TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karl Kevin Klais
154908IAPB

LÕPMATULT GENEREERIVA MAAILMAGA IOS RAKENDUSE LOOMINE MÄNGUMOOTORIS UNITY3D

Bakalaureusetöö

Juhendaja: Jaagup Irve
MSc

Tallinn 2018

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: Karl Kevin Klais

19.05.18

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on luua töötav mobiilirakendus „Dead Sprint“ ning näidata, kuidas erinevaid loodavaid loogikaid on võimalik optimeerida rakenduse näitel.

Töös antakse ülevaade mängude peamistest aspektidest. Töös kirjeldatakse rakenduse loomiseks mõeldud töövahendeid. Töövahenditega on arendatud rakendus iOS platvormile ning kirjeldatud mänguloogikate loomine.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 28 leheküljel, 7 peatükki, 29 joonist, 0 tabelit.

Abstract

Infinitely generating world application development with Unity3D game engine for iOS platform

The aim of this thesis is to create working mobile application called “Dead Sprint” and to also demonstrate how the various game logic can be optimized as an example of an application.

The work gives an overview of the main aspects of virtual games. In this thesis are described the tools for creating the application, which are used to develop mobile application for iOS platform. Game logics which relate to mobile application are specified in this work.

The thesis is in Estonian and contains 28 pages of text, 7 chapters, 29 figures, 0 tables.

Lühendite ja mõistete sõnastik

iOS	Iphone Operating System
macOS	Macintosh Operating System
watchOS	Watch Operating System
tvOS	Television Operating System

Sisukord

Jooniste loetelu

1 Sissejuhatus

Bakalaureusetöös antakse ülevaade arvutimängude olemusest. Töös tutvustatakse mänguloogika loomise võimalusi mängumootoris Unity. Tutvustatakse erinevate loodud rakenduste stiile. Töös luuakse mängijat ümbritseva maailma generaator, liikumisekontroller, kokkupõrgete kontroller ja vastase tehisintellekt. Erinevate loogikate loomise tulemuse tervikuna valmib rakendus „Dead Sprint“, kus mängija eesmärk on joosta kooljate eest nii kaua kui võimalik. Töös tuuakse välja peamised arenduse käigus tekkinud probleemid ning nende lahendused.

Kõike, mida on võimalik programmeerida, on ka võimalik optimeerida. Optimeerimisega saavutatakse üldiselt kiirem tööaeg või lihtsustakse probleemi lahendamise keerukust. Töös on kirjeldatud optimeerimisevõimalusi rakenduse näitel.

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

2 Taust

2.1 Arvutimäng

Arvutimäng on süsteem programmeeritud reeglitest, kus tegevus toimub virtuaalmaailmas. Üldiselt on mängud kas harivad või meelelahutuslikud. Eksisteerivad erinevat sorti mängud. Näiteks on mängukategooriateks võitlus-, seiklus-, simulatsiooni, spordimängud, MMO, MMORPG, MOBA – online mitme mängijaga mängud. [9] [12]

2.2 Virtuaalmaailm

Virtuaalmaailm on simulatsioon keskkonnast, mida selle elanikud peavad iseseisvaks. See on koht, kus kujutletav saab reaalseks. Inimene, kellel on juurdepääs virtuaalmaailma on mängija. Mängija omab kontrolli maailmas paikneva(te) isiku(te) üle, keda nimetatakse karakteri(te)ks. Virtuaalmaailmas toimuva tegevuse mõjutamist mõistetakse mängimise all. [10]

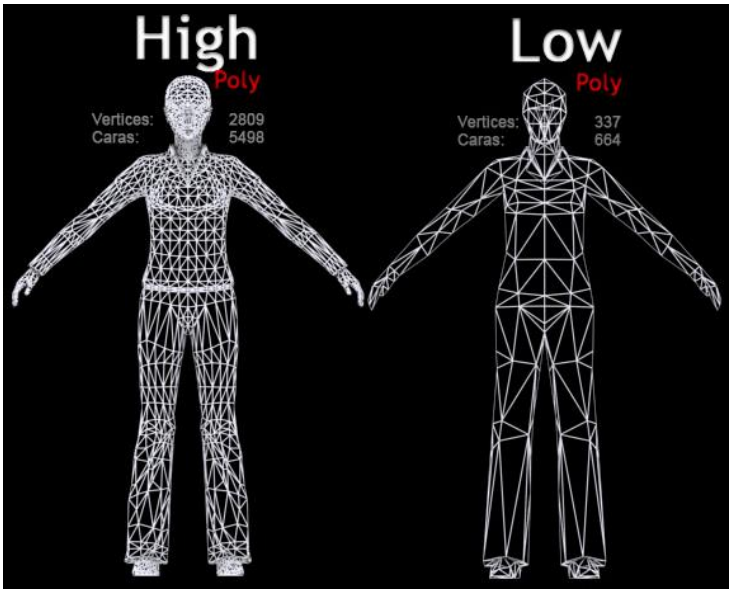
Üldiselt liigub mäng mööda kindlat ajakava- varem või hiljem jõutakse järgmisesse kontrollpunkti, mis asub loodud maastikul. Keskkond, mis mängijat ümbritseb, on tavaliselt paika pandud ja näiteks ümbritsetud kõrgete mägedega, mida ei saa ületada, veega milles ei saa ujuda või teiste piiravate detailidega.

2.3 Arvutimängude graafika stiil

Arvutigraafika stiile on mitmeid, inimeste jaoks kõige suurem erinevus on 2D ja 3D vahel, kuid neid on võimalik täpsemalt defineerida. Näiteks on kunstistiilideks järgnevad: abstraktne, minimalistlik, kontseptuaalkunst, piksli kunst, *sprite* graafika, *low poly* ja *high poly* graafika.

Loodud rakenduses kasutatakse *low poly* graafikat, mille põhiidee järgi luuakse objekt nii lihtsalt kui võimalik. Modelleerimisel detailide väljatoomisele rõhku ei panda ning võrreldes *high poly* graafikaga on hulknurkade arv kordades väiksem, üldiselt

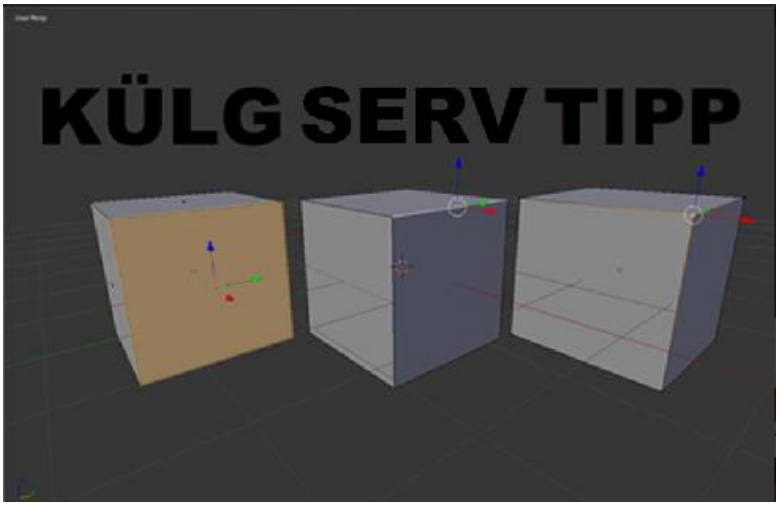
eelistatakse kasutada kolmnurki, kuid võib kasutada ka suuremaid hulknurki (vt. Joonis 1).



Joonis 1. *Low poly* ja *high poly* graafika võrdlus. [19]

2.3.1 Graafikaelementide baasvõrk

Hulknurkade tippude asetusest moodustub objekti baasvõrk. Graafikadisaini loomisel redigeeritakse hulknurga külgi, servi või tippe (vt. Joonis 2). [13]



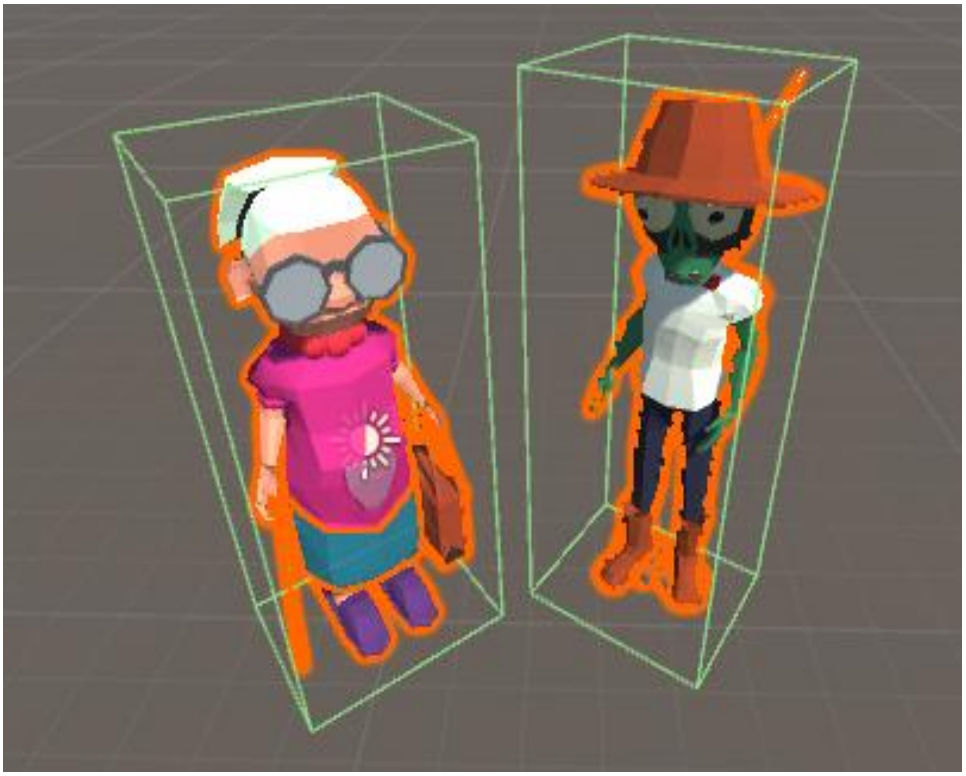
Joonis 2. Objekti võrgustiku redigeerimine programmis Blender.

2.3.2 Graafika animatsioonid

Animatsioon on karakteri positsiooni ja orientatsiooni muutmine sõltuvalt sisendist, näiteks võib animatsiooniks olla jooksmine või hüppamine. Karakteri animatsioon koosneb üldiselt erinevatest disaineri loodud poosidest. Interpolatsiooni abil luuakse poosidest lõplik animatsioon. [11]

2.4 Kokkupõrgete registreerimine

Kokkupõrgete registreerimisega määratakse kindlaks kas, kus ja millal kaks või rohkem objekti ristuvad. Kokkupõrked objektide vahel on peamine virtuaalmaailma mõjutamise viis. Näiteks on kokkupõrgete registreerimine oluline maailmas, kus karakter asub mööblit täis toas ning karakteri liikumise korral ei tohi olla lubatud liikumine mööbli või seina sisse. Üldiselt ümbritsetakse kokkupõrke registreerimiseks karakter ja objektid tihedate piiridega ning kontrollitakse nende piiride kattuvust, kuid triviaalsem on keerulise disaini korral ümbritseda objekt väikseima võimaliku, kuid kõiki külgi katva hulknurgaga. See tähendab, et põrkepiirid ei kattu objekti kujuga. (vt. Joonis 3) [11]



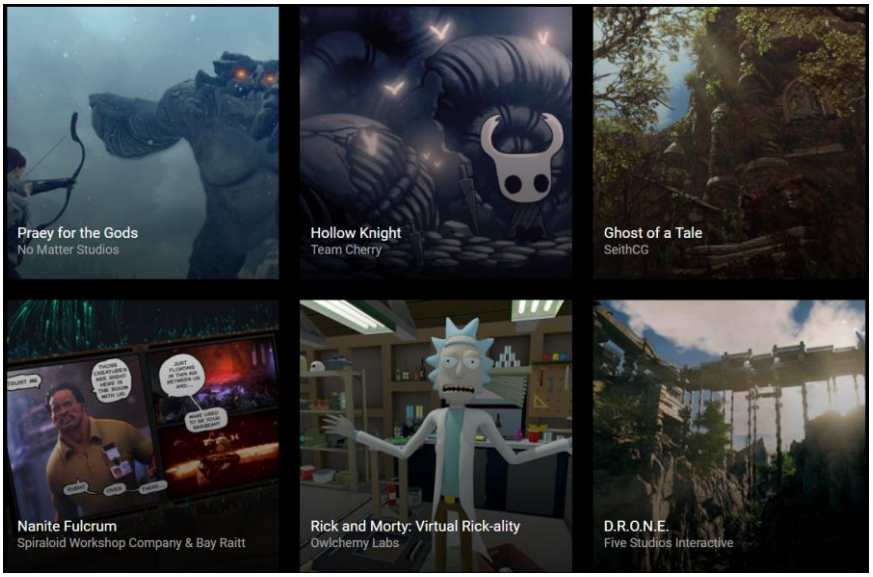
Joonis 3. Objekti ümbritsev hulknurk mis ei kattu objekti kujuga.

3 Mobiilirakenduse loomise vahendid

Rakenduse loomise protsess ei sõltu vaid programmeeritud koodi pikkusest. Järgnevalt on kirjeldatud vahendid, mida kasutatakse erinevate arenduseetappide korral.

3.1 Unity3D

Unity on mängumootor, mida kasutakse „Dead Sprint“ rakenduse loomiseks. Unity on tasuta allalaetav, kuid eksisteerib ka Unity Pro, mis annab lisafunktsioone tasu eest. Unity toetab erinevate programmide loodud objekte, sealjuures ka eelnevalt kirjeldatud Blenderit. Unity toetab Boo, C# ja JavaScript programmeerimiskeeli. Unity mootorit on võimalik kasutada eri stiili mängude loomiseks. (vt. Joonis 4) [2]



Joonis 4. Erinevat stiili mängud, mis on loodud Unity-s.

3.1.1 Peamised Unity3D funktsionaalsused

Unity toetab nii kahedimensiooniliste kui kolmedimensiooniliste rakenduste funktsionaalsuse arendust. Lisaks on võimalik arendada rakendusele kasutajaliidesed. [16]

Mängumootoris on näiteks sisseehitatud füüsikaseadused, mittemängija tehisintellekti navigatsioonisüsteem ja ligipääs *Asset Store* keskkonnale, kus on Unity

kasutajaskonna üleslaetud tööriistad, materjalid või muud elemendid, mida arendaja saab kasutada enda projektis.

Unity's on erinevad kaameraga seonduvad tööriistad, mis annavad võimaluse luua filmikaadreid nagu režissöör, erinevaid kaadrite järjestusi, värviredigeerimise ja efektide võimalus järeltöötluses.

Võimalus animeerida nii Unity's loodud objektide, kui ka erinevate kolmandate osapoolte loodud objektide. Võimalus ehitada virtuaalmaailma ja erinevaid tasemeid.

Arendus erinevatele seadmetele on tehtud väga mugavaks, sest Unity toetab 27 erinevat platvormi. Näiteks iOS-i, Android-i, Playstation-it ja paljusid muid platvorme. Lisaks on võimalik arendada rakendusi virtuaalreaalsuse ja liitreaalsuse jaoks ning luua rakendusi, mis internetipõhiselt annavad võimaluse mitmemängijaga mängudeks.

Unity annab võimaluse hoida projekti pilves. Tänu millele on meeskonnal lihtsam uuendada projekti ressursse ning sünkroniseerida loodud uuendused teistega.

Unity'ga loodavast rakendusest tulusaamisvõimalus on kasutada rakendusesiseseid reklaame, mille integreerimiseks saab kasutada olemasolevat liidest. Lisaks on liides tulu teenimiseks läbi rakendusesiseste ostude.

3.2 MonoDevelop

MonoDevelop on arenduskeskkond, mis on vaikimisi Unity skriptide redaktor. Lisaks JavaScriptile ja C#, mida toetab Unity, toetab redaktor veel Boo, C, C++, CIL, D, F#, Vala, Java, Oxygene ja .NET raamistikke. [15]

3.3 Blender

Blender on vabatarkvara, mille funktsionaalsust kasutab autor 3D-graafika loomiseks ja animeerimiseks, kuid sellega ei piirdu programmi võimalused. [1]

3.3.1 Peamised Blenderi funktsionaalsused

3.3.1.1 Modelleerimine

Blenderit kasutatakse ilmselt kõige rohkem modelleerimiseks. Lisaks objektide baasvõrgu redigeerimisele, pakub Blender sisseehitatud modifikaatoreid, mille käsitsi loomine oleks väga ajamahukas (vt. Joonis 5). [14]

Joonis 5. Blenderi sisseehitatud modifikaatorid.

Näiteks on väga oluline modifikaator peegel, tänu millele saab disainer luua vaid poole objektist. Juhul kui objekt on valitud telje suhtes identne, siis genereeritakse teine pool automaatselt vastavalt X, Y või Z telje suhtes.

3.3.1.2 Animeerimine

Eelnevalt kirjeldatud modelleerimisega loodud objekti on võimalik Blenderis animeerida. Blender pakub kiiret mudeli muutmist poseeritavaks tegelaseks ning loodud pooside interpoleerimist. [14]

3.3.1.3 Mängu loomine

Blender pakub võimalust luua mänguloogikat kasutades disainitud objekte. Sisseehitatud on Python programmeerimiskeele toetus, baasfüüsika integreeritus, 3D heli ja mängu esitamine ilma kompileerimise ja eeltöötlemiseta. [14]

3.4 xCode

xCode on integreeritud arenduskeskkond, mis on mõeldud ainult Apple seadmetele ning mis on vajalik iOS rakenduse publitseerimiseks. Antud platvormile suunatud rakenduste publitseerimine ei ole võimalik teiste seadmete pealt. [3]

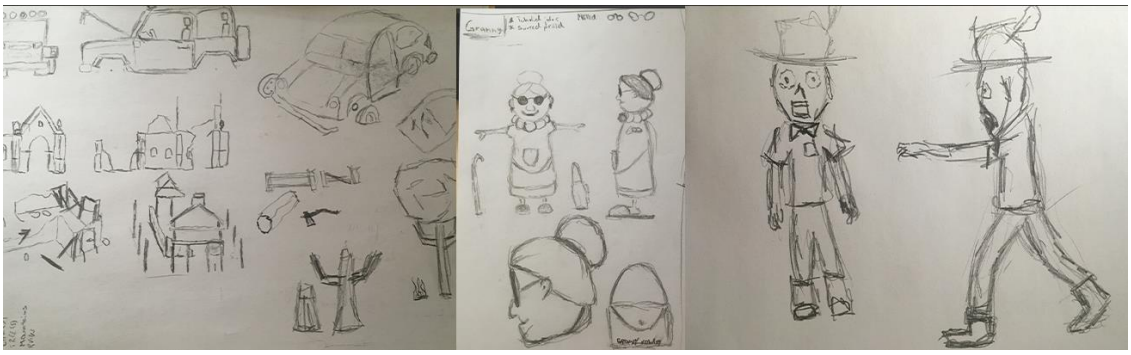
4 Mobiilirakenduse arendamine

4.1 Rakenduse loomise protsess

Järgnevalt kirjeldatakse „Dead Sprint“ loodud rakendust alates karakterite disainist kuni mänguloogika loomiseni.

4.1.1 Objektide disain

Iga elemendi disain algab kontseptsioonist paberil, millest luuakse graafikaloomiseprogrammis lõplik kujutis (vt. Joonis 6). [17]



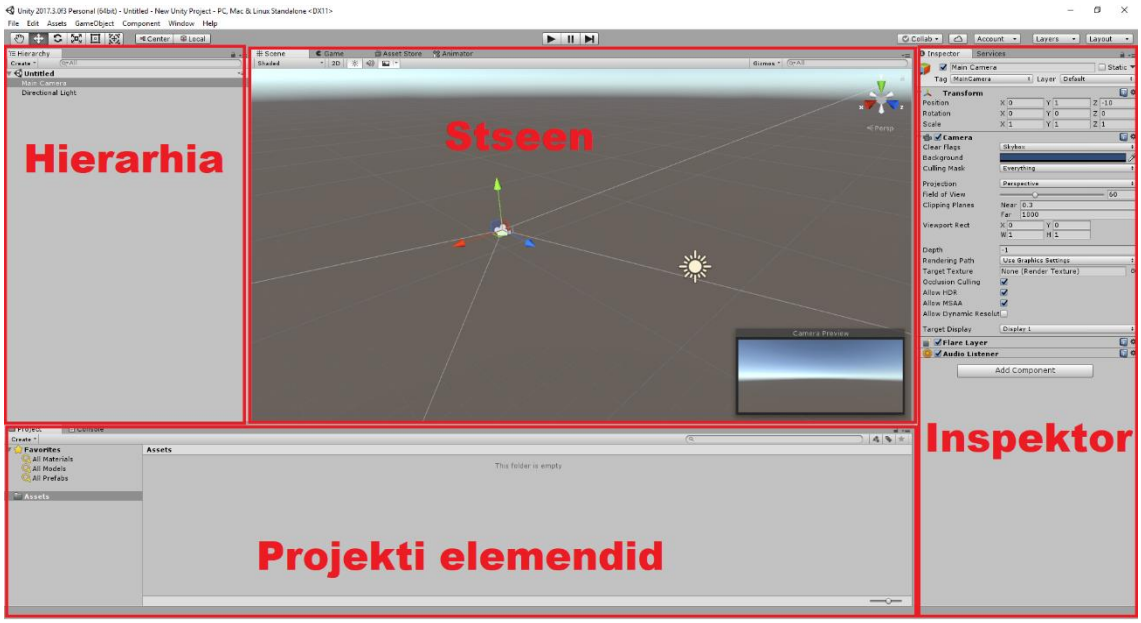
Joonis 6. „Dead sprint“ peamised objektid paberil.

Antud rakenduse korral luuakse objektid programmis Blender. Objektide stiil on *low poly* ning lõplik disain eksporditakse mängumootorisse Unity3D, kus luuakse mänguloogika (vt. Joonis 7).

Joonis 7. Eksportimisvalmis karakterid.

4.1.2 Uue projekti loomine programmis Unity

Unity käivitamisel tuleb luua uus projekt ning määrata projekti nimi, mängu graafika stiil, projekti asukoht ning mängu looja organisatsioon. Peale esimesi seadeid on kasutajal ees alati algpaigutus programmi tööpaneelidega (vt. Joonis 8).



Joonis 8. Unity käivitusekraan.

4.1.3 Unity tööpaneelid

Stseen: saates on näha, millised elemendid mängus asetsevad ning kuidas üksteise suhtes paiknevad. Mängu loomine toimub antud vaates. Vaadet on võimalik muuta ülevalt vasakult servast mängu vaateks, mida näeb kaamera mängus. Mängu saab testida vajutades käivitamisnuppu stseeni vaate kohal, kuivõrd käivitatakse mäng samuti vaates, mida näeb kaamera ning peale käivitamist objektide muutmisel efekti pole.

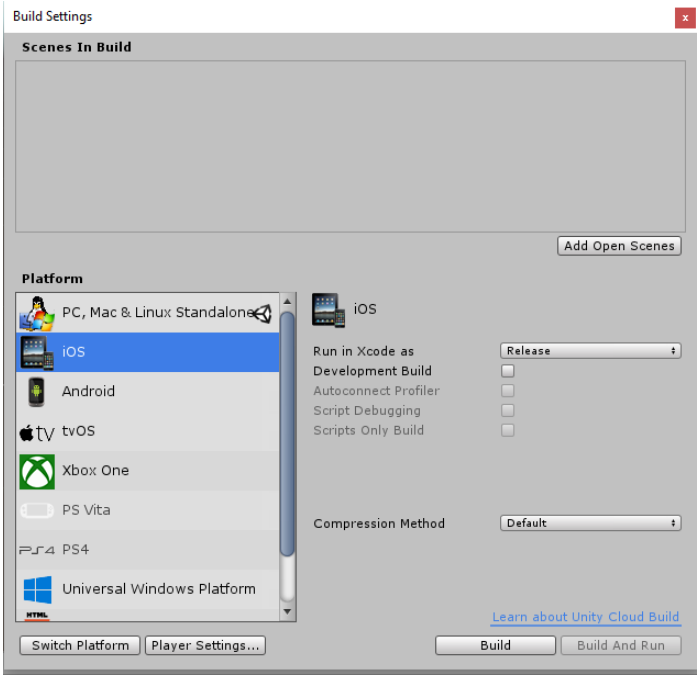
Hierarhia: sisaldab listi kõigis stseenis paiknevatest elementidest. Uue projekti korral on vaikimisi hierarhias põhikaamera ning valgustus.

Inspektor: sisaldab aktiivse objekti omadusi, mida on võimalik antud paneelist muuta.

Projekti elemendid: paneelil paiknevad kõik projektis kasutusel olevad elemendid, mitte ainult konkreetse stseeni elemendid.

4.1.4 Unity platvormi valik

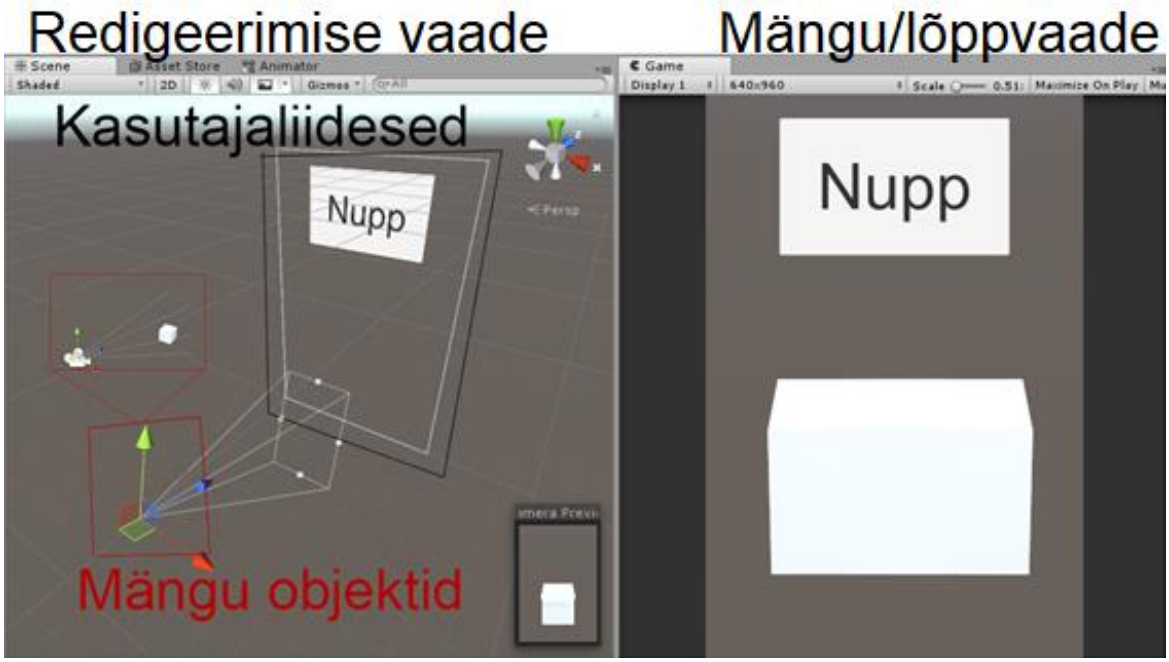
Erinevatele platvormidele mängu loomiseks tuleb luua iga platvormi kohta eraldi rakenduse variant. Platvormi muutmiseks tuleb liikuda „File → Build settings" ning avanevast aknast valida eelistatav platvorm ning kinnitada nupust „Switch Platvorm" (vt. Joonis 9).



Joonis 9. Unity loodava rakenduse platvormi muutmine.

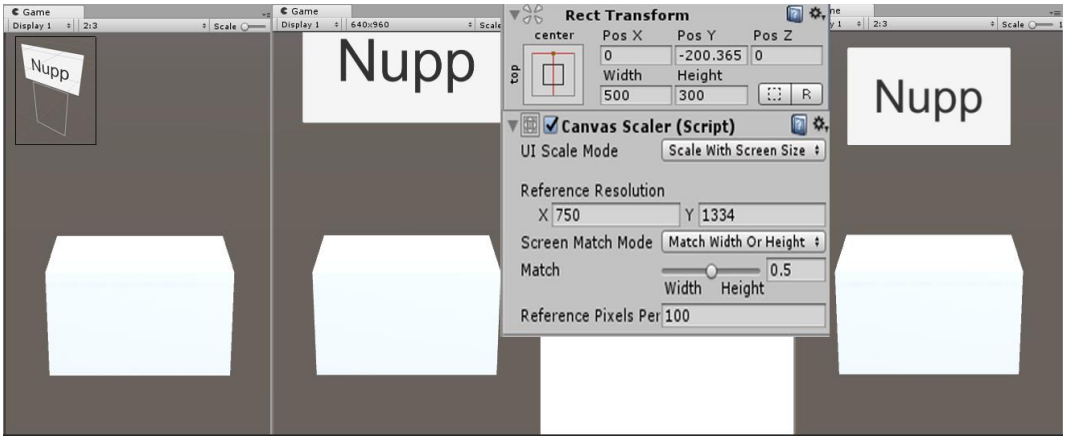
4.1.5 Rakenduse kaamerad

Uue projekti loomise järel on mõistlik paika panna peamine vaade, kuidas mängija mängu näeb. Siinkohal tuleb eristada kahte kaamera tüüpi. Esiteks kaamera, mis näitab mängijale kasutajaliidese elemente, milleks on kõikvõimalikud nupud, tekstid, tekstisisestusväljad ja muud elemendid. Teiseks kaamera, milles reaalselt paikneb mängija, vastased ja ülejäänud mänguga seonduvad objektid. Joonisel 10 on näha poole ekraani suurust kuupi ning poole ekraani suurust nuppu, mis on redigeerimisel täiesti erinevad, kuid lõppvaates tunduvad kõrvuti ning sarnaste mõõtudega.



Joonis 10. Unity erinevad kaamerad.

Vaikimisi üritavad kaamerad säilitada enda suurust ükskõik mis suurusega ekraani korral. Suuruse säilitamine viib selleni, et ühe ekraani korral võivad loodud elemendid paikneda perfektselt, kuid väiksema ekraani korral paiknevad elemendid väljaspool ekraani. Vältimiseks tuleb anda kaameratele viide, mille järgi paiknevus seatakse. Rakenduses „Dead Sprint“ on viiteks iPhone 8 ekraani suurus [4]. Joonisel 11 on näha objektide paiknevus ekraanil suurusega 640x960 ja kuvasuhtega 2x3, enne viite loomist ja pärast. Lisaks on mõistlik määrata erinevate elementide ankurpunktid, mille järgi asukoht seatakse. Näiteks pildil oleva nupu ankur oleks ekraani ülemise serva keskkoht (vt. Joonis 11).



Joonis 11. Kaamerate suuruse viite loomine.

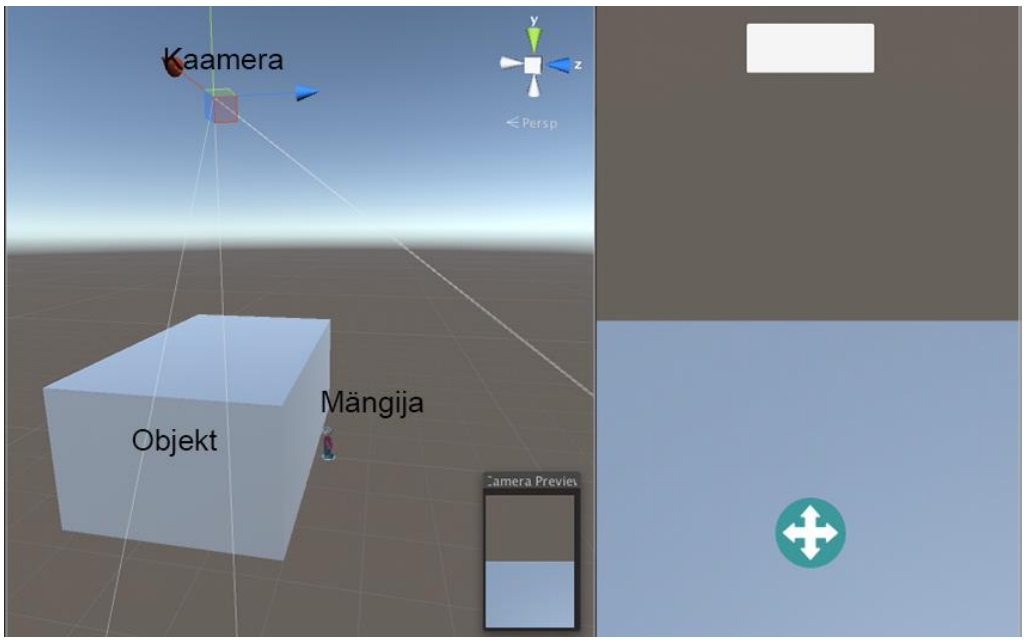
4.1.6 Tegelase liigutamine maailmas

Blenderiga loodud karakteri liigutamise võimalusi Unity's on mitmeid. Sõltuvalt seadmest, saab tegelast liigutada arvutiklaviatuuri, -hiirega, telefoniekraani puudutusega ja teiste kontrolleritega. Käesolevas mängus luuakse juhtimiseks ekraanile juhtkangi kujutis, mille horisontaalse või vertikaalse liigutamise korral liigutatakse ka tegelast.

4.1.7 Kaamera sidumine tegelasega

Karakteri liigutamise loogika loomise järel seotakse kaamera karakteri objektiga, mis on lihtsaim viis tegelase jälitamiseks. Rakenduses on loodud loogika, kus kaamera liigub objekti järel viivitusega, et vältida hüplemist näiteks juhul, kui mängija otsustab muuta liikumissuunda positiivsest negatiivseks.

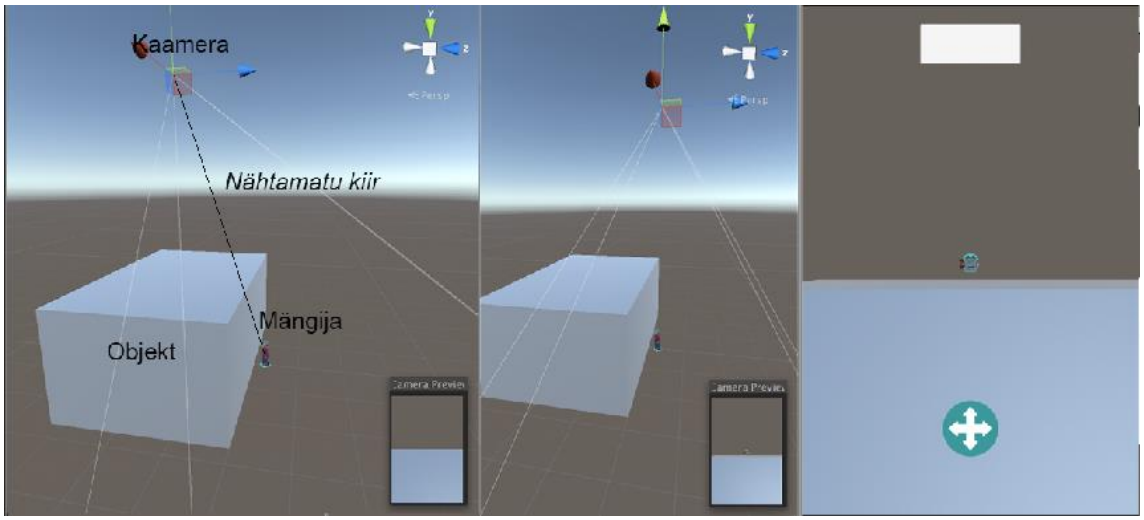
Rakenduse kaamera paikneb pealtvaates kontrollitava selja taga. Kaamera ei paikne tegelase Y telje peal, mis tähendab, et tegelane ei ole enam nähtav, kui tema ja kaamera vahel on mingi objekt. Objektiks võib olla mängukontekstis kirik, tempel, auto, puu või maja varemed (vt. Joonis 12).



Joonis 12. Kaamera vaatevälja blokeeriv objekt.

Mängus peab olema sellistes olukordades kindlasti otsene vaade tegelasele, sest pimeda nurga taga võib olla ka vastane, kelle mittenägemine antud olukorras ei tohi olla lubatud. Pimeda nurga vältimiseks kasutatakse rakenduses *Raycast* meetodit, mille

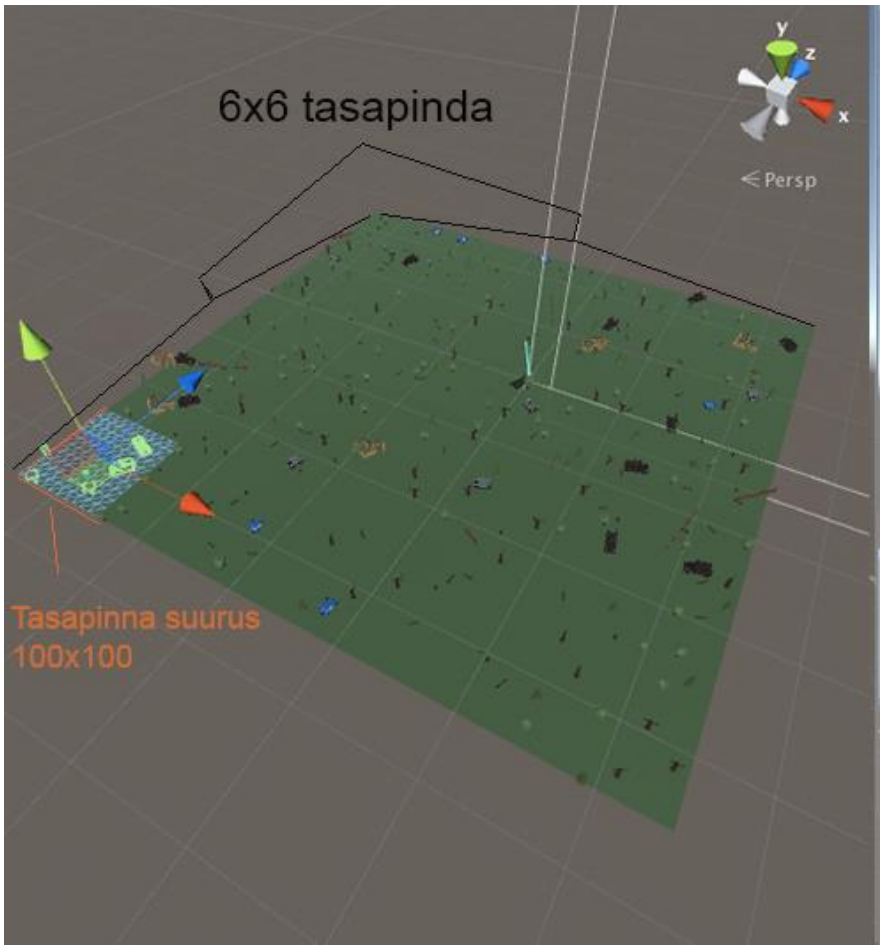
kohaselt luuakse nähtamatu kiir sihtmärgi ja valitud lähtekoha vahel. [5] Juhul, kui kiir põrkab kokku „objekt“ tüüpi esemetega, muudab kaamera kaldenurka ning oma asukohta (vt. Joonis 13).



Joonis 13. *Raycast* ja kaamera asukoha muutmine.

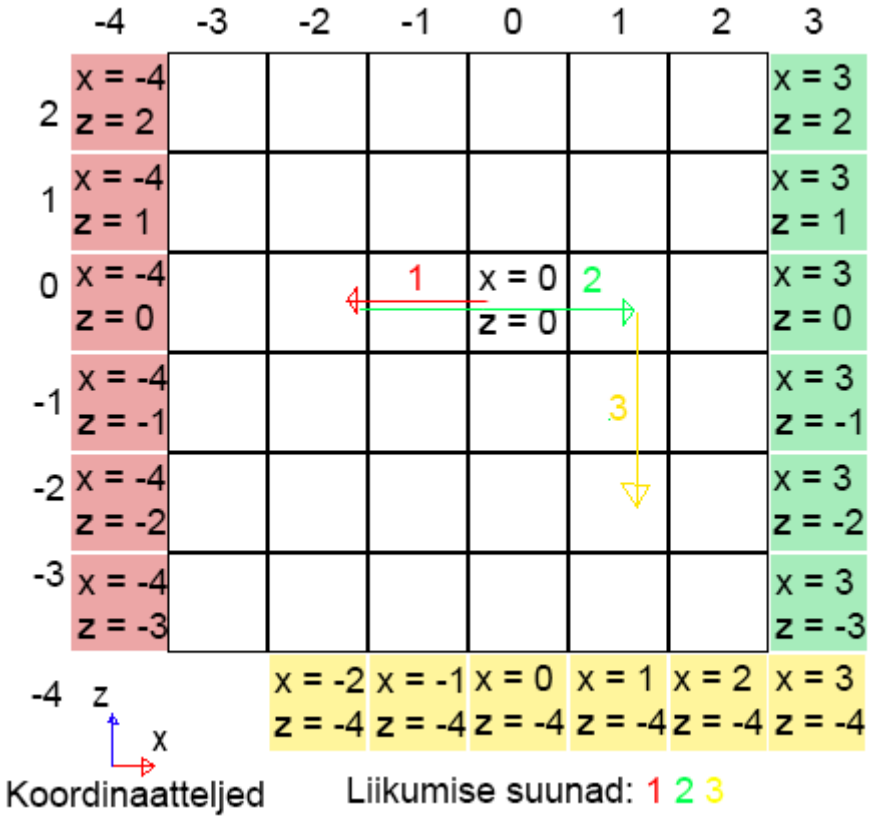
4.1.8 Maailma genereerimine

Rakenduses luuakse keskkond igal sessioonil erinevalt, välja arvatud koordinaatidel (0; 0; 0) paikneva tasapinna objektid, mis on alguspositsiooniks igas mängus. Piiramatust liikumisest tingituna tuleks genereerida lõpmatu maailm, mida ei ole võimalik töödelda. Lõpmatu maailma genereerimise vältimiseks genereeritakse vaid osa maailmast, millel paiknevad esemed on erinevas kohas iga käivitamise korral. Genereeritud maailm koosneb 6x6 tasapindadest ning iga tasapind on suurusega 100x100 (vt. Joonis 14).



Joonis 14. Esimene genereeritud osa maailmast.

Edasine maailma genereerimine sõltub mängija koordinaatidest. Juhul kui mängija jõuab „ääre“ peale, kus puudub edasine genereeritud maailm, siis lisatakse puuduvatele positsioonidele tasapinnad. Näiteks liikudes suunas üks, joonisel 15, siis esimene tasapinna genereerimine toimub kohtadele (x-4, z-3; x-4, z-2; … ; x-4, z+2). Liikudes suunas kaks– (x+3,z+2; x+3,z+1; … x+3, z-3). Suunas kolm– (x-2, z-4; … x+3, z-4). Genereerimine toimub nii kaua, kuni mängija kooljate poolt kinni püütakse.



Joonis 15. Maailma genereerimine vastavalt liikumisele.

4.1.9 Objektide asukoha genereerimine maailmas

Iga loodud aluspind maailmas loob enda peale objektid, mis on seotud vaid tema ja suurte objektide korral tema kõrval asetsevate pindadega, mitte terve loodud maailmaga. Objektide loomine on juhuslik, samuti tema paigutus genereeritud punktist suurima objekti raadiuse ulatuses. Objektide lõplikust paigutusest tekib maailm, kui tervik.

Objekti loomise tõenäosus on 2:3, iga 10 ühiku tagant. Loodud objekt võib olla kas puu, auto, aed, maja, kirik või tempel, vastavate tõenäosustega 29:40, 1:8, 1:40, 1:40, 1:100 ja 1:100. Kirikut ja templit võib mängus olla ainult üks ning aia pikkus võib olla kuni 10 aeda (vt. Joonis 16).

Puu	Auto	Aed	Maja	Kirik	Tempel
Suurus 10 x 10	Suurus 10 x 20	Suurus 10 x 2 - 100 x2	Suurus 20 x20	Suurus 80 x 100	Suurus 84 x 118
Tõenäosus 29:40	Tõenäosus 1:8	Tõenäosus 1:40	Tõenäosus 1:40	Tõenäosus 1:100	Tõenäosus 1:100

Joonis 16. Genereeritavate objektide suurused ja tõenäosused.

Objektide ebakorrapäratul paigutamisel tuleb kontrollida, et ei eksisteeriks samal kohal teist objekti. Juhul, kui ala on vaba, salvestatakse valitud koordinaadid hõivatuks. Iga uus objekt kontrollib, kas temale genereeritud koordinaadid on hõivatud või mitte.

4.1.10 Koolja

Mängu kõige tähtsam osa on kooljatel, kellega peab vältima kontakti nende eest ära joostes. Selleks, et toimuks tagaajamine, on loodud loogika, mis jälgib mängija asukohta ning sõltuvalt sellele liigutab koolja asukohta.

Peale koolja asukoha initsialiseerimist käivitatakse elluärkamise animatsioon, mis annab esimese võimaluse mängijale muuta oma liikumise suunda, vältimaks kokkupõrget kooljaga. Igal kooljal on määratud raadius, milles mängija paiknemise korral enda asukohta maailmas liigutatakse.

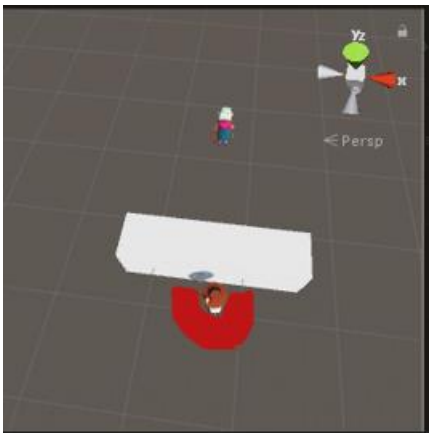
Liikumine toimub fikseeritud kiirusega, mis jääb 6% alla mängija kiirusele. Liikumine toimub ainult ette, seetõttu määratakse rotatsioon mängija suhtes iga kaadri korral juhul kui mängija paikneb nägemisraadiuses. Koolja mängija nägemise raadiusest väljas paiknemise korral asukohta ei muudeta.

4.1.11 Objektide kokkupõrked

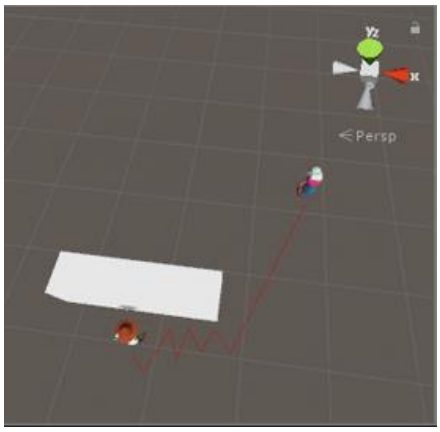
Objektid, mis paiknevad maailmas, peavad aitama mängijat põgenemiseks. Selleks, et vältida läbi seinte jooksmist, tuleb objektidele lisada *RigidBody* moodul, mis seob objekti füüsikaseadustega, neile mõjub gravitatsioon ning objektide kokkupõrkeid on võimalik registreerida, kui on lisatud ka kokkupõrkepiirid. [7] [8]

Mängija ja vastase kokkupõrke puhul registreeritakse erinevate märgistustega elemendid ning tegevusega lõpetatakse mäng. Mängija ja keskkonnaobjektide kokkupõrke tähendab vaid seda, et edasine liikumine pole võimalik ja mängija asendit samas suunas muuta pole võimalik. Sõltuvalt mängija sisendist kontrollerisse, muudetakse mängija asukohta vabades suundades.

Koolja ja keskkonnaobjekti kokkupõrke registreerimisel kutsutakse välja meetod, mis paneb surnu liikumistrajektoori muutma. Ilma kokkupõrketa leitakse otsene tee mängijaga, kuivõrd seda tehtaks peale kokkupõrget, oleks ilmselt otsene tee mängijani kas paremalt või vasakult poolt objekti. Teatavasti pole kooljad targad ning kinnijäämisefekt eesoleva seina taha lahendatakse teises suunas liikumisega. Üritatakse liikuda mängija poole veidi vasakult või paremalt justkui läbi seina, (vt. Joonis 17) välja arvatud juhul, kui mängija paikneb objekti ja koolja suhtes vähemalt 30 kraadise nurga all (vt. Joonis 18).

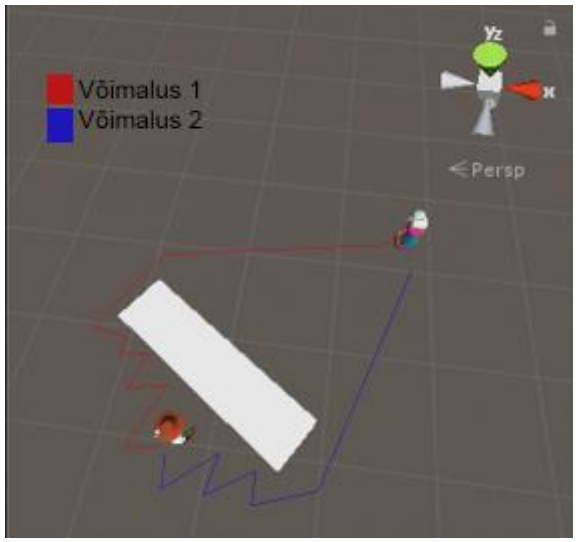


Joonis 17. Koolja trajektoori muutmise raadius.

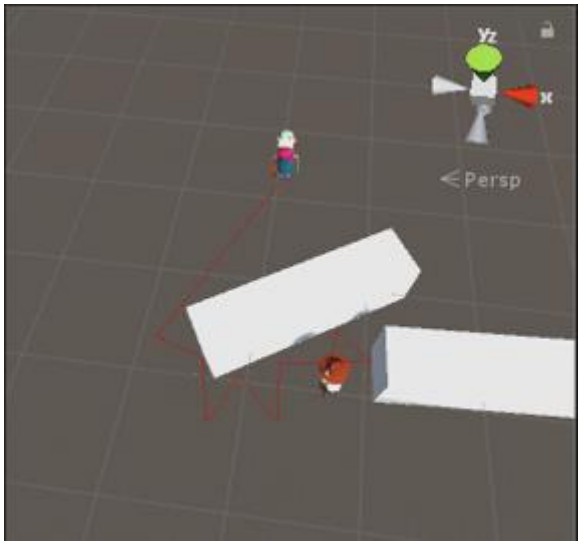


Joonis 18. Koolja uus trajektoor mängijani.

Kokkupõrke korral võib nurk objekti ja koolja vahel määrata objektist mööda mineku, kus ei proovita enam läbi seina minna muudetud trajektooriga, vaid üritatakse minna kõrvalt (vt. Joonis 19). Sama toimub ka olukorras, kus peale suuna muutust toimub järgmine kokkupõrge (vt. Joonis 20).



Joonis 19. Koolja ja objekti kokkupõrkenurgast tingitud liikumine.



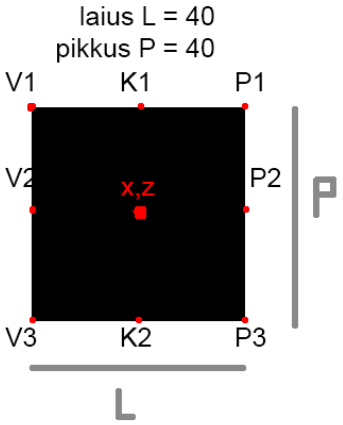
Joonis 20. Mitmekordne kokkupõrge.

Tehisintellektide liigutamise viisist tingituna ei registreeri kooljad kokkupõrget samatüüpi objektidega, st teiste kooljatega. Probleemi lahendamiseks on lisatud veel üks kokkupõrke registraator ning koolja asukohta ei muudeta, kui ollakse teise koolja sisemises kokkupõrkeraadiuses.

4.1.12 Mängu optimeerimine

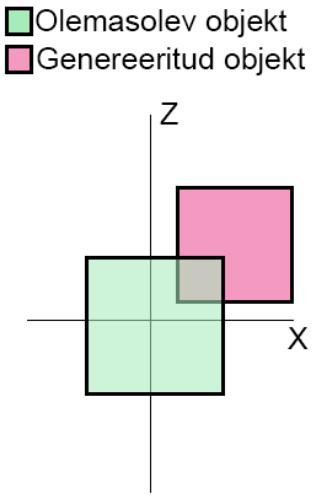
Uue lisatava objekti genereeritud asukoha oleku kontroll. Juhul, kui lisatav objekt on kõige suuremate võimalike mõõtudega ning mängus on juba näiteks 562 objekti, tekitab kõikide koordinaatide läbivaatamine suurema koormuse seadmele ning tekib viivitus mängu kuvamisel kasutajale– on nähtav hetkeline kaamera seisma jäämine koormuse tõttu.

Probleemi lahendamiseks kontrollib programm objektide olemasolevaid koordinaate vaid osaliselt: iga objekt salvestab enda keskpunkti ning oma suuruse, mida uus objekt võrdleb enda valitud asukoha ja suurusega. Arvutuste tulemusel võrreldakse näiteks neljasaja punktiga objekti korral vaid üheksat punkti iga olemasoleva objekti üheksa punktiga. Iga objekti võrreldavate punktide paigutus on näha joonisel 21, kus objekti keskkoht paikneb asukohal (x, z), lisaks kontrollitakse punkte V1(x-L/2, z+P/2); V2(x-L/2, z); …; P2(x+L/2, z); P3(x+L/2, z-P/2).



Joonis 21. Võrreldavate punktide paigutus.

Optimeerimisega väheneb suur kontrollitavate punktide hulk, kuid on võimalik sarnane paigutus nagu joonisel 22.



Joonis 22. Optimeerimise järel tekkiv paigutuse viga.

Viga välditakse kokkupõrke registreerimisega ning juhul, kui uue objekti paigutamine toimuks joonis x järgi, uus objekt soovitud positsiooni hõivatuks ei märgiks. Juhul kui lisatav objekt oleks tempel või kirik, siis eemaldatakse olemasolevad objektid, sest kiriku ja templi suurus on suurem, kui ühe aluspinna suurus ning olemasolevad objektid võivad pärineda teiselt genereeritud pinnalt. Ainult kokkupõrke kontrollimise järel objektide paigutamine ei ole võimalik, kuna tekib lõpmatu ahelreaktsioon olukorras, kus iga eelnev objekt on tempel ning lisatav objekt on kirik – kirik kustutab templi, tempel kustutab kiriku.

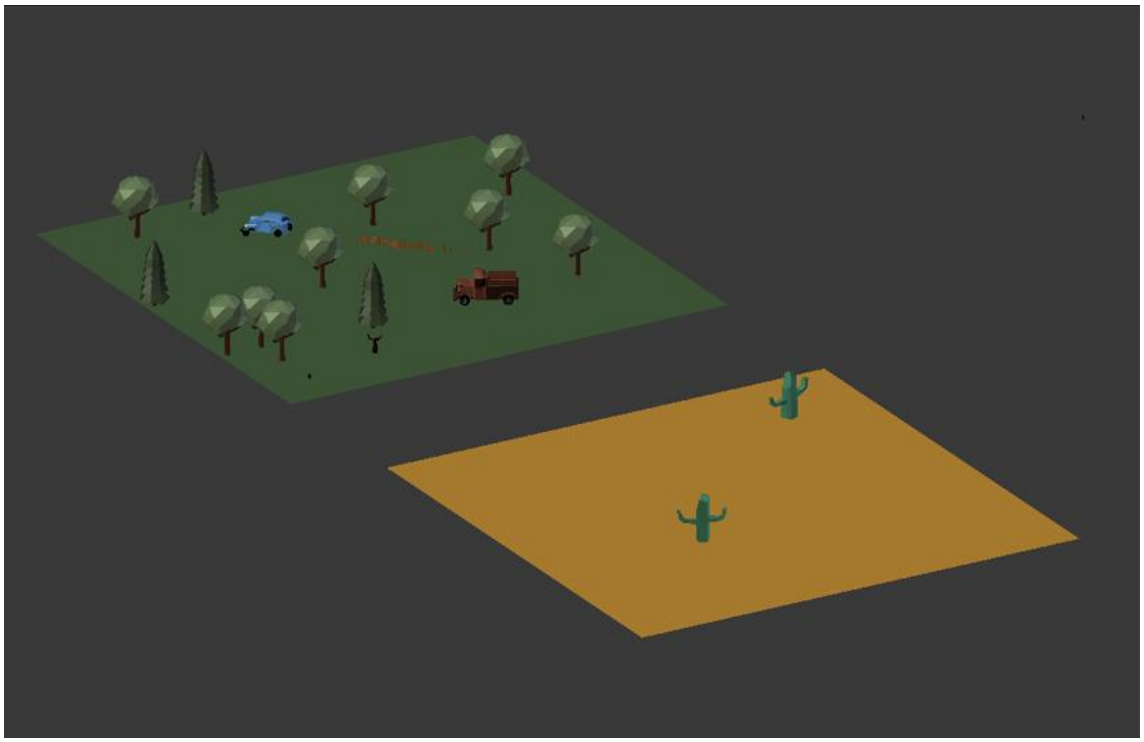
Kõikvõimalike objektide loomine. Eelnevalt kirjeldatud kaamera hüplemine tekib ka iga tasapinnaga seotud eseme genereerimisel. Esemete algväärtustamine on sobiv olukorras, kus genereeritakse mittelevinud objekte, mille kogus ei aeglusta nähtavalt seadme tööd. Olukorras, kus agente on sadu ja millisekundite jooksul tuleb paika panna palju objekte, ei ole iga elemendi loomine mõistlik. Kaamera hüplemise ja konstantse objektide loomise vältimiseks toimub esemete loomine rakenduse esmasel käivitamisel, mitte mängus olles. Maailmas ringi joostes kontrollitakse genereeritud asukoha olekut, juhul kui asukoht on vaba– seotakse loodud objekti asukoht genereeritud asukohaga, genereeritakse rotatsioon ning märgitakse objekt aktiivseks.

Objektide taaskasutamine. Rakenduse käivitamisel ei saa luua lõpmatut hulka objekte, mõistliku koormusena seadmele on piiratud objektide arv 3002-le, elavsurnute hulk 500-le. Kuigi elavsurnute suurim võimalik hulk ekraanil on 213 +- 10, siis loogika järgi paikneb ümbruses veel sadakond elavsurnut (vt. Joonis 23).

Joonis 23. Suurim võimalik elavsurnute hulk ekraanil.

Teoreetiliselt pole selline hetk mängus saavutatav, sest kontakt tekiks sisemise ringi kooljate ja mängija vahel enne ülejäänud 90% intellektide genereerimisest ning mäng loetakse lõppenuks. Vältimaks 500 vastase paiknemist maailmas nii, et ükski neist mängijat taga ei aja ja uute loomine ei ole võimalik, sest suurim vaenlaste hulk mängus on saavutatud, muudetakse liiga kaugele mängijast jäänud koolja mitteaktiivseks ning teda on võimalik taaskasutada uue koolja loomisel.

Kõikide genereeritud aluspindade ärakasutamine. Kõikide algselt loodud objektide ärakasutamine võtaks umbes 100 minutit, joostes maailmas vaid ühes suunas. Keskmine mängu aeg on umbes kaks minutit ning seetõttu ei tule loodud pindadest puudust. Juhul, kui kellelgi peaks õnnestuma 100 minutiline mäng, siis mängija jõuab kõrbesse, näiteks liikudes suunas üks joonisel 15, juhul kui kõik pinnad on kasutatud, initsialiseeritakse küll uued aluspinnad, aga loodavate elementide hulk on kuus aluspinda pluss maksimum kaksteist pindade peal paiknevat kaktust ning enam ei kontrollita objektide paiknevust rohelisel alal, sest kahe aluspinna suuruseid objekte ei genereerita. Kuue kuni kaheksateistkümne elemendi initsialiseerimine on kordades lihtsam, kui algne tuhandete elementide loomine ning tänapäevasele seadmele probleemi ei valmista (vt. Joonis 24).



Joonis 24. Genereeritud rohelise ala ja kõrbe näide.

5 Publitseerimine

Selleks, et rakendus jõuaks mängijateni, tuleb see üles laadida rakenduste poodi. Järgnevalt on kirjeldatud protsess iOS platvormile.

5.1 Arendaja staatus

Rakenduse lõppversiooni üleslaadimiseks Apple App Store, peab arendaja tasuma 99$ aastase Apple arendaja staatuse eest, mis annab õiguse laadida rakendusi virtuaalpoodi iOS, macOS, watchOS, tvOS operatsioonisüsteemidele. [18] Soovitatav on Apple seadmetele arenduse korral arendaja staatust omada juba faasis, kus soovitakse testida rakendust vastava platvormi peal, sest staatus annab ka võimaluse testida rakendust suunatud operatsioonisüsteemil, mitte ainult Unity siseselt.

5.2 Rakenduse laadimine App Store

Rakenduste eristamiseks tuleb määrata rakenduse identifikaator, mida Unity’s saab teha projekti seadetest. Identifikaator on igal rakendusel unikaalne, üldiselt on identifikaator kujul „com.arendaja_nimi.rakenduse_nimi“. Näiteks on „Dead Sprint“ identifikaatoriks „com.KevDev.DeadSprint“.

Unity kompileeritud rakendus tuleb avada programmis xCode, kust saab selle laadida Apple andmebaasi. Protsessi lõppedes tuleb avada arendajakonsool, kus määratakse kõik App Store turustamiseseaded.

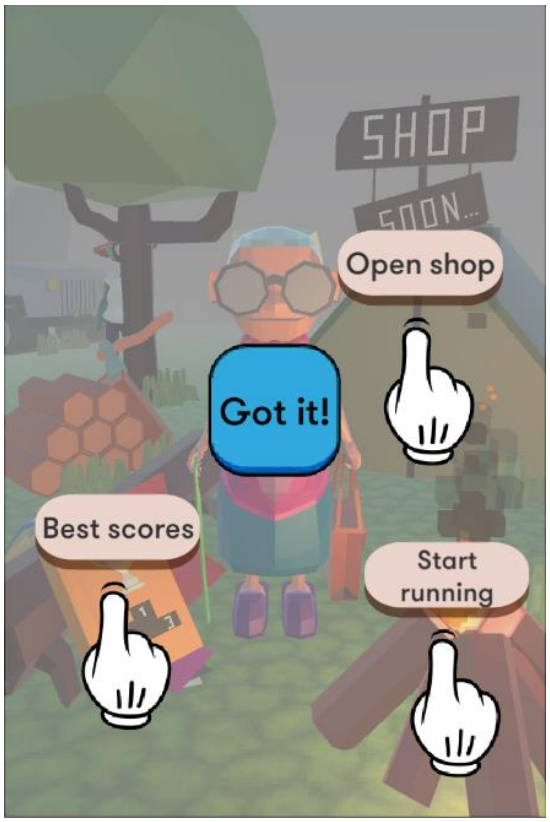
Seadetes tuleb määrata peamine informatsioon rakenduse kohta, milleks on rakenduste poes kuvatav nimi, rakenduse kirjeldus, rakenduse hind, rakenduse ikoon, tuleb lisada ekraanipildid rakenduse olemusest ja soovi korral saab lisada ka tutvustava video. Seadete määramise järel saab esitada rakenduse ülevaatamiseks. Kui kõik on Apple jaoks sobiv, lisatakse rakendus rakenduste poodi.

6 Rakenduse kirjeldus

Loodud rakendus on 3D lõputust jooksjast inspireeritud ellujäämismäng, kus mängija eesmärk on põgeneda kooljate eest nii kaua kui võimalik. Karakteri liigutamiseks mängu maailmas on ekraani allosas juhtkang. Mängus on erinevad ehitised, mille vahel mängija saab joosta ning kasutada neid põgenemiseks. Mängu maailm genereeritakse iga mäng erinevalt, väljaarvatud osa maailmast, kust mäng hakkab. Genereeritud maailm on piiramatu ning karakteri positsiooni muutumise korral genereeritakse ümbrusesse uus osa maailmast, mida mängija veel avastanud ei ole. Sõltuvalt mängija võimest hoida oma karakterit kooljate kokkupuutest eemal, kiireneb kooljate juurde tekkimine.

6.1 Rakenduse töökäik

Mängu käivitamisel genereeritakse kõik vajalikud objektid, mida mäng kasutama hakkab. Juhul, kui mängija avab mängu esimest korda, kuvatakse talle juhised, kuidas mängu mängida (Vt. Joonis 25).



Joonis 25. Mängujuhised „Dead Sprint“ esimesel käivitamisel.

Juhiste sulgemise järel on mängija menüüstseenil (Vt. Joonis 26).

Joonis 26. Esimene vaade rakenduses.

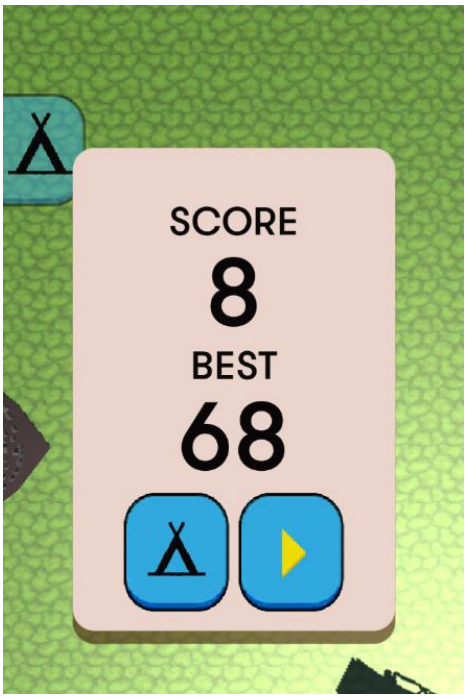
Mängijal on võimalus vajutada menüüstseenil paiknevatele objektidele. Objekte, mida saab vajutada, toimivad kui nupud. Näiteks saab siseneda poestseenile vajutades telgile. Poestseenil on võimalus vajutada matkakotile, mis avab uute karakterite ostuvaliku (Vt. Joonis 27).

Joonis 27. „Dead Sprint“ poestseen.

Telgi uksele vajutades minnakse tagasi menüüstseeni. Menüüstseenil raamatule vajutades kuvatakse kõrgeim mängus saavutatud punktiskoor. Vajutades lõkkele, sisenetakse mängustseenile (Vt. Joonis 28).

Joonis 28. „Dead Sprint“ mängustseen.

Mängustseenile sisenedes ilmub ekraanile kontroller, mida liigutades liigub ka karakter. Mängus tekivad kooljad karakteri ümbrusesse ning nende tekkimine kiireneb ajapikku. Iga koolja üritab liikuda peale tekkimist karakteri suunas juhul, kui asub reageerimisraadiuses. Koolja reageerib vastavalt keskkonnas olevate objektidega kokkupõrkele järgnevalt: juhul, kui objektiks on teine koolja– välditakse üksteisesse astumist. Juhul, kui objekt on mängija– loetakse mäng lõppenuks ning võrreldakse mänguskoori kõrgeima punktisummaga. Kui skoor on suurim olemasolevatest, kirjutatakse skoor kõrgeimate hulka. Juhul, kui objekt on midagi muud, muudetakse oma liikumise suunda ning üritatakse liikuda karakteri poole muudetud trajektoorilt. Mängu lõpu korral kuvatakse mängus saavutatud punktisumma, kõikide mängude kõrgeim punktisumma ning kaks nuppu (Vt. Joonis 29).



Joonis 29. „Dead Sprint“ mängu lõpp.

Vajutades telgiikooniga nupule, väljub mängija mängust ning ekraanil on jälle menüüstseen. Teisele nupule vajutades käivitatakse mängustseen uuesti. Lisaks on võimalik mängust võimalik lahkuda igal hetkel– vajutades ekraani vasakul üleval nurgas paiknevat telgiikooniga nuppu.

7 Kokkuvõte

Käesoleva töö eesmärgiks oli arendada mobiilirakendus „Dead Sprint“ iOS platvormi jaoks ning tutvustada rakenduse loomiseks kasutatud mängumootorit.

Rakendus valmis programmis Unity3D ning töös loodi peamised mängukomponendid, sealjuures mängijat ümbritseva maailma generaator, kokkupõrgete kontroller ja mänguga seonduvad stseenid.

Kokkuvõtteks võib öelda, et töö täitis oma eesmärki. Peamiste loogikate arenduse tulemusena on võimalik navigeerida erinevates vaadetes, kontrollida oma karakterit joostes kooljate eest nii kaua kui võimalik. Töös on kirjeldatud ka peamiste loogikate opereerimine ning optimeerimine.

Kasutatud kirjandus

[1] Blender, „https://www.blender.org/about/, “ [Võrgumaterjal]

[2] Unity3D, „https://unity3d.com/public-relations“ [Võrgumaterjal]

[3] xCode, „https://developer.apple.com/xcode/“ [Võrgumaterjal]

[4] Apple seadmete ühilduvus,
„https://developer.apple.com/library/content/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html“ [Võrgumaterjal]

[5] Unity3D Raycast,
https://docs.unity3d.com/ScriptReference/Physics.Raycast.html [Võrgumaterjal]

[6] Unity3D Update, „https://docs.unity3d.com/Manual/ExecutionOrder.html“ [Võrgumaterjal]

[7] Unity3D RigidBody,
„https://docs.unity3d.com/ScriptReference/Rigidbody.html“ [Võrgumaterjal]

[8] Unity3D Collider, „https://docs.unity3d.com/ScriptReference/Collider.html“ [Võrgumaterjal]

[9] Thought Catalog, „https://thoughtcatalog.com/jane-hurst/2015/02/12-types-of-computer-games-every-gamer-should-know-about/“, [Võrgumaterjal]

[10] R. A. Bartle, „Introduction to Virtual Worlds,“ *Designing Virtual Worlds,* 2003, pp. 1-4;125.

[11] D. H. Eberly, „Collision Detection,“ *3D Game Engine Design,* 2001, pp. 185-186; 341-342.

[12] T. Castillo, J. Novak, „Game Genres & Level Structure“ *Game Development Essentials: Game Level Design,* 2008, pp. 29-42

[13] T. Castillo, J. Novak, „Modeling“ *Game Development Essentials: Game Level Design*, 2008, pp. 168-172

[14] Blenderi funktsioonid, „https://www.blender.org/features/,“ [Võrgumaterjal]

[15] MonoDevelop, „http://www.monodevelop.com/“ [Võrgumaterjal]

[16] Unity3D funktsioonid, „https://unity3d.com/unity“ , [Võrgumaterjal]

[17] T. Castillo, J. Novak, „Paper Design“ *Game Development Essentials: Game Level Design*, 2008, pp. 80-84

[18] Apple Developer Console, „https://developer.apple.com/“ [Võrgumaterjal]

[19] Pinterest, *Low Poly character design*, „https://pin.it/dwvn42hwfu5iap“ [Võrgumaterjal] [Accessed: 19-May-2018]