourceManager);
// ucitaj scenu
sceneLoader.loadScene(UserData.getCurrentScene(), view);
```

SLIKA 27: UČITAVANJE SCENE

Zbog jednostavnosti i brzine, komponenta igrača se učitava programski iz Overlap2D spremljene knjižnice čime uklanja potrebu dodavanje iste komponente na svaku razinu igre.

```
player = new Player(sceneLoader.world, this);

ItemWrapper playerWraper = new ItemWrapper(
        createCompositeComponent(ResourceNames.LIB_PLAYER,
                                 ResourceNames.DEFAULT_LAYER));
playerWraper.addScript(player);
player.setPosition(30, 70);
```

SLIKA 28: DODAVANJE KOMPONENTE IGRAČA IZ KNJIŽNICE

Na slici su prikazana dva različita objekta. Jedan objekt je tipa *Player* a drugi *ItemWrapper*. Objekt klase Player je objekt nad kojim se bazira logika igre, odnosno provjeravaju kolizije te se pomiče igrač, dok objekt klase *ItemWrapper* predstavlja omotač oko vizualne komponente igrača.

U konstruktoru omotača poziva se metoda *createCompositeComponent (name,layer)* koja dohvaća komponentu iz Overlap2D spremljene knjižnice, kreira entitet za tu komponentu kojeg dodaje u jezgru scene, te vraća taj entitet.

```
public Entity createCompositeComponent(String name, String layer) {
    CompositeItemVO ci = sceneLoader.loadVoFromLibrary(name);
    ci.layerName = layer;
    ci.x = -100;

    Entity en = sceneLoader
                        .entityFactory
                        .createEntity(sceneLoader.getRoot(), ci);
    sceneLoader.entityFactory
               .initAllChildren(sceneLoader.getEngine(), en, ci.composite);
    sceneLoader.getEngine().addEntity(en);
    return en;

}
```

SLIKA 29: METODA CREATECOMPOSITECOMPONENT

Komponente koje imaju ista obilježja sadrže jedinstvene oznake. To znači da svi zlatnici koji se nalaze na sceni imaju oznaku „coin“, svi vanzemaljci imaju oznaku „alien“, a ljudožderi imaju oznaku „cramp“.

Dodavanjem oznaka na komponente u Overlap2D editoru pojednostavljuje se učitavanje tih komponenata kao i definiranje njihovog ponašanja u igri. Način učitavanja komponenata s oznakom i definiranjem njihovog ponašanja isti je za sva tri navedena slučaja. Razlika je jedino u ponašanju komponenata.

Primjer vanzemaljaca

Prvi korak je učitavanje svih komponenata s oznakom „alien“ u scenu, te dodavanje tim komponentama neku novu klasu koju će one sadržavati i po kojoj će se razlikovati od ostalih. Klasa *Alien* je jednostavna klasa koja samo implementira sučelje *Component*.

```
sceneLoader.addComponentsByTagName(ResourceNames.ALIEN_TAG, Alien.class);
```

SLIKA 30: UČITAVANJE SVIH KOMPONENTA SA TAGOM „ALIEN"

Istanciranje novog iterirajućeg systema i dodavanje tog sistema u jezgru scene:

```
AlienSystem alienSystem = new AlienSystem(player, this);

sceneLoader.getEngine().addSystem(alienSystem);
```

SLIKA 31: DODAVANJE ITERIRAJUĆI SISTEM U SCENU

Klasa *AlienSystem* nasljeđuje klasu *IteratingSystem*. U konstruktoru klase pozivanjem konstruktora nad klase navode se komponente kroz koje taj sistem iterira tako da se u konstruktor predaju sve koje sadrže klasu *Alien*:

```
@SuppressWarnings("unchecked")
public AlienSystem(Player player, GameScreen gameScreen) {
    super(Family.all(Alien.class).get());
```

SLIKA 32: POZIVANJE SUPER KONSTRUKTORA SA SVIM KOMPONENTAMA KOJE SADRŽE ALIEN KLASU

Za primjenjivanje ponašanja na te komponente u klasi *AlienSystem* mora se nadjačati metodu *processEntity(entity, deltaTime)*. Ta metoda poziva se za svaki entitet (komponentu) na svakom pozivu ažuriranja *(update(deltaTime))* klase *EntitySystem*, što znači da se ta metoda poziva konstantno.

U toj metodi definirano je pravilo da ako se vanzemaljac nalazi u blizini igrača, on ispaljuje zraku smrti prema igraču. Sve dok zraka ne nestane s ekrana vanzemaljac ne ispaljuje drugu zraku, inače bi bilo nemoguće proći igru. Također se prilikom ispaljivanja zrake smrti mijenja animacija vanzemaljca na animaciju ispaljivanja zrake, te se nakon uništavanja zrake smrti vraća natrag na početnu animaciju. Nakon što se zraka uništila i animacija vratila na početnu vanzemaljac je spreman ispaliti sljedeću zraku u sljedećoj iteraciji ako igrač bude u blizini.

```
@Override
protected void processEntity(Entity entity, float deltaTime) {
    // dohvati komponentu za poziciju aliena-a
    final TransformComponent transformComponent = ComponentRetriever.get(entity, TransformComponent.class);

    // pucaj metke samo ako je alien u blizini player-a i da je player sa desne strane
    if (Math.abs(transformComponent.x - player.getX()) < 100 &&  player.getX()<transformComponent.x) {
        // dohvati animacijsko stanje aliena-a
        final SpriteAnimationStateComponent animationStateComponent = ComponentRetriever.get(entity,
                SpriteAnimationStateComponent.class);

        // radi samo ako animacija vec nije postavljena na pucanje i player
        // nije u koliziji, odnosno alien ne ispaljuje death ray-ove
        if (animationStateComponent.allRegions != shootAnimationState.allRegions && !player.collision) {
            // postavi animaciju na pucanje
            animationStateComponent.allRegions = shootAnimationState.allRegions;
            animationStateComponent.set(shootAnimation);

            // kreiraj deathRay
            final Entity deathRay = gameScreen.createCompositeComponent(ResourceNames.LIB_DEATH_RAY);


            // dodaj u system provjeravanja kolizija
            BulletSystem.addToSystem(deathRay);

            // dodaj akcije na deathRay.
            Actions.addAction(deathRay, Actions.sequence(Actions.delay(0.5f), Actions.run(new Runnable() {

                @Override
                public void run() {
                    // dohvati komponentu za pozicioniranje deathray-a
                    TransformComponent deathRayPos = ComponentRetriever.get(deathRay, TransformComponent.class);
                    deathRayPos.rotation = 40;
                    // postavi deathray iznad aliena-a
                    deathRayPos.x = transformComponent.x + 14;
                    deathRayPos.y = transformComponent.y + 19;
                    MyGdxGame.raySound();
                }

            }), Actions.moveTo(player.getX()+precision, 150, 0.7f), Actions.run(new Runnable() {

                @Override
                public void run() {
                    // makni iz systema, nije vise potrebno
                    // provjeravat
                    BulletSystem.removeFromSystem(deathRay);
                    // makni ga sa scene
                    getEngine().removeEntity(deathRay);
                    // vrati nazad na pocetnu animaciju
                    animationStateComponent.allRegions = stayAnimationState.allRegions;
                    animationStateComponent.set(stayAnimation);}})));
        }
        // ako se nebi provjeravala kolizija i sa samim alienon
        // onda bi se moglo prolazit kroz njega
        if(Math.abs(player.getX()-transformComponent.x)<20 && !player.isShieldActive() && !player.collision){
            DimensionsComponent dimensionsComponent = ComponentRetriever.get(entity, DimensionsComponent.class);
            player.collision = Intersector.overlaps(player.getRectangle(), new Rectangle(transformComponent.x,
                    transformComponent.y, dimensionsComponent.width, dimensionsComponent.height));

        }
    }
}
```

SLIKA 33: NADJAČANA METODA PROCESSENTITY

5.3 Kretanje igrača

Igrač se u igri dodirom na ekran podiže prema vrhu a nakon što se otpusti ekran pada prema dolje. Nadalje, nakon što se prvi puta dodirne ekran igrač se počne pomicati udesno. Igrač može ubrzavati ili usporavati pritiskom ili otpuštanjem gumba „Boost“. Pomicanje igrača udesno poziva se u *render(delta)* metodi *GameScreen* klase, ako igrač nije u koliziji:

```
if (!player.collision)
    player.moveForward();
```

SLIKA 34: POMICANJE IGAČA AKO NIJE U KOLIZIJI

U metodi *moveForward()* se na x os pribrojava trenutno postavljena brzina pomicanja udesno koja se množi s prošlim vremenom od zadnjeg renderiranja ekrana. Na početku je brzina pomicanja udesno postavljena na nulu, što znači da se na x os pribrojava nula, što ne pomiče igrača. Kada se dodirne ekran ispituje se je lis brzina pomicanja nula, te ako je, postavlja je na početno zadanu vrijednost. U sljedećoj iteraciji brzina pomicanja udesno više nije nula što znači da se igraču na x os pribrojavaju neke vrijednosti čineći tada igrača da se sve više pomiče udesno.

```
public void moveForward(){

    transformComponent.x+=speed.x*Gdx.graphics.getDeltaTime();
}
```

SLIKA 35: METODA KOJA MIJENJA X OS IGRAČA

Nadalje, u istoj metodi *(render(delta))* pomiče se i kamera jer bi inače igrač samo nestao s ekrana:

```
// pozicioniranje kamere za playerom
((OrthographicCamera) view.getCamera()).position.x = player.getX() + 120;
```

SLIKA 36: PREMJEŠTANJE KAMERE U ODNOSU NA IGRAČA

Pomicanje igrača gore-dolje izvršava se u metodi *act(delta)* u klasi Player. Ta metoda se poziva na svakom renderiranju ekrana.

Ako je korisnik pritisnuo ekran varijabla *jump* se postavlja na „*true“* te se vertikalnu brzinu igrača postavlja na vrijednost varijable *jumpSpeed*, ali se i oduzima za vrijednost gravitacije(gravitacija je negativna) pomnoženom s vrijednosti delta.

Ako se igrač trenutno pomiče prema gore onda treba prikazivati i mlazni pogon ruksaka te se tada koordinate mlaznog pogona postavljaju na približne vrijednosti igračevih koordinata. Ako se igrač trenutno ne kreće prema gore potrebno je skriti mlazni pogon izvan ekrana. On će se sakriti odmah u sljedećoj iteraciji nakon što korisnik otpusti ekran. Ako je igrač iznad dna, odnosno vertikalna brzina nije nula, potrebno ga je spuštati dodavanjem vertikalnoj brzini umnožak gravitacije i delte.

Jednom kada igrač dođe do vrha ekrana, tu se zaustavlja, tj. postavlja se vertikalna brzina na minus deset kako bi igrač počeo padati. Ako je igrač pao na pod, vertikalna brzina se postavlja na nulu da igrač ne bi otišao izvan ekrana. Na kraju se na y os igrača pridodaje izračunata vertikalna brzina. Ako je onda negativna, igrač se pomiče prema dolje, ako je pozitivna pomiče se prema gore, a ako je nula igrač se ne miče.

Na kraju act(delta) metode provjerava se da li igrač ima aktiviran štit te ako ima, postavlja koordinate štita tako da štit zaokruži igrača. Ako igrač nema aktiviran štit provjeravaju se kolizije o čemu se govori u sljedećem poglavlju.

```
//ako player skace pomici ga prema gore i pomici i vatru za njim

if(jump){
    speed.y=jumpSpeed;
    speed.y+=gravity*delta;
    powerTransformComponent.x = this.getX()-3.5f;
    powerTransformComponent.y = this.getY()+1;
}
// ako ne skace a vatra se nalazi na ekranu makni je sa strane
else if(powerTransformComponent.x>0){
    powerTransformComponent.x=-10;
}
// spustaj playera prema dolje ako vec igra
if(speed.y!=0)
    speed.y+=gravity*delta;
//ako je player dosai do vrha vracaj ga dolje
if(dimensionComponent.height+transformComponent.y>155)
    speed.y=-10;
// ako dode do dna ekrana zaustavi ga
if(transformComponent.y<10 && speed.y<1+gravity*delta){
    speed.y=0;
}
//premjesti playera za izracunatu brzinu
transformComponent.y+=speed.y*delta;

//ako ima stit ona postavi stit na playera, inace provjeri sudaranje
if(shieldActive) {
    shiedlPos.x=getX()-20;
    shiedlPos.y=getY()-15;
}
else{
checkCollisions();
```

SLIKA 37: METODA ACT(DELTA) U PLAYER KLASI

5.4 Otkrivanje kolizija

U ovom pod poglavlju govori se o dva načina otkrivanja kolizija igrača sa objektima scene. Prvi način koristi metodu iz biblioteke Box2D, dok drugi način koristi metodu iz LibGDX okruženja.

5.4.1 Otkrivanje kolizija s objektima scene

Kao što je već prije spomenuto u odjeljku 4.3, sve komponente koje se nalaze na sceni u Overlap2D editoru mogu imati još tri komponente. Na komponente koje mogu biti u koliziji s igračem, kao što je i prije navedeno, dodane su *physics* i *polygon* komponente. Dodavanjem tih komponenata na komponente koje mogu biti u koliziji s igračem, dobiva se mogućnost jednostavnog provjeravanja kolizija između igrača i tih komponenata pozivajući metodu *rayCast(callback, point1, point2)* iz biblioteke Box2D koja prolazi kroz sve Box2D objekte na sceni, tj. komponente koje u sebi imaju *physics* komponentu. Ta metoda uzima tri parametara: klasu koja implementira sučelje *RayCastCallback*, te dva vektora.

Sučelje ima jednu metodu koja se poziva kada se dogodi kolizija. Dva vektora koji se predaju kao parametri su početna i završna točka zamišljene linije za koju se provjerava da li prolazi kroz neku Box2D poligon komponentu. U ovoj igri ta zamišljena linija kreće od vrha igrača do dna igrača, te je malo ukošena jer je i slika igrača malo ukošena.

Za potrebe demonstracije dodan je prikaz te linije na igrača:



SLIKA 38: PRIKAZ ZAMIŠLJENE LINIJE ZA KOJU SE PROVJERAVA KOLIZIJA

Ako ta linija prolazi kroz bilo koji objekt koji sadrži Box2D poligon kao što je spomenuto prije, javlja se kolizija:



SLIKA 39: PRIMJER KOLIZIJE

Vektor *rayFrom* je zapravo točka koja se postavlja na točku u gornjem desnom kutu igrača, dok je vektor *rayTo* točka koja se postavlja na točku u desnom donjem kutu umanjena za 20 piksela na x osi. Obje koordinate moraju se množiti s brojem skaliranja svijeta zato je Box2D koristi jedinice svijeta (eng. *world units*), a ne veličinu ekrana.

Kada se dogodi kolizija poziva se metoda *reportRayFixture (fixture,point, normal, fraction)* u kojoj se varijabla *collision* postavlja na „*true*“.

```
private void checkCollisions(){
    //ne provjeravaj ako se player ne krece
    if(speed.x==0)return;

    //tocka koja je pozicionirana u gornjem desnom kutu player slike na ekranu
    rayFrom.x = (transformComponent.x+dimensionComponent.width)*PhysicsBodyLoader.getScale();
    rayFrom.y = (transformComponent.y+dimensionComponent.height)* PhysicsBodyLoader.getScale();

    //tocka koja je pozicionirana u donjem desnom kutu player slike na ekranu
    //tj. u desnom donjem kutu minus 20 pixela posto player stoji ukoso
    rayTo.x = (transformComponent.x+dimensionComponent.width-20)*PhysicsBodyLoader.getScale();
    rayTo.y =(transformComponent.y)* PhysicsBodyLoader.getScale();

    //provjeri dali se nesto sudara sa linijom koja povezuje ova dva vektora
    world.rayCast(new RayCastCallback() {

        @Override
        public float reportRayFixture(Fixture fixture, Vector2 point, Vector2 normal, float fraction) {

            collision = true;

            return 0;
        }
    }, rayFrom, rayTo);
}
```

SLIKA 40: METODA KOJA POZIVA PROVJERU KOLIZIJA

5.4.2 Otkrivanje kolizija s objektima kreiranih u vrijeme izvođenja igre

Za otkrivanje kolizija za objekte koje se lansira prema igraču (vatrena kugla i zraka smrti) korištena je LibGDX klasa *Intersector*. Ta klasa sadržava statičke metode koje rade različite vrste provjera na geometrijskim objektima: od provjeravanja sadrži li trokut točku, poklapaju li se dva pravokutnika, do računanja udaljenosti između točke i linije.

Za otkrivanje ovih kolizija također se je mogao koristiti prije navedeni pristup, odnosno dovoljno je bilo u Overlap2D editoru kreirati kompozitnu komponentu od animacije (vatrene kugle ili zrake smrti) te na nju dodati kao što je prije navedeno *physics* i *polygon* komponente. Da bi se ju moglo dohvatiti iz koda trebalo bi je i dodati u knjižnicu. Zato što je već napisan kod koji provjerava kolizije igrača sa svim komponentama koje imaju *physics* i *polygon* komponente ne bi bilo potrebno pisati nikakav dodatni kod, ali se je zbog brzine kretanja komponenata koje se lansiraju na igrača događalo da te komponente prođu kroz igrača bez prijave kolizije. Zbog toga je definirana nova klasa koja sadržava listu komponenata koje su trenutno ispucane prema igraču te se kroz njih iterira pri svakom iscrtavanju ekrana.

```
public static void checkCollisions() {

    for (Entity entity : bullets) {
        TransformComponent transformComponent = ComponentRetriever.get(entity, TransformComponent.class);
        DimensionsComponent dimensionsComponent = ComponentRetriever.get(entity, DimensionsComponent.class);

        boolean collision = Intersector.overlaps(player.getRectangle(),
                new Rectangle(transformComponent.x,transformComponent.y,
                        dimensionsComponent.width, dimensionsComponent.height));

        if (collision && !player.isShieldActive()) {

            player.collision = true;

        }
    }

}
```

SLIKA 41: METODA PROVJERAVANJA KOLIZIJA

Metoda *checkCollisions()* prolazi kroz sve komponente ispaljene na igrača, dohvaća njihove objekte za pozicioniranje i veličinu te ako se pravokutnik koji okružuje igrača sudara s pravokutnikom koji okružuje te komponente. Ako se ta dva

pravokutnika sudare, a igrač trenutno nema postavljen štit, postavi istu varijablu na „*true*“ kao i metoda u prethodnoj sekciji. Korištenjem ovog pristupa osigurano očitavanje svih kolizija, dok bi se korištenjem prethodne metode očitale samo neke kolizije.



SLIKA 42: SUDARANJE IGRAČA I VATRENE KUGLE

Na slici 42. prikazano je sudaranje igrača s komponentom vatrena kugla te pravokutnici koji ih okružuju. Za prepoznavanje kolizija koristi se metoda *overlaps(rectangle, rectangle),* što znači da je svejedno ako se pravokutnici samo dodiruju ili se jedan pravokutnik nalazi unutar drugog.

6. Zaključak

Overlap2D editor nudi mnoštvo korisnih mogućnosti koje uvelike pomažu u razvoju igre. Neke od najvažnijih mogućnosti su kreiranje kompozitnih komponenata gdje se od više komponenata slaže jedna komponenta koju se tada može koristiti u svakom trenutku, bez ponovnog definiranja kompozitnih komponenata i njihovog razmještaja. Mogućnost dodavanja tag-ova na komponente uvelike pomaže za globalno definiranje ponašanja određene grupe komponenata. Editor također nudi definiranje jedinstvenog ključa po kojim se raspoznaje pojedina komponenta.

LibGDX okruženje čini razvoj igre jednostavnim uz svoj širok opseg mogućnosti. Pruža podršku za sve aspekte igre koje bi mogle trebati razvojnom programeru, od podrške za zvuk, animacije, slike, fontove do podrške korisničkih akcija. Također pruža jednostavan način stvaranja ekrana koristeći metodologiju pozornica – glumac. Glumcima se s lakoćom mogu dodavati akcije kao što su pomicanje, rotiranje, sakrivanje, prikazivanje i slično, za koje se okruženje samo brine. Također se može definirati i sekvencijalno izvršavanje akcija glumca što još proširuje uporabljivost akcija. Uz sve navedeno, pruža i metode otkrivanja presjeka određenih geometrijskih objekata kao i udaljenosti između određenih točaka, točaka i pravca i sl., što se koristi kod otkrivanja kolizija objekata na sceni. U većini slučajeva to su pravokutnici koji se pomiču zajedno sa objektom na sceni te u svakom trenutku reprezentiraju pravu poziciju i veličinu objekta scene.

Izrada igre kombiniranjem Overlap2D i LibGDX okruženja ima veliku prednost - brzinu. Overlap2D editor je odličan alat koji se konstantno unaprjeđuje, te ako se uzme u obzir da taj alat razvija nekolicina ljudi i da je on potpuno besplatan, nekoliko pogrešaka unutar alata gotovo su zanemarive. Također, za njih programeri već znaju i one će u budućim verzijama biti popravljene.

Igra se može dodatno nadograditi s novim razinama, uvođenjem novih objekata i karaktera na scenu, dodavanjem novih artikala koje igrač može kupiti te mogućnosti dijeljenja igre s prijateljima i sl. Ako bi bila potreba za podršku više platformi, potrebno bi bilo samo napraviti novi LibGDX projekt sa svim željenim platformama te kopirati

kod i resurse u novo kreirani projekt. Trenutno projekt ima samo podršku za *desktop* računala i Android mobilne uređaje zbog dostupnosti platformi za razvoj i testiranje.

7. Literatura

[1] Avetis, (08.09.2014.), Making Flappy Bird with Overlap2D – Overlappy Tutorial, http://www.gamefromscratch.com/post/2014/09/08/Guest-Tutorial-Making-Flappy-Bird-using-Overlap2D-and-LibGDX.aspx, 30.06.2017.

[2] Barcellos C., (27.02.2015.), Nostalgia: Consoles 8 e 16 bits, http://cassiogameplays.blogspot.hr/2015/02/nostalgia-consoles-8-e-16-bits.html, 18.08.2017.

[3] Belov V., (21.06.2015.), WIP Cinematic camera, https://wildfiregames.com/forum/index.php?/topic/19949-wip-cinematic-camera/, 18.08.2017.

[4] Borgerding M., (20.06.2013.), Kiss FFT, https://sourceforge.net/projects/kissfft/, 01.07.2017.

[5] Gaming O. P., (09.01.2014.), Quake, http://www.oldpcgaming.net/quake-review/comment-page-1, 18.08.2017.

[6] Geek D., (10.08.2017.), Video Game Firsts – The Cathode Ray Tube Amusement Machine, http://www.warpedfactor.com/2015/08/video-game-firsts-cathode-ray-tube.html, 18.08.2017.

[7] Guerra M., (08.03.2017.), Introduction, https://github.com/libgdx/libgdx/wiki/Introduction, 01.07.2017.

[8] Halfbrick Studios, (08.03.2016.), Jetpack Joyride, http://jetpack-joyride.en.uptodown.com/android, 18.08.2017.

[9] Hina,(17.04.2016.), Industrija igara u Hrvatskoj raste više od 50% godišnje, https://www.24sata.hr/tech/industrija-igara-u-hrvatskoj-raste-vise-od-50-godisnje-470009, 10.07.2017.

[10] Hipp M., Orgis T., (n.d.), mpg123 – Fast console MPEG Audio Player and decoder library, http://www.mpg123.de/, 01.07.2017.

[11] Khronos Group, (n.d.), OpenGL Overview, https://www.opengl.org/about/, 01.07.2017.

[12] Leahy P., (09.07.2017.), What is Java?, https://www.thoughtco.com/what-is-java-2034117, 09.07.2017.

[13] Lemberg W., (13.05.2017.), FreeType, https://www.freetype.org/, 01.07.2017.

[14] Lightweight Java Game Library 3, (n.d.), LWJGL Lightweight Java Game Library 3, https://www.lwjgl.org/, 01.07.2017.

[15] Margaret R., (2016), Java, http://www.theserverside.com/definition/Java

09.07.2017.

[16] Margaret R., (2017.), Eclipse, http://searchmicroservices.techtarget.com/definition/Eclipse, 05.07.2017.

[17] McCollor M., (15.08.2013.), Grand Theft Auto 5 Multiplayer Revealed!, http://www.smosh.com/smosh-pit/articles/grand-theft-auto-5-multiplayer, 18.08.2017.

[18] Oracle, (n.d.), About the Java Technology, https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html, 9.07.2017.

[19] Parviainen O., (n.d.), SoundTouch Audio Processing Library, http://www.surina.net/soundtouch/, 01.07.2017.

[20] Sagcal J., (13.05.2017.), Indie vs. AAA: What's the difference?, http://sirusgaming.info/2017/05/indie-vs-aaa-whats-difference/ , 05.08.2017.

[21] Saltares D., (15.12.2015.), What is Ashley?, https://github.com/libgdx/ashley/wiki, 01.07.2017.

[22] Schultz W., (04.03.2017.), AAA Game, https://www.thoughtco.com/what-is-aaa-game-1393920 , 05.08.2017.

[23] Seemann(14.04.2013.), Free moving camera for GTA San Andreas, http://www.gamemodding.net/en/gta-san-andreas/gta-sa-cleo-scripts/15472-svobodnoe-peremeschenie-kamery.html, 18.08.2017.

[24] Sierra K., Bates B., Head first java(2nd edition)

[25] Techopedia, (n.d.), Side scroller, https://www.techopedia.com/definition/27153/side-scroller
10.07.2017.

[26] The Eclipse Foundation, (n.d.), About the Eclipse Foundation, http://www.eclipse.org/org/, 07.07.2017.

[27] Xiph.Org., (n.d.), xiph.org, https://www.xiph.org/vorbis/, 01.07.2017.

[28] Zechner M., (n.d.), libGDX, https://libgdx.badlogicgames.com/documentation.html, 05.07.2017.

[29] ***, (n.d.), Battlefield 4TM, https://www.origin.com/irl/en-us/store/battlefield/battlefield-4/standard-edition, 18.08.2017.

[30] ***, (n.d.), Box2D - About http://box2d.org/about/, 01.07.2017.

[31] ***, (17.07. 2013.), Evolution of the joystick, https://wereallgeeks.wordpress.com/tag/control-panel/, 18.08.2017.

[32] ***, (n.d.), Game Definitions , http://www.mobygames.com/glossary/genres, 04.08.2017.

[33] ***, (29.02.2012.), Just what is ISOMETRIC?, http://www.rpgcodex.net/forums/index.php?threads/just-what-is-isometric.69829/, 18.08.2017.

[34] ***, (n.d.), OpenAL, https://openal.org/, 01.07.2017.

[35] ***, (n.d.), Overlap2D, http://overlap2d.com/documentation/, 05.07.2017.

[36] ***, (n.d.), overlap2d editor, https://github.com/UnderwaterApps/overlap2d , 01.07.2017.

[37] ***, (n.d.), Povijest računalnih igara, https://dmkbj10.wikispaces.com/Povijest+racunalnih+igara 10.07.2017.

8. Popis slika

Slika 1: Tenis za dvoje [6] 4
Slika 2: Igra Pong [31] 5
Slika 3: Super Mario Bros [2] 6
Slika 4: 3D igra Quake [5] 6
Slika 5: Battlefield 4 [29] 7
Slika 6: Grand Theft Auto 5 [17] 7
Slika 7: Izometrijska, trimetrijska i dimetrijska perspektiva [33] 9
Slika 8: Prikaz putanje filmske kamere [3] 10
Slika 9: Slobodno-lutajuća kamera u igri GTA San Andreas [23] 11
Slika 10: 2D pomicanja u igri Jetpack joyride [8] 12
Slika 11: Prikaz radnog okruženja Eclipse Mars 2 17
Slika 12: Prikaz radnog okruženja Overlap2D 20
Slika 13: Slika komponente igrača 21
Slika 14: Sastav komponente igrača 21
Slika 15: Slika elementa s polygon komponentom 22
Slika 16: Početni zaslon igre 24
Slika 17: Zaslon s prikazom razina 25
Slika 18: Zaslon s prikazom trgovine 25
Slika 19: Prozor za kupnju proizvoda 26
Slika 20: Zaslon za vrijeme igranja 27
Slika 21: Inicijalizacija pozornice 28
Slika 22: Inicijalizacija tablice 28
Slika 23: Dodavanje slike u tablicu 28
Slika 24: Dodavanje glumca na pozornicu 29
Slika 25: Dodavanje akcija na glumca 29
Slika 26: Dodavanje osluškivanja dodira 29
Slika 27: Učitavanje scene 30
Slika 28: Dodavanje komponente igrača iz knjižnice 30
Slika 29: Metoda createCompositeComponent 31
Slika 30: Učitavanje svih komponenta sa tagom „alien" 32
Slika 31: Dodavanje iterirajući sistem u scenu 32
Slika 32: Pozivanje super konstruktora sa svim komponentama koje sadrže Alien klasu 32

Slika 33: Nadjačana metoda processEntity 33
Slika 34: Pomicanje igača ako nije u koliziji 34
Slika 35: Metoda koja mijenja x os igrača 34
Slika 36: Premještanje kamere u odnosu na igrača 34
Slika 37: Metoda act(delta) u Player klasi 36
Slika 38: Prikaz zamišljene linije za koju se provjerava kolizija 38
Slika 39: Primjer kolizije 38
Slika 40: Metoda koja poziva provjeru kolizija 39
Slika 41: Metoda provjeravanja kolizija 40
Slika 42: Sudaranje igrača i vatrene kugle 41