|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU  **FAKULTET TEHNIČKIH NAUKA**  **U NOVOM SADU** |  |

Aleksandar Komazec

**Razvijanje multiplatformske video igre sa podrškom za igranje preko mreže**

DIPLOMSKI - MASTER RAD

**Mentor:**

**prof. dr Predrag Teodorović**

Novi Sad, 2022

|  |  |
| --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫ **FAKULTET TEHNIČKIH NAUKA** 21000 NOVI SAD, Trg Dositeja Obradovića 6 |
| **KLJUČNA DOKUMENTACIJSKA INFORMACIJA** |

|  |  |  |  |
| --- | --- | --- | --- |
| Redni broj, **RBR**: | |  | |
| Identifikacioni broj, **IBR**: | |  | |
| Tip dokumentacije, **TD**: | | Monografska publikacija | |
| Tip zapisa, **TZ**: | | Tekstualni štampani materijal | |
| Vrsta rada, **VR**: | | Master rad | |
| Autor, **AU**: | | Aleksandar Komazec | |
| Mentor, **MN**: | | Prof. Dr Predrag Teodorović | |
| Naslov rada, **NR**: | | Razvijanje multiplatformske video igre sa podrškom za igranje preko mreže | |
| Jezik publikacije, **JP**: | | Srpski | |
| Jezik izvoda, **JI**: | | Srpski | |
| Zemlja publikovanja, **ZP**: | | Srbija | |
| Uže geografsko područje, **UGP**: | | Vojvodina | |
| Godina, **GO**: | | 2022 | |
| Izdavač, **IZ**: | | Autorski reprint | |
| Mesto i adresa, **MA**: | | Novi Sad, Trg Dositeja Obradovića 6a | |
| Fizički opis rada, **FO**: (poglavlja/strana/citata/tabela/slika/grafikona/priloga) | | 7/59/31/8/35/0/1 | |
| Naučna oblast, **NO**: | | Elektrotehnika | |
| Naučna disciplina, **ND**: | | Napredni mikroprocesorski sistemi | |
| Predmetna odrednica/Ključne reči, **PO**: | | Video igra, mreže, više platformi | |
| **UDK** | |  | |
| Čuva se, **ČU**: | | Biblioteka FTN, Novi Sad, Trg Dositeja Obradovića 6a | |
| Važna napomena, **VN**: | | Nema | |
| Izvod, **IZ**: | | Većina modernih video igara pruža svojim korisnicima mogućnost igranja sa drugim korisnicima širom sveta, kao i čuvanje dosegnutih rezultata na računarima za pohranu podataka koji se mogu nalaziti stotinama hiljada kilometara od mesta gde se nalazi korisnik. Takođe je sve vrlo česta mogućnost da se istoj video igri pristup sa različitih uređaja poput računara, telefona, tableta što korisnicima daju mogućnost da se zabave u bilo kom trenutku i na bilo kom mestu. | |
| Datum prihvatanja teme, **DP**: | | 03.10.2022 | |
| Datum odbrane, **DO**: | | 15.10.2022 | |
| Članovi komisije, **KO**: | Predsednik: | Doc. Dr Rastislav Struharik |
|  | Član: | Prof. Dr Miodrag Temerinac |
|  | Član: | Dr Ivan Mezei | Potpis mentora |
|  | Mentor: | Prof. Dr Predrag Teodorović |  |

|  |  |
| --- | --- |
|  | UNIVERSITY OF NOVI SAD ⚫ **FACULTY OF TECHNICAL SCIENCES** 21000 NOVI SAD, Trg Dositeja Obradovića 6 |
| **KEY WORDS DOCUMENTATION** |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic Publication | |
| Type of record, **TR**: | | Textual materials, printed | |
| Contents code, **CC**: | | Master thesis | |
| Author, **AU**: | | Aleksandar Komazec | |
| Mentor, **MN**: | | Prof. Dr Predrag Teodorović | |
| Title, **TI**: | | Developing Multiplatform video game that contains support to play online | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | Serbian | |
| Country of publication, **CP**: | | Serbia | |
| Locality of publication, **LP**: | | Vojvodina | |
| Publication year, **PY**: | | 2022 | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | FTN, Novi Sad, Trg Dositeja Obradovića 6a | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 7/59/31/8/35/0/1 | |
| Scientific field, **SF**: | | Electrical Engineering | |
| Scientific discipline, **SD**: | | Advanced microprocessor systems | |
| Subject/Key words, **S**/**KW**: | | Video game, network, multiplatform | |
| **UC** | |  | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, Novi Sad, Trg Dositeja Obradovića 6 | |
| Note, **N**: | | None | |
| Abstract, **AB**: | | Most modern video games provide their users with the ability to play with other users around the world, as well as save the results achieved on data storage computers that may be located hundreds of thousands of kilometers from where the user is located. It is also very common to access the same video game from different devices such as computers, phones, tablets, which give users the opportunity to have fun at any time and in any place. | |
| Accepted by the Scientific Board on, **ASB**: | | 03.10.2022 | |
| Defended on, **DE**: | | 15.10.2022 | |
| Defended Board, **DB**: | President: | Doc. Dr. Rastislav Struharik |
|  | Member: | Prof. Dr Miodrag Temerinac |
|  | Member: | Dr Ivan Mezei | Menthor's sign |
|  | Mentor: | Prof. Dr Veljko Malbaša |  |

|  |  |  |
| --- | --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫ **FAKULTET TEHNIČKIH NAUKA** 21000 NOVI SAD, Trg Dositeja Obradovića 6 | Broj: |
|  |
| **ZADATAK ZA MASTER RAD** | Datum: |
|  |

*(Podatke unosi predmetni nastavnik - mentor)*

| Vrsta studija: | 1. Master studije |
| --- | --- |
| Studijski program: | Energetika, elektronika i telekomunikacije |
| Rukovodilac studijskog programa: | ??? |

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Stefan Tomić | Број индекса: | E1-31/2011 |
| Оblast: | Napredni mikroprocesorski sistemi | | |
| Mentor: | Prof. Dr Predrag Teodorović | | |
| NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA, IZDAJE SE ZADATAK ZA MASTER RAD, SA SLEDEĆIM ELEMENTIMA:   * problem – tema rada; * način rešavanja problema i način praktične provere rezultata rada, ako je takva provera neophodna; * literatura | | | |

**NASLOV MASTER RADA:**

|  |
| --- |
| Razvijanje multiplatformske video igre sa podrškom za igranje preko mreže |

**ТЕКСТ ЗАДАТКА:**

|  |
| --- |
| ??? |

|  |  |
| --- | --- |
| Rukovodilac studijskog programa: | Mentor rada: |
|  |  |

|  |
| --- |
| Primerak za : - Studenta; - Studentsku službu fakulteta |

Obrazac **Q2.НА.11-03** – Izdanje 3

**Lista skraćenica**

ADC - analogno-digitalni konvertor

AoA - *Angle of Arrival*

APIT - *Aproximmate Point in Triangle*

APS *- Ad Hoc Positioning*

BSM - Bežična senzorska mreža

DV-Hop - *Distance Vector Hop*

GL - Greška u lokalizaciji

GPS *- Global Positioning System*

LS *- Least Squares*

LSS - *Least Squares Solution*

max - maksimalan

MDS *- Multidimensional Scaling*

min - minimalan

NLSS -*Norm Least Squares Solution*

PGL- Prosečna greška u lokalizaciji

QP *- Quadratic programming*

RSSI - *Recevied Signal Strength Indicator*

SČ - Senzorski čvor

SDP - *Semidefinite Programming*

Sr. Vr. - srednja vrednost

TDoA - *Time Diefference of Arrival*

**Lista slika**

[Slika 1.1.1. Struktura senzorskog čvora 9](#_Toc337907798)

[Slika 1.1.2. Tipična struktura BSM 10](#_Toc337907799)

[Slika 2.1.1. Ilustracija delova sistema za lokalizaciju 14](#_Toc337907800)

[Slika 2.2.1. Model TDoA sistema 16](#_Toc337907801)

[Slika 2.2.2. Ilustracija rada TDoA 16](#_Toc337907802)

[Slika 2.2.3. Ilustracija AoA tehnike 17](#_Toc337907803)

[Slika 2.3.1. a) Trilateracija idealan slučaj b) Realan slučaj 20](#_Toc337907804)

[Slika 2.3.2. Hiperbolična lateracija 21](#_Toc337907805)

[Slika 2.3.3. Ilustracija greške u određivanju pozicije 21](#_Toc337907806)

[Slika 2.3.4. Ilustracija triangulacije 21](#_Toc337907807)

[Slika 2.3.5 Primer preseka kvadrata oko tri sidra. 22](#_Toc337907808)

[Slika 2.4.1. Ilustracija APIT algoritma 24](#_Toc337907809)

[Slika 2.4.2. Pomeranje čvora M 24](#_Toc337907810)

[Slika 2.4.3. Čvor M izvan trougla 25](#_Toc337907811)

[Slika 2.4.4.](#_Toc337907812) *[Approximate PIT](#_Toc337907812)* [test 25](#_Toc337907812)

[Slika 2.4.5 Ilustracija lokalizacije pomoću težišta 26](#_Toc337907813)

[Slika 2.4.6. DV Hop, ilustracija plavljenja počev od sidra A 28](#_Toc337907814)

[Slika 2.4.7. DV-Hop primer 30](#_Toc337907815)

[Slika 2.4.8. Euklidov algoritam primer 34](#_Toc337907816)

[Slika 3.2.1 a) Fiksni radijus b)Varijabilni radijus 38](#_Toc337907817)

[Slika 3.2.2. a) Kružno ograničenje b) Ograničenje trougla c) Presek dva konveksna ograničenja 38](#_Toc337907818)

[Slika 4.1.1. Ilustracija iDV-Hop-a 40](#_Toc337907819)

[Slika 4.2.1. Ilustracija Quad DV-Hop-a 42](#_Toc337907820)

[Slika 5.1.1. Ilustracija mrežaste raspodele čvorova 44](#_Toc337907821)

[Slika 5.1.2. Zavisnost PGL od komunikacionog dometa 45](#_Toc337907822)

[Slika 5.1.3. Zavisnost PGL od broja usidrenih čvorova 45](#_Toc337907823)

[Slika 5.2.1. Mrežasta raspodela čvorova sa šupljinom u obliku slova C 47](#_Toc337907824)

[Slika 5.2.2. Zavisnost PGL od komunikacionog dometa 48](#_Toc337907825)

[Slika 5.2.3. Zavisnost PGL od broja usidrenih čvorova 48](#_Toc337907826)

[Slika 5.3.1. Pseudoslučajna raspodela čvorova 50](#_Toc337907827)

[Slika 5.3.2. Zavisnost PGL od komunikacionog dometa 51](#_Toc337907828)

[Slika 5.3.3. Zavisnost PGL od broja usidrenih čvorova 51](#_Toc337907829)

[Slika 5.4.1. Mreža sa pseudoslučajnom raspodelom čvorova sa šupljinom u obliku slova C 53](#_Toc337907830)

[Slika 5.4.2. Zavisnost PGL od komunikacionog dometa 54](#_Toc337907831)

[Slika 5.4.3. Zavisnost PGL od broja usidrenih čvorova 54](#_Toc337907832)

**Lista tabela**

[Tabela 5.1.1. Min, maks, srednje vrednosti greške za mrežastu raspodelu, R parametar 46](#_Toc336186338)

[Tabela 5.1.2. Min, maks, srednje vrednosti greške za mrežastu raspodelu, Na parametar 46](#_Toc336186339)

[Tabela 5.2.1. Min, maks, srednje vrednosti greške za mrežastu raspodelu sa šupljinom, R parametar 49](#_Toc336186340)

[Tabela 5.2.2. Min, maks, srednje vrednosti greške za mrežastu raspodelu sa šupljinom, Na parametar 49](#_Toc336186341)

[Tabela 5.3.1. Min, maks, srednje vrednosti greške za mrežu sa pseudoslučajnom raspodelom, R parametar 52](#_Toc336186342)

[Tabela 5.3.2. Min, maks, srednje vrednosti greške za mreža sa pseudoslučajnom raspodelom, Na parametar 52](#_Toc336186343)

[Tabela 5.4.1. Min,maks, srednje vrednosti greške za mrežu sa pseudoslučajnom raspodelom sa šupljinom, R parametar 55](#_Toc336186344)

[Tabela 5.4.2. Min, maks, srednje vrednosti greške za mrežu sa pseudoslučajnom raspodelom sa šupljinom, Na parametar 55](#_Toc336186345)

Lista skraćenica…………………………………….……….………………………………….5

Lista slika…………………………………………...……….…………………………………6

Lista tabela………………………….……………………….………………………………....7

[1. Uvod 9](#_Toc336455863)

[1.1 Bezične senzorske mreže 9](#_Toc336455864)

[1.2 Lokalizacija 11](#_Toc336455865)

[1.3 Klasifikacija algoritama za lokalizaciju 12](#_Toc336455866)

[1.4 Range-based metode 12](#_Toc336455867)

[1.5 Range-free metode 13](#_Toc336455868)

[1.6 Pregled rada 13](#_Toc336455869)

[2. Pregled literature 14](#_Toc336455870)

[2.1 Komponente sistema za lokalizaciju 14](#_Toc336455871)

[2.2 Procena rastojanja/ugla 15](#_Toc336455872)

[2.2.1 Merenje jačine primljenog signala (eng. RSSI) 15](#_Toc336455873)

[2.2.2 Metode zasnovane na merenju vremena(eng. ToA i eng. TDoA) 15](#_Toc336455874)

[2.2.3 Merenje ugla primljenog signala (eng. AoA) 17](#_Toc336455875)

[2.3 Određivanje pozicije senzorskog čvora 18](#_Toc336455876)

[2.3.1 Lateracija 18](#_Toc336455877)

[2.3.2 Multilateracija 19](#_Toc336455878)

[2.3.3 Hiperbolična lateracija 20](#_Toc336455879)

[2.3.4 Triangulacija 21](#_Toc336455880)

[2.3.5 Bounding Box 22](#_Toc336455881)

[2.4 Algoritmi za lokalizaciju 23](#_Toc336455882)

[2.4.1 GPS 23](#_Toc336455883)

[2.4.2 APIT 24](#_Toc336455884)

[2.4.3 Lokalizacija pomoću težista 26](#_Toc336455885)

[2.4.4 MDS–MAP 27](#_Toc336455886)

[2.4.5 Ad Hoc Pozicioniranje 28](#_Toc336455887)

[2.4.5.1 DV-Hop Algoritam 28](#_Toc336455888)

[2.4.5.2 Poboljšani DV-Hop 31](#_Toc336455889)

[2.4.5.3 Amorfna lokalizacija 32](#_Toc336455890)

[2.4.5.4 DV-Distance algoritam 33](#_Toc336455891)

[2.4.5.5 Euklidov algoritam 33](#_Toc336455892)

[3. Konveksno programiranje u BSM 35](#_Toc336455893)

[3.1 Uvod 35](#_Toc336455894)

[3.2 Formulizacija problema i SDP programiranje 37](#_Toc336455895)

[4. Nova pobošljanja DV-Hop-a 39](#_Toc336455896)

[4.1 iDV-HOP 39](#_Toc336455897)

[4.2 Quad DV-HOP 41](#_Toc336455898)

[5. Rezultati simulacije 43](#_Toc336455899)

[5.1 Mrežasta raspodela čvorova 44](#_Toc336455900)

[5.2 Mrežasta raspodela čvorova sa šupljinom u obliku slova C 47](#_Toc336455901)

[5.3 Mreža sa pseudoslučajnom raspodelom čvorova 50](#_Toc336455902)

[5.4 Mreža sa pseudoslučajnom raspodelom čvorova sa šupljinom u obliku slova C 53](#_Toc336455903)

[6. Zaključak 56](#_Toc336455904)

[7. Literatura 57](#_Toc336455905)

# Uvod

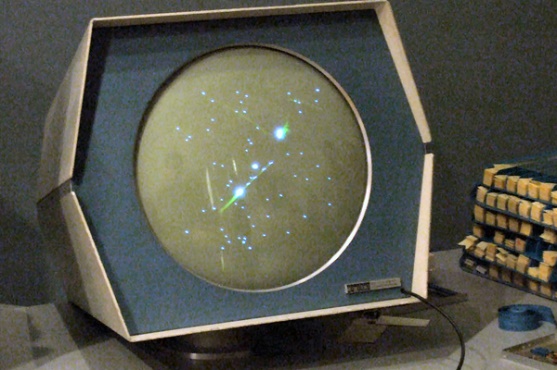
U ovom radu će biti predstavljena struktura online platform video igre kao i sam razvoj video igre i svi korišteni alati. Igra se sasotji od dve nezavisne komponente (Klijent i server). Gde je klijent razvijan koristeći biblioteke koje pripadaju random okviru pod nazivom LibGDX, dok je serverska aplikacija razvija pomoću Node rune-time okruženja koji omogućava prevođenje javaScript koda na operativnim sistemima, to jeste zahvaljući V8 engine-u koji se nalazi u Node-u (Inače taj engine se nalazi u Google Chrome pretraživaču i služi da prevede javaScript kod u mašinski kod) moguće je pisati i sistemske aplikacije, a ne samo aplikacije koje zahtevaju korišćenjje pretraživača.

# Teorijski koncepti

U ovom poglavlju je dat kratak pregled o istoriji video igara, video igrama uopšteno, o industriji video igara, kao i teorijskim konceptima

## Video igre, istorijat i žanrovi

Istorija video igara počinje početkom pedesetih godina prošlog vega kada su naučnici dizajnirali prve jednostavne video igre kao i simulacije. “Spacewar” je jedna među prvim video igrama koju su napravili studenti na MIT-u 1962. godine. Tema same igre je bila galaksija kao I svemirski brodovi.



Slika 1. “Spacewar” na jednom od prvih računara [1]

S obzirom da se ljudska potreba za zabavom nikada nije smanjila, tako je i broj video igara drastično rastao godinama, a sa porastom broja video igara, nastao je i veliki broj žanrova kojima su igre pripadale spram cilja igre. Neki od najpopularnih žanrova su:

* RPG žanr se zasniva na tome da igrač preuzima ulogu fiktivnog junaka koji se nalazi u izmišljenom svetu sa tačno određenim pravilima. Glavne odlike RPG žanra su postojanje nivoa razvijenosti glavnog junaka, zadaci koje glavni junak treba da uradi (Vrlo često se zadaci zasnivaju na tome da glavni junak treba da ubije određeni broj zlih kreatura, da pronađe izgubljeni predmet od velike vrednost i tako dalje). Junak obično ima moći koje koristi kako bi pobedio protivnike. Takođe uglavnom postoji mogućnost jačanja tih moći. U najkompleksnijim igrama postoji filmski scenario po kome se sve dešava što bi trebalo da dopuni užitak igraču tokom igranja. Glavni predstavnici ovog žanra su “The Witcher 3: Wild Hunt”, “Elden Ring”, “Horizon Forbidden West”, “Monster Hunter: World” i tako dalje.
* MMO RPG žanr je noviji žanr u odnosu na čistokrvni RPG i razlikuje se po tome što junak koga igrač upravlja sada interaguje sa hiljadama, milionima drugih igrača širom sveta, a glavni predstavnici ovog žanra su: “World of Warcraft”, “Guild Wars 2”, “EverQuest”, “Tera” i tako dalje.
* Akcioni žanr se sastoji od mnoštvo borbi sa protivnicima gde je od ključnog značaja isključivo brza reakcija samo igrača, a ne jačina oružija ili magije koju igrač koristi. Predstavnici ovog žanra su: “God of War”, “Devil May Cry 5”, “Spider-Man” i tako dalje.
* Platform žanr se zasniva na platformama po kojima glavni junak skače u potrazi za nekom vrstom novčića. Glavni predstavnici su: “Super Mario”, “Cup Head”, “Spelunky 2”, “Hollow Knight”, “Dead Cells”, “It Takes Two” i tako dalje
* Sandbox žanr je najbrližniji RPG/MMO RPG žanru osim što je glavna razlika u tome što ne postoje restrikcije kao u tim žanrovima. Igrač može da otkriva svet na način na koji on to želi i da radi ono što želi. Glavni predstavnici ovog žanra su: “Minecraft”, “Terraria”,   
  “GTA 5” i tako dalje

## Razvoj video igara

Drastičan rast video igara je posledica standardizacije razvoja video igara kao i zbog razvoja raznih alata i sofverskih radnih okvira. Softverski radni okvir je funkcionalnost ili skup funckionalnosti (Na primer radni okviri koji služi za razvijanje video igre obično obuhvata funkionalnost čitanja pritisnutog tastera na tastaturi) predstavljen u vidu računarskog koda. Svrha softverskog radnog okvira je da olakša posao tako što su bazične funkcionalnsti implementirane tako da programeri mogu da koriste te funckionalnosti ubacujući svoj kod koji razvijaju unutar koda samog sofverskog radnog okvira. Neki od softverskih radnih okvira su: “libGDX”, “Phaser”, “Game Maker”, “PyGame” I tako dalje. S obzirom da se svaka pored same logike igre sastoji i iz resursa (grafike, animacija, zvučnih efekata i muzike), postoj pregršt alata pomoću koji je moguće kreirati te resurse.

### Game Engine – Razvojno okruženje za kreiranje video igara

Glavna razlika između softverskog radnog okvira i game engine-a je u tome što je game engine skup softverskog radnog okvira kao i svih potrebnih alata za razvoj video igara. Glavna ideja game engina je standarizacija razvoja videa igara kako bi se došlo do bržeg i unfiromnog razvoja na način da svi programeri, umetnici, animatori koriste iste alate, to jeste alate koje im pruža razvojno okruženje. Tako da se u game engine-u pored softverskog radnog okvira kojeg koriste programeri nalaze i alati za razvoj nivoa u video igri, alat za kreiranje zvučnog i vizuelnog sadržaja. Takođe vrlo često najpoznatiji game engine-i pružaju i mogućnost jednostavnog umetanja projektnih fajlova iz nekih drugih alata za kreatore koji bi želeli da svoj projekat nastave unutar okvira alata game engine-a, a do sada su koristili neki drugi alat.

Neki od najpoznatijih game engine-a su:

* Unity – Verovatno jedan od najkorišćenijih game engine-a kod programera koji rade na samostalnim projektima, kao i kod manjih grupa programera. Jezici koji se koriste u ovom engine-u su C# koji omogućava fine performanse kao i veliku održivost koda, kao i Lua jezik koji služi za pisanje raznih skripti. Do relativno skoro je bio pogodan za razvisanje dvo dimenzionalnih igara ali je u međuvremenu razvijen Unity 3D koji omogućava efikasan razvoj trodimenzionalnih vide igara.
* Unreal – Pored Unity-ja jedan od najkorišćenijih game engine-a koji se uglavnom koristi za kreiranje trodimenzionalnih video igara. Jezik koji se koristi je C++, tako da je glavni adut ovog engine-a mogućnost razvoja igara sa fantastičnim performansama.
* Mnogi drugi poput Godota, GameMaker-a, CryEngine-a i tako dalje

### Pozicije u industriji video igara

Jednostavnije igre vrlo često razvija jedna do dve osobe koristeći sve pogodnosti današnjice, na primer gotove resurse(poznatijih kao asseta poput animacija karaktera, pozadina, gotovih dizajnova nivoa, muzike, zvučnih efekata), koristeći game engine koji u sebi imaju simulaciju fizike, kao i sve potrebne alate kako bi jedna ili nekoliko osoba mogli da razviju video igru. Dok kod kompleksnih video igara postoje različite pozicije gde svaka pozicija ima svoj specijalizovan zadatak. Neke od tih pozicija su:

* Programer: Pozicija programera se na većim projektima obično deli na sledeće pozicije:
  + Gameplay programer – Zadatak ove pozicije je da razvije mehaniku same igre
  + Engine programer – Zadatak ove pozicije je da radi na održavanju i razvijanju internog game engine-a jer velike firme obično razvijaju sopstveni interni game engine
  + Tool programer – Zadatak ove pozicije je da razvije sve potrebne interne softverske alate koje mogu da koriste svi članovi tima na projektu
  + Testeri – Testiraju igru tokom razvoja programirajući potrebne testove
  + Inženjer grafike – Zadatak ove pozicije je da razvija grafiči deo engine-a koristeći veliko znanje matematike kao i znanje vezano za računarsku grafiku
  + Inženjer mreža – Zadatak ove pozicije je održavanje i razvijanje baza podataka, raznih mrežnih protokola kaoi i online servirsa u video igrama koje imaju mogućnost povezivanja na internet
* Uemtinci: Pozicija umetnika podrazumeva kreiranje vizuelnih resursa potrebnih za video igru
* Modelar: U slučaju da se razvija trodimenzionalna igra, pozicija modelara podrazumeva kreiranje trodinemzionalnih resursa potrebnih za video igru
* Tehnički umetnik: Predstavlja most između programera i umetnika.
* Animator: Pozicija koja je zadužena za kreiranje animacija na osnovu vizuelnih resursa dobijenih od strane umetnika
* Dizajer video igara: Osoba na ovoj poziciji kreira nivoe, niveliše težinu same igre, ima veliko iskustvo u igranju video igara tako da razume koje elemente je potrebno da ima igra koja se razvija
* Lider Inženjer ima solidno znanje svih do sada nabrojanih pozicija i zadužen je da vodi grupu.

## Bitni koncepti

U ovom delu poglavlja biće objašnjeni teorijski koncepti na koje se naslanjaju neke od tehnologija korištenih u ovom radu.

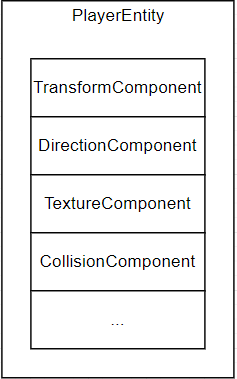
### Entity Component System (ECS)

Ovaj arhikteturalni patern se najčešće koristi u razvoju video igara i sastoji se od nekoliko elemenata:

* **Komponenta** je najmanja jedinica i predstavlja kontenjer sa podacima. Komponente se dodaju entitetima
* **Entitet** je skup komponenti i predstavlja obejkat u video igri. Na primer heroj u igri može biti predstavljen jednim entitetom koji u sebi ima više komponenti (Komponenta za kretanje, grafička komponenta, komponenta vezana za fizičke karakteristie i tak odalje)
* **Sistem** predstavlja izvršni kod koji usko specijalizovan da izvršava jednu funckionalnost nad tačno određenim entitetima (Na primer sistem za kretanje radi isključivo sa entitetima koji u sebi imaju komponentu za kretanje. Taj sistem je zadužen za modifikaciju kordinata u prostoru)
* **Motor** ili engine je odgovoran je za sve potrebne operacije (Dodavanje sistema, periodično pokretanje sistema, dodavanje i brisanje entiteta)

Dobar primer entiteta bi bio igrač koji se sastoji iz komponenti kao štu su:

* **Transform** komponenta sadrži trenutnu poziciju entiteta, kao i informaciju o skaliranju trneutne pozicije
* **Direction** komponenta sadrži informaciju o trenutnom smeru entiteta (levo ili desno)
* **Texture** komponenta sadrži informaciju koja sličica je aktiva za dati entitet u datom trenutku (U slučaju animacije sličice se menjaju u tačno određenom intervalu)
* **Collision** komponenta sadrži referencu na strani entitet sa kojim je domaći entitet u sudaru
* I tako dalje



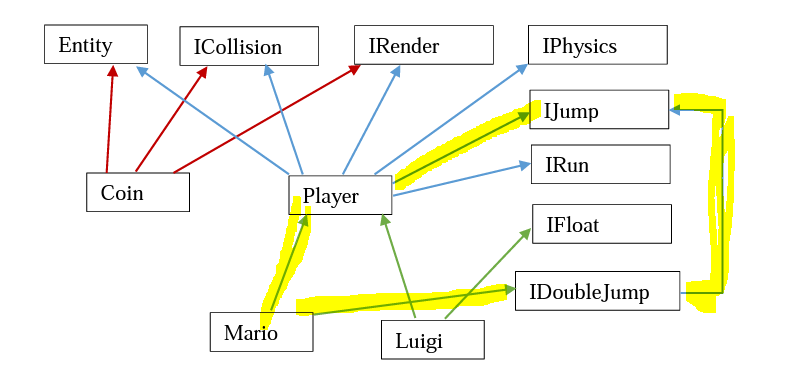
Slika 2. Primer entiteta i komponenti u ECS

Sistemi koji su manipulišu nad ovim entitetom:

* **Sistem za renderovnaje** tekstura koristi informacije vezane za poziciju entiteta kako bi znao na kom mestu treba da prikaže odgovarajuću teksturu. Sistem za renderovanje takođe koristi i informaciju o smeru entiteta kako bi znao da li je entitet okrenut ka levo ili desno i spram toga po potrebi odradila transformacija teksture koja je dobijena od strane komponentu za teksture.
* **Sistemu za razrešavanje** sudara radi sa entitetima koji u sebi imaju komponentu za sudare. Na osnovu te komponente ovaj sistem ima uvid u to koje dve komponente su se sudarile i spram toga zna kako da razreši taj sudar. Na primer kada se igrač sudari sa srcem tada je potrebno da se igraču dodaju životni poeni i da se objekat srce uništi sa nivoa jer je pokupljen

#### Koji problem ECS rešava?

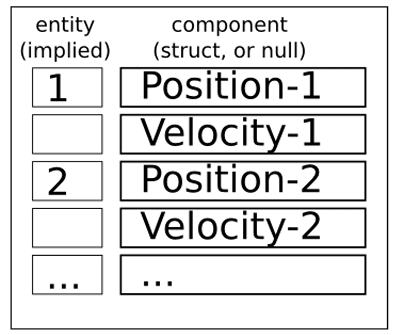
* Tradicionalan način programiranja to jeste Objketno orijentisano programiranje se zasniva na nasleđivanju koje u velikim i kompleksnim sistemima može da dovede do poznatog “dijamant smrti” problema (Slika 3.).
* Takođe je moguće doći do situacije gde jednu klasu nasleđuje četiri klase od kojih recimo troma klasama trebaju svi članovi iz roditeljske klase, a četvrtoj ne trebaju svi članovi već samo neki i na taj način dolazi do rasipanja memorije. Alternativa za ovaj i predhodni problem je korišćenje kompozicije ali i taj pristup ima nedostataka.
* Objketno orijentisan pristup takođe propagira korišćenje enkapsulacije što zahteva korišćenje gettera i settera koji predstavljaju boilerplate kod, to jeste dovode do nagomilavanja koda.



Slika 3. Dijamant problem u kompleksnom sistemu [2]

#### Prednosti ECS

* Dobro implementiran ECS skladišti entitete, komponente I sisteme jedan za drugim u memorijii tako se postiže efikasnost u iskorišćenosti memorije i omogućava brzo čitanje i upis u memoriju.



Slika 4. Pozicioniranje entiteta u memoriji[3]

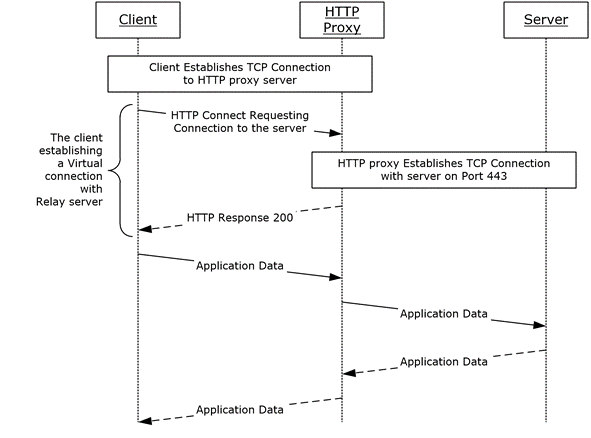
* Korišćenjem ovog paterna omgućava se jednostavno i jasno razdvajanje funckionalnosti kroz kreirajne različitih sistema koji su specijalizovani za tačno određenu namenu.
* Paralelzno izvršavanje koda je takođe lako izvodljivo i poprilčno sigurno bez bojazni da dođe do desinhronizacije između niti.
* Jednostavna komunikacija između međuzavisnih sistema.
* Jednostavno pisanje unit testova
* Jednostavno razvijanje novih mogućnosti softvera
* Dinamičko kreiranje i uništavanje sistema, entiteta i komponenti koje omogućava da se na primer neki sistem više ne koristi, kao i vrlo jednostavna implementacija nekih mogućnosti kao na primeru gde je potrebno da se igraču onemogući kretanje ako je zaleđen. U tom slučaju bi iz entiteta igrača samo izbacila komponenta za kontrolu.

#### Mane ECS-a

* ECS paradigma je efikasna u slučaju kada sistem obrađuje veliki količinu entiteta. U slučaju video igre gde postoji isključivo jedan igrač, sistem koji je zadužen za upravljanje nad igračem če biti vrlo malo korišćen i na taj način taj sistem neće ispuniti pun potencijal.
* Debagovanje problema može biti kompleksno s obzirom da komponente, entiteti i sistemi treba da budu generički
* Poprilično nepoznat patern u širokoj primeni tako da je potrebno određeno vreme da se tim osposobi da koristi taj patern.

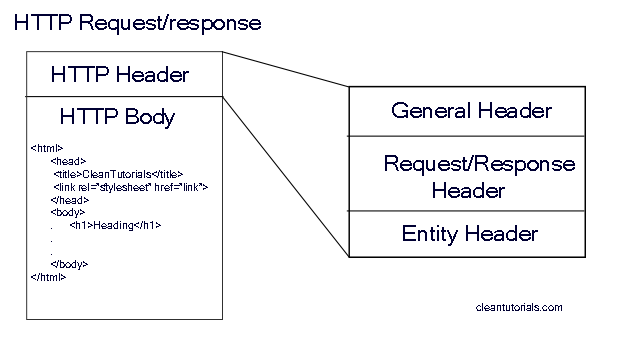
### Hypertext Transfer Protocol (HTTP)

Http je unidirekcioni komunikacioni protokol baziran na TCP prokolu korišten u komunikaciji između klijenta i servera. Klijent komunicira sa serverom tako što pošalje zahtev i očekuje odgovor od servera. Server prima zahtev i šalje odgovor klijentu koji je poslao zahtev. S obzirom da je TCP protokol orijentisan ka konekciji to znači da pri svakom slanju zahteva od strane klijenta uspostavlja se nova konekcija (Slika 5.)



Slika 5. Komunikacija između servera i klijenta [4]

Poruka zahteva i odgovora se sastoji od uvoda i tela poruke (Slika 6.)



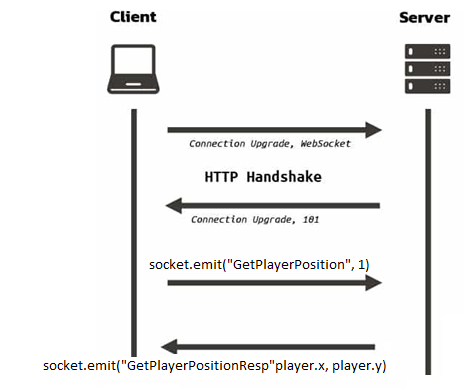
Slika 6. Komunikacija između servera i klijenta [5]

Glavni nedostatak ovog protokla se primećuje u situaciji kada je potreban čest prenos podataka jer svaki put se mora uspostaviti konekcija. Takođe i uvod svake poruke bespotrebno zatrpava prenost podataka

### WebSocket protokol

WebSocket je bidirekcioni, full-duplex protocol baziran na TCP i koristi se za komunikaciju između klijenta i servera. Početak komunikacije je identičan kao kod HTTP-a. To jeste, klijent šalje zahtev za konekicju ka server i u pozitivnom scenariju konekcija se uspostavlja. Naredni korak se razlikuje u odnosu na HTTP, a to je da je komunikacija moguća u oba smera, to jeste klijent može da šalje poruku serveru ali takođe server može da šalje poruku klijentu. Mnogo bibiloteka implementira ili proširuje implementaciju ovog protokola, kao na primer socket.io biblioteka koja se zasniva na događajima koji se sastoje od unikatnog imena pomoću koga se postiže jedinstvenost i razrešava se šta je potrebno uraditi kada se primi poruka za razliku od HTTP-a gde se jedinstvenost postiže na osnovu podataka iz uvoda poruke (Jedan od važnih podataka u uvodu HTTP poruke je krajnja adresa (na primer <https://www.geeksforgeeks.org/javascript/>) kao i standardna metoda poput GET, PUT I tako dalje) i podataka koje treba poslati ili primiti. Takođe komunikacija se zatvara kada jedna od strana prekine komunikaciju namerno ili usred određenih problema.

Na primeru ispod (Slika 7.) se može videti proces komunikacije između servera i klijenta.  
Uspostavlja se komunikacija, takozvano rukovanje. Nakon toga klijent šalje poruku serveru u kome traži trenutnu poziciju igrača čiji identifikacioni broj 1. Server po prijemu poruke proverava naziv poruke (“GetPlayerPosition”). Pošto je server pronašao naziv te poruke u mapi imena poruka, tada šalje odgovor klijentu u kojoj je navedeno jedinstveno ime poruke kao i tražene informacije o poziciji igrača.



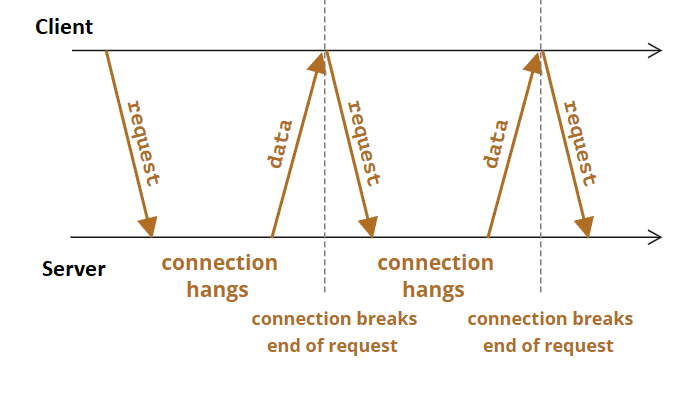
Slika 7. Komunikacija između servera i klijenta zasnovana na WebSocket protokolu [6]

### Long Pooling

Long Pooling je tehnika koja se koristi kako bi se održala konekcija između servera i klijenta

Proces Long pooling-a (Slika 8.):

1. Zahtev se šalje serveru.
2. Server ne zatvara vezu dok ne dobije poruku za slanje.
3. Kada se pojavi poruka – server šalje poruku kao odgovor na zahtev klijentu
4. Klijent odmah šalje novi zahtev.



Slika 8. Long pooling tehnika [7]

# Alati, eksterne biblioteke, radni okviri

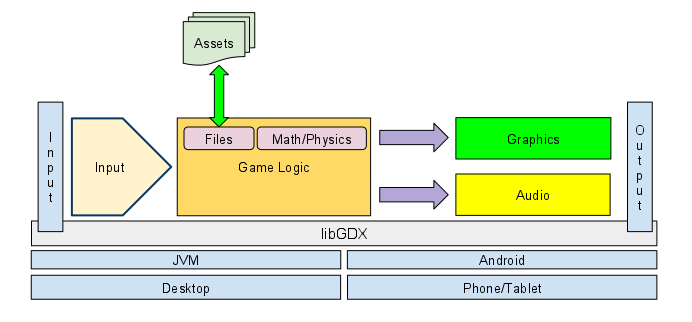
Kao što je već pomenuto u uvodu, igra se sastoji od klijenta, servera i baze podataka, tako da će svi korišćeni alati I biblioteke biti grupisani u te tri grupe.

## Klijent

Video igra to jeste klijent je razvijana koristeći radni okvir pod nazivom “LibGDX” koji se bazira na programskom jeziku Java.

### “Libgdx” radni okvir

Ovaj radni okvir ima podršku za razvijanje video igara na različitim platformama i sistemimapoput Windows, Mac, Linux, Android, iOS, and HTML5 kao što je prikazano **na slici 2.**



Slika 9. Jednostavna arhitektura video igre razvijanje u “Libgdx” random okviru [8]

Desktop backend se uglavnom oslanja na LWJGL (Light Weight Java Game Library). U isto vrijeme, LWJGL je izgrađen na vrhu veličanstvene OpenGL (Open Graphics Library). Zabavna činjenica je da je Minecraft kreiran pomoću LWJGL. [9]

Za razvoj Androida, Libgdx pronalazi svoje resurse na službenom Android SDK-u, kao i na ugrađenoj verziji OpenGL-a specifičnoj za sistem, koja se zove OpenGL ES. [9]

Ovo postaje mnogo teže kada je u pitanju podrška za HTML5 jer tehnologije su dosta različite. HTML5 može prikazati neverovatno dobru vizulenu

aplikaciju putem WebGL-a i JavaScript-a.

Kompatibilnost sa pretraživačima se postiže putem GWT koji prevodi Java kod u optimizovan JavaScript kod. [9]

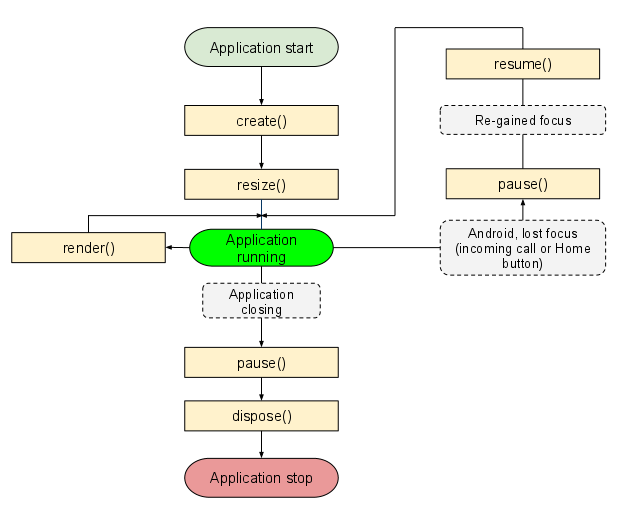
iOS podrška koja se oslanja na RoboVM. Ovaj softver prevodi java kod u ARM ili x86 kod [9]



Slika 10. Opšti prikaz Libgdx bekenda za različite paltforme [9]

Libgdx je brz, zahvaljujući upotrebi JNI koji omogućava da se C++ omota i koristi u projektima gde se koristi Java programski jezik. Njegova okvirna priroda omogućava da se odabere koje komponente se koriste jer ne nameće jedan način rada kao većina dosta game engine-a.

Svaka igra u sebi ima takozvanu glavnu petlju koja se izvršava određeni broj puta u sekundi (30, 60 I više puta u sekundi). Unutar te petlje programmer razvija samu logiku igre. U Libgdx-u ne postoji eksplicitna glavna petlja već render() metoda predstavlja ulazno mesto za kod programera kao što je prikazano na slici 4.

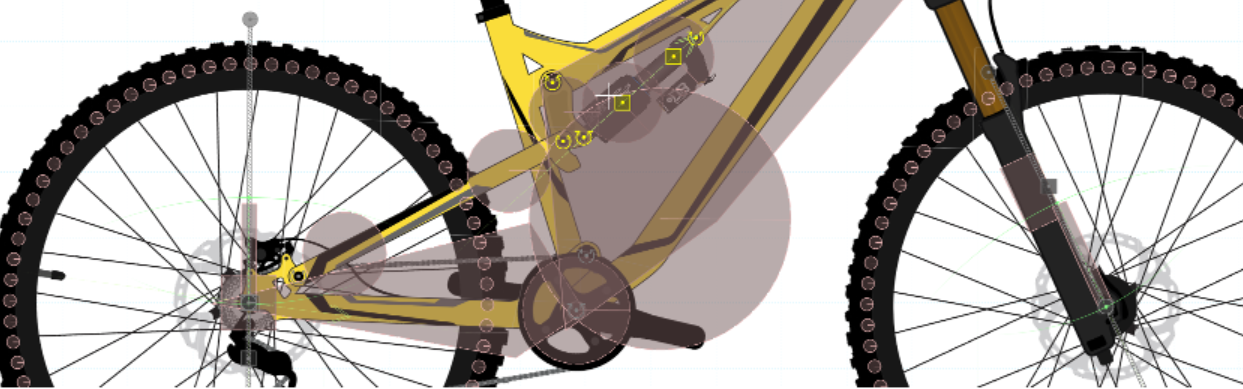


Slika 11. Životni ciklus LibGDX aplikacije [8]

create() metoda se poziva jedanput nakon sto se aplikacija kreira, resize() metoda se poziva kada se promeni veličina ekrana, render() metoda predstavlja glavnu ulaznu tačku kao što je već pomenuto, pause() metoda se na android poziva kada telefon registruje dolazeći mobilni poziv, a na računaru kada je prozor spušten, resume() metoda se poziva nakon što se focus vrati aplikaciji,  
dispose() metoda se poziva nakon što je aplikacija uništena

### “Box2D” engine

Box2D je engine za simuliranje fizike u dvodimenzionalnom prostoru. Napisan je u C++ programskom jeziku tako da je veoma brz. Box2D engine programerima pruža API pomoću koga je moguće kreirati realističan svet. Zahvaljujući JNI Box2D engine je moguće koristiti u java projektima uključujući i LibGDX projekte. Pored jednostavnih tela moguće je kreirati i kompleksna tela sastavljena od desetina i stotina objkata povezanih takozvanih zgblovima. Libgdx radni okvir sam po sebi ne sadrži alate za kreiranje kompleksnih tela, pa je u te svrhe potrebno koristiti neki eksterni alat na primer R.U.B.E. Na slici 5. je prikazan bicikl koji je sastavljen od velikog broj jednostavnih tela koja su međusobno povezana.



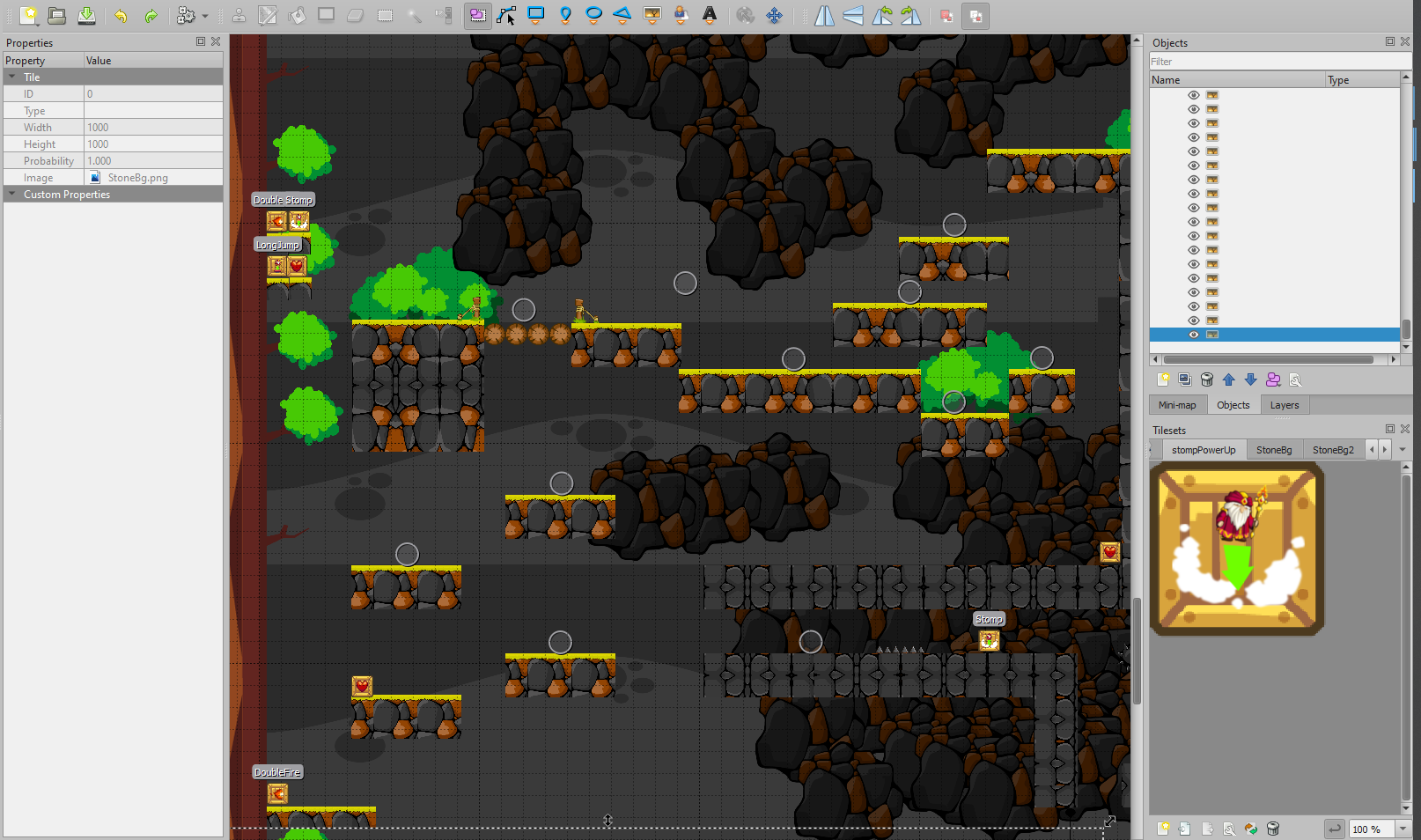
Slika 12. Projektovanje realisitčnog kompleksnog tela u alatu R.U.B.E[10]

### “Ashley” biblioteka

Ashley je biblioteka zasnovana na ECS arhitekturalnom softverskog paternu koji je se obično kreira isključivo pri razvoju video igara. ECS softverski patern se obično koristi za razvoj sveta u video igrama, a manje za logiku vezanu za menije, forme za logovanje i tako dalje.

### “Tiled” biblioteka i softverski alat

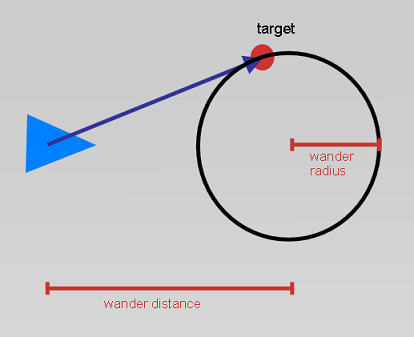
Tiled biblioteka omogućava jednostavnu integraciju nivoa kreiranih u “Tiled” softverskom alatu za dizniranje nivoa za video igre.



Slika 13. Kreiranje nivoa u softverskom alatu “Tiled

### “gdxAI” radni okvir

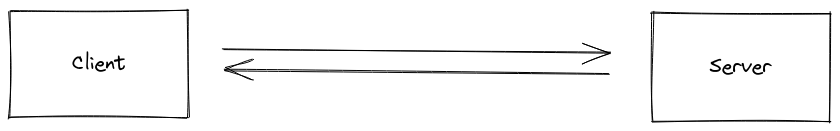
gdxAI predsavlja radni okvir za kreiranje veštačke inteligencije u video igrama. Obično se koristi kada želi da određenom protivniku dodeli željeno ponašanje. Jedna od situacija gde se ovaj radni okvir može primeniti je kod kretanja protivnika. Na primer moguće je dodeliti protivniku ponašanje da luta po nivou dok ne dođe na određenu distancu od svoje mete i tada prelazi u napadački režim kao što je prikazano na slici 14.



Slika 14. Protivnik kontrolisan veštačkom inteligencijom u stanju lutanja[11]

### “socket.io-client” biblioteka

Socket.IO predstavlja biblioteku za biderekcionu komunikaciju između klijenta i servera (Slika 15.) zasnovanu na događajima. Takođe ova biblioteka pruža jako mala kašnjennja pri razmeni poruka.



Slika 15. Bidirekciona komunikacija Server-Klijent [12]

Ova biblioteka ne predstavlja implementaciju WebSocket protokola već se zasniva na već gotovoj implementaciji WebSocket protokola, to jeste unapređuje mogućnosti implementacije pomenutog protokola. Socket.io-client biblioteka predstavlja klijentsku stranu API-ji za ovu biblioteku su razvijeni za jezike poput Java, C++, Python, Javascript. Najpopularnija verzija API-ja je u JavaScript-u, a to se odlikuje sa gomilom primera i dokumentacije, kao i redovnih unapređenjima biblioteke dok je implementacija API-ja za Javu dosta zapostavljena što može da predstavlja problem ukoliko se koristi Java programski jezik za razvijanje klijentske strane.

Tehnička unapređenja koje donosi Socket.io biblioteka u odnosu na čistu implementaciju WebSocket-a su:

* Vraćanje na HTTP long-polling predstavlja mogućnost da se pređe na HTTP long-polling u slučaju da konekcija koristeći web sokete ne uspe
* Automatska rekonekcija koja omogućava da se klijent rekonektuje ako je došlo do diskonekcije. Ovo je omogućeno implementacijom “Heartbeat“ mehanizma na serverskoj strani biblioteke
* Baferovanje paketa omogućava da se paket koji je bio spreman za slanje sačuva u slučaju da se klijent diskonektuje i da se paket pošalje pri ponovnoj konekciji klijenta
* Potvrda predstavlja opcionu mogućnost da pošiljalac paketa očekuje potvrdu od primaoca da pošalje potvrdu kada je primio paket.
* Emtiovanje podatka od strane servera na više klijenata
* Multipleksiranje konekcija

### “gdx-setup”

Gdx-setup je softverski alat koji omogućava da se kreira prazan projekat koji koristi LibGdx radni okvir. Takođe je moguće pri kreiranju projekta označiti i neke potrebne LibGDX biblioteke da budu dopremljene pri samom kreiranju projekta. Mada postoji mogućnost nakadnog dopremanja eksternih biblioteka koristeći “Gradle” koji predstavlja alat koji automatski razrešava sve moguće zavisnosti, tačnije doprema sve eksterne bibloteke sa interneta potrebne u projektu



Slika 16. Gdx-setup alat za kreiranje projekta [13]

### “Texture Packer”

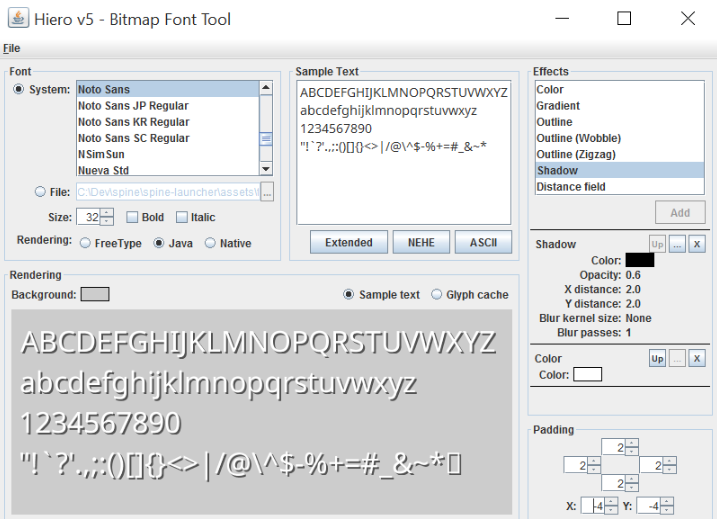
Texture Packer predstavlja alat koji služi za sjedinjavanje više sličica na jedan papir, kako bi se grupisale sve potrebne sličice koje čine animaciju (Slika 17.)



Slika 17. Animacija kretanja čarobanja sastavljena od sličica koristeći Texture Packer

### “Hiero”

Hiero predstavlja alat koji se koristi za kreiranje fontova različtih veličina, oblika, boja koji se koriste u video igrama.



Slika 18. Hiero alat za kreiranje fontova[14]

### “Krita”

Krita je alat za kreiranje vizuelnih resursa za dvodimenzionalne igre. Alat je korišćen prevashodno za promenu dimenzija i boje nekim resursima korišćenim u igri.

### “Android Studio”

Android studio predstavlja razvojno okruženje koje omogućava razvoj softvera na više platformi sa akcentom na Android platformu.

## Server i baze podataka

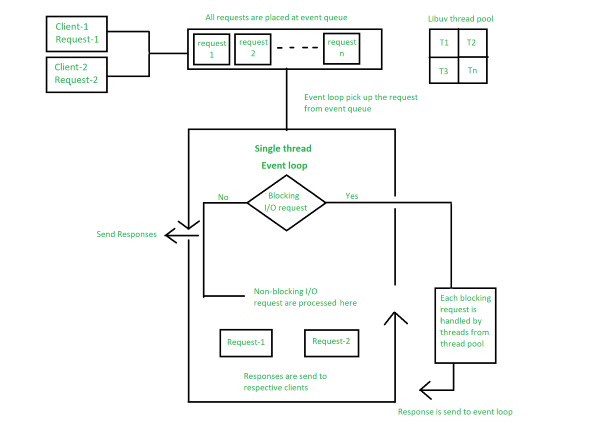
Server razvijan za potrebe video igre pisan je u programskom jeziku JavaScript.

### “Node.js ”

Node je asihnrono runtime okruženje koje se zasniva na V8 engine-u koji je implementiran u pretraživač “Google Chrome” i služi za konvertovanje JavaScript koda u mašinski kod. Tako da pojavom Node-a moguće je pored web aplikacija razvijati i sistemske aplikacije, to jeste nije potreban pretraživač kako bi se izvršila aplikacija napisana u JavaScript kodu već je aplikaciju moguće pokrenuti i iz terminala pozivajući glavni java script fajl iz nekog Node projekta.

Važne stvari koje treba napomenuti o Node-u:

* Node je event loop, single thread, to jest kada se pokrene Node aplikacija sav kod koji ne spada u inicijalizaciju se izvršava u petlji događaja koristeći jednu nit.
* Iako Node koristi isključivo jednu nit, moguće je da obrađuje stotine pristiglih zahteva paralelno
* Zbog korišćenja jedne niti ne preproučuje se implementacija operacija koje zahtevaju jaku procesorku moć
* Implementacija sistema koji odgovara na zahteve se zasniva na Libuv bilbioteci napisanoj u C programskom jeziku koja apstrahuje niske slojeve poput fajl sistema, mreže, operativnog sistema
* Bazen niti je takođe vrlo moćna mogućnost Node-a. Iako Node koristi jednu nit kada je pristigli zahtev ne blokirajuci, dok u slučaju blokirajućih zahteva dodatne budu upošljene da obrate blokirajuće zahteve
* Red događaja predstavlja red u koji se smeštaju zahtevi klijenata jedan za drugim



Slika 19. Obrađivanje zahteva u Node-u [15]

### “Express.js”

Express.js predstavlja radni okvir baziran na već postojećem Node.js. To jeste proširuje mogućnosti postojećeg web server modula koji je sastavi deo Node-a. Poboljašanja koja su uvedena u Express u odnosu na Node su:

* Middlewares – Parče koda koje može da sadrži proverve pri dospeću zahteva. Na primer pri zahtevu korisnika da se prijavi na društvenu mrežu će se proveriti da li se njegovi kredencijali nalaze u bazi podataka. Ako je odgovor potvrd korisnik će dobiti potvrdan odgovor od servera i konekcija će biti uspostavljena.
* Rutiranje pristiglih zahteva spram krajnje tačke definisane u zahtevu
* Manje vremena uloženog na razvijanje

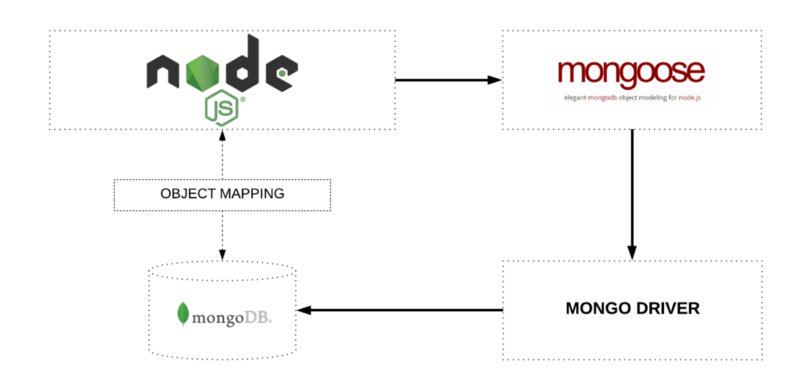
### “socket.io-server”

Socket.io biblioteka na serverskoj strani zahteva http web server zbog toga kao što je već pomenuto socket.io biblioteka je bazirana na implementaciji web soket protokola koji pri uspostavljanju konekcije dobija od klijenta HTTP zahteva i klijentu vraća HTTP odgovor. Više o biblioteci se moži naći u poglavlju 3.1.6

### “Mongose” i “MongoDB”

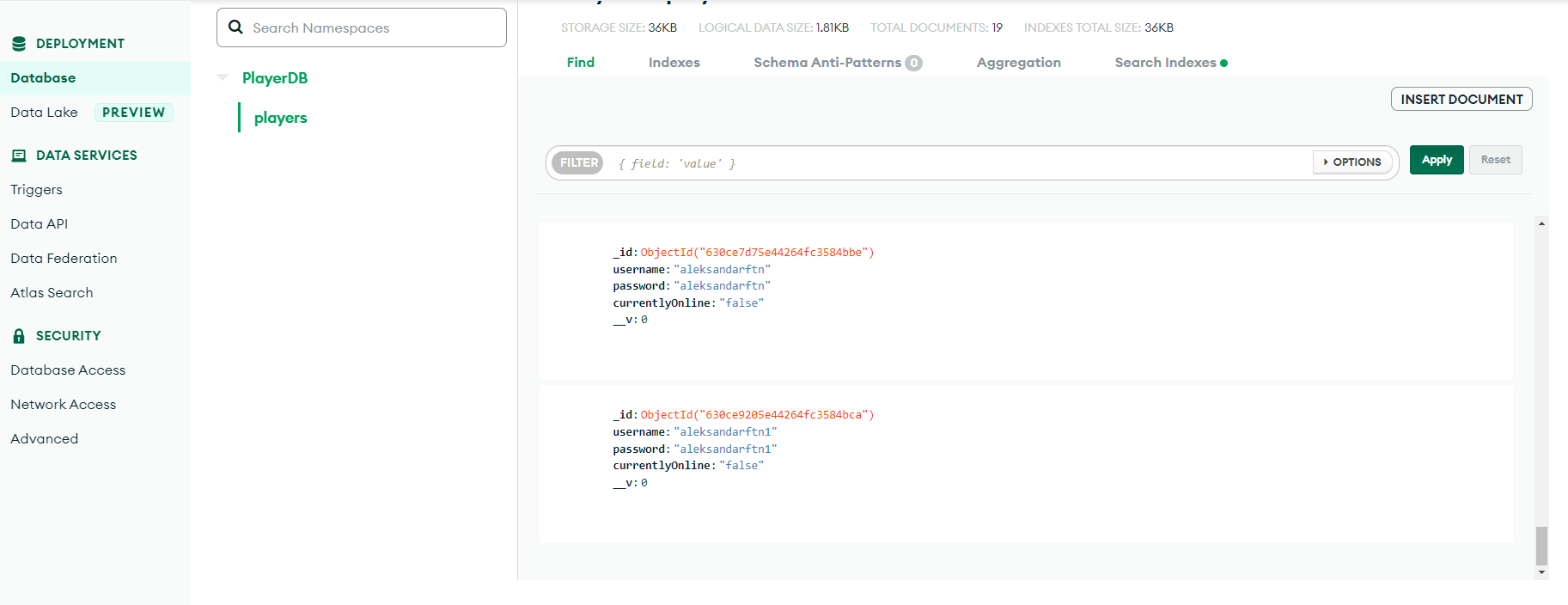
MongoDB je NoSQL baza podataka dokumenata bez šeme. To znači da se u tu takvu bazu podataka može sačuvati JSON dokumenti, a struktura ovih dokumenata može varirat. Ovo je jedna od prednosti korištenja NoSQL-a jer ubrzava razvoj aplikacija i smanjuje složenost implementacije.

Mongoose je biblioteka za modelovanje podataka objekata (ODM) za MongoDB i Node.js. Upravlja odnosima između podataka, pruža proveru validnosti šeme i koristi se za prevođenje između objekata u kodu i predstavljanje tih objekata u MongoDB. [16]



Slika 20. Maprianje objekata između Node-a i MongoDB pomoću Mongoose-a[16]

MongoDB pruža mesto na njihovim serverima tako da korisnici MongoDB baze podataka ne moraju da hostuju MongoDB server na lokalnom računaru. Pistup MongoDB bazama podatak se vrši pristupom na ovicijalni sajt.



Slika 21. Primer MongoDB baze podataka

### “Npm”

Npm ili ti Node Package Manager je mandžer paketa pomoću koga se iz komandne linije jednostavno može skinuti bilo koja potreba externa biblioteka za Node.js.

### “DigitalOcean”, “Putty/WinSCP, PM2”

DigitalOcean je Američki provajder koji je korišćen kako bi se serverska aplikacija radila sa udaljenosti bez potrebe da se lokalni računari konfigurišu kao serverske mašine. Pristup server je moguće izvršiti korišćenjem Putty. Putty pruža mogućnost da korisnik pristupi sadržaju na serveru koristećni terminal. Obično se iz terminala vrše promene vezane za prava pristupa fajlova pošto je u pitanju Linux operativni sistem. Takođe se pokreće serverska aplikacija kao fokusirani proces ili kao pozadinski proces koristeći PM2 (proces menadžer za node aplikacije) dok WinSCP pruža mogućnost da korisnik pristupi sadržaju servera koristeći grafički interfejs (Ovo se obično koristi da se jednostavnim povlačenjem fajlova doda ili obriše neki fajl sa servera, alternativa bi bila korišćenje nekog source code management alata poput git-a)

### Git/Git Extensions

Git predstavlja sors kod menadžment alat pomoću koga programeri vrlo lako mogu da čuvaju stabilne verzije koda na serveru. Git komande se izvršavaju u terminalu zvanog git bash.  
Do sada je razvijeno mnogo alata sa grafičkim interfejsom za lakšu upotrebu gita, a jedan od takvih alata je Git Extensions.

### WebStorm

WebStrom je razvojno okruženje u kome se razvijaju JavaScript aplikacije.

# Razvoj 2D online platformer video igre

Naziv video igre koju pokriva ovaj rad je “Brick it”. Naziv igre je nastao tako što se kombinuju engleska reči “Brick” što znači cigla i engleska reč “Break” što znači polomiti. Ceo nivo je izrađen od kvadratnih ciklica različitih oblika koje mogu biti uništene od strane igrača. Igrač ima motivaciju da uništava nivo kako bi pronašao različite nagrade poput blaga ili različitih super moći. Igra je inspirisana igrama poput “Minecraft”, “Terraria”, “Rust”. Igra se sastoji od nivoa gde igrači treba da nađu portal koji će ih odvesti na sledeći nivo.

Igra ima mogućnost samostalnog igranja, kao i mogućnost igranja preko interneta sa drugim igračima u kooperativnom modu sa još jednim igračem ili u “Player vs Player” modu protiv jednog igrača. Igra takođe podržava više platformi, to jeste može se igrati na personalnom računaru ili Android telefonu.

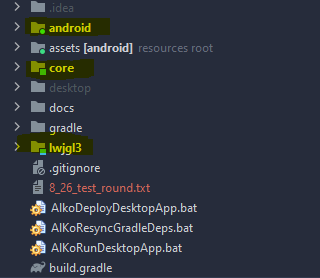
Igra se sastoji iz tri velika sistema:

* + Administrativni sistem koji obuvhata obradu podataka koji su neophodni za početak meča ( Sve klase neophodne za korisnički interfejs, delegatori ekrana i tako dalje)
  + Gameplay sistem koji obuvhata obradu svih podataka kad meč počne
  + Mrežni sistem koji obuhvata komponente odgovorne za slanje i pirmanje podataka preko mreže

## Administartivni sistem

Administrativni sistem obuhvata sve komponente od kad igrač pokrene aplikaciju pa do momenta kada počinje meč.

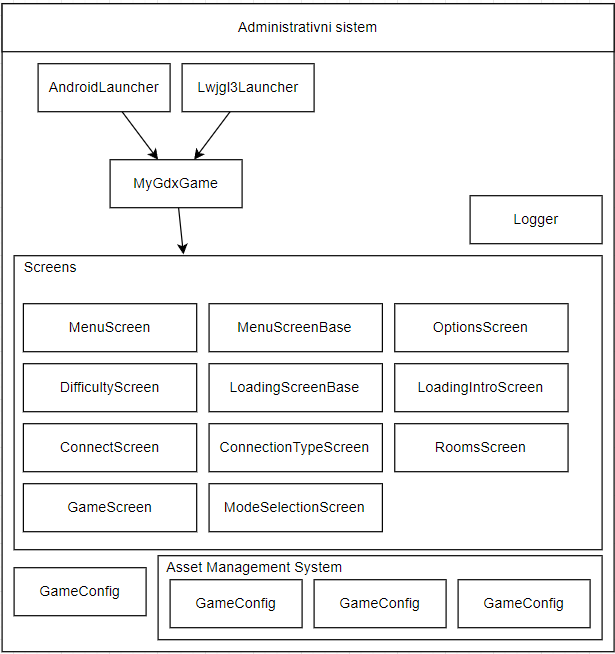
Strutura projekta (Slika 22.) se sastoji iz tri velike celine, a to folder za **android** stvari, folder za stvari vezane za računar (**lwjgl3** folder, naziv foldera je po istoimenoj biblioteci koja predstavlja bekend za računar, više o tome pročitati u poglavlju 3.1.1 “Libgdx” radni okvir) kao i **core** folder u kome se implementira sva logika vezana za igru.



Slika 22. Struktura LibGdx projekta

Administrativni sistem se sastoji od (Slika 23):

* Specijalizovanih platformskih pokretača (AndroidLauncher i DesktopLauncher)
* Game klase (MyGdxGame)
* Grupa korisničkih ekrana (Screens)
* Mandežera resursa (Asset Management System)
* Sistem za logovnaje poruka (Logger)
* Konfiguracijone klase (GameConfig)

 Slika 23. Administrativni sistem

### Platformski pokretači (AndroidLauncher i DesktopLauncher)

Kao što je već pomenuto igra podržava više različitih platformi. Za desktop računar bekend je lwjgl verzija 3. Pokretač aplikacije za desktop računar omogućava da se podesi konfiguracija kao na primer da li će vertikalno osvešavanje biti uključeno, naslov aplikacije, veličina aktivnog prozora, koje slike će se koristiti kao ikonice, brzina osvežavanja ekrana i tako dalje. Ovaj pokretač kreira aplikacioni slušalac (ApplicationListener) to jeste glavni objekat same igre koji se na slici 23. naziva MyGdxGame.

ApplicationListener u sebi ima metode poput create, resize, render, pause, resume i dispose i predstavlja ulaznu tačku LibGDX radnog okvira koji implementira životni ciklus aplikacije identičan životnom ciklusu na Androidu (Detaljnije u poglavlju 3.1.1.“Libgdx” radni okvir).

Identičan ciklus se dešava i kod pokretača za Android osim što za Android postoji i manifest fajl nazvan AndroidManifest.xml u kome se mogu konfigurisanti određene stvari vezane za android verziju aplikacije kao na primer davanje dozvole aplikaciji da pristupi internetu, što je bilo neophodno u ovom projektu.

### Game klasa (MyGdxGame)

MyGdxGame je glavna klasa samo jezgra igre i ova klasa nasleđuje klasu Game koja ima dve funcionalnosti:

1. Implementira ApplicationListener interfejs, tako da se Game klasa smatra adapter klasom. Adapter klasa služi da implementira neku klasu na način da sve implementirane metode ostanu prazne i da neko ko želi da implementira neku od tih metoda samo treba da nasledi tu klasu i da implementira isključivo metode koje su mu neophodne, a ne sve (Kada se interfejs implementira tada je po sintaksnim pravilima trebaju implementirati sve metode, što dovodi do bespotrebnog gomilanja godina u slučaju da neki interfejs treba da se implementira na mnogo mesta, a pritom sve metode nisu od značaja)
2. Služi kao delegator ekrana, to jeste menja trenutni aktivan ekran

Pored toga MyGdxGame sadrži sve objekte neophodne dok god je aplikacija aktivna, poput:

* Menadžera resursima koji je odgovoran da svi vizuelni kao i ostali resursi budu učitani u virtualnu memoriju grafičke karte
* Hendlera koji služi za parsiranje nivoa u igri
* Kreatora sveta u video igri koji po potrebi kreira određeni objekat u obliku ECS entiteta tokom samog meča
* ECS engine-a koji predstavlja pokretački motor sistema i entiteta tokom samog meča (Više o ovome u 2.3.1 Entity Component System (ECS) )
* Heš mapu referenci ekrana koji sa ovom klasom imaju odnos kompozicije, to jeste MyGdxGame klasa ih poseduje i kroz changeScreen metodu ima kontrolu nad tim da može postojati samo jedan objekat određenog prostora
* Stek na koji se smeštaju svi ekrani koji su bili aktivni barem jednom

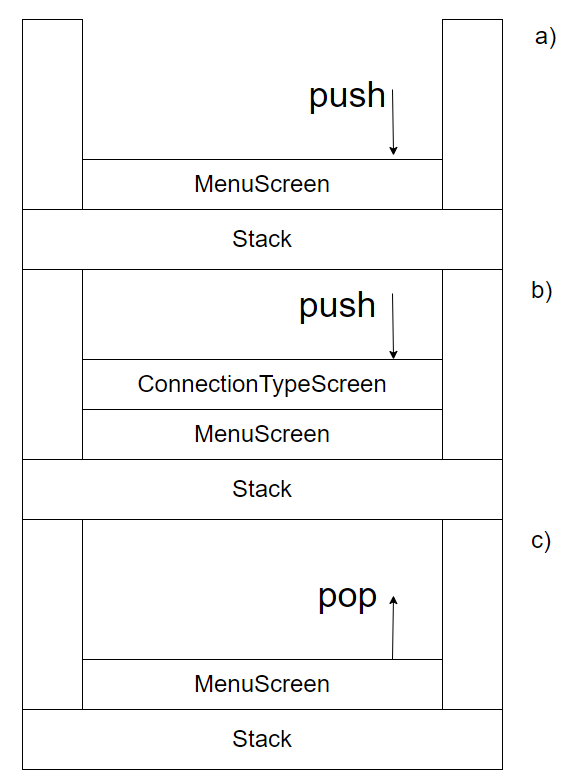
Dodatne funckionalnosti ove klase su:

1. Globalni komunikacioni bafer, to jeste mnoge klase komuniciraju preko ove klase na način što imaju ili asocijativnu vezu ili agregacionu vezu.
2. Implementira jednostavnu verziju Observer paterna. Ova klasa dobija informaciju od mrežnog sistema o tome kada je meč pokrenut, kada se promenilo stanje soba, kada je igrač napustio sobu i obaveštava sve registrovane članove kada dobije željenu informaciju

#### Delegiranje korisničkih ekrana

MyGdxGame klasa ima poputnu kontrolu nad ekranima. Kada se pokrena igra početni vidljiv ekran je MenuScreen koji se dodaje na stek i taj ekran predstavlja trenutni aktivan ekran. Ako igrač odluči da pritisne

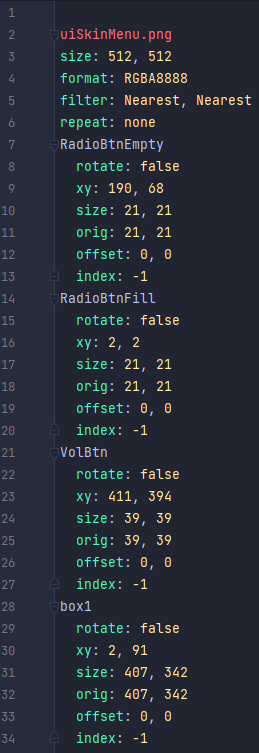
“Start Game” tada sledeći aktivan prozor postaje ConnectionbTypeScreen i on se takođe dodaje na stek. U slučaju da igrač odluči da se vrati jedan korak u nazad tada se izbacuje ekran koji je na vrhu steka a to je ConnectionTypeScreen i nakon toga se prelazi na ekran koji je na vrhu steka, a to je MenuScreen. Ovo je odličan način da se implementira funckionalnost “Back” dugmeta, jer programer ne mora da “Back” dugmetu na svakom ekranu eksplicitno kaže gde na koji prozor treba da se vrati.



Slika 24. Delegiranje ekrana a) b) Dodavanje elmenta na stek c) skidanje elementa sa steka

### Grupa korisničkih ekrana (Screens)

Radni okvir LibGDX pruža alat koji naziva TexturePacker koji omogućava da se više slika stavi na jedan papir kao rezultat se dobije velika slika kaon a primer slika pod nazivom uiSkinMenu.png (Slika25. a))

1. b)

Slika 25. a) Prikaz slike na kojoj se nalaze elementi b) .atlas fajl koji opisuje uiSkinMenu.png

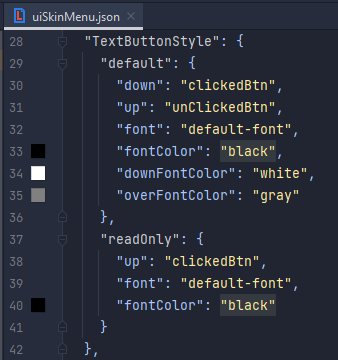
Rezultat obrade pojedinačnih slika u alatu TexturePacker je i fajl koji ima ekstenziju .atlas, to jeste u ovom slučaju uiSkinMenu.atlas (Slika 25. b)). S obzirom da radni okvir LibGDX ima biblioteku koja parsira .atlas fajl i parsirane podatke smešta u obkejat koji je tipa TextureAtlas , vrlo lako se može sa velike slike pročitati bilo koji element.

#### Prednosti korišćenja TextureAtlas-a

Da bi se razumele prednosti korišćenja metoda gde se teksture grupišu na jedno platno (jedan fajl) prvo se treba razumeti kako dolazi do iscrtavanja teksture na ekran. U radnom okviru LibGDX postoji klasa SpriteBatch koja kao ulazni parametar prima teksturu kao i poziciju gde je potrebno iscrtati u odnosu na trenutni vidljivi deo ekrana (Za to je zadužena klasa Camera). Tako da u slučaju da je potrebno iscrtati tri teksture koje se ne nalaze na u istom fajlu tada će grafičkoj karti biti poslata tri zahteva za iscrtavanje na ekran, dok u slučaju kada bi sve teksture bile u jednom fajlu bio poslat isključivo jedan zahtev za iscrtavanje.

#### Kreiranje korisničnog menija (ekrana)

Radni okvir LibGDX pruža još jednu sjajnu mogućnost, a to je da se u jednom JSON fajlu koji zapravo nije klasičan JSON fajl već se vodi kao LibGDX skin (Slika 26) namapiraju fontovi i teksture na određene elemente menija poput labela, ček boksova, tekstualnih dugmadi i tako dalje.



Slika 26. Deo fajla uiKsinMenu.json gde se opisuje izgled TextButton-a

Tako da se pri kreiranje korisničkom menija koristi Skin klasa koja parsira Skin JSON i programer može jednostavno da kreira labele, dugmadi i tako dalje kako bi kreirao meni (Slika 27.)



Slika 27. Korisnički meni, početna strana

#### Korisčniki ekrani i njihove funckionalnosti

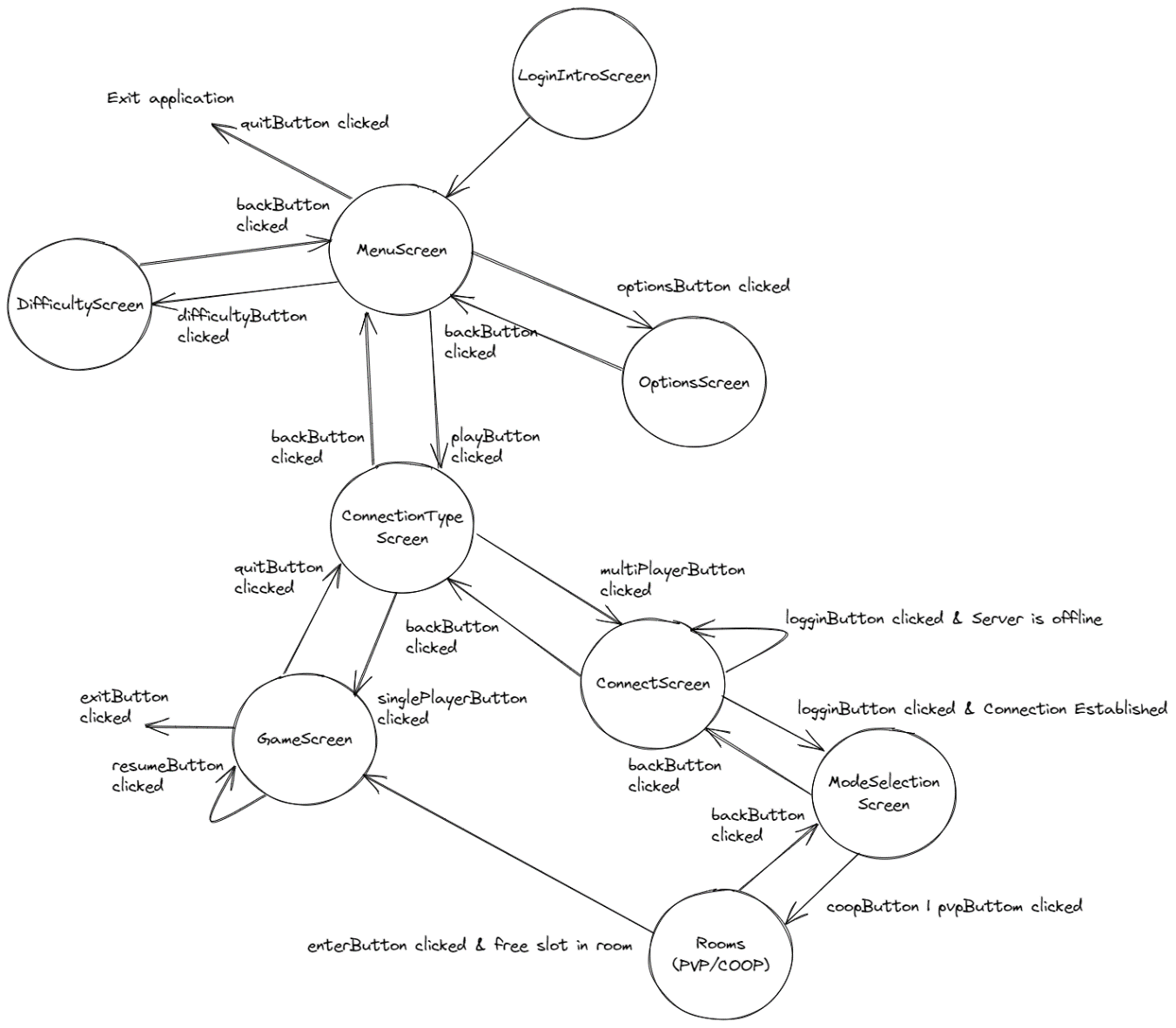
U nastavku je prikazana tabela svih ekrana kao i njihove funckionalnosti (Tabela 1.)

Tabela 1. Lista ekrana i njihove funcionalnosti

|  |  |
| --- | --- |
| **Naziv ekrana** | **Funkcionalnost** |
| MenuScreenBase | Klasa za učitavanje zvuka i pozadinske muzike |
| LoadingScreenBase | Bazna klasa za učitavanje resursa |
| MenuScreen | Klasa koja predstavlja početni ekran |
| OptionsScreen | Klasa koja predstavlja ekran sa opcijama |
| DifficultyScreen | Klasa koja predstavlja ekran gde može da se podešava težina igre za offline igranje |
| LoadingIntroScreen | Klasa za učitavanje svih potrebnih tekstura |
| ConnectScreen | Klasa koja prikazuje potrebnu formu za priključenje igrača na mrežu ili sa registraciju igrača |
| ConnectionTypeScreen | Klasa koja predstavlja ekran gde igrač može izabrati tip konekcije to jeste offline ili konekciju preko mreže |
| RoomsScreen | Klasa koja predstavlja ekran na kome je prikazano trenutno stanje soba za dati mod |
| GameScreen | Klasa koja predstavlja ekran na kome se prikazuje trenutno stanje meča u kome se nalazi igrač |
| ModeSelectionScreen | Klasa koja predstavlja ekran na kome se prikazuju modovi kao što su kooperativni mod, kao i mod gde se igrač bori protiv drugog igrača |

#### Dijagram stanja promene korisničkih ekrana

Kada se pokrene aplikacija prvo stanje u koje se dođe je LoginIntroScreen koji je zadužen za učitavanje svih potrebnih resursa za samu aplikaciju koji se nalaze u folderu assets (Slika 22.), gde se nakon toga bez bilo kakvih uslova prelazi u početni ekran pod nazivom MenuScreen. Odatle igrač može da pristupi određenim podešavanjima prelazeći u OptionsScreen, može da promeni težinu igre prelazeći u DifficultyScreen (Ova mogućnost još nije u potpunosti implementirana), takođe igrač pritiskom na play dugme prelazi u ekran gde ima mogućnost da bira konekciju, to jeste ako želi da igra sam, može da izabere offline konkeciju gde odmah nakon toga počinje meč, dok ukoliko želi da igra sa drugim igračima može da izabere online konektivnost. U slučaju da je igrač izabrao onlajn konekciju tada se prelazi u ConnectScreen ekran gde igrač ima opciju da unese kredencijale svog naloga ili u slučaju da nema svoj nalog da se registruje. Nakon logovanja, sledeći ekran koji se prikazuje je ModeSelectionScreen gde igrač može da bira mod koji želi (COOP/PVP) i po izboru moda se dolazi do ekrana Rooms gde je prikazano stanje svih soba (U koje sobe se može ući) i nakon odabira sobe počinje meč tako što se prelazi u GameScreen ekran.



Slika 28. Dijagram stanja promene korisničkih ekrana

### Menadžer resursima (Asset Management System)

Ulazno/Izlazne operacije poput čitanja ili upisa sa diskova su veoma spore sa razliku od operacija računanja. U slučaju da su te operacije implementirane kao blokirajuće to znači da će ceo program da stane sa izvršavanjem dok se ne završi operacija čitanja/pisanja na disk. Tako da se kod video igara učitavanje resursa vrši pri samom pokretanju igre. Za vreme učitavanja resursa korisniku je prikazan takozvani ekran za učitavanje (Loading Screen). Kod korisnika koji pokreći neku veliku modernu igru sa HDD-a to učitavanje može da traje i po nekoliko minuta. Alternativa su brži diskovi pod nazivmo SSD.

Asset menadžer sistem služi kako bi učitao sve potrebne resurse koji se nalaze na disku i smešta ih u određenu memoriju (Teksture se učitavaju na virtualnu memoriju na grafičkoj karti, dok se ostali resursi učitavaju u RAM ). Asset menadžer koristi AssetDeskriptore koji opisuju željeni resurs koristeći ime i tip, tako da interni handler fjalova jednostavno može da proveri da li željeni resurs postoji u internom folderu za resurse i da se nakon toga pristupi učitavanju tog fajla

### Sistem za logovnaje poruka (Logger)

Tokom razvoja i održavanja aplikacije postoji potreba da se vidi u koja je sve stanja aplikacija ušla i koje akcije su se desile. Najlakši način je štampanje poruka. Problem koji se tu javlja jeste preveliki broj štampanih poruka, takovana šuma poruka koja može da uzrokuje više problema:

* U velikoj šumi poruka postaje teže naći poruke od interesa
* U slučaju da se poruke šalju na internet (Neki internu bazu podataka) višak poruka povećava potrošnju kao i što povećava log fajl, pa i samo slanje na internet traje duže
* Ispis poruka se vrši iz serijski, dakle jedna po jedna poruka tako da to smanjuje performanse programa koji su napisani da rade paralelno (Da se neko parče koda izvršava na više procesorskih niti)

Rešenje za ovaj problem je dodeljivanje poruci nivo bitnosti, takozvani log level. Radni okvir Libgdx ima ugraćen Logging sistem koji omogućava da se poruka označi sa nivoima: NONE, ERROR, INFO, DEBUG. Tako da programer koji razvija aplikaciju će koristiti ERROR nivo za poruke koje ispisuju kada je aplikacija ušla u neko pogrešno stanje, to jeste kada je došo do greške. INFO nivo se koristi kada je poruka informativnog karaktera i DEBUG nivo se koristi za poruke koje daju više informacije u odnosu na poruke koje su označene sa INFO nivoom i obično se koriste pri razvoju aplikacije ili tok proučavanja kako je došlo do nekog problema. Nivoi poruka imaju svoje vrednosti od najniže ka najvišoj (NONE = 0, … DEBUG = 3) tako da ako se sistemski log level podesi na ERROR, tada se ispisuju samo poruke označene sa ERROR, dok ako je sistemski log levle podešen na INFO, tada se ispisuju poruke označene sa INFO i ERROR i tako dalje. Na ovaj način se mogu filtrirati poruke u sistemu. Takođe postoji mogućnost da se poruke filtriraju na nivou klase, tako što se kreira interni logger za klasu, pa se na taj način poruke mogu finije filtrirati, to jeste na nivou same klase. Preporuka da se kod stabilne verzije programa koristi ERROR nivo.

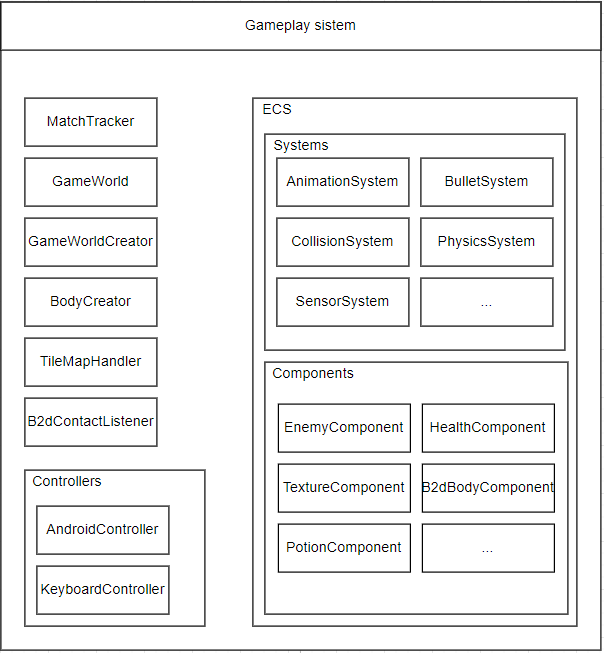
### Konfiguracijone klase (GameConfig)

Konfiguraciona klasa služi za čuvanje konstanti koje se koriste dalje u kodu. Potrebno je izbegavati takozvane magične brojeve, to jeste da se u kodu koriste numeričke vrednosti bez objašnjenja. Vrlo lako može doći do slučaja da na tom kodu rade drugi programeri koji nisu upućeni zašto su baš te vrednosti upotrebljene, a drugi razlog zašto je dobro sve konstantne vrednosti smesititi u konfiguracioni fajl je taj što u slučaju da se koriste magični brojevi na više mesta, prilikom izmene te vrednosti, na svakom mestu mora da se napravi izmena kako ne bi došlo do pravljenja problema u radu aplikacije.

## Gameplay sistem

Gameplay sistem (Slika 29.) sačinjavaju dve velike grupe:

* Moduli koji se nalaze u sklopu ECS-a, poput ECS komponenti koje predstavljaju kontenjere za podatke kao i ECS sistemi koji predstavljaju specijalizovane funckionalnosti gde svi sistemi prođeani u jedan red i sinhrono se izvršavaju prilikom svake iteracije glavne igračke petlje.
* Svi ostali moduli koji nisu deo ECS grupe poput MatchTracker klase, BodyCreator, TileMapHandler, Input kontroler klasa i ostalih. To su funcionalni moduli koji pružaju određeni servis po pozivu ili asinhrono poput B2dContactListener klase ili Android/Keyboard kontrolera



Slika 29. Gameplay sistem osnovni prikaz

### ECS grupa

Kao što je već pomenuto u sekciji 2.3.1 Entity Component System (ECS), ECS je programerski patern koji ima mnogo benefita i najkorišćeniji je u razvoju video igara. Sastoji se od komponenti (Tablea 2) koji predstavljaju isključivo kontenjere sa podacima, entitija (Tabela 3) koji su skup komponenti i sistema (Tabela 4) koji manipuliši nad određenim entitijima.

#### ECS komponente

Tabela 2. ECS Lista komponenti

|  |  |
| --- | --- |
| **Naziv komponente** | **Podatke koje sadrži** |
| AIEnemyComponent | Služi samo kao family filter (Objašnjenje u 4.2.1.3), sadrže je samo AI protivnici |
| AnimationComponent | Sadrži mapu objekata tipa Animation |
| B2dBodyComponent | Sadrži informacije vezano za telo koje se predstavlja objekat u Box2D svetu (Simulacija fizike) |
| Box2dLocation | Sadrži poziciju Box2D tela u World jedinici |
| BrickComponent | Sadrži informacije vezane sa jednistveni sloj kome pripada (Sloj platforma) u mapi (Više o slojevima i Tiled mapama u 4.2.2) |
| BulletComponent | Sadrži informacije vezane za metak poput brzine kretanje po x i y osi, preostalo vreme života kao i reference na Box2D telo koje je isplailo metak (Kako vlasnik metka ne bi mogao da bude povređen od strane spostvenog metka) |
| CharacterStatsComponent | Sadrži analitičke i vizuelne informacije o igraču poput broja preostalih života |
| CollectibleBasicArrayComponent | Sadrži niz Boolean promenljivih koji označavaju da li igrač ima odreženu super moć. |
| CollectibleBasicComponent | Sadrži informacije vezane za jedinstveni sloj kome pripada (Sloj stvari koje mogu da se pokupe) |
| CollisionComponent | Sadrži reference na entitet sa kime je vlasink CollisionComponent doživeo sudar |
| CollisionEffectComponent | Sadrži merač vremena koliko je entiet sa ovom komponentom proveo vremena u životu |
| ControllableComponent | Služi samo kao family filter. Ova komponenta je od važnosti sistemima koji mogu da budu kontrolisani od strane igrača. Izbacivanjem ove komponente tokom igre vrši se onemogućivanje objekta da se kreće što može biti vrlo korisno na primer ako je igrač zaleđen neko vreme. |
| ControlledInputComponent | Sadrži Boolean niz koji signalizira da li je neko dugme pritisnuto |
| CoolDownComponent | Sadrži informacije o tome koliko je vremena prošlo od nekog događaja. Koristise kod svih objekata koji ispaljuju magije kako bi se ogranila frekvencija kojom objekti mogu da ispaljuju magije |
| DirectionComponent | Sadrži informaciju o smeru objekta |
| EnemyComponent | Sadrži informaciju o tipu protivnika, o tome ko je primećena meta, brzinu kretanja, broj koraka za skriptirane protivnika i tako dalje |
| HealthComponent | Sadrži informaciju o broju životnih poena |
| HurtableObjectComponent | Sadrži informaciju o sloju (Smrtonosna entitet) |
| LimitAreaComponent | Sadrži informaciju o sloju kome pripada, a to je limitirajući sloj koji ne dozvoljava igraču da prođe kroz taj objekat i na taj način igrača ograničava da ne može izaći sa nivoa i tako upasti u nedefinisano stanje. |
| LocalInputComponent | Služi samo kao family filter i sadrže je samo lokalni igrači |
| PlayerComponent | Sadrži informacije vezane za igrača poput refrence na kameru jer kamera prati kretanje igrača, tip igrača (Online ili Local) i tako dalje |
| PortalComponent | Sadrži informaciju o sloju kome pripada, a to je portal sloj, to jeste entitet sa ovom komponentu predstavlja objekat sa kojim kad se dođe u kontakt, mapa se završava. |
| PotionComponent | Sadrži informaciju o sloju kome pripada, a to je sloj napitak. |
| RemoteInputComponent | Služi samo kao family filter i sadrže je samo online igrači |
| SensorComponent | Sadži reference na vlasnika senzora, kao i na entitet sa kome je sensor došao u kontakt.  Senzori su Box2D tela koja se pri kontaktu ne razrešavaju kao sudari, već se dobija informacija o kontaktu. |
| StateComponent | Sadrži informaciju o stanju poput normalno, skok, padanje, kretanje,… |
| SteeringComponent | Sadrži informacije poput stanja kretanja AI entiteta poput lutanje, traženje, bežanje i tka dalje. Takođe sadrže informaciju o ubrzanju AI entiteta |
| TextureComponent | Sadrži refrencu na teksturu koju je potrebno iscrtati |
| TiledMapComponent | Sadrži reference na Map texture koju je potrebno iscrtati, kao i informaciju kom sloju na mapi pripada tekstura |
| TransformComponent | Sadrži informaciju o poziciji u pikselima |
| TypeComponent | Sadrži informaciju o tipu entieta (Igrač, protivnik, itd) |

#### ECS entiteti

Entiteti se kreiraju pozivanje metoda GameWorldCreator klase, a lista entiteta I njihova začenja su prikazana u tabeli ispod:

Tabela 3. ECS Lista Entiteta

|  |  |
| --- | --- |
| **Naziv entiteta** | **Objašnjenje** |
| FlyingBat | Leteći Ai protivnik koji oko sebe ima senzor (ViewArea senzor) koji mu pomaže da uoči igrača. U momentu kada ga uoči, to jeste kada igrač dođe u kontakt sa senzorom tada biva napadun od strate FlyingBat-a. |
| ViewArea | Senzor koji pomaže protivnicima da uoče protivnika i spram toga reaguju na određeni način |
| Portal | Entitet koji predstavlja objekat koji označava kraj nivoa |
| Platform | Entitet koji predstavlja objekat po kome igrači i protivnici mogu hodati |
| HurtableObject | Entitet koji predstavlja smrtonosan objekat |
| LimitAreaObject | Entiet koji sprečava igrače da izađu van definisanog nivoa |
| Player | Entitet koji predstavlja igrača |
| Enemy | Entitet koji predstavlja protivnika |
| BasicCollectible | Entitet koji predstavlja super moć koja može da se pokupi |
| Potion | Entitet koji predstavlja napitak koji igrač može pokupiti |
| Bullet | Entitet koji predstavlja magičnu loptu koju igrači i neki protivnici mogu da ispale |
| ExplosionEffect | Entitet koji predstavlja ekspolzivni efekat |

#### ECS sistemi

Radni okvir LibGDX pruža implementaciju ECS pod nazivom Ashley. Osnovna klasa koja predstavlja sistem u ovoj biblioteci se naziva EntitySystem. Mada se obično koristi klasa koja nasleđuje ovu klasu i naziva se IteratingSystem (Slika 30.). Tokom svake iteracije igračke petlje (Odvija se kroz LibGdx životni ciklus, tačnije kroz render() metodu) poziva se engine update() metoda koja prolazi kroz niz registrovanih sistema i poziva njihove update() metode (s1.update(), s2.update(), i tako dalje). Iterativni sistemi imaju unapređenje update() metode tako da pri pozivanje ove metode prolazi se kroz svaki entitet koristeći metodu processEntity(Entity entity) koji odgovara podešenim family filterima datog sistema. Family filter je takođe unapređenje iterativnih sistema. Suština ovog filtera je da se iterira samo kroz odgovarajući skup entiteta. Svaki sistem overrajduje processEntity metodu i implementira logiku za specijalizovanu funckionalnost.

Primer EnemySystem ima filter:

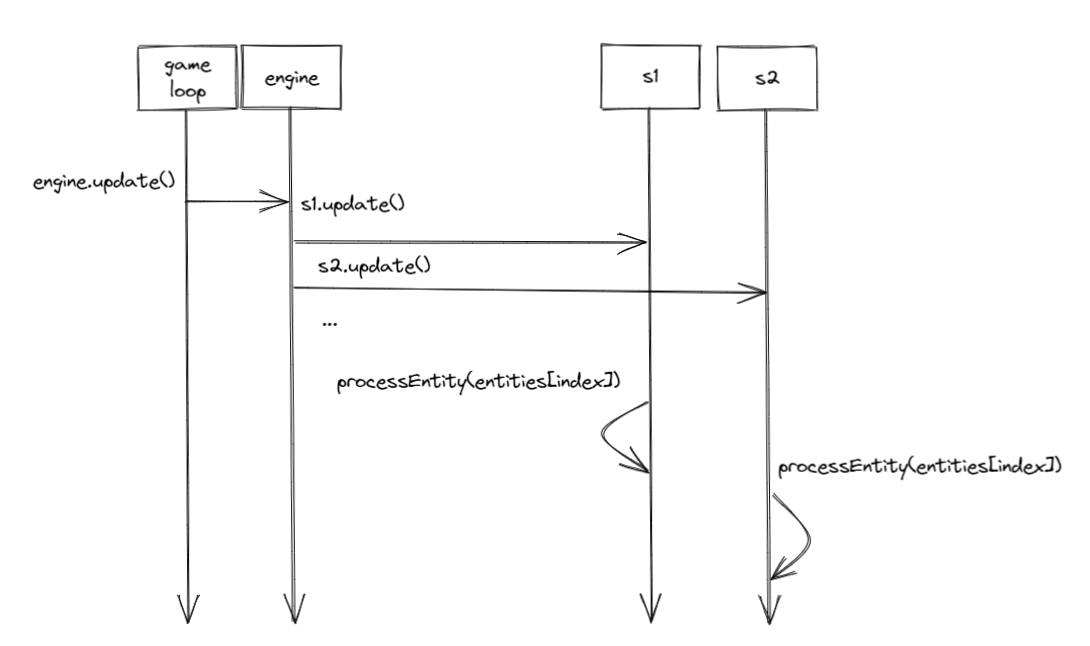
super(Family.all(EnemyComponent.class).get());

Znači da će sistem prolaziti kroz sve entitete koje moraju imati EnemyComponent

Primer HealthManagerSystem ima filtere:

super(Family.all(HealthComponent.class, CollisionComponent.class, B2dBodyComponent.class).get());

Znači da će sistem prolaziti kroz sve entitete koje moraju imati HealthComponent, CollisionComponent i B2dBodyComponent, a to su igrači i protivnici

 Slika 30. Iteracija engine-a kroz sisteme

Lista sistema prikazana je u tabeli ispod.

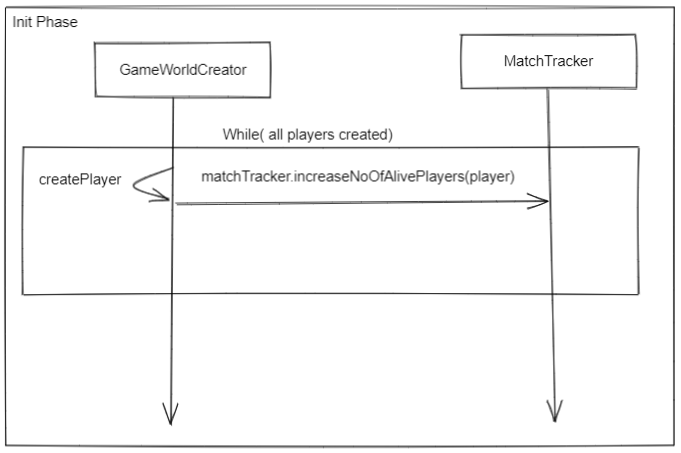
|  |  |
| --- | --- |
| **Imena sistema** | **Opis funkcionalnosti** |
| AnimationSystem | Delegira trenutnu aktivnu teksturu |
| B2dContactSystem | Entitetima između kojih je došlo do kontakta proseđuje informaciju o entitetu sa kojim su došli u kontakt |
| BulletSystem | Proverava stanje metka ( leteće magične lopte ), tačnije poluživot kao i da li je metak u međuvremenu uništen |
| CharacterStatsSystem | Osvežava trenutnu poziciju i broj preostalih života entitetima koji sadrže CharacterStatsComponent |
| CollectibleBasicManagerSystem | Obrađuje zahteve igrača koji imaju neku od super moći (dvostruki pucanj, dugački skok i udarac nogom o zemlju) |
| CollisionEffectsSystem | Meri vreme poluživota entiteta koji predstavljaju ekplozivni efekat. Ekplozivni efekat treba da nestane kada se sekvencijalno prikažu sve sličice u vidu animacije |
| CollisionSystem | Razrešava sudare između dva tela (Informacije prima od B2dContactSystem). |
| DataReceivingSystem | Sistem zadužen za primanje i obrađivanje informacija koje su pristigle preko interneta i tiču se samih igrača |
| DataTransmittingSystem | Sistem zadužen za obrađivanje i slanje informacija preko interneta i tiču se samih igrača |
| EnemyDataReceivingSystem | Sistem zadužen za primanje i obrađivanje informacija koje su pristigle preko interneta i tiču se protivnika. |
| EnemySystem | Obrađuje poziciju i ponašanje protivnika |
| HealthManagerSystem | Izvršava akcije poput smanjivanja životnih poena entitetima, povećavanje životnih poena kao i uništavanje objekata |
| InputManagerAndroidSystem | Sinhroni modul koji obrađuje pristigle komande od asinhronog AndroidController modula koji čita komande korisnika koji koristi android uređaj |
| InputManagerSystem | Sinhroni modul koji obrađuje pristigle komande od asinhronog KeyboardController modula koji čita komande korisnika koji koristi tastaturu |
| PhysicsDebugSystem | Iscrtava pozicije na kojima se nalaze Box2D tela u služi isključivo u svrhe debagovanja. |
| PhysicsSystem | Provara da li je entitet koji predstavlja Box2D telo živ, ako nije obrađuje uništavanje tela. |
| PlayerControlSystem | Obrađuje novopristigle komande od strane InputMangerSystem-a ili InputManagerAndroidSystem-a |
| RenderAndroidControllerSystem | Renderuje kontrolnu šemu za  Android |
| RenderCharacterHudSystem | Renderuje informacije dobijene od CharacterStatsSystem, u ovoj verziji igre je to broj preostalih života svih karaktera. |
| RenderGameHud | Renderuje korisnički interfejs tokom meča. Na primer dugme za pauzu |
| RenderingSystem | Renderuje sve entitije koji imaju  TextureComponent poput igrača, protivnika, magične lopte i tako dalje. |
| RenderTiledMapSystem | Renderuje sve entitije vezane za TiledMap nivo poput tekstura platformi, entitija koji mogu da se pokupe |
| SensorSystem | Proverava da li je došlo do kontakta između senzora i igrača. Kada dođe do kontakta dešava dešava se određena operacija u zavisnosti ko je vlasnik senzora. |
| SteeringSystem | Sinhrni sistem koji ažuriraju stanje SteerinComponent entiteta  Napomena: u SteerinComponent nisu poštovana pravila ECS paradigm tako da ta komponenta nije samo kontenjer za podatke već ima implementirane neke funkionalnosti za veštačku inteligenciju |

### Ostali moduli

U ovu grupu spadaju svi ostali moduli. Za razliku od ECS modula koji predstavljaju sinhrone sistem, ostali moduli su aktivni samo po pozviu ili su u pitnaju asinhroni sistemi

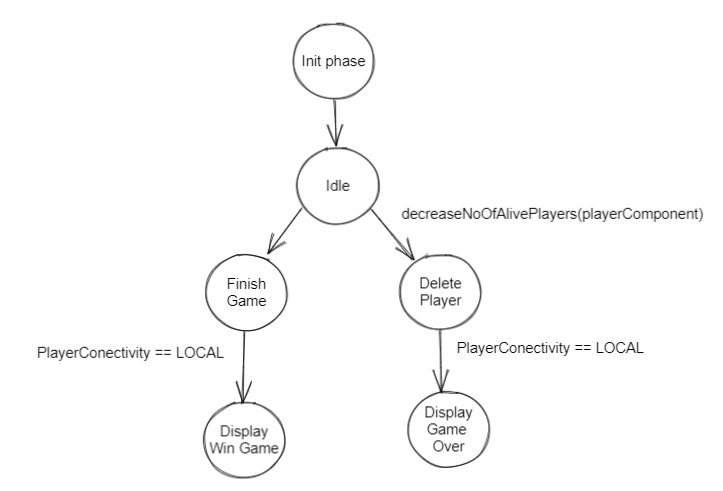
#### MatchTracker

MatchTracker je modul koji je zadužen da prati da li je meč gotov ili nije. Pri inicijalizaciji meča se između ostalog kreiraju igrači i samim tim i faza inicijalizacije MatchTracker-a (Slika 31.)



Slika 31. Init faza MatchTracker-a

Nakon modul čeka zahteva da smanji broj igrača u igri, a to se dešava kada igrač pogine i PhysicsSystem pre nego što obriše entitet prvo javi MatchTracker-u da je igrač poginuo. Druga mogućnost je da je igrač ušao u portal za sledeći nivo i tada CollisionSystem razrešava sudar igrača i portala tako što šalje zahtev MatchTracker-u da je igrač pobedio. Pa se spram zahteva poziva keran koji obaveštava igrača da je pobedio ili izgubio.



Slika 32. Matchtracker obrađuje zahteve

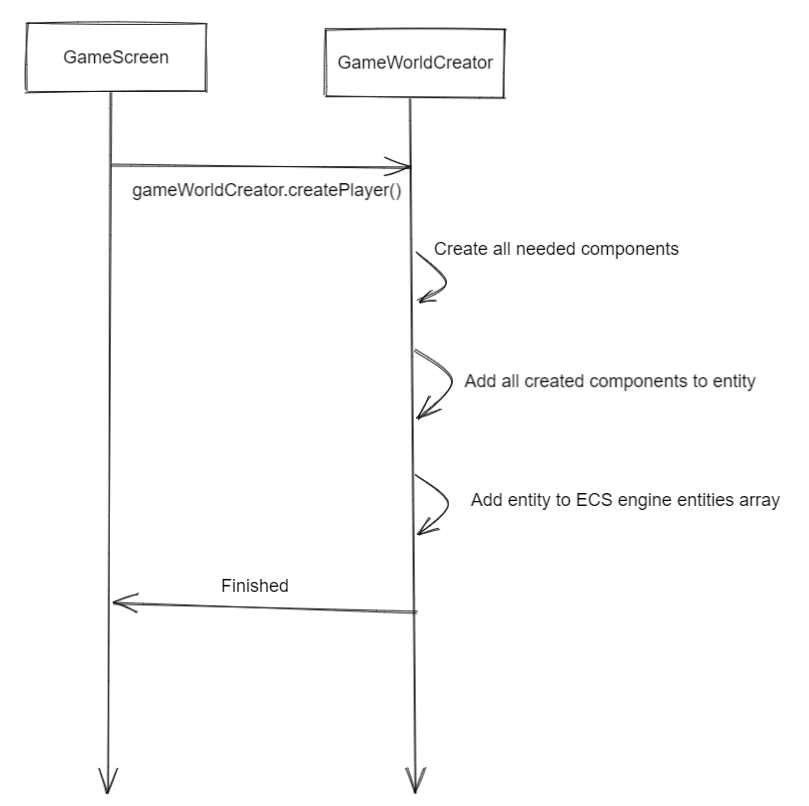
#### GameWorld

GameWorld klasa predstavlja kontenjer klasu koja u sebi čuva sledeće podatke:

* Box2D world objekat klase World koja se koristi za skladištenje svih Box2D tela kao i za simulaciju Box2D sveta
* Niz referenci entiteta koji predstavljaju igrače. Izvorne reference na objekat u memoriji se nalaze u ECS engine-u, ovde se samo prosleđue referenca kako bi se pristupilo Player Entitetima bez iteracije kroz ECS engine niz entiteta.
* TiledMap-u to jeste nivo u igrici
* Nazivi slojeva iz TiledMape koji se koriste za parsiranje

#### GameWorldCreator

GameWorldCreator klasa je zadužena za kreiranje game objekata tačnije ECS entiteta. Entitete koje GameWorldCreator može da kreira se nalaze u tabeli 3. Klijent GameWorldCreator-a, na primer GameScreen pozove metodu za kreiranje određenog entiteta, na primer createPlayer() (Slika 33.)



Slika 33. Matchtracker obrađuje zahteve

Kreiranje objekta se dešava u tri faze:

1. Kreiranje svih potrebnih komponenti i setovanje podataka u te komponente. Na primer za kreiranje igrača su potrebne sledeće komponente:

* B2dBodyComponent – jer Igrač treba da predstavlja dinamičko telo u Box2D svetu
* PlayerComponent – kako bi igrač dobio jedinstveni igrački ID kao i tip, na primer lokalni igrač
* TransformComponent – Kako bi pozicija igrača bila predstavljena u pikselima
* TextureComponent – jer igrač takođe treba da bude predstalvjen vizuelno
* AnimationComponent – kako bi igrač bio animiran
* TypeComponent – Svaki entitet ima tip, ova komponenta je tipa player
* StateComponent – Početno stanje igrača je normal
* CharacterStatsComponent – Igrač poseduje informacije o broju života
* ControllableComponent – Igrač ima mogućnost da sluša komande poslate sa tastature ili android kontrolera
* CoolDownComponent – Igraču je limitirana frekvencija kojom može da ispaljuje magične kugle
* LocalInputComponent – Ako je igrač lokalni onda mu se dodeljuje ova komponenta
* RemoteInputComponent – Ako je igrač remote onda mu se dodeljuje ova komponenta

1. Dodavanje svih komponenti u entitet
2. Dodavanje entiteta u niz entiteta koji se nalazi unutar ECS engiene-a

#### BodyCreator

BodyCreator klasa služi za kreiranje tela u Box2D svetu. Odgovara na zahteve GameWorldCreator klase.

Bitni pojmovi:

BodyDef, je definicija tela koja sadrži:

* Tip tela koji može biti:

Statičan – Statična tela se ne mogu pomeriti, niti na njih deluje gravitacija

Dinamičan – To su tela koja se kreći i na njih deluje svaka vrsta sila

Kinematičan – To su tela nad kojim se nema kontrola ali na njih deluje sila

* Pozicija tela
* Linearno, ugaono ubrzanje i tako dalje

Fixture, je klasa koja sadrži:

* Oblik koji može biti okrugli, kvadratni i tako dalje
* Filter koji sadrži polja:

categoryBits – Predstavlja unikatan Collision ID neke grupe objekata. Na primer objekti koji su tipa igrač imaju categoryBits 1

maskBits – Predstavlja skup categoryBits-a sa kojima telo može doći u kontakt

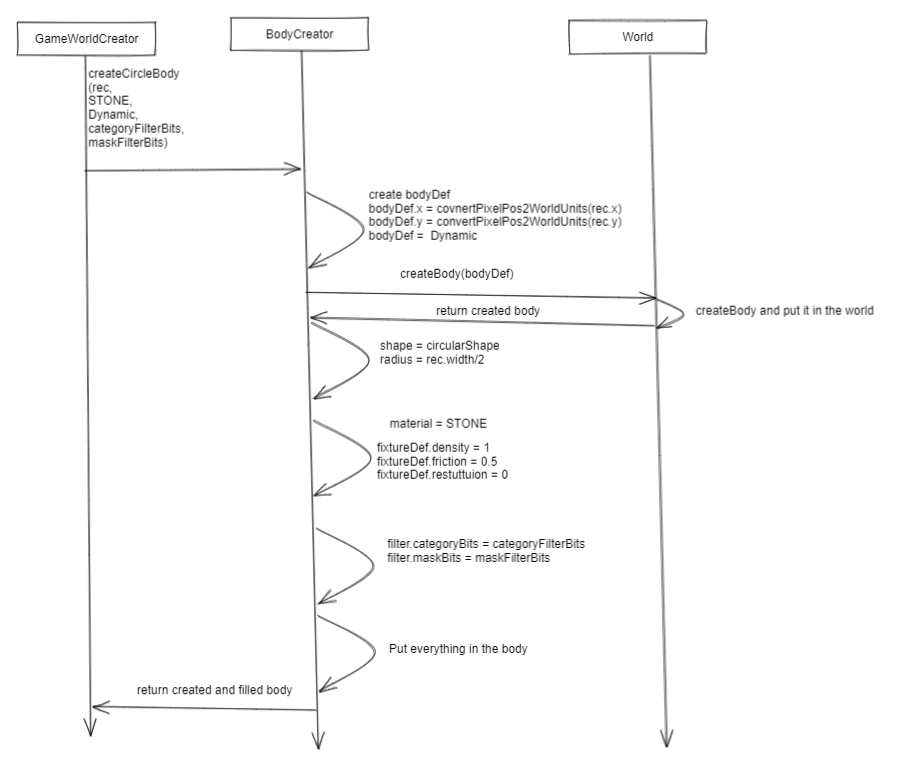
* UserData je referenca na Java objekat, tako da je vrlo zgodno da se tu stavi šta god je bitno, na primer pri kreiranju tela u filterData je stavljana referenca na Filter kako bi CollisionSystem na osnovu te reference znao koja dva tela su se sudarila

FixtureDef, je klasa koja sadrži:

* Density to jest gustinu tela
* Friction to jest trenje tela
* Restituion to jest elastičnost tela

Primer kreiranja Box2d tela igrača (Slika 34.) gde BodyCreator dobije zahteva od GameWorldCreator-a da treba napraviti Box2D telo. Ulazni parametri su rectangle, tacnije kontenjer klasa koja u sebi informacije o duzini i visini objekta kao i x i z poziciju, materijal od koga treba telo biti napralvjeno (STONE, kamen), kojeg je tipa telo (Dinamično jer je u pitanju igrač), categoryFilterBits (za igrača je to 10 binarno, to jeste 2 decimalno), maskFilterBits (Igrač može da ostvari kontakt sa platformom, stvarima koje može da pokupi, sa napitkom, sa portalom, sa senzorima protivnika, sa magičnom kuglom, sa objketima koji limitiraju kretanje, kao i sa smrtonosnim objketima).

Prvo se kreira definicija tela (bodyDef) gde se setuje pozicija tela (tako što se pozicije pragaonika konvertuju u world unite). To se radi zbog toga što je u igri na primer definisano da je karakter veličine 200 piksela x 200 piksela a u Box2D svetu će to značiti da je veličine 200 metara x 200 metara tako da se na taj način dobija veoma teško telo pod uslovom da se koristi realistična gravitacija, kao na planeti zemlji. Tako da pozicije kao i dužina/visina koja dolaze iz sveta koji nije Box2D konvertuju tako što se podele sa dogovorenim koeficijentom takozvanim PPM koji je u ovom projektu 200 piksela. Na taj način se dobija telo veličine 1 metar puta 1 metar. Kada se čitaju dimenzije i pozicija tela iz Box2D sveta tada se sve množi sa PPM. Takođe se setuje i tip tela, to jeste u ovom slučaju je telo dinamičko.

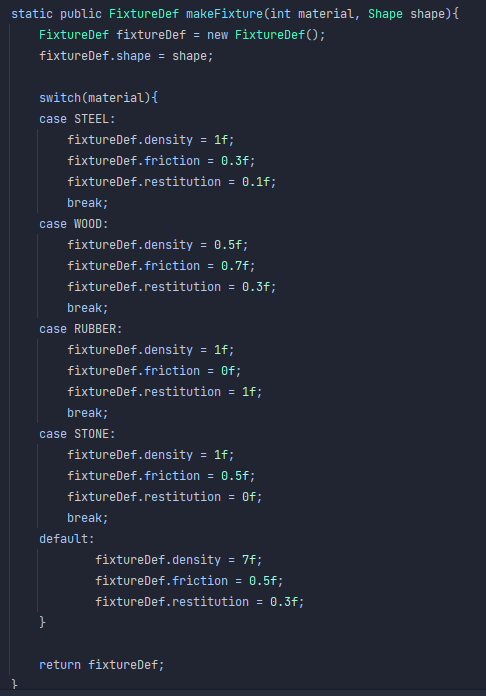


Slika 34. Primer keriranja tela za entitet igrača

Nakon toga se pošalje zahteva Box2D world klasi da napravi telo sa tom definicijom tela i da ga smesti u svoj niz Box2D tela i da izgrađeno telo vrati.

Nakon toga se kreira oblik tela to je u ovom slučaju krug sa prečnikom rec.width/2.

Nakon toga se kreira fixtureDef sa tim što je materijal kamen. Setovanjem gustine, trenja i elastičnosti mogu se simulirati kretanja različitih materijala (Slika 35.)



Slika 35. Metoda koja vraća FixtureDef za određeni material

Nakon toga se setuju filteri za telo i tako potpuno kreirano i inicijalizovano telo se vraća klijentu koju je zatražio kreiranje tela.

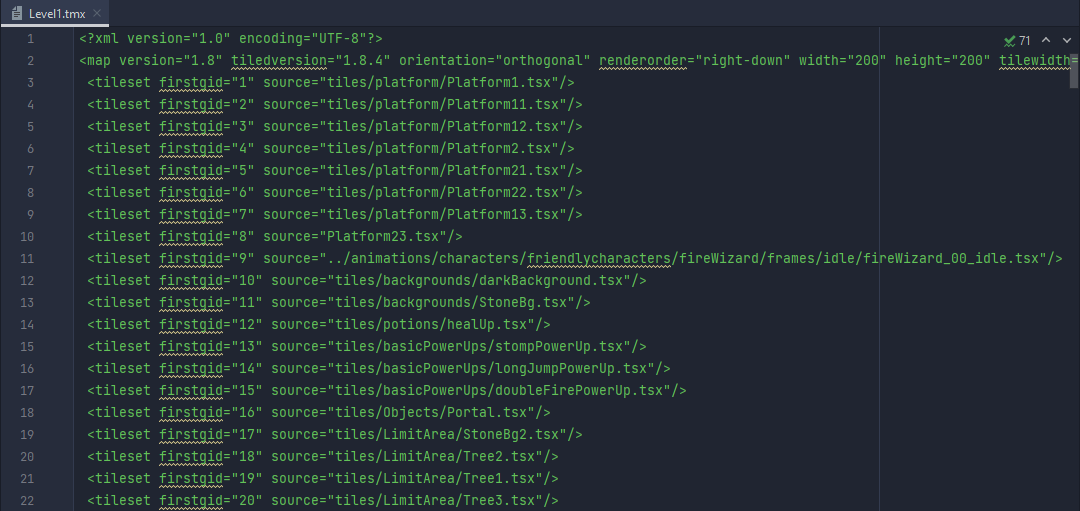
#### TileMapHandler

TileMapHandler je klasa koja je zadužena za učitavanje TiledMap-e. Kao i za renderovanje mape tako što prosleđuje reference svog renderera sistemu RenderTiledMapSystem koji zatim renderuje entite koji moraju da imaju barem jednu od ovih komponenti

(BrickComponent, CollectibleBasicComponent, PotionComponent)

TiledMap-a iz perspective TiledMap loadera nije nita drugo nego xml koji se parsira (Slika 36.)

Gde se mapa sastoji od tsx elementa koji su reprezentacija slike u xml-u. Dakle svaka slika korištena pri kreiranju mape mora da ima svoj tsx fajl i obrnuto (Ako se obriše neka slika koja je korištena pri kreiranju mape, taj element više nije validan iako ima svoj tsx fajl što je i logično jer je tsx fajl samo mapa koja opisuje sliku kao i njenu lokaciju)



Slika 36. Deo fajla koji predstavlja Level1 u xml formatu

#### B2dContactListener

B2dContactListener je asinhroni modul koji prati pozicije svih tela tokom meča i zadužen je da prijavi kada se dogodi kontakt izmeđi dva tela. Kada se dogodi kontakt, Fixture-i oba tela se smeštaju u niz kao bufferOfFixtures iz razloga kako bi se događaj sinhronizovao što je jako bitno pri emitovanju događaja preko mreže. U slučaju kada u projektu postoje sinhroni i asinhroni sistemi što je slučaj u svakoj igri, potrebno je “usporiti” asinhrone sisteme da ne bi došlo do problema.  
Tokom rada na ovom projektu dok nije postojala sinhronizacija B2dContactListener-a dešavalo se da se ne registruje kontakt između igrača i nekog drugog objekta priliko stres testova. Desi se situacija da B2dContactListener pozove beginContact nakon što se u trenutnoj iteraciji rad Collision sistema završio i na taj način se propusti kontakt između dva tela.

#### Kontroleri

Postoje dva kontrolera, na računaru so to tastatura, a na android je to android kontroler u vidu vizualne šeme s obzirom da se na android telefonu igra dodirujući ekran. Ovi sistemi su takođe asinhroni i svakom promenom stanja tastura (dugme pritisnutno/dugme nije pritisnuto) se ažurira bafer abInputCommandList koji se nalazi u komponenti ControlledInputComponent svakog igrača tako da PlayerControlSystem-u, to jeste sistemu za kotrolisanje igrača nije bitno koji uređaj igrač koristi kako bi zadavao komande.

## Mrežni sistem

Mrežni sistem video igre je zadužen za slanje i primanje podataka preko mreže u slučaju da igrač učestvuje u online meču. ClientHandler je modul koji je zadužen za slanje i primanje poruka.

ClientHandler čine:

* Soket – soket koji pripada socket.io biblioteci (Poglavlje 3.1.6) baziran na web soket protokolu.
* IO.Options – je takođe klasa koja pripada socket.io biblioteci i služi za konfigurisanje opcija poput broja pokušaja da se uspostavi konekcija kao i drugih korisnih opcia
* Bafer za skladištenje poruka koje treba da se pošalju serveru
* Bafer za skladištenje poruka koje su priljene od strane servera

S obzirom da su soketi implementirani kao event-driven, ClientHandler je odgovoran za registrovanje kalbek funkcija za svaki događaj koji je poznat klijentu. Imena događaja treba da budu intuitivna kako bi se već na osnovu imena moglo zaključiti o čemu se radi. Takođe i dijagami protoka podataka mogu biti od velike pomoći kada postoji veliki broj događaja u sistemu.  
Neki od događaja koji su poznati klijentu su:

* “assignID2Player” – To je poruka u kojoj server šalje id za igrača koji je zahtevao da uđe u sobu i pokrene meč.
* “playerFiredMagic” – Server javlja klijentu da je onlajn igrač ispalio magiju i ova poruka sadrži informacije poput pozicije na kojoj je onajln igrač ispalio magiju, kao i brzinu kretanja magije (Pozitivna brzina po x osi je smer na desno, a negativna brzina po x osi je smer levo)
* “createAllEnemies” – Server javlja klijentu da je potrebno da se kreiraju svi protivnici na mapi (Trenutak kada su svi igrači ušli na mapu, to jeste kada je završena faza inicijalizacije igrača) i tako dalje.

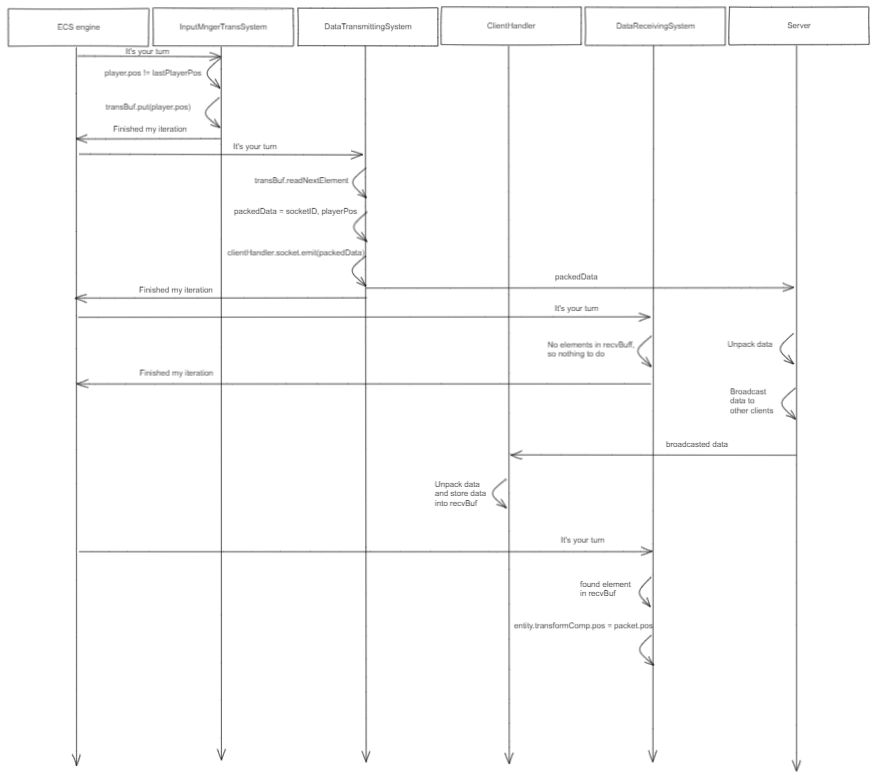
Takođe je potrebeno definisati i događaje koje klijent treba da pošalje serveru, a neki od tih događaja su:

* “magicFired” – predstavlja poruku kojom se javlja serveru da je igrač ispalio magiju i sadrži iste informacije kao u gore pomenutoj događaju “playerFiredMagic”
* “goOutFromRoom” – Šalje se zahtev serveru da klijent želi da izađe iz sobe
* “collisionEvent“ – Šalje se informacija procesru da se dogodio sudar između dva tela. Informacije koje se šalju u ovoj poruci su unikatni identifikacioni brojevi tela, kao i kojeg su tipa tela (U smislu da li je igrač, platforma, protivnik). Ime tela služi isključivo u svrhe debagovanja tokom razvijanja kako bi programer odmah video o koja dva tela se radi na osnovu imena. I tako dalje.

### Primer komunikacije klijent - server (Ažuriranje pozicije igrača)

U slučaju onaljn meča gde učestvuju dva igrača, svaki klijent u svom svetu vidi sebe kao lokalnog igrača i preostale igrače (U ovom slučaju jednog igrača) kao onlajn igrača. Svaki igrač je dužan da nakon promene pozicije na mapi pošalje svoju poziciju serveru i da server prosledi poziciju tog igrača preostalim igračima. U nastavku će biti opisan proces ažuriranja pozicije u kome učestvuju dva igrača sa jednistvenim ID-jevima (0 i 1). Oba igrača pokreću aplikaciju na svom nezavisnom uređaju tako da su to dva nezavisna procesa kod kojih se svi sistemi izvršavaju deterministički.

Igrač 0 promeni poziciju (Slika 37.). Nakon toga InputManagerTranssmiting sistem proverava da li je došlo do promene pozicije u odnosu na predhodnu iteraciju. S obzirom da je došlo do promene pozicije nova pozicija se upisuje kao poslednja pozicija i ti podaci se stavljaju u bafer za slanje podataka. Zatim dolazi red na DataTransmittingSystem koji u svojoj iteraciji čita nulti element iz bavera za slanje podataka. Pravi paket za slanje tako što pored podataka o poziciji pakuje i soket ID kao i naziv događaja (“updatePlayerInputPosition”) i nakon toga emituje paket koristeći soket iz ClientHandler-a. Server registruje novi događaj, raspakuje podatke, pakuje ih i prosleđuje podatke svim ostalim klijentima (Igraču 1) koristeći događaj pod imenom “updatePlayerInputPositionResp”. Client handler igrača 1 registruje pristigli događaj, raspakuje podatke, pakuje ih u formatiranu poruku i smešta u bafer za primljene poruke. Dolazi red na DataReceivingSystem igrača 1 koji u trenutnoj iteraciji proverava da li ima sadržaja u baferu za primljene poruke i pošto ima, čita element iz bafera i za transfomComponent za dati entitet (Pošto je iterativni sistem u pitanju jer je vezan za entitete igrača, metoda processData kao input parametar sadrži neophodan entitet) setuje novu poziciju. Tačnije za entitet čiki je ID 0 (s obzirom da je igrač za ID-jem 0 promenio poziciju)



Slika 37. Ažuriranje pozicije igrača

## Server

Serverska komponenta je razvijena korišćenjem Node runtime okruženja (Više o tome u poglavlju 3.2.1) kao rezultat istraživanja Node se čini najboljom opcijom. Druge opcije:

* Nakama – LibGDX radni okvir nema support za Nakama server tako da je ova mogućnost odbačena
* Firebase – Rešenje koje nudi API za zatvorenu serversku implementaciju, baze podataka, autentikaciju i tako dalje. Ovo rešenje nije izabrano iz više razlogu, a to su kao prvo ne postoji oficijalna podrška za radni okvir LibGDX već nekakvo bridge rešenje koje se zove gdx-fireapp. Dok je drugi razlog enkapsulacija ponuđenog rešenja i na taj način bi se sigurno dobilo mnogo efikasnije rešenje od trenutnog implementiranog rešenja ali pošto je ovo projekat isključivo edukativne svrhe, benefiti samostalnog kreiranja serverske aplikacije kao i baze podataka su dosta znanja i iskustvo

Kao web server je korišten http server koristeći Express.js radno okruženje (Sekcija 3.2.2). Za bidirekcionu komunikaciju se koriste soketi iz biblioteke socket.io. Baza podataka koja je korištena je MongoDB. Mongoose biblioteka se koristi kako bi se jednostavno definisale kolekcije podataka kao i jednostavnije korišćenje upita.

S obzirom da je izabran Node koji koristi javaScript, bila je ideja i da ostale komponente na serverskoj strani budu razvijane koristeći isti jezik. Srećom pa je baza javaScript biblioteka ogromna.

Takođe treba imati na umu da je Node predviđen za spore operacije poput I/O/networking operacija kaon a primer slanje i primanje paketa kao čitanje i pisanje sa eksternog diska i da nije namenjen za zahteve proceskroske operacije tako da serverska aplikacija uglavnom služi sa prosleđivanje poruka između klijenata.

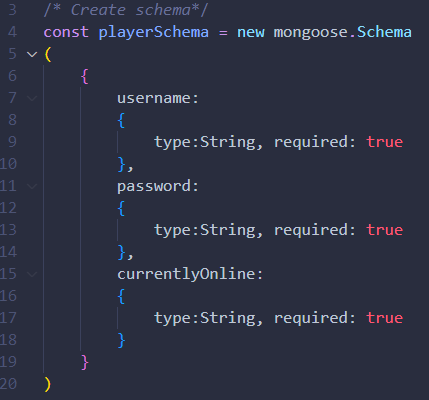
### ****Struktura servera****

Server se sastoji iz dve komponente:

* Main modul koja predstavlja ulaznu tačku za serversku aplikaciju. Ima ulogu da se pri inicijalizaciji konektuje za MongoDB serverom, inicijalizuje se HTTP server (setuje se ip adresa i port na kojem server treba da sluša). Vrši inicijalizaciju TrafficHandler modula. Odgovoran je za registraciju i prijavu klijenata kao i za slanje upita bazi podataka
* TrafficHandler modul je zadužen za komunikaciju sa klijentima kao i za praćenje stanja soba i igrača tokom meča.

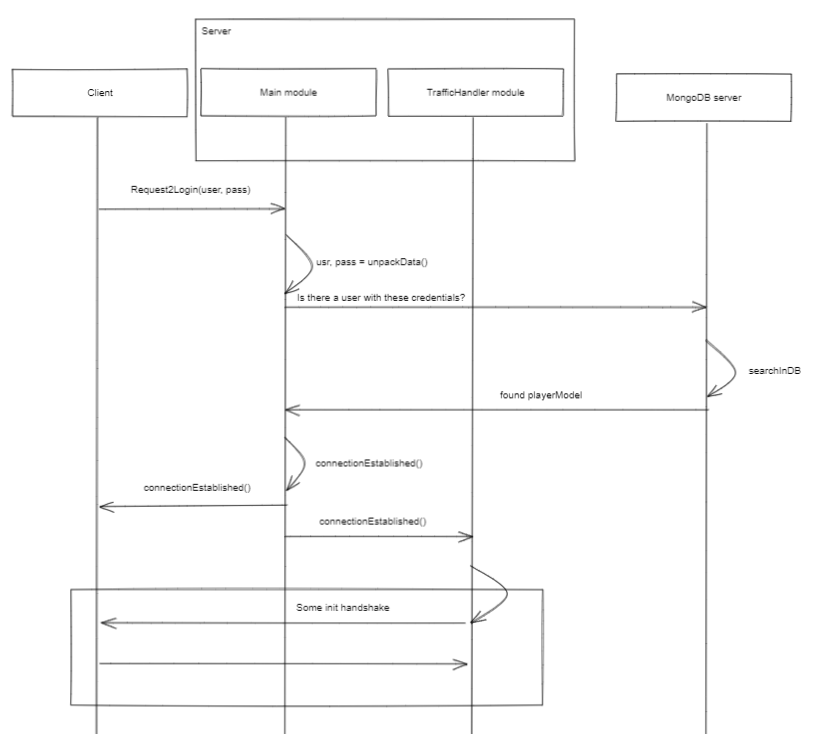
### Main modul

Kao što je već pomenuto ovaj modul između ostalog komunicira sa bazom podataka. Iz tog razloga je bitno da postoji definisan format podataka koji se upisuje, čita iz baze podataka. Moongose omogućava korišćenje šema gde se definišu polja u strukturi i da li je polje obavezno. Nakon toga se šema prevede u model i tada se pri upitima koji se šalju bazi podataka koristi željeni model koji je preveden iz odeređene šeme. U ovom projektu postoji jedna šema, a to je šema igrača (Slika 38.) koja ima polja username, password, currentlyOnline. Ozbiljno unapređenje bi bio resume mehanisam koji bi omogućavao da u situaciji kada padne server i ponovo se podigne da svi mečevi nastave gde su stali, a to bi između ostalog zahtevalo dodatne šeme koje bi verno prezentovale svet.



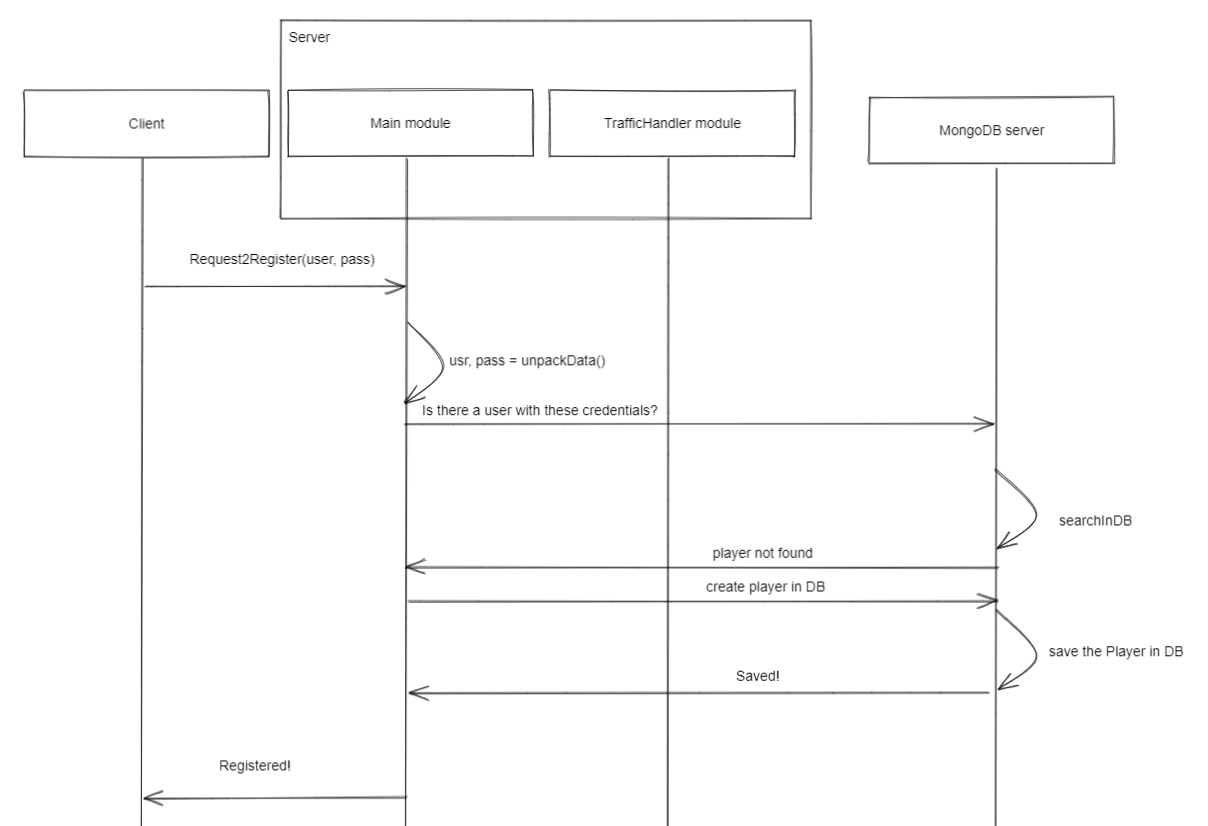
Slika 38. Primer mongoose šeme

#### Proces prijavljivanja igrača



Slika 39. Primer prijavljivanja klijenta na server

#### Proces registrovanja igrača na server



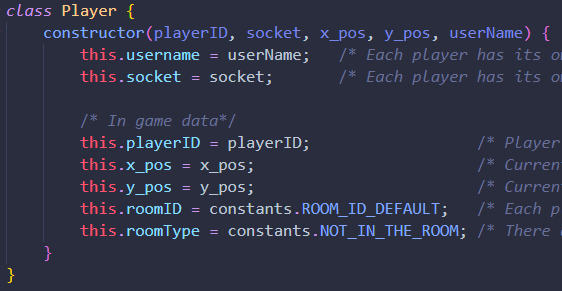
Slika 40. Primer registracije klijenta

### TrafficHandler modul

TrafficHandler modul poseduje sve neophodne podatke potrebne u komunikaciji sa klijentima poput:

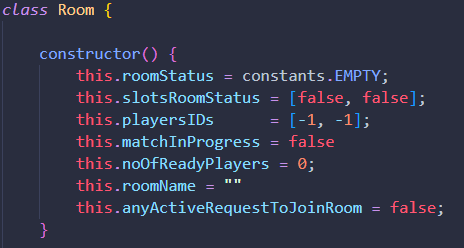
* Niz konektovanih igrača
* Registrovane kalbak funckije za sve neophodne događaje
* soketIO server
* Niz soba

Konketovani igrač je predstavljen klasom (Slika 41) koja ima polja poput korisničkog imena, konektovanog soketa, unikatnog ID-a u igri, x,y pozicija igrača, ID sobe u kojoj se nalazi, kao i tip sobe (PVP, COOP).



Slika 41. Player klasa

TrafficHandler modul prati stanje soba, tačnije popunjenost soba kao i to koji igrač u određenoj sobi tako što su podaci u sobi predstavljeni poljima poput(Slika 42.) roomStatus, stlosRoomStatus, playerIDs, matchInProgress, roomName, anyActiveRequestToJoinRoom



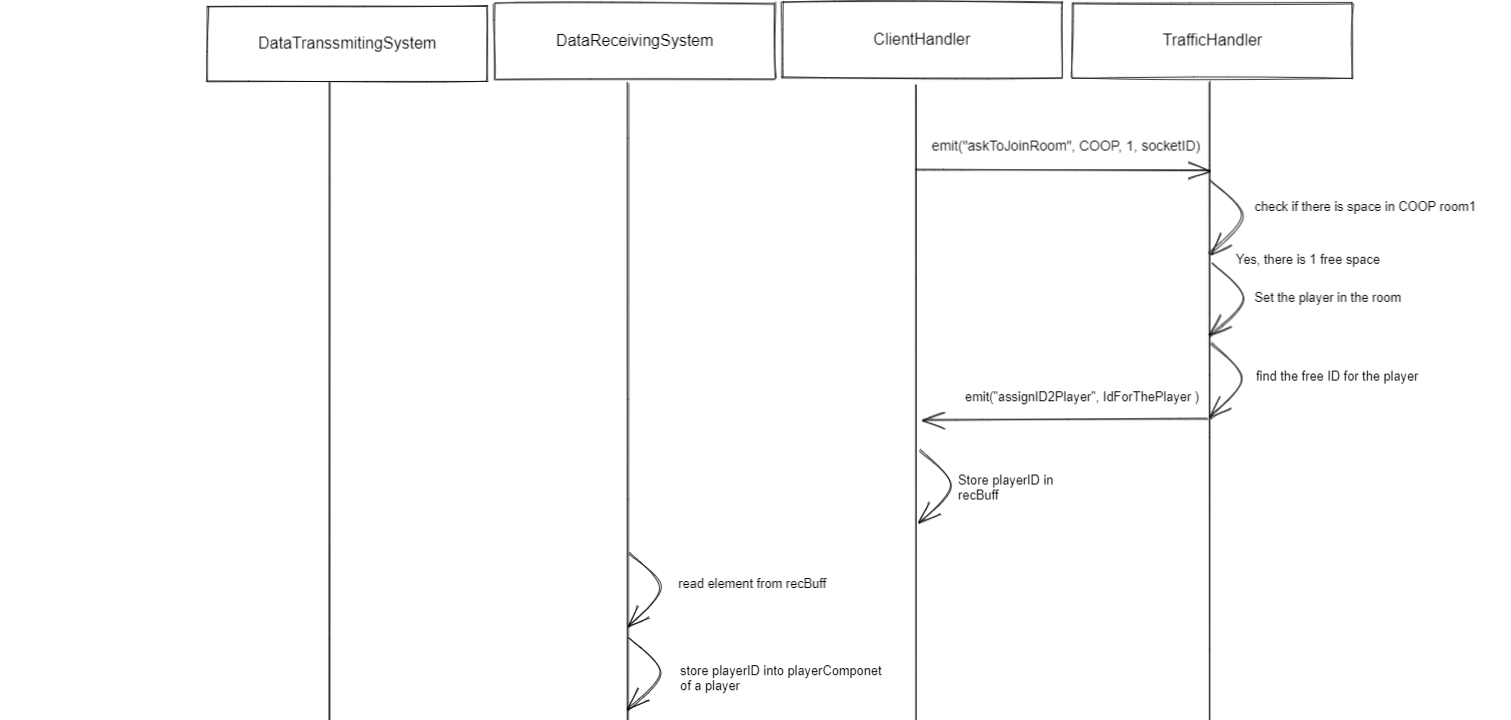
Slika 42. Room klasa

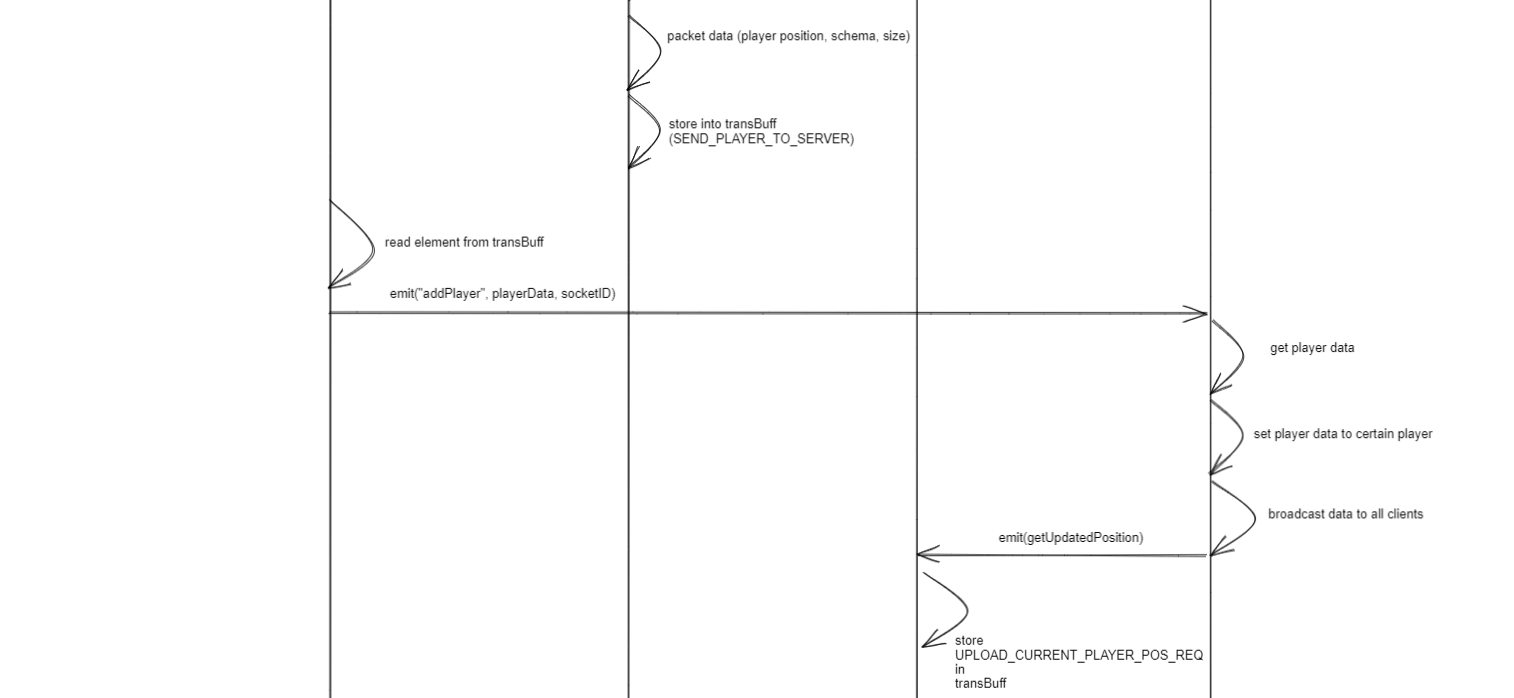
anyActiveRequestToJoinRoom flag služi kako bi se obezbedilo da se samo jedan zahtev u trenutku obrađivao. Zato što je noOfReadyPlayers deljeni resurs od koga zavisi da li je igra treba da počne ili je potrebno sačekati da ostali igrači uđu u igru.

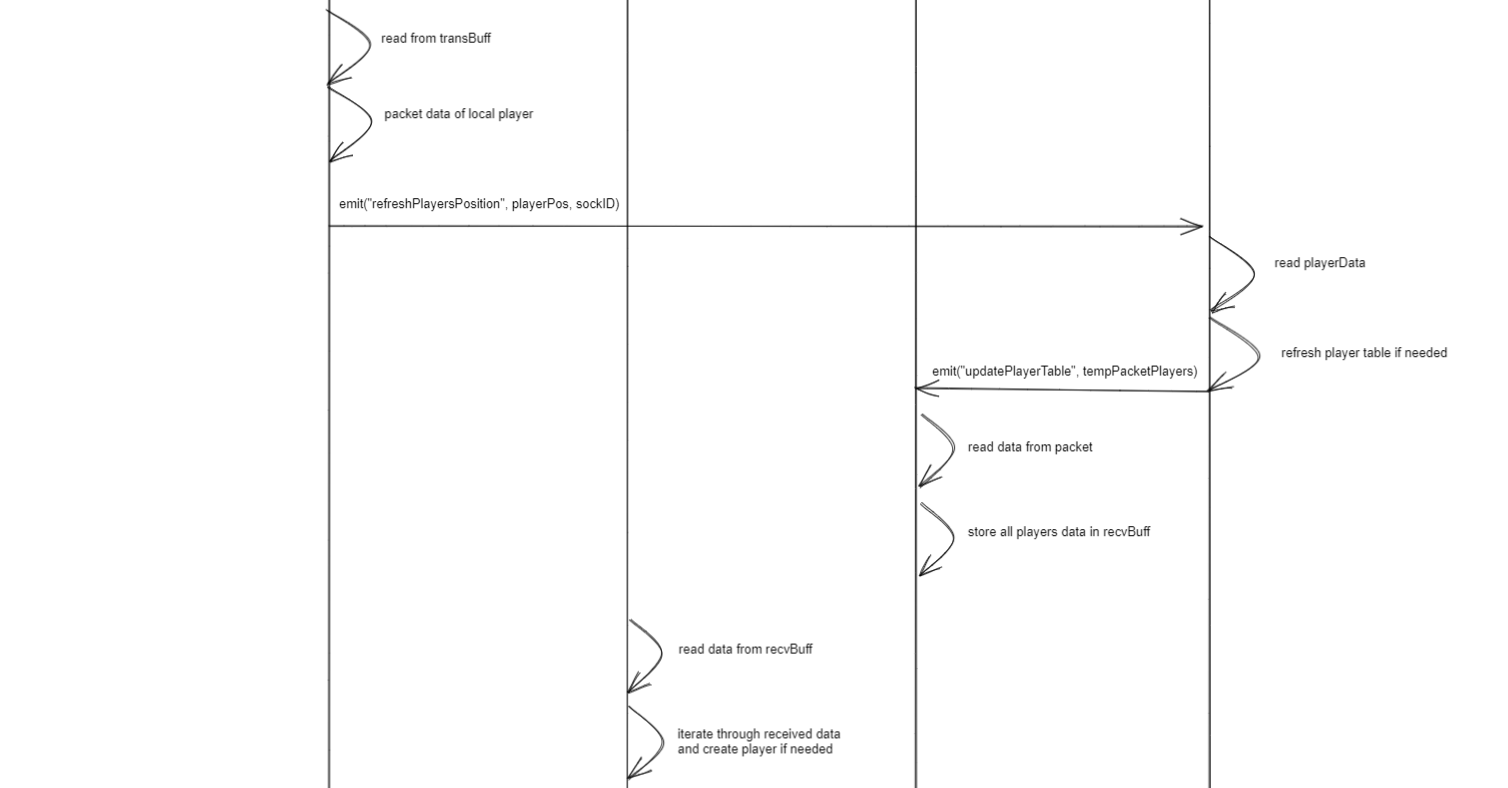
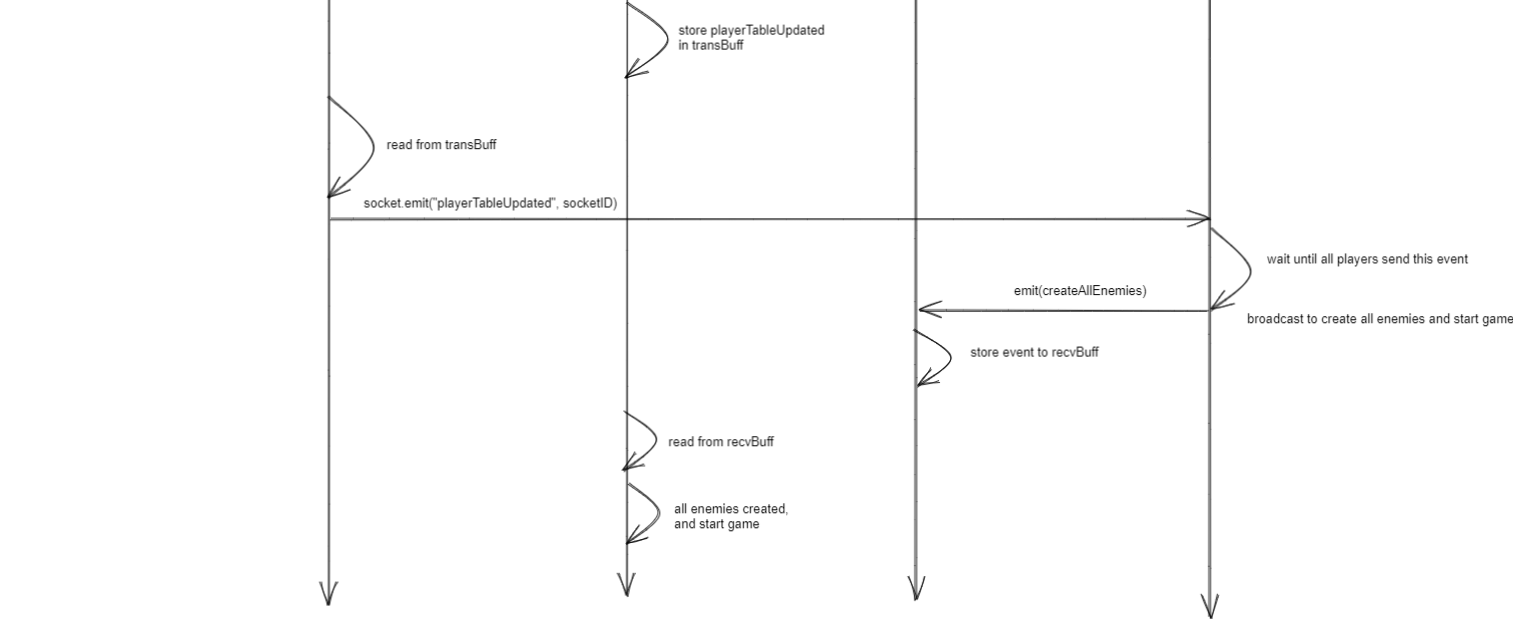
##### Proces ulaska igrača u sobu

Na narednom dijagramu biće prikazan proces ulaska igrača u sobu. Stim što je jedan igrač već u sobi.

Igrač zatraži da uđe u sobu tako što emituje događaj u kome pošalje tip sobe kao i broj sobe. Server proveri da li ta soba slobodna i s obzirom da jeste mapira igrača na sobu i pronaće slobodan ID za igrača koji želi da uđe u sobu. Emituje događaj u kome šalje id klijentu koje je zatražio da uđe u sobu. ClientHandler tog klijenta odpakuje podatke iz primljenog događaja i smešta ih u bafer za primanje poruka, zatim DataReceivingSystem tog klijenta čita naredni element iz bafera i primljeni ID dodeljuje lokalnom igraču, gde nakon toga u bafer za slanje poruka upisuje podatke o poziciji, šemi i veličini lokalnog igrača. Zatim DataTranssmitingSzstem čita primljenu poruku od DataReceivingSystema i emituje događaj addPlayer. Server kada primi poruku emituje zahtev svim klijentima (Oba klijenta koja se nalaze u sobi) da su mu potrebne najsvežije pozicije igrača. Klijenti prime poruku i koristeći već obajšnjeni princip (Koristeći DataReceivingSystem i DataTranssmittingSystem) pošalji serveru događaj “refreshPlayerPosition” gde šalju najsvežiju poziciju. Gde nakon toga server šalje svim klijentima zahtev “updatePlayerTable” gde prosleđuje listu igrača kako bi klijenti kod sebe na lokalu kreirali online igrača sa kojim igraju meč. Kada klijenti prime i izvrše zahtev da kreiraju online igrača na lokalu, šalju događaj pod nazivom “playerTableUpdated”. Kada server primi taj događaj od oba klijenta tada server šalje poruku da je potrebno napraviti sve protivnike (“createAllEnemies”) na mapi i da igra može da počne. Klijenti prime zahtev i kreiraju sve protivnike na mapi i tada meć može da počne.





Slika 43. Ulazak igrača 1 u sobu

## Tehnička unapređenja

U ovom poglavlju će biti reči o nedostacima u samom projektu. Takođe će za neke nedostatke biti ponuđena rešenja.

### ****Logika u ECS komponenti****

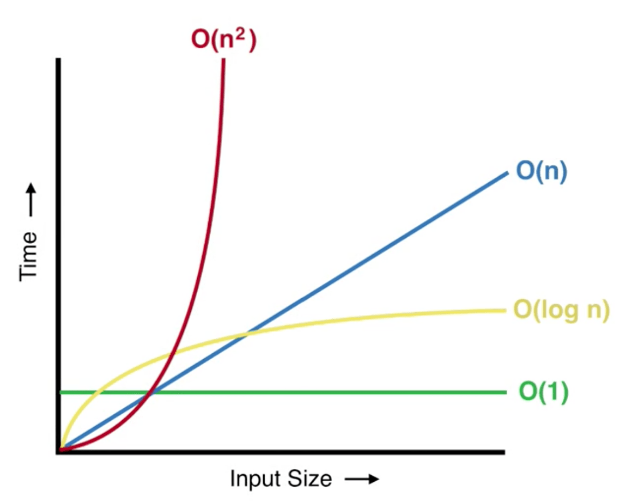
Na primer SteeringComponent u sebi ima implementiran deo logike koji je vezan za ažuriranje AI protivnika. S obzirom da se po pravilima ECS paterna, u komponentama nalaze isključivo podaci, sva logika bi trebala da se izmesti u zaseban sistem.

### ****Coupling ECS komponenti****

Unutar PlayerComponent postoje dva flega (fired i readyToTransmitBullet) koji se koriste kao most između PlayerComponent i InputManagerTransmitingSystem. Svaka komponenta treba da ima podatke isključivo za oblast za koju je vezana. Potrebno je PlayerComponent izbaciti ta polja tako da funckionalnost i dalje ostane ista, a da komponenta bude očišćena, to jeste da se napravi dobar decoupling između komponenti.

### ****getComponent usporava izvršavanje aplikacije****

U projektu je za čitanje komponente iz entitija korišćena metoda getComponent čija je kompleksnost O(logn), a u slučaju da se koristi ComponentMapper za čitanje komponenti je je O(1) (Slika 44.).



Slika 44. Vremenska kompleksnost [17]

### ****Jak klijent, slab server****

Na projektu je raspodela posla urađena tako da klijent radi skoro sav posao, a server manje više samo prosleđuje podatke klijentima. Razlog zašto je ovo urađeno na taj način je manjak iskustva pri kreiranju servera, a i literature o arhitekturi servera i nema.

Problem kod ovog pristupa je to što jedan od klijenata mora biti host. Po trenutnoj implementaciji host je klijent koji prvi uđe u sobu. Host je klijent čije stanje tokom meča je istinito stanje. Igrači jedan drugome šalju informacije o poziciji ali samo host koristi komponente za detekciju kontakta. Host određuje da li se desio kontakt i nakon toga sudar. U slučaju kada host detektuje kontakt između dva tela, šalje informaciju o ta dva tela drugom igraču. Te informacije se čuvaju u baferu fikstura kod drugog igrača i onda drugi igrač razrešava sudar između dva tela.

Na ovaj način host mora da obezbedi dovoljno dobar protok interneta.

Bolje rešenje:

Da server radi većinu posla i da klijente samo obaveštava kada se stanje promeni.

Detaljnije:

Iz klijentske strane bi trebao da se izbaci engine za fiziku. Zatim bi trebao da se promeni pristup obrađivanja komandi primljenih sa recimo tastature, na način da kada klijent želi da pomeri svog igrača recimo u levo, da se pošalje zahtev serveru koji u svom procesu pomeri igrača u levo i svim klijentima vrati novu poziciju tog igrača. Dakle klijent bi samo slao zahteve serveru, i slepo primao informacije od servera gde bi na klijentu dolazilo samo do grafičkih promena.

Na serverskoj strani bi trebale da rade dve vrste procesa. Prvi proces je onaj koji i sada radi, a to je serverska aplikacija dok bi drugi proces predstavljaju igru. Tačnije to bi bio headless (proces bez grafike pošto su serveri obično headless Linux. Pa mu se pristupa preko terminala) proces koji pri uključivanju pokreće meč. Ta match aplikacija bi u sebi imala Box2D engine za fiziku (Dakle obrnuto od klijentske aplikacije) Takođe bi ta aplikacija morala da ima i kontakt sa serverskom aplikacijom na primer preko socket.io soketa kako bi serverska aplikacija mogla biti na jednom računaru, a match aplikacija na drugom. Serversku aplikaciju je potrebno proširiti tako da komunicira sa match aplikacijom. Serverska aplikacija bi po zahtevu klijenata pokrenula match aplikaciju i tokom meča bi komunicirala sa match aplikacijom, slala joj zahteve od klijenata i onda bi promene u meču nastale u match aplikaciji slala klijentima koji bi samo replicirali promene.

Na ovaj način bi se rešio i problem 4.5.5

### Replikacija podataka

Takođe je postojala ideja da se preko mreže šalje minimalan skup podataka i da će aplikacija moći da replicira stanje meča u realnom vremenu pošto poseduje fizički engine.

U praksi se pokazalo da dolazi do desinhronizacije. To jeste sam meč se ne odvija istovetno kod oba klijenta zbog toga što se informacije sa jednog klijenta ne šalju dovoljno brzo drugom i tako se izgube informacije.

Implementirano rešenje je to da se repliciraju pozicije igrača (Jedna drugi šalju trenutu poziciju), a da se preostala desinhronizacija reši na način da host replicira kontakte (Host šalje informacije o kontkatima drugom igraču koji onda razrešava sudare i na taj način replicira stanje meča kao kod hosta)

Bolje rešenje:

Host bi trebalo da šalje već razrešene situacije poput ubijenog protivnika, pokupljene super moći I tako dalje umesto što šalje podatke o kontaktu dva tela. Ovo zahteva dodatnu izmenu trneutne arhitekture kao i pažljivu implementaciju ovog principa.

### Promena tehnologije

Razvijati igru u LibGdx random okviru je bilo zanimljivo iskustvo ali se tokom razvoja naišlo na nekoliko problema:

* Ne postoji skup iternih alata sa grafičkim interfejsom koji bi znatno mogli ubrzati razvoj određenog segmenta igre, već je potrebno tražiti 3rd party alate koje obično ili ne postoje ili nisu kompatibilni sa LibGDX
* Previše je programattical pristup (Vezano za prvu stavku), tačnije sve se programira umesto da postoje vizuelni editori
* Baza biblioteka nije velika
* LibGDX nije toliko popularan kao Unity i saim tim je broj ljudi okupljen oko njega manji, pa je manje i dokumentacije, tutorijala i svega

Rešenje:

Najpopularnije rešenje za pravljenje 2D igara je Unity engine koji ima mnogo dokumentacija, tutorijala, kao i sve potrebne alate unutar samog engine-a.

# Zaključak

Razvijanje video igre zahteva znanje iz puno oblasti.

# Literatura

1. M. Guoqiang, F. Baris, B. D. O Anderson, “Localization Algorithms and Strategies for Wireless Sensor Networks”, Information science reference New York , 2009.
2. L. Wang, Q. Xu, GPS-Free Localization Algorithm for Wireless Sensor Networks. Sensors 2010, 10, pp. 5899-5926.
3. A. Pal, “Localization Algorithms in Wireless Sensor Networks: Current Approaches and Future Challenges”, Network Protocols and Algorithms ISSC, Vol.2, No.1, 2010. pp.1943-3581.
4. B. Bulusu, J. Heidemann, D. Estrin, “Density adaptive algorithms for beacon placement in wireless sensor networks”, in: IEEE ICDCS'01, Phoenix, AZ, April 2001.
5. G. Young, A. S. Householder, “Discussion of a set of points in terms of their mutualdistances”, Psychometrika, Vol. 3, 1938, pp. 19–22.
6. K. Langendoe, N. Reijers, “Distributed localization in wireless sensor networks:a quantitative comparison”, Computer Networks 43, 2003, pp. 499–518.
7. T. He, C. Huang, B. M. Blum, J. A. Stankovic, T. Abdelzaher “Range-Free Localization Schemes for Large Scale Sensor Networks” , 2003. pp. 81-95.
8. K. Chen, Z. H. Wang, M. Lin, M. Yu, “An Improved DV-Hop Localization Algorithm for Wireless Sensor Networks” IET International Conference on Wireless Sensor Network, 2010, pp. 255-259.
9. Y. Shang, W. Ruml, Y. Zhang, and M. P. J. Fromherz, "Localization from Mere Connectivity", Proceedings of ACM MobiHoc, 2003, pp. 306-310.
10. I. Stojmenović, “Handbook of Sensor Networks Algorithms and Architectures”, Wiley Series on Parallel and Distributed Computing, 2005, pp. 277-311.
11. M. Terwilliger, “Localization in Wireless Sensor Networks”, Western Michigan University, PhD Thesis, 2006.
12. D. Niculescu and B. Nath,“Ad-hoc Positioning System”, Global Telecommunications Conference (GlobeCom), IEEE, Volume 5, Nov 25-29, 2001, pp. 2926-2931.
13. N. B. Priyantha, H. Balakrishnan, E. Demaine and S. Teller, “Anchor-free Distributed Localization in Sensor Networks”, Technical Report 892, MIT Laboratory for Computer Science, April 2003.
14. N. Bulusu, J. Heidemann and D. Estrin, “GPS-less Low Cost Outdoor Localization for Very Small Devices”, IEEE Personal Communications Magazine, Volume 7, Issue 5, Oct.2000, pp. 28-34.
15. L. Girod, V. Bychovskiy, J. Elson and D. Estrin, 2002. Locating tiny sensors in time and space: A case study. Proceedings of the IEEE Processors, Sept. 2002, Freiburg, Germany, pp. 214-219.
16. A. Harter, A. Hopper, P. Steggles, A. Ward and P. Webster, 2002. “The anatomy of a context- aware application. Wireless Networks, 8”, pp. 187-197.
17. D. Niculescu, B. Nath, 2003. “Ad hoc positioning system (APS) using AOA”. Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies, March 30-April 3, Rutgers University, Piscataway, pp. 1734- 1743.
18. P. Bahl, and V.N. Padmanabhan, 2000. RADAR: An in-building RF-based user location and tracking system. Proceedings of 19th Annual Joint Conference of the IEEE Computer and Communications Societies, March 26-30, Tel Aviv, Israel, pp. 775-784.
19. L. Doherty, K. S. Pister, and L. E. Ghaoui, “Convex Position Estimation in Wireless Sensor Networks,” IEEE ICC’01, vol.3, Anchorage, AK, April 2001, pp.1655-1663.
20. J. Albowicz, A. Chen, and L. Zhang, “Recursive Position Estimation in Sensor Networks,” 9th Int’l. Conf. Network Protocols, Nov. 2001, pp. 35–41.
21. J. Savarese, Rabaey, and K. Langendoen, “Robust Positioning Algorithm for Distributed Ad- hoc Wireless Sensor Networks”, USENIX Technical Annual Conference, Monterey, June 2002. CA, pp. 317-327.
22. K. Chen, Z. Wang, M. Lin, M. Yu:”An Improved DV-Hop Localization Algorithm for Wireless Sensor Networks”, IET International Conference on Wireless Sensor Network, 2010, pp. 255-259.
23. X. Yi, , Y. Liu, L. Deng, Y. He, “An Improved DV-Hop Positioning Algorithm with Modified Distance Error for Wireless Sensor Network”, The 2th International Symposium on Knowledge Acquisition and Modeling, Vol. 2, 2009, pp. 216-218.
24. Q. Qian, X. Shen, and H. Chen, “An Improved Node Localization Algorithm Based on DV- Hop for Wireless Sensor Networks”, Comput. Sci. Inf. Syst., 2011, pp. 953-972.
25. H. Chen, K. Sezaki, P. Deng, H. Cheung So, “An Improved DV-Hop Localization Algorithm for Wireless Sensor Networks”, Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on, pp. 1557 - 1561.
26. B. Parkinson, J. Spilker, “Global Positioning System:Theory and apllication(American Institute of Aeronautics and Astronautics)”, 1996. pp. 409-433.
27. G. Milovanović, P. Stanimirović, “Simbolička Implementacija Nelinearne Optimizacije”, Niš, avgust 2002, pp. 213-214.
28. N. Karmarkar, "A new polynomial-time algorithm for linear programming", Combinatorica, vol. 4, 1984, pp. 373-395.
29. Y. Nesterov, A. Nemirovskii, Interior-Point Polynomial Algorithms in Convex Programming, SIAM, 1994, pp. 217-248.
30. J. C.Ruiz, J. G. Rosiles, E. Sifuentes and P. Rivas- Perea, “A Low-Complexity Geometric Bilateration Method for Localization in Wireless Sensor Networks and Its Comparison with Least-Squares Methods”, Sensors 2012, 12, pp. 839-862.
31. A. Boukerche, H.A.B.F. Oliviera, E.F. Nakamura, A.A.F.Loureiro, Localization systems For Wireless Sensor Networks, IEEE Wireless Communications, December 2007, pp. 6-12.