TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Kaspar Romulus 155401IAPB

3D TULISTAMIS- ELLUJÄÄMISMÄNGU ARENDUS UNITY MÄNGUMOOTORIL

Bakalaureusetöö

Juhendaja: Tanel Tammet
Ph.D.

Tallinn 2019

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Kaspar Romulus

09.01.19

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua kolmemõõtmeline ellujäämismäng, milles mängija eesmärk on võimalikult palju vaenlasi hävitada, kaua elus püsida ja münte koguda, millega oma tegelast uuendada.

Töös antakse ülevaade mängu loomiseks kasutatud rakendustest Unity, Blender ja JetBrains Rider. Kirjeldatakse erinevaid võimalusi mida need rakendused pakuvad ja kuidas need võimalused aitavad kaasa mängu arendamisel.

Töös kirjeldatakse mängu loomise protsessi kõiki osasid. Üldise mänguloogika loomisest, tegelaste ja mängutaseme modelleerimisest kuni menüüde disainini ja seda kuidas Unity mängumootor kõiki neid tegevusi arendaja jaoks lihtsamaks teeb.

Töö tulemusena valminud mängu on võimalik alla laadida https://ufile.io/ez9hd.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 38 leheküljel, 5 peatükki, 30 joonist, 0 tabelit.

Abstract

3D survival shooter game development on Unity game engine

The aim of this thesis is to use Unity game engine to develop a 3D survival shooter game, where the player must kill as many enemies as possible while trying to survive.

This thesis gives an overview of the software used to develop this game. Most importantly Unity game engine, but also Blender modeling software and JetBrains Rider code editor. Also the opportunities these software provide and how they help in the development of the game.

In this thesis the author explains all the steps necessary to create such a game from setting the basic game logic, to modeling the characters and game level and creating menus and HUD´s and how Unity can make all of these steps easier for the developer.

The game that was created for this thesis can be downloaded from https://ufile.io/ez9hd.

The thesis is in Estonian and contains 38 pages of text, 5 chapters, 30 figures, 0 tables.

Lühendite ja mõistete sõnastik

iOS	Apple mobiilne operatsioonisüsteem
HUD	Osa kasutajaliidesest mille abil kuvatakse informatsiooni
UI	Kasutajaliides

Sisukord

Jooniste loetelu

1 Sissejuhatus

Antud bakalaureusetöö eesmärgiks on luua kolmemõõtmeline tulistamis-ellujäämismäng, milles mängija peab proovima võimalikult palju vaenlasi tappa, kasutades Unity mängumootorit.

Käesoleva töö autor on varasemalt mängude arendusega kokku puutunud kahel korral. Mõlemal neist on kasutatud Unity mängumootorit. Esimene arendus piirdus limiteeritud teadmiste tõttu lihtsalt mängumootoriga tutvumisega ja mõnede lihtsamate asjade katsetamisega. Teine arendus valmis koolis paaristööna ning tegemist oli lihtsa mobiilimänguga, kus tegelane jooksis ning ekraanile vajutustega üle takistuste hüppas.

Töö esimeses pooles uuritakse kui populaarsed on erinevad mobiiliplatvormid ning tutvustatakse lühidalt mobiilimänge. Lisaks antakse ülevaade Unity mängumootori võimalustest mängude loomisel ja Blender modelleerimistarkvara võimalustest kolmemõõtmeliste objektide loomisel, tegelaste loomisel ja animeerimisel.

Töö teises pooles kirjeldatakse mängu loomise protsessi ja erinevaid probleeme mida selle protsessi käigus lahendada tuli. Alustades lihtsamatest asjadest nagu tegelaste ja vaenlaste prototüüpide loomisest, mis on lihtsad tänu Unity mängumootori poolt pakutud võimalustele. Hiljem luuakse nendele prototüüpidele, aga ka kolmemõõtmelised mudelid mille loomine osutus raskeimaks osaks arenduse juures. Lõpetades erinevate lisa vaadete loomisega, nagu avamenüü vaade ja mängija uuendamise vaade.

Töö tulemusena valminud mängu on võimalik alla laadida https://ufile.io/ez9hd.

Töö on koostatud aastal 2018 Tallinna Tehnikaülikooli informaatika bakalaureuseõppe eriala lõputööna.

2 Taust

2.1 Erinevad mobiiliplatvormid

Kaks suurimat mobiilset operatsioonisüsteemi on Android ja iOS, mis kahepeale katavad peaaegu 99% kogu turust. Nendest kahest populaarsem on Android, mida kasutab ligi 70% nutitelefoni [1]. Mängu üleslaadimiseks iOS Apple Store'i tuleb maksta iga aasta 99 dollarit. Androidi Play Store'i mängu üleslaadimiseks tuleb aga maksta ühekordne tasu summas 25 dollarit [2]. Kuigi Unity toetab mõlemale operatsioonisüsteemidele arendamist ilma suuremate muudatuste tegemiseta mängus, on otsustatud mäng arendada ainult Android operatsioonisüsteemil.

2.2 Mobiilimäng

Mobiilimäng on mäng, mida saab mängida telefoni või tahvelarvutiga. Esimene populaarseim mobiilimäng ilmus aastal 1997 kui Nokia telefonidega tuli kaasa mäng nimega Snake. Seoses mobiilseadmete kiire arenguga on ka mobiilimängud kiirelt arenenud. Aastal 2018 on mobiilimängude turu väärtus 50 miljardit dollarit ja ligikaudu 2,1 miljardit inimest üle maailma mängib mobiilimänge [3].

3 Mobiilimängu loomiseks kasutatud vahendid

Järgnevalt kirjeldatakse kasutatuid tehnoloogiad ja võimalusi, mida need mängu loomisel pakuvad.

3.1 Unity

Unity on 2005. aasta Juunis ilmunud mängumootor. Tänase seisuga pakub Unity 27 platvormi tuge mille hulgas on PC, Max ja Linux, Android, iOS, Xbox One, PS4, WebGL veebimängude jaoks ja Facebook. Mängumootorit saab kasutada nii kahe- kui ka kolmemõõtmeliste mängude loomiseks. Unity peamiseks skriptimiskeeleks on C#. Käesolevas töös on kasutatud Unity Personal versiooni, mis on kõigile kasutajatele tasuta, piirangul, et loodud mängu tulu ei ületa 100 000 dollarit. Lisaks Personal versioonile on olemas ka tasulised versioonid Plus umbes 25 dollari eest kuus ja Pro 125 dollari eest kuus [4].

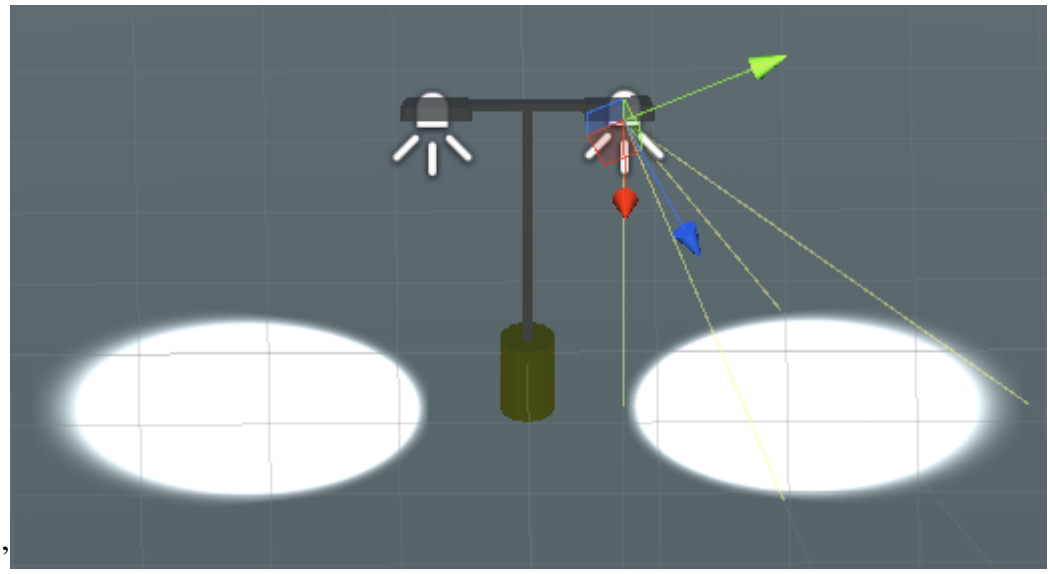
3.1.1 Unity peamised funktsionaalsused

Unity teeb lihtsaks erinevate tasemete ja vaadete loomise ja vahetamise, kasutades selleks *scene*. *Scene* võib olla mängu tase või lihtsalt menüü vaade.

Unity teeb lihtsaks nii kahe- kui kolmemõõtmeliste põhiliste objektide lisamise (vt. Joonis 1).

Joonis 1. Unity abil loodud kolmemõõtmelised objektid silinder, kuup ja kera.

Unity’s on nelja tüüpi valgustust, mida kõiki saab arendaja on äranägemise järgi muuta. Põhilised omadused valgusel on valguse tugevus, valguse värv ja mis tüüpi varjud valguse mõjul jäävad (vt. Joonis 2).



Joonis 2. Unity abil loodud Spot Light.

Unity muudab lihtsaks kokkupõrgete vältimise et objektid üksteise sisse ei läheks ja samuti kokkupõrgete tuvastamise juhul kui objektid teineteisele piisavalt lähedal on. Unity kasutab selleks *collidereid* mida saab märkida *triggeriks* juhul kui soovitakse et kokkupõrkel midagi juhtuks.

Unity võimaldab objektide jaotamist *tag*’i ehk sildi ja *layer*’i ehk kihi kaudu. *Tag*’e saab kasutada näiteks skriptide seest teiste objektide otsimiseks. *Layer*’i abil saab jagada hulga objekte ühele kihile, et hiljem mänguloogikat kergem teha oleks.

Unity’s on iga uue stseeni loomisel koheselt olemas ka kaamera, läbi mille hakatakse mängija ekraanile toimuvat kuvama.

Unity võimaldab luua loodud objektidest *prefab*’e mis on sisuliselt koopia loodud objektist. See tuleb kasuks juhul, kui on vaja sama objekti korduvalt uuesti luua, antud töö raames näiteks vaenlaste tekitamisel.

Unity teeb mugavaks kasutajaliideste implementeerimise, kasutades selleks *canvas* ehk lõuend tüüpi mänguobjekti. Sellele saab lisada nuppe ja muud teavet, mida mängijale vaja kuvada on ja see on alati mängijale näha.

Unity pakub ka oma pilvesüsteemi *cloud build*, mis teeb arendamise suurema meeskonna puhul mugavamaks. Samuti annab see võimaluse paremaks versioonihalduseks.

Unity abil loodud rakenduste abil tulu saamiseks on võimalusteks reklaamide lisamine mängu ja mängusiseste ostude implementeerimine.

Unity *asset store* võimaldab kasutajal alla laadida erinevaid valmis mängu osasid, nagu näiteks tegelaste mudeleid või tegelase liigutamise skripte.

3.2 JetBrains Rider

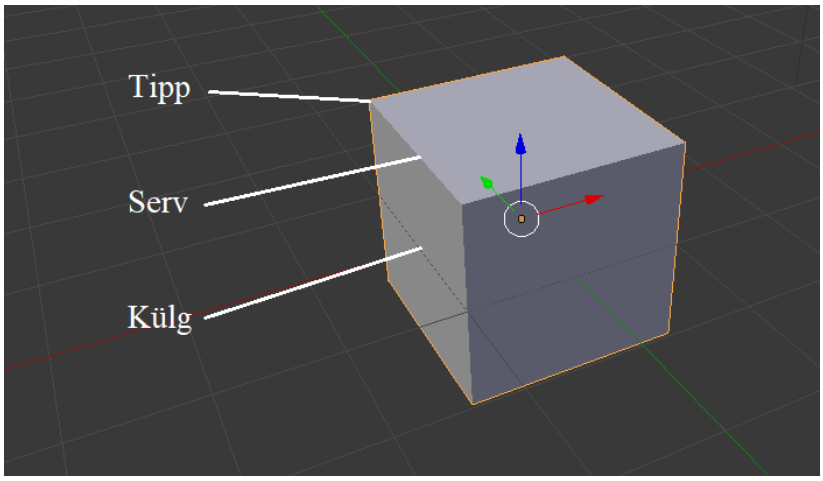
JetBrains Rider on integreeritud arenduskeskkond loodud C# ja .NET arendamiseks. Aitab arendamisel kaasa koodi kirjutamisel sõnade lõpetamisega, koodi refaktoreerimisega ja märgib muutujad ja funktsioonid eri värvides, et koodi lihtsam lugeda oleks [5].

3.3 Blender

Blender on vabavarana saadav tarkvara, mille abil saab luua ja animeerida kolmemõõtmelisi objekte [6].

3.3.1 Modelleerimine

Modelleerimine on Blenderi põhiline funktsionaalsus, mis võimaldab kasutajal luua baaskujundeid nagu näiteks kuup ja kera. Samuti on Blenderil palju võimalusi nende loodud kujundite modifitseerimiseks kasutades selleks tippe, servasid või külgi (vt. Joonis 3).



Joonis 3. Kujund Blenderis.

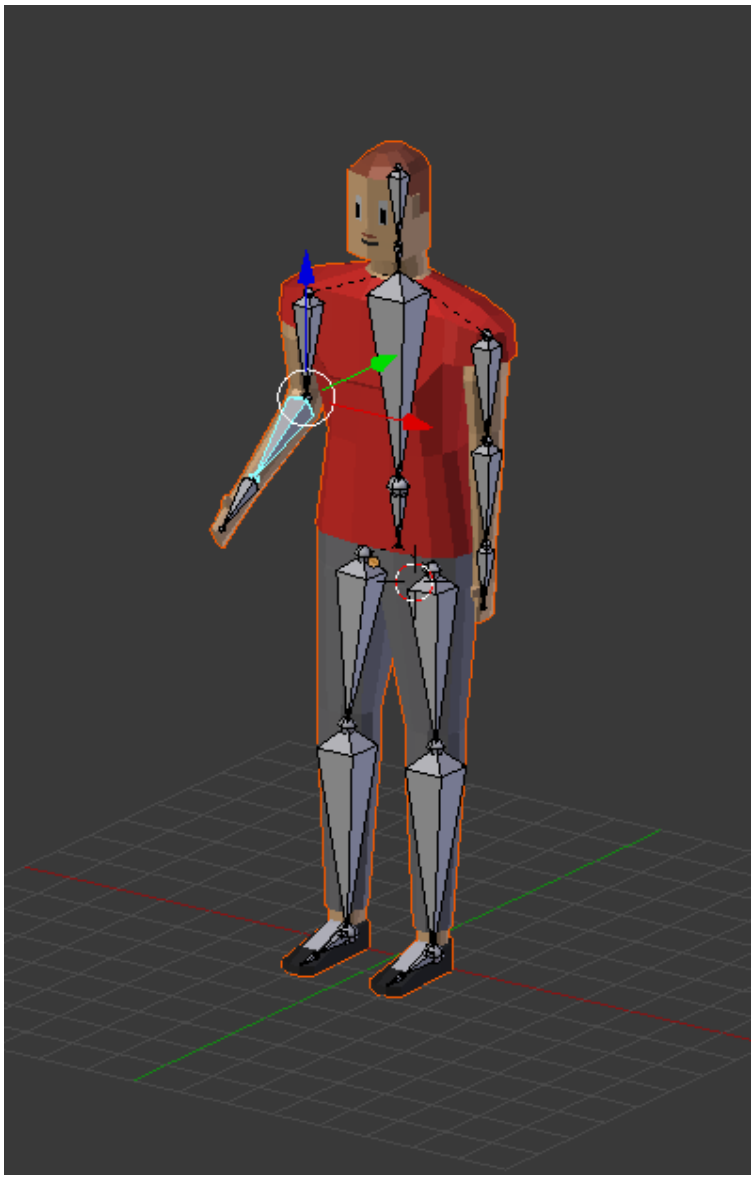
Samuti on Blenderil palju sisseehitatud modifikaatoreid, millest olulisemad antud töö juures on peegel modifikaator, mis võimaldab kopeerida objekti X, Y või Z telje suhtes ja *boolean*, mis lubab ühte objekti teisega ühendada, ühte objekti teisest lahutada või leida objektide ühisosa (vt. Joonis 4).

Joonis 4. Blenderi erinevad modifikaatorid.

3.3.2 Animeerimine

Blender teeb lihtsaks loodud tegelase armatuuri, ehk luustiku lisamise (vt. Joonis 5). Selle järgi saab tegelast lihtsasti liikuma panna, pöörates erinevaid armatuuri osasid ehk luid.

Nii saab teatud kaadritel tegelase poosi, *keyframe*'i kasutades, paika panna ja kõikidel kaadritel, mis *keyframe*'ide vahele jäävad suudab Blender interpoleerimist kasutades tegelase poosi ise määrata.



Joonis 5. Armatuuriga tegelane.

3.3.3 Exportimine

Unity võimaldab kasutada ka *.blend* tüüpi ehk Blenderi projektide faile. Antud töö raames on kõik mudelid segaduse vältimiseks siiski *export´*itud *.fbx* failidena.

4 Mobiilirakenduse arendamine

Järgnevalt kirjeldatakse mängu „Human Apocalypse“ loomist ideest kuni valmis mänguni.

4.1 Mängu sisu

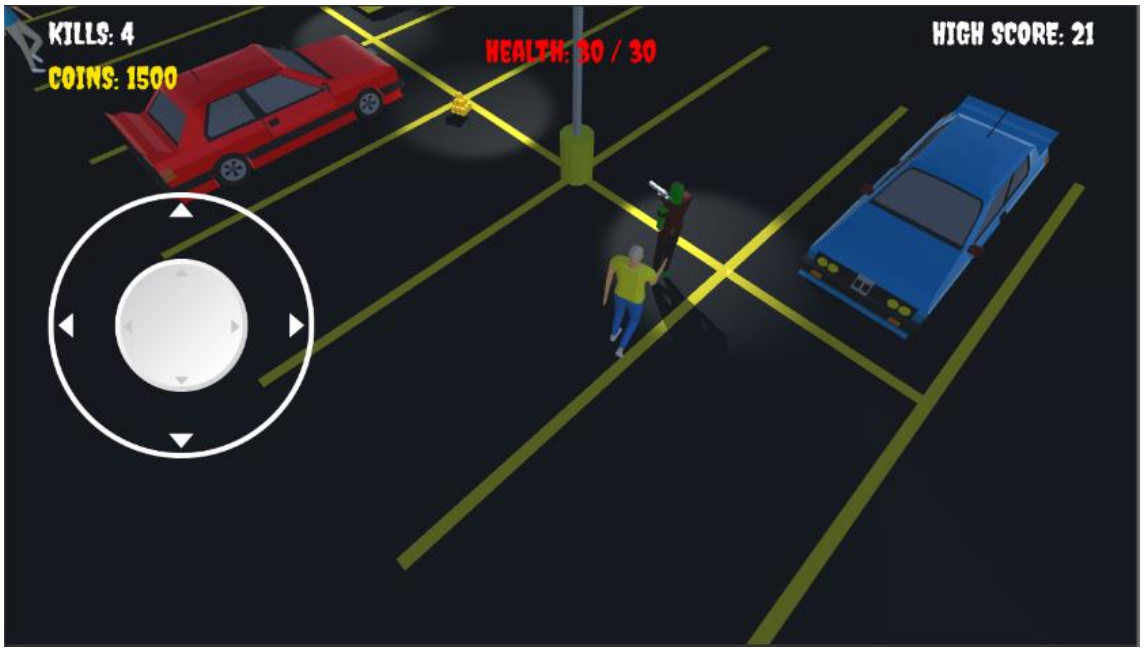
Mäng on olemuselt tüüpiline tulistamis- ellujäämismäng, kus mängija eesmärk on elus püsida ja võimalikult palju vaenlasi tappa. Vaenlasi tappes suureneb mängija skoor ja mängu eesmärgiks võib nimetada ka võimalikult suure skoori saamist. Mängu alguses on mänguväljakul ainult mängija tegelane, aga aja möödudes hakkavad sinna, varem ette määratud punktidesse, tekkima ka vastased. Nende vastaste ainus eesmärk on liikuda mängija poole ja juhul kui nad mängijale piisavalt lähedale jõuavad, löövad nad mängijat, mis vähendab mängija elusid. Kui mängijal elud otsa saavad, lõppeb mäng.

4.2 Mängu käik

Mängu alustades kuvatakse mängijale ava menüü stseen (vt. Joonis 6). Edasi on kasutajal valik, kas ta soovib alustada uut mängu või minna tegelase uuendamise vaatesse.

Joonis 6. Ava menüü stseen.

Kui kasutaja valib uue mängu alustamise algab mäng, kus kasutaja saab tegelast juhtida kasutades ekraani vasakus servas asuvat juhtkangi ja vaenlasi tulistada, ekraanil nende suunas vajutades (vt. Joonis 7). Vaenlase tapmiseks tuleb teda tulistada kaks korda. Iga vaenlase tapmisega suurendatakse mängija skoori. Vaenlaste suremisel võib nende asemele tekkida münte, mida kasutaja saab ülesse korjata üle nende kõndides ja hiljem kasutada tegelase kiiremaks muutmiseks või tegelase elude arvu suurendamiseks. Mängu käigus proovivad vaenlased mängijat rünnata ja kui mängija elud otsa saavad lõppeb mäng. Et vältida olukorda kus mäng võiks kesta lõpmatult, hakkab aja jooksul vaenlasi tekkima tihedamalt ja suurendatakse tekkinud vaenlaste kiirust.



Joonis 7. Mängu stseen.

Mängu lõppedes kuvatakse kasutajale kõigepealt vaadet kus pakutakse võimalust eelnevas mängus teenitud punktide arv kahekordistada, vaadates lühikest reklaami (vt. Joonis 8). Selle järel kuvatakse kasutajale taas ava menüü stseen.

Joonis 8. Mängu lõpu stseen.

Teine valik avamenüüs on mängija uuenduste vaade (vt. Joonis 9). Seal on kasutajal võimalik mängudes teenitud müntide eest oma tegelast kiiremaks muuta või tegelase elude arvu suurendada.

Joonis 9. Uuenduste stseen.

4.3 Mängija juhtimine

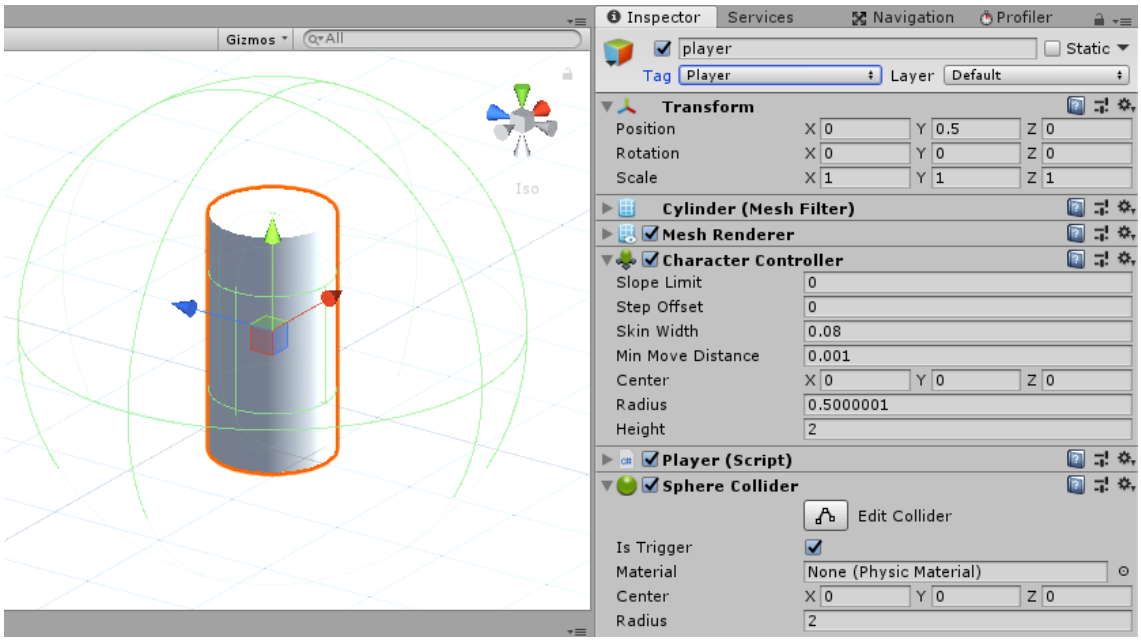
Mängija esialgsel loomisel ei ole oluline milline ta välja näeb ja selle tõttu kasutan Unity´s olevat *cylinder* tüüpi kolmemõõtmelist objekti. Et mängijat oleks võimalik liigutada, tuleb mänguobjektile lisada *character controller* ja skript. Samuti tuleb mängijale lisada üks mängijast suurem *collider* mille järgi hakkavad vaenlased aru saama, et nad on mängijale piisavalt lähedal. See tuleb märkida *trigger*'iks, et kokkupõrkeid lihtsam tuvastada oleks. Samuti tuleb mängija märkida sildiga „Player", et hiljem saaks näiteks vaenlase puhul kindlaks teha, et *trigger* millesse ta sisenes kuulub just mängijale.

Mängija reaalseks liigutamiseks on ekraanile lisatud mängukang, mis on alla laetud *asset store*'st [8]. See tuleb lisada lõuend tüüpi mänguobjektile. Lõuendi loomisel lisatakse automaatselt ka *eventtrigger* tüüpi mänguobjekt, mille abil tuvastatakse puuteid lõuend tüüpi mänguobjektil.

```
void Move(Vector3 movement)
{
    var acceleration = Math.Min(Math.Abs(movement.x) + Math.Abs(movement.z), 1f);
    movement = movement * speed * Time.deltaTime * acceleration;
    playerController.Move(movement);
}
```

Joonis 10. Tegelase liigutamise skript.

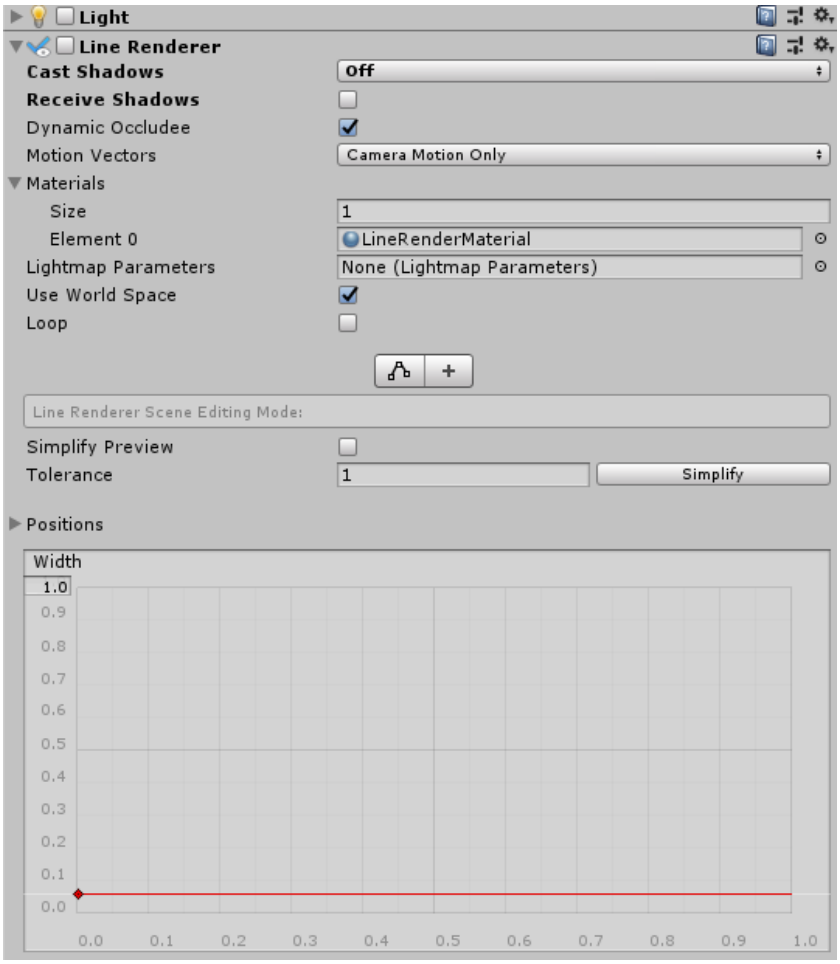
Skripti osadeks on sisend *movement*, mis tuleb juhtkangilt ja määrab kuhu suunas liigutatakse. *Acceleration* määrab kiiruse juhul kui juhtkangi ei liigutata serva ja tegelane peab liikuma aeglasemalt kui tema tegelik kiirus. *PlayerController* on varasemalt määratud mänguobjekti komponent *character controller* (vt. Joonis 6).



Joonis 11. Tegelase prototüüp vajalike komponentidega.

4.4 Tulistamine

Tulistamise tegemiseks tuleb luua uus mänguobjekt, mille asukoht peab olema relva tipus ja mis tuleb suunata relvast eemale. Kõige olulisem komponent sellel objektil on *line renderer*, mille abil hakatakse kuule kuvama. Lisaks sellele lisatakse ka valguse komponent, et tulistamist natuke reaalsemaks teha (vt. Joonis 8). Ka see mänguobjekt on mõistlik märkida sildiga, et seda pärast lihtsam leida oleks.



Joonis 12. Tulistamiseks vajalikud komponendid.

Sihtimise loogika seisneb selles, et tegelane keerab ennast sinna suunda, kuhu mängija on ekraanil vajutanud. Et puute asukohta kindlaks määrata kasutatakse *quad* tüüpi mänguobjekti, mis tuleb teha nähtamatuks. Selleks tuleb mänguobjektilt eemaldada *mesh renderer* komponent. See mänguobjekt peab olema vähemalt sama suur kui mängutase ja kõrgus peab olema sama mis maapinnal, samuti luuakse selle mänguobjekti jaoks uus kiht „*Floor*“. Kui puude ekraanil on selle peal, antakse mängijale käsklus ennast sinna poole keerata.

Et puuteid tuvastada, tehakse terve lõuendi ulatuses nupp. Tulistamiseks ei pea iga lasu jaoks uuesti vajutama, vaid mängija võib oma näppu ekraanil hoides ja lohistades järjest tulistada. Iga lasu vahel on siiski väike ajavahemik.

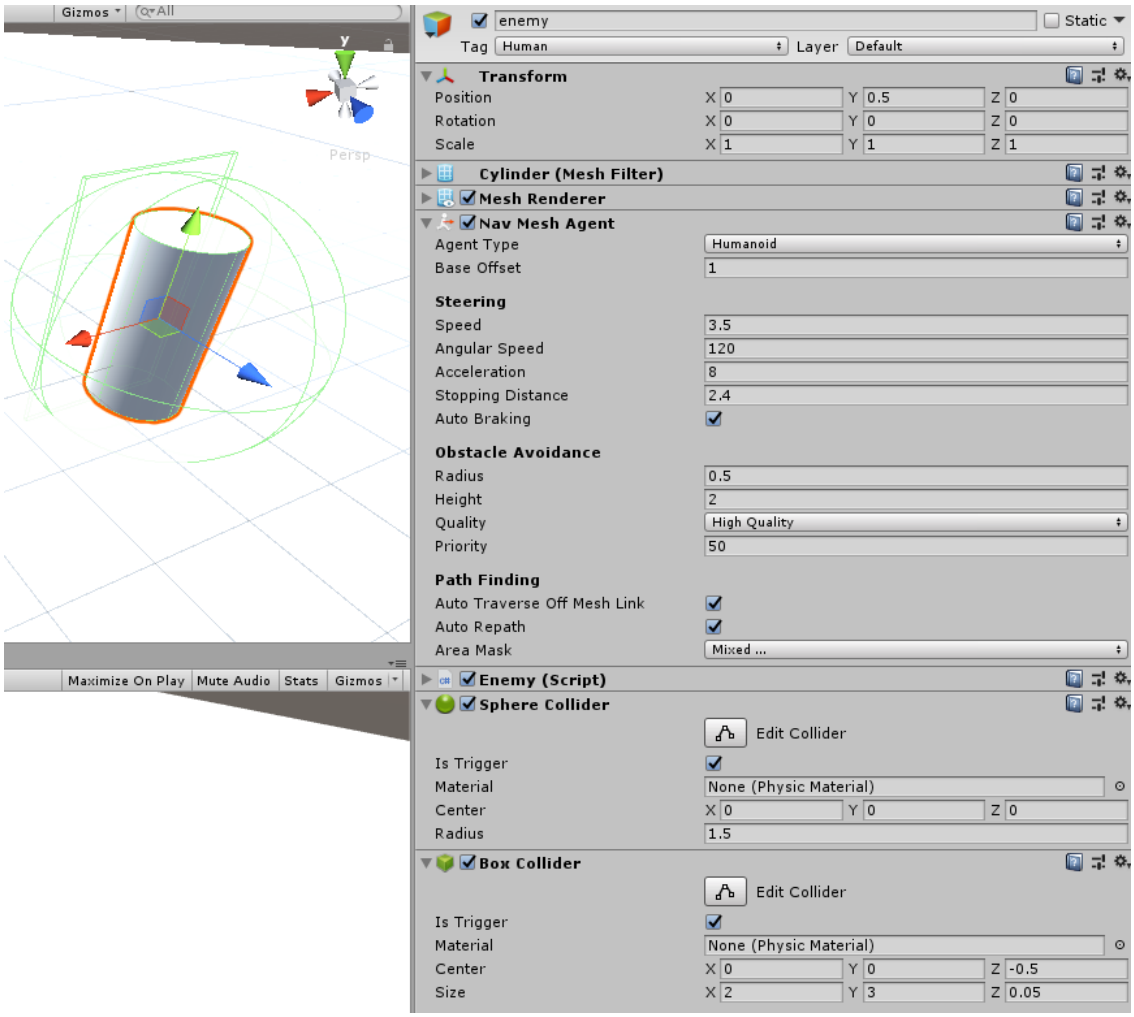
Tulistamise kokkupõrgete tuvastamiseks kasutatakse *raycast*'i. See on nähtamatu joon, mis liigub tulistamise suunas. Kui sellel tuvastatakse kokkupõrge mõne mänguobjektiga, mis asub tulistavate objektide kihil nagu näiteks vaenlased või mõni muu kõrgem

mängutaseme osa, siis märgitakse joone lõpp-punktiks see punkt, kus kokkupõrge toimus. Vastasel juhul kuvatakse joon piisavalt pikalt, et mängija ei näeks selle lõpp-punkti. Kui mängija tabab vastast vähendatakse vastase elusid ühe võrra.

4.5 Vaenlaste loomine

Vaenlase loomisel alustan samuti *cylinder* tüüpi mänguobjektist. Kõige olulisem komponent vaenlase juures on *nav mesh agent*, mille abil hakkab vaenlane koguaeg mängija poole liikuma. Lisaks sellele muidugi vaenlase skript ja *trigger* tüüpi *collider*'it, millega hakatakse kontrollima kas mängija on piisavalt lähedal, et vaenlane saaks teda rünnata ning laskude korral, kas need olid piisavalt lähedal, et minna vaenlasele pihta. See *collider* peab olema suurem kui tegelane ise, sest vaenlane ei pea olema täielikult mängija vastas, vaid sobib kui ta on natuke eemal enne kui ründamist alustab. Samuti teeb see natuke lihtsamaks sihtimise, kuna väikese eksimise korral suunatakse sihik siiski vaenlase peale. Juhuks kui vaenlane on mängija vahetus läheduses tuleks lisada veel üks *trigger* tüüpi *collider* mis asub vaenlasest natuke taga pool, kuna laskmisel kontrollitakse *collider*'isse sisenemisest ja ei registreerita lasku pihta saanuks, kui laskmise hetkel on relva ots vaenlase *collider*'i sees.

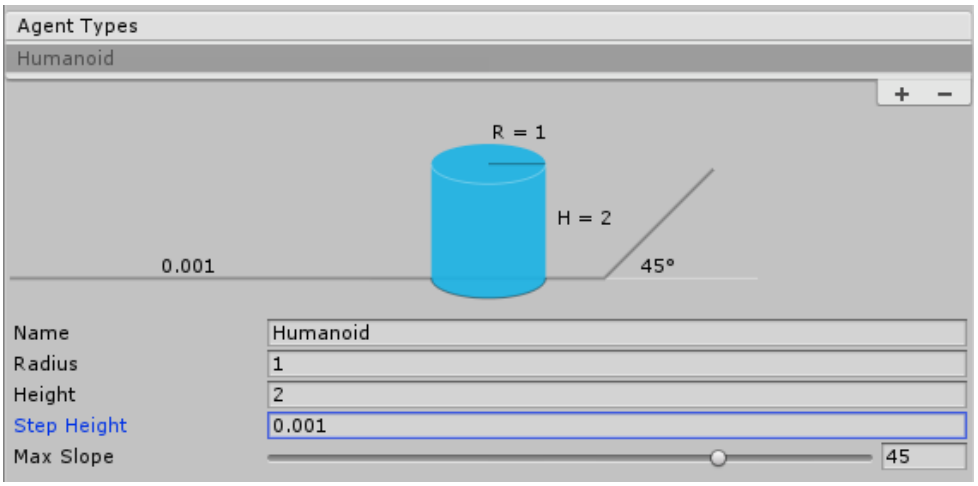
Mängija ründamiseks kutsutakse ühe sekundilise viitega mängija skriptist funktsioon, mis vähendab mängija elusid kümne võrra. Viide on lisatud, et löögi animatsioon ja hetk, millal mängija elud vähenevad langeksid kokku. Samuti tagab viide, et juhul kui mängija suudab vahepeal vaenlase tappa, ei kaota mängija elusid. Igal vaenlased on kaks elu, ehk mängija peab teda tulistama kaks korda, et ta sureks.



Joonis 13. Vaenlase prototüüp vajalike komponentidega.

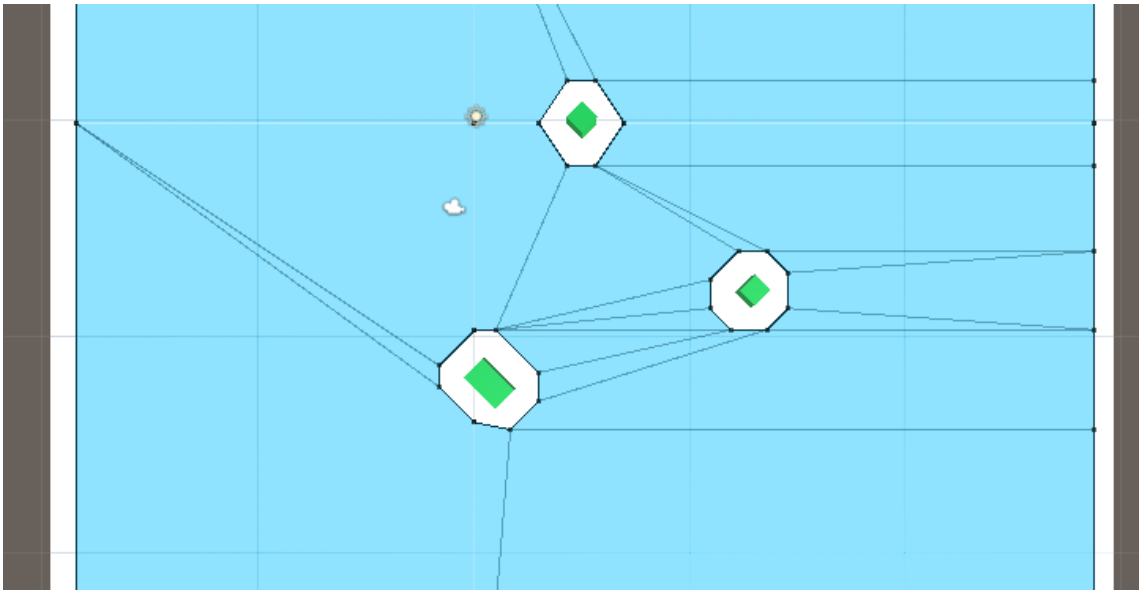
4.6 Vaenlaste liigutamine

Vaenlaste liigutamiseks kasutatakse Unity´s olevat navigeerimise süsteemi. Selle sees tuleb kõigepealt määrata agendi tüüp ja vastavad muutujad. Agendi nimi, suurus, maksimaalne astme kõrgus ja maksimaalne kallak (vt. Joonis 10).



Joonis 14. AI agendi muutujad.

Seejärel tuleb kõik objektid, mille peal agent liikuda saab ühele kindlale kihile panna. Samuti tuleb need objektid märkida tüübiks *navigation static*, mis tähendab, et need ei saa enam liikuda. Seejärel tuleb *navigation* süsteemi kasutades luua ala, kus agendid liikuma hakkavad (vt. Joonis 11).



Joonis 15. Näide Navigationi abil loodud alast, kus agent liikuda saab.

Nüüd piisab koodi poolelt ainult *nav mesh agent* komponendi ja mängija leidmisest ja iga *update* funktsiooni korral uue sihtkoha määramisest (vt. Joonis 12).

```
void Awake()
{
    player = GameObject.FindGameObjectWithTag ("Player");
    nav = GetComponent <UnityEngine.AI.NavMeshAgent> ();
}

void Update () {
    nav.SetDestination (player.transform.position);
}
```

Joonis 16. Vaenlase liigutamine mängija poole.

4.7 Tegelaste modelleerimine

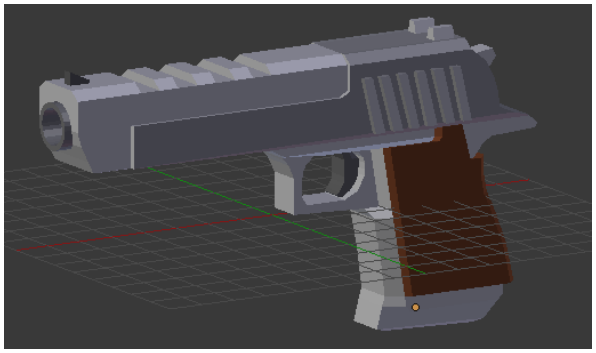
Kuna mängija tegelane ja vaenlased on väga sarnased alustan vaenlase mudeli tegemisest. Vaenlase modelleerimiseks kasutatakse peegel modifikaatorit, mille abil saame luua ainult parema poole tegelasest. Modifikaator tagab, et tegelase vasak pool tuleb identne. Esialgu luuakse tegelase kuju väga ebamäärane, kõik kehaosad luuakse kastist. Seejärel kasutatakse nendel kastidel *loop cut* tööriista, mille abil jagatakse kastid külgede abil. Nüüd saame neid külgi liigutades muuta tegelase reaalsemaks. Tegelase modelleerimine osutus üheks raskeimaks ja ajakulukamaks tegevuseks mängu loomise juures. Vaenlase värvimiseks tuleb valida vastavad tahud ja nende värv. Vaenlase mudel on näha joonisel 13.

Joonis 17. Vaenlase mudel.

Mängija tegelase loomiseks võetakse aluseks sama mudel, aga tehakse sellel väikeseid muudatusi, et ta näeks välja rohkem nagu elav surnu. Eemaldatakse ühe käe alumine osa ja asendatakse see luuga ja luuakse tegelasele peahaav. Samuti eemaldatakse tegelaselt pöial, sest mängus relva hoides ta seda ei vaja. Tegelase mudel on näha joonisel 14.

Joonis 18. Tegelase mudel.

Samuti on oluline osa mängija juures relv. Relv modelleeritakse eraldi ja see ei ole otseselt tegelasega seotud, vaid liigub lihtsalt tegelasega kaasas (vt. Joonis 15).



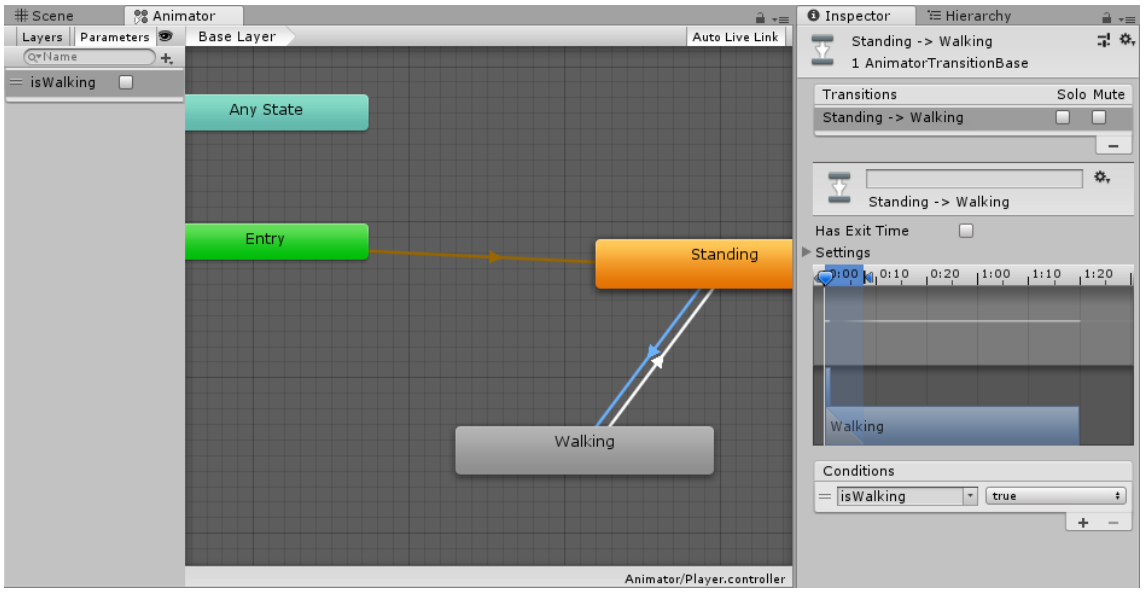
Joonis 19. Relva mudel.

4.8 Tegelaste animeerimine

Mängija tegelase animeerimiseks alustan poosist, kus tegelane hoiab paremat kätt enda ees üleval, nagu hoiaks ta relva. See on poos, mida kasutatakse kui mängija on paigal. Järgmiseks tuleb animeerida kõndimine, kõndimise animeerimiseks kasutatakse kokku üheksat poosi ja kõndimise animatsiooni mängitakse korduvalt.

Vaenlase animeerimiseks kasutatakse tegelasele loodud kõndimise animatsiooni ja luuakse kaks uut animatsiooni. Esimene ründamiseks, mis käivitatakse siis kui vaenlane on tegelasele piisavalt lähedal ja lööb teda. Teine on mudeli algolek, mida kasutatakse kui vaenlane ei saa liikuda, sest teised vaenlased on tal ees või ründamise animatsioonide vahel.

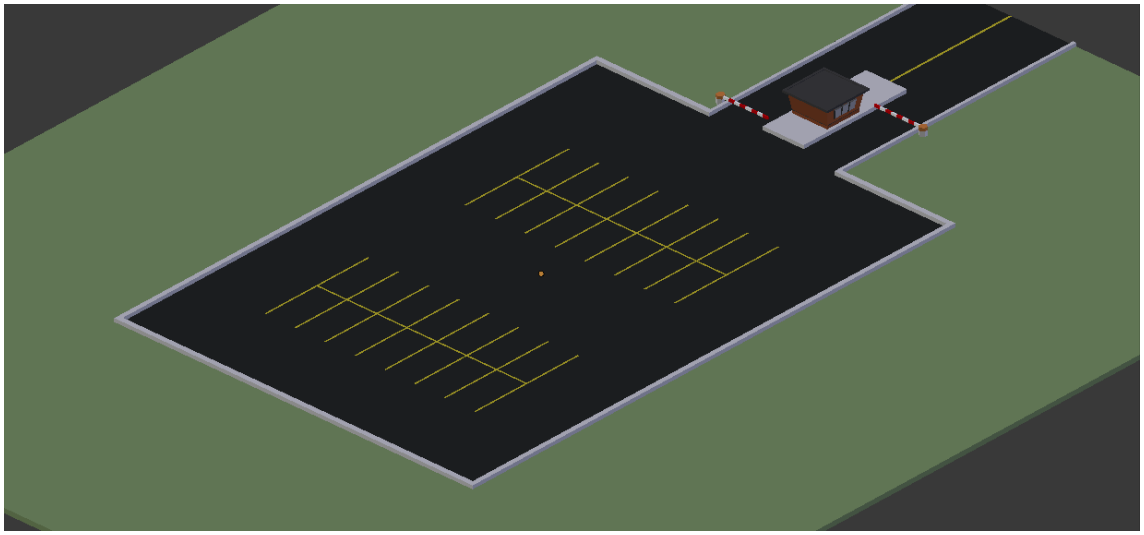
Animatsioonide alustamiseks, lõpetamiseks ja vahetamiseks kasutab Unity *animator controller*'it. Mänguobjektile tuleb lisada *animator* komponent ja sellele igale tegelasele loodud *animator controller*. *Animator controller* teeb mugavaks transatsioonid ühest animatsioonist teise. Selle tegemiseks tuleb lisada parameetreid, mille järgi neid muutusi tuvastatakse ja seosed animatsioonide ja parameetrite vahel, millal mängitakse ühte või teist animatsiooni. Seejärel piisab tegelasega seotud koodis *animator* komponendi leidmisest ja seal parameetrite muutmisest, et mängus animatsioonid muutuksid. Tänu Unity pakutud võimalustele animatsioonide vahetamiseks, osutus tegelaste animeerimine lihtsaks (vt. Joonis 16).



Joonis 20. Mängija *animator* (vasakul) ja üleminek seismisest kõndimisse (paremal).

4.9 Mängutaseme loomine

Mängutase luuakse kolmest mudelist, milleks on parkla, auto ja tänavavalgustus. Parkla mudel luuakse ühes tasapinnas ja piiratakse see äärekivide ja tõkkepuuga, nendest saab mängutaseme piir. Samuti on osa mängutasemest piletiputka. Parkla mudel on näha joonisel 17.

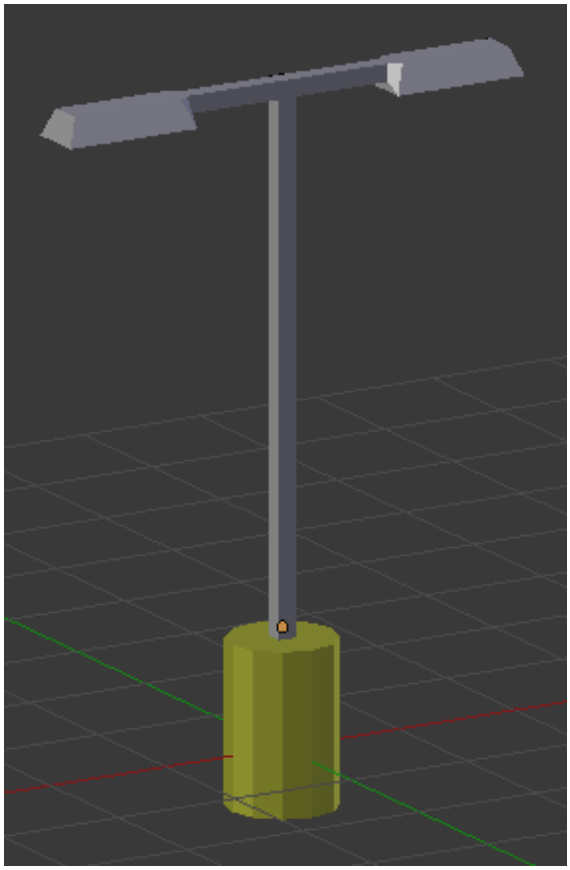


Joonis 21. Parkla mudel.

Teine mudel, mis mängutaseme jaoks luuakse on auto. Mudeli loomiseks kasutatakse peegel modifikaatorit ja luuakse selle abil auto kere. Seejärel lisatakse detaile nagu tuled, esivõre, numbrimärk, antenn ja rattad. Auto mudel on näha joonisel 18.

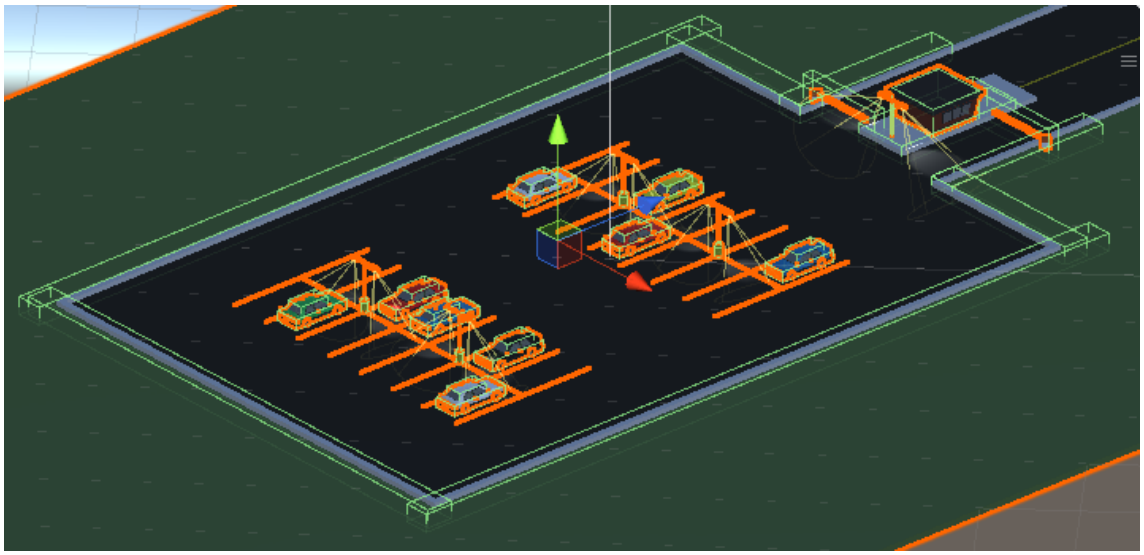
Joonis 22. Auto mudel.

Kolmas mudel on tänavalgustus, millest saab oluline valgusallikas mängus. Selle loomise juures on oluline, et mudel ei oleks liiga suur kuna nii võib ta hakata mängija vaatevälja segama. Mudel luuakse kahest objektist, alumine silinder ja ülemine kast, millest tuleb teha tänavavalgusti post. Posti loomiseks saab kasutada peegel modifikaatorit. Tänavavalgusti mudel on näha joonisel 19.



Joonis 23. Tänavavalgusti mudel.

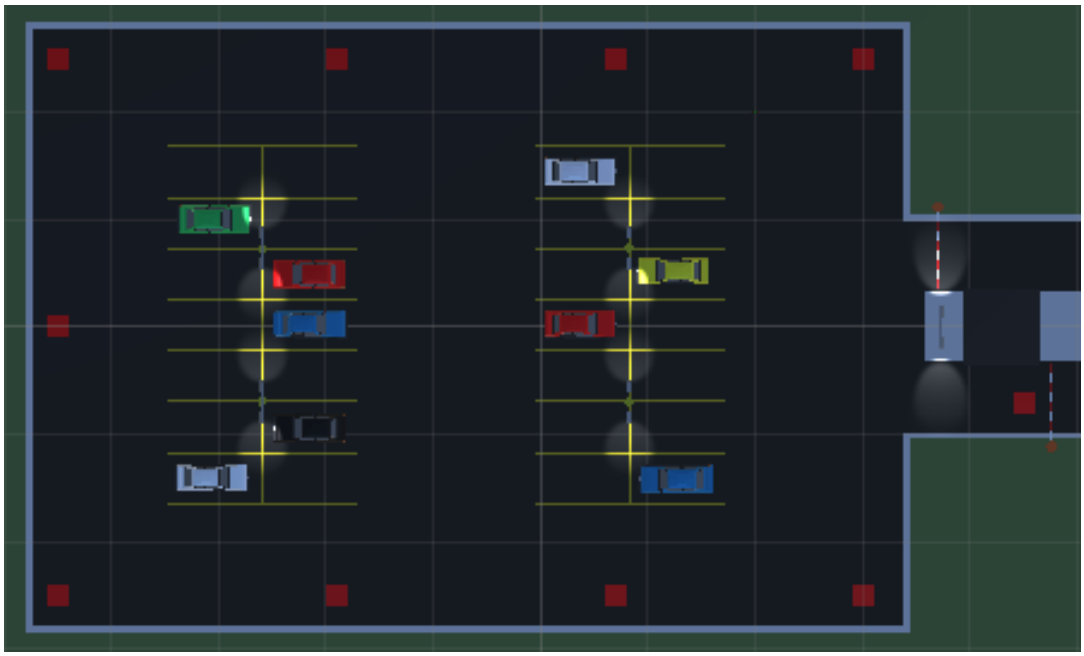
Unity’s taseme loomisel tuleb panna kõikidele objektidele külge *collider*. Seejärel lisatakse kõigepealt parkla mudel ja siis hakatakse sellele lisama autosid ja tänavalgusteid. Tänavavalgustile lisatakse ka kaks *spot light* tüüpi valgusallikat. Kogu stseeni valgustamiseks luuakse üks *directional* tüüpi valgusallikas ja lisatakse sellele sinine alatoon, et valgus meenutaks kuu valgust [9]. Lõplik mängutaseme mudel on näha joonisel 20.



Joonis 24. Lõplik mängutase koos *collider*'itega ja valgustusega.

4.10 Vaenlaste tekitamine

Vaenlaste tekitamiseks tuleb kõigepealt lüüa ühest vaenlase mänguobjektist *prefab*, mis võimaldab meil mängu jooksul sellest mänguobjektist koopiad luua. Vaenlased hakkavad tekkima kümnest varem paika pandud punktist mis kõik asuvad kaardi ääres ja on märgitud sildiga (vt. Joonis 21). Mängu alustamisel luuakse list vastava sildiga märgitud mänguobjektidest ja iga kord valitakse nende hulgast suvaliselt üks.



Joonis 25. Võimalikud punktid, kus vaenlased tekkima hakkavad on märgitud punase kastiga.

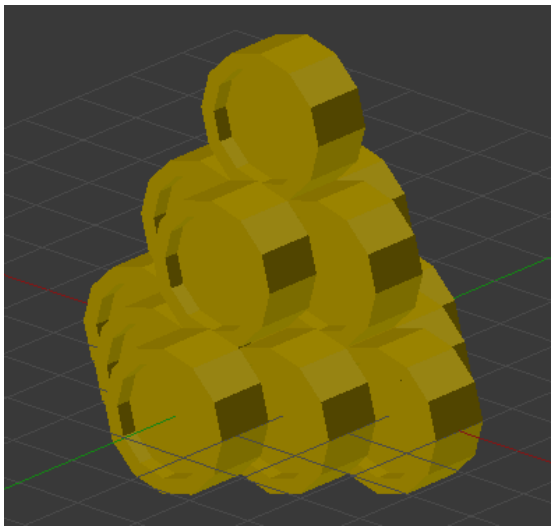
Vaenlaste loomisel muudetakse tema mudeli materjalide värve, et mängupilt liiga üksluine ei oleks. Selle jaoks tuleb kõigepealt mudelist materjalid Unity'sse importida. Seejärel luuakse listid juuste, särgi, pükste ja jalanõude jaoks ja lisatakse nendesse erinevaid värve. Iga kord kui uus vaenlane tekitatakse, valitakse igast listist suvaline väärtus ja nii ei näe kõik vaenlased samasugused välja.

Iga mängu alguses sünnivad vaenlased iga kolme sekundi tagant ja nende kiirus on kolm ühikut sekundis. Et mäng muutuks igal mängukorral järjest raskemaks hakatakse iga viie sekundi tagant vähendama aega kui tihti vaenlased sünnivad 0,2 sekundi võrra kuni see on 1 ja suurendama sündinud vaenlaste kiirust 0,5 ühiku võrra. Iga vaenlase sündimise vahele peab jääma vähemalt sekund, et vältida olukorda kus neid tekiks liiga palju. Vaenlase kiiruse suurendamisega välditakse olukorda kus mäng võiks kesta sisuliselt lõpmatuseni.

4.11 Vaenlaste tulistamine ja suremine

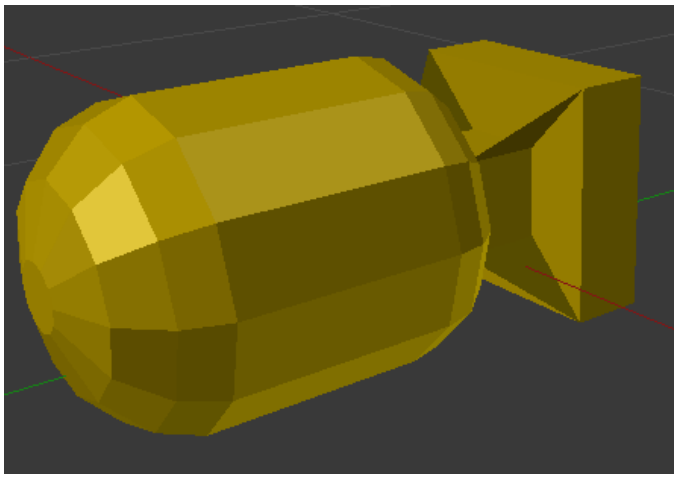
Nagu eelnevalt tulistamise kirjeldamises räägitud, kasutatakse kokkupõrgete tuvastamiseks *raycast*'i. Puutel tulistaval kihil oleva mänguobjektiga saame kontrollida, kas antud objektiga on seotud vaenlase skript. Kui on, tuleb selles skriptis välja kutsuda funktsioon, mis vähendab vaenlase elusid. Et vaenlane sureks on vaja teda tabada kaks korda. Vaenlase surma korral suurendatakse kõigepealt mängija skoori ühe võrra.

Vaenlase mänguobjektiga on seotud neli varem loodud *prefab*'i. Vaenlase surma korral on 75% tõenäosus, et vaenlase asemel tekivad mündid, mida mängija saab üles korjata neist üle kõndides ja hiljem kasutada. Üles korjatud mündid suurendavad mängija müntide arvu saja võrra. Müntide mudel on näha joonisel 22.



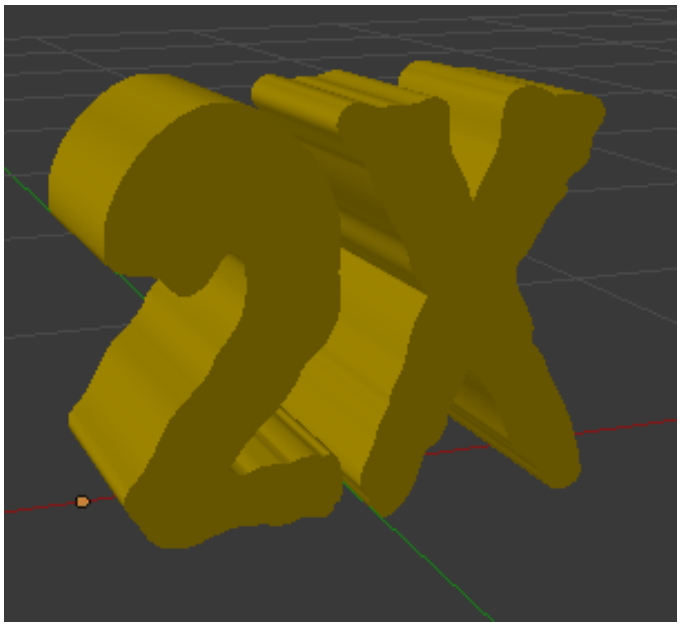
Joonis 26. Müntide mudel.

Kui seda ei juhtu on 25% tõenäosus, et vaenlase asemele tekib kas pomm, topelt punktid või kirst, mida mängija saab samuti ülesse korjata. Kui mängija korjab ülesse pommi, surevad kõik vaenlased kes mängutasemel sellel hetkel on. See on aga vaenlase surma erijuht ja kui vaenlased nii surevad, siis võib nende asemel tekkida ainult münte. Pommi mudel on näha joonisel 23.



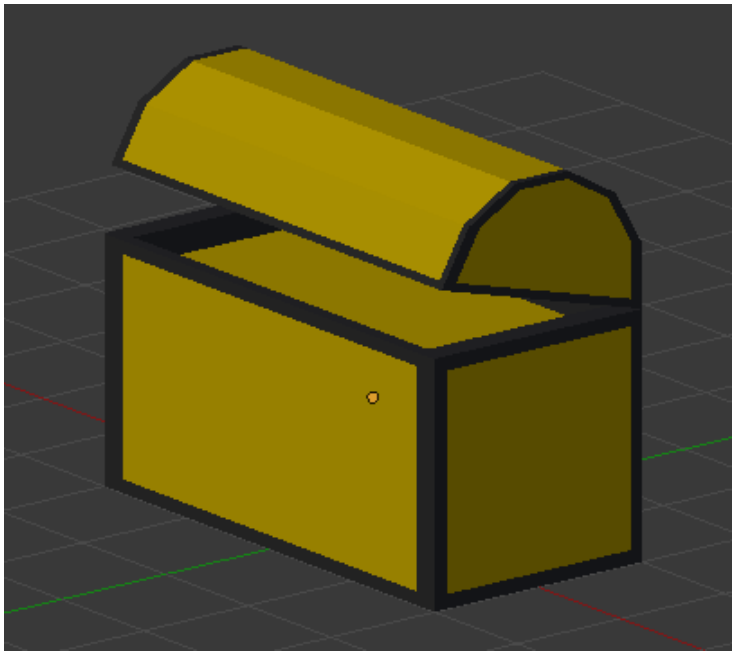
Joonis 27. Pommi mudel.

Kui mängija korjab ülesse topelt punktid, siis kuvatakse ekraanile taimer ja kõik mündid mis selle taimeri aktiivsuse ajal ülesse korjatakse suurendavad mängija kogu müntide arvu 200 võrra. Topelt punktide taimer kestab 30 sekundit. Selle mudeli ja ka kõikide teiste tekstide jaoks kasutatakse fonti „*Creepster*“ [10]. Topelt punktide mudel on näha joonisel 24.



Joonis 28. Topelt punktide mudel.

Kui mängija korjab ülesse kirstu suurendatakse tema müntide arvu 2500 võrra. Juhul kui samal ajal on aktiivne ka topelt punktid, suurendatakse müntide arvu 5000 võrra. Kirstu mudel on näha joonisel 25.

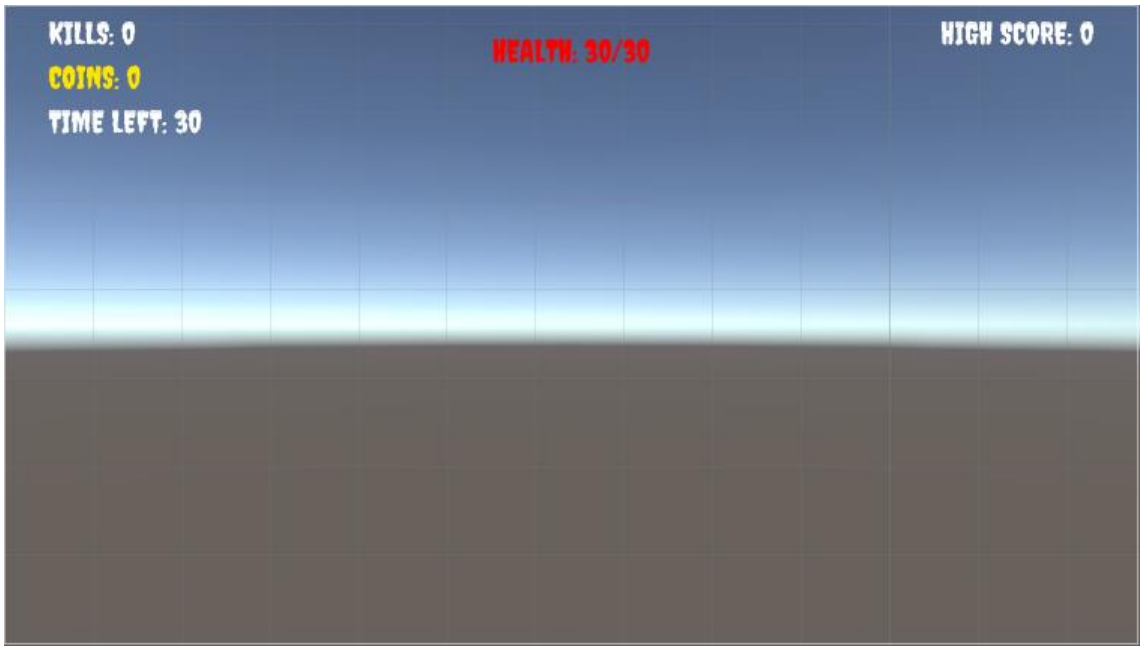


Joonis 29. Kirstu mudel.

Kõik need mänguobjektid, mis tekkivad peale vaenlase surma kaovad kui mängija neid üles ei korja. Kümme sekundit peale nende mänguobjektide tekkimist hakkavad need vilkuma, et mängija aru saaks, et need varsti kaovad ja peale 15 sekundit need kaovad. Samuti on pandud kõik antud objektid keerlema.

4.12 HUD

HUD on osa kasutajaliidesest, mille abil kuvatakse kasutajale olulist informatsiooni. Erineb ülejäänud kasutajaliidesest selle poolest, et sellega ei saa kasutaja mängu kontrollida. Sellel põhjusel on *HUD* ja *UI* jagatud kahele erinevale lõuendile ja *UI* toodud nendest kahest ette poole, et vältida olukordi kus kasutaja tahab nupule vajutada, aga kuna mõni teine element on ees siis vajutust ei registreerita. *HUD*'il on alati neli olulist elementi, milleks on käesoleva mängu skoor, mängija müntide arv, mängija elud ja kõrgeim skoor. Kui on aktiivne topelt punktid, siis kuvatakse ka selle taimer (vt. Joonis 26).



Joonis 30. *HUD* ja sellel kuvatavad elemendid.

4.13 Helid

Mängus kasutatakse kokku viit tüüpi heli. Relvaga seotud heli on püssilask [11], mida mängitakse iga kord kui tulistatakse. Mängijaga seotud helisid on kaks. Esimene on müntide kõlin [12], mida mängitakse kui korjatakse ülesse mündid või kirst ja mängija müntide arv suureneb, Teine on pommi plahvatus [13], mida mängitakse kui mängija korjab ülesse pommi. Vaenlasega seotud heli on rusika löök [14], mida mängitakse kui vaenlane mängijat ründab. Viimane heli on vaenlase surma jaoks [15], mis on kaudselt vaenlasega seotud, aga Unity's tuleb ta siduda mänguobjektiga, mida kasutatakse vaenlaste tekitamiseks sellel põhjusel, et samal hetkel kui seda heli mängida tuleks, tuleb

hävitada vastava vaenlase mänguobjekt. Helide mängimiseks kasutatakse komponenti *audiosource*.

4.14 Mängu lõppemine

Mäng lõppeb kui mängijal elud otsa saavad. Mängu lõppedes kuvatakse kasutajale stseen, kus on kirjas tema viimase mängu skoor ja teenitud müntide arv. Seejärel pakutakse kasutajale teenitud müntide arv kahekordistada vaadates reklaami. Reklaamide implementeerimise teeb lihtsaks Unity enda teenus nimega *Unity Ads*. Mängu lõpu stseen on näha joonisel 8.

Mõlemale nupule vajutades on lõpp tulemuseks see, et mängija kuvatakse jälle ava menüü stseen. Kui mängija otsustab aga reklaami vaadata näidatakse talle umbes 30 sekundi pikkust reklaami, mida ei saa kinni panna ning seejärel kahekordistatakse viimases mängus teenitud müntide arv.

4.15 Ava menüü

Ava menüü stseen koosneb kahest valikust. *Play* nupp alustab mängu ja kasutajale kuvatakse mängu stseen. *Upgrade* nupp viib kasutaja uuenduste menüüsse. Samuti kuvatakse mängijale tema kõrgeim skoor. Ava menüü stseen on näha joonisel 6.

4.16 Mängija uuendamine

Et mäng liiga kiiresti ära ei tüütaks ja teenitud müntidel mingi eesmärk oleks, on loodud kasutajale võimalus oma mängijat uuendata. Kasutaja saab suurendada mängija kiirust ja elude arvu. Nende ja ka mõnede teiste muutujate, mida on vaja erinevate stseenide vahel kasutada salvestamiseks kasutatakse *playerprefs*'e. Need on muutujad, mis salvestuvad ka juhul kui mängu näiteks uuendatakse. Alguses on mängija kiirus kolm ühikut sekundis ja elude arv 30. Esimene uuendamine maksab 1000 münti. Iga järgnev uuendamine suurendab hinda 1000 võrra. Uuendamisel suurendatakse kiirust alati ühe võrra ja elude arvu alati kümne võrra. Paremal üleval servas olev nupp *back* viib kasutaja tagasi ava menüüsse. Uuenduste stseen on näha joonisel 9.

5 Kokkuvõte

Käesoleva bakalaureusetöö eesmärgiks oli luua kolmemõõtmeline tulistamis-ellujäämismäng, Android operatsioonisüsteemile kasutades Unity mängumootorit.

Töös kirjeldati paljusid mobiilimängu loomise aspekte ja anti ülevaade, kuidas neid Unity mängumootorit kasutades saavutada. Unity mängumootorist oli väga palju kasu kuna Unity võimaldas väga lihtsalt lahendada muidu keerulisi probleeme. Tänu Unity mängumootorile sujus mängu enda arendus mitte liiga ajakujukalt. Paljud keerulised asjad, nagu näiteks vaenlaste liigutamine mängija tegelase poole, mille koodimine oleks võinud olla väga keeruline, sai tänu Unity navigeerimis süsteemile lahendatud väga kiiresti. Kõige raskemaks ja ajakulukamaks osutusid modelleerimised, eriti just tegelaste modelleerimised.

Töö täitis oma eesmärki, kuna töö tulemusena valmis Android operatsioonisüsteemil töötav mäng. Kasutajal on võimalik navigeerida erinevate vaadete vahel, mängida mängu, milles kasutaja eesmärk on võimalikult palju vaenlasi tappa samal ajal nende eest ära joostes ning koguda münte, millega oma tegelast uuendada.

Kasutatud kirjandus

[1] J. McKane, “https://mybroadband.co.za/news/software/258733-the-most-popular-operating-systems-for-desktop-and-mobile.html” [Võrgumaterjal]

[2] A. Beck, https://www.quora.com/How-much-does-it-cost-to-publish-a-mobile-app” [Võrgumaterjal]

[3] “http://mediakix.com/2018/03/mobile-gaming-industry-statistics-market-revenue/#gs.XXVkw=w” [Võrgumaterjal]

[4] Unity, “https://unity3d.com/unity” [Võrgumaterjal]

[5] JetBrains Rider, “https://www.jetbrains.com/rider/” [Võrgumaterjal]

[6] Blender, “https://www.blender.org/about/” [Võrgumaterjal]

[7] Survival Shooter tutorial, “https://unity3d.com/learn/tutorials/s/survival-shooter-tutorial” [Võrgumaterjal]

[8] JoyStick Pack, “https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631” [Võrgumaterjal]

[9] “https://80.lv/articles/7-tips-for-better-lighting-in-unity/” [Võrgumaterjal]

[10] Creepster, “https://www.1001fonts.com/creepster-font.html#license” [Võrgumaterjal]

[11] Gun shot,“https://freesound.org/people/schots/sounds/382735/” [Võrgumaterjal]

[12] Coin Drop, “https://freesound.org/people/jon.k.clancy/sounds/402935/” [Võrgumaterjal]

[13] Explosion_001, “https://freesound.org/people/cydon/sounds/268557/” [Võrgumaterjal]

[14] Fist punch or kick, “https://freesound.org/people/rcroller/sounds/424144/” [Võrgumaterjal]

[15] Grunt2 – Death Pain, “https://freesound.org/people/AlineAudio/sounds/416838/” [Võrgumaterjal]