TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikita Koikov 142857 IAPB

VIRTUAALREAALSUSMÄNGU ARENDUS UNITY MÄNGUMOOTORIL

Bakalaureusetöö

Juhendaja: Jaagup Irve
Tehnikateaduste magister

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: Nikita Koikov

09.05.2018

Annotatsioon

Lõputöö eesmärkideks on tutvustada virtuaalreaalsuse kontseptsiooniga ja arendada VR mängu nutitelefonidele.

Esimeses osas uuritakse, mis on VR ja kuidas seda kasutatakse. Kirjeldatakse kasutatavaid platvorme, mille abil toimub sukeldumine VR maailmasse ning uuritakse meetodeid VR maailma objektidega suhtlemiseks. Lisaks antakse ülevaade kõige populaarsematest tänapäeva mängumootoritest.

Teises osas luuakse VR mäng nutitelefonidele Unity mängumootori abil ning kirjeldatakse arendamisprotsessi. Antakse ülevaade peamisest Unity's kasutatud komponentidest. Pakutakse lahendusi probleemidele, mis võivad tekkida VR mängu arendamise käigus, näiteks dünaamilise suhtlemisdistantsi loomine. Mängus on kaks juhtimisviisi. On realiseeritud meetod, mis annab mängijale võimaluse teostada erinevaid tegevusi Google Cardboard'il asuvat nupu kasutades.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 32 leheküljel, 4 peatükki, 18 joonist, 1 tabelit.

Abstract

Virtual reality game development with Unity game engine

The purpose of this thesis is to explore the concepts of virtual reality and to create a VR game for smartphones. The thesis is divided into two parts.

First part describes, what is VR and how it is used, provides an overview of methods and controllers used for interaction with objects in VR world. Two most popular game engines for VR development were also analyzed.

Second part provides an overview of most used Unity components, concepts and design of the game. Are described problems, which can occur during VR game development, such as dynamic interaction distance with different types of objects or player movement normalization. Found solutions are provided.

As a result of second part a VR game is created with Unity game engine. The game consist of several levels and the goal of each level is to destroy a tower of blocks using a catapult. The player has to grab a ball, place it in the catapult and press a button for shot. For flexible shooting the player can move and rotate the catapult. Different types of tower blocks and balls were added. The game has two control modes: first is for Bluetooth controller with VR glasses for smartphones and second for most common VR glasses - Google Cardboard, with one single button on it. Using second control mode the player is able to execute different actions with one button as an input.

The thesis is in Estonian and contains 32 pages of text, 4 chapters, 18 figures, 1 table.

Lühendite ja mõistete sõnastik

Editor	Visuaalne keskond rakenduse arendamiseks
GUI	*Graphical user interface*, graafiline kasutajaliides
HMD	*Head mounted display*, peale kinnitatud kuvar - VR prillid
SDK	*Software development kit*, tarkvaraarenduskomplent
Skript	*Script*, fail programmikoodiga
VR	*Virtual reality*, virtuaalreaalsus

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Antud töö on seotud virtuaalreaalsuse teemaga ning koosneb kahest osast. Töö eesmärkideks on tutvustada, mida endast kujutab VR ja kus seda kasutatakse, ning luua VR mäng mängumootori abil.

Esimeses osas tutvutakse VR üldise kontseptsiooniga. Uuritakse platvorme, mis toetavad VR rakendusi, ja vahendeid, mille abil saab selliseid rakendusi arendada. Tutvutakse suhtlemismeetoditega, mis võimaldavad suhelda objektidega VR maailmas. Lisaks antakse ülevaade kahest kõige populaarsemast mängumootorist.

Teises osas kirjeldatakse VR mängu arendamisprotsessi Unity mängumootoril. Tutvutakse olulisemate komponenditega, mis olid kasutatud mängu loomise jooksul. Samuti kirjeldatakse mängu kontseptsiooni ning disaini. Käsitletakse probleeme, mis võivad tekkida VR mängude arendamise käigus, sellised nagu mängija liikumise normaliseerimine, muutuva suhtlemisdistantsi loomine või mänguobjekti liikumisala piiramine. Pakutakse omapoolseid lahendusi tekkinud probleemidele.

Praktilise osa tulemuseks on VR mäng nutitelefonidele, kus on mitu taset ja iga taseme eesmärgiks on lammutada suur ehitis kasutades selleks heitemasinat ja kuule. Heitemasinat on võimalik keerata ja lükata. Tulistamiseks tuleb asetada kuul selleks ettenähtud süvendisse heitemasinal ning vajutada nupule. Mängus on olemas kaks juhtimisrežiimi: kasutades Google Cardboard prille ja selle peal asuvat nuppu või teisi VR prille nutitelefonide jaoks koos *Bluetooth* kontrolleriga. On realiseeritud meetod erinevate tegevuste täitmiseks ühe nupuga.

Töö on tehtud Tallina Tehnikaülikooli informaatika eriala bakalaureuse õppeastme lõputööna ja esitatakse 2018. aastal Tallinnas.

2 Virtuaalreaalsuse kontseptsioon

Virtuaalreaalsus (ingl. *Virtual reality*, lüh. VR) – on arvuti poolt genereeritud keskkond, mis imiteerib mõnel määral reaalmaailma. Kasutaja saab tunnetada VR maailma selliste aistingute kaudu nagu nägemine, kuulamine, kompimine ja teised. Kõik võimalikud objektid VR maailmas imiteerivad reaalmaailma objektide käitumist ning reageerivad kasutaja tegevustele reaalajas, mis annab talle teises maailmas olemise tunde.[1]

2.1 VR rakenduste kasutusala

VR rakenduste kasutusala on väga suur. Selliste rakenduste kaudu imiteeritakse situatsioone, mis võivad reaalmaailmas tekitada otsest ohtu elule või muutuda liialt kulukaks tegevuseks. Näiteks on võetud VR rakendusi kasutusele lennukoolides ning sõjaväeges, et inimestele muutuks õhusõiduki juhtimine ohutumaks ning kättesaadavamaks. VR rakendusi on ka võimalik kasutada linnade ning erinevate suurte ehitiste projekteerimiseks. Samas on VR tehnoloogia väga levinud ning arenenud meelelahutuses: filmid ja videod, kus on võimalik ringi vaadata ja sind ümbritsevaga lähemalt tutvuda, või mängud, mis imiteerivad tervenisti teist maailma. Viimasel ajal suureneb VR *chat*'ide populaarsus, kus inimesed suhtlevad üksteistega viibides samal ajal teises maailmas.[1]

2.2 Sukeldumine VR maailmasse

VR maailmasse sukeldumiseks peab kasutaja mingil määral olema isoleeritud teda ümbritsevast reaalmaailmast. Üldiselt seda tehakse nägemise kaudu: kasutajale antakse võimalus tunnetada ainult virtuaalmaailma, aga reaalmaailma eemaldatakse kasutaja vaateväljast. Selleks kasutatakse nn peale kinnitatud kuvarit (ingl. *Head mounted display*, lüh. HMD) või lihtsalt VR prille. Peale isolatsiooni reaalmaailmast VR prillid registreerivad samuti kasutaja pealiigutusi ning tagavad suuremat nägemisvälja võrreldes tavaliste monitoridega.[1]

2.2.1 HDM omadused

Tänapäeval eksisteerib väga palju erinevaid VR prille, kuid kõigil nendest on ühesugused omadused.

Selleks, et VR prillid saaksid kasutajale teises maailmas olemise tunde tekitada, on nende sees kasutusel andurid, mis aitavad inimese pealiigutusi registreerida. Selliste andurite hulka kuuluvad magnetomeeter[1], kiirendusandur[2] ja güroskokop[3].[2]

VR prillidel on suhteliselt suur nägemisväli: tänapäevastel monitoridel ja televiisoritel on tavaliselt 30-40-kraadine nägemisväli, kuid VR prillidel see võib varieeruda 80 kuni 210[4] kraadini. Suurem nägemisväli võimaldab edastada pilti perifeersele nägemisele. Edastatava maailma täisväärtuslikuks ja sujuvaks tajumiseks on pildi värskendussagedus tavaliselt umbes 60 kaadrit sekundis.[3]

Veel üks oluline HMD omadus on pildiedastamine stereoskoopilise pildi kaudu. Stereopilt koosneb kahest pildist, kus on kujutatud sama stseen või objekt vaadates erinevatest kohtadest (Joonis 1). Erinevus kohtade vahel on umbes sama, mis on inimese silmade vahel. Tänu stereopildile kasutajal tekkib tunne, et VR maailm on kolmemõõtmeline. Lisaks VR prillide sees asuvad tavaliselt kaks läätse, mille kaudu iga silm näeb oma pilti.[3]

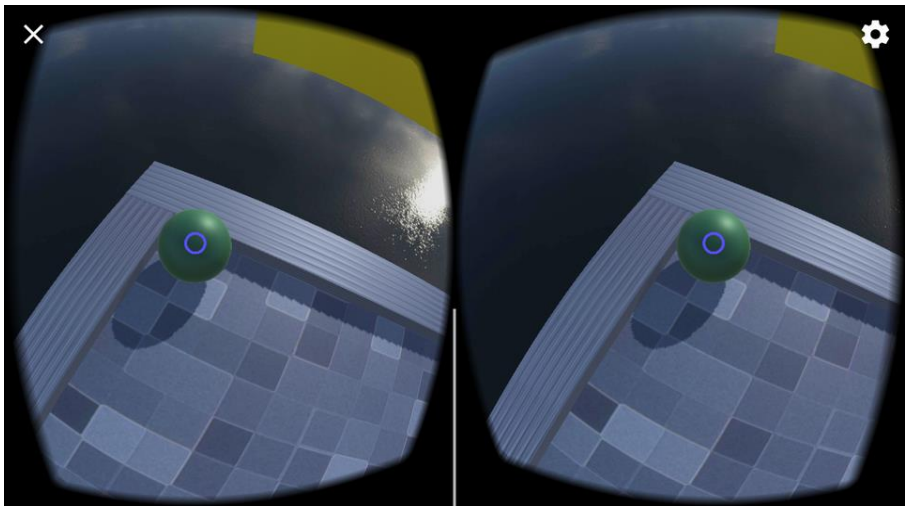
---

[1] Magnetomeeter on vahend magnetvälja paraametrite mõõtmiseks. VR rakendustes kasutatakse kompassprintsiibi järgi maailmajaode kindlaksmääramiseks

[2] Kiirendusandurit (ingl. *Accelerometer*) kasutatakse näiliku kiirenduse projektsiooni mõõtmiseks. VR rakendustes aitab registreerida VR prillide asukoha muutmist

[3] Güroskoop on seade, mis võimalab registreerida keha orientatsiooninurgad ning säilitada ruumilist orientatsiooni. VR rakendustes kasutatakse pea kaldenurgade registreerimiseks

[4] Kõige laiem nägemisväli on VR prillidel StarVR (http://virtualrealitytimes.com/2017/03/06/chart-fov-field-of-view-vr-headsets/)



Joonis 1. Stereopildi näidis. Piltide keskel asub sihik-kursor

2.2.2 HDM tüübid

Kõiki VR prille on võimalik jagada kaheks põhitüübiks, mis erinevad omavahel VR rakenduste käivitamisplatvormi olemasolust.

Portatiivsetel VR prillidel asub kõik VR rakenduse kasutamiseks vajalik prilli enda sees, sisseehitatuna: läätsed, kuvar ja riistvara. Riistvarale on installeritud tarkvara, mis vastutab andurite andmetöötluse ning rakenduse enda esituse eest. Kuvar koos riistvaraga võivad omakorra olla portatiivsed, kuna nende funktsioone täidab tihti nutitelefon.[3]

Statsionaarsed VR prillid – selliste prillide sees asuvad ainult läätsed, kuvar ja andurid. Kogu informatsioon koos rakendusega töödeldakse eraldi masinas. Selliste prillide kasutusele võtmiseks tuleb arvestada võimeka arvuti või mängukonsooli olemasoluga.[3]

2.2.3 Lisakontrollerid

VR rakendusi kasutades pealiigutuste registreerimine on tihti ebapiisav sellel põhjusel, et erineva tegevuse täitmiseks tekib vajadus lisasisendiks. Sellepärast kasutatakse koos VR prillidega sageli ka kontrollereid.

Müügis võib kohtuda palju erinevaid kontrollereid. Mõned on mõeldud kasutamiseks ühe käega, teisi on vaja hoida mõlema käega. On olemas kontrollerid, mis asuvad HMD peal. Lisaks nupudele, võib kontrolleri peal asuda ka puuteplaat. Kontrollerite põhilised

ühendustüübid on kas traadiga või traadita, näiteks *Bluetooth*[1] kaudu. Mõned kontrollerid jälgivad käe asukohti ning see võimaldab imiteerida kasutaja käeliigutusi VR maailmas.[4]

Tänapäeval on olemas selliseid kontrollereid, mis pakkuvad rohkem huvi. Näiteks VR kindad[2], mis jälgivad sõrme- ja käeliigutusi, või VR dress[3], mis lisaks terve inimese kehaliigutuste jälgimisele oskab luua ka survet mõnele kehaosale ja sellega tekitada kasutajal kompimistunde. Need, kellele meeldib jalutada, võivad osta endale VR jalaplatvormi[4], mis osaliselt jälgib kehaliigutusi ning võimaldab kasutajal kükkida, jalutada ja joosta kohapeal.[4]

Ilma kontrolleriteta jääb ainult üks suhtlemisviis VR maailmaga – nn *gaze input.* Pildi keskel asub väike ring ehk sihik-kursor (ingl. *Reticle pointer*), mida kasutaja pealiigutustega suunab objektidele ning kui ring suureneb (Joonis 1), saab kasutaja aru, et objektiga on võimalik suhelda. Objekt reageerib tavaliselt siis, kui kasutaja pilk on suunatud objektile mõne aja jooksul.[5]

2.2.4 Populaarsed VR komplektid

VR komplekt (ingl. *VR set*) on see, mida inimene vajab VR rakenduse kasutamiseks. Komplekt koosneb VR prillidest ning tavaliselt ühest või kahest kontrollerist. Tänaseks on saanud populaarseks päris mitu VR komplekti ning peamine erinevus nende vahel seisneb kontrollerites ja edastatava pildi kvaliteedis, millepärast on ka suur erinevus müügihindade vahel. Mõned populaarsed VR komplektid ja nende omadused on esitatud Tabelis 1.[3]

[1] https://www.techopedia.com/definition/26198/bluetooth

[2] https://manus-vr.com

[3] https://teslasuit.io/teslasuit

[4] http://www.virtuix.com

Nimetus	Riistvara	Kontrollerid	Nägemisväli (kraad)	Kuvarilahutus (pixels per eye)	Hind (USD)
HTC Vive	PC	2 tk, traadita, käe jälgimine, puuteplaadid	100	1080x1200	500
Playstation VR	Sony Playstation 4	2 tk, traadita, käe jälgimine	100	960x1080	300
Gear VR	Nutitelefon	1 tk, traadita, puuteplaat	101	Varieeruv	200
Teised VR prillid nutitelefonidele	Nutitelefon	Bluetooth kontroller	90-100	Varieeruv	10-50
Google Cardboard	Nutitelefon	1 nupp HMD peal	96	Varieeruv	Kuni 20

Tabel 1. Populaarsete VR komplektide omadused

Tabeli andmeid analüüsides võib teha järelduse, et alustada VR rakenduste kasutamist läbi nutitelefoni võib endale lubada iga inimene suhteliselt madala hinna eest, aga parema VR kogemuse saamiseks tuleb kulutada tunduvalt rohkem raha VR komplektile ning arvatavasti veel rohkem platvormile, tänu millele mängud ja rakendused töötama hakkavad.

2.3 VR rakenduste arendus

VR rakenduste arendamine on väga sarnane kolmemõõtmelise mängu arendamisega. Peamine erinevus seisab juhtimisviisides ning suhtlemises VR objektidega.

Arendamiseks kasutatakse mängumootoreid (ingl. *Game engine*) – see on tarkvara, mis pakkub arendajatele mõningaid vahendeid ja komponente, näiteks mängusisene füüsika, animatsioonid ja objektide mudelid. Vahendite hulka kuuluvad sellised asjad nagu *game editor*[1] või *build tools*[2].

[1] *Editor*'is toimub visuaalne arendamine, nt 3D mudelite lisamine (https://docs.unity3d.com/Manual/UsingTheEditor.html)

[2] *Build tool* on vahend, millega saab luua fail installeerimiseks (https://www.techopedia.com/definition/16359/build-tool)

Tänapäeval kasutatakse mitut mängumootorit VR rakenduste arendamiseks: Unity, Unreal Engine 4, AppGameKit VR, libGDX, CryEngine ja teised. Eriti populaarsed on esimesed kaks.[6]

2.3.1 Unity

Unity on mängumootor, mis pakub palju võimalusi 2D ja 3D rakenduste arendamiseks. Kõigepealt see mängumootor on väga populaarne algajate hulgas kasutusmugavuse pärast.[7]

Unity's on graafiline keskkond, kus on võimalik luua ja näha rakendust eemalt lisades sinna komponente ja nendega manipuleerides. Arendamiseks pakutakse rohkem kui 20 sihtplatvormi: mobiilseadmed, personaalarvutid, mängukonsoolid ja teised. Igaüks saab alustada rakenduste arendamisega selle mängumootori abil, sest tema litsent lubab kasutada Unity't tasuta personaalsetes või madala sissetulekuga projektides. Programmerimiskeel skriptide kirjutamiseks on C#.[7]

Antud mängumootori nõrkadeks külgedeks tuuakse sageli välja suhteliselt madalat jõudlust suurtes projektides ja samuti mängumootori lähtekoodi kättesaamatust tasuta versioonidel.

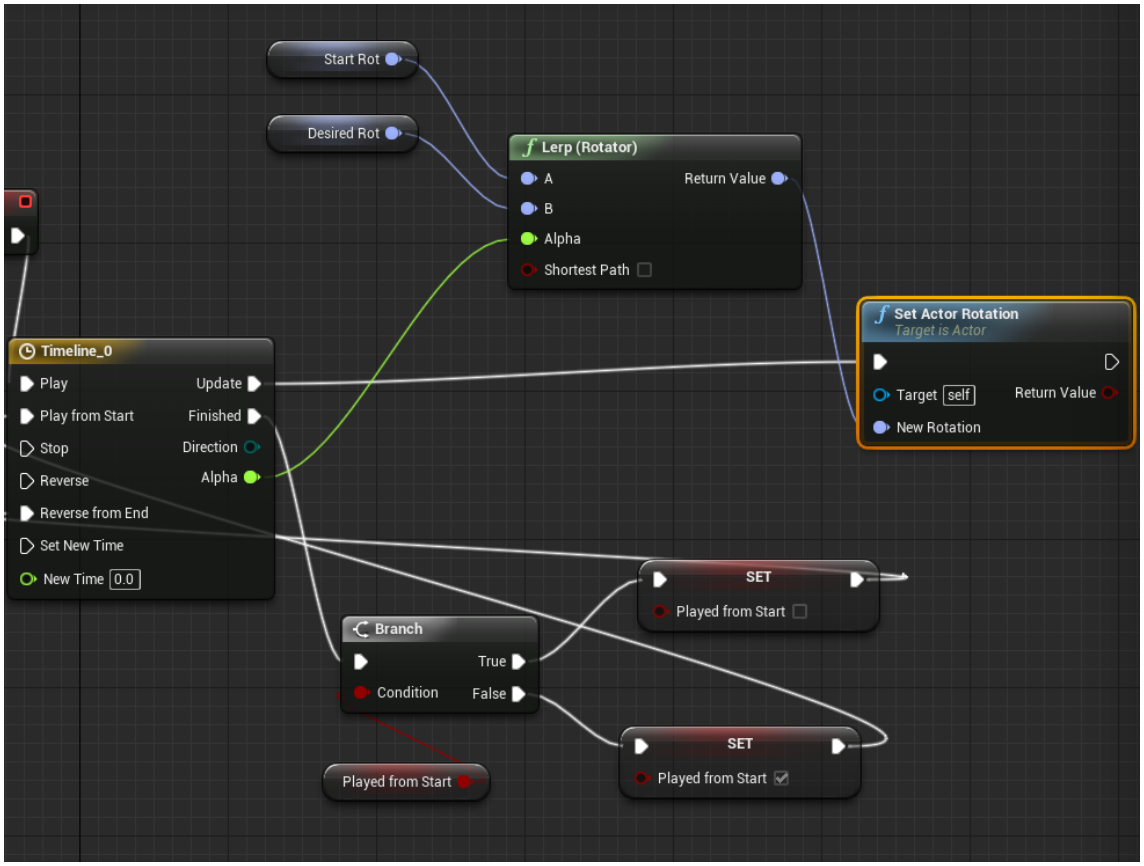
2.3.2 Unreal Engine

Unreal Engine on populaaruse poolest teine mängumootor VR rakenduste arendamiseks. Erinevalt Unity'st see mängumootor on avatud lähtekoodiga ja see võimaldab arendajatel vajaduse korral kontrollida, mis toimub mängumootori sees. Litsensi järgi edukate projektide omanikud on kohustatud maksma 5% projekti sissetulekust.[8]

Programmerimiskeel skriptide kirjutamiseks on C++, lisaks kasutatakse visuaalset programmerimist – *Blueprint*[1] (Joonis 2). *Blueprint* on piisavalt paindlik vahend mänguloogika loomiseks ja see võib täiesti asendada C++ programmerimist. Unreal Engine sobib hästi projektidele, mis vajavad suurt jõudlust, näiteks kõrge graafikakvaliteedi jaoks. See mängumootor võimaldab samuti luua rakendusi paljudele sihtplatvormidele.[8]

[1] https://docs.unrealengine.com/en-us/Engine/Blueprints

Unreal Engine ei ole soovitatav algajate jaoks oma rikkaliku funktsionaalsuse ja järelikult keerulise kasutajaliidese pärast. Lisaks ta võib olla ebamugav arendamiseks visuaalse programmerimise pärast nendele, kes on harjunud koodi kirjutama. Selle mängumootori kohta on olemas vähem dokumentatsiooni artikleid ning õppematerjale võrreldes Unity mängumootoriga, mis samuti raskendab kasutamist esimesel ajal.[9]



Joonis 2. *Blueprint*'i näidis

3 Mänguarendus kasutades Unity

Selles peatükis on kirjeldatud mängu „*VR Catapult*“ loomiseprotsessi. Põhjendatakse kasutusele võetud tehnoloogiate valikut. Antakse ülevaade kõige olulisematest mängumootori komponentidest. Tutvutakse mängu kontseptsiooni ja disainiga. Kirjeldatakse põhilisi tekkinud probleeme ja pakutakse neile lahendusi.

3.1 Tehnoloogiad

Kuna autoril puudub varajasem kogemus mängu arendamises tuli hakata valima kahe populaarseima mängumootori vahel: Unity ja Unreal Engine. Peamine tegur oli programmerimiskeelte erinevus ning autoril oli suurem programmerimiskogemus C# keeles, kui võrreldes C++’iga. Samuti valikut mõjutas ka see, et Unity on üldiselt kergem mängumootor algajatele ning veebis on rohkem infot ja õppematerjale just Unity kohta.

VR mängu mugavaks arendamiseks ja selle korrektseks töötamiseks nutitelefonidel oli võetud kasutusele raamistik Google VR SDK. Antud raamistik on tasuta ja on olemas erinevate mängumootorite jaoks, Unity kaasa arvatud. Raamistik annab arendajale võimaluse imiteerida pea kallutusi ja pöördeid Unity *editor*’is kasutades selleks arvutihiirt ja klaviatuuri. Google VR SDK pakub levinuid komponente VR rakendustes, näiteks skript *gaze input*’i jaoks, mille kasutades tekib ring piltide keskel.[10]

Sellel põhjusel, et Unity’s on piisavalt primitiivne kolmemõõtmilise objektide loomise mehhanism, arendajatel tekib vajadus importida oma 3D mudelid Unity projekti. Üks levinumatest rakendustest 3D mudelite loomiseks ja importimiseks on Blender[1]. Antud rakenduses on väga lai vahendite valik professionaalseks 3D modellerimiseks. Varem modellerimiskogemus autoril puudub ning sellega oli valitud Blender oma populaarsuse pärast.

[1] https://www.blender.org

Programmikoodi kirjutamiseks kasutatakse Visual Studio 2017[1]. Koodi säilitamiseks ja versioneerimiseks oli valitud resurss bitbucket.org[2] ning repositooriumiga suhtlemiseks kasutatakse SmartGit[3].

3.2 VR platvormi valimine

Platvormi valimine, millel planeeritakse VR mängu arendada, peamiselt sõltub selle platvormi levimusest ja kättesaadavusest müügiturul nii arendajate kui ka kasutajate jaoks. Ilmselt tehnoloogiaid poole tuhande euro eest ostab väiksem osa inimestest võrreldes suhteliselt odavate VR komplektidega, mis koosnevad VR prillidest, *Bluetooth* kontrollerist ja nutitelefonist, mida omab peaaegu iga kaasaegne inimene. VR komplekt nutitelefonidele oli valitud platvormiks oma kättesaadavuse pärast.

Sihtoperatsioonisüsteemiks oli valitud Android, sest autoril on testimiseks nutitelefon sellise operatsioonisüsteemiga olemas ning mängu mugava levitamisvõimaluse pärast läbi Google Play[4].

3.3 Mängu kontseptsioon

Selles osas on kirjeldatud mängu „*VR Catapult*“ kontseptsioon ja selle funktsionaalsuse põhiülesanded.

3.3.1 Mängu idee ja eesmärk

Mängus tuleb lammutada plokkidest koosnev ehitis kasutades heitemasinat ja kuule. Selleks on mängijal võimalus võtta kuul nähtamatusse kätte, asetada kuul heitemasinal asetsevasse süvendisse ja seal samas heitemasina kõrval olevat nupu vajutades lasta kuul lendu. Lisaks on võimalus muuta heitemasina asendit. Mängus on mitu taset, iga taseme eesmärk on lammutada peaaegu terve ehitis ja kulutada selleks võimalikult vähe kuule.

[1] https://www.visualstudio.com

[2] https://bitbucket.org

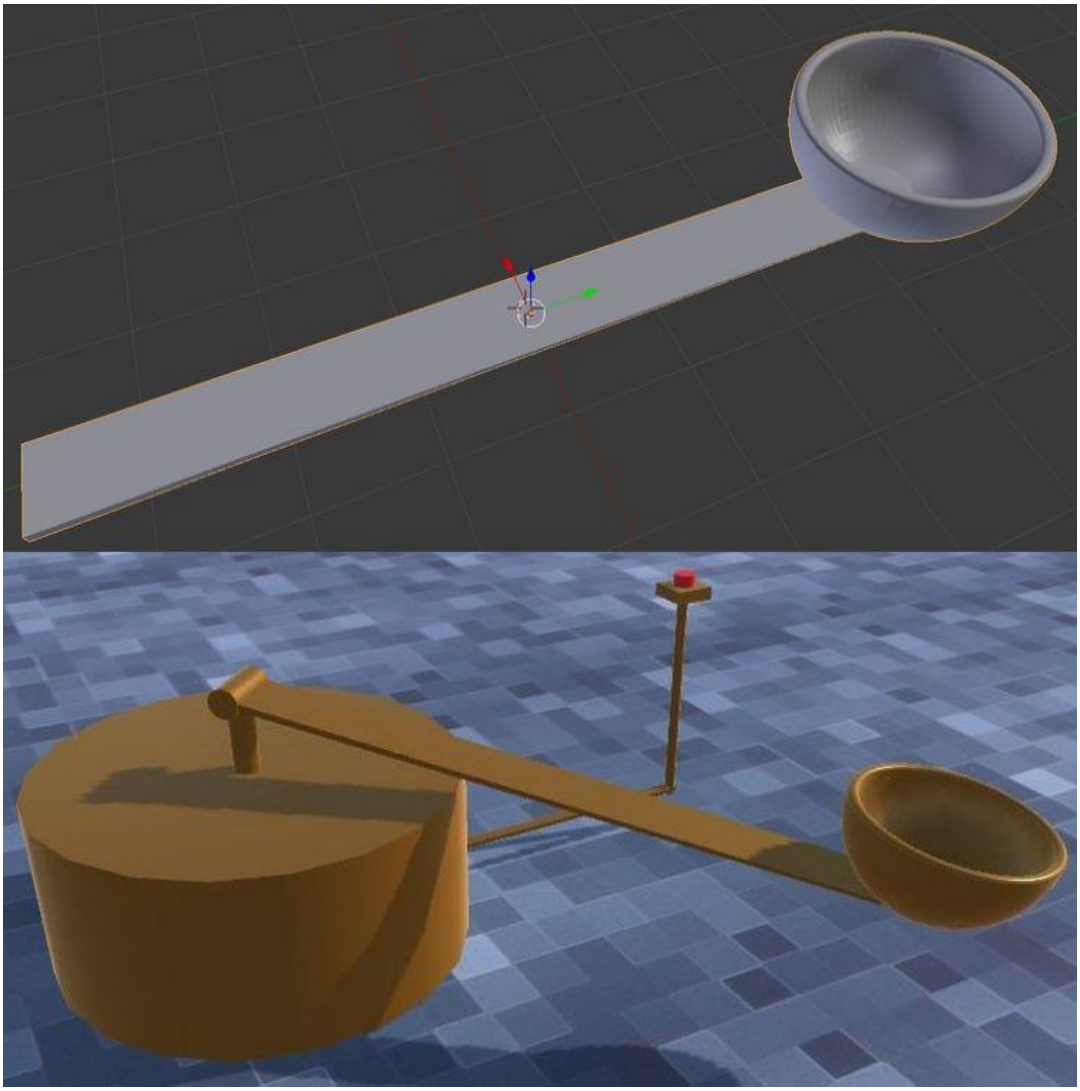
[3] https://www.syntevo.com/smartgit/

[4] https://play.google.com

3.3.2 Heitemasin

Iga taseme alguses mängija näeb enda ees mänguvälja keskel asuvat heitemasinat. Mängijal tuleb panna kuule heitemasinasse ja tulistamiseks vajutada nupule, mis asub heitemasina kõrval. Selleks, et kuuli heide oleks tõhusam, saab mängija heitemasinat keerata endale sobivas suunas, puudutades selleks osa, millesse pannakse kuule. Samal põhjusel mängijal on võimalus muuta heitemasina asukoha mänguvälja piirides ja selleks tuleb tõugata heitemasina põhiosa.

Omapärase kuju tõttu heitemasina loomiseks oli vaja kasutada modellerimisprogrammi Blender. Selle abil oli tehtud süvend, kuhu pannakse kuule (Joonis 3). Lisaks heitemasinale oli loodud veel kaks mudelit Unity projekti importimiseks.



Joonis 3. Heitemasinaosa Blender'is (üleval) ja heitemasin Unity projektis

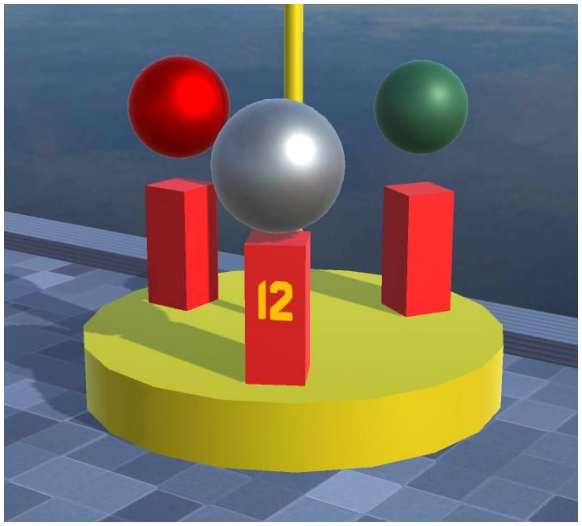
3.3.3 Heitemasina kuulid

Mängus on olemas kolme tüüpi kuule: tavalised, rasked ja lõhkevad. Nendel on erinevad omadused ning igal tasamel on erinev kogus kuule kasutamiseks. Kõik kuulid ilmuvad platvormi peal (Joonis 4), mida on võimalik liigutada samal kombel nagu heitemasinat.

Tavalistel kuulidel eriomadused puuduvad ning nende arv on tavaliselt piiramatu.

Rasked kuulid on suurema massi ja diameetriga võrreldes tavaliste kuulidega ning heitemasin tulistab raskemaid kuule suurema jõuga. Sellega antakse kuulidele suurem lammutamisejõud, siis kui toimub kokkupõrge ehitisega. Raskete kuulide arv on tavaliselt piiratud.

Lõhkevad kuulid on sarnased tavaliste kuulidega massi ja suuruse mõttes, kuid ehitisega kokkupõrke puhul toimub plahvatus, mis lükkab lähedal asuvaid plokke eemale. Ka nende kuulide arv on tavaliselt piiratud.



Joonis 4. Platvorm kuulidega

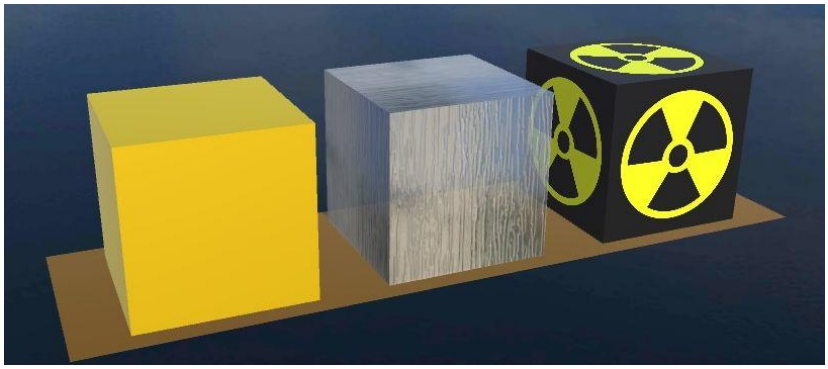
3.3.4 Ehitis ja plokkid

Ehitis koosneb plokkidest. Mängus on olemas kolme tüüpi plokke: tavalised, klaasplokid ja lõhkevad plokid (Joonis 5).

Tavalised plokid on peamised plokid, millest koosneb ehitis. Nad purunevad, kui kukkuvad ehitise põhjanivoost madalamale.

Klaasplokid annavad mängijale võimaluse lammutada ehitist kiiremini. Suure kiirusega kuuli pihtasaamise puhul antud plokid purunevad ja lasevad kuuli läbi vähendades kuulikiirust. Klaasplokidel on läbipaistev tekstuur ning kuuliga kokkupõrkemise puhul nad purunevad väikseteks osadeks.

Lõhkevatel plokidel on omadus kuuliga kokkupõrke puhul plahvatada ning tõugata naaberplokid eemale. Kui plahvatusalal asuvad teised plahvatusplokid, siis nad plahvatavad ka lühiajalise viivitusega, kuid klaasplokid lihtsalt purunevad.



Joonis 5. Ehitise plokkid

3.3.5 Suhtlemine mänguobjektidega

VR maailma objektidega suhtlemiseks kasutajal on vaja suunata oma pilk objektile ning kui pildi keskel asuv ring suureneb, tuleb vajutada tegevuse nupp – *action button.*

Arendades VR mängu tavaliselt piiratakse distantsi, mille puhul suhtlemine objektidega on lubatud. Näiteks antud mängus tuleb kasutajal liikuda kuuli kõrvale, et oleks võimalus seda võtta, või olla heitemasina kõrval vajutades nuppu tulistamiseks. Lisaks tavalistele objektidele mängus on olemas GUI nupud, mis asuvad suure seina peal. Mugavamaks mänguprotsessiks otsustati teha suhtlemisdistants GUI nupudega piiramatuks, et oleks võimalus vajutada neid igalt kohalt mänguväljal.

3.3.6 Tuul

Mänguprotsessi mitmekesistamiseks oli mängu lisatud tuul, mis mõjutab kuuli liikumistrajektoori lennuajal. Tuule kiirus ja suund juhuslikult muutuvad iga poole minuti tagant. Igal tasemel on erinev minimaalne ja maksimaalne tuule kiirus. Mängija

saab tuule parameetreid jälgida tänu tuulelipu olemasolule (Joonis 6). Tuulelipp asub samal platvormil koos kuulidega.

Joonis 6. Tuulelipp

3.3.7 Mängu juhtimisviisid

Mängu kasutamiseks lähevad vaja VR prille nutitelefonide jaoks koos *Bluetooth* kontrolleriga. Kontrolleri peal asuvat *joystick*'ut[1] kasutades on võimalik liikuda igas suunas erineva kiirusega ning vajutades tegevuse nuppu suhelda mänguobjektidega.

Üheks levinumatest VR prillidest nutitelefonidele on Google Cardboard (Joonis 7) oma madala hinna tõttu ning sellepärast, et seda on võimalik konstrueerida kodus oma kätega[2]. Mängimise ajal originaalset Google Cardboard'i on vaja hoida ühe käega silmade ees ning sellega seoses on võimatu kasutada lisasisendiks *Bluetooth* kontrollerit, sest kontrollerit hoitakse mõlema kätega. Antud prillidele on lisatud omapärane sisendimeetod – üks nupp, mis asub prillide peal.

Mugavamaks mängimiseks kasutades Google Cardboard prille otsustati lisada võimalus teha mitu erinevaid tegevusi kasutades selleks ainukest prillide peal asuvat nuppu. Konkreetselt antud mängu jaoks: suhelda mänguobjektidega ning liikuda erineva kiirusega erinevates suundades.

---

[1] https://www.techopedia.com/definition/31108/joystick

[2] https://vr.google.com/cardboard/get-cardboard/

Juhtimisrežiimi saab kasutaja valida ja muuta põhimenüüs vaadates selleks mõeldud nupu peale paari sekundi jooksul.

Joonis 7. Google Cardboard

3.3.8 Disain

Mängu olid lisatud sellised disainiomadused, mis on reeglina igas mängus, nagu helid, animatsioonid ja GUI.

Animatsioonid on olemas paljudel objektidel. Esiteks on animeeritud seina peal (Joonis 8) asuvate suurte nupude vajutamised. Lisaks seinal asuvad paneel tasemevalimiseks ja juhtimisrežiimi valimise nupp, millised on samuti animeeritud. On olemas nupp mängu alustamiseks, mängu sulgemiseks ning nupp heli sisse- ja väljalülitamiseks.

Kirjeldatud GUI elemendid asuvad põhimenüüs. Mänguväljaku seinal (Joonis 8) lisaks heli nupule asuvad nupud taseme uuesti alustamiseks ja üleminekuks põhimenüüsse. Kui ehitis on lammutatud, ilmub nupp järgmisele tasemele üleminekuks. Seina peal on olemas ka informatsioon teksti kujul: teenitud punktid lammutamise eest, tehtud laskude arv ja välja on toodud ka protsent, mille võrra on ehitis lammutatud. Seina paremas osas asub rekorditabel kolme reaga. Igas reas on kirjutatud punktide ja laskude arvud. Tabelit värskendatakse iga sekundi tagant.

Joonis 8. GUI põhimenüüs (üleval) ja mänguväljal.

On loodud kuulide ilmumise, heitemasina lasu hetke ja tuulelippu animatsioonid. Klaasplokid purunevad väikseteks tükkideks ja tavalised muutuvad aeglaselt nähtamatuteks. On olemas plahvatuste animatsioonid. Taseme lõppu on lisatud võiduefekt.

Helid on lisatud nupude vajutustele, heitemasina laskudele, kuulide kokkupõrgetele teiste objektidega ja plahvatustele. On olemas mängija sammude heli, heitemasina pööramise ja liikumise helid.

3.4 Olulisemad Unity komponendid

Edasi on kirjeldatud enim kasutatud komponente *editor*'is ja koodis, mida pakub välja Unity rakenduse arendamiseks.

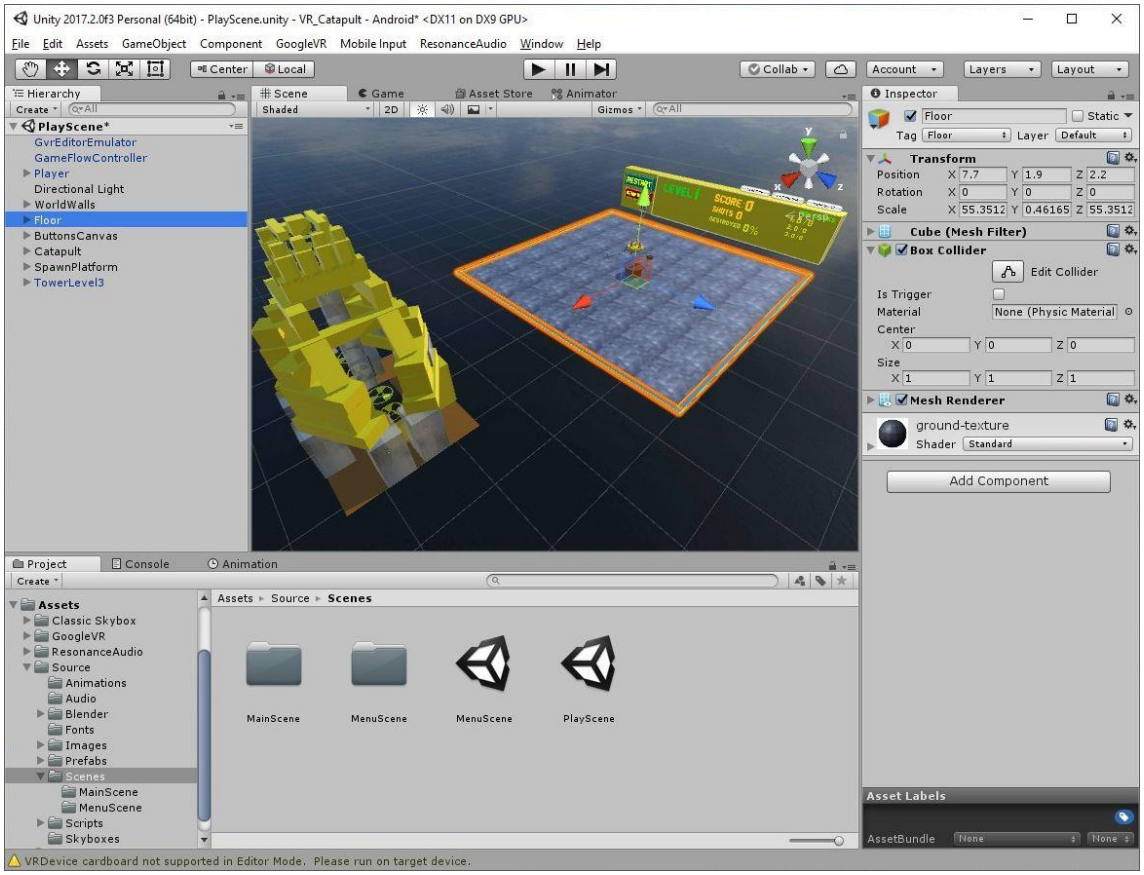
3.4.1 Mänguobjekt

Mänguobjekt (ingl. *Game object*) on konteiner, mis sisaldab informatsiooni VR maailma objekti kohta. Mänguobjektile on võimalik lisada tekstuuri ja muuta objekti suurust või selle saab jätta nähtamatuks ja kasutada näiteks helide esitamiseks. Mänguobjekt võib sisaldada teisi mänguobjekte ja komponente. Mänguobjekti

komponendiks nimetatakse faili programmikoodiga ehk **skripti**, mis lisab eriomadused ning mõjutab mänguobjekti käitumist.[11]

3.4.2 Stseen

Stseen (ingl. *Scene*) on mänguobjektidest koosnev konteiner, mis samuti sisaldab informatsiooni relatsioonidest objektide vahel. Unity *editor*'is arendaja lisab stseenile mänguobjekte ning nendele võib omakorda lisada komponente (Joonis 9). Mänguprotsessi jooksul mängija asub alati ühel stseenidest.[12] Kui on olemas mitu stseeni, siis muuta aktiivset stseeni on võimalik skripti kaudu, näiteks selleks, et mängija läheks menüüst mänguväljale.



Joonis 9. Unity *editor*

3.4.3 Transform komponent

Transform komponent on kõige levinum mänguobjekti komponentidest, sest ta on olemas igal mänguobjektil. Antud komponent sisaldab informatsiooni objekti asetuse kohta füüsikalises ruumis. Muutes selle komponendi omaduste väärtuseid on võimalik mänguobjekti **keerata**, **nihutada** ja **muuta** objekti **suurust**.[13]

VR rakendustes antud komponenti kasutades imiteeritakse pealiigutusi, ja nimelt **kaamera** objekti asetuse muutmist. Kaamera[1] (ingl. *Camera*) on kohustuslik mänguobjekt igal stseenil, sest selle objekti kaudu mängija näeb rakendust.

3.4.4 Rigidbody komponent

Rigidbody[2] komponent lisab mänguobjektile mõned füüsikalised omadused. Peamised nendest on **mass** ja **gravitatsiooni mõjutus**. Antud komponent aitab imiteerida reaalmaailma objektide käitumist ja on olemas paljudel liikuvatel objektidel mängus, näiteks heitemasina kuulidel.

3.4.5 Collider komponent

Collider komponenti kasutatakse selleks, et mänguobjektil oleks **füüsilised piirid**. Sellega antud komponent samuti määrab objekti **füüsilist kuju**: kera, kuup jms. See komponent **registreerib kokkupõrke** mänguobjektide vahel ja annab võimaluse skripti kaudu sellele reageerida. Kokkupõrke registreerimiseks kahe objekti vahel lisaks *Collider* komponendile kasutatakse ka *Rigidbody* komponenti, sellest tulenevalt on sageli mänguobjektidele lisatud mõlemad komponendid.[14]

Antud komponenti kasutatakse mängus peaaegu kõikjal näiteks selleks, et mängija ei kukuks mänguväljast läbi või ploki ja kuuli vahelise kokkupõrke registreerimiseks.

3.4.6 Animator komponent

Animator[3] komponent lisab mänguobjektile animatsioonideloendi, mida esitatakse alati või kutsutakse skriptist välja. Animatsiooni loomine ja lisamine loendisse on Unity's lihtne: animatsiooni paneeli peal vajutatakse salvestusnuppu, seatakse ajahetk ja stseeni peal muudetakse animeeritava objekti omaduste väärtused, edasi vajutatakse nuppu salvestuse peatamiseks ja animatsioon salvestatakse loendisse.[15]

Mänguobjekti omaduste väärtused animatsiooni esitamise jooksul on konstantsed ja neid seadistatakse animatsiooni loomisel. Sellega seoses ei ole mugavat võimalust teha

---

[1] https://docs.unity3d.com/Manual/Cameras.html

[2] https://docs.unity3d.com/Manual/RigidbodiesOverview.html

[3] https://docs.unity3d.com/Manual/class-Animator.html

omaduste väärtuseid dünaamilisteks näiteks selleks, et kuul liiguks sujuvalt mängija kätte. Sellisel juhul on vaja teha animatsiooni koodi abil.

3.4.7 Event trigger komponent

Event trigger komponent võimaldab mänguobjektidel registreerida sündmused, mis on seotud **sisendiga** kasutaja poolt, näiteks hiireklõps objekti peal (*OnPointerClick*[1]). Rakendustes antud komponenti kasutatakse tavaliselt GUI elementide realiseerimiseks.[16]

Kasutavates Google VR SDK rakendustes *Event trigger k*omponent lisatakse samuti mänguobjektidele, millistega kasutaja saab suhelda, ning kasutatakse koos *GvrReticlePointer*[2] komponendiga, mis omakorda vastutab sihik-kursori eest. Registreeritakse selline sündmus, kui ring visuaalselt läheb objekti peale (*OnPointerEnter*[3]), tänu millele ring hakkab suurenema ja kasutaja saab teada, et on võimalik suhelda antud mänguobjektiga.

3.4.8 Coroutine

Coroutine[4] kasutatakse ainult koodis ja see on mugav mehhanism erinevate tegevuste täitmiseks üheaegselt. *Coroutine* funktsioonid on sarnased *thread*'itega[5], kuid esimesi täidetakse **asünkrooniliselt** ning teisi eraldi vooludes. Antud mehhanism võimaldab reguleerida funktsioonide täitmist ning ajahetki täitmiste vahel. Koodis see on *IEnumerator*[6] tüüpi funktsioon. [17]

Rakendustes *coroutine* on võimalik kasutada mõne tegevuse täitmiseks teatud ajaperiiodi tagant. Näites (Joonis 10) *coroutine* funktsioon värskendab tabeliandmeid iga paari sekundi tagant, kuni muutuja *bool stopUpdating* on väärtusega *false*.

---

[1] https://docs.unity3d.com/ScriptReference/EventSystems.IPointerClickHandler.OnPointerClick.html

[2] https://developers.google.com/vr/unity/reference/class/GvrReticlePointer

[3] https://docs.unity3d.com/ScriptReference/EventSystems.IPointerEnterHandler.OnPointerEnter.html

[4] https://en.wikipedia.org/wiki/Coroutine

[5] https://www.techopedia.com/definition/27857/thread-operating-systems

[6] https://msdn.microsoft.com/en-us/library/system.collections.ienumerator

```
IEnumerator UpdateTable() {
   while (true) {
      UpdateText();
      yield return new WaitForSeconds(2);

      if (stopUpdating)
         yield break;
   }
}
```

Joonis 10. *Coroutine* funktsiooni näidis

Mängus antud mehhanism samuti kasutatakse dünaamiliste väärtustega animatsioonide loomiseks, mis on ebamugav realiseerida Unity animatsioonide abil.

3.5 Probleemid ja lahendused

Selles osas tutvutakse probleemidega, mis võivad tekida VR mängude arendamisel ning millele Unity ega Google VR SDK valmis lahendusi ei pakku. Samuti kirjeldatakse ka leitud lahendusi.

3.5.1 Mängija liikumise normaliseerimine

Üks esimestest asjadest, mida arendajal tuleb hakata realiseerima mängus, kus mängijal on võimalus liikuda, on mängija objekti liikumine.

VR maailmas mängija liikumissuund sõltub tavaliselt sellest suunast, kuhu mängija parasjagu vaatab. Algne üldine realisatsioon on lihtne: kaamera objekti *Transform* komponendist võetakse kolmemõõtmiline **vektor**[1], mis näitab suunda, kuhu on kaamera objekti esikülg pööratud (*forward*[2]), edasi selles suunas liigutatakse mängija objekti, mis sisalsab endas kaamera objekti. Mängija liikluskiiruse reguleerimiseks kasutatakse muutuja *float speed* ning see sõltub kontrolleri *joystick*'u asendist (Joonis 11).

[1] Vector3 on andmetüüp punkti koordinaadide esitamiseks, mis hoiab X, Y, Z väärtused (https://docs.unity3d.com/ScriptReference/Vector3.html)

[2] https://docs.unity3d.com/ScriptReference/Transform-forward.html

```
float speed = Input.GetAxis("Vertical");
Vector3 direction = camera.transform.forward;
player.transform.position += direction * speed;
```

Joonis 11. Mängija liikumine

Sellel viisil mängija objekt liigub samas suunas, kuhu vaatab kasutaja. Niisugune liikumine ei sobi, kui mängijal on määratud liikuda ainult XZ tasandil, sest mängijal on võimalus liikuda ülesse ja alla. Siin üheks võimalikeks lahendusteks võiks olla vektori **Y** väärtuse **nulliks** muutmine. Selline lahendus eemaldab võimaluse lennata ja liikuda mänguväljast läbi, kuid tekib veel üks probleem: kui mängija vaatab otse ülesse või alla, siis ta ei hakka liikuma üldse.

Tekkinud probleemi lahendamiseks on vaja võtta *forward* vektorit teiselt mänguobjektilt, mis oleks pööratud nagu kaamera objekt, kuid ilma kallutusteta ülesse ja alla. Eemaldada kallutused on võimalik Unity poolt pakutud funktsiooniga *Quaternion.LookRotation*[1]. Koodis ei ole võimalik teha *Transform* komponendi koopiat, sest see komponent peab alati olema mänguobjektile lisatud. Sellega on vaja stseenile lisada **uus objekt**, mida **pööratakse sarnaselt kaameraga** skripti kaudu ning millelt võetakse sobiv vektor. Koodis (Joonis 12) uus objekt on nimega *compass*.

```
Vector3 forward = Camera.main.transform.forward;
forward.y = 0;
compass.transform.rotation = Quaternion.LookRotation(forward);

Vector3 direction = compass.transform.forward;
player.transform.position += direction * speed;
```

Joonis 12. Normaliseeritud mängija liikumine

Mängus on realiseeritud liikumine igas suunas XZ tasandil. Selleks kasutatakse lisaks vektor *transform.right*[2], mis näitab paremat poolt mänguobjektilt.

[1] https://docs.unity3d.com/ScriptReference/Quaternion.LookRotation.html

[2] https://docs.unity3d.com/ScriptReference/Transform-right.html

3.5.2 Objekti liikumisala piiramine

VR mängu arendamise jooksul võib tekkida vajadus mõne objekti liikumisala piiramiseks. Antud mängul on vaja piirata heitemasina liikumist, et mängijal puuduks võimalus tõugata heitemasinat mänguväljade piiridest välja.

Esialgul oli katse kasutada mõnes mõttes selleks ettenähtud komponente: lisada heitemasina põhiosale *Rigidbody* ja *Collider* komponente ning luua mänguvälja piirides pardaid, et heitemasin ei liiguks nendest läbi. Selline meetod sobiks, kui heitemasinal ei oleks olnud imporditud mudelit Blender'ist, sest Unity mängumootori realisatsiooni tõttu **ei ole võimalik hoida** samal objektil üht objekti ***Rigibody*** komponendiga ja teist objekti ***MeshCollider***[1] komponendiga. Ilma *MeshCollider* komponendita oleks kuulide jaoks süvendi füüsikaline kuju vale ja asendatud kuul kukkuks maha põrandale.

Lõpuks oli otsustatud muuta heitemasina asukohta **koodis** ja samal kohal **piirata** liikumisala. Skriptis (Joonis 13) on võimalik pärast heitemasina liikumist kontrollida, kas heitemasina põhiosa piirid lõikuvad mänguvälja parrastega või mitte, kasutades selleks Unity poolt pakutud funktsiooni ***Bounds.Intersects***[2], ning kui jah, siis liigutada heitemasinat eelmisele kohale tagasi.

```
Vector3 prevPos = catapult.position;
catapult.position += direction * speed;

foreach(Collider border in borders) {
   if(this.collider.bounds.Intersects(border.bounds)) {
      catapult.position = prevPos;
      break;
   }
}
```

Joonis 13. Heitemasina liikumisala piiramine *Intersects* funktsiooni abil

Antud meetod töötab, aga mitte alati: tekkivad olukorrad, kui heitemasin jääb mõnda pardasse kinni ning ei ole võimalik edasi heitemasinat kasutada.

Tänu sellele, et mänguväli on **ristkujuline** on olemas sobivam meetod liikumisala piiramiseks. On vaja fikseerida minimaalsed ja maksimaalsed mänguvälja **nurkade**

[1] https://docs.unity3d.com/Manual/class-MeshCollider.html

[2] https://docs.unity3d.com/ScriptReference/Bounds.Intersects.html

koordinaadid ning heitemasina liikumisel **ümardada** järgmise asukoha vektori väärtused nurkade piirides.

```
float Clamp(float value, float min, float max) {
   if(value < min)
      return min;
   if(value > max)
      return max;
   return value;
}

void Move() {
   Vector3 nextPos = catapult.position + direction * speed;
   nextPos.x = Clamp(nextPos.x, minX, maxX);
   nextPos.z = Clamp(nextPos.z, minZ, maxZ);
   catapult.position = nextPos;
}
```

Joonis 14. Heitemasina liikumisala piiramine ümardamisega

Nurkade koordinaatide leidmine on lihtne Unity poolt pakutud funktsioonidega ning näites (Joonis 14) on nad esitatud muutujatega *minX, minZ, maxX, maxZ.* Antud meetod töötab stabiilselt, aga ainult ristkujuliste ja kuupkujuliste liikumisaladega ning sellest piisab antud mängu puhul.

3.5.3 Dünaamiline suhtlemisdistants mänguobjektidega

Mängus kõik GUI nupud asuvad suure seina peal. Mugavama kasutamise jaoks on suhtlemisdistants GUI nupude ja mängija vahel piiramatu. Mäng oleks liiga lihtne, kui mängija saaks võtta kuulid või vajutada heitemasina nupule igalt kohalt ja sellepärast on vaja piirata suhtlemisdistants selliste objektidega.

Kasutusele võetud raamistik Google VR SDK pakkub komponendi sihik-kursori jaoks – *GvrReticlePointer*. Seda komponenti kasutades on võimalik distantsi väärtus ette anda, millega pildi keskel asuv ring „näeb“ objekte ehk hakkab suurenema. Mehhanism, mis võimaldaks anda ette erinevaid distantse erinevate objektide kohta, puudub ning etteantud distants töötab kõikide mänguobjektide jaoks.

Vajaliku mehhanismi realiseerimiseks oli vaja kirjutada kaks skripti. Esimene skript *InteractionTarget* (Joonis 15) pärib *EventTrigger* komponendist ning seal toimub funktsioonide ***OnPointerEnter/OnPointerExit*** ümberdefineerimine selleks, et registreerida sündmused, kui sihik-kursor läheb objekti peale ja läheb temalt ära. Antud

komponent lisatakse kõikidele mänguobjektidele, millistega on võimalik suhelda: nupudele ja kuulidele. Esimese sündmuse puhul kontrollitakse ja võib olla piiratakse suhtlemisdistants ja teise sündmuse puhul pannakse maksimaalne suhtlemisdistants paika.

```
class InteractionTarget : EventTrigger {
   override void OnPointerEnter(PointerEventData data) {
      InteractionController.CheckInteractionDistance();
   }

   override void OnPointerExit(PointerEventData data) {
      InteractionController.EnableInteraction();
   }
}
```

Joonis 15. Esimene komponent dünaamilise suhtlemisdistantsi jaoks

Teine skript realiseerib suhtlemisdistantsi piiramist ja seal samas kasutatakse ***GvrReticlePointer*** komponenti.

```
class InteractionController {
   float normalGrowthSpeed;
   float initialDistance;
   GvrReticlePointer pointer;

   void CheckInteractionDistance() {
      GameObject target = pointer.RaycastResult.gameObject;
      if (target.layer != GameConstants.IDX_LAYER_UI) {
         float distance = Vector3.Distance(target.transform.position,
                                           camera.transform.position);

         if (distance > GameConstants.INTERACTION_DISTANCE) {
            StartCoroutine(DisablePointer());
            return;
         }
      }
      EnableInteraction();
   }

   IEnumerator DisableInteraction() {
      pointer.maxReticleDistance = GameConstants.INTERACTION_DISTANCE;
      while (pointer.ReticleInnerDiameter > 0.0002f)
         yield return new WaitForFixedUpdate();

      pointer.maxReticleDistance = initialDistance;
      pointer.reticleGrowthSpeed = 0;
   }

   void EnableInteraction() {
      pointer.reticleGrowthSpeed = normalGrowthSpeed;
   }
}
```

Joonis 16. Teine komponent dünaamilise suhtlemisdistantsi jaoks

Näites (Joonis 16) on lühendatud skripti versioon. Google VR SKD raamistik võimaldab teada saada, mis mänguobjekti peal sihik-kursor asub ja see kasutatakse esimeses funktioonis kontrollimiseks, kas mänguobjekt on GUI nupp või tavaline objekt. Objekte võib kontrollida erinavalt, näiteks nimiosa või *tag*'i[1] järgi. Üks parematest võimalustest GUI nupude eraldamiseks on nende lisamise eraldi **kihisse**[2] (ingl. *Layer*) ning kihi järgi kontrollitakse objekte esimeses funktsioonis. Kui mänguobjekt ei kuulu GUI nupude hulka, siis **kontrollitakse** mängija ja objekti vahelist

[1] https://docs.unity3d.com/Manual/Tags.html

[2] https://docs.unity3d.com/Manual/Layers.html

distantsi ning kui see on rohkem kui lubatud, kutsutakse suhtlemise väljalülitust ehk teist funktsiooni välja.

Teise funktsioonina kasutatakse *coroutine*, et koodi täitmine toimuks asünkrooniliselt. Esialgul pannakse **piiratud distants** ette. Piiratud distantsi tõttu sihik-kursor rohkem „ei näe“ objekti ja hakkab vähenema ning c*oroutine* funktsioon **ootab**, kuni ringi sisediaameter saab **väikseks**. Edasi pannakse ette **maksimaalne suhltemisdistants**, aga ringi **suurenemiskiirust** muudetakse **nulliks**, et sihik-kursor hakkaks objekti „nägema“, kuid samal ajal ei suureneks. Mänguobjekt saab niimodi registreerida sündmust, kui ring läheb objektilt ära, et lülitada suhtlemine sisse. Suhtlemise sisselülitamine toimub lihtsalt: pannakse ette tavaline ringi suurenemiskiirus, mis annab ringile normaalset käitumist.

3.5.4 Erinevad tegevused Google Cardboard nupuga

Selleks, et juhtimine Google Cardroard’iga oleks sarnane juhtimisega kasutades *Bluetooth* kontrollerit, on vaja lisada võimalust teha ühe nupuga järgmised tegevused: ***action*** – kuuli võtmine või nupule vajutamine ning **liikumine** igas suunas erineva kiirusega.

Oli otsustatud teha *action* nuppu vajutamisel ja alustada liikumist, kui nupp oli hoitud all lühikese ajaperioodi jooksul. Skriptis **registreeritakse** nupu **allhoidmist** ja kutsutakse *coroutine* funktsiooni välja (Joonis 17).

```
bool preparingToMove = false;
IEnumerator PrepareToMove() {
   if (!preparingToMove) {
      preparingToMove = true;
      float preparingTime = 0.35f;
      while ((preparingTime -= Time.deltaTime) > 0f) {
         if (Input.GetButtonUp(GameConstants.ACTION_BUTTON)) {
            OnCardboardClick();
            preparingToMove = canMove = false;
            yield break;
         } else {
            yield return new WaitForFixedUpdate();
         }
      }
      canMove = true;
      preparingToMove = false;
   }
}
```

Joonis 17. *Coroutine* funktsioon liikumise alustamiseks

Näites *coroutine* ootab umbes pool sekundit ja pärast seda lubab alustada liikumist. *Time.deltaTime*[1] näitab, kui palju aega sekundites kestis eelmine kaadrivärskendus mängus. Kui ootamise ajal VR prillide nupp oli vabastatud (*Input.GetButtonUp*[2]) siis see peetakse **vajutamiseks** ja kutsutakse välja tegevus, nagu oleks vajutatud nupu kontrolleri peal.

Liikumissuuna defineerimiseks VR rakendustes on võimalik kasutada **peakallutusi**. Kui mängija vaatab alla või otse, siis ta liigub otse, ning kui ta suunab oma pilgu kõrgemale, toimub liikumine tagasi. Kui mängija kallutab oma pea paremale või vasakule, siis ta hakkab liikuma kõrvale. Mida rohkem on **kaldenurk**, seda rohkem on **liikumiskiirus**. Informatsiooni kallutuse kohta on võimalik saada **kaamera** objekti ***Transform*** komponendilt. Selleks oli tehtud mitu funktsiooni. Näites (Joonis 18) on lihtsustatud kood peamisest funktsioonist liikumiseks.

[1] https://docs.unity3d.com/ScriptReference/Time-deltaTime.html

[2] https://docs.unity3d.com/ScriptReference/Input.GetButtonUp.html

```
private void TryToMove() {
   if (Input.GetButton(GameConstants.ACTION_BUTTON)) {
      if (!canMove) {
         StartCoroutine(PrepareToMove());
         return;
      }
      Vector3 direction;
      bool left = false;
      if (MovingSideward(out left)) {
         direction = compass.transform.right;
         if (left)
            direction *= -1;
      } else {
         direction = compass.transform.forward;
         if (MovingBack())
            direction *= -1;
      }
      transform.position += direction * speed * speedCoeff;
   } else if (canMove && Input.GetButtonUp(GameConstants.ACTION_BUTTON)) {
      canMove = isMoving = false;
   }
}
```

Joonis 18. Peamine funktsioon liikumiseks

Pärast seda, kui esimeses *coroutine* funktioonis oli liikumine lubatud, kasutatakse abifunktsiooni (*bool MovingSideward(bool out left)*), mis tagastab *true*, kui mängija kallutab pea kõrvale. See funktsioon annab lisaks väärtust sisendparaametrile, mida kasutatakse peakallutuse suuna defineerimiseks: paremale või vasakule. Abifunktsioonis puuduvad erilised mehhanismid, seal võrreldakse kaamera objekti lokaalset **kaldenurka** Z-teljel konstantse väärtustega: 10-85 liikumiseks vasakule ja 275-350 paremale. Kui mängija ei kallutanud pead kõrvale, siis teises abifunktsioonis (*bool MovingBack*) võrreldakse kaldenurka X-teljel ning kui ta on 275 kuni 350 kraadini, siis toimub liikumine tagasi. Mõlemates abifunktsioonides antakse lisaks väärtust **kiiruskoefitsiendile**: mida lähemalt on kaldenurk 0 (360) kraadile, seda vähem on kiirus.

Antud meetodiga Google Cardboard'i peal asuva nupuga mängija saab võtta kuulid ja vajutada nupudele ning liikuda igas suunas erineva kiirusega. Sellel hetkel antud meetod katab kõik juhtimisvõimalused mängus. Kui tulevikus tekib vajadus lisada veel tegevusi, siis nende eraldamiseks võib kasutada vajutamise arvu või kombineerida tavaline vajutus peakallutustega.

3.6 Skriptid

Edasi on kirjeldatud olulisemate skriptide funktsionaalsust. Praegusel momendil mängus kasutatakse 33 skripti autorilt, ligikaudne üdlmaht on 2450 rida.

PlayerController – skript mängija objekti jaoks. Seal on realiseeritud mängija liikumine ja suhtlemine objektidega.

InteractionController ja InteractionTarget – skriptid, mis realiseerivad dünaamilist suhtlemisdistantsi mänguobjektidega.

GameFlowController – skript mänguseisu reguleerimiseks. Seal toimub aktiivsete stseeni ja taseme muutmine.

DataManager – skript püsivate mänguandmete salvestamiseks ja lugemiseks. Selliste andmete hulka kuuluvad tasemete rekordid, juhtimisrežiim ja heliseis. Kõik andmeid kirjutatakse mälusse binaarses formaadis[1] ning taastatakse mängu käimapanekul.

DefaultBlock, ExplosiveBlock ja GlassBlock – skriptid ehitise plokkide jaoks. Seal registreeritakse kokkupõrke plokke ja mänguvälja või nähtamatu põranda vahel, misjärel plokkid purunevad. Teises skriptis on lisaks realiseeritud plahvatuse mõjutus naaberplokkidele. Skriptide loomisel kasutati klasside pärimist ja polümorfismi[2].

UIButton – skript nupudele. Kasutatakse sellistel GUI elemenditel nagu nupud mängu alustamiseks või sulgemiseks ja lisaks kujutab endast põhiskripti teiste nupude skriptide jaoks. Antud skriptis on loodud kolm virtuaalset[3] funktsiooni, mis kutsutakse vajutamisanimatsioonist välja.

ScoreCounter – skript selliste mänguandmete arvestuseks nagu punktid lammutamise eest, laskude arv ja lammutamise protsent. Seal samuti toimub tekstikujulise informatsiooni värskendamise mängimise jooksul.

---

[1] msdn.microsoft.com/en-us/library/system.runtime.serialization.formatters.binary.binaryformatter

[2] https://www.techopedia.com/definition/3235/object-oriented-programming-oop

[3] https://www.techopedia.com/definition/24299/virtual-method

CatapultEngine, CatapultMovement, CatapultRotation – skriptid heitemasinale. Esimesel on realiseeritud heitemasina lask, teisel ja kolmandal liikumine ja pööramine.

BallBehaviour – põhiskript kuulide jaoks. Seal toimub kokkupõrke registreerimine teiste objektidega. Mehhanism, mis võimaldab kuuli võtta ja lahti lasta. Lisaks on koodiga realiseeritud animatsioon, et kuul liiguks sujuvalt mängija kätte.

WindVane – antud skriptis on realiseeritud animatsioonid tuulelippu jaoks. Seal on samuti informatsioon tuulekiiruse ja suuna kohta, mida värskendatakse iga poole minuti tagant.

3.7 Tulemuse analüüs

Teise osa tulemuseks Unity mängumootori abil oli loodud mäng nutitelefonidele Android operatsioonisüsteemiga. Mängida on võimalik kasutades VR prille nutitelefonidele koos *Bluetooth* kontrolleriga või ainult Google Cardboard prille.

Mängus on realiseeritud dünaamiline suhtlemisdistanst erinevate objekti hulkade jaoks. Koodi abil on heitemasina liikumisala piiratud. *Coroutine* funktsioonidega on realiseeritud animatsioonid dünaamiliste väärtustega.

Antud hetkel on loodud 6 taset. On lisatud animatsioonid ja helid ning realiseeritud rekordite salvestamine.

Ettevalmistatud disainimustrite puuduse tõttu arendamise jooksul kulutati kaunis palju aega disaini loomiseks ja veel rohkem ümbertegemiseks, eriti erinevate ehitiste loomisel. Sellel põhjusel oli otsustatud edaspidi planeerida ja kujundada keerulise disainiga mänguelemente „paberi peal“, misjärel alustada loomist mängumootoris. Samuti kulus palju aega oma 3D mudelite modelleerimiseks, sest varem asjakohane kogemus puudus. Sellega võib öelda, et produktiivse mänguarendamise jaoks disaineri olemasolu projektis on kohustuslik.

Oli tehtud järeldus, et isegi väiksete projektide puhul on kasulik luua teste funktsionaalsuse jaoks, kuna võib juhtuda, et arendaja ei märka alles tekinud viga ja laeb koodi repositooriumi ülesse, millepärast tulevikus kulutatakse aega veakoha ülesotsimiseks.

Edaspidi planeeritakse muuta disaini: lisada paremaid tekstuure ning kirjeldust mõne mänguobjektide kohta. Samuti planeeritakse luua uusi tasemeid ja laadida mäng Google Play ülesse.

3.7.1 Mängumootori Unity hinnang

Unity mängumootor ei ole keeruline arusaamiseks ja sobib suurepäraselt programmerijatele, kellel puudub mängude arendamise kogemus.

Üks olulisematest tugevatest külgedest on see, et puudub vajadus mõelda sihtplatvormide erinevusele ja sellega luua fail installeerimiseks Android, iOS või Windows platvormidele on võimalik mitme vajutusega.

Primitiivsete kolmemõõtmiliste mudelite loomiseks on võimalik hakkama saada Unity poolt pakutud mudelitega. Kui tekkib vajadus luua keerukamat mudelit, siis tuleb kasutada eraldi modellerimis tarkvara ja importida mudelid Unity projekti.

Olulisemate stseenide ja koodi komponendite kasutama õppimiseks ei kulu palju aega, kui arendajal enne Unity kasutamist oli programmerimis kogemus C# keeles.

Üldjuhul antud mängumootorist on jäänud ainult positiivsed muljed ning see sobib mugavaks VR rakenduste arendamiseks.

4 Kokkuvõte

Lõputöö eesmärkideks olid tutvustamine VR teemaga ja kirjutada VR mäng kasutades Unity mängumootorit.

Töös olid kirjeldatud olulisemad printsiibid VR rakenduste kasutamisel, meetodid VR maailma objektidega suhtlemiseks ning populaarsemad VR platvormid. Lisaks oli antud ülevaade kahest populaarseimast mängumootorist VR rakenduste arendamiseks.

Praktilise osa tulemuseks oli loodud valmis kasutamiseks mäng nutitelefonidele. Olid kirjeldatud enim kasutatud mängumootori komponendid, mängudisain ja kontseptsioon. Olid käsitletud peamised probleemid, mis olid tekinud arendamise käigus, ning kirjeldatud rakendatud lahendusi. Mängus on realiseeritud mehhanismi erinevate tegevuste täitmiseks kasutades üht Google Cardboard'i peal asuvat nuppu. On olemas juhtimisrežiim *Bluetooth* kontrollerile teiste VR prillidega mängimiseks.

Autori arvates kõik töös püstitatud ülesanded on täidetud. Tekinud probleemidele lahendused olid üles leitud. Olid väljatoodud peamised vead arendamisprotsessis ning põhjendatud Unity sobivus VR rakenduste arendamiseks. Loodud mäng on unikaalne ning mugavalt mängitav kõige levinumate VR prillidega nutitelefonidele.

Kasutatud kirjandus

[1] What is Virtual Reality? [WWW] https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html (04.04.2018)

[2] Understanding Sensors: Magnetometers, Accelerometers and Gyroscopes [WWW] https://www.vrs.org.uk/virtual-reality-gear/motion-tracking/sensors.html (04.04.2018)

[3] Head-mounted Displays (HMDs) [WWW] https://www.vrs.org.uk/virtual-reality-gear/head-mounted-displays (05.04.2018)

[4] VR Controllers Archives [WWW] https://www.roadtovr.com/category/vr-controllers (05.04.2018)

[5] UI Interaction in mobile Virtual Reality [WWW] https://productcoalition.com/ui-interaction-in-mobile-virtual-reality-4-reasons-to-not-use-the-touch-pad-on-samsungs-gear-vr-6b1c86a8f140 (14.04.2018)

[6] Best game engines for Virtual Reality development [WWW] https://www.slant.co/topics/2202/~game-engines-for-virtual-reality-development (14.04.2018)

[7] Unity Game Engine Review [WWW] https://www.gamesparks.com/blog/unity-game-engine-review (15.04.2018)

[8] What is Unreal Engine 4 [WWW] https://www.unrealengine.com/en-US/what-is-unreal-engine-4 (15.04.2018)

[9] Should I Learn Unreal or Unity for VR Development [WWW] http://www.vudream.com/should-i-learn-unreal-or-unity-for-vr-development (18.04.2018)

[10] Google VR SDK [WWW] https://developers.google.com/vr/develop (18.04.2018)

[11] Unity - Manual: GameObjects [WWW] https://docs.unity3d.com/Manual/GameObjects.html (18.04.2018)

[12] Unity - Manual: Scenes [WWW] https://docs.unity3d.com/Manual/CreatingScenes.html (18.04.2018)

[13] Unity - Manual: Transforms [WWW] https://docs.unity3d.com/Manual/Transforms.html (20.04.2018)

[14] Unity - Manual: Colliders [WWW] https://docs.unity3d.com/Manual/CollidersOverview.html (20.04.2018)

[15] Unity - Manual: Creating a New Animation Clip [WWW] https://docs.unity3d.com/Manual/animeditor-CreatingANewAnimationClip.html (20.04.2018)

[16] Unity - Skripting API: EventTrigger [WWW] https://docs.unity3d.com/ScriptReference/EventSystems.EventTrigger.html (21.04.2018)

[17] Unity - Manual: Coroutines [WWW] https://docs.unity3d.com/Manual/Coroutines.html (21.04.2018)