Tallinna Ülikool

Informaatika Instituut

Libgdx raamistik ja 2D arvutigraafika õppematerjal

Seminaritöö

Autor: Raner Piibur

Juhendaja: Jaagup Kippar

Autor: ..... ” ..... ”2015

Juhendaja: ..... ” ..... ”2015

Instituudi direktor: ..... ” ..... ”2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev seminaritöö on minu töö tulemus ja seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

.... (kuupäev) (autor)

Sisukord

Sissejuhatus

Mobiilsed seadmed on muutunud aastatega järjest populaarsemaks, millega on suurenenud mängijate hulk, kes mängiksid mänge just nendel seadmete. Lisaks on suurenenud platvorm mängude populaarsus, mis ei nõua suurt jõudlust ning mida on võimalik arendada nii mobiilsetele seadmetele kui ka personaalarvutile. Turg on kasvanud suuremaks kui kunagi varem ja on avatud ka arendajatele, kellel ei ole suur eelarve. Vaja on raamistikku, mis oleks võimalikult odav või isegi tasuta ning mida ei oleks tülikas kasutada väikesel meeskonnal. Turgu valdab ka suur killustatus ja kasumlikum oleks luua rakendus raamistikuga, millega on võimalik mäng luua kõikidele platvormidele korraga, kasutades sama koodibaasi, ilma et peaks muretsema koodi portimise pärast teisele platvormile. Autor valis neid spetsifikatsioone arvestades raamistiku Libgdx, millel autorile teadaolevalt eestikeelne õppematerjal puudub.

Teema valiku põhjuseks on autori isiklik huvi Libgdx ja 2D graafika vastu, eestikeelse materjali puudumine ning korraliku plaatmootor[1] põhimõttel töötava Libgdx rakenduse õppematerjali puudumine. Valikut mõjutas ka Tallinna Ülikooli Informaatika Instituudi õppekavas olevad kohustuslikud ained, mille läbinud tudengid saaksid selles materjalis kasutada, nii teadmiste täiendamiseks kui ka kinnitamiseks.

Seminaritöö käigus loome õppematerjal, mille abil oleks võimalik iseseisvalt õppida Libgdx raamistikku. Materjal peaks andma ülevaate Libgdx raamistikust ja kuidas selle raamistikuga mitmeplatvormiline 2D arvutigraafika rakendus luua. Seminaritöö peaks tekitama huvi Libgdx raamistiku ning mängude loomise vastu.

Esimeses peatükis annab autor Libgdx arendusjuhendi alusel ülevaate raamistikust, selle võimalustest, toetatud platvormidest, moodulitest ja elutsüklist.

Teises peatükis annab autor ülevaate juba olemasolevatest õppematerjalidest, kus toome välja nende positiivsed ja negatiivsed küljed.

Kolmandas peatükis analüüsime loodavat õppematerjali. Sõnastame õppematerjali eesmärgi, selgitame välja nõuded, meediumi, õpiväljundid, ülesehituse, õpiteed ja toome välja testimise käigus saadud tulemused.

[1] Plaatmootor (*Tile engine*) on tehnika, kus graafika renderdamiseks kasutatakse väikeseid tekstuure, mida saab korduvalt kasutada, et joonistada suurem tekstuur. Tehnika säästab sellega mälu ja tõstab renderdamiskiirust. Plaatmootoris olevat plaatkaarti võib vaadata kui 2-dimensioonilist massiivi, kus iga element sisaldab infot maailma kohta.

1 Libgdx ülevaade

Libgdx on arvutigraafika arenduse raamistik, mida kasutatakse põhiliselt mängude loomiseks. Raamistik on iseenesest väga paindlik ja seda võib kasutada ükskõik millise projekti loomiseks, kus eesmärgiks on visualiseerimine. Libgdx'i abil saab kasutada sama lähtekoodi mitme platvormi jaoks. Tegemist on Java raamistikuga ja seega on võimalik kasutada kõiki produktiivsust tõstvaid tööriistu, mis selles ökosüsteemis leiduvad. Raamistiku eesmärk ei ole arendajale peale suruda spetsiifilist disaini, vaid eesmärgiks on varustada arendajat mooduliteg, mis tööd lihtsustavad.

Libgdx võimaldab programmeerida madalatasemeliselt, suheldes otse operatsioonisüsteemi failisüsteemiga, sisendseadmetega ja graafikaprotsessoriga OpenGL liidese abil (ühendatud OpenGL ES 2.0 ja ES 3.0 liides). Nende vahendite peale on ehitatud suutlik API (*Application Programming Interface*), millega on võimalik renderdada tekstuure ja teksti, luua kasutajaliideseid ja menüüsid, mängida ja voogesitada heliefekte ning muusikat, tegeleda lineaaralgebra ja trigonomeetriaga jne. Raamistik toetab ka paljusid kolmanda osapoole teeke. Paindlikus on selle raamistiku sünonüüm (Libgdx Kommuun, 2014).

Toetatud platformid on järgmised:

- Windows
- Linux
- Mac OS X
- Android (2.2+)
- iOS
- HTML5 (Javascript/WebGL)

Libgdx-i saab integreerida mitmeid kolmanda osapoole teenuseid:

- **Admob** – suurimaid mobiilikuulutuste võrgustikke. Admobi abil saab rakendust reklaamida ja analüüsida.
- **Swarm** – võimaldab rakendusele lisada võrgus olevaid tabloosid, erinevaid sotsiaalseid elemente (sõnumite saatmine mängijate vahel jms.), pilveteenust rakenduse andmete salvestamiseks. Swarmi abil on on võimalik rakendusele lisada ka nüüdseks ühte populaarsemat ärimudelit *freemium*[2].
- **Nextpeer** – võimaldab rakendusele lisada sünkroonne või asünkroonne võrgutugi.

[2] Freemiumi ärimudelis on tarkvara kasutamine tasuta, kuid erinevate lisateenuste eest tuleb kasutajal maksta. Näiteks tegelase kosmeetika, karakteri edasiarendamine, tasemetest edasipääsemine.

- **Google Play Game Services** – võimaldab lisada erinevaid sotsiaalseid funktsionaalsusi, pilveteenust rakenduse andmete salvestamiseks, võrgutuge, kaitset piraatluse eest.

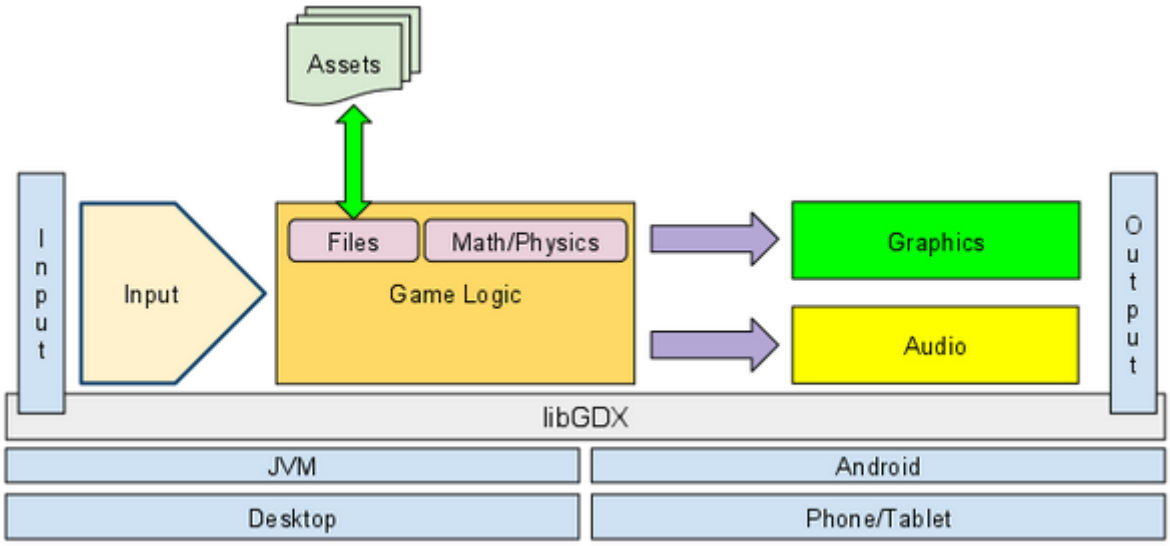
Libgdx kasutab mitmeid 3. osapoole teeke, millest tähtsamad on järgmised:

- **LWJGL** (*Lightweight Java Game Library*) – teek, mis võimaldab arendajal kasutada OpenGL-i (graafika), OpenCL-i (paralleelne programmeerimine) ja OpenAL-i (heli), et arendajal oleks võimalik luua kõrge efektiivsusega arvutigraafika rakendus Java programmeerimiskeeles.
- **FreeType** – teek, mille abil fonte renderdada.
- **Mpg123** – teek, mille abil on võimalik esitada MPEG formaadis helifaile.
- **Xiph** – teek võimaldab esitada OGG formaadis helifaile.
- **SoundTouch** – heli manipuleerimiseks mõeldud teek, mille abli saab muuta heli tempot, kõrgust ja taasesituse kiirust.
- **Box2D** – füüsikamootor, mille abil saab simuleerida jäike kehi kahedimensioonilises maailmas.
- **Kiss FFT** – teek võimaldab sooritada matemaatilise analüüsi ülesandeid kõrge efektiivsusega, näiteks reaalajas helispektrumi analüüsimine.

1.1 Libgdx moodulid

Libgdx koosneb kuuest moodulist, mis operatsioonisüsteemiga suhtlevad (LibGDX kommuun, 2014). Nendeks on:

- **Rakendus** (*Application*)**:** jooksutab rakendust ja annab teada rakendustasemel olevatest sündmustest nagu akna suuruse muutmine, rakenduse peatamine jne. Annab juurdepääsu logimisvahenditele, näiteks mälu kasutuse kohta.
- **Failid** *(Files)*: annab juurdepääsu platvormi failisüsteemile.
- **Sisend** *(Input)*: informeerib API klienti kasutaja suhtlusest hiire, klaviatuuri, puudutuse või kiirendusmõõturi abil. Toetatud on nii sündmus-, kui ka pollimise põhimõttel töötavad protsessid.
- **Heli** *(Audio)*: võimaldab mängida heliefekte, voogesitada muusikat ja isegi otsest ligipääsu heliseadmetele.
- **Graafika** *(Graphics)*: paljastab OpenGL ES 2.0 või ES 3.0 API ja võimaldab pärida või sätestada video režiime jms.
- **Võrk** *(Net)*: moodul, mille abil rakendada HTTP või TCP suhtlust serveri ja kliendi vahel.



Joonis 1. Libgdx moodulite arhitektuur (Libgdx dokumentantsioon, 2014)

1.2 Rakenduse elutsükkel

Libgdx-i rakenduse elutsüklile saab ligi implementeerides „ApplicationListener“ kasutajaliidese. Võimalik on laiendada ka oma põhiklassi abstraktse klassi „Game“-ga, mis haldab erinevaid ekraane (menüü, seaded, tabloo jne). Kuna Android on sündumspõhine, siis rakendab seda ka Libgdx ja tavalist mängutsüklit raamistikust ei leia.

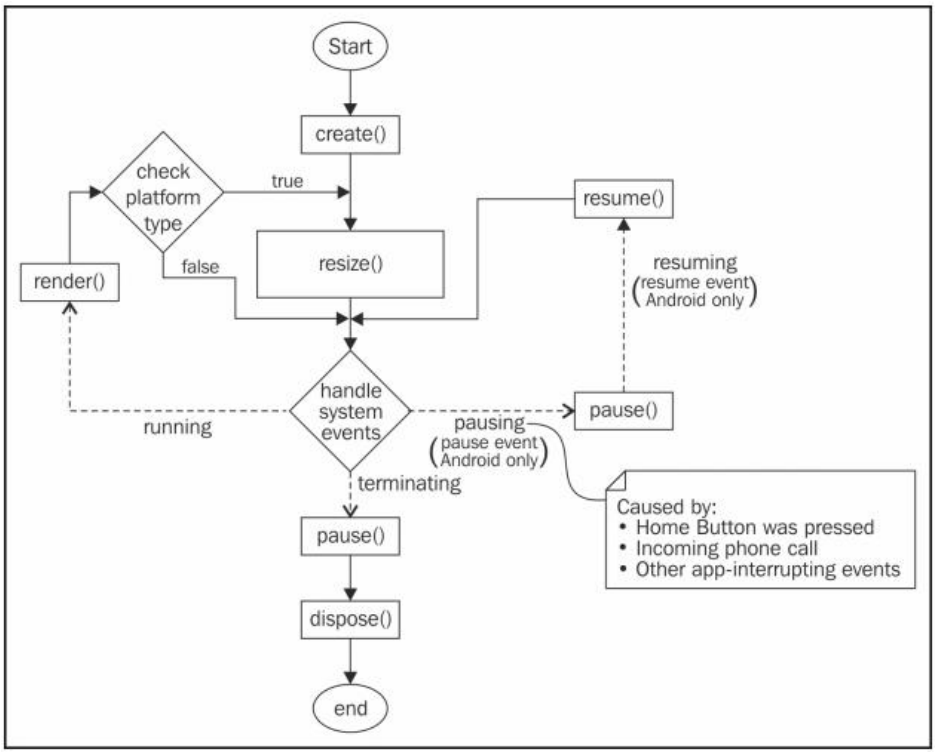
Elutsükkel koosneb järgnevatest meetoditest/sündmustest (Joonis 7):

- **create()**: Meetod kutsutakse välja siis, kui rakendus luuakse.
- **resize()**: Meetod kutsutakse igakord, kui ekraani suurust muudetakse ja rakendus ei ole puhkestaatuses. Meetod kutsutakse korra välja ka päris **create()** meetodit. Parameetriteks on rakenduse laius ja kõrgus.
- **render()**: Meetod, mis kutsutakse välja igakord, kui on vaja pilt uuesti joonistada. Toimub mängutsükli sees. Samas meetodis toimuvad ka tavaliselt mänguloogika uuendused.
- **pause()**: Androidi platvormil kutsutakse meetod juhul, kui „Home“ nuppu on vajutatud või sisse on tulemas kõne. Windowsis/Linuxis kutsutakse meetod välja enne **dispose()** meetodit. Tavaliselt toimub siin mängustaatuse salvestamine.
- **resume()**: Meetod kutsutakse välja siis, kui rakendus jätkab pärast puhestaatuses väljatulemist.
- **dispose():** Meetod kutsutakse välja siis, kui rakendus hävitatakse.

```
package com.me.justanothertowerdefense;

import ...

public class JustAnotherTowerDefense extends ApplicationAdapter {

    @Override
    public void create () {
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void render () {
    }

    @Override
    public void pause() {
        super.pause();
    }

    @Override
    public void resume() {
        super.resume();
    }

    @Override
    public void dispose() {
        super.dispose();
    }
}
```

Joonis 2 Rakenduse elutsükkel

Rakenduse olekuid iseloomustab kõige paremini diagramm, mis on näidatud joonisel 3.



Joonis 3 Rakenduse olekud (Learning Libgdx Game Development, 2013)

2 Olemasolevate õppematerjalide analüüs

Selles peatükis anname ülevaate õppematerjalidest, mida on võimalik kasutada Libgdx'i õppimiseks. Toome välja nende positiivsed ja negatiivsed küljed. Libgdx kohta on loodud autorile teadaolevalt vaid üks põhjalik raamat „Learning Libgdx Game Development" autorilt Andreas Oehlke. Ülejäänud materjal on loodud Libgdx kommuuni poolt ajaveebi stiilis.

2.1 Libgdx arendusjuhend (Libgdx kommuun, 2014)

Arendusjuhend asub Libgdx Github-i hoidlas[3]. Materjal on loodud raamistiku enda kommuuni poolt ja täieneb pidevalt. Materjal on väga uudne ja 2013. aasta teises pooles, kui autor alustas Libgdx õpinguid oli materjal üpriski kasine, hetkel aga leiab Libgdx raamistikku kasutav arendaja sealt väga palju vajalikku informatsiooni erinevate funktsionaalsuste ja moodulite kohta.

Arendusjuhendi eelisteks on täpsus, kuna on loodud arendajate endi poolt. Lisaks leiab sealt alati informatsiooni uute moodulite ja muudatuste kohta.

Autor leiab, et arendusjuhend töötab pigem arendaja käsiraamatuna ja ei käsitle erinevate klasside sulandumist ühtseks rakenduseks, mis on tihti põhiraskuseks algajatele mänguloojatele, kellele valmistab raskusi tervikliku pildi kujunemine. Materjal on ka suhteliselt kaootiline, seega kui õppijal ei ole täpselt selge, mida ta tahab teha, võib orienteerumisel tekkida raskusi. Puudub ka terve lähtekood näidete kohta, mis raskendab omakorda õppimist.

Autori soovitab arendusjuhendit kasutada sünkroonselt teiste õppematerjalide, kuna lisab õpingutele lisamõõtme ja teeb raamistiku tundmaõppimise kergemaks.

2.2 A Libgdx tutorial (Gustavo Steigert, 2012)

See õpetus asub Gustavo Steigerti ajaveebis[4]. Õpetus on aastast 2012, seega võib arvata, et materjal ei ole ajakohane. Õnneks on autor kommentaarides välja toonud muudatused, mis on vaja sisse viia, et kood ka uutes versioonides töötaks.

Eelisteks teiste materjalide ees on õpetus, kuidas luua Scene2D[5] abil erinevaid mänguekraane, mis on hästi nii pildimaterjaliga, kui ka sõnadega seletatud. Lisaks asjalik õpetus, kuidas kasutada TWL (*Themable Widget Library*) teeki, mida võib kasutada alternatiivina Scene2D asemel, et luua

[3] Libgdx arendusjuhend asub aaddressil https://www.github.com/libgdx/libgdx/wiki
[4] A Libgdx tutorial asub aaddressil http://www.steigert.blogspot.pt
[5] Moodul, millega saab luua kasutajaliideseid või.

kasutajaliideseid. Iga näite kohta on ka terviklik lähtekood, mis annab õpetusele suure lisaväärtuse ja mille puudumine tekitaks nii mõnelgi frustratsiooni.

Miinuseks tooks välja lähtekoodi asukoht, milleks on Google Code. Autor on kasutanud Google Code-i, Bitbucket-it ja Github-i ning nendest kolmest on Google Code algaja vastu kõige ebasõbralikum. Google Code-s ei ole selge, kuidas lähtekoodi alla laadida. Kuna tegemist on siiski Libgdx raamistiku õppematerjaliga ei ole mõistlik anda õppijale lisakohustus õppida asjaga mitteseotud süsteemi, vaid parem on see laadida kohta, kust on seda võimalik lihtsalt kätte saada.

Autor leiab, et sellest materjalist on hea õppida, kuidas luua erinevaid mänguekraane ja õpetuse abil on võimalik otsustada kumba moodulit kasutada, kas Scene2D või TWL.

2.3 LibGDX Tutorial Series (GameFromScratch, 2014)

Õppematerjal asub erinevaid mänguarendusteemasid käsitleval veebilehel GameFromScratch[6]. Ajakohane materjal, mis käsitleb mängude loomist Scene2D mooduli abil.

Õpetus on väga algajasõbralik ja seletab üksikasjalikult lahti, mida mingi osa koodist teeb, lisaks kasutatakse ohtralt visuaalseid näiteid, kuidas rakendus peaks töötama. Eelisteks teiste õppematerjalide ees on see, et leheküljel asuvad ka teised mänguarendamiseks vajalikud õpetused erinevate rakenduste ja tehnikate kohta. Lisaks käsitleb materjal ka võrku, mis on Libgdx raamistikus väga uudne.

Materjal on veel arendamisjärgus ja ei ole veel täielik, mis on ka tema puuduseks. Autor leiab, et lähtekood võiks olla ka allalaaditav, sest kogemus on näidanud, et mõnikord võib jääda arusaamatuks, mida kuhugi kirjutama peab ja võib juhtuda, et näidist ei saadagi korralikult tööle.

Autori arvates hea koht, kust saada baasteadmised, leida juhiseid, kuidas luua võrgutoega rakendus ja kuidas luua rakendus kasutades selleks Scene2D-d.

2.4 Game Development in Android using Libgdx (Abhishek Batra, Rahul Srivastava, 2013)

Õppematerjal asub Rotating Canvas Games veebilehel[7]. Tegemist on materjaliga, mille autorid on Libgdx raamistikuga ehitatud mängu Android market keskkonnas välja andnud.

[6] GameFromScratch veebiaadress on http://www.gamefromscratch.com/
[7] Game Development in Android using Libgdx asub aadressil http://rotatingcanvas.com/tutorials-index/

Materjal on veebist leiduvatest õpetustest kõige põhjalikum. Sisaldab ka Box2D füüsikamootori kasutamist, mida edasijõudnud tahavad kindlasti kasutada, kui on mõttes luua huvitav, kaasaegne mäng. Õppimise käigus valmib töötav rakendus ning võimalik on õppida isegi pikslivarjundaja abil (*Fragment Shader*) vee simuleerimist, mida OpenGL-i väheke tundvad inimesed kindlasti sooviks osata.

Kahjuks puudub enamikel õpetustel lähtekood mõnes hoidlas, seega võib õpilasel tekkida vea korral porbleeme rakenduse töölesaamisega. Õpetustes ei erista ka kirjastiililt koodi muust tekstist ja on selle tõttu halva loetavusega.

Autor leiab, et materjal ei sobib algajatele, vaid edasijõudnutele.

2.5 Getting Started in Android Game Development with Libgdx (Tamas Jano, 2013)

Õppematerjal asub Tamas Jano-se ajaveebis[8]. Ajaveeb sisaldab lisaks Libgdx raamistikule ka muud arvutigraafikat puudutavat materjali. Materjali käigus valmib töötav rakendus ja selle käigus näidatakse erinevaid tehnikaid, mida mängude loomisel kasutatakse.

Õpetuse tekst on hästi struktureeritud, kood on muust tekstist hästi eraldatud ja lahti seletatud. Materjal annab hästi edasi seda osa, kuidas mängu projekteerimisel tuleks mõelda ja samas näidates, kuidas Libgdx-i kasutada. Iga peatüki kohta on võimalik õpilasel alla laadida ka lähtekood.

Materjal ei ole täielik ja autori arvates jääb õpetusest saadud teadmistest väheks. Kuna materjal on kaks aastat vana, siis ei kasutata seal nüüdseks valminud uusi mooduleid. Hetkel on selles stiilis mängu projekteerimiseks paremaid lahendusi. Materjal sobib siiski hästi algajale.

2.6 Learning Libgdx Game Development (Andreas Oehlke, 2013)

Autorile teadaolevalt ainuke raamat, mis on ilmunud Libgdx raamistiku kohta. Raamat on väga põhjalik ja seega ka väga mahukas. Õppematerjali läbides valmib Super Mario stiilis mäng ja selle käigus tutvustatakse raamistiku erinevaid võimalusi ja kolmanda osapoole teeke ning erinevaid 2D graafikas kasutatavaid tehnikaid.

Materjal on hästi struktureeritud ning kood Libgdx enda arendusjuhendi stiilis lahti seletatud. Erinevalt muudest materjalidest leiab õpikus ka väga häid objektide diagramme. Õpetuse läbinud õpilane peaks olema suuteline iseseisvalt Libgdx abil *side-scroller*[9] stiilis rakendusi looma.

[8] Tamas Janose mänguteemaline ajaveeb asub aadressil http://www.obviam.net/

3 Õppematerjali analüüs

Seminaritöö käigus loodava õppematerjali vajadus tuleneb eestikeelse õppematerjali puudusest. Lisaks on puudus õppematerjalist, mis käsitleks Libgdx funktsionaalsust, kuidas luua rakendus kasutades plaatmootor tehnikat. Libgdx õppematerjali loomise suurteks faktoriteks on ka autori huvi graafiliste rakenduste loomise vastu, Libgdx raamistiku suhteliselt hästi tundmine ja suur tahe luua Libgdx raamistikku käsitlev õppematerjal.

Materjali loomisel annab autor edasi ka enda kogemusi Libgdx raamistikuga, mis on omandatud mitmete Libgdx õppematerjalide läbimisega, Libgdx lähtekoodi ning mängude loomisel kasutatavate tehnikate uurimisega.

Autor lähtub materjali koostamisel ADDIE mudelist. Vajadusel liigutakse tagasi eelmistesse faasidesse, et nendesse korrektuure sisse viia.

3.1 Eesmärk

Eesmärgiks on luua Libgdx raamistiku õppematerjal, kus õppija oleks kaassõitja ja näeks, kuidas töötav 2D arvutigraafika rakendus valmib, lugedes ja kirjutades palju koodi. Õppematerjali abil on võimalik iseseisvalt Libgdx raamistikku tundma õppida.

Töö käigus Luuakse *tower-defense*[10] stiilis mängu näidis. Valiku põhjuseks oli žanri küllaltki lihtne üleseehitus ja žanri enda populaarsus. Mängu loomiseks kasutame plaatmootor tehnikat, mis on implementeeritud Libgdx-is suhteliselt hiljuti ja materjali selle kohta leidub vähe. Eesmärk ei ole luua Libgdx-i kõiki funktsionaalsusi kajastav materjal, vaid materjal, mis õpetaks, kuidas Libgdx raamistikus orienteeruda ja erinevat funktsionaalsust omavahel ühendada.

3.2 Nõuded

Selle õppematerjali kasutamiseks peab õppuril olema programmeerimiskogemus Java programmeerimiskeeles. Kindlasti peab õpilane omama mingit kontseptsiooni sellest, mis on klass, polümorfism, tehase meetod, mitmelõimeline protsess, graaf, pinu, vektor, andmestruktuur. Kasuks tuleb eelnev kogemus mängude arendamises. Õppur peaks olema läbinud ained „MLM6222 Kõrgem Matemaatika“, „IFI6083 Algoritmid ja andmestruktuurid“, „IFI6069 Programmeerimise põhikursus“ ja „IFI6059 Rakenduste programmeerimine“ või omama samaväärseid teadmisi, mis ainete läbimisel

[9] *Side-scroller* on mäng, kus vaateava maailmale asetseb külje peal ja tegelased liiguvad vasakult paremale. Populaarseim *side-scroller* on kindlasti Nintendo EAD poolt loodud Super Mario.

[10] Strateegiamängu alamliik, kus mängija eesmärgiks on peatada vaenlased jõudmaks teise kaardi osasse, ehitades erinevaid kaitsetõkkeid ja ehitisi.

omandatakse. Õppematerjali on võimalik läbida ka ilma eelnevalt mainitud teadmisi omamata, kuid nende eelteadmistega on õppematerjali omandamine tunduvalt lihtsam. Nõudeid arvesse võttes on õppematerjal sobilik eelkõige Tallinna Ülikooli Informaatika Instituudi bakalauruseõppe kolmanda kursuse tudengile, kellel peaks olema vajalikud eelteadmised, et õppematerjal edukalt läbida. Kõige tähtsaim omadus on kindlasti lähtekoodi lugemis oskus, ilma milleta ei ole võimalik materjali kasutada.

Autor seletab lahti kõik terminid, mis õppematerjali käigus kasutusele tuleva ja mida autor arvab, et õppur seni kuulnud ei ole, mistõttu ei pea õppur terminitega kursis olema.

Õppematerjali loomisel kasutatakse IntelliJ IDEA arenduskeskkonda, kuid kasutada võib ka Eclipse-i või Netbeans-i. Tavaliste redaktorite nagu Notepad++ kasutamine on tungivalt mittesoovitatav. Õppuril peab seega olema operatsioonisüsteem, millega oleks võimalik jooksutada ühte eelnevalt mainitud arenduskeskkonda ja omama seal ka arenduskogemust. Lisaks arenduskeskkonnale on vajalik Android SDK olemasolu.

Autor leiab, et Libgdx'iga töötamiseks on vaja, et arvuti vastaks järgmistele nõuetele:

- Vähemalt 4GB RAM'i. Soovitatav on 8GB või rohkem.
- Haswell, Broadwell või Ivy Bridge arhitektuuriga CPU. Miinimum i5, soovitatavalt i7.
- Arendamiseks sobib ka HDD, kuid mugavaks arendamiseks on kindlasti soovitatav SSD kõvaketas.

3.3 Meedium

Õppematerjali meediumiks valiti PDF, mis on väga universaalne ja probleeme ei tohiks tekkida ühelgi õppuril. Meedium on väga neutraalne ja annab õppurile võimaluse materjali lugeda nii arvutist kui ka paberkandjalt. Esimesel korral soovitab autor materjali lugeda küll arvutist, et ka koodi kirjutada, kuid väga hea on lugeda õppematerjali ka paberilt, arvutist eemal.

3.4 Õpiväljundid

Materjali läbinud õppur omandab järgmised oskused:

- Oskab seadistada operatsioonissüsteemi keskkonnamuutujaid ja arenduskeskkonda, et oleks võimalik Libgdx-i kasutada.
- Suudab iseseisvalt teostada oma ideid, kasutades selleks Libgdx raamistikku ja Tiled redaktorit.

- Oskab koodi struktureerida selliselt, et rakenduse komplekssuse kasvades oleks seda võimalik ka hallata.

3.5 Ülesehitus

Õppematerjal on üles ehitatud nii, et see tuleks läbida samm-sammu haaval ja peatükke vahele jättes jäävad teadmised kindlasti ka lünklikuks. Materjal on üritatud kontstrueerida erinevat liiki õppuritele, keda on autori arvates kolm: kasutab materjali näidisena, loeb materjali läbi, kuid ei täida lisaülesandeid ja läbib õppematerjali täies mahus. Lähtudes õppurite liikidest üritab autor luua materjali, mis sobiks kõigile kolmele.

Õppematerjali sissejuhatuses kirjeldatakse, mida hakatakse looma, mis on soovitatavad eelteadmised ja soovitused, kuidas oleks hea õppematerjali omandada.

Esimeses peatükis tutvustatakse Libgdx raamistikku, õpetatakse, kuidas raamistikku paigaldada ja oma esimest programmi käivitada.

Teises peatükis näidatakse, kuidas Libgdx-i Windows-is ja Linux-is konfigureerida. Tutvustatakse ka Android-i rakendustes olevat **AndroidMainfest.xml** faili.

Kolmandas peatükis tehakse selgeks, milline näeb välja Libgdx rakenduse elutsükkel. Tutvustatakse elutsükli erinevatel etappidel väljakutsutavaid meetodeid.

Neljandas peatükis antakse lühike ülevaade plaatkaardi redaktorist **Tiled.** Näidatakse, kuidas atribuute lisada ja milline näeb välja redaktori abil loodud **XML** fail.

Viiendas peatükis luuakse näidisrakendus. Näidisrakendus luuakse samm-sammult ja igas peatükis käsitletakse ühte kindlat teemat. Igas peatükis seletatakse, mida hakatakse tegema. Järgnevalt tuuakse välja lähtekood ning seletatakse see detailideni lahti.

Kuuendas peatükis anname õppurile lahendada erinevaid ülesandeid, mis oleksis seotud viiendas peatükis käsitletud teemadega. Ülesannete läbimiseks kasutab õppur valminud näidisrakendust.

Õppematerjali lõpus antakse soovitusi, kuidas õppimist sellel teemal jätkata.

3.6 Õpiteed

Õppematerjali on soovitatav esmakordsel lugemisel kasutada arvutis, sest võimalik on vajadusel kasutada ka Libgdx kasutusjuhendit. Paberile trükitud materjali on hea kasutada näitena või ideede allikana oma rakenduse loomisel.

Õppematerjali on soovitatav kasutada koos Libgdx arendusjuhendi ja dokumentatsiooniga, et õpilane suudaks pärast õppematerjali läbimist Libgdx-is iseseisvalt orienteeruda. Õppematerjali tuleks lugeda kasutades kõiki kognitiivseid protsesse ja veenduda, et näidetest saadakse täielikult aru.

Autor ei suru peale ühte kindlat viisi, kuidas peaks õppematerjalile lähenema ja materjali võib kasutada ka näidisena, et arendada omaenda rakendust.

Näidisrakenduse arendamise käigus on õppuril siiski soovitatav peatükiga kaasasolev näidisrakendus alguses käivitada, kuid järgnevalt siiski sõna-sõnalt enda käega ümber kirjutada. Lähtekoodi kopeerimisega teadmisi ei omandata.

3.6.1 Õppimise jätkamine

Libgdx raamistiku kohta eestikeelset materjali ei leidu, kuid Libgdx arendusjuhendis leidub teemasid, mida õppur on võimeline valminud rakendusele lisama või uue rakenduse loomisel implementeerima. Arendusjuhend asub aadressil: https://github.com/libgdx/libgdx/wiki

Kui õppurit huvitab hoopis 3D rakenduse loomine võib õppur suunduda veebiaadressile http://blog.xoppa.com/basic-3d-using-libgdx-2, kust leiab kõik vajaliku selleks, et alustada 3D rakenduse loomist Libgdx raamistikuga.

Libgdx kohta leidub ka väga põhjalik raamat „Learning Libgdx Game Development“ autorilt Andreas Oehlke, mille õppur võib muretseda, kui tahab Libgdx raamistiku kõikide funktsionaalsustega tutvuda ja seda süvitsi õppida.

3.7 Õppematerjali testimine

Autor kasutas testijate valikul mugavusvalimit. Õppematerjali testimiseks oli vaja, et testijad täidaksid järgmiseid kriteeriumid:

- Eelnev kogemus Java programmeerimiskeelega
- Arenduskeskkonna IntelliJ IDEA kasutusoskus
- Huvi või eelnev kogemus lihtsate mängude arendamisel

Valituks osutusid kaks inimest, kellel lasti materjali testida. Testija 1 uuris materjali iseseisvalt ja Testija 2 läbis materjali koos autoriga. Järgnevalt toome välja testimise käigus saadud tulemused.

Testija 1

Testija 1 uuris õppematerjali iseseisvalt. Materjali testis ta umbes 150 minutit. Õppematerjali läbimine takerdus koheselt Android SDK, keskkonnamuutujate seadistamise taha ning näidisrakendust Testija 1 tööle ei saanudki.

Testijal tekkinud probleem on väga tavaline ja keskkonna töövalmis seadistamine on tihtilugu väga aeganõudev. Saadud tagasisidet kasutas autor ära ja viis õppematerjali sisse muudatused, kus üritas veelgi selgemalt ja täpsemalt õpetada, kuidas Android SDK-d installeerida ja vajalikud keskkonnamuutujad sedistada.

Testija 2

Testija 2 uuris õppematerjali koos autoriga. Testimine kestis umbes 210 minutit. Ka sellel korral oli suhteliselt raske keskkonda töökorda seada. Autori juhendamisel sai seekord Android SDK ja kõik vajalikud muutujad seadistatud.

Testimise vältel läbiti kiirelt õppematerjali esimesed neli peatükki. Testija 2 sai vähesel abistamisel esimese näidisrakenduse tööle.

Lõpetuseks sai testitud 5. peatüki esimest osa „Peaklass, mänguekraan ja sätted“. Osa läbimisel tekkisid kohe probleemid, sest kuskilt ei olnud leida täpset juhendit, kuidas antud osa käsitleda ja mida peaks täpselt tegema. Probleeme tekitas ka selle osaga kaasa antud näitekoodi avamine.

Positiivsena leidis testija, et „Rakenduse loomine“ alapeatükkide struktuur on väga hea.

Õppematerjali sai järgnevalt täiendatud juhendiga, mis selgitaks, kuidas materjali kasutada ja kuidas näitekood arenduskeskkonnas tööle saada.

Kokkuvõte

Libgdx raamistik on hea alternatiiv suurtele mängumootoritele, kui eesmärgiks on arendada 2D mäng või mõni muu arvutigraafika rakendus. Siiani puudus raamistiku kohta eestikeelne õppematerjal.

Seminaritöö eesmärgiks oli luua õppematerjal, mille abil oleks võimalik omandada teadmised, kuidas Libgdx-i kasutada, et luua 2D arvutigraafika rakendus. Eesmärk oli õppematerjali koostada selliselt, et valmiks täiesti töökorras rakendus, mis kasutaks erinevaid raamistiku osasid ja mille valmimist lugedes ning ümber kirjutades omandatakse vajalikud teadmised.

Käesolevas seminaritöös anti esiteks ülevaade Libgdx raamistikust, tema lisamoodulitest ja elutsüklist. Järgnevalt analüüsiti olemasolevate õppematerjalide positiivseid ja negatiivseid külgi. Seejärel analüüsiti õppematerjali loomist. Loomise protsessis sõnastati täpne eesmärk, defineeriti õppematerjali kasutamiseks vajalikud eelnõuded, valiti meedium, sõnastati materjali õpiväljundid, toodi välja struktuur ja võimalikud õpiteed ning käsitleti materjali testimist.

Õppematerjali koostamise käigus loodud näidiskoodi osasid on võimalik kasutusele võtta ka reaalsetes rakendustes.

Tulenevalt arenduskeskkonna komplekssusest ja süsteemide erinevusest, materjali otsast lõpuni testida ei suudetud, mistõttu ei saa täiesti kindel olla, mis võivad olla materjali puudujäägid. Üks eesmärk sai kindlasti täidetud – seminaritöö käigus valmis õppematerjal, kus luuakse täiesti töökorras rakendus, mida on õppuritel võimalik edasi arendada. Autor, kes on Libgdx kasutaja, leiab, et materjalist on kasu Libgdx ja 2D arvutigraafika huvilistele.

Autor õppis seminaritöö kirjutamise käigus Libgdx raamistikku paremini tundma. Seminaritöö käigus õppis autor ka õppematerjali koostamise protsessi, eriti seda, kuidas õppematerjali struktureerida ja asju näha läbi õppija pilgu.

Kasutatud kirjandus

Libgdx kommuun (2014, 3. märts). *Introduction*. Kasutamise kuupäev 30. märts 2014. a., allikas https://github.com/libgdx/libgdx/wiki/Introduction

Libgdx kommuun (2014, 31. märts). *The application framework.* Kasutamise kuupäev 31. märts 2014. a., allikas https://github.com/libgdx/libgdx/wiki/The-application-framework

Libgdx kommuun (2014, 11. jaanuar). *Modules overview.* Kasutamise kuupäev 31. märts 2014. a., allikas https://github.com/libgdx/libgdx/wiki/Modules-overview

Libgdx kommuun (2014, 9. märts). *Starter classes & configuration*. Kasutamise kuupäev 31. märts 2014. a., allikas https://github.com/libgdx/libgdx/wiki/Starter-classes-%26-configuration

Stegert, G. (2012*).* A Libgdx tutorial*.* Kasutamise kuupäev 7. aprill 2014. a., allikas http://www.steigert.blogspot.pt/

Batra, A., Srivastava, R. (2013). *Learn Libgdx Game Development.* Kasutamise kuupäev 8. aprill 2014. a., from http://rotatingcanvas.com/tutorials-index/

Game From Scratch (2013*). LibGDX Tutorial series*. Kasutamise kuupäev 10. aprill 2014. a., allikas http://www.gamefromscratch.com/page/LibGDX-Tutorial-series.aspx

Jano, T. (2013). *Getting Started in Android Game Development with libgdx.* Kasutamise kuupäev 14. aprill 2014. a., allikas http://obviam.net/

Patel, A. (2014). *Introduction to A*.* Kasutamise kuupäev 29. aprill 2014. a., allikas http://theory.stanford.edu/~amitp/GameProgramming/index.html

Oehlke, A. (2013). *Learning Libgdx Game Development.* Birmingham : Packt Publishing

Lisad

Lisa 1 - õppematerjal

Tallinna Ülikool

Informaatika Instituut

Raner Piibur

Libgdx raamistik ja 2D arvutigraafika õppematerjal

Tallinn 2015

Sisukord

Sissejuhatus

Käesoleva õppematerjaliga on võimalus iseseisvalt õppida Libgdx raamistikku ja 2D mängumootori loomist. Materjal peaks andma ülevaate Libgdx raamistiku põhilistest omadustest ja kuidas seal olevaid klasse rakendada, et luua töötav rakendus.

Selle õppematerjali kasutamiseks peab õppuril olema programmeerimiskogemus Java programmeerimiskeeles. Kindlasti peab õpilane omama kontseptsiooni sellest, mis on klass, polümorfism, tehase meetod, mitmelõimeline protsess, graaf, pinu, vektor, andmestruktuur. Kasuks tuleb eelnev kogemus mängude arendamises.

Õppematerjali käigus loome töötava *tower defense* stiilis mängu ning käsitleme järgnevaid teemasid:

- Libgdx raamistiku paigaldamine ja näidisprogrammi käivitamine.
- Projekti paigutus.
- Libgdx raamistiku elutsükkel.
- Rakenduse konfigureerimine.
- Mis on mänguekraan.
- Ülevaade plaatkaardi redaktorist ja TMX formaadis plaatkaardi laadimine.
- Kaamera liigutamine, suumimine ja maailmapiiride tuvastamine.
- Objektide lisamine, eemaldamine ja objektidevaheline suhtlus.
- Objektide liigutamine ja pööramine.
- Graafilise kasutajaliidese loomine Scene2D abil.
- Sisendprotsessorid ja nende eraldamine.
- Objektide liikumisraja tuvastamine, kasutades Astar algoritmi.

Mäng luuakse samm-sammult ja ei hakata tooma lihtsaid isoleerituid näiteid, mida osad materjalid kasutavad, vaid asjad üritatakse selgeks teha rakenduse loomise käigus, kus tuleb lugeda ja uurida lähtekoodi iseseisvalt. See lähenemine peaks muutma õppimise lõbusamaks ja huvitavamaks, sest lõpuks valmib midagi töötavat, mida teistele ka näidata. Ei tasu sattuda frustratsiooni, kui asjadest aru ei saada ja tihti tekib arusaamine siis, kui valmis rakendust katsetada ja muuta.

Sinu suurim ülesanne on kood oma käega ümber kirjutada. Tavapärased ülesanded toome välja täiesti lõpus ning lahendamiseks kasutage kõige viimast rakenduse versiooni. Ülesannete edukaks lahendamiseks peab olema eelnevalt läbi töötatud terve materjal. Ülesannetes käsitleme samasuguseid teemasid nagu rakenduse loomisel ja kui tekib probleeme, siis vaadata vastava peatüki lähtekoodi ja seletusi.

Intellij IDEA kasutajatel on võimalik lisainformatsiooni saada kasutades **CTRL + VASAK HIIREKLAHV** kombinatsiooni huvipakkuva meetodi või klassi peal. Selle kombinatsiooniga avatakse vastav fail ja kuna Libgdx on väga hästi kommenteeritud saab oma küsimustele tihti vastuse just sealt.

Materjal sobib viidana, kui soovitakse arendada enda rakendust, kuid ei osata kuskilt alustada. Tuleb mainida, et antu materjali käigus loodud rakendus ja selle arhitektuur ei ole ainuõige, vaid üks võimalikest lahendustest. Nii mõnigi asi on lihtsustatud ja tõsise rakenduse tarbeks oleks näiteks objektide hoidmiseks vaja luua keerulisema struktuuriga konteinerid, sektsioonideks jagatud maailma jms.

Esimesel lugemisel kirjuta kindlasti lähtekood sõna-sõnalt enda käega ümber. Vastasel juhul materjalis olevad teadmised jäävad kindlasti lünklikuks. Eriti positiivne on see, kui sa leiad, et näidiskoodis olev lahendus ei ole parim ning soovid koodi ümber teha. Võid julgelt koodi muuta.

Mugavaks kasutamiseks leian, et arvuti peaks täitma järgmised nõuded:

- Vähemalt 4GB RAM'i. Soovitatav on 8GB või rohkem.
- Haswell, Broadwell või Ivy Bridge arhitektuuriga CPU. Miinimum i5, soovitatavalt i7.
- Arendamiseks kõlbab ka HHD, kuid mugavaks arendamiseks on kindlasti soovitatav SSD kõvaketas.

Õppematerjali läbimiseks on vaja alla laadida lähtekood ja ressursid, mis asuvad aadressil https://github.com/twtChnz/justanothertowerdefense.

Õppematerjaliga paralleelselt on soovitatav uurida ka Libgdx-i arendusjuhendit aadressil https://github.com/libgdx/libgdx/wiki.

Materjali projektide loomisel kasutati Java SE Development Kit 8-t, mille saab alla laadida aadressilt http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

1 Libgdx-i paigaldamine ja käivitamine

Libgdx paigaldamiseks on vajalik, et installeeritud oleks järgmine tarkvara:

- Java Development Kit 7+ (versioon 6 ei tööta).
- Android SDK

Peale JDK ja Android SDK installeerimist on vaja konfigureerida keskkonnamuutujad, juhul kui need seadistatud ei ole.Vaja on sätestada **JAVA_HOME** muutuja, **ANDROID_HOME** muutuja ja **PATH** muutujasse lisada täideviivate (*exe*) programmide asukoht. Windowsi all on võimalik need seadistada ka kasutajaliidese abil liikudes: **Control Panel -> System and Security -> System -> Advanced system settings -> Environment Variables.** Command Prompt-is või Terminal-is saab lisada muutujad järgnevate käskudega:

Windows

set JAVA_HOME=C:\<JDK kausta tee>

set PATH=%PATH%;%JAVA_HOME%\bin

set ANDROID_HOME=C:\<Android SDK kausta tee>

set PATH=%PATH%;%ANDROID_HOME%\tools;%ANDROID_HOME%\platform-tools

Unix

export JAVA_HOME=/<JDK kausta tee>

export PATH=${PATH}:$JAVA_HOME/bin

export ANDROID_HOME=/<ANDROID SDK kausta tee>

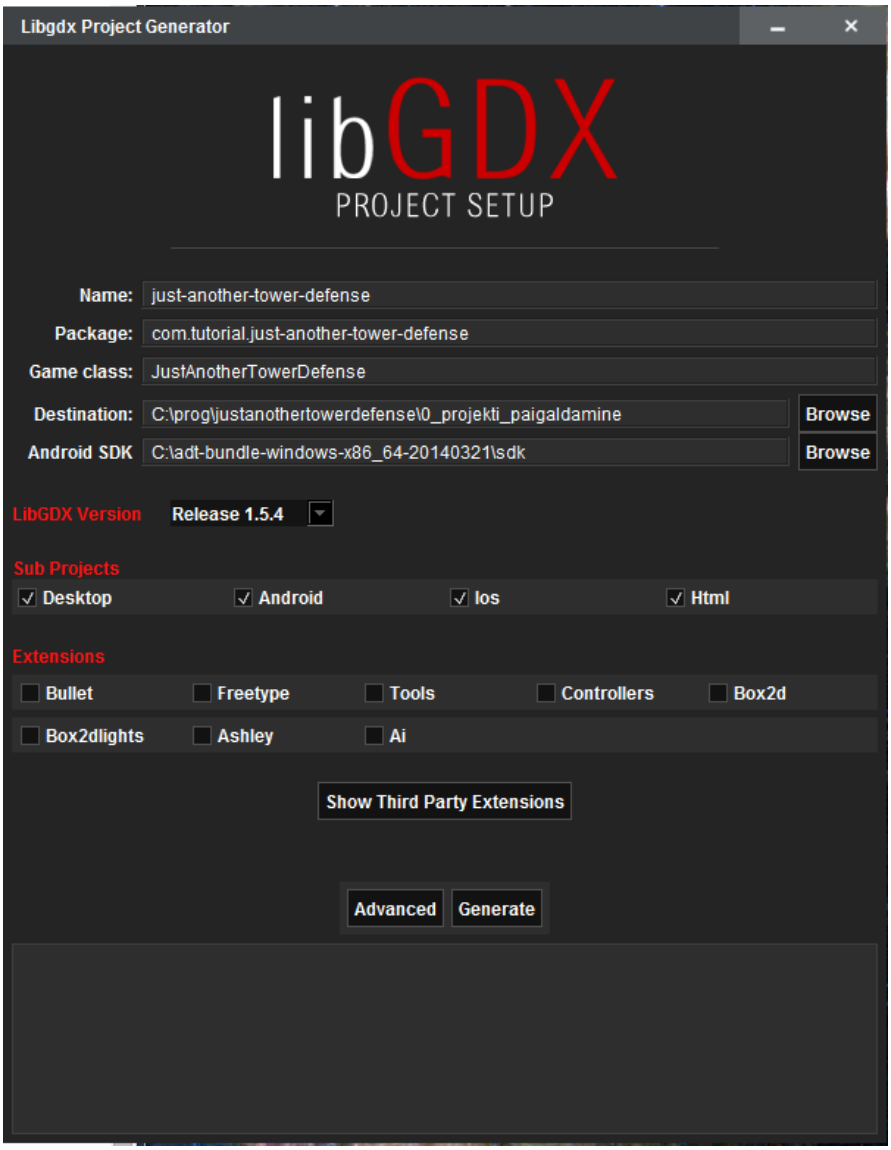
export PATH=${PATH}:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools

Libgdx paigaldamiseks on vaja alla laadida *gdx-setup.jar* veebiaadressilt:

- http://libgdx.badlogicgames.com,

Selles õppematerjalis kasutatakse Libgdx versiooni 1.5.4.

Kävitades *gdx-setup.jar*-i avaneb Libgdx paigaldaja, kus on võimalik spetsifitseerida projekti alamprojekte ja erinevaid kolmanda osapoole teeke (Joonis 1).



Joonis 4 Projekti generaator

Õppematerjalis on vajalikud Desktop ja Android alamprojektid, kuid näiteks paigaldame kõik. See installeerija vajab internetiühendust, seega ei tohiks ühendust genereerimise ajal välja lülitada.

1.1 Valminud projekti paigutus

Projekti generaatoriga tekitati 5 kausta/projekti.

- **Tuum projekt** („core“): Selles projektis asub rakendus, mida looma hakatakse, välja arvatud käivitusklassid ja vara nagu pildid, heliklipid jms. Kõik teised projektid on seotud tuum projektiga.
- **Androidi projekt** („android“): sisaldab Androidi käivitusklassi ja muid vajalikke faile, et rakendust saaks jooksutada Androidi platvormil. Sisaldab ka kõiksugu välist vara, millega on seotud kõik projektid (vara asub kaustas „assets“).

- **Töölaua projekt** („desktop“): Selles projektis on käivitusklass, et jooksutada rakendust Windowsis või Linuxis, kus toimub rakenduse arendustegevus.
- **HTML5 projekt** („html“): sisaldab käivitusklassi, et jooksutada HTML5 rakendust.
- **iOS RoboVM projekt** („ios“): sisaldab käivitusklassi ja vajalikke faile, et jooksutada rakendust iOS-is läbi RoboVM.

Kuna kõik projektid kasutavad Androidi projekti varakausta, siis kõik vajalikud failid on soovitatav salvestada „**android\assets\data**“ kausta, vastasel juhul on vaja kõike mõttetult duplikeerida.

1.2 Rakenduse käivitamine

Kõigepealt tuleb seadistada SDK Manager ja siis loodud projekt importida endale sobivasse arenduskeskkonda. Libgdx raamistiku toetatud keskkonnad on Eclipse, IntelliJ IDEA ja Netbeans. Õppematerjali raames kasutame IntelliJ Idea Community Edition arenduskeskkonda, sest võimaldab kergesti projektiga tööle asuda ilma, et peaks erinevaid laiendusi installeerima. Juhul, kui õppur mingil põhjusel ei saa või ei taha soovitatud arenduskeskkonda kasutada leiab Libgdx arendusjuhendist küllalt informaatsiooni, kuidas endale sobiv keskkond seadistada ja Libgdx projekt importida.

Aadress: https://github.com/libgdx/libgdx/wiki/Setting-up-your-Development-Environment-%28Eclipse%2C-Intellij-IDEA%2C-NetBeans%29.

1.2.1 SDK Manager-i seadistamine

Enne arenduskeskonda importimist tuleb meil seadistada Androidi SDK Manager ja installeerida põhikomponendid, mida materjalis kasutame.

Leida Android SDK installeerimiskaustast **SDK Manager.exe** ja see kävitada. On võimalik, et linnukestega on märgitud juba mingid installeerimist vajavad tööriistad. Peale nende tuleb kindlasti valida **Android SDK Build-tools 19.1** ja **Android API19**. Neid kasutame kõikides näidisprojektides, mida on võimalik https://github.com/twtChnz/justanothertowerdefense aadressilt alla laadida. Võimalik on muidugi kasutada ka uuemaid versioone, kuid siis tuleb seadistada **build.gradle** ja **AndroidManifest.xml** vastavalt valitud tööriistadele.

1.2.2 Käivitamine Intellij IDEA-s

Projekti importimiseks IDEA keskkonda tuleb läbida järgmised punktid:

- Valida **Import Project**.
- Valida kaust, kus projekt asub.
- Valida **Import project from external model** ning märkida **Gradle** ja vajutada **Next**.
- Avanenud aknas kontrollida, et valitud oleks **Use default gradle wrapper (recommended)** ja vajutada **Finish**.

Tuleks veenduda, et projekt sai õigesti loodud ja kas platvormi seaded genereeriti õigesti. Selleks Valida **File -> Project Structure -> Project**. Veenduda, et Project SDK on seadistatud. Lisaks **Platform Settings** alt veenduda, et **JDK home path** oleks JDK kaust nt C:\Program Files\Java\jdk1.8.0. Kui leidub veel probleeme, mida lahendada ei suudeta on soovitatav kontrollida, kas keskkonnamuutujad on õigesti seadistatud, järgnevalt genereerida või tõmmata uus projekt ja üritada see uuesti importida.

Kui probleemid ei esine tuleb seadistada **Run/Debug** konfiguratsioonid. Valida tööriistarealt **Run -> Edit Configurations**. Androidi seadistamine on valikuline, sest materjali loome kasutades Winows/Linux käivitajat **DesktopLauncher.**

- ***Android*** *(valikuline)*: lisada **Android Application** konfiguratsioon. „Activity“ alalt valida „Launch“ ning otsida **AndroidLauncher**, mis asub Androidi projekti kaustas. **Target Device** alalt võib kasutada enda Android seadet (seadel peab olema sisse lülitatud **USB debugging**) või kasutada emulaatorit, mida on võimalik lihtsasti luua.
- ***Windows/Linux***: lisada **Application** konfiguratsioon. **Main class** real valida töölaua projektis asuva klassi **DesktopLauncher**. Väga tähtis on, et **Working directory** real oleks valitud Androidi projektis olev varade kaust („**android\assets**“). **Use classpath of module** real valida töölaua projekt **desktop**.

Lõpuks saab projekti käivitada valides ülevalt paremalt tööriistarealt vastava konfiguratsiooni ning vajutades rohelisele noolekesele või kasutades klahvikombinatsiooni **Shift + F10**.

2 Käivitusklassid ja konfigureerimine

Erinevad käivitusklassid on põhjus, miks Libgdx raamistikuga on võimalik rakendusi arendada mitmele platvormile korraga. Igal platvormil käivitatakse vastava operatsioonisüsteemi jaoks mõeldud käivitusklass.

2.1 Windows-i/Linux-i käivitusklass

See klass (vt joonist 2) annab võimaluse muuta akna suurust ja kas kasutada **OpenGL ES** versioon 1.0-i või 2.0-i. Kui konfiguratsioon on valmis, luuakse **LWJGL** rakendus, mis vajab kahte objekti: rakenduse kuularit (ingl.k application listener) ja konfiguratsiooni faili. Terve rakenduse loogika toimub kuulari sees.

```
package com.me.justanothertowerdefense.desktop;

import com.badlogic.gdx.backends.lwjgl.LwjglApplication;
import com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;
import com.me.justanothertowerdefense.JustAnotherTowerDefense;

public class DesktopLauncher {
    public static void main (String[] arg) {
        LwjglApplicationConfiguration config = new LwjglApplicationConfiguration();
        config.title = "just-another-tower-defense";
        config.useGL30 = false;
        config.width = 480;
        config.height = 320;

        new LwjglApplication(new JustAnotherTowerDefense(), config);
    }
}
```

Joonis 5 Windows/Linux käivitusklass

2.2 Android-i käivitusklass

Androidi rakendused ei kasuta **Main** meetodit aga nõuavad hoopis **Activity** tegevust. Nagu näha on Androidi käivitusklass (vt joonist 3) eelmisega praktiliselt identne. Androidi rakendused kasutavad väga tihti erinevate osade jaoks mitut tegevust, siis Libgdx raamistik kasutab ühte. Erinevad ekraanid nagu näiteks peamenüü, mänguekraan, seaded jne. Luuakse Libgdx-i siseselt. Probleem seisneb selles, et iga uue tegevuse loomiseks on vaja luua uus **OpenGL** kontekst ja kõik ressursid tuleb uuesti laadida, mis on mõistagi ajakulukas.

```java
package com.me.justanothertowerdefense.android;

import ...

public class AndroidLauncher extends AndroidApplication {
    @Override
    protected void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();
        initialize(new JustAnotherTowerDefense(), config);
    }
}
```

Joonis 6 Android-i käivitusklass

2.2.1 AndroidManifest.xml fail

Lisaks käivitusklassis olevale konfiguratsioonile konfigureeritakse Androidi rakendust ka **AndroidManifest.xml** failis (vt joonist 4).

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.justanothertowerdefense.android"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="19" />

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="just-another-tower-defense"
        android:theme="@style/GdxTheme" >
        <activity
            android:name="com.me.justanothertowerdefense.android.AndroidLauncher"
            android:label="just-another-tower-defense"
            android:screenOrientation="landscape"
            android:configChanges="keyboard|keyboardHidden|orientation|screenSize">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Joonis 7 AndroidManifest.xml fail

Mõningad olulised teadmised on järgmised:

- Väga tähtis on, et **targetSdkVersion** oleks suurem või võrdne kui 6, mis tekitaks meile soovimatuid efekte, kui see nii ei ole. Android 2.2 API tase on 8, mida kasutavad praegusel hetkel väga vanad telefonid nagu HTC Wildfire.

- **screenOrientation** atribuudiga saab muuta rakenduse orientatsiooni, kui see omadus konfiguratsioonifailist puudub võib rakendus asetseda nii piku- kui ka laiupidi.
- **configChanges** atribuut on väga tähtis ja peab ilmtingimata sisaldama joonisel 6 olevaid väärtuseid. Juhul kui need puuduvad, käivitub rakendus igakord uuesti, kui neid tegevusi sooritada. Näiteks juhul **orientation** atribuut puudub ja rakenduse orientatsioon muutub, siis muutub ka meie rakenduse suurus ja orientatsioon, mida me muidugi üldjuhul ei soovi.

- Faili on võimalik lisada ka igasuguseid lubasid, näiteks kui meie rakendusel on vajalik juurdepääs võrku, siis on vaja lisada atribuut:

  <uses-permission android:name="android.permission.INTERNET>.

3 Rakenduse elutsükkel

Libgdx-i rakenduse elutsüklile saab ligi implementeerides **ApplicationListener** kasutajaliidese. Võimalik on laiendada ka oma põhiklassi abstraktse klassi **Game**-ga, mis haldab erinevaid ekraane, mida võib rakendus kasutada (menüü, seaded, tabloo jne). Kuna Android on sündumspõhine, siis rakendab seda ka Libgdx ja tavalist mängutsüklit raamistikust ei leia.

```
package com.me.justanothertowerdefense;

import ...

public class JustAnotherTowerDefense extends ApplicationAdapter {

    @Override
    public void create () {
    }

    @Override
    public void resize(int width, int height) {
    }

    @Override
    public void render () {
    }

    @Override
    public void pause() {
        super.pause();
    }

    @Override
    public void resume() {
        super.resume();
    }

    @Override
    public void dispose() {
        super.dispose();
    }
}
```

Joonis 8 Rakenduse elutsükkel

Elutsükkel koosneb järgnevatest meetoditest/sündmustest (vt ka joonist 5):

- **create()**: meetodit kutsutakse välja siis, kui rakendus luuakse.
- **resize()**: meetodit kutsutakse igakord, kui ekraani suurust muudetakse ja rakendus ei ole puhkestaatuses. Meetod kutsutakse korra välja ka päris **create()** meetodit. Parameetriteks on rakenduse laius ja kõrgus.
- **render()**: meetod, mis kutsutakse välja igakord, kui on vaja pilt uuesti joonistada. Toimub mängutsükkli sees. Samas meetodis toimuvad ka tavaliselt mänguloogika uuendused.
- **pause()**: Androidi platvormil kutsutakse meetod välja juhul, kui „Home“ nuppu on vajutatud või sisse on tulemas kõne. Windowsis/Linuxis kutsutakse meetod välja enne **dispose()** meetodit. Tavaliselt toimub siin mängustaatuse salvestamine.
- **resume()**: meetod kutsutakse välja siis, kui rakendus jätkab pärast puhestaatuses väljatulemist.
- **dispose():** kutsutakse välja siis, kui rakendus hävitatakse.

4 Ülevaade Tiled redaktorist

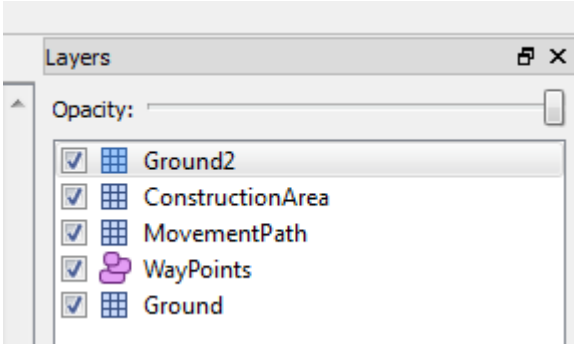
Selles õppematerjalis kasutame maailma loomiseks nö plaatkaardi (ingl k tiled map) põhimõtet. Järgnevalt anname lühiülevaate, mida plaatkaart endast kujutab.

Plaatkaardi põhimõte tähendab, et mängumaastik luuakse 2-dimensioonilises massiivis olevatest viitadest. Viidad sisaldavad endas informatsiooni mingi ala kohta maailmas. Informatsiooniks on näiteks, mis tekstuur sellel alal on, kas alal on võimalik liikuda jne. Seda põhimõtet kasutades on võimalik luua endale sobilik mõõtkava ja kuna maailm on jagatud väikesteks tavaliselt ruudukujulisteks tükkideks on mälukasutus väiksem ning jookseb paremini ka nõrgemates masinates. Kuna Libgdx-s on moodul, mis võimaldab kasutajal laadida **.tmx** formaadis kaarte sai välja valitud Tiled redaktor (versioon 0.9.1). Võimalik on kasutada ja ise luua maastik ükskõik millist plaatkaardi redaktorit kasutades, peaasi, et ülesehitus oleks samasugune, mis meie materjalis, et see ka lähtekoodiga töötaks.

Esiteks tuleb alla laadida Tiled redaktor addressilt http://www.mapeditor.org/download.html ja see installeerida.

Järgmiseks võib käivitada Tiled redaktori ja avada autori poolt juba loodud faili „level1.tmx“, mis tuli materjaliga kaasa ja asub „data“ kaustas. Juhul kui õppija ei ole veel materjali alla laadinud, siis asuvad kõik vajalikud failid aadressil https://github.com/twtChnz/justanothertowerdefense.

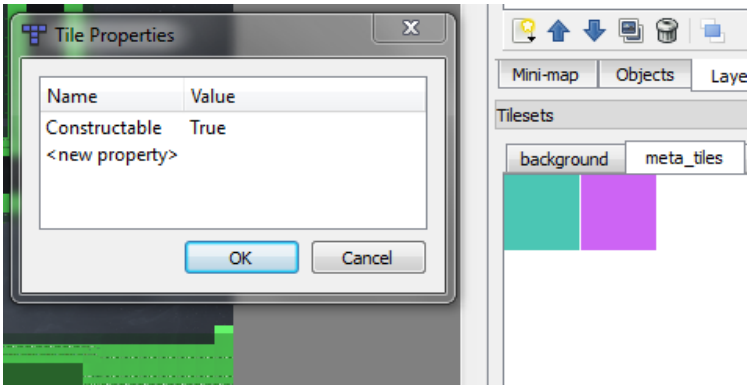
Autori poolt loodud maailm koosneb viiest kihist (vt joonist 6). Kihte on võimalik välja lülitada vajutades linnukesele nime kõrval.



Joonis 9 Maailma kihid

- **Ground**: plaatkiht, mis on maailma taustaks.
- **Ground2**: plaatkiht, mida saab kasutada maastikku loomiseks. Maastik luuakse kasutades plaathulki (ingl. k tileset), mis on eksporditud mingist graafika failist. Soovitatav on kasutada .PNG formaati, sest sisaldab informatsiooni läbipaistvuse kohta.

- **ConstructionArea**: sisaldab informatsiooni selle kohta, kuhu on võimalik maailmas ehitada. Selleks kasutame „metaplaate“, millel on üks atribuut, mis on siis kas tõene või väär (vt joonist 7).
- **MovementPath**: sisaldab informatsiooni ala kohta, kus on võimalik maailmas liikuda.
- **WayPoints**: erinevalt teistest on see kiht objektikiht. Selle kihi eesmärk on anda informatsiooni näiteks selle kohta, kust ilmuvad vaenlased, kus asub finiš jms.



Joonis 10 Ehituse metaplaadi atribuut

4.1 Uue plaathulga ja atribuudi lisamine

Uut plaathulka saab lisada läbides järgmised sammud:

- Valida tööriistarealt **Map -> New Tileset**.
- Valida nimi ja asukoht.
- **Tile width** ja **Tile height** määravad ühe ruudu laiuse ja pikkuse. Peaksid üldjuhul olema võrdsed. Meie kasutame48x48 piksli suuruseid plaate.
- **Drawing Offset** x ja y koordinaatide nihutust kasutatakse harva. Mõte on selles, et plaati nihutada algpositisioonist kõrvale. Ei soovita kasutada.
- **Margin** määrab ära, kui suur on ühe plaadi piir.
- **Spacing** määrab ära, kui suur on vahe kahe plaadi vahel.

Uue plaathulga lisamisel jagatakse kasutaja poolt etteantud tekstuur niisuurteks osadeks nagu kasutaja spetsifitseeris. Kindlasti tuleb jälgida, et tekstuuri suurus oleks 2. aste, näiteks 256x128 pikslit. Tegu on optimiseerimisega, sest riistavaraliselt on korrutamine ja jagamine 2. astmega kiirem, lisaks on erinevad operatsioonid järjekorras (ingl. k pipeline) kiiremad ja lihtsamad.

Atribuuti saab plaathulga ühele plaadile lisada vajutades parema hiireklahviga plaadile ja valides **Tile Properties**. Atribuut võib omandada ükskõik, mis väärtust ja on seega väga paindlik.

4.2 TMX faili struktuur

TMX fail on struktuurilt XML fail (vt joonist 8)

- Peaelemendiks on element **map**. See element sisaldab kõike vajalikku sellejaoks, et piiritleda meie maailma: **width** ja **height** – mitmest plaadist maailma koosneb ning **tilewidth** ja **tileheight** – kui suur on üks plaat pikslitest.
- Element **tileset** sisaldab plaathulga pildi/tekstuuri allikat ja lisaks selles hulgas asuvate plaatide atribuute.
- Element **layer** tähistab erinevaid kihte ja sisaldab infot plaatide paiknevuse kohta.
- Element **objectgroup** sisaldab endas erinevaid objekte, millel on oma koordinaadid ja nagu ka jooniselt 10 näha, siis x ja y koordinaadid ei ole mõõtkavasse viidud, seega enne kui nendega tegeleda tuleks need konverteerida.

```
<?xml version="1.0" encoding="UTF-8"?>
<map version="1.0" orientation="orthogonal" width="42" height="42" tilewidth="48" tileheight="48">
 <tileset firstgid="1" name="background" tilewidth="48" tileheight="48">
  <image source="background.png" width="2048" height="2048"/>
 </tileset>
 <tileset firstgid="1765" name="meta_tiles" tilewidth="48" tileheight="48">
  <image source="meta_tiles.png" trans="00ff00" width="128" height="64"/>
  <tile id="0">
   <properties>
    <property name="Constructable" value="True"/>
   </properties>
  </tile>
  <tile id="1">
   <properties>
    <property name="Walkable" value="True"/>
   </properties>
  </tile>
 </tileset>
 <tileset firstgid="1767" name="ground_tiles" tilewidth="48" tileheight="48">
  <image source="ground_tiles.png" trans="00ff00" width="256" height="256"/>
 </tileset>
 <tileset firstgid="1792" name="ground_tiles_connectors" tilewidth="48" tileheight="48">
  <image source="ground_tiles_connectors.png" trans="00ff00" width="256" height="256"/>
 </tileset>
 <tileset firstgid="1817" name="ground_tiles_outer" tilewidth="48" tileheight="48">
  <image source="ground_tiles_outer.png" trans="00ff00" width="256" height="256"/>
 </tileset>
 <layer name="Ground" width="42" height="42">
  <data encoding="base64">
   AQAAAAIAAAADAAAABAAAAAUAAAAGAAAABwAAAAgAAAAJAAAACgAAAAsAAAAMAAAADQAAAA4AAAAPAAAAEAAAABEAAAASAAAAE
  </data>
 </layer>
```

Joonis 11 .TMX faili struktuur

5 Rakenduse loomine

Järgnevalt loome *tower defense* stiilis mängu näidise. Mäng on pealtvaates ja hiire või näpuga on võimalik maailmas ringi liikuda. Loomulikult on taolises mängus vaenlased, mis ühest kaardi otsast teise liiguvad ning kaitseehitised, mis neid hävitada püüavad.

Kuidas õppida?

Kasuta eelnevalt loodud projekti, mida hakkad järgnevate peatükkide vältel täiustama. Kirjuta kood sõna-sõnalt enda käega ümber ja kasuta peatükkides toodud seletusi, et koodist aru saada. Vajadusel uuri ka koodis kasutatud Libgdx meetodeid ja klasse, kasutades **CTRL + VASAK HIIREKLAHV** kombinatsiooni huvipakkuva meetodi või klassi peal.

Iga peatükiga on kaasa antud lähtekood, mille leiad veebiaadressilt:

- https://github.com/twtChnz/justanothertowerdefense

Lähtekoodid asuvad peatükiga samanimelises kaustas. Kaustas oleva projekti saada avada importides need samamoodi nagu näitasime peatükis **1.2.2 Käivitamine IntelliJ IDEA-s.**

Peatükkide struktuur

Iga peatüki alguses seletame, mida tahame saavutada. Järgnevalt toome välja lähtekoodi, mis on vaja sul oma käega ümber kirjutada. Pärast lähtekoodi seletame lahti konstruktori, muutujad ja funktsioonid.

5.1 Peaklass, mänguekraan ja sätted

Enne kui midagi üldse ekraanile kuvada, peame looma mitmeid klasse, mis on rakenduse alustaladeks. Võib tunduda, et kuidagi liiga palju tööd, kuid rakenduse komplekssuse kasvades on see ülimalt vajalik.

5.1.1 Peaklass

Peaklass („**JustAnotherTowerDefense.java**“) on nö stardipunkt, kust rakendus alguse saab, kuid peaklass ei ole hea koht, kus toimuks terve mänguloogika. Peaklass peaks olema koht, kus toimub erinevate ekraanide (tabloo, seaded jne) vahetamine. Laiendades peaklassi klassiga **Game** on meil võimalus vahetada ekraane.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Game;
```

```
import com.badlogic.gdx.Gdx;

public class JustAnotherTowerDefense extends Game {

    @Override
    public void create() {
        setScreen(new GameScreen());
    }

    @Override
    public void dispose() {
    }

    @Override
    public void render() {
        getScreen().render(Gdx.graphics.getDeltaTime());
    }

    @Override
    public void resize(int width, int height) {
        Settings.WIDTH = width;
        Settings.HEIGHT = height;
    }

    @Override
    public void pause() {
    }

    @Override
    public void resume() {
    }
}
```

Peaklass on suhteliselt lihtne. Rakenduse käivitamise ajal loome uue mänguekraani ja määrame selle atkiivseks ekraaniks meetodiga **setScreen()**. Selle meetodiga on võimalik hiljemalt mänguekraane vahetada.

Joonistamistsüklis **render()** anname hetkel aktiivsele ekraanile delta aja (ingl. k delta time) - praeguse ja eelmise kaadri ajavahe millisekundites - mis on vajalik selleks, et erineva jõudlusega arvutid jooksutaks rakendust sama kiirusega.

Meetodiga **getScreen()** on võimalik kätte saada hetkel aktiivne ekraan ja seal omakorda **render()** meetod välja kutsuda. Rakenduse suuruste muutmise sündmuses **resize()** anname uued mõõtmed sättete klassis asuvatele globaalsetele muutujatele.

5.1.2 Mänguekraan

Mänguekraan („**GameScreen.java**“) on klass, kus peaks toimuma rakenduses vajalike objektide loomine, sest teeb testimise kergemaks. Samuti on see koht, kus toimub tasandite vahetamine, tasandis vajalike objektide loomine ja seadistamine. Ekraanide vahetamisel või rakenduse sulgedes tuleb kindlasti **pause()** meetodi abil rakenduse olek salvestada.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Screen;

public class GameScreen implements Screen {
```

```
    public GameScreen() {
    }

    @Override
    public void render(float deltaTime) {

    }

    @Override
    public void resize(int width, int height) {

    }

    @Override
    public void show() {

    }

    @Override
    public void hide() {

    }

    @Override
    public void pause() {

    }

    @Override
    public void resume() {

    }

    @Override
    public void dispose() {

    }

}
```

Ekraan on väga sarnane peaklassile. Käivitusmeetodi asemel kasutame käivitamiseks konstruktorit, kus toimub vajalike objektide loomine. Joonistusmeetodis on meil kaks uut käsku, mis on ka ainukesed käsud, mida on vaja teada OpenGL-i kohta, kui kasutada klasse **Texture** ja **SpriteBatch**. Meetod **glClearColor**() määrab ekraani puhastusvärvi. Esimesed 3 parameetrit defineerivad RGB värvi, mille väärtused võivad olla 0-1-ni ning 4. parameeter määrab läbipaistvuse. Käsk **glClear()** puhastab ekraani eespool defineeritud värviga.

5.1.3 Sätted

Sättete („**Settings.java**“) all on mõningased globaalsed muutujad, näiteks akna suurus, nuppude suurus, graafika kvaliteet jms. Tulevikus on võimalik seadeid näiteks XML failist sisse lugeda.

```
package com.me.justanothertowerdefense;

public class Settings {

    public static int WIDTH = 1280;
    public static int HEIGHT = 720;
    public static int VIEWPORT_SCALE = 45;

    public static int UI_WIDTH_SCALE = 7;
```

```
    public static float CAMERA_ZOOM = 1f;

    public static boolean DEBUG = false;

}
```

- **WIDTH** ja **HEIGHT:** akna suurus pikslites.
- **VIEWPORT_SCALE:** jagamisel akna dimensioonidega annab meile plaatide arvu aknas. Ei ole teadus, vaid muutujat häälestades saab saavutada tulemus, mis tundub kõige mõistlikum.
- **UI_WIDTH_SCALE**: jagamisel akna laiusega saame kasutajaliidese tabeli laiuse.
- **CAMERA_ZOOM:** määrab ära, kas maailm on sisse- või välja suumitud. Arv väiksem, kui üks suumib sisse.
- **DEBUG**: määrab, kas kuvatakse konsoolis informatsiooni, kasutajaliidese tabelite piirdeid jms. Kindlasti tuleb valmis rakenduses välja lülitada, sest raiskab mobiilsetes seadmetes niigi vähest ressurssi.

5.2 Mänguobjektide konteiner ja TMX faili laadmine

Loodavas mängus kasutame *TMX* formaadis loodud plaatkaarti. Kaardi abil saame joonistada maastikku, mis sai redaktoris loodud. Loodud kaart on vaja kuidagi rakendusse laadida ja sealt informatsiooni saada. Meil on vaja ka konteinerit, kus mänguobjekte hoida ja vajadusel kustutada.

5.2.1 Mänguobjektide konteineri loomine

Esiteks loome konteineri, mis täidaks eelpool soovitud tingimusi („**World.java**“). Kaardi laadimisel peame saama kätte liikumiskihi ja ehitamiskihi, et teada saada, mis aladel on võimalik liikuda ja ehitada. Lisaks peaks teada saama maailma suurused ja erinevad kontrollpunktid, mida saame kasutada punktidena, kuhu vaenlased ilmuvad. Mõistlik on luua 2-dimensioonilised massiivid, mis hoiaksid lihtsat jah- ja ei informatsiooni kihtide kohta.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.maps.MapLayer;
import com.badlogic.gdx.maps.MapProperties;
import com.badlogic.gdx.maps.tiled.TiledMap;
import com.badlogic.gdx.maps.tiled.TiledMapTileLayer;
import com.badlogic.gdx.math.Vector2;

enum TileState {
    UNWALKABLE(0), WALKABLE(1), UNCONSTRUCTABLE(2), CONSTRUCTABLE(3);
    private int state;

    private TileState(int state) {
        this.state = state;
    }

    public int getState() {
        return state;
    }
```

```
}

public class World {

    private TiledMap map;

    private TiledMapTileLayer groundLayer;
    private TiledMapTileLayer movementPathLayer;
    private TiledMapTileLayer constructionLayer;
    private MapLayer wayPointsLayer;

    private Vector2 startPosition = new Vector2();
    private Vector2 endPosition = new Vector2();

    private Vector2 mapDimensions = new Vector2();
    private Vector2 viewPortDimensions = new Vector2();
    private float unitScale;

    private int[][] constructionArea;
    private int[][] movementPath;

    public World(TiledMap map) {
        this.map = map;
        setLayers();
        calculateWorldSizes();
        setStartAndEndPosition();
        constructionArea = createConstructionAreaArray();
        movementPath = createMovementPathArray();

    }

    public TiledMap getMap() {
        return map;
    }

    public float getMapIntegerProperty(String property) {
        return map.getProperties().get(property, Integer.class);
    }

    public Vector2 getMapDimension() {
        return mapDimensions;
    }

    public Vector2 getViewPortDimension() {
        return viewPortDimensions;
    }

    public float getUnitScale() {
        return unitScale;
    }

    private void setLayers() {
         groundLayer = (TiledMapTileLayer).getLayers().get("Ground");

        movementPathLayer = (TiledMapTileLayer) map.getLayers().get("MovementPath");
        movementPathLayer.setOpacity(0f);

        constructionLayer = (TiledMapTileLayer) map.getLayers().get("ConstructionArea");
        constructionLayer.setOpacity(0f);

        wayPointsLayer = map.getLayers().get("WayPoints");
        wayPointsLayer.setOpacity(0f);
    }

    private void calculateWorldSizes() {
        mapDimensions.x = getMapIntegerProperty("width");
        mapDimensions.y = getMapIntegerProperty("height");
        unitScale = getMapIntegerProperty("tilewidth");

        viewPortDimensions.x = Gdx.graphics.getWidth() / Settings.VIEWPORT_SCALE;
        viewPortDimensions.y = Gdx.graphics.getHeight() / Settings.VIEWPORT_SCALE;
    }

    private void setStartAndEndPosition() {
        MapProperties startProperties = wayPointsLayer.getObjects()
                                        .get("StartWayPoint").getProperties();
```

```
        startPosition.set(
                  startProperties.get("x", Float.class) / unitScale,
                startProperties.get("y", Float.class) / unitScale
                     );

        MapProperties endProperties = wayPointsLayer.getObjects()
                                .get("EndWayPoint").getProperties();

           endPosition.set(
                     endProperties.get("x", Float.class) / unitScale,
                     endProperties.get("y", Float.class) / unitScale
                     );
    }

     private int[][] createMovementPathArray() {
         int[][] pathArray = new int[movementPathLayer.getWidth()]
                                    [movementPathLayer.getHeight()];

         for (int x = 0; x < movementPathLayer.getWidth(); x++) {
            for (int y = 0; y < movementPathLayer.getHeight(); y++) {

                    if (movementPathLayer.getCell(x, y) != null
                        &&
                      Boolean.parseBoolean(movementPathLayer.getCell(x, y)
                                                            .getTile()
                                                            .getProperties()
                                                           .get("Walkable", String.class)))
                        pathArray[x][y] = TileState.WALKABLE.getState();
                    else
                        pathArray[x][y] = TileState.UNWALKABLE.getState();

                }
            }

         return pathArray;
     }

     private int[][] createConstructionAreaArray() {
         int[][] constructionArray =
                                        new int[constructionLayer.getWidth()]
                                               [constructionLayer.getHeight()];

         for (int x = 0; x < constructionLayer.getWidth(); x++) {
            for (int y = 0; y < constructionLayer.getHeight(); y++) {
                 if (constructionLayer.getCell(x, y) != null
                             &&
                       Boolean.parseBoolean(constructionLayer.getCell(x, y)
                                                            .getTile()
                                                            .getProperties()
                                                            .get("Constructable", String.class)))
                     constructionArray[x][y] = TileState.CONSTRUCTABLE.getState();
                 else
                     constructionArray[x][y] = TileState.UNCONSTRUCTABLE.getState();
                }
            }

         return constructionArray;
     }
}
```

Konstantide hoidla TileState loomine

Tuleb luua eeldefineeritud konstantite hoidmise klass, kuna plaatide olekud võivad arenduse käigus ja uute tasemete loomisel muutuda. Vigade vältimiseks ja loetavuse parandamiseks on see klass ülimalt vajalik. Klass hoiab nelja erinevat olekut: plaadil on võimalik liikuda, plaadil ei ole võimalik liikuda, plaadile ei saa ehitada ja plaadile saab ehitada.

Muutujate defineermine

Defineerime kõik kihid, mida meil vaja läheb (**groundLayer, movementLayer** ja **constructionLayer**). Need muutujad on **TiledMapTileLayer** objektid, mille abil saab küsida kihi mõõtmeid ja ruute. Võimalik on ka ruute muuta ja keerata.

Vaenlaste ilmumis- ja lõpp-punkt, kaardi- ja vaateava dimensioonid ning mõõtkava on **Vector2** objektid. Klass võimaldab meil teha tehteid vektoritega. Tuleb meeles pidada, et kasutades ükskõik, millist meetodit, muudab meetod vektori enda koordinaate, mitte ei tagasta uut objekti.

Lisaks defineerime 2-dimensioonilised massiivid hoidmaks informatsiooni selle kohta, kus saab maailmas liikuda ja kuhu ehitada.

Konstruktor

Konstruktoril on üks parameeter, milleks on Libgdx klassi **TmxMapLoader** abil loodud plaatkaart. Erinevat liiki muutujate määramiseks kutsume välja selleks operatsiooniks vajalikud meetodid.

Meetod getMap()

Kuna meil läheb vaja plaatkaarti ka Libgdx joonistusklassis on meil vajalik anda ligipääs kaardile ka väljaspoolt, seega defineerime tavalise **get()** meetodi.

Meetod getMapIntegerProperty()

Meil tuleb mitmeid kordi küsida informatsiooni kaardi erinevate täisarvuliste omaduste kohta ja mõistlik on luua eraldi funktsioon, mis seda operatsiooni täidaks.

Meetod **getProperties()** tagastab **MapProperties** objekti, millega on võimalik kaardi omadusi lisada, küsida ja kustutada. Olemasolevaid omadusi saab näha avades Androidi projektis asuvas „**assets/data**“ folderis oleva **level1.tmx** faili.

Meetodid getMapDimension(), getViewPortDimension() ja getUnitScale()

Lisaks on meil vaja väljaspoolt ligipääsu kaardi- ja vaateava dimensioonidele ning informatsiooni mõõtkava kohta. Selleks loome tavalised **get** meetodid.

Meetod setLayers()

Plaatkaardi sisselugemise vajame informatsiooni kolmelt erinevalt kihilt: maastik **Ground**, liikumisala **MovementPath** ja ehitusala **ConstructionArea**. Mõistlik on määrata igale kihile eraldi muutuja.

Kuna **get()** meetod tagastab klassi **MapLayer**, aga meil on vaja lisameetodeid, mis asuvad klassis **TiledMapTileLayer** peame me **MapLayer** objekti vormima (ingl. k cast) **TiledMapTileLayer** objektiks. Me ei soovi ka, et liikumis- ja ehituskiht oleks nähtav ning selleks kasutame meetodit **setOpacity()**, millega määrame kihi läbipaistvaks.

Meetod *calculateWorldSizes()*

Maailma piiride tuvastamiseks ja kaamera vaateava määramiseks on meil vajalik määrata maailma erinevad dimensioonid.

Määrame kaardi x ja y dimensioonid. Määrame ka ühiku skaala, milleks võib kasutada nii plaadi laiust, kui ka pikkust. Kuna **unitScale** muutuja on skalaararv on see ka põhjuseks, miks plaadid peaksid olema ruudukujulised. Vaateava dimensioonid on akna dimensioonide suhe sättetes oleva **VIEWPORT_SCALE** muutujaga.

Meetod *setStartAndEndPosition()*

Tower Defense mängudes ilmuvad vaenlased tavaliselt kaardi ühest punktist ja liiguvad läbi kaardi teise punkti. Vajame kaardilt informatsiooni nende punktide koordinaatide kohta.

Punkti omaduste saamiseks on vaja küsida kontrollpunktide kihi objekte meetodiga **getObjects()**, järgnevalt vastavanimelist objekti meetodiga **get()** ning siis tema omadusi meetodiga **getProperties()**. Punktid on vaja viia õigesse mõõtkavasse, mistõttu tuleb koordinaate jagada skalaararvuga **unitScale**.

Meetod *createMovementPathArray()*

Vaja on määrata liikumisala massiiv, millega oleks võimalik kiiresti määrata, kas plaadile on võimalik liikuda või mitte.

Meetodi alguses loome 2-dimensioonilise massiivi, mille mõõtmed on liikumisala kihi mõõtmed. For-tsüklis kontrollime, kas plaat on defineeritud, ja kas plaadil on võimalik liikuda ning määrame olekuks **WALKABLE** või **UNWALKABLE**.

Meetod *createConstructionPathArray()*

Samamoodi tuleb luua ka ehitusala massiiv. Meetod on praktiliselt identne eelmisele meetodile, seega ei vaja seletust.

5.2.2 Konteineri defineerimine ja TMX faili laadimine

Lõpuks võime mänguekraani („**GameScreen.java**“) objektis konteineri luua.

Esiteks tuleb importida vajalik klass.

```
import com.badlogic.gdx.maps.tiled.TmxMapLoader;
```

Järgmiseks tuleb defineerida muutuja.

```
    private World world;
```

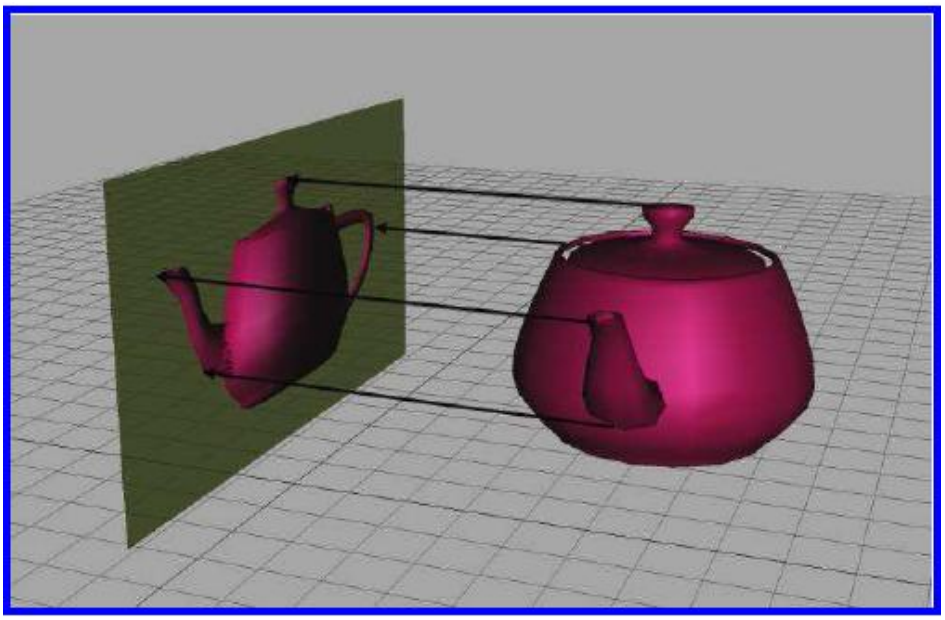
Viimaks tuleb objekt konstruktoris määrata.

```
        world = new World(new TmxMapLoader().load("data/level1.tmx"));
```

Libgdx raamistikus on moodul, millega on võimalik TMX formaadis faile lugeda. Laadimiseks kasutame laadijas olevat **load()** meetodit, mille parameetriks on faili asukoht töötavas kaustas (töötav kaust on Androidi projektis asuv „assets“ kaust).

5.3 Ortograafiline kaamera

Rakenduses kasutame ortograafilist kaamerat, mida kasutatakse 2D keskkonnas, kuna rakendab ortograafilist projektsiooni (vt joonist 9). Kaamera on vaja luua enne, kui maailma renderdama hakatakse, sest sisaldab informatsiooni vaateava paiknevuse kohta. Ilma kaamerata ei ole meil võimalik ekraanil midagi näha.



Joonis 12 Ortograafiline projektsioon (3D Math Primer for Graphics and Game Development (2nd Edition))

Ortograafilise projektsiooni korral ei arvestata mõõdet, mis on kaamera vaateavaga risti, meie rakenduse puhul seega z-koordinaat. Näitena võib tuua CAD-programmides olev külgvaade, kus näeme ainult kahte mõõdet.

Libgdx kaameraga on võimalik maailmas ringi liikuda, suumida sisse ja välja, vahetada vaateava ning projekteerida punkte rakenduse aknasse/maailma.

Kaamera loome mänguekraanis („**GameScreen.java**“). Enne on vaja kindlasti teada maailma dimensioone, sest muidu ei tea me kaamera vaateava/vaateulatust.

Kaamera defineerimine.

```
    private OrthographicCamera camera;
```

Konstruktorisse vajalike meetodite lisamine

```
    public GameScreen() {
        world = new World( new TmxMapLoader().load("data/level1.tmx"));
        setCamera();
    }
```

Loome konstruktoris uue kaamera ja lisame meetodi **setCamera()** kutse.

Meetodi setCamera() loomine

```
    private void setCamera() {
        camera = new OrthographicCamera();
        camera.setToOrtho(false,
                                world.getViewPortDimension().x,
                                world.getViewPortDimension().y);

        camera.zoom = Settings.CAMERA_ZOOM;
        camera.update();
```

Esiteks loome uue kaamera. Järgmiseks määrame kaamera ortograafilise projektsiooni, kus esimene parameeter on kas y-koordinaat näitab alla, teised kaks on vaateava dimensioonid. Kolmandaks määrame kaamera suumi, mis tuleb sättetest. Lisaks on pärast igat kaamerale tehtud muudatust vaja kaamerat uuendada. Kaamera uuenduse meetodis „update“ toimub projektsiooni maatriksi ümber kalkuleerimine ilma milleta aknale midagi ei ilmu või vaateava on vale.

5.4 Maailma renderdamine

Mängumootoris on hea tava renderdamine hoida mänguloogikast eraldi ja selle jaoks loome maailmarenderdaja („**WorldRenderer.java**“). Renderdajas toimub ainult objektide joonistamine ja mitte midagi muud.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;

public class WorldRenderer {

    private World world;
    private OrthographicCamera camera;
    private OrthogonalTiledMapRenderer renderer;

    private SpriteBatch batch;
    private BitmapFont font;

    public WorldRenderer(World world, OrthographicCamera camera) {
        this.world = world;
        this.camera = camera;
        this.batch = new SpriteBatch();
        this.font = new BitmapFont();

        renderer = new OrthogonalTiledMapRenderer(world.getMap(), 1 / world.getUnitScale());
    }

    public void render(float deltaTime) {
        camera.update();
        renderer.setView(camera);
        renderer.render();

        batch.begin();
        font.draw(batch, "FPS: " + Gdx.graphics.getFramesPerSecond(), 10, 20);

        batch.end();
    }
}
```

OrthogonalTiledMapRenderer klassi abil joonistamine

Objektide ja maastiku renderdamiseks kasutame Libgdx-s olevat **OrthogonalTiledMapRenderer** klassi, mis vajab **TiledMap** objekti ja maailma mõõtkava vahemikus 0-1. Tuleb meeles pidada, et enne objektide renderdamist tuleb kaamera kindlasti uuendada kasutades **update()** meetodit. Kaamera asukohta muutes tuleb renderdajas välja kutsuta meetod **setView()**, mis sätestab uue asukoha maailmas. Viimaseks operatsiooniks on vaja renderdajas kutsuda välja meetod **render()**, mis plaatkaardi aknasse joonistab.

SpriteBatch klassi kasutamine

Maailmarenderdajas kasutame ka **BitMapFont** objekti, mis laseb meil ekraanile renderdada teksti. Tekstuuride renderdamisel kasutades **SpriteBatch** objekti või mõnda muud objekti on vajalik, et see toimuks **begin()** ja **end()** meetodite sees. Iseärasus tuleneb sellest, et OpenGL-s on vaja enne objekti renderdamist sätestada programm, millega objekti ekraanile joonistatakse ning erinevate objektide jaoks on võimalik kasutada erinevaid programme. **BitMapFont** objekt vajab joonistamiseks **SpriteBatch** objekti, joonistatavad teksti ja teksti asukoha koordinaate.

Meetod getFramesPerSecond()

Libgdx-s on sisseehitatud meetod, mis võimaldab meil teada saada mitu kaadrit sekundis ekraanile renderdatakse, selleks meetodiks on **getFramesPerSecond()**, mis asub **Graphics** liideses.

5.5 Kaamera liigutamine sisendprotsessori abil

Kaamera liigutamiseks kasutame Libgdx liidest **InputProcessor.** Klass muretseb selle eest, et saadakse infot nii perifeeriaseadmetelt nagu klaviatuur ja hiir kui ka mobiilsetelt seadmetelt puudutuse abil.

Sisendprotsessori kaudu implementeeritakse järgmised meetodid:

- **touchDragged():** kutsutakse välja, kui hiire klahv on alla vajutatud ja seda lohistatakse ekraanil või näppu lohistatakse mobiilse seadme ekraanil. Annab teada asukoha koordinaadid ja osuti indeksi.
- **touchUp():** kutsutakse välja, kui näpp tõstetakse ekraanilt ära või hiireklahv lastakse vabaks. Annab teada viimased teadaolevad koordinaadid, osuti indeksi ja lahti lastud hiire klahvi (puutetundlikel ekraanidel alati **Buttons.LEFT**).
- **touchDown()**: kutsutakse välja, kui näpp pannakse ekraanile või vajutatakse alla hiireklahv. Annab teada koordinaadid, osuti indeksi ja alla vajutatud hiire klahvi (puutetundlikel ekraanidel alati **Buttons.LEFT**).
- **keyDown():** kutsutakse välja, kui klahv vajutatakse alla. Annab teada klahvi koodi.
- **keyUp():** kutsutakse välja, kui klahv lastakse lahti. Annab teada klahvi koodi.
- **keyTyped():** kutsutakse välja, kui klaviatuuri abil genereeritakse *Unicode* tähemärk. Implementeerides tekstialasid ja sarnaseid kasutajaliidese elemente on seda meetodit hea kasutada.
- **mouseMoved():** kutsutakse välja, kui hiir liigub üle ekraani. Ei ole võimalik kasutada mobiilsetes seadmetes, mis kasutavad puutetundlikku ekraani.
- **scrolled():** kutsutakse välja, kui hiire kerimisrulli keerati. Annab teada arvu vahemikus -1 kuni 1 olenevalt sellest, mis suunas rulli keerati.

Iga sisendprotsessori meetod tagastab *boolean* tüüpi muutuja. See omadus on vajalik selleks, et kui kasutada sisendmultiplekserit, mida me ka teeme, siis tõese väärtuse korral ei hakkaks multiplekser manatud sündmust sisendprotsessorite ahelas järgmistele protsessoritele jagama.

5.5.1 Kaamera sisendprotsessori loomine

Kaamera liigutamiseks on meil vaja luua kaamera sisendprotsessor („**CameraInputProcessor.java**“), kasutades **InputProcessor** kasutajaliidest. Klass tegeleb kaamera liigutamisega lohistamise teel ja tal on omadus tuvastada kaardi piire.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.math.Vector3;

public class CameraInputProcessor implements InputProcessor {

    private final Vector3 currentLocation = new Vector3();
    private final Vector3 lastLocation = new Vector3(-1, -1, -1);
    private final Vector3 delta = new Vector3();

    private OrthographicCamera camera;

    private float mapWidth;
    private float mapHeight;


    public CameraInputProcessor(OrthographicCamera camera, float mapWidth, float mapHeight) {
        this.camera = camera;
        this.mapWidth = mapWidth;
        this.mapHeight = mapHeight;
    }

    @Override
    public boolean touchDragged (int x, int y, int pointer) {
        camera.unproject(currentLocation.set(x, y, 0));
        if (!(lastLocation.x == -1
                    && lastLocation.y == -1
                    && lastLocation.z == -1)) {
            camera.unproject(delta.set(lastLocation.x,
                                                    lastLocation.y, 0));

            delta.sub(currentLocation);
            camera.position.add(delta.x, delta.y, 0);

            resolveWorldBounds();
        }
        lastLocation.set(x, y, 0);

        return true;
    }

    @Override
    public boolean touchUp (int x, int y, int pointer, int button) {
        lastLocation.set(-1, -1, -1);

        return true;
    }

    private void resolveWorldBounds() {
        float positionX = camera.viewportWidth * camera.zoom / 2;
        float positionY = camera.viewportHeight * camera.zoom / 2;

        if(camera.position.x  - positionX < 0)
            camera.position.x = positionX;
        else if(camera.position.x  + positionX > mapWidth)
            camera.position.x = mapWidth - positionX;

        if(camera.position.y  - positionY < 0)
            camera.position.y = positionY;
        else if(camera.position.y  + positionY > mapHeight)
            camera.position.y = mapHeight - positionY;

    }
```

```
    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchDown(int screenX, int screenY, int pointer, int button) {
        resolveWorldBounds();
        return false;
    }

    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        return false;
    }

    @Override
    public boolean scrolled(int amount) {
        return false;
    }
}
```

Konstruktori defineerimine

Kaamera liigutamiseks on vajalik, et meil oleks ligipääs rakenduses kasutatavale kaamerale ja maailma piiride tuvastamiseks on meil vaja teada plaatkaardi dimensioone. Defineerime muutujad ja määrame konstruktoris neile väärtused.

Meetod touchDragged()

Meetodis **touchDragged()** toimub kaamera liigutamine. Lohistuskauguse ja uue positsiooni määramiseks kasutamine Libgdx-i matemaatikateegis olevat **Vector3** klassi, millega on võimalik teha vektorite liitmist, lahutamist, korrutamist jms. Meetodi **unproject()** abil saame hiire või näpu koordinaadid, mis asuvad ekraaniruumis viia maailmaruumi, kus asub meie kaamera. Ilma **unproject()** meetodita ei oleks meil võimalik arvutada uut positsiooni ning sellepärast tuleb esiteks **touchDragged()** meetodis hetkepositsioon (**currentLocation**) ja viimane positsioon (**lastLocation**) viia maailmaruumi **unproject()** abil. Meetodit **set()** kasutame selleks, et muuta vektori koordinaate. Meetod nõuab kolme koordinaati, milleks on x, y ja z. Koordinaat z on alati 0 sellepärast, et meie maailm on kahemõõtmeline.

Ei ole mõtet arvutada uut positsiooni, kui hiirt ei ole lohistatud ja see on ka põhjus, miks positsiooni arvutamine asub tingimuslause sees. Kuna positsioon ei saa olla kunagi (-1; -1; -1), siis sobivad need koordinaadi hästi tingimusteks, et vältida mõttetut arvutamist. Meetodis **touchUp()** ehk igakord, kui

näpp tõstetakse ekraanilt või hiireklahv lastakse lahti määrame viimaseks positsiooni (**lastLocation**) koordinaatideks (-1; -1; -1).

Kaamera uue positsiooni arvutamiseks loome muutuja **delta**, mis on vektorsuurus ja tähistab vektorit viimasest positsioonist hetkepositsioonini. Matemaatikast on teada, et selleks kasutame vektorite lahutamist ja Libgdx-s on selleks meetod **sub()**. Uue kaamerapositsiooni määramiseks liidame saadud vektori kaamera koordinaatidega kasutades meetodit **add()**.

Meetod resolveWorldBounds()

Lõpetuseks on meil vaja luua meetod **resolveWorldBounds()**, mis tegeleb sellega, et me kaardi piiridest väljaspoole ei saaks minna. Esiteks on meil vaja teada suurust ekraani keskelt vasakule-paremale ja ülevalt-alla. Arvutamisel tuleb arvesse võtta kaamera suumi ja seega valemiks tuleb *vaateava * suum / 2.* Konstruktoris määrasime maailma laiuse (**mapWidth**) ja pikkuse (**mapHeight**), mille abil saame teada, kas on mindud välja paremalt või ülevalt. Vasakuks ja alumiseks piiriks on väärtus 0. Vasaku ja parema piiri saamiseks lahutame või liidame kaamera x-koordinaadist **positionX** muutuja. Ülemise ja alumise piiri saamiseks lahutame või liidame kaamera y-koordinaadist **positionY** muutuja.

Sisendprotsessori loomine mänguekraanis

Kaamera sisendprotsessori loome mänguekraani („**GameScreen.java**“) klassis. Kasutame sellejaoks sisendmultiplekser klassi **InputMultiplexer**, mis võimaldab meil kasutada mitud protsessorit. Sisendmultiplekser hoiab kasutajaliidese ja mängumaailma manipuleerimise eraldi klassides.

Esiteks defineerime sisendmultiplekseri.

```
    private InputMultiplexer multiplexer;
```

Järgmiseks loome multiplekseri ja kutsume konstruktoris meetodi, mis loob ja sätestab sisendprotsessorid.

```
        multiplexer = new InputMultiplexer();
        setInputProcessors();
```

Meetod on ise järgmine:

```
    private void setInputProcessors() {
        multiplexer.addProcessor(new CameraInputProcessor(camera,
                                                world.getMapDimension().x,
                                                world.getMapDimension().y)
                                        );
        Gdx.input.setInputProcessor(multiplexer);
```

```
    }
```

Sisendmultiplekser toetab nelja erinevat sisendprotsessorit. Uusi sisendprotsessoreid saab lisada **addProcessor()** meetodi abil. Rakenduse sisendsündmuste kuularit saab määrata meetodiga **setInputProcessor()**. Kuular kutsutakse välja iga kaader enne **render()** meetodit.

5.6 Kasutajaliidese loomine Scene2D abil

Rakendusele loome kasutajaliidese, mille abil on võimalik lisada objekte (kaitseehitisi) ja kaamerat suumida kasutades Scene2D-d. Scene2D kasutab tegutsejate (*actors*) hierarhiat. Scene2D abil on võimalik luua tervet rakendust, kuid tema üheks suureks probleemiks on see, et tegutsejad sisaldavad nii vaadet (*renderdamine*) kui ka mudelit (*suurus, positsioon*). See on ka põhjus, miks ei tasu tervet rakendust arendada kasutades ainult Scene2D-d. Kasutajaliidese ehitamisel see meid ei häiri ja seega on Scene2D suurepärane valik.

Scene2D sisaldab järgmiseid omadusi:

- Terve grupi pööramist ja suurendamist rakendatakse kõikidele alamtegutsejatele. Alamtegutsejad tegutsevad enda koordinaatsüsteemis ja peategutsejate muundused (pööramine, suurendamine, liigutamine) lisatakse alamtegutsejatele automaatselt.
- 2D joonistamine **SpriteBatch** klassi abil. Iga tegutseja joonistab ennast enda koordinaatsüsteemis, mis on pööramata ja suurendamata. Koordinaadid (0; 0) on vasak alumine nurk.
- Tabamuse avastus töötab ka suurendatud ja pööratud tegutsejatele. Iga tegutseja kasutab tabamuse tuvastamiseks enda koordinaatsüsteemi.
- Sisendsündmuste marsruutimine tegutsejale, mida vajutati. Sündmuste haldamise süsteem võimaldab peategutsejatel sündmust käsitleda enne alamtegutsejaid.
- Erinevaid tegevusi on võimalik aheldada ja võimalik on saavutada keerulisi efekte.

5.6.1 Kasutajaliidese loomine

Kasutajaliidese jaoks loome klassi („**UserInterface.java**“), mis implementeerib ka **InputProcessor** liidest, et meil oleks võimalik tulevikus erinevaid kaitseehitisi lisada.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.InputProcessor;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.InputListener;
```

```
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import com.badlogic.gdx.scenes.scene2d.ui.Label;
import com.badlogic.gdx.scenes.scene2d.ui.Skin;
import com.badlogic.gdx.scenes.scene2d.ui.Table;

public class UserInterface implements InputProcessor {

    private Stage ui = new Stage();
    private Table rightPanel = new Table();
    private Table leftPanel = new Table();
    private Skin skin = new Skin(Gdx.files.internal("data/uiskin.json"));

    private int uiWidth = Gdx.graphics.getWidth() / Settings.UI_WIDTH_SCALE;

    OrthographicCamera camera;

    World world;

    public UserInterface(OrthographicCamera camera, World world) {
        this.camera = camera;
        this.world = world;

        setLeftPanel();
        setRightPanel();
    }

    private void setRightPanel() {
        rightPanel.setSize(uiWidth, Gdx.graphics.getHeight());
        rightPanel.setPosition(Gdx.graphics.getWidth() - uiWidth , 0);
        rightPanel.setSkin(skin);
        rightPanel.top();
        rightPanel.setClip(true);

        ui.addActor(rightPanel);
    }

    private void setLeftPanel() {
        leftPanel.setSize(uiWidth, Gdx.graphics.getHeight());
        leftPanel.setPosition(0, 0);
        leftPanel.top();
        leftPanel.setClip(true);

        ui.addActor(leftPanel);

        Label zoomLabel = new Label("Zoom", skin);
        leftPanel.add(zoomLabel);
        leftPanel.row();

        Image zoomPlusButton = new Image(
                                new Texture(Gdx.files.internal("data/zoom_plus_icon.png")));

            zoomPlusButton.addListener(new InputListener() {
            @Override
            public boolean touchDown (InputEvent event, float x, float y, int pointer,
                                      int button) {
                if(camera.zoom > 0.5f)
                    camera.zoom -= 0.1f;
                return false;
            }
        });
        leftPanel.add(zoomPlusButton);
        leftPanel.row();

        Image zoomMinusButton = new Image(
                            new Texture(Gdx.files.internal("data/zoom_minus_icon.png"))
                                    );

        zoomMinusButton.addListener(new InputListener() {
            @Override
            public boolean touchDown (InputEvent event, float x,
                                              float y, int pointer,
                                              int button) {
                if(camera.zoom < 1f)
                    camera.zoom += 0.1f;
                return false;
```

```
            }
        });
        leftPanel.add(zoomMinusButton);
        leftPanel.row();
    }

    public Stage getUI() {
        return ui;
    }

    public void render(float deltaTime) {
        ui.act(deltaTime);
        ui.draw();

        if (Settings.DEBUG) {
            leftPanel.debug();
            rightPanel.debug();
        }
    }

    @Override
    public boolean touchDown(int screenX, int screenY, int pointer,
                                        int button) {
        return false;
    }

    @Override
    public boolean keyDown(int keycode) {
        return false;
    }

    @Override
    public boolean keyUp(int keycode) {
        return false;
    }

    @Override
    public boolean keyTyped(char character) {
        return false;
    }

    @Override
    public boolean touchUp(int screenX, int screenY, int pointer, int button) {
         return false;
    }

    @Override
    public boolean touchDragged(int screenX, int screenY, int pointer) {
          return false;
    }

    @Override
    public boolean mouseMoved(int screenX, int screenY) {
        return false;
    }

    @Override
    public boolean scrolled(int amount) {
        return false;
    }


}
```

Muutujate defineerimine

Klass **Stage** on 2D lava graaf (Scene2D peaklass), mis sisaldab kõiki tegutsejaid, mida hakkame looma. Objektid **rightPanel** ja **leftPanel** on vastavalt parem ja vasak paneel. Klassi **Skin** abil on võimalik laadida kasutajaliidese disain pildifailist. Muutuja **uiWidth** määrab ära, kui lai on üks paneel.

Klass **Skin** on konteiner, mis sisaldab fonte, värve, tekstuure jms. Meie kasutame Libgdx lähtekoodis olevaid tekstuure ja faile. Vajalikud on järgmised failid: iskin.png (tekstuurid), uiskin.atlas (tähistab erinevaid regioone tekstuuril), uiskin.json (defineerib disainielemendid), default.fnt (sisaldab tähemärkide fonti). Vaadata lisaks aadressilt https://github.com/libgdx/libgdx/wiki/Skin.

Konstruktori loomine

Kasutajaliideses vajame kaamerat, et nuppude abil suumida ja maailmakonteinerit, et tulevikus erinevaid objekte lisada. Meetodite **setLeftPanel()** ja **setRightPanel()** abil seadistame vasakut ja paremat paneeli.

Meetod setRightPanel()

Meetodis seadistame ekraani paremas ääres olevat paneeli. Paneel hakkab sisaldama nuppe, mille abil saab kaitseehitisi maailmaruumi lisada. Meetodi **setSize()** abil määrame paneeli laiuse ja kõrguse. Tabeli positiooni aknas määrame meetodiga **setPostion().** Kui on vajadus muuta paneeli disaini, näiteks tausta, siis on vajalik, et sätestatud oleks **Skin** meetodiga **setSkin()**. **Table** klass võimaldab muuta ka seal paiknevate tegutsejate joondust. Meie rakenduses joondame tegutsejad ülesse kasutades meetodit **top()**. Mõistlik oleks ka sätestada, et paneelis asuvad objektid ei läheks üle piiride ning seda saab teostada meetodida **setClip()**. Meetod **debug()** on selleks, et näidata tabeli piirjooni. Iga tegutseja on vaja lisada ka 2D lavale meetodiga **addActor()**.

Meetod setLeftPanel()

Meetod algab sarnaselt eelmisele ja ei vaja seletamist. Järgnevalt loome sildi, mis tähistab, mida on võimalik nupule vajutades saavutada.

Scene2D on võimalik nuppe luua kasutades **Button** või **Image** klassi. Rakenduses kasutame viimast, et näidata, kuidas tekstuure laadida.

Image klass on Scene2D samuti tegutseja. Talle on võimalik lisada sündmuste kuulareid ja teda on võimalik pöörata, liigutada, suurendada. **Image** loomiseks on vaja, et loodud oleks **Texture** objekt, mis on tavaline OpenGL ES tekstuur ehk pildifail, mis on laetud graafikakaardi mälusse. **Texture** objekt vajab **FileHandle** objekti, mida on võimalik luua kasutades **Gdx.files.internal()** meetodit. See

meetod otsib faili „Working Directory“ kaustast ehk meie rakenduses „android/assets“ kaustast. Nupule saab kuularit lisada meetodiga **addListener()**, mis vajab parameetriks **InputListener** klassi. Suumimisnupu määrame vaid ühe kuulari, milleks on **touchDown().** Kui nupp on valmis on vaja see **add()** meetodiga lisada paneelile. Pärast tegutseja lisamist on vaja minna uuele reale meetodiga **row()**. Sissesuumimiseks on vaja kaamera suumi vähendada. Järgmiseks loome nupu välja suumimiseks. Väljasuumiseks on vaja kaamera suumi suurendada.

Meetod getUI()

Sisendmultiplekseri jaoks on vaja luua tavaline **get()** meetod, et meie Scene2D objekt lisada kuularite ahelasse.

Meetod render()

Kuna Scene2D sisaldab nii mudelit kui ka vaadet tuleb luua **render()** meetod, mille abil saaks nuppe ekraanile joonistada.

Meetod **act()** kutsutakse välja igas tegutsejas, mis **Stage** objektis asetsevad. Meetod tagab selle, et me saaks objekte animeerida. Ekraanile joonistamiseks kutsume välja meetodi **draw()**.

InputProcessor liidese meetodid

Kuna meie klass implementeerib **InputProcessor** liidest, siis peame implementeerima ka meetodid. Hetkel me meetodeid ei muuda, aga tulevikus läheb neid meetodeid vaja.

5.6.2 Mänguekraanis vajalike muudatuste sisseviimine

Kasutajaliidese klassi valmides on vaja see ka meie rakendusse implementeerida. Esiteks on vaja defineerida muutuja.

```
    private UserInterface ui;
```

Mänguekraani konstruktor loome uue **UserInterface** objekti.

```
        ui = new UserInterface(camera, world);
```

Kindlasti vaadata, et see toimuks pärast kaamera ja maailmakonteineri loomist. Lõpuks on vaja sisendmultiplekserisse lisada uued protsessorid.

```
        multiplexer.addProcessor(ui.getUI());
        multiplexer.addProcessor(ui);
```

Lisamine peab toimuma enne kaamera sisendprotsessorit.

5.6.3 DesktopLauncher konfiguratsiooni muutmine

Lõpetuseks muudame ka meie Windowsis/Linuxis asuva akna suurust, et rakendust parem testida oleks. Klass asub „desktop“ kaustas.

```
        config.title = "just-another-tower-defense";
        config.width = Settings.WIDTH;
        config.height = Settings.HEIGHT;
```

Muudame akna pealkirja ja määrame akna suurusteks sätetest tulevad väärtused.

Rakendust käivitades peaks üleval vasakul asetsema nupud, millele on võimalik hiire või näpuga vajutada ja sellega kaamerat sisse- või välja suumida (Joonis 10).

Joonis 13 Kaamera suumimise nupud

5.7 Mänguobjektide liidese ja üldkasutatava abstraktse klassi loomine

Enne kui loome kaitseehitisi ja vaenlaseid on vaja luua liides, mille abil mänguobjektidega suhelda saaks. Loome ka abstraktse dünaamiliste objektide klassi, mis sisaldab meetodeid, mida vajavad kõik dünaamilised objektid maailmas. Klassi peaks laiendama kõik dünaamilised objektid meie mängumaailmas.

5.7.1 Mänguobjetkide liidese loomine

Esiteks tuleb meil luua liides, mille abil on võimalik mänguobjektiga suhelda. Kõik mänguobjektid olgu need siis vaenlased või kaitseehitised vajavad meetodeid, millega oleks võimalik neid joonistada, uuendada ja saada informatsiooni nende asukoha kohta. Mõistlik on seega luua ühtne liides („**GameObject.java**“), mille abil rakendus nendega suhtleb.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.math.Vector2;

public interface GameObject {

    public void update(float deltaTime);

    public void draw(Batch batch);

    public void setVelocity(Vector2 velocity);

    public Vector2 getPosition();

    public void setPosition(Vector2 position);
}
```

- **update()**: meetod, mida hakatakse välja kutsuma igakord enne ekraanile joonistamist. Hakkab sisaldama loogikat nagu sihtmärgi lukustamin, liikumine, pööramine.
- **draw():** joonistab objekti ekraanile.
- **setVelocity():** meetodi abil on võimalik muuta liikuva objekti kiirust.
- **getPosition()**: võimalik küsida objekti koordinaate maailmaruumis.
- **setPosition()**: võimaldab muuta objekti koordinaate maailmaruumis.

Hetkel tundub, et rohkem vaja ei lähe. Kui tulevikus peaks selgume, et mõni meetod on puudu on lihtne see lisada ja olla kindel, et kõik objektid selle ka implementeeriks.

5.7.2 Dünaamilise mänguobjekti loomine

Enne, kui alustame tornide klassi loomist on vaja luua abstraktne klass, mis oleks aluspind, millest luuakse kõik dünaamilised objektid maailmaruumis. Üldiselt vajavad kõik dünaamilised objektid samu meetodeid ja muutuste korral oleks väga tülikas neid kõikidel klassides eraldi muuta. Dünaamilise mänguobjekti puhul tuleb implementeerida loodud liides **GameObjet** ja ka **Comparable**, et objekte oleks võimalik võrrelda.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.MathUtils;
import com.badlogic.gdx.math.Vector2;


public abstract class DynamicGameObject implements GameObject, Comparable {

    protected String ID;

    protected Vector2 velocity = new Vector2();
    protected Vector2 position;
    protected float rotation = 0;

    protected TextureRegion textureRegion;

    public DynamicGameObject(Texture texture, String ID) {
        this(texture, ID, new Vector2());
    }

    public DynamicGameObject(Texture texture, String ID,
                                        Vector2 position) {
        this.textureRegion = new TextureRegion(texture);
        this.ID = ID;
        this.position = position;
        this.velocity = new Vector2();
    }

    public String getID() {
        return ID;
    }

    @Override
    public boolean equals(Object o) {
        if(!(o instanceof DynamicGameObject))
            return false;

        return ID.equals(((DynamicGameObject)o).getID());
    }

    @Override
    public int hashCode() {
        int hash = 1;
        hash = hash * 31 + ID.hashCode();
        return hash;
    }

    @Override
    public int compareTo(Object o) {
        return getID()
                    .hashCode() -
                    ((DynamicGameObject)o).getID().hashCode();
    }

    @Override
    public void update(float deltaTime) {
```

```
    }

    @Override
    public void draw(Batch batch) {
        if(rotation != 0)
            batch.draw(textureRegion, position.x, position.y,
                               0.5f, 0.5f, 1, 1, 1, 1, -rotation);
        else
            batch.draw(textureRegion, position.x, position.y,
                                0, 0, 1, 1, 1, 1, -rotation);
    }

    protected void calculateRotation(Vector2 directionNormal) {
        float angle = MathUtils.atan2(directionNormal.x,
                                                directionNormal.y);
        rotation = MathUtils.radiansToDegrees * angle;
    }

    @Override
    public Vector2 getPosition() {
        return new Vector2(position.x, position.y);
    }

    @Override
    public void setVelocity(Vector2 velocity) {
        this.velocity = velocity;
    }

    @Override
    public void setPosition(Vector2 position) {
        this.position = position;
    }
}
```

Muutujate defineerimine

Igat mänguobjekti peab olema võimalik tuvastada ja sellepärast peab igal objektil olema unikaalne id (**ID**). Objekt võib maailmaruumis ringi liikuda mingi kiirusega ja selleks on meil vaja vektorit (**velocity**). Igal objektil on paiknevus (**position**). Objekt võib olla pööratud (muutuja **rotation**). Joonistamiseks läheb meil vaja **TextureRegion** objekti.

Konstruktorid

Lähtekoodi vaadates on näha, et oleme defineerinud kaks konstruktorit. Põhjuseks on see, et mõne mänguobjekti puhul ei ole loomise hetkel teada, mis on tema positsioon ja ta peab seda mujalt küsima. Täpsemalt räägime sellest järgmistes peatükkides. Objektide puhul kasutame **TextureRegion** klassi, sest temaga on võimalik tekstuure ka manipuleerida.

Meetod getID()

Meetod on vajalik selleks, et küsida objekti unikaalset id-d.

Comparable liidese equals(), hashCode() ja compareTo() meetodid

Meetodid on vajalikud selleks, et võrrelda pinus, massiivis, puus või ükskõik millises konteineris olevaid objekte. Meetod **equals()** võrdleb kahe objekti id-sid. Juhul, kui tegu ei ole dünaamilise

mänguobjektiga ei ole objektid kindlasti võrdsed. Meetod **hashCode()** tekitab hashitud koodi ja koodi unikaalsuse tagab algarv 31. Meetod **compareTo()** võrdleb eelmises funktsioonis loodud koode. Tõene väärtus saadakse, kui lahutuse tulemuseks on 0. **Meeles tuleb pidada, et mõlemad meetodid *equals()* kui ka *compareTo()* peavad alati tagastama sama vastuse. Kui equals() tagastab, et kaks objekti on võrdsed, peab tagastama selle ka compareTo().**

Meetod update()

Meetod jääb tühjaks, sest erinevatel objektidel ei ole tegutsemisloogika sama.

Meetod draw()

Meetodis on vajalik tingimuslause selletõttu, et jättes alguspunkti koordinaatideks (0; 0) ja tekstuuri pöörates toimub see teljel, mis on risti tekstuuriga ja asub koordinaatidel (0; 0). Kui toimub pööramine on meil vaja pööramistelg nihutada tekstuuri keskele ja kuna tasand asub koordinaatidel 0st 1ni on meie pööramistelg koordinaatidel (0.5; 0;5).

Meetod calculateRotation()

Objekti keeramiseks kasutame suunavektorit (**directionNormal**)**.** Suunavektor näitab kuhu poole objekt vaatab. Lepime kokku, et nurk 0° näitab otse ülesse ja seega peaks objekti jaoks loodud pilt olema näoga ülespoole. Sel viisil on pööramist hea sooritada.

Paljudele võib olla tundmatu, mis meetod on **atan2()**. Nagu teame trigonomeetriast, jagatakse nurk neljaks veerandiks. Koolis õpitud funktsioonide **sin(), cos()** ja **tan()** abil ei ole meil võimalik teada**,** mis veerandis suunavektor (**directionNormal**) asub. Appi tuleb **atan2()** meetod, mis võtab parameetriteks kaks arvu. Kuna meetod tagastab nurga radiaanides, aga **Batch** klass vajab nurka kraadides on meil vaja nurk konverteerida kasutades valemit: *180f / PI * nurk.*

5.8 Kaitseehitiste loomine

Nüüd kui liides ja üldkasutatav klass on valmis on aeg niikaugel, et saame luua kaitseehitise. Peale kaitseehitise klassi loomise on meil vaja muuta ka kasutajaliidese klassi, kuhu lisada nupud, et oleks võimalik lohistades ehitisi maastikule lisada ja neid vajadusel ka eemaldada. Kasutajaliidese tarvis on vaja luua ka uus tegutseja klass, mida saaks kasutada indikaatorina, millist torni hakatakse ehitama ja kas torni on võimalik valitud plaadile ehitada või mitte. Meil on vaja luua ka loend maailmakonteineris, mis kõiki torne hoiaks ja meetodid, et ehitisi oleks võimalik eemaldada ja luua. Uuendada on vaja ka maailmarenderjat, et tornid ekraanile ja joonistataks.

5.8.1 Torni loomine

Hetkel tahame torne vaid ekraanile saada ja ei tegele vaenlaste tuvastamise ja torni pööramisega. Torni klass („**Tower.java**“) laiendab dünaamilise mänguobjekti klassi ja seega peale konstruktori ei ole hetkel vaja midagi muud lisada.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

public class Tower extends DynamicGameObject {

    public Tower(Texture texture, String ID, Vector2 position) {
        super(texture, ID, position);
    }

}
```

5.8.2 Tornide tehas

Hea oleks luua eraldi tehas, kasutades tehase meetodit, millega oleks võimalik luua kõiki olemasolevaid kaitseehitisi, ja mis hoiaks endas unikaalsete kaitseehitiste id-de loendurit. Loome ka konstantide klassi, mis hoiaks endas kõiki olemasolevaid torne. See meetod sobib meie väikese rakenduse jaoks, kuid suurema mängumootori jaoks on vaja kindlasti ehitada süsteem, kus olemasolevate objektide kohta saadakse infot XML, JSON või mõnest muust sobilikust failist.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

enum TowerType {
    TEST_TOWER
}

public class TowerFactory {

    private static int idCounter = 0;

    public static Tower makeTower(TowerType towerType, Vector2 position) {
        idCounter++;

        if(towerType == TowerType.TEST_TOWER)
            return makeTestTower(position);
        else
            return makeTestTower(position);
    }

    private static Tower makeTestTower(Vector2 position) {
        return new Tower( new Texture(Gdx.files.internal("data/tower.png")),
                          String.valueOf(idCounter),
                  position
                  );
    }
}
```

Konstantide klass TowerType

Klass hoiab kaitseehitiste konstante, mida on võimalik luua.

Muutujate defineerimine

Muutuja **idCounter** on loendur, mis loendab, mitu ehitist on loodud ja sama loenduri põhjal saab anda uutele ehitistele unikaalse id.

Meetod makeTower()

Ainuke avalik meetod, millega on võimalik tehasega suhelda ja uusi ehitisi luua. Andes meetodile ehitise tüübi ja positsiooni on võimalik vastav kaitseehitis luua.

Tingimuslause ei ole tegelikult vajalik, vaid on näitamaks, kuidas tuleks meetodit kasutada e. olenevalt saadud tüübist kutsub **makeTower()** välja vastava ehitise loomise meetodi ja tagastab ehitise objekti.

Meetod makeTestTower()

Loob soovitud torni ja tagastab selle.

5.8.3 Maailmakonteineris vajalike muudatuste sisseviimine

Enne nupude loomist peame looma ehitiste konteineri ja vajalikud meetodit, et ehitisi oleks võimalik lisada ja eemaldada. Peame looma ka meetodi, mille abil saaks teada, kas plaadile on võimalik ehitada või mitte. Vajalik on muuta maailmakonteinerit („**World.java**“) ja sinna kõik järgnev lisada.

Vajalik importimine

```
import java.util.concurrent.CopyOnWriteArrayList;
```

Defineerime muutujad

```
    private CopyOnWriteArrayList<Tower> towers = new CopyOnWriteArrayList<Tower>();
```

Libgdx tegutseb sündmuspõhiselt ja kõik meetodit ei toimu sünkroonselt, seega kui me kustutame loendist mõne objekti ja samal ajal soovitakse objektile ligipääsu näiteks **render()** meetodis saame *runtime error* tüüpi veateate. Selle probleemi lahendab **CopyOnWriteArrayList** loend, kus on võimalik kaitseehitisi hoida.

Meetod getTowers()

```
public CopyOnWriteArrayList<Tower> getTowers() {
    return towers;
}
```

Meetod tagastab kaitseehitiste loendi.

Meetod setConstructionLayerOpacity

```
public void setConstructionLayerOpacity(float opacity) {
    constructionLayer.setOpacity(opacity);
}
```

Uue torni lohistamisel oleks hea, kui mängija näeks visuaalselt ehitusalapiire. Selle meetodiga on võimalik ala läbipaistvust muuta.

Meetod isTileConstructable()

```
public boolean isTileConstructable(int x, int y) {
    if(x < 0 || x >= constructionArea.length || y < 0 || y >= constructionArea[0].length)
        return false;
    if(constructionArea[x][y]
           ==
         TileState.CONSTRUCTABLE.getState())
        return true;
    else
        return false;
}
```

Meetodis kontrollime, kas plaadile on võimalik ehitada või mitte. Esimene tingimuslause kontrollib, kas koordinaadid asuvad kaardist väljaspool, mille korral ei saa mitte mingil juhul nendele koordinaatidele ehitada. Teine tingimuslause kontrollib plaadi staatust: saab ehitada, ei saa ehitada.

Meetod createTower()

```
public void createTower(TowerType towerType, Vector2 position) {
    movementPath[(int)position.x][(int)position.y] =
                                              TileState.UNWALKABLE.getState();
    constructionArea[(int)position.x][(int)position.y] =
                                        TileState.UNCONSTRUCTABLE.getState();

    towers.add(TowerFactory.makeTower(towerType, position));

}
```

Loome vastava tornikonstanti alusel uue kaitseehitise. Lisaks sätestame liikumisala ja ehitusala massiivides, et plaadile ei ole võimalik ehitada ning ei ole võimalik plaadile liikuda.

Meetod removeTower()

```
public void removeTower(Vector2 mousePosition) {
    for (Tower tower : towers) {
        if (tower.position.x < mousePosition.x &&
             tower.position.x + 1 > mousePosition.x &&
```

```
                tower.position.y < mousePosition.y &&
                tower.position.y + 1 > mousePosition.y) {

                towers.remove(tower);

                movementPath[(int)mousePosition.x][(int)mousePosition.y] =
                                                    TileState.WALKABLE.getState();

                constructionArea[(int)mousePosition.x][(int)mousePosition.y] =
                                                TileState.CONSTRUCTABLE.getState();
            }
        }
    }
```

Meetodis kustutame torni, mis asub hiire või näpu paiknevuse koordinaatidel. Ala määramiseks kasutame ruutu. On teada, et üks ehitis võtab ruumi täpselt ühe ruudu. Juhul, kui plaadil asub torn, siis ehitis kustutatakse. Lisaks sätestame liikumisala ja ehitusala massiivides, et plaadil on võimalik ehitada ning plaadile on võimalik liikuda.

5.8.4 Tegutseja lisamine kasutajaliidesesse

Me tahame, et kasutajaliideses oleks indikaator, mis näitaks, kas plaadile on võimalik ehitada või mitte. Indikaator võiks vahetada ka värvi. Lisaks oleks hea anda informatsiooni selle kohta, milline torn on valitud. Selle jaoks loome eraldi klassi, mis laiendab tegutseja **Actor** klassi.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.Batch;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.scenes.scene2d.Actor;

public class TextureActor extends Actor {

    TextureRegion textureRegion = new TextureRegion();

    private float alpha = 1;

    private Color drawColor = new Color(1, 1, 1, alpha);

    public TextureActor() {
    }

    public void setTexture(TextureRegion texture) {
        this.textureRegion = texture;
    }

    public void setDrawColor(float r, float g, float b) {
        drawColor.r = r;
        drawColor.g = g;
        drawColor.b = b;
    }

    @Override
    public void draw(Batch batch, float alpha) {
        batch.setColor(drawColor.r, drawColor.g, drawColor.b, alpha);

        batch.draw(textureRegion, getX(), getY(), 0, 0,
                        textureRegion.getRegionWidth(),
                        textureRegion.getRegionHeight(), 1, 1, 0);
    }
}
```

Muutujate defineerimine

Indikaatori joonistamiseks on vajalik, et meil oleks tekstuur (**textureRegion**). Kuna soovime muuta tekstuuri värvust olenevalt sellest, kas plaat on ehituskõlblik või mitte on meil vaja määrata läbipaistvuse muutujat **alpha.** Värvuse hoidmiseks loome r, g, b värvide hoidmise objekti **drawColor** ja määrame algseks värviks valge.

Konstruktor

Konstruktoris me tekstuuri ei määra, sellepärast et tegutseja muudab oma tekstuuri olenevalt sellest, mis torn on valitud.

Meetod setTexture()

Meetodi abil vahetame tegutseja tekstuure.

Meetod setDrawColor()

Olenevalt plaadi staatusest muudame joonistamise värvi muutest r, g, b komponente.

Meetod draw()

Enne igat joonistamismeetodit määrame **setColor()** meetodi abil joonistamisvärvuse. Värv seguneb tekstuuri iga piksliga ja moodustab pikslile uue värvuse. Meetod **draw()** joonistame indikaatori ekraanile. **Actor** klassis on meetodit **getX()** ja **getY()**, mille abil saame küsida, kus see tegutseja ekraanil paikneb.

5.8.5 Kasutajaliideses vajalike muudatuste sisseviimine

Oleme niikaugel, et viia sisse muudatused kasutajaliideses („**UserInterface.java**“), pärast mida on võimalik torne lisada. Vaja on luua uued nupud paremasse paneeli ja neile kuularid lisada. Peale kuularite vajame meetodit, mis arvutaks koordinaadid maailmaruumis. Lisaks on vaja muuta **touchUp()** kuularit kasutajaliidese objektis endas, et hiireklahvi lahti lastes või näppu ekraanilt tõstes lisatakse ehitis kaardile. Meetodis **touchDragged()**

Vajalik importimine

```
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.math.Vector3;
```

Defineerime muutujad

```
    private TextureActor textureActor = new TextureActor();
```

Defineerime ja loome indikaatortegutseja.

```
    private boolean towerConstructable = false;
    private boolean towerRemovable = false;
    private TowerType constructableTowerType;
```

Nupule vajutades muudame staatust, kas valmistume kaitseehitise rajamiseks või kustutamiseks. Loome kaks *boolean* tüüpi muutujat. Vajame informatsiooni ka selle kohta, mis tüüpi torni hakkame ehitama ja loome **TowerType** väärtust hoidva muutuja.

Meetodi setRightPanel() muudatused

```
        setTowers();
```

Kutsume välja meetodi, mis lisab paremasse paneeli nupud. Lisada pärast paneeli seadistusi.

Meetod setTowers()

```
    private void setTowers() {
        Label towersLabel = new Label("Towers", skin);
        rightPanel.add(towersLabel);
        rightPanel.row();

        Image unknownTowerButton = new Image(new TextureRegion(
                            new Texture(Gdx.files.internal("data/tower_icon.png"))));
        unknownTowerButton.addListener(new InputListener() {

            @Override
            public boolean touchDown (InputEvent event, float x,
                                                float y, int pointer,
                                                   int button) {

                if (!towerConstructable) {
                    towerConstructable = true;
                    world.setConstructionLayerOpacity(0.2f);
                    constructableTowerType = TowerType.TEST_TOWER;

                    textureActor.setTexture(new TextureRegion(
                                        new Texture(Gdx.files.internal("data/tower_icon.png"))
                                             );

                    ui.addActor(textureActor);
                }

                return false;
            }
        });
        rightPanel.add(unknownTowerButton);
        rightPanel.row();

        Label utilityLabel = new Label("Tools", skin);
        rightPanel.add(utilityLabel);
        rightPanel.row();

        Image towerRemoveButton = new Image(new TextureRegion(
                    new Texture(Gdx.files.internal("data/tower_remove_icon.png"))));
```

```
        towerRemoveButton.addListener(new InputListener() {
            @Override
            public boolean touchDown (InputEvent event, float x, float y, int pointer,
                                                    int button) {
                if (!towerRemovable) {
                    towerRemovable = true;

                            textureActor.setTexture(new TextureRegion(
                                 new Texture(
                                      Gdx.files.internal("data/tower_remove_icon.png"))));

                    ui.addActor(textureActor);
                }
                return false;
            }
        });
        rightPanel.add(towerRemoveButton);
        rightPanel.row();
    }
```

Esiteks loome meetodis sildi, mis tähistaks, mida nupud teevad. Meetodiga **add()** saab paneelile objekte lisada. Meetod **row()** läheb uuele reale. Nupud loome täpselt samamoodi nagu tegime seda suumimisnuppude puhul.

Lisamisnupu kuular **touchDown()** erineb küllaltki suumimisnuppudest. Esiteks paneme käsud tingimuslause sisse, sest me ei taha, et kasutaja ekraanil lohistades aina uuesti nuppu aktiveeriks, vaid et lohistamise ajal see ei toimiks. Järgmiseks määrame staatuseks (**towerConstructable**), et oleme valmis ehitust lisama ja **setConstructionLayerOpacity()** meetodi abil näitame kasutajale ehitusala. Seda nuppu kasutame *TEST_TOWER* torni loomiseks. Järgmiseks sätestame eespool loodud indikaatortegutseja tekstuuri, milleks on *tower_icon.png* pilt. Viimaseks lisame tegutseja Scene2D lavale.

Kustutamisnupu kuularis **touchDown()** määrame staatuseks, et oleme valmis kustutamiseks. Tekstuuriks kasutame *tower_remove_icon.png* faili. Lõpetuseks lisame tegutseja Scene2D lavale.

Meetod getWorldSpaceCoordinates()

```
    private Vector2 getWorldSpaceCoordinates(int screenX, int screenY) {
        Vector3 unprojectedVector = new Vector3(screenX, screenY, 0);
        camera.unproject(unprojectedVector);

        return new Vector2(unprojectedVector.x, unprojectedVector.y);
    }
```

Meetodis tagastame hiire või näpu koordinaadi maailmaruumis, et teada millisele plaadile tahame ehitada või milliselt kustutada. Ekraanilt saame muidugi kaks koordinaati, kuid OpenGL-s on neid kolm. Selle jaoks loome **Vector3** tüüpi objekti. Maailmakoordinaatide saamiseks kasutame juba

kasutaud **unproject()** meetodit, mille abil saame maailmaruumi koordinaadid. Meetodis tagastame aga **Vector2** tüüpi muutuja, sest meil ei ole kolmandaga midagi peale hakata.

Meetod touchUp()

```
@Override
public boolean touchUp(int screenX, int screenY, int pointer, int button) {
    if (towerConstructable) {
        Vector2 position = getWorldSpaceCoordinates(screenX, screenY);

        if(world.isTileConstructable((int)position.x, (int)position.y))
            world.createTower(TowerType.TEST_TOWER, new Vector2((int)position.x,
                                                                (int)position.y));

        ui.getActors().removeValue(textureActor, true);
        world.setConstructionLayerOpacity(0);
        towerConstructable = false;

        return true;
    }
    else if (towerRemovable) {
        ui.getActors().removeValue(textureActor, true);
        Vector2 mousePosition = getWorldSpaceCoordinates(screenX, screenY);
        world.removeTower(mousePosition);
        towerRemovable = false;
    }
    return false;
}
```

Kui kasutaja tõstab näpu ekraanilt või laseb lahti hiireklahvi, kontrollime, kas on määratud kumbki eespool määratud staatustest: valmis ehitamiseks või valmis eemaldamiseks.

Kui oleme valmis ehitamiseks küsime **getWorldSpaceCoordinates()** meetodiga maailmaruumis olevad koordinaadid. Meetodiga **isTileConstructable()** küsime, kas plaadile on võimalik ehitada ja juhul, kui on, lisame torni plaadile. Pärast lisamist on vaja **getActors().removeValue()** meetodiga indikaator lavalt ka eemaldada ja **setConstructionLayerOpacity** abil ehitusala kiht jälle läbipaistvaks muuta.

Kui oleme valmis ehitist eemaldama, kustutame torni maailmakonteineris olevast loendist meetodi **removeTower()** abil.

Meetod touchDragged()

```
@Override
public boolean touchDragged(int screenX, int screenY, int pointer) {
    if (towerConstructable || towerRemovable) {
        textureActor.setX(screenX);
        textureActor.setY(Math.abs(screenY - Gdx.graphics.getHeight()));

        Vector2 position = getWorldSpaceCoordinates(screenX, screenY);
        if(world.isTileConstructable((int)position.x, (int)position.y))
            textureActor.setDrawColor(0, 0.8f, 0.6f);
        else
            textureActor.setDrawColor(1f, 0f, 0f);


        return true;
    }
```

```
        return false;
    }
```

Meetodis liigutame indikaatorit. Kindlasti tuleb käsud panna tingimuslause sisse. Indikaatori koordinaatideks määrame hiire või näpu koordinaadi ekraanil. Valemiga *Math.abs(y - ekraanilaius)* pöörame ringi y-telje, sest ekraan kasutab teistpidist koordinaatsüsteemi. Meetodiga **getWorldSpaceCoordinates()** saame maailmaruumi koordinaadid. Tingimuslausega saame määrata, mis värvi indikaator värvitakse, olenevalt plaadi staatusest.

5.8.6 Maailmarenderdajas vajalike muudatuste sisseviimine

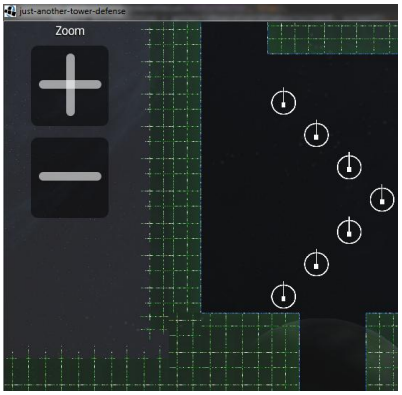
Lõpetuseks on vaja maailmarenderdajas („**WorldRenderer.java**“) loodud tornid maastikule joonistada. Seda teeme me loomulikult **render()** meetodis.

```
        Batch tiledMapBatch = renderer.getBatch();
        tiledMapBatch.begin();
        for (Tower tower : world.getTowers()) {
            tower.update(deltaTime);
            tower.draw(tiledMapBatch);
        }
        tiledMapBatch.end();
```

Enne joonistamist on vaja saada **OrthogonalTiledMapRenderer** objektist joonistaja **tiledMapBatch** meetodi **getSpriteBatch()** abil. Tegevus on vajalik selleks, et tornid joonistatakse maailmaruumi koordinaate arvesse võttes, mitte ekraani koordinaatidele. Nagu ikka tuleb joonistusmeetodid kutsuda pärast **begin()** meetodit ja enne **end()** meetodit. For-tsükliga joonistame kõik tornid maailma, kasutades objetki liideses defineeritud **draw()** meetodit.

5.8.7 Rakenduse käivitamine

Rakendust käivitades saab hiirt või näppu lohistades torne ehitusalale lisada ja neid ka kustutada (Joonis 11). Tornid hetkel midagi ei tee, vaid lihtsalt staatiliselt seisavad.



Joonis 14 Kaitseehitiste lisamine

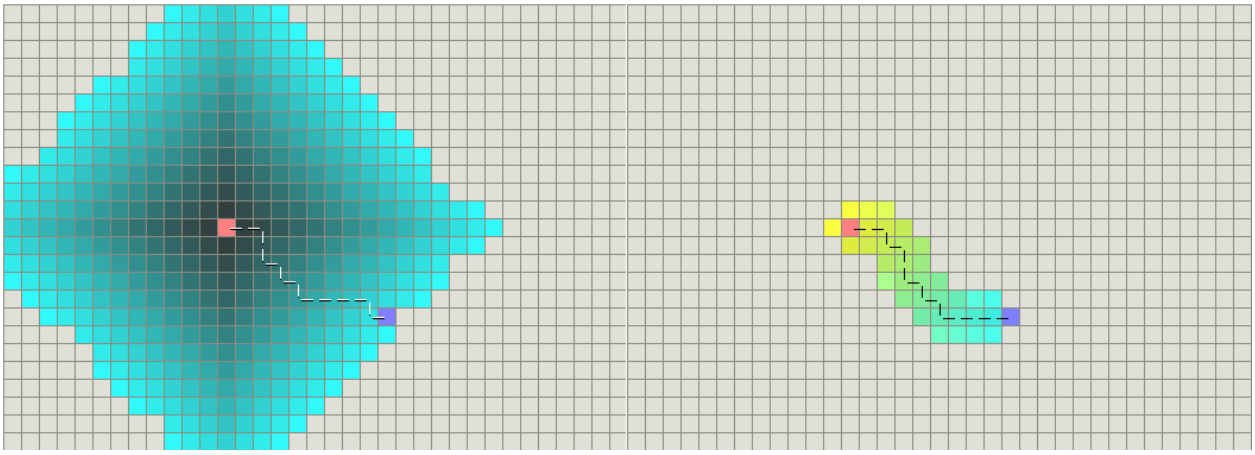
5.9 Vaenlase loomine ja rajaleidja abil liikumine

Selles peatükis loome vaenlase, mis liiguks üle kaardi kasutades rajaleidjat. Seega on vaja luua ka rajaleidja klass, mis otsiks igale kaardile ilmunud vaenlasele liikumisteekonna. Vaenlasele eriomadusi ei hakka lisama ja ainukesteks omadusteks on võime liikuda algpunktist lõpp-punkti ning elud, mille lõppedes vaenlane mängust eemaldatakse.

5.9.1 Rajaleidja loomine

Rajaleidja loomisel on väga tähtis, millist algoritmi kasutada. Meie rakendus on lihtne ja ei toimu vaenlaste omavahelisi kokkupõrkeid. Vajalik oleks, et vaenlane liiguks mööda eeldefineeritud rada ja vajadusel saaks teekonda kohendada olenevalt sellest, kas lühemal teekonnal asub takistus või mitte.

Probleemi lahendamiseks sobib hästi A* algoritm, mis on Dijkstra algoritmi edasiarendus. A* algoritm kasutab erinevalt Dijkstra-st parim enne otsingut (vt joonist 12). A* otsib lühemat teekonda punktis a punkti b läbides graafi tippe.



Joonis 15 Dijkstra algoritm (vasakul) **ja A* algoritm** (paremal)**, (Amit Patel, 2014)**

Teekonna arvutamiseks kasutab valemit **f = g + h**, kus **g** on teekonna maksumus siiamaani ja **h** on hinnang palju läheb maksma sellest tipust sihtpunkti jõudmine. Selle valemiga on võimalik valida parim lähim naaber ja efektiivsemalt rada otsida.

Õppematerjalis ei hakka detailideni laskuma, kuid huvi korral leiab veebiaadressilt http://theory.stanford.edu/~amitp/GameProgramming/index.html väga põhjaliku seletuse, kuidas algoritm töötab.

Selgusele jõutud, et A* on just see, mida me vajame. Loome rajaleidja („**AstarPathFinder.java**“), mis seda algoritmi kasutaks. Meie rakenduses töötavad plaadid graafi tippudena, seega on implementeerimine päris mugav.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.math.Vector2;

import java.util.*;

public class AStarPathFinder {

    private final float MOVEMENT_COST = 1f;
    private final float MOVEMENT_COST_DIAGONAL = 1.4f;

    private int nodeIdCounter = 0;

    private class Node implements Comparable {
        public int ID;
        public int x, y;
        public float f, g, h;
        public Node parent;

        public Node(int x, int y) {
            this(x, y, null);
        }

        public Node(int x, int y, Node parent) {
            nodeIdCounter++;
            this.ID = nodeIdCounter;
            this.x = x;
            this.y = y;
            f = g = h = 0;
            this.parent = parent;
        }

        @Override
        public int compareTo(Object o) {
            Node n = (Node)o;
            return (int)(f - n.f);
        }

        @Override
        public boolean equals(Object o) {
            Node n = (Node)o;

            if(n.x == x &&  n.y == y)
                return true;
            else
                return false;
        }

        @Override
        public int hashCode() {
            int hash = 23;
            hash = hash * 31 + Integer.toString(x).hashCode();
            hash = hash * 31 + Integer.toString(y).hashCode();
            return hash;
        }
    }

    private Node startNode;
    private Node finalNode;

    private int[][] map;

    private PriorityQueue<Node> openList;
    private HashMap<Node, Float> openListF;

    private PriorityQueue<Node> closeList;
    private HashMap<Node, Float> closeListF;

    public AStarPathFinder(int[][] map) {
```

```
        this.map = map;
    }

    public List<Vector2> getPathList(int startX, int startY, int finalX, int finalY) {
        openList = new PriorityQueue<Node>();
        openListF = new HashMap<Node, Float>();

        closeList = new PriorityQueue<Node>();
        closeListF = new HashMap<Node, Float>();

        List<Vector2> pathList = new ArrayList<Vector2>();

        startNode = new Node(startX, startY);
        finalNode = new Node(finalX, finalY);
        calculateNodeCost(startNode, startNode);
        openList.offer(startNode);
        openListF.put(startNode, startNode.f);

        while (!openList.isEmpty()) {
            Node currentNode = openList.poll();

            List<Node> successors = generateSuccessors(currentNode);
            Iterator<Node> it = successors.iterator();
            while(it.hasNext()) {
                Node successor = it.next();
                calculateNodeCost(successor, currentNode);

               if(successor.x == finalNode.x && successor.y == finalNode.y) {
                    for (Node n = successor; n.parent != null; n = n.parent)
                        pathList.add(new Vector2(n.x, n.y));

                    return pathList;
                }

                if (!(openList.contains(successor) &&
                      openListF.get(successor) <= successor.f) &&
                    !(closeList.contains(successor) && closeListF.get(successor)
                      <= successor.f)
                   ) {
                    openList.offer(successor);
                    openListF.put(successor, successor.f);
                }
            }

            closeList.offer(currentNode);
            closeListF.put(currentNode, currentNode.f);
        }
        return pathList;
    }

    public List<Node> generateSuccessors(Node parent) {
        List<Node> successors = new LinkedList<Node>();

        if (parent.x - 1 >= 0 && parent.y - 1 >= 0) {
            if (map[parent.x - 1][parent.y - 1] == TileState.WALKABLE.getState()
                && map[parent.x][parent.y - 1] == TileState.WALKABLE.getState()
                && map[parent.x - 1][parent.y] == TileState.WALKABLE.getState())

                successors.add(new Node(parent.x - 1, parent.y - 1, parent));
        }
        if (parent.x - 1 >= 0) {
            if(map[parent.x - 1][parent.y] == TileState.WALKABLE.getState())
                successors.add(new Node(parent.x - 1, parent.y, parent));
        }
        if (parent.x - 1 >= 0 && parent.y + 1 < map[0].length) {
            if(map[parent.x - 1][parent.y + 1] == TileState.WALKABLE.getState() &&
               map[parent.x - 1][parent.y] == TileState.WALKABLE.getState() &&
               map[parent.x][parent.y + 1] == TileState.WALKABLE.getState())

                successors.add(new Node(parent.x - 1, parent.y + 1, parent));
        }
        if (parent.y - 1 >= 0) {
            if(map[parent.x][parent.y - 1] == TileState.WALKABLE.getState())
                successors.add(new Node(parent.x, parent.y - 1, parent));
        }
        if (parent.y + 1 < map[0].length) {
```

```
            if(map[parent.x][parent.y + 1] == TileState.WALKABLE.getState())
                successors.add(new Node(parent.x, parent.y + 1, parent));
        }
        if (parent.x + 1 < map.length && parent.y + 1 < map[0].length) {
            if(map[parent.x + 1][parent.y + 1] == TileState.WALKABLE.getState() &&
               map[parent.x + 1][parent.y] == TileState.WALKABLE.getState() &&
               map[parent.x][parent.y + 1] == TileState.WALKABLE.getState())

                successors.add(new Node(parent.x + 1, parent.y + 1, parent));
        }
        if (parent.x + 1 < map.length) {
            if(map[parent.x + 1][parent.y] == TileState.WALKABLE.getState())
                successors.add(new Node(parent.x + 1, parent.y, parent));
        }
        if (parent.x + 1 < map.length && parent.y - 1 >= 0) {
            if(map[parent.x + 1][parent.y - 1] == TileState.WALKABLE.getState()
               && map[parent.x][parent.y - 1] == TileState.WALKABLE.getState()
               && map[parent.x + 1][parent.y] == TileState.WALKABLE.getState())

                successors.add(new Node(parent.x + 1, parent.y - 1, parent));
        }

        return successors;
    }

    public void setUnwalkable(int x, int y) {
        map[x][y] = TileState.UNWALKABLE.getState();
    }

    public void setWalkable(int x, int y) {
        map[x][y] = TileState.WALKABLE.getState();
    }

    private void calculateNodeCost(Node currentNode, Node parentNode) {
        currentNode.g = g(parentNode, currentNode);
        currentNode.h = h(currentNode, finalNode);
        currentNode.f = f(currentNode.g, currentNode.h);
    }

    private float f(float g, float h) {
        return g + h;
    }

    private float h(Node from, Node to) {
        float dx = Math.abs(from.x - to.x);
        float dy = Math.abs(from.y - to.y);

        return MOVEMENT_COST * (dx + dy) + (MOVEMENT_COST_DIAGONAL - 2
                                    * MOVEMENT_COST) * Math.min(dx, dy);
    }

    private float g(Node from, Node to) {
        if(from.x == to.x && from.y == to.y)
            return 0.0f;
        else if((from.x == to.x && (from.y - 1 == to.y || from.y + 1 == to.y))
                    ||
                (from.y == to.y && (from.x - 1 == to.x || from.y + 1 == to.y))
               )
            return from.g + MOVEMENT_COST;
        else
            return from.g + MOVEMENT_COST_DIAGONAL;
    }
}
```

Muutujate defineerimine

Rakenduses töötavad plaadid graafi tippudena. Igalt tipult on võimalik liikuda horisontaalselt, vertikaalselt ja diagonaalselt. Vertikaalselt ja horisontaalselt liikudes on maksumuseks **MOVEMENT_COST**, diagonaalselt **MOVEMENT_COST_DIAGONAL**. Vaja on teada ka millisest tipust alustame ja millisele peame jõudma - loome **startNode**-i ja **finalNode**-i. Liikumisala massiivi

salvestame muutujasse **map**, mis tornide ehitamise käigus muutub. Avatud tippude hoidmiseks, mida ei ole veel läbitud loome prioriteetpinu **openList**, kus eelisõigus on tipul, mille maksumus on väikseim. Loome ka paisktabeli **openListF**, mille abil oleks võimalik kiiresti leida maksumus sellele tipule. Samamoodi tuleb luua ka **closeList** ja **closeListF** hoidmaks juba läbitud tippe.

Meetod generateSuccessors()

Igal tipul on kaheksa naabrit. Meetodis kontrollime, kas naabritele on võimalik liikuda või mitte. Kõik naabrid, millele on võimalik liikuda lisame loendisse, mille lõpuks tagastame.

Alamklass Node

Iga **Node** objekt on üks graafi tipp (kaardil üks plaat). Objekti salvestame tema koordinaadid ja erinevad maksumused. Objekt **Node** peab implementeerima ka **Comparable** liidest, mis võimaldab meil prioriteetpinus olenevalt maksumusest tipp järjekorda seada. Objektil on ka vanemtipp, mille järgi on võimalik tagasi minna alguspunkti ja selle abil luua rada.

Meetod h()

Meetod on heuristiline funktsioon. Kauguse arvutamiseks kasutame Chebyshev kaugust, sest me võime liikuda ka diagonaalselt ja annab meile sobiva hinnangu, palju maksab tipust jõudmine sihtkohta.

Meetod g()

Arvutame maksumuse, mis kulub selleks, et naabertipule jõuda. Juhul, kui tahame minna tippu, kus oleme, siis maksumus puudub. Diagonaalse liikumise puhul peame maksma rohkem, kui horisontaalse tipu puhul.

Meetod f()

Meetodis arvutame kogumaksumuse, millega on võimalik määrata tipu prioriteeti.

Meetod getPathList()

Meetodis arvutame vaenlase teekonna. Esiteks on vaja algväärtustada loendid, mis hakkavad hoidma tippe. Vaja on luua ka **pathList** loend, mis hakkab hoidma kontrollpunkte, mille abil on võimalik maailmas orienteeruda. Loome ka alguspunkti (**startNode**), milleks on vaenlase asukoht ja sihtpunkti (**finalNode**), kuhu vaja liikuda. Me peame arvutama ka tipu maksumuse sihtpunkti **calculateNodeCost()** abil. Iga uus tipp on vaja lisada ka avatud prioriteetpinusse **openList** meetodi

offer() abil, mis vaatab tipu **f** maksumust ja lisab vastavalt järjekorda. Lisaks lisame **f** maksumuse paisktabelisse.

While-tsüklis toimub graafi läbimine ja kestab niikaua, kuni oleme jõudnud sihtpunkti või kui kõik tipud on läbitud. Esiteks on vaja **poll()** abil järjekorras esimene tipp loendist välja võta. Järgmiseks saame kõik naabrid, millele on võimalik liikuda meetodiga **generateSuccessor().** Järgmiseks peame **while**-tsükliga läbima kõik genereeritud naabrit.

Esimene tingimuslause tsüklis kontrollib, kas oleme jõudnud sihtpunkti. Juhul, kui oleme sihtpunktis salvestame iga tipu loendisse, läbides graafi tagurpidi algusesse.

Teises tingimuslause läbimisel lisatakse naaber avatud tippude loendisse. Lisamine toimub ainult siis, kui on täidetud järgmised tingimused:

- tipp ei ole avatud tippude loendis või avatud tippude loendis oleva sama tipu maksumus on suurem
- tipp ei ole läbitud tippude loendis või läbitud tippude loendis oleva tipu maksumus on suurem.

Viimaseks tuleb lisada tipp läbitud tippude loendisse.

5.9.2 Vaenlase loomine

Vaenlane peab olema võimeline leidma parima tee läbi kaardi ja kontrollpunkte kasutades liikuma läbi kaardi sihtpunkti. Vaenlast („**Enemy.java**“) peab laiendama **DynamicGameObject** klassiga ja ta peab teadma ka rajaleidja olemasolust.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

import java.util.Stack;

public class Enemy extends DynamicGameObject {

    private float speed = 2f;
    private float health = 50f;

    private AStarPathFinder pathFinder;


    private Vector2 endPosition = new Vector2();
    private Vector2 wayPoint = new Vector2();
    private Stack<Vector2> wayPoints = new Stack<Vector2>();

    public Enemy(Texture texture, AStarPathFinder pathFinder,
                 String ID, Vector2 startPosition, Vector2 endPosition) {

        super(texture, ID);
        this.pathFinder = pathFinder;
        this.position.set(startPosition);
```

```
        this.endPosition.set(endPosition);

        getMovementPath();
    }

    public void getMovementPath() {
        if(!wayPoints.empty())
            wayPoints.clear();

        velocity.set(0, 0);
        wayPoints.addAll(pathFinder.getPathList((int) position.x,
                                                (int) position.y,
                                                (int) endPosition.x,
                                                 (int) endPosition.y));

        if(wayPoints.empty())
           wayPoints.add(endPosition);
    }

    @Override
    public void update(float deltaTime) {
        calculateVelocity();
        this.position.add(velocity.x * deltaTime, velocity.y * deltaTime);
    }

    private void calculateVelocity() {

        if(!wayPoints.empty() && (velocity.isZero()
                || position.epsilonEquals(wayPoint, 0.3f))) {
            wayPoint = wayPoints.pop();
            Vector2 normal = wayPoint.cpy().sub(position).nor();
            velocity.set(normal.x * speed, normal.y * speed);

            calculateRotation(normal);

        }
        else if(position.epsilonEquals(endPosition, 0.2f))
            velocity.set(0, 0);
    }
}
```

Muutujate defineerimine

Vaenlasel peab olema mingi kiirus, millega ta kaarti läbib, selleks loome muutuja **speed.** Järgmises peatükis on võimalik vastaseid ka hävitada ja seega võiks meil olla ka elud (**health**). Rajaleidmiseks vajame abiklassi ja defineerime **pathFinder**-I, mille saame konstruktoris kaasa. Vaenlane peaks teadma ka sihtpunkti, kuhu ta tahab jõuda (muutuja **endPosition**). Kaardi liikumiseks peame teadma kontrollpunkti, kuhu tahame jõuda (muutuja **wayPoint**) ja kõiki kontrollpunkte, mida läbime (muutuja **wayPoints**).

Konstruktor

Vastase loomisel määrame kasutatava tekstuuri ja unikaalse id. Lisaks anname kaasa rajaleidja, mis luuakse maailmakonteineris. Plaatkaardilt saame algpositsiooni ja sihtpunkti, mis tuleks sätestada. Konstruktoris võiks küsida ka liikumisrada, mille jaoks loome meetodi **getMovementPath().**

***Meetod** getMovementPath()*

Esimeses tingimuslauses puhastame kontrollpunktide pinu **clear()** abil, juhul kui pinu ei ole tühi. See olukord võib juhtuda, kui rakenduse käigus selgub, et peame teekonda uuesti küsima. Järgmisena meetodiga **set()** peatame vastase. Võib juhtuda, et arvutades uut teekonda liigub objekt liikumisalast välja. Meetodi **addAll()** abil lisame rajaleidja abil arvutatud raja kontrollpunktide loendisse. Teises tingimuslauses määrame järgmiseks kontrollpunktiks sihtpunkti, juhul kui ühtegi võimaliku rada ei leitud. Õigem oleks muidugi liikuda üle tornide või mitte lasta torni ehitada kohta, mis paneks tee kinni, kuid see jäägu lisaülesannete alla.

Meetod calculateVelocity()

Meetodis arvutame kiirusvektori, et vastast mingis suunas liigutada. Uue kiirusvektori arvutamine toimub ainult siis, kui kontrollpunktide pinu ei ole tühi ja kontrollpunktini, kuhu me tahame jõuda on lähemal kui 0.3 ühikut. Meetodiga **epsilonEquals()** on võimalik määrata tolerantsust, mida võib vastavalt vajadusele häälestada. Meetodiga **pop()** on võimalik pinust kontrollpunkt kätte saada. Suunavektori (muutuja **normal**) arvutame nagu ikka vektorite lahutamise teel. Tuleb meeles pidada, et vektor tuleb normaliseerida meetodiga **nor(),** et oleks võimalik suunavektori kiirust määrata ja tekstuuri pöörata. Meetodiga **calculateRotation()** pöörame tekstuuri vastavalt sinna suunda, kuhu vastane vaatab. Teises tingimuses jätame objekti seisma, kui ta on jõudnud sihtpunkti.

5.9.3 Vaenlaste tehase loomine

Vaenlaste loomiseks kasutame samuti tehasemeetodit. See tehas on praktiliselt identne tornide tehasele ja ei vaja seletust. *Else* tingimus tagastab küll sama objekti, aga on vajalik, et kood kompileeruks, sest muidu võib juhtuda, et meetod ei tagasta mitte midagi.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

enum EnemyType {
    TEST_ENEMY,
}

public class EnemyFactory {

    private static int idCounter = 0;

    public static Enemy makeEnemy(AStarPathFinder pathFinder, EnemyType enemyType,
                                  Vector2 startPosition, Vector2 endPosition) {
        idCounter++;

        if(enemyType == EnemyType.TEST_ENEMY)
            return makeTestEnemy(pathFinder, startPosition, endPosition);
        else
            return makeTestEnemy(pathFinder, startPosition, endPosition);
    }

    private static Enemy makeTestEnemy(AStarPathFinder pathFinder,
                                       Vector2 startPosition, Vector2 endPosition) {
```

```
        return new Enemy( new Texture(Gdx.files.internal("data/enemy.png")),
                          pathFinder,
                          String.valueOf(idCounter),
                          startPosition, endPosition);
    }
}
```

5.9.4 Maailmakonteineris vajalike muudatuste sisseviimine

Enne kui saame vaenlasi maailmasse tuua, peame maailmakonteineris (“**World.java**”) initsialiseerima vaenlaste loendi ja implementeerima vajalikud meetodit, mille abil oleks võimalik vaenlasi luua ja kustutada. Nagu ka tornideloendi puhul loome *CopyOnWriteArrayList*-I, mida on turvaline kasutada, kui rakenduses kasutatakse mitut lõime.

Defineerime muutujad

```
    private CopyOnWriteArrayList<Enemy> enemies = new CopyOnWriteArrayList<Enemy>()
```

Selles loendis hakkavad asetsema kõik maailmas asetsevad vaenlased.

```
    private AStarPathFinder pathFinder;
```

Rajaleidja vaenlaste jaoks.

Konstruktor

```
        pathFinder = new AStarPathFinder(movementPath);
```

Initsialiseerima rajaleidja, mida hakkavad kasutama kõik vaenlased, kes maailmasse luuakse.

Meetod getEnemies()

```
    public CopyOnWriteArrayList<Enemy> getEnemies() {
        return enemies;
    }
```

Meetodid vajame maailmarenderdajas, kus toimub objektide joonistamine ekraanile.

Meetod removeEnemy()

```
    public void removeEnemy(Enemy enemy) {
        enemies.remove(enemy);
    }
```

Meetodi abil kustutame vaenlase jäädavalt maailmast.

Meetod createEnemy()

```
    public void createTower(TowerType towerType, Vector2 position) {
```

```
        movementPath[(int)position.x][(int)position.y] = TileState.UNWALKABLE.getState();

        constructionArea[(int)position.x][(int)position.y] =
                                                TileState.UNCONSTRUCTABLE.getState();

        towers.add(TowerFactory.makeTower(towerType, position));

    }
```

Kutsume välja, kui on vaja luua uus vaenlane. Alg- ja lõppkoordinaadid saame plaatkaardilt. Loomulikult on rajaleidjaks eespool initsialiseeritud rajaleidja.

5.9.5 Maailmarenderdajas vajalike muudatuste sisseviimine

Maailmarenderdajas (“**WorldRenderer.java**”) on vaja luua lihtne *for*-tsükkel, mille abil kõik vaenlased ekraanile joonistada.

Meetod render()

Nüüd näeb meetod välja järgmine:

```
    public void render(float deltaTime) {
        camera.update();
        renderer.setView(camera);
        renderer.render();
        Batch tiledMapBatch = renderer.getBatch();

        tiledMapBatch.begin();
        for (Enemy enemy : world.getEnemies()) {
            enemy.update(deltaTime);
            enemy.draw(tiledMapBatch);
        }
        for (Tower tower : world.getTowers()) {
            tower.update(deltaTime);
            tower.draw(tiledMapBatch);
        }
        tiledMapBatch.end();

        batch.begin();
        font.draw(batch, "FPS: " + Gdx.graphics.getFramesPerSecond(), 10, 20);
        batch.end();
    }
```

5.9.6 Mänguekraanis vajalike muudatuste sisseviimine

Lõpetuseks lisame mänguekraani (“**GameScreen.java**”) **render()** meetodisse *if*-tingimuse, mis hakkab mingi aja vältel vaenlaseid tekitama.

Defineerime vajalikud muutujad

```
    private float elapsedTime;
```

Aja jälgimiseks loome **elapseTime** muutuja, mis hoiab aega selle kohta, millal tekitati viimane vaenlane.

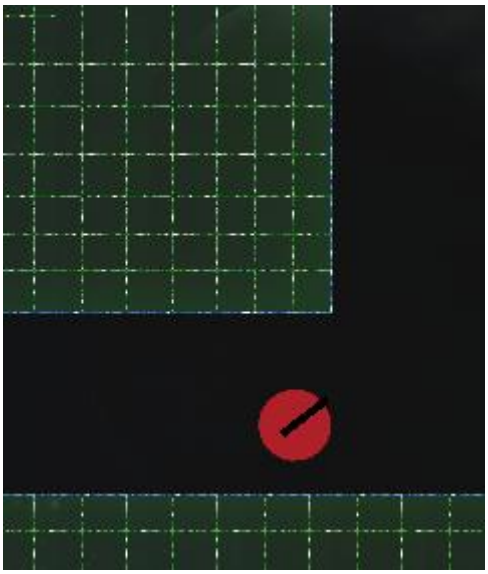
Meetod render() muutmine

```
        elapsedTime += deltaTime;
        if (elapsedTime > 5) {
            world.createEnemy(EnemyType.TEST_ENEMY);
            elapsedTime = 0;
        }
```

Tingimuse kirjutame **render()** meetodi algusesse. Aeg antakse Libgdx raamistikus sekunditena ja tingimus läbitakse, kui möödas on viis sekundit, seega loome uue vaenlase iga viie sekundi tagant.

5.9.7 Rakenduse käivitamine

Rakenduse käivitamisel peaksid vaenlased liikuma mööda ühte rada ja vastavalt suunale ka ennast pöörama (Joonis 13). Nagu näha, siis vaenlased, mis maailmas juba liiguvad, ei ole teadlikud tornidest, mis luuakse nende liikumise hetkel.



Joonis 16 Vaenlase liikumine

5.10 Relva loomine ja vaenlase tuvastamine

Käes on viimane peatükk, mille eesmärgiks on luua kaitseehitisele lihtne relv, mille abil oleks võimalik vaenlasi tulistada ja lihtne relva juhtimissüsteem, mis tuvastaks, kas vaenlane asub tornile piisavalt lähedal, et ehitis saaks sihtmärgi lukustada.

5.10.1 Rakettide tehase loomine

Tehas on meile juba tuttav. Tehaste loomine võib tunda tüütuna, kuid meetod hoiab loomise mugavalt ühes kohas ja muutuste korral on vaja muudatusi teha ainult ühest kohast.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

enum RocketType {
    TEST_ROCKET,
}

public class RocketFactory {
    private static int idCounter = 0;

    public static Rocket makeRocket(RocketType rocketType, Vector2 position, Enemy target,
TowerGuidanceSystem towerGuidanceSystem) {
        idCounter++;

        if(rocketType == RocketType.TEST_ROCKET)
            return makeTestRocket(position, target, towerGuidanceSystem);
        else
            return makeTestRocket(position, target, towerGuidanceSystem);
    }

    private static Rocket makeTestRocket(Vector2 position, Enemy target, TowerGuidanceSystem
towerGuidanceSystem) {
        return new Rocket(new Texture(Gdx.files.internal("data/rocket.png")),
                          String.valueOf(idCounter), position, target,
                         towerGuidanceSystem);
    }
}
```

5.10.2 Raketi loomine

Esiteks loome raketi (“**Rocket.java**”), mis teaks, millisest tornist ta tuli ja kes on ta sihtmärk. Loomulikult laiendab raketi klass samuti **DynamicGameObject** klassi, et oleks vähem kirjutamist.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

public class Rocket extends DynamicGameObject {

    private float speed = 3f;
    private float damage = 2f;

    private TowerGuidanceSystem towerGuidanceSystem;
    private Enemy target;

    public Rocket(Texture texture, String ID, Vector2 position, Enemy target,
                  TowerGuidanceSystem towerGuidanceSystem) {

        super(texture, ID, position);
        this.towerGuidanceSystem = towerGuidanceSystem;
        this.target = target;
    }

    @Override
    public void update(float deltaTime) {
        Vector2 normal = target.getPosition().sub(position).nor();
        velocity.set(normal.x * speed, normal.y * speed);
        position.add(velocity.x * deltaTime, velocity.y * deltaTime);

        if (target.getPosition().epsilonEquals(position, 0.5f)) {
            towerGuidanceSystem.contactAchieved(this, target);
        }

        calculateRotation(normal);
    }
```

```
    public float getDamage() {
        return damage;
    }
}
```

Muutujate defineerimine

Muutuja **speed** hoiab meie raketi kiirust ja **damage** määrab ära, kui palju teeb rakett vaenlasele kahjustust. Muutujad **towerGuidanceSystem** hoiab viidet sihtimissüsteemi kohta. Muutuja **target** hoiab viidet sihtmärgi kohta.

Konstruktor

Esiteks kutsume välja laiendusklassi konstruktori. Järgmiseks määrame raketi sihtimissüsteemi ja raketi sihtmärgi.

Meetod update()

Meetodis arvutame normaliseeritud suunavektori ja lisame kiirusvektori raketi positsioonile. Standardne protseduur, mida oleme teinud mitmeid kordi.

Tingimuslauses kontrollime, kas rakett on sihtmärgile lähemal kui 0,5 ühikut, sest vaenlase diameeter maailmaruumis on 1 ühik. Tingimuslause läbides kutsume välja sihtimissüsteemi meetodi **contactAchieved()**, et rakett hävitada ja vajadusel sihtmärgilukustus tornidelt eemaldada.

Lõpetuseks peame meetodiga **calculateRotation()** arvutama uue tekstuuri suuna.

5.10.3 Raketi sihtimissüsteemi loomine

Sihtimissüsteemis (“**TowerGuidanceSystem.java**”) toimub uute sihtmärkide hankimine, rakettide tulistamine (loomine). Lisaks meetod, mis teostab operatsioonid pärast seda, kui rakett on vaenlast tabanud.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.math.Vector2;

public class TowerGuidanceSystem {

    World world;

    public TowerGuidanceSystem(World world) {
        this.world = world;
    }

    public Enemy acquireTarget(Tower tower) {
        for (Enemy enemy : world.getEnemies()) {
            if(tower.getPosition().epsilonEquals(enemy.getPosition(),
               tower.getTargetLockRadius()))

                return enemy;
```

```
        }
        return null;
    }

    public void contactAchieved(Rocket rocket, Enemy enemy) {
        if (enemy.hit(rocket.getDamage())) {
            world.removeEnemy(enemy);

            for (Tower tower : world.getTowers()) {
                if (enemy.equals(tower.getTarget()))
                    tower.setTargetLockOff();
            }
        }

        world.removeRocket(rocket);
    }

    public void fireRocket(RocketType rocketType, Vector2 position, Enemy target,
                           TowerGuidanceSystem towerGuidanceSystem) {

        world.createRocket(rocketType, position, target, towerGuidanceSystem);
    }
}
```

Muutujate defineerimine

Sihtimissüsteemis vajame viidet maailmakonteineri kohta, et saada ligipääs vaenlastele ja kaitseehitistele.

Konstruktor

Konstruktoris määrame maailmakonteineri, mida hakkame kasutama.

Meetod acquireTarget()

Meetodis otsime parameetriks määratud tornile uue sihtmärgi, läbides vaenlaste loendi lihtsa *for*-tsükliga. Kontrollime, kas vaenlane asub torni laskeulatuses või mitte. Tõese väärtuse korral tagastame esimese leitud vaenlase.

Meetod contactAchieved()

Kui rakett tabab sihtmärki kutsutakse välja see meetod. Esimeses tingimuslauses kontrollime, kas vaenlase elud on otsas või mitte. Selleks loome pärastpoole vaenlases meetodi **hit()**, mis tagastab *boolean* väärtuse. Läbides tingimuslause tuleb vaenlane meetodi **removeEnemy()** abil maailmast eemaldada. Lisaks peame *for*-tsükliga käima läbi kõik kaitseehitised ja meetodi **equals()** abil kontrollima, kas eelpool eemaldatud vaenlane oli torni sihtmärgiks või mitte ning tões väärtuse korral sihtmärgilukustus eemaldada. See tegevus on vajalik selleks, et torn ei jääks õhku tulistama.

Lõpetuseks tuleb rakett ka mängust eemaldada kasutades **removeRocket()** meetodit.

Meetod fireRocket()

Selles meetodis loome uue raketi.

5.10.4 Kaitseehitises vajalike muudatuste sisseviimine

Kui meil oli kaitseehitise klass (“**Tower.java**”) enne tühi, siis nüüd kirjutame selle praktiliselt ümber. Meil on nüüd sihtimissüsteem, mis tuleks tornile lisada. Tornil peaks olema ka tulistamissagedus ja tulistamisraadius. Torn peaks suutma ka sihtmärki hankida ja sihtmärgi asukohast vastavalt ennast pöörama.

```
package com.me.justanothertowerdefense;

import com.badlogic.gdx.graphics.Texture;
import com.badlogic.gdx.math.Vector2;

public class Tower extends DynamicGameObject {

    private Vector2 lookAt = new Vector2();

    private float targetLockRadius = 3f;
    private float fireRate = 2f;
    private float coolDown = 0;

    private Enemy target = null;
    private boolean targetLockOn = false;
    private TowerGuidanceSystem towerGuidanceSystem;

    public Tower(Texture texture, String ID, Vector2 position,
                 TowerGuidanceSystem towerGuidanceSystem) {

        super(texture, ID, position);
        this.towerGuidanceSystem = towerGuidanceSystem;
    }

    @Override public void update(float deltaTime) {
        coolDown -= deltaTime;
        if (targetLockOn && coolDown <= 0) {
            towerGuidanceSystem.fireRocket(RocketType.TEST_ROCKET,
                                           new Vector2(position.x, position.y),
                                           target, towerGuidanceSystem);
            coolDown = fireRate;
        }

        if (!targetLockOn) {

            Enemy target = towerGuidanceSystem.acquireTarget(this);
            if (target != null)
                setTargetLockOn(target);

        } else {

            lookAt = target.getPosition().cpy().sub(position).nor();
            if (!position.epsilonEquals(target.getPosition(), targetLockRadius))
                setTargetLockOff();
            else
                calculateRotation(lookAt);
        }
    }

    public float getTargetLockRadius() {
        return targetLockRadius;
    }

    public Enemy getTarget() {
        return target;
    }
```

```
    public void setTargetLockOn(Enemy enemy) {
        targetLockOn = true;
        target = enemy;
    }

    public void setTargetLockOff() {
        targetLockOn = false;
        target = null;
    }

}
```

Muutujate defineerimine

Meil on vaja **lookAt** muutujat, et torni tekstuuri asukohta määrata. Loome ka muutuja **targetLockRadius** tulistamisraadiuse tarbeks. Kuna kaitseehitis ei tohiks tulistada koguaeg, loome muutuja **fireRate**, mis määrab ära tulistamistiheduse. Muutuja **coolDown** määrab ära aja, pärast mida on võimalik jälle tulistada. Järgmiste rakettide tulistamisel on meil vaja sihtmärki, milleks on **target.** Sihtmärgi staatuse jaoks loome muutuja **targetLockOn** ja **towerGuidanceSystem** sihtimissüsteemi tarbeks.

Konstruktor

Konstruktoris määrame torni sihtimissüsteemi ja seega on vajalik muuta ka kaitseehitist tehast.

Meetod update()

Esimene tingimuslause kontrollib, kas jahtumisaeg on läbi ja kas sihtmärk on olemas. Läbides tingimuslause tulistame raketi kasutades **fireRocket()** meetodit ja määrame uue jahtumisaja.

Juhul kui sihtmärk puudub toimub teises tingimuslauses uue sihtmärgi hankimine meetodiga **acquireTarget()**. Kuna me ei pruugi sihtmärki saada on vaja kontrollida, et see on saadud ja alles siis sihtmärk lukustada.

Sihtmärgi olemasolu korral määrame uue vaataesuuna (muutuja **lookAt)**, arvutades seda sihtmärgi positsiooni arvestades. Juhul kui sihtmärk asub kaugemal kui laskeulatus, eemaldame sihtmärgilukustuse meetodiga **setTargetLockOff()**. Kui aga sihtmärk on olemas tuleb **calculateRotation()** abil tekstuuri pöörata.

Meetod getTargetLockRadius()

Tavaline **get()** meetod, millega laskeraadiust teistele toimetada.

Meetod getTarget()

Vajame meetodit sihtimissüsteemis märklaua küsimiseks.

Meetod setTargetLockOn()

Kasutame, et lukustada vaenlane, mida hakkame tulistama.

Meetod setTargetLockOff()

Sihtmärgi lukustuse mahavõtmiseks.

5.10.5 Tornide tehases vajalike muudatuste sisseviimine

Lisasime kaitseehitise konstruktorisse sihtimissüsteemi ja seega tuleb se lisada ka tehasesse ("**TowerFactory.java**"). Muudetud meetodid näevad välja järgmised:

```
public static Tower makeTower(TowerType towerType, TowerGuidanceSystem towerGuidanceSystem,
                              Vector2 position) {
        idCounter++;

        if(towerType == TowerType.TEST_TOWER)
            return makeTestTower(position, towerGuidanceSystem);
        else
            return makeTestTower(position, towerGuidanceSystem);
    }

    private static Tower makeTestTower(Vector2 position,
                                       TowerGuidanceSystem towerGuidanceSystem) {

        return new Tower(new Texture(Gdx.files.internal("data/tower.png")),
                         String.valueOf(idCounter), position, towerGuidanceSystem);
    }
```

5.10.6 Maailmakonteineris vajalike muudatuste sisseviimine

Maailmakonteineris ("**World.java**") on vaja luua rakettide loend ja sihtimissüsteem. Lisaks tavalised meetodid kustutamiseks, lisamiseks ja küsimiseks.

Muutujate defineerimine

```
    private CopyOnWriteArrayList<Rocket> rockets = new CopyOnWriteArrayList<Rocket>();

    private TowerGuidanceSystem towerGuidanceSystem;
```

Konstruktor

```
        towerGuidanceSystem = new TowerGuidanceSystem(this);
```

Meetod getRockets()

```
    public CopyOnWriteArrayList<Rocket> getRockets() { return rockets; }
```

Meetod createRocket()

```
public void createRocket(RocketType rocketType, Vector2 position, Enemy target,
                         TowerGuidanceSystem towerGuidanceSystem) {

    rockets.add(RocketFactory.makeRocket(rocketType, position,
                                         target, towerGuidanceSystem));
}
```

Meetod removeRocket()

```
public void removeEnemy(Enemy enemy) {
    enemies.remove(enemy);
}
```

5.10.7 Vaenlases vajalike muudatuste sisseviimine

Raketi tabamuse korral on võimalik vaenlaselt („**Enemy.java**“) elusid võtta.

Meetod hit()

```
public boolean hit(float healthLost) {
    health -= healthLost;
    if (health <= 0)
        return true;
    else
        return false;
}
```

Vähendame vaenlase elusid ja kontrollime, kas elud on otsas või mitte. Kui vaenlasel on elud otsas tagastame tõese väärtuse, mille korral on võimalik ta mängust eemaldada.

5.10.8 Maailmarenderdajas vajalike muudatuste sisseviimine

Nagu iga objekti korral peame **render()** meetodis *for*-tsükli abil objektid ekraanile joonistama ja erand ei ole ka rakett. Meetodite **begin()** ja **end()** vahele lisame järmise jupi:

```
    for (Rocket rocket : world.getRockets()) {
        rocket.update(deltaTime);
        rocket.draw(tiledMapBatch);
    }
```

5.10.9 Rakenduse käivitamine

Nüüd peaksid kaitseehitised vaenlaseid tulistama, kui neid maastikule lisada (Joonis 14). Juhul kui vaenlased on liiga tugevad võib alati vaenlaste klassis („**Enemy.java**“) nende elupunkte vähendada.

Joonis 17 Vaenlaste tulistamine

6 Ülesanded

Praeguseks peaksid sa olema materjali läbi lugenud ja peaksid omama ettekujutust, kuidas mängu luua. Iga ülesande juures võta lahti peatükk, kus seda teemat käsitlesime. Võta lahti ka teemaga seotud lähtekood ja uuri veelkord, kuidas oleme mingi asja implementeerinud. Väga positiivne on kui leiad, et näidiskoodis olev lahendus ei ole parim ja soovid seda ümber teha.

1. Võta lahti **level1.tmx** fail ja loo selle põhjal uus tasand. Kasuta tasandit rakenduses.
2. Loo uus ekraan, mis võimaldaks meil valida tasandeid. Võimalda valikut teha kahe tasandi vahel.
3. Muuda nuppude tõmmiseid.
4. Muuda vaenlase ja torni tõmmiseid.
5. Võimalda kasutajal nupuga valida mängu kiirust.
6. Inverteeri kaamera liikumissuund.
7. Lisa uut tüüpi vaenlane.
8. Lisa uut tüüpi kaitseehitis.
9. Lisa heliefekt, mida mängitakse siis, kui kaitseehitis tulistab. *Vihje: vaata LibGDX dokumentatsiooni*
10. Loo lihtne 2D animatsioon kaitseehitisele või vaenlasele. *Vihje: vaata LibGDX dokumentatsiooni.*
11. **Raske.** Võimalda kasutajal mäng salvestada. *Vihje: Vaja on luua uus fail ja võimaldada see ka sisse laadida. Vaja on salvestada: vaenlaste asukohad ja rajad, kaitseehitiste asukohad ja sihtmärgid, rakettide asukohad ja sihtmärgid.*
12. **Raske.** Optimiseeri rajaleidjat nii, et kõik vaenlased kasutaks sama rada, mida kasutatakse kontrollpunktide leidmiseks.