TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Merli Lall 142392 IAPB

VIRTUAALREAALSUSE MÄNGU LOOMINE UNREAL ENGINE 4 MÄNGUMOOTORIS

Bakalaureusetöö

Juhendaja: Eduard Petlenkov
professor

Tallinn 2019

Autorideklaratsioon

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

Autor: Merli Lall

9.01.2019

Annotatsioon

Antud lõputöö peamiseks eesmärgiks oli luua virtuaalreaalsuse kogemust edasiandev mäng, mille käigus õpiti mängumootori Unreal Engine 4 võimalusi. Antud lahendust võib pidada eelmainitud mängumootori õppematerjaliks ühe konkreetse virtuaalreaalsuse mängu näitel.

Mäng loodi tellija (Swedbank) kirjelduse alusel Mektory virtuaalreaalsuslaboris. Arendajateks olid 3 tudengit. Lõputöö autori ülesanneteks oli luua mängutasemete kujundus ja programmeerida mängu töötamiseks vajalikud mehhanismid: mänguloogika, interaktiivsete objektide loogika, mängukarakter, mängutasemete vahetuse tingimused, punktide kogumise võimalus.

Tulemuseks loodi virtuaalreaalsuse mäng, mis kasutab peakomplekti HTC Vive, mille pulte saab mängija kasutada sisendi andmiseks ning asukoha muutmiseks ruumis. Mängija ülesandeks on korjata määratud aja jooksul võimalikult palju objekte, millel on kindlad punktiväärtused. Mängu õppimiseks on loodud eraldi mängutase. Mäng on edukalt kasutuses kõikides Balti riikides virtuaalreaalsust tutvustava rakendusena. Eestis on mängu kasutatud üritustel nagu näiteks Robotex, Võti Tulevikku ja Teadlaste öö.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 51 leheküljel, 9 peatükki, 50 joonist, 2 tabelit.

Abstract

Creation of a Virtual Reality Game using Unreal Engine 4

The purpose of this thesis project was to create a virtual reality video game experience. Along the developing process the author learned the aspects of the Unreal Engine 4 game engine, of which insights are also presented in the thesis. The solution can be considered as a learning material on one specific example of how to create virtual reality game in Unreal Engine 4.

The game was designed based on the client's idea of a virtual reality experience around one of the company's symbols represented by an oak tree. The game was developed by 3 students of Tallinn University of Technology in the Mektory Virtual and Augmented Reality Laboratory. The author's tasks in the team were level design and programming the game logic using the Unreal Engine 4 blueprint scripting language. The game logic had to cover all the necessities of a functioning game: general flow of the game, creating interactive objects, game pawn, level switching logic and implementation of game point acquisition system (scoring).

The result was a virtual reality game which uses the HTC Vive headset for the input and output interfaces. The player's objective is to collect as many objects as possible within the given time limit. Each object has a specific point value. One of the most important decisions during development was putting a virtual time counter on the player's wrist thus making use of the interactive potential of the virtual reality. In this case the countdown did not disturb the player while the player always had an option to check the remaining time. Audio cues were additionally given at specific moments to remind the user of the remaining time. Therefore, the 3D virtual reality user interfaces should exist within the game space and use more senses to provide a more realistic experience.

It was also concluded that if the player has an objective within the defined space it is less important for the player to find out what happens outside of the defined area.

Therefore, less foliage was used in the level design in order to gain higher frames per second rate of the game.

A tutorial level was created before the actual game in order to learn about the mechanics without the time limit. The main game experience was enhanced greatly by the development of the tutorial level. However, the instructional aspect could be improved by presenting the user with less information and having interactive guides or even an artificial intelligence companion as the guide. The development of the main level and tutorial level is described along with the Unreal Engine 4 blueprints.

This virtual reality game successfully performs its task. A working system has been created to allow the player to register, learn and successfully complete the game.

The game is being used in all Baltic countries as an introductory application for virtual reality by Swedbank. In Estonia, the game has been used in larger events like Robotex, *Võti Tulevikku* and Night of Scientists.

The thesis is in Estonian and contains 51 pages of text, 9 chapters, 50 figures, 2 tables.

Lühendite ja mõistete sõnastik

Actor	Objekti asetamist või ilmumist mängumaailma võimaldav *blueprint* kogumik [1].
Blueprint	Funktsioonikastikestest koosnev visuaalse skriptimiskeele võimalus Unreal Engine 4 mängumootoris [2]. Antud mõiste võib esindada ka *blueprint* kogumikku ehk klassi (*Blueprint Class*), mida arendaja võib olemasolevale mängufunktsionaalsusele juure lisada [1].
Collision Box	Kolmemõõtmeline kast, mis vajadusel teavitab, kui miski mängu sees satub tema piirialale või sisse.
GameInstance	Globaalselt kättesaadav objekt, kuhu saab talletada muutujaid üle mitme mängutaseme. Luuakse mängu alguses ning eemaldatakse mängu lõppedes [3].
HMD *(Head Mounted Display)*	Peas kantav ekraan ehk virtuaalreaalsuse prillid
HTC Vive	Firma HTC poolt toodetud virtuaalreaalsuse kogemust pakkuv peakomplekt (*headset*) koos HMD, pultide ja neid jälgivate majakatega.
Hz	Herts
Level design	Mängutasemete planeerimine ja kujundamine
MotionController	*Actor*, mis saab kasutajapoolse sisendi vastu võtta, võimaldab interaktsiooni virtuaalreaalsuses mängumaailmaga [4]. Antud projektis visuaalseks representatsiooniks on mannekeeni käsi.
Mängukarakter	*Actor,* mida mängija saab juhtida. Unreal Engine 4 mängumootoris kutsutakse seda nimetusega *Pawn* [1].
Mängumootor	Arvutimängude loomiseks mõeldud kogumik erinevate tööriistade ja komponentidega.
Mängutase	Arendaja defineeritud mänguala, mis Unreal Engine 4 mängumootoris salvestatakse *.umap* failivormingus. Eelmainitud mängumootoris kutsutakse seda nimetustega *Level* kui ka *Map* [5], [6].
Static Mesh	Staatiline hulknurkadest koosneva kolmemõõtmelise objekti ümbris ehk staatiline polügonaalvõre.
Teleportation	Asukoha muutmine ruumis
Template	Mall, näidis. Antud lõputöös mängumootori poolt pakutud näidis virtuaalreaalsuse mängutaseme loomeks.
UE4	Unreal Engine 4

VR	Virtuaalreaalsus
VRPawn	VR mängukarakter, millel on virtuaalreaalsuse kogemise otstarbeks loodud atribuudid ning mis saab sisendväärtused *MotionController* poolt.
2D	Kahemõõtmeline
3D	Kolmemõõtmeline

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Ülemaailmset virtuaalreaalsuse turgu hinnati 2017. aastal 3,13 miljardile USA dollarile ja prognooside kohaselt on see 2023. aastaks 49,7 miljardit USA dollarit. Mitmed rahvusvahelised korporatsioonid nagu Sony ja HTC osalevad selles tururuumis [7]. Seoses virtuaalreaalsust pakkuvate seadmete tarbijaturule jõudmisega on tekkinud küsimus, mis tarkvara nendele seadmetele luua. Tallinna Tehnikaülikooli Mektory laborisse pöördus Swedbank sooviga luua tudengite kaasabil virtuaalreaalsuse kogemust pakkuv mäng. Eesmärgiks oli luua võistlusliku sisuga mäng, kus mängija saab looduslikus keskkonnas piiratud aja jooksul koguda punkte, mis kantakse punktide edetabelisse. Lisaülesandeks oli luua virtuaalreaalsuse õpetustuba, kus kasutaja õpiks mängu mängima ja virtuaalreaalsust kasutama. Mängumootoriks valiti Unreal Engine 4 (UE4).

Töö teises peatükis selgitatakse, mis on virtuaalreaalsus. Kolmandas peatükis antakse täpsem taustinfo projekti nõuete ja kasutavate seadmete kohta. Neljandas peatükis selgitatakse mängumootori valikut ning antakse lühitutvustus olulistest mõistetest. Viiendas peatükis lahendatakse arendusprotsessis tekkinud probleeme seoses mängumootori visuaalse skriptimiskeele (*blueprint*) struktureerimisega redaktoris, kasutajaliidestega virtuaalreaalsuses ning mängumaailma loomes. Kuuendas peatükis selgitatakse õpetustoa loomise protsesse võttes aluseks viienda peatüki informatsiooni. Töö seitsmendas peatükis esitatakse tulemused. Kaheksandas peatükis tulemusi analüüsitakse ning on pakutud projekti edasiarendamise ideed ja nõuanded virtuaalreaalsusmängu loomiseks.

2 Mis on virtuaalreaalsus

Virtuaalreaalsus (VR) on arvutiga genereeritud maailm, mis jätab kasutajale mulje, et ta on tegelikult selle genereeritud maailma sees. Eestlaste seas kutsutakse seda ka tehistegelikkuseks, tehistõelisuseks, võltsreaalsuseks, küberreaalsuseks [8]. Virtuaalreaalsuse mõiste tekkis 1960. aastate alguses ning seda saab jagada kaasahaaravaks (*immersive)* ja mittekaasahaaravaks (*non-immersive)*. Mittekaasahaarav VR on arvutiga genereeritud keskkond, mis simuleerib reaalseid või väljamõeldud kohti, millele saab ligi arvuti ja tavapärase kuvariga. Täielik ehk kaasahaarav VR loob kohalolekutunde simuleeritud keskkonnas ning nõuab lisaseadmete, peamiselt peas kantava ekraani (HMD – *head mounted display*) ja teisi keha asukoha jälgimisseadmete kasutamist [9]. Jälgimisseadmeteks on hetkel levinumad peakomplektis kaasas olevad käes hoitavad puldid ja HMD. Loodud on ka seadmeid, mida saab kinnitada objektide külge, mida soovitakse virtuaalreaalsuses jälgida [10]. Teiste meelte sidumine on samuti oluline, kuid hetkel suudetakse kõige paremini virtuaalreaalsusega siduda nägemis- ja kuulmismeelt.

Kaasahaarava virtuaalreaalsuse saab jagada kaheks: passiivseks (*passive*) ja interaktiivseks (*interactive*) virtuaalreaalsuseks. Passiivses virtuaalreaalsuses toimuvad tegevused juhitult kellegi teise, näiteks tehisintellekti poolt ning kasutaja ei saa ise maailmas peale jälgimise midagi muud teha. Tehisintellektiks võib olla näiteks virtuaalreaalsusesse programmeeritud robot [11]. Interaktiivses virtuaalreaalsuses saab kasutaja maailmas ringi liikuda, objekte käsitseda ning maailma ise oma panusega muuta [11, lk. 3]. Antud kirjelduste nimekiri ei ole lõplik, sest tegemist on areneva valdkonnaga ning aja jooksul võivad kujuneda uued viisid, kuidas virtuaalreaalsust paremini kirjeldada. Näiteks on loodud uuringuid ja soovitusi, mis käsitlevad erinevate VR keskkondade loomist [12]. Antud lõputöös luuakse kaasahaaravat interaktiivset virtuaalreaalsuse kogemust andvat mängu.

3 Projekti taustinfo

Juulis 2017 alustas projekti autor virtuaalreaalsuses mängitava mängu tegemist. Mängu loomisele aitasid kaasa veel kaks tudengit, kes tegelesid mänguprogrammi alustava kahemõõtmelise (2D) registreerimisvaatega, punktitabelite ja kolmemõõtmeliste (3D) objektide loomisega. Kahel teisel tiimiliikmel oli olemas eelnev kogemus modelleerimisprogrammiga Autodesk Maya ning mängumootoriga Cry Engine 5, kuid mängumootor UE4 oli kõigile arendajatele uus. Seega lisaks mängu loomisele pidi õppima ka uut tarkvara kasutama.

Arvuti, mille peal Mektory VR laboris projekti arendati ja testiti, omas põhiparameetreid:

- Graafikakaart GeForce GTX 980Ti
- Protsessor Intel i7 6700K
- Muutmälu (RAM) 32 gigabaiti

Tellija otsustas mänguliideseks kasutada arvutiga ühendatud VR kogemust pakkuvat peakomplekti HTC Vive, mille koosseisu kuulusid kasutaja sisendit edasiandvad puldid, HMD ning peakomplekti riistvara jälgivad majakad. Mänguala suuruseks sooviti kliendi poolt 4 ruutmeetrit, sest peakomplekt võimaldab realiseerida ruumipinda kasutavat kogemust.

Arendamisele eelnes tellijapoolne üleskutse luua VR mäng nii turunduse kui ka uute tehnoloogiate, eelkõige virtuaalreaalsuse tutvustamise eesmärgil, lisaks arendada ka kompetentsi VR rakenduste loomise alal.

3.1 Esialgne kirjeldus tellija poolt

Mektory VR laborisse esitati Swedbanki poolt 5. märtsil 2017. aastal esialgne mängu kirjeldus. Idee teostus ning lisaomadused jäid tiimi otsustada. Lisaks võis tiimiliige,

kellele anti konkreetne ülesanne, läheneda ülesandele oma oskustele ja teadmistele vastavalt.

Kliendi kirjelduse kohaselt määrati mängule esialgsed nõuded:

- Mängukeskkond peab jäljendama loodust.
- Mängukeskkonnas peab olema tammepuu.
- Mängu põhiülesanne peab toimuma tammepuu läheduses.
- Virtuaalsetel tammelehtedel peab olema korjamisvõimalus.
- Mängijal võiks olla võimalus keskkonnas ajatult ringi vaadata.
- Mängijal võiks olla võimalus võistelda teiste mängijatega.
- Mängul peab olema töötav punktisüsteem.
- Mängul peab olema ajaline piirang.

Arendustiim võis teha ülejäänud valikud, mis tulid hiljem kooskõlastada tellijaga. Tiimiga arutledes jõuti otsusele, et mäng tuleb eelkõige siiski võistlusliku eesmärgiga. Seega puudus mängus otsene võimalus ainult ringi vaadata, sest selline tegevus oleks pikendanud mängu ajalist piirangut. Hiljem otsustati arenduse käigus õpetustoas pultide tundmaõppimiseks anda selline ajatu võimalus, sest see oli hilisemast pärismängust eraldi olev mängutase, kuid siiski VR mängu kogemuse oluline osa. Tellija jäi selliste valikutega rahule.

4 Unreal Engine 4 tutvustus

Mängumootorite kasutamine lihtsustab mängu loomise protsessi ning annab arendajale juba läbi töötatud võimalusi mängu optimeerimiseks. Antud lõputöös kasutati virtuaalreaalsuse mängu loomiseks mängumootorit Unreal Engine 4 (UE4). Projekti loomist alustati 2017. aastal, mil valiti versioon 4.16 ning uuendati hiljem versioonile 4.17. Järgnevalt otsustas autor välja tuua valiku põhjenduse ning lõputöö arusaadavuse huvides mõned olulised mõisted ja selgitused.

4.1 Valiku põhjendus

Üheks alternatiiviks oleks võinud olla Cry Engine 5. Lõputöö autoriga samas projektis olid veel kaks tudengit, kes tegid enda lõputöö kasutades Cry Engine 5 mängumootorit, mille loomise protsesse on kirjeldatud nende tiimiliikme töös [13]. Hiljem tekkinud tarkvara töötamise probleemid ei andnud soovitud tulemust. Seda mängumootorit ei soovitanud ka projektijuht. Seetõttu tundus lõputöö autorile UE4 töökindlam variant.

Teiseks lahenduseks oleks sobinud Unity mängumootor. Unity pakub samuti palju võimalusi virtuaalreaalsuse mängu loomiseks. Aastal 2017 puudus aga Unity mängumootoril pakutud näidis virtuaalreaalsuse mängutaseme loomeks (*template).* Antud võimalust tutvustati alles 2018. aasta alguses [14]. UE4 mängumootoril oli virtuaalreaalsuse *template* projekti loomise ajal olemas, seega tundus valitud mängumootor kasutajasõbralikum.

Projektijuht otsustas samuti UE4 kasutamise poolt. Lisaks on seda eelnevalt kasutatud edukate lõputööde loomisel Mektory VR laboris, näiteks Martin Luige lõputöös [15]. Nendel põhjustel peeti käesolevat mängumootorit sobilikuks valikuks.

4.2 Erinevad redaktorid

Valitud mängumootor on komplektne, sisaldades võimalusi objekte luua, kuvada, paigutada mängumaailma ning nendele ülesandeid programmeerida. Erinevate redaktoritega saab lahendada konkreetseid funktsionaalseid ülesandeid.

UE4 terminite tõlkimiseks eesti keelde on võimalusel aluseks võetud Thomas-Bairam Toodo 2016. aastal valminud bakalaureusetöö [16]. Eelmainitud töös oli otsustatud mõisteid *blueprint* ning redaktorite nimesid eesti keelde mitte mugandada. Lisaks ei tõlkinud käesoleva lõputöö autor objekti asetamist või ilmumist mängumaailma võimaldavat *blueprint* kogumikku (*actor)*. Autor pidas lahendust sobilikuks, sest säilib mõiste arusaadavus ning on vajadusel erinevatest allikatest lugejale lihtsamini leitav.

Järgnevalt on välja toodud mõned olulisemad redaktorid ning nende lühikirjeldused, mis on vajalikud lõputöö lahenduste mõistmiseks.

4.2.1 Level Editor

Selles redaktoris saab mängutaset luua, muuta ja täiendada. Selle jaoks asetatakse *actor* elemendid sobilikele kohtadele kasutades selleks ettenähtud tööriistu. Mängutasemele saab juurde lisada ka loogikat kasutades selleks erinevaid *blueprint* võimalusi.

Antud projektis on loodud kolm mängutaset: kasutaja registreerimine, õpetustuba ning põhimäng. Esimene mängutase on autori tiimikaaslaste loodud. Kahe viimase mängutaseme loogika ja kujundus on lõputöö autori poolt tehtud. Hilisemates peatükkides selgitatakse täpsemalt kahe viimase mängutaseme loomet.

4.2.2 Material Editor

Materjali (*material*) elemendid seotakse mängumaailma objektiga, et anda sellele värv ja näiline ruumiline sügavus (*normal map*). Antud vaadet kasutatakse materjali elementide loomiseks või muutmiseks, pakkudes selleks vajaminevaid tööriistu [17].

Selle vaate kasutamise roll jäi mitme tiimiliikme kasutusse. Lõputöö autor pidi seda vaadet kasutama maapinna mustri loomiseks ning objektidele *Level Editori* sees värvi varieeruvuse tagamiseks, sest oli vaja tagada mängule looduslähedane välimus. Antud redaktor oli seetõttu oluline visuaalse poole parendamiseks.

4.2.3 Blueprint Editor

Tegemist on põhilise redaktoriga, mida antud lõputöö kirjeldab. Selles redaktoris saab kasutaja luua ja muuta *blueprint* elemente, mis juhivad erinevaid aspekte mängu juures. Tegemist on visuaalse skriptimiskeele võimalusega, mida kasutatakse nii prototüüpimisel kui ka terviklike arvutimängude loomisel. *Blueprint* elemente saab kasutada mitmel otstarbel ja seetõttu võib antud redaktorit kohata mängumootori erinevates kohtades [18]. Näiteks saab redaktorit kasutada kasutajaliidestes, objektidele interaktiivsuse loomiseks, animatsioonide käivitamiseks, mängutegelase loogikas ja paljude muude funktsionaalsuste tagamiseks.

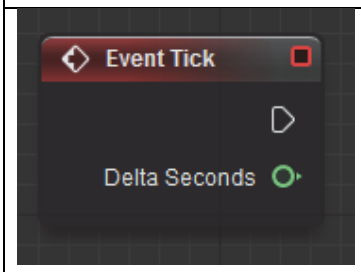
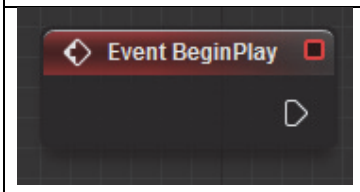
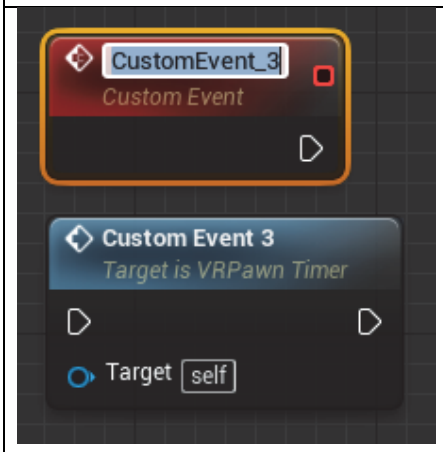
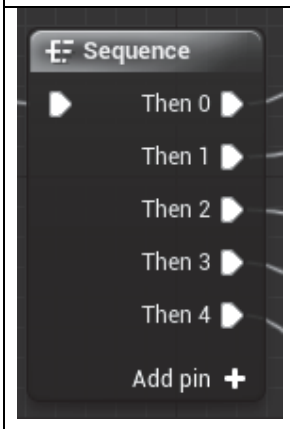
Blueprint Editor kasutab funktsioonikastikesi, seega tekstiredaktorit kasutavate programmeerimiskeeltega võrreldes näevad *blueprint* muutujad visuaalselt teistsugused välja. Visuaalsel *Blueprint* skriptimiskeelel esinevad puhta koodi (*clean code*) probleemid nagu igal programmeerimiskeelel. Arenduse käigus tuli teha palju muudatusi, mis parandasid *blueprint* elementide loetavust ja loogilist ülesehitust.

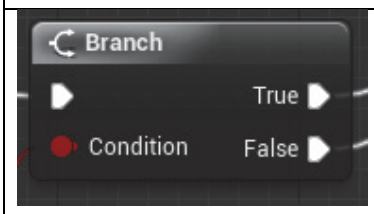
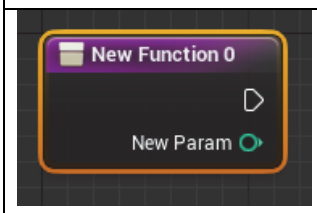
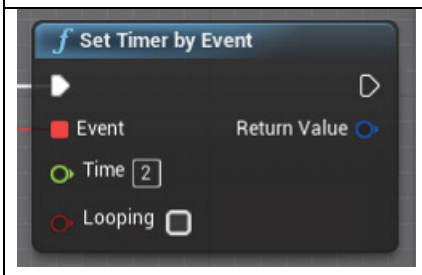
Blueprint elementide loetavuse parandamiseks ning funktsioonide loogilisemate üleskutsete käivitamiseks võttis autor aluseks koodi refaktoreerimise põhitõed [19]. Lisaks sai otsitud infot *blueprint* elementide refaktoreerimise kohta, kus muidu tuntud puhta koodi probleemidele leiti, et funktsioonikastikeste vedamisel tekib lugejale liigselt arusaamatuid jooni [20]. Sellisel teemal allikaid leidub hetkeseisuga vähe ning seega pidi lähtuma vähesest, mis saadaval on.

Antud lõputöös kasutatakse nii funktsioone kui ka sündmusi (*event*) teatud tegevuste läbiviimiseks. Sündmustel ja funktsioonidel on erinevusi. Sündmuste jaoks saab kasutada ajast sõltuvaid elemente nagu näiteks viivitus (*delay)* ja ajajoon *(timeline).* Funktsioonide puhul selliseid elemente välja kutsuda ei saa. Funktsioonid saavad tagastada väärtusi, kuid sündmused mitte. Seega sõltuvalt eesmärgist kasutati ühte või teist lahendust.

Antud lõputöö arusaadavuse huvides on järgnevalt toodud selgitav tabel, mis viib vastavusse UE4 *blueprint* funktsioonikastikesed, nende mõisted ning võimalusel sarnasused üldtuntud programmeerimismõistetega (Tabel 1).

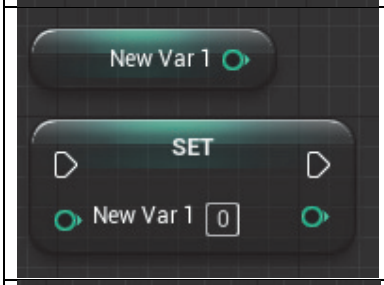
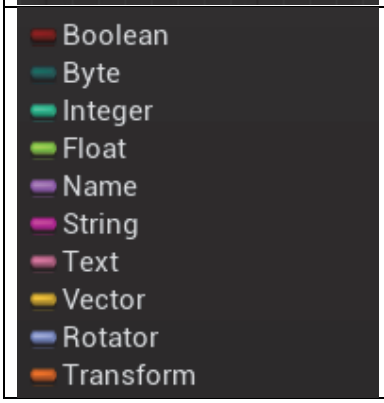
Tabel 1. *Blueprint* elementide selgitav tabel

***Unreal Engine 4* element**	**Nimetus**	**Tähendus**
	*Event Tick*	Sündmus, mis kutsutakse välja iga uue kaadri puhul.
	*Event BeginPlay*	Sündmus, mis kutsutakse välja *blueprint* jooksutamise alguses üks kord.
	*Custom Event*	Arendaja poolt loodud sündmuse defineerimine, millele saab anda oma nimetuse ja tegevustiku [21]. Allpool on selle väljakutsumine klassis *VRPawn_Timer*. Joonisel on sündmuse nimeks *CustomEvent_3*.
	*Sequence*	Element, mida kasutatakse paralleeltegevuste loomiseks. Tegevused, mis klemmidest väljuvad, käivitatakse samaaegselt.

***Unreal Engine 4* element**	**Nimetus**	**Tähendus**
	*Branch*	Element, mis realiseerib *if*-tingimust. Tingimuse (*condition)* elemendiks võetakse kahendväärtus (*boolean)*, mille tõesel väärtusel jätkatakse tegevusi *True* klemmis ning eitava väärtuse korral *False* klemmis.
	*AND Boolean*	Element, mida kasutatakse kahe või mitme kahendväärtuse konjunktsioonis. Sarnased elemendid on ka OR ja XOR.
	*NewFunction0*	Näide kasutaja poolt loodava funktsiooni *NewFunction0* loomisest. Antud juhul on funktsiooni sisendiks *NewParam*.
	*SetTimerByEvent*	Näide funktsiooni väljakutsest. Antud juhul kutsub funktsioon teatud aja (*time*) peale välja sündmuse (*event*) ning seda saab vajadusel korduma panna (*looping*).

Järgnevalt on välja toodud projektis kasutatud muutujad ning nende vastavus programmeerimiskeeltes tuntud mõistetega (Tabel 2).

Tabel 2. Muutujate tabel

***Unreal Engine 4* muutuja**	**Tähendus**
	Kahendväärtuse (*boolean*) poole pöördumine (*get*) ja määramine (*set*). Linnuke tähistab tõese väärtuse määramist. Joonisel on muutuja nimeks *NewVar_0*.
	Täisarv (*integer*) poole pöördumine (*get*) ja määramine (*set*) arvulise väärtusega 0. Joonisel on muutuja nimeks *NewVar_1*.
	*Actor* objekti poole pöördumine (*get*) ja määramine (*set*). Joonisel on muutuja nimeks *NewVar_2*.
	*Blueprint* kogumiku (klassi) poole pöördumine (*get*) ja määramine (*set*). Joonisel on muutuja nimeks *NewVar_3*.
	Ülejäänud muutujad neile vastavate indikaatorvärvidega. Muutujate poole pöördumine ja määramine toimub analoogselt eelmainitutega.

5 Virtuaalreaalsuse mängu loomine

Lõputöö autori ülesanne oli programmeerida mängu loogika: punktide kogumine ja liitmine, objektidega suhtlemine, mänguaja piiramine, ühest mängu tasemest teise üleminekud, lisaks sellele ka mängualade planeerimine ja kujundamine (*Level Design*) kahes mängutasemes. Järgnevates peatükkides ongi täpsemalt välja toodud antud VR mängu loomise protsess. Tulemusi näeb peatükis 7, kus on samuti välja toodud ka mängu eduka läbimise tegevusdiagramm (vt Joonis 45, lk 52).

Virtuaalreaalsuse tarkvara kasutamine ning arendamine on tõusutrendis. Uuringute kohaselt oli 2017. aastal ülemaailmne virtuaalreaalsuse tarkvara turu suurus 0,6 miljardit USA dollarit, mida 2019. aastaks ennustatakse 4,4 miljardi USA dollarini [22]. Projekti alustamise ajal oli oluliselt vähem virtuaalreaalsust kasutavat tarkvara. Enamus edukaid tarkvarasid olid tasulised ning seetõttu autorile kättesaamatud. Põhiliseks inspiratsiooniks sai mängumootori enda poolt pakutud näidis (*template*), mida on täpsemalt kirjeldatud järgnevates alapeatükkides.

5.1 Projekti algusfaas

Projekti valmimise efektiivsuse huvides otsustas autor kasutada VR *template* võimalust, mida UE4 versioon 4.16 esialgse projekti loomisel pakkus. *Template* võimaldas realiseerida projekti loomisfaasil sisend-ja väljundliideste planeerimist, mis on oluline interaktiivse VR rakenduste loomisel [12, lk. 239]. Mängumootor pakkus selleks ise oma lahenduse. Näiteks tuli *template* kaasa oma VR mängukarakteri *(VRPawn*) varaga (*asset*)*,* millel oli olemas:

- asukoha muutmise ruumis (*teleportation)* võimalus;
- mängija kätt imiteeriv visuaalne mannekeen koos animatsioonidega;
- kaardistatud funktsionaalsed nupud (*trigger*, *trackpad*).

Lisaks oli *template* valik kasulik mängumootori tundmaõppimiseks, sest oli võimalus uurida, kuidas on teatud komponendid omavahel seotud. Sellega sai komponente vajadusel täiendada või luua uusi eelnevate põhjal. *Template* lahendusega tuli samuti kaasa palju soovitatud seadistusi, mis olid arendajale kasulikud, sest tagasid mängu kohese kasutuskõlblikkuse.

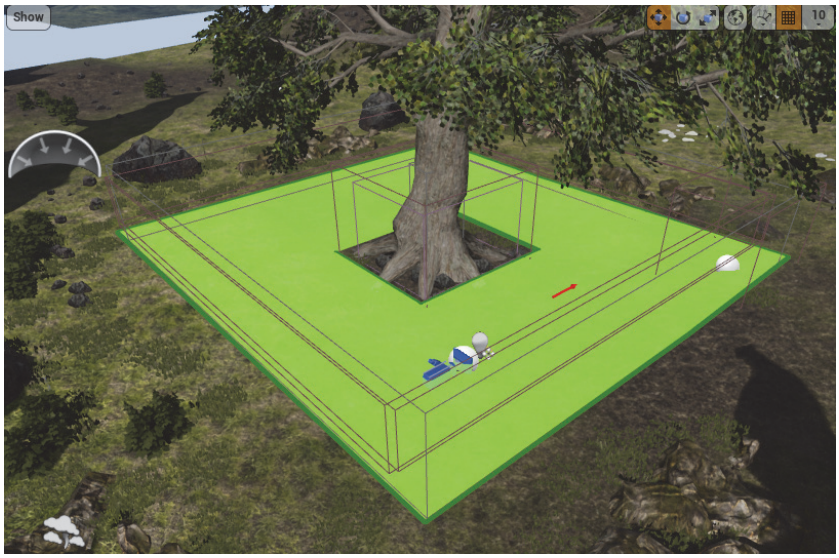
Antud projektis alustas autor VR mängu loomist põhimängust, sest esialgu ei planeeritud õpetavat mängutaseme ruumi teha. Hiljem loodi õpetustuba kasutades põhimängus olevaid objekte. Kasutaja registreerimise mängutase jäi teise tiimiliikme ülesandeks, kuid andmete kättesaamine ja uuendamine lahendati globaalselt kättesaadava objekti abil, kuhu sai talletada muutujaid üle mitme mängutaseme (*GameInstance*). Seetõttu oli autorile oluline projekti algusfaasis mängu põhifunktsionaalsuse saavutamine.

5.2 Põhilise mänguala loomine

UE4 virtuaalreaalsuse *template* sisaldab kahte mängutaset, millel on erinevad funktsionaalsused ja nõuded arvutiriistvara osas. Põhimängu loomise inspiratsiooniks sai mängutase nimega *MotionControllerMap*, mis toetas projekti jaoks kasutatava peakomplekti HTC Vive riistvara [23]. Lisaks oli antud mängutase hea näidis mänguala piiritlemise, pultidele funktsionaalsuse tagamise ning objektidega interaktsiooni loomise osas. Antud *template* olemasolu lihtsustas oluliselt eelmainitud protsesse ning võimaldas tegeleda rohkem mängu sisuga. Tegemist oli taaskasutatava ning soovitatud ressursiga mängumootori poolt, seega otsustas projekti autor teha mängutasemest duplikaadi ning luua mängu jaoks olulised erispetsiifikad pakutud raamistiku peale.

Põhimängu loomiseks eemaldati *template* mängutasemelt kõik ebavajalikud elemendid ning loodi uus maastiku (*landscape)* element *Level Editori* tööriistadega. Teine võimalus oleks olnud luua uus mängutase ning teha vajaminevate komponentide üle toomine (*migration*), mis oleks andnud sama tulemuse. Esimene variant oli vähemate sammudega, sest lisaks komponentidele on oluline ka näiteks valgustuse ning muude parameetrite korrektsus, mis esimese variandi puhul jäid alles ilma lisaseadistusteta.

Tiimiliikmete poolt loodud 3D objektid asetati lõputöö autori poolt *Level Editori* tööriistu kasutades põhimängu maastikule. Mängijale liikumiseks lubatud mänguala määrati tammepuu objekti alla (Joonis 1).



Joonis 1. Põhimängus mängijale lubatud liikumisala. Tähistatud erkrohelisega.

Level Design puhul pidi arvestama piirialadele objektide (kivid, põõsad) lisamist, et mängijal kaoks soov minna mängualast kaugemale uudistama. Valgustuse puhul tuli silmas pidada. et VR mängud on väga ressursimahukad ja seetõttu pidi kaadrisageduse suurendamiseks välja lülitama kõik dünaamilised ja enamus tavavarjud kõikidelt objektidelt. Jõudluse parandamiseks tuli näiteks kasutada 3D objektide asemel kauguses ka 2D objekte ning need paigutada mängumaailma selliselt, et jätaksid ruumilise mulje.

Arenduse ja modelleerimise protsessid toimusid paralleelselt. Vajamineva objekti puudumisel asendas autor selle näidisega, kuni modelleeritud objekt valmis sai. Selline lähenemine võimaldas samaaegselt arendada mänguloogikat.

Piiriala sisse loodi *actor* element, mis genereerib puulehti. Keerukuse tõttu on selle loomist kirjeldatud alapeatükis 5.6.

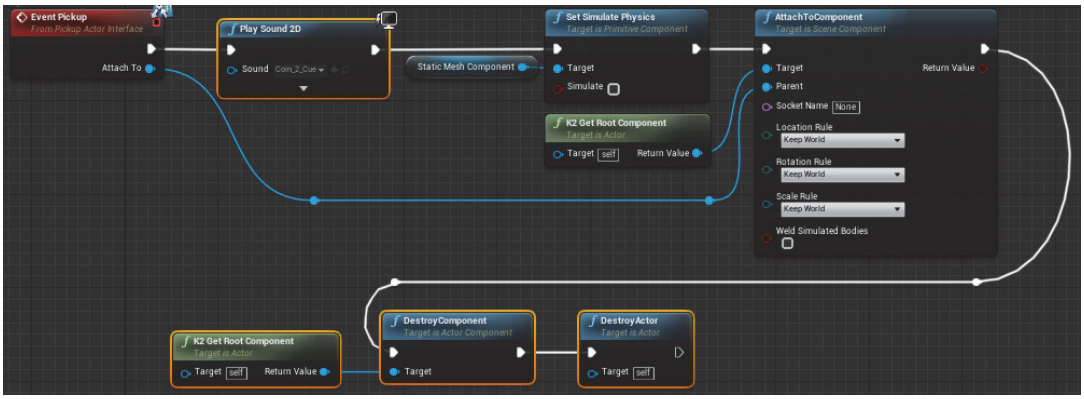
5.3 Interaktiivsete objektide loomine

Mängutase *MotionControllerMap* oli abiks interaktiivsete objektide loomisel. Nimelt leidus seal *blueprint* nimega *BP_PickupCube*, mis oli korjatav tänu korjamise

sündmusele (*Event Pickup*). Lehtede ja müntide loomiseks tehti eelmainitud objektist duplikaat ja muudeti 3D objekti ümbris (*Static Mesh)* tiimiliikme poolt loodud lehe või mündi omaks. Atraktiivsuse huvides lisati objektidele pööramisefekt (*RotatingMovement*), et need ei oleks välja kutsudes statsionaarselt paigal. Loodi 4 objekti, mida mängus aja peale korjata saab:

- Münt;
- Kollane tammeleht;
- Punane tammeleht;
- Roheline tammeleht.

Loodud objektidelt eemaldati *BP_PickupCube* poolt kaasa tulnud objekti vabastamise sündmus (*Event Drop)*, sest ei leidnud antud mängus rakendust. Korjamise sündmusele lisati funktsionaalsust juurde. Sündmusele lisati üleskorjamise heli (*PlaySound2D*). Lisati kaks uut funktsiooni, et mängija saaks pärast objekti üleskorjamist uusi objekte korjata. Selleks kasutati funktsioone *DestroyComponent,* et mannekeeni käsi (*MotionController)* realiseeriks objektist kinni võtmise animatsiooni ja *Destroy Actor*, et kasutaja poolt korjatud *actor* registrist kustutada (Joonis 2).



Joonis 2. *Event Pickup* arendus. Kollase kastiga on märgitud lisatud funktsionaalsused.

5.4 Mängukarakteri loomine

Template elemente kasutusele võttes loodi *VRPawn* duplikaat, et mängija saaks realiseerida kõiki peatükis 5.1 kirjeldatud võimalusi. *VRPawn* kasutab virtuaalreaalsuses mängumaailmaga interaktsiooni võimaldavaid sisendeid (*MotionController*). Muudetud

duplikaadile lisati funktsioone juurde. Üheks põhiliseks lisafunktsiooniks oli muutujate lisamine, mis tegelesid mänguaja piiramisega. Teiseks lisati erinevad mängijat informeerivad elemendid kaamera külge. Nendeks elementideks olid esialgselt:

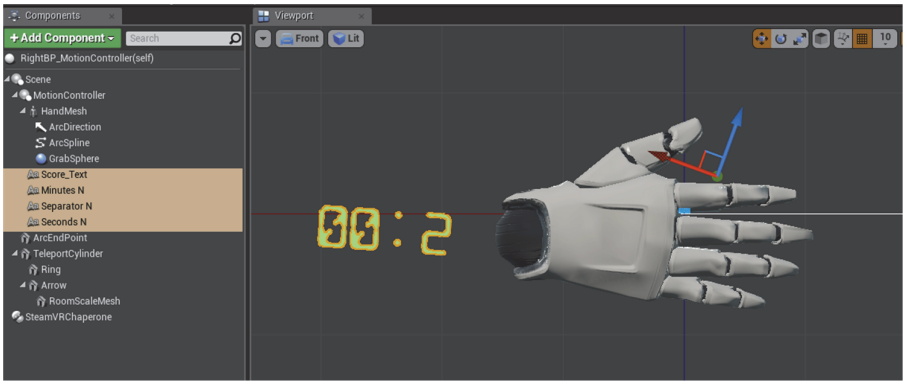
- Ajalised parameetrid (minutid, sekundid);
- Mängu algust selgitav tekst;
- Mängu lõppu selgitav tekst;
- Lõplik punktiskoor.

Pärast esimest iteratsiooni saadi kliendilt tagasiside, et ajaliste parameetrite kaamerasse kuvamine ei olnud virtuaalreaalsuse omadusi arvestades ja mängija heaolu arvestades sobiv lahendus. Mängija ülesanne oli mängus pidevalt ringi vaadata, seega ei saanud kasutajaliidese puhul pidada mõistlikuks sekundite kuvamist mängija vaatevälja kaamerasse. Seepärast tuli ülesanne lahendada teisiti.

Tavaliste programmeerimiskeeltega sarnaselt on võimalik *blueprint* elementidega anda sama funktsionaalust edasi mitmeti. Järgnevalt on loodud võimalikud variandid antud mängus VR kogemuse parendamiseks.

5.4.1 MotionController arendus

Võrreldes 2D *desktop* rakendustega imiteerib VR 3D maailma ning seega tuli autoril idee kuvada mänguaja sekundeid mitte kaameras, vaid mängija *MotionController* küljes, kus mängijale antakse võimalus aega vaadata justkui käekellalt (Joonis 3).



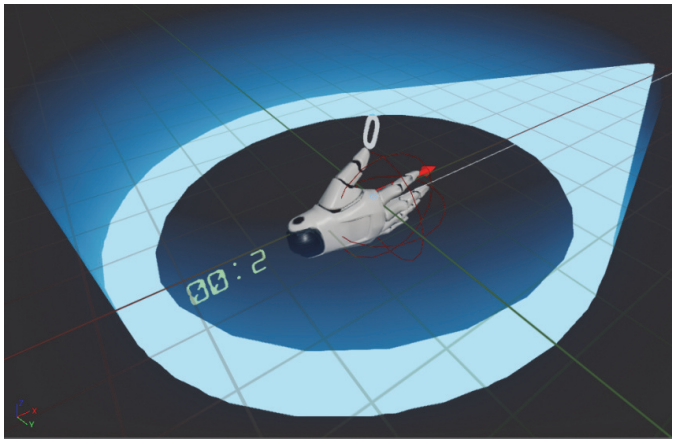
Joonis 3. *MotionController* muudetud duplikaat täiendatud muutujatega (vasakul kollasega) ja visuaalne representatsioon paremal. Pildilt on peidetud elemendid, mida muutma ei pidanud, välja arvatud mannekeeni käsi.

Seega *VRPawn* muudetud duplikaadile lisati kaamera külge järgnevad komponendid:

- Mängu algust selgitav tekst;
- Mängu lõppu selgitav tekst;
- Lõplik punktiskoor;
- Viimased sekundid.

MotionController muudetud duplikaadi külge seoti komponendid (Joonis 4):

- Ajalised parameetrid (minutid, sekundid) randmel;
- Ühe käega kogutud punktide arv sõrme juures.



Joonis 4. *MotionController* duplikaat kõikide elementidega. Visuaalne representatsioon.

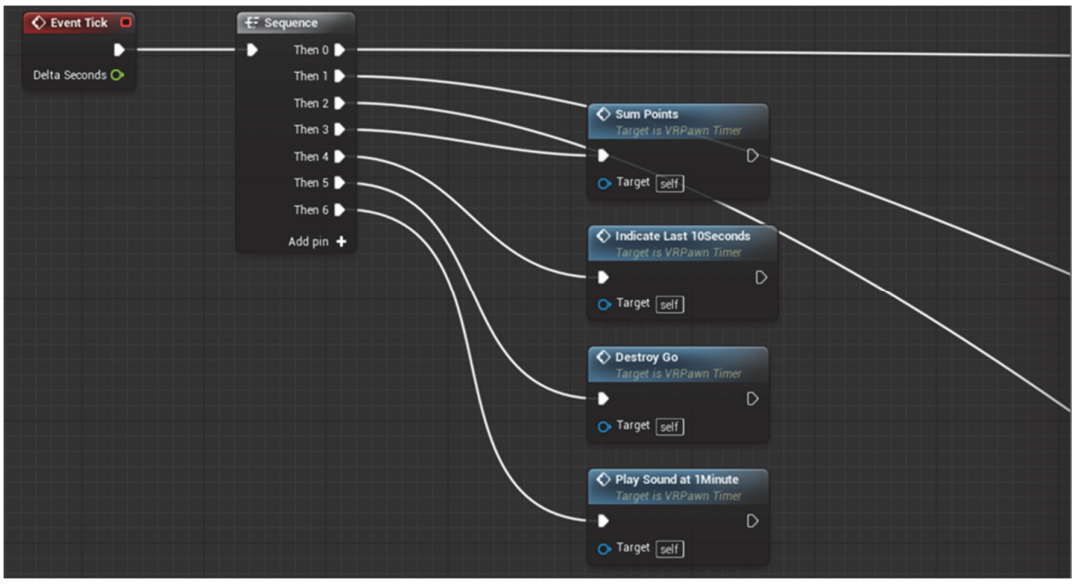
Nimetatud elemente on võimalik kasutada ka muudes mängutasemetes ja projektides, seega tegemist on taaskasutatava tervikuga.

Kapselldusprintsiibi (*The Single Responsibility Principle)* alusel oleks võinud aja muutujad lisada ka mängukarakterist eraldi. Näiteks oleks võinud kasutada mängu reeglit, liiki ja tulemust hoidvat *blueprint* kogumikku (*GameMode).* Autor otsustas aja parameetrite muutmise jätta mängukarakterisse, sest antud lõputöö on üksikmängijale mõeldud ning ei vajanud enamus eelmainitud *blueprint* kogumiku poolt pakutud lisafunktsionaalsusi, nagu näiteks sisselogimine ja mitme mängija haldamine [24].

5.4.2 VRPawn_Timer arendus

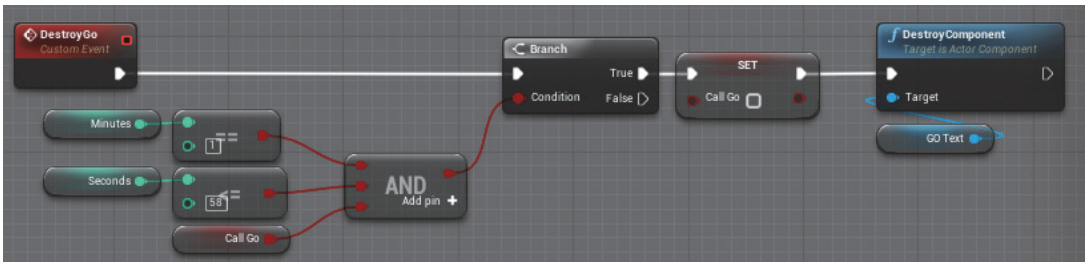
Põhiliseks VR mängukarakteriks sai nimetatud *VRPawn_Timer,* mis kasutas mängija vasaku ja parema käe jaoks *MotionController* muudetud duplikaati (vt Joonis 3, lk 27). Lisaks tuli VR mängukarakterit täiendada uute funktsioonide ja sündmustega. Järgnevalt on toodud nende täpsem kirjeldus ning muudatused arenduse käigus.

Kuigi mäng töötas, oli esialgu sündmuse *Event Tick* külge programmeeritud liiga palju sõltuvust. Arenduses ei ole see hea tava, sest igal kaadril kutsutakse välja kõiki tema klemmide küljes olevaid sündmusi. Seega otsustas autor teatud funktsioonid välja kutsuda teisiti kui järgneval joonisel (Joonis 5).



Joonis 5. Esialgne versioon *Event Tick* sündmuste sõltuvusest. Siniste kastidega on kujutatud autori loodud sündmused, väljaminevad jooned on *VRPawn* oma funktsioonid ning üks joontest ka eraldiseisev kellaaja loendamine, mis ei olnud omaette sündmuseks tehtud.

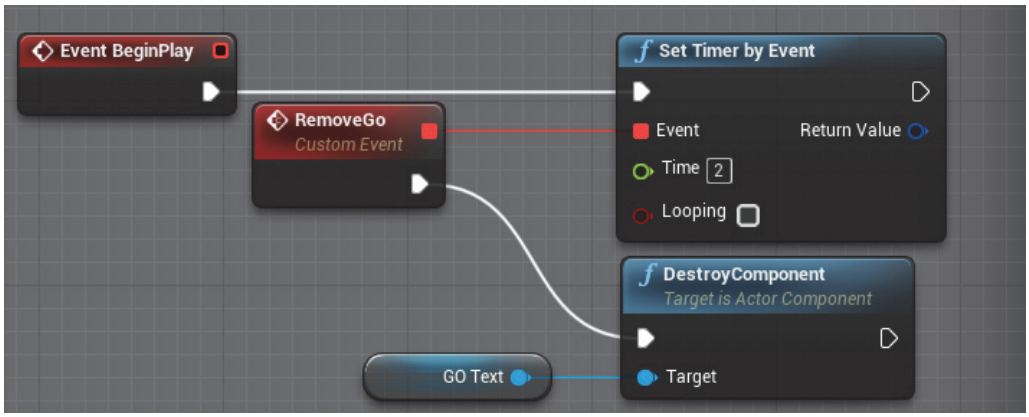
Autor arvas esialgu, et *Event Tick* küljes olevat sündmust saab välja kutsuda vaid üks kord, kui kasutada selleks kahendväärtust (*Call Go)*. Peale esimest kutsumist muudetaks kahendväärtus eituseks ning seega element *AND Boolean* ei saaks enam mängu jooksul tõene olla. Samas oleks see tähendanud, et antud tingimust oleks iga kaadri puhul välja kutsutud. Esialgu nägi *DestroyGo* välja järgnev (Joonis 6).



Joonis 6. Sündmuse *DestroyGo* esialgne versioon.

Esialgsest lahendusest järeldusi tehes polnud tegelikult vaja lugeda muutujaid *Minutes*, *Seconds, Call Go*. Sündmus *DestroyGo* sai üldse lahendatud ilma *Event Tick* sõltuvuseta.

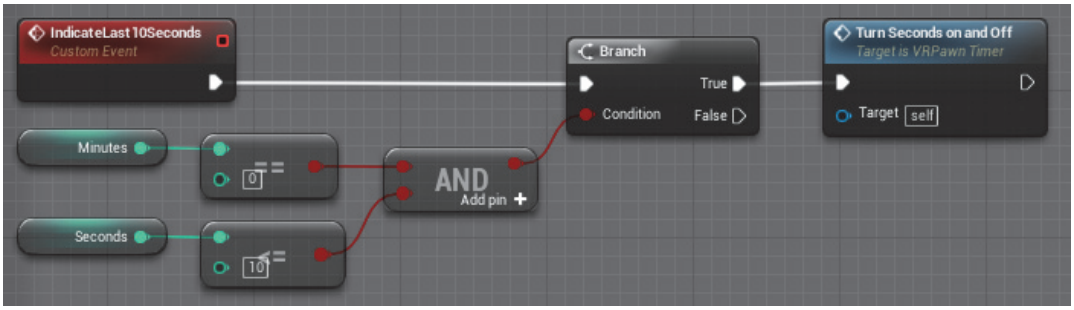
Eelneva asemel otsustas autor kasutada sündmuse *Event BeginPlay* külge asetatud funktsiooni *Set Timer by Event* ja kutsuda 2 sekundit hiljem välja sündmus *RemoveGo*. Seega uues versioonis sündmused ja funktsioonid kutsuti välja ainult üks kord. Lisaks tuleb mainida, et *Time* sisend kasutab täisarvude asemel ujukomaarve. Nende kasutamine võib lisada ajaliselt juurde mängule mittevajalikke sekundilisi murdosi. Seetõttu ei ole ujukomaarvude kasutamine täisarvude asemel üksühele sama, kuid antud juhul kuvati mängijale ainult teksti ning mängivale inimesele selline erinevus välja ei paista (Joonis 7).



Joonis 7. Sündmuse *RemoveGo* uuendatud versioon. Funktsioon *SetTimerByEvent* kutsub välja sündmuse 2 sekundi pärast, mis kutsub välja komponendi *Go Text* kustutamise funktsiooni *DestroyComponent*.

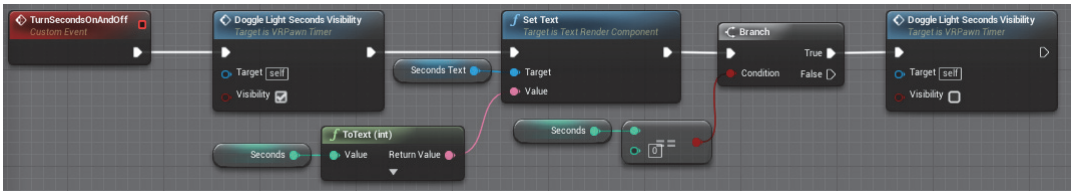
Sarnaselt sai muudetud sündmus *PlaySoundAt1Minute*. Funktsioon *SetTimerByEvent* kutsub välja sündmuse 60. sekundil pärast mängu algust, mis käivitab tiksuva kella heli funktsiooni *PlaySound2D* abil. Seega sai ajast vähemsõltuvad sündmused lahendada sündmusest *Event Tick* eraldi, vähendades sellega olulisel määral väljakutsumiste arvu.

Autor proovis sarnaselt tööle panna sündmuse *IndicateLast10Seconds*, kuid selle sündmuse muutujate kuvamine oli otseses sõltuvuses muutujatest *Seconds* ja *Minutes*. Ujukomaarvude kasutamine oleks andnud ebatäpseid lahendusi. Seetõttu pidi sündmuse väljakutse jääma antud lahenduses sündmuse *Event Tick* külge (Joonis 8).



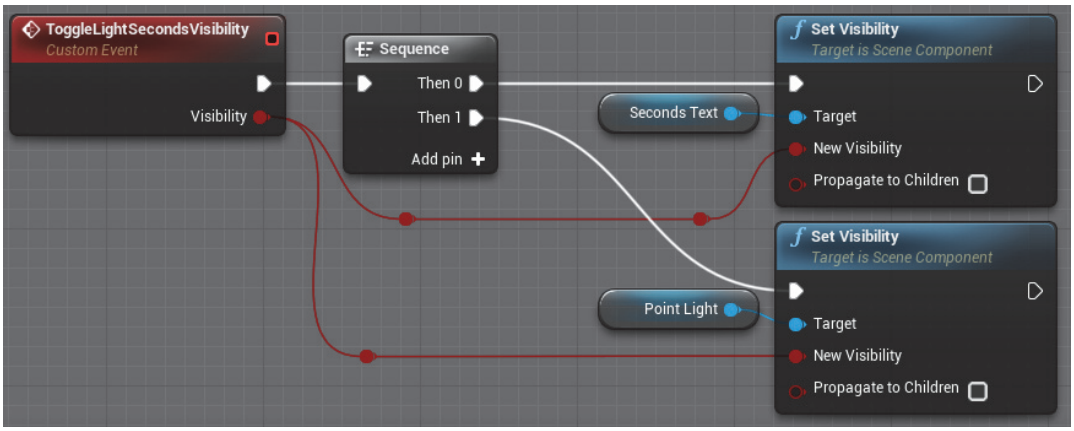
Joonis 8. Sündmuse *IndicateLast10Seconds* loomine. Sündmuse käivitamisel küsitakse, kas minuteid on järgi 0 ning kas sekundeid on järgi väiksem või võrdselt kui 10. Juhul kui muutujad vastavad tingimustele, näidatakse alati kümnendast sekundist alates numbreid.

Sekundite ekraanile kuvamiseks kasutati sündmust *TurnSecondsOnAndOff*, kus lülitatakse vajadusel sisse või välja ekraanile kuvatavad sekundimuutujad ja valgus. Valgust kasutati selleks, et mängija tähelepanu ergastada (Joonis 9).



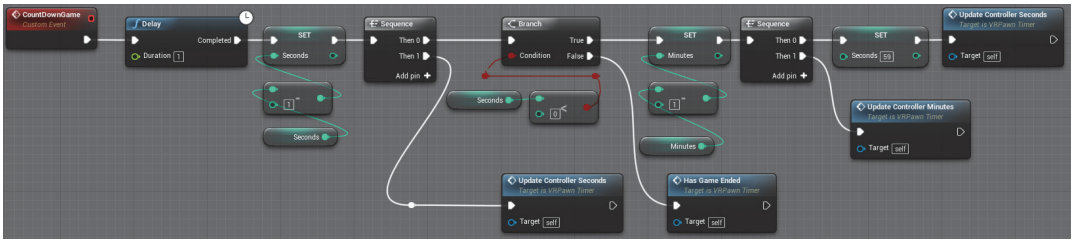
Joonis 9. Sündmuse *TurnSecondsOnAndOff* loomine. Esile kutsutakse sündmus *ToggleLightSecondsVisibility*, määratakse muutujatele tekstiline väärtus ning sekundite jõudmisel nulli korratakse eelmainitud sündmuse väljakutset.

Sündmus *ToggleLightSecondsVisibility* on kombinatsioon kahest *SetVisibility* funktsioonist, mille kahendväärtuse saab arendaja ise määrata ning sõltuvalt kahendväärtusest muutujaid kas kuvatakse või mitte (Joonis 10).



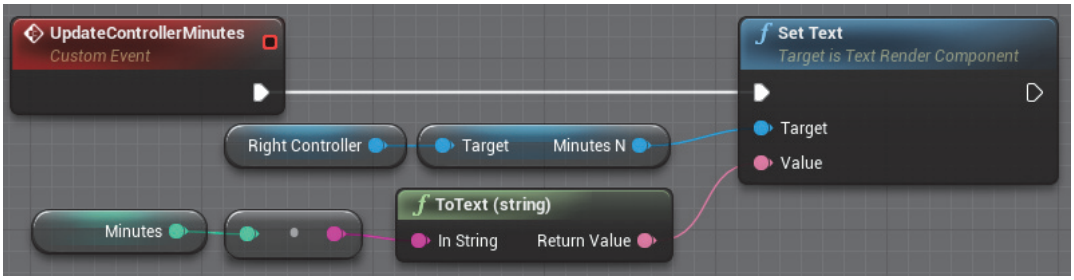
Joonis 10. Sündmuse *ToggleLightSecondsVisibility* loomine. Sisendiks on *Visibility* kahendväärtus, mida saab tõeseks või eituseks määrata ning seega teisi vastavaid *SetVisibility* funktsioonide sisendeid juhtida.

Mängu ajalise kontrolli jooksutamiseks loodi uus sündmus *CountDownGame*, mis tegeles muutujate *Minutes* ja *Seconds* uuendamisega. Lisaks seoti selle sündmuse külge mängu lõpptingimusi kontrolliv sündmus *HasGameEnded* (Joonis 11).



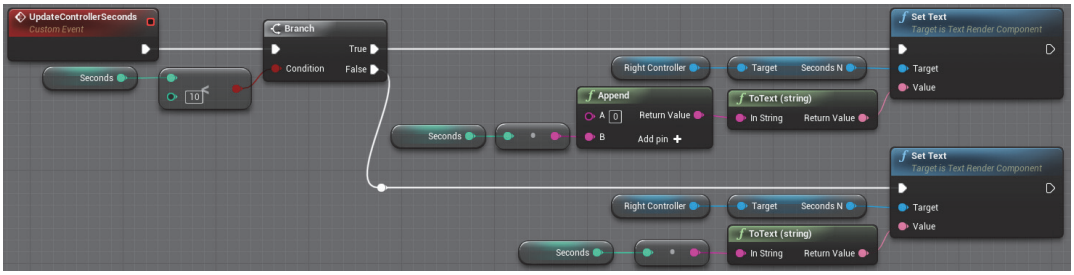
Joonis 11. Sündmuse *CountDownGame* loomine. Funktsiooniga *Delay* aeglustatakse kogu ülejäänud loogika väljakutsumine ühele sekundile, millele järgneb muutuja *Seconds* määramine ning selle uuendamine. Juhul kui sekundite väärtus pole jõudnud alla 0, kontrollitakse lõpptingimust. Vastasel juhul uuendatakse *Minutes* ja *Seconds* väärtuseid.

Mängija parema käe randmel oleval kellal (vt Joonis 3, lk 27) minutite kuvamiseks ja uuendamiseks loodi refaktoreerimisel sündmus *UpdateControllerMinutes* (Joonis 12).



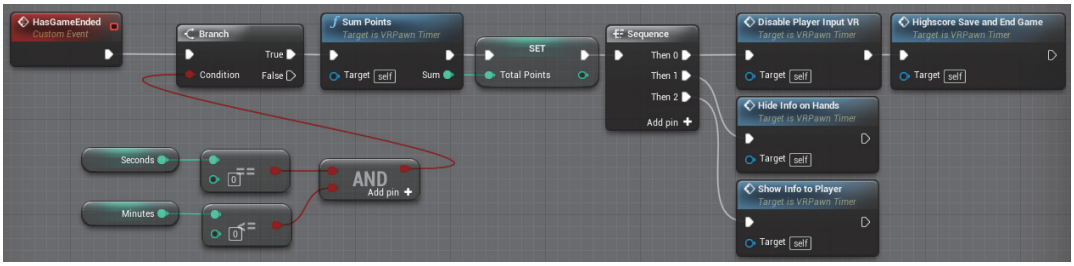
Joonis 12. Sündmuse *UpdateControllerMinutes* loomine. *Minutes* muutuja väärtust arvesse võttes muudetakse see tekstiliseks väärtuseks *ToText* funktsiooniga ning määratakse uueks visuaalseks väärtuseks *Set Text* funktsiooniga, pöördudes selleks muutuja parema käe *MotionController* poole.

Samuti loodi kella sekundite kuvamiseks ja uuendamiseks analoogne sündmus, kuid mis kahekohaliste arvude tõttu sisaldas loogikat ka väiksemate arvude korrektseks kuvamiseks (Joonis 13).



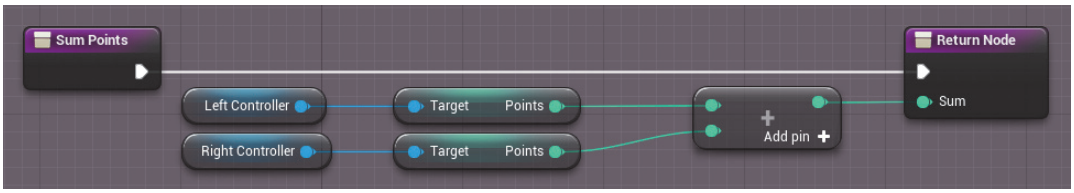
Joonis 13. Sündmuse *UpdateControllerSeconds* loomine. Sekunditele lisati kontroll, et kui sekundeid on vähem kui 10, siis koos sekundi numbriga lisatakse ajale juurde 0 sõne (*string*). Üldjuhul rakendati sarnane funktsionaalsus, mis *UpdateControllerMinutes* puhul.

HasGameEnded sündmuses küsitakse, kas mäng on lõpufaasi jõudnud. Kui tingimus on tõene, liidetakse punktid kokku ning mäng peatatakse (Joonis 14).



Joonis 14. Sündmuse *HasGameEnded* loomine. Kui minutid ja sekundid on jõudnud väärtuseni 0, liidetakse punktid kokku funktsiooniga *SumPoints,* määratakse *TotalPoints* väärtus ning käivitatakse mängu lõpetavad sündmused.

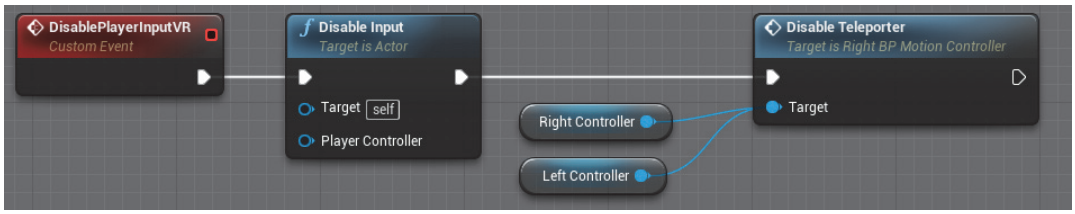
HasGameEnded sündmuses punktide liitmiseks loodi *MotionController* täiendatud duplikaadile funktsioon *SumPoints*. Selleks pöördutakse vasaku ja parema *MotionController* poole ning kutsutakse välja nende muutujad (*Points*), mis omavahel liidetakse (Joonis 15).



Joonis 15. Funktsiooni *SumPoints* loomine.

Mängu peatamisel tekkisid esialgu probleemid. Mängumootoril oli olemas mängu peatamise funktsioon *SetGamePaused*, mida autor esialgu proovis, kuid ilmnes, et virtuaalreaalsuses ei saa seda funktsiooni mängu lõpetamiseks kasutada. Funktsioon *SetGamePaused* hakkas virtuaalreaalsuse vaatepilti tükeldama ning tekitas silmadele ebamugavust, seega pidi autor looma analoogse mängu peatamiseks kasutatava funktsionaalsuse.

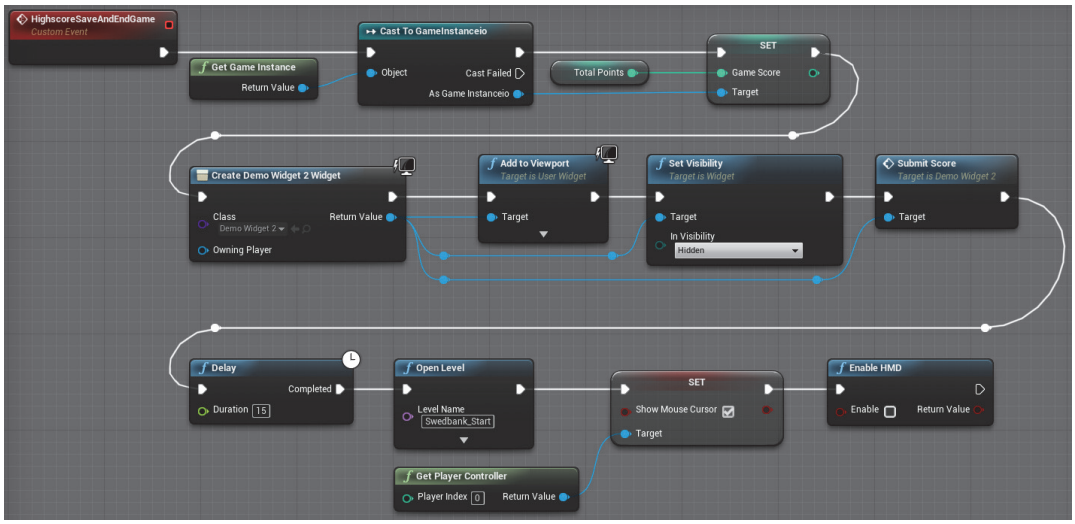
Sündmus *DisablePlayerInputVR* eemaldab VR mängukarakteri sisendid, kuid visuaalselt jätab mannekeeni käed nähtavaks. Sisuliselt võetakse mängijalt liikumise võimalus ära. Sündmus *DisableTeleporter* eemaldab *teleportation* sisendid. Mängija võis kogemata *teleportation* nuppu all hoida ja *DisableTeleporter* välistas sellise võimaluse. Seega oli mängu peatamise funktsionaalsus loodud (Joonis 16).



Joonis 16. Sündmuse *DisablePlayerInputVR* loomine.

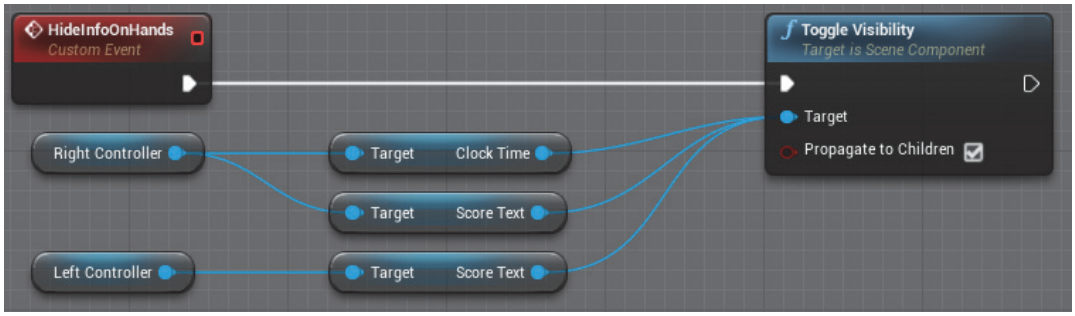
Järgnev sündmus *HighScoreSaveAndEndGame* on kaastudengi ja autori ühise koodi osa. Põhiliseks on tulemuse salvestamine ning selle sisestamine punktitabelisse, millele järgneb registreerimisvaatesse tagasi pöördumine. Ideaalis oleks võinud antud tegevused lahendada erinevates alamfunktsioonides või alamsündmustes, kuid autor otsustas ühise koodi osa ühe sündmuse alla jätta, sest tegemist on ühise tervikuga mängu lõppemise realiseerimisel.

Mängu tulemus (*TotalPoints*) salvestatakse globaalsesse *GameInstance* muutujasse, samuti saadetakse tulemus kaastudengi loodud elementidele (*Widget*), mis saadavad tulemuse punktitabelisse. Pärast seda andis autor mängijale võimaluse 15 sekundiks ringi vaadata (*Delay*), et mängija saaks puldid anda vahendajale. Lõpuks liigub mäng tagasi esimesse vaatesse kasutades funktsiooni *OpenLevel*, seega tuli eemalda funktsioonis *Enable* HMD linnuke, et 2D registreerimisvaates saaks järgmine kasutaja oma nime sisestada (Joonis 17).



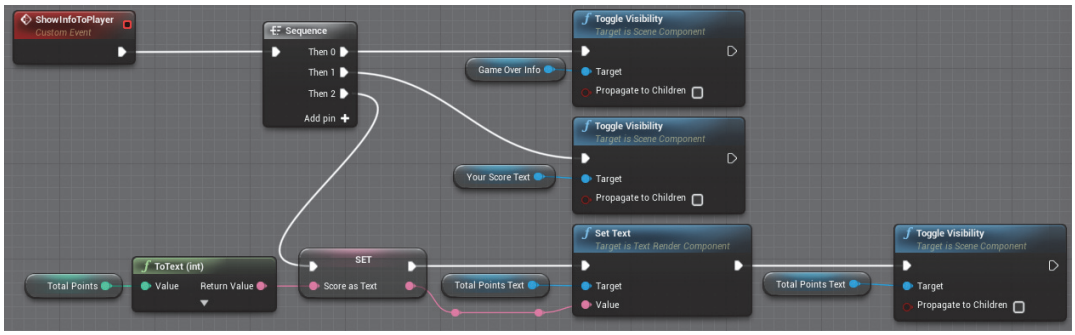
Joonis 17. Sündmuse *HighScoreSaveAndUpdate* loomine.

HasGameEnded sündmuses (vt Joonis 14, lk 33), on sündmuse *HideInfoOnHands* ülesehitus järgmine. Eesmärgiks oli mängu lõppedes mannekeeni kätelt eemaldada informatsioon. Selleks pöörduti elemendi *Clock Time* poole, et peita ajalised muutujad ning element *Score Text*, et peita punktid. Selleks kasutati funktsiooni *Toggle Visibility* (Joonis 18).



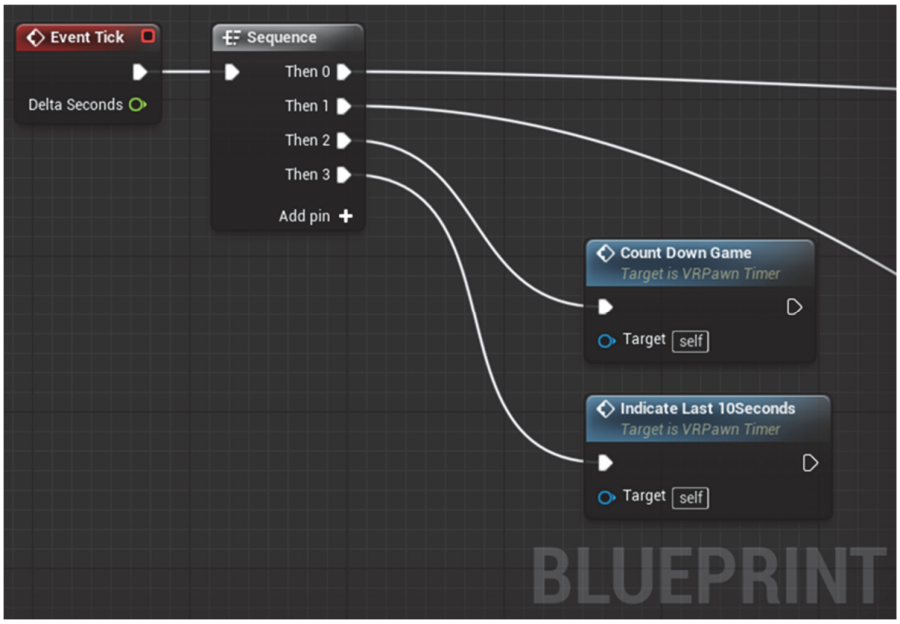
Joonis 18. Sündmuse *HideInfoOnHands* loomine.

HasGameEnded sündmuses (vt Joonis 14, lk 33), on sündmuse *ShowInfoToPlayer* ülesehitus järgmine. Sündmus kuvab mängijale mängu lõppedes info (*GameOver*) ning punktitulemuse (*TotalPoints*) mängija ekraanile. Selleks oli loodud peatüki alguses kirjeldatud mängijat informeerivad elemendid kaamera külge. Antud sündmuses tehakse need elemendid funktsiooni *Toggle Visibility* abil nähtavaks (Joonis 19).



Joonis 19. Sündmuse *ShowInfoToPlayer* loomine.

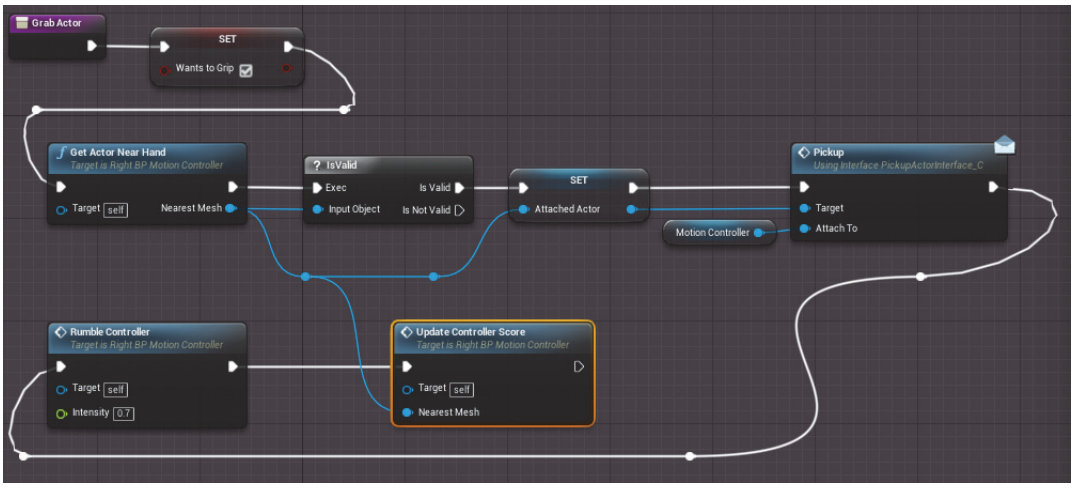
Lõpuks vähenes *Event Tick* otsesest sõltuvusest mitu sündmust, mis leidsid rakendust nende jaoks sobilikel aegadel. Otsesesse sõltuvusse jäid autori poolt loodud sündmustest *CountDownGame* ja *IndicateLast10Seconds*. Selle asemel, et paralleelselt kutsuda iga kaardisageduse peale sündmusi, on antud lahendus parem, sest sündmused kutsutakse välja läbi loogilise järjestuse ning liigsed funktsioonide läbijooksutamised on peatatud *branch* elementidega sündmuste sees. Antud lahendus on suund parema ja loogilisemalt järjestatud lahenduse suunas (Joonis 20).



Joonis 20. Viimane versioon *Event Tick* sündmuste sõltuvusest.

5.5 Punktide kogumine

Antud lõputöös kasutati punktide kogumiseks peatükis 5.3 tutvustatud interaktiivseid objekte ning *MotionController* sisendit. Selleks täiendati *MotionController* muudetud duplikaadi funktsiooni *GrabActor*. Autor lisas eelnimetatud funktsioonile ühe uue sündmuse *UpdateControllerScore*. Lisaks oli *GrabActor* juba varustatud funktsiooniga, mis kontrollis, et tegemist oleks korrektse sisendobjektiga (*InputObject*). Objekt saadetakse edasi teistesse funktsioonidesse, seega ei pea antud kontrolli kordama (Joonis 21).

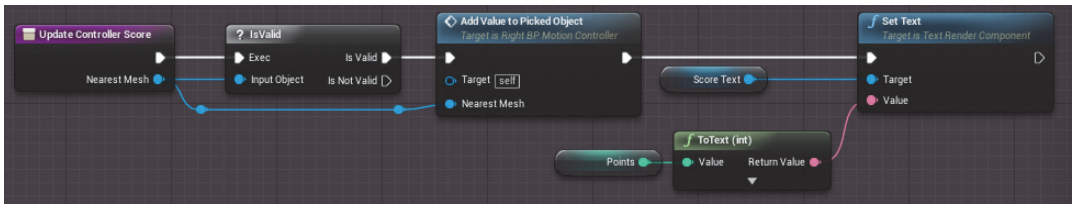


Joonis 21. Funktsioon *GrabActor* ja selle täiendus. *UpdateControllerScore* on märgitud kollase kastiga. Objekti korrektsust kontrollib *IsValid* funktsioonikastike.

Punktide kogumise funktsionaalsust võib realiseerida mitmeti. Järgnevalt on toodud kaks eri lahendust samale *UpdateControllerScore* funktsioonile, millest töö käigus valiti üks ning mille valikut on põhjendatud järgnevas töö osas.

5.5.1 Esimene versioon

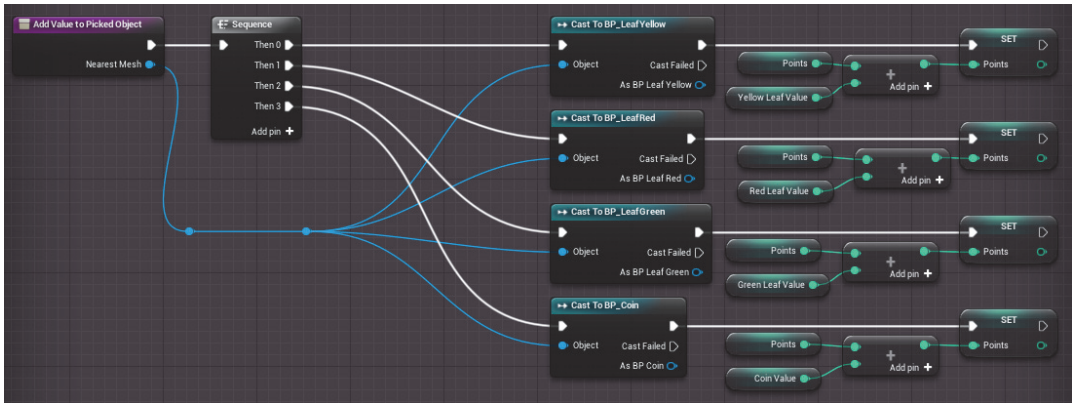
UpdateControllerScore võtab sisendiks lähima 3D objekti, mis mängija käe juures *GrabActor* sündmuse realiseerides oli (*Nearest Mesh*). Siis määratakse korjatud objektile väärtus (*AddValueToPickedObject*) ning kuvatakse kogutud punktisummat (*Score Text)* mängija sõrme juures kasutades funktsiooni *Set Text* (Joonis 22).



Joonis 22. Funktsiooni *UpdateControllerScore* loomine.

Funktsioon *AddValueToPickedObject* töötab põhimõttel, et korjatud objekt pöördub erinevate loodud interaktiivsete objektide poole, kasutades selleks *CastTo* meetodit. Ainult üks nendest pöördumistest õnnestub ning seega saab sellele määrata punktiväärtuse ning liita olemasolevale summale (Joonis 23). Selle jaoks loodi konstantsed muutujad:

- Kollase lehe väärtus (*YellowLeafValue*), mis on väärt 1 punkt.
- Rohelise lehe väärtus (*GreenLeafValue*), mis on väärt 2 punkti.
- Punase lehe väärtus (*RedLeafValue*), mis on väärt 3 punkti.
- Mündi väärtus (*CoinValue*), mis on väärt 10 punkti.

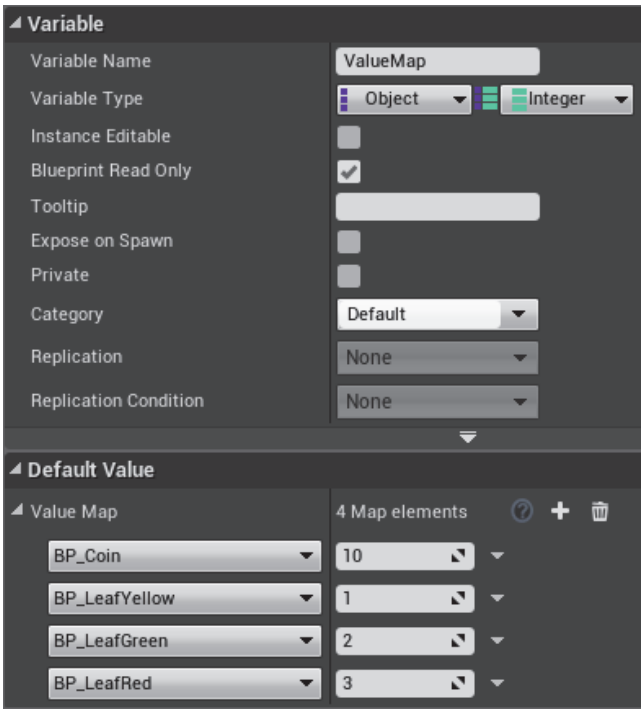


Joonis 23. Funktsiooni *AddValueToPickedObject* loomine.

Eelnev lahendus realiseerib eesmärgi, kui korjatavaid objekte on vähe. Juhul kui erinevaid objekte oleks rohkem, tuleks mõelda massiivi või mõne muu lahenduse suunas. Järgnevas lahenduses on kasutatud kujutise andmemuutujat (*map*).

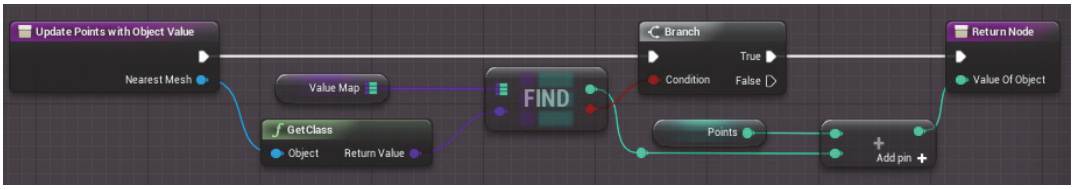
5.5.2 Teine versioon

Töö edenedes leiti autori poolt efektiivsem lahendus, mis ideaalis toetab ka rohkemate objektide loomist ning annab kompaktsema ülevaate erinevatest objektidest. Selleks loodi konstantne kujutise muutuja *ValueMap*, mille võtmeks võeti objekti klass ning väärtusteks esimeses versioonis eelnimetatud muutujate väärtused (Joonis 24).



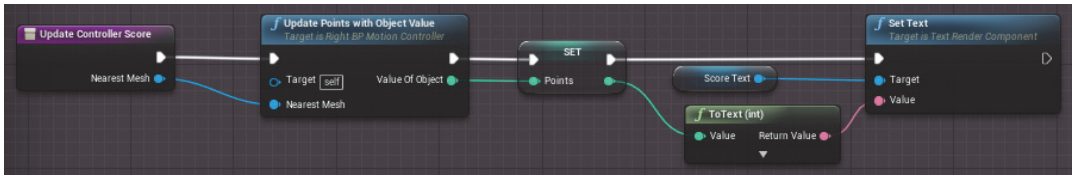
Joonis 24. Kujutise muutuja *ValueMap* defineerimine. Objekti klassi poole pöördumine on kuvatud lilla värviga, täisarvu poole pöördumine rohelisega, konstantseks muutujaks määramine *Blueprint Read Only* linnukesega. Väärtused (*Default Value*) määrati neljale mängus kasutatavale objektiklassile.

AddValueToPickedObject klass nimetati ümber *UpdatePointsWithObjectValue.* Punktide arvestamiseks päritakse korjatud objekti *blueprint* kogumik (*Get Class*). Edasi uuritakse otsimise (*find)* funktsiooniga, kas korjatud objekt kuulub *ValueMap* võtmete alla. Tagastatakse võtmele vastav väärtus, mis liidetakse *Points* muutujale. Summa tagastatakse funktsiooni väljundis (*Return Node*) (Joonis 25).



Joonis 25. Funktsiooni *UpdatePointsWithObjectValue* loomine.

UpdateControllerScore jäi põhimõtteliselt samaks, kuid punktide määramine muutujale toimub eelnevast funktsioonist väljaspool ning niimoodi on arendajal parem kasutada funktsiooni *Set* väljundit funktsiooni *ToText* sisendina (Joonis 26).



Joonis 26. Funktsiooni *UpdateControllerScore* loomine.

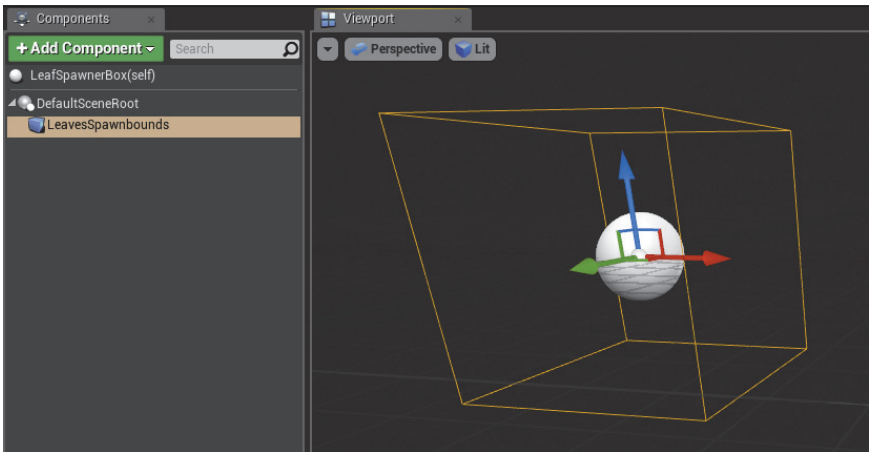
Lahendus nõuab vähemate kastide kasutamist *Blueprint Editor* sees, toetab ka teiste objektide lisamist ilma täiendava keerukuse lisamiseta ning sellepärast jäeti teise versiooni valiku juurde.

Antud funktsioonide tulemusena toimub täiendatud *MotionController* elemendiga objektide korjamise järel objektidele väärtuse andmine ja nende kuvamine sõrme juures. Alguses otsustati sõrme juurde kuvatud tulemust kasutada programmi silumise eesmärgil, kuid tegelikult osutus see sobivaks lahenduseks punktide näitamiseks mängijale. Sõrme juurde kuvatud loendurinäidud viitavad konkreetse *MotionController* abil korjatud punktiseisule, seega mängijale on näha, kas ta on mängu jooksul korjanud enam vasaku või parema käega.

5.6 Objektide genereerimine mängumaailma

Blueprint Editor on võimalik kasutada mängumootori erinevates kohtades, seega põhimängu failis lisati mänguala keskel oleva tammepuu ümber *actor*, mis võimaldab genereerida mängumaailma teatud piiritletud asukoha juurde interaktiivseid lehe- ja mündiobjekte. Antud *blueprint* kasutab eelnevalt selgitatud funktsioone ja mõisteid, seepärast otsustas autor *actor*'i loomet selgitada hilisemas peatükis.

Lehtede genereerimiseks loodi *actor* nimega *LeafSpawnerBox.* Antud *actor* koosneb 3D kastist, mis vajadusel registreerib, kui mõni muu *actor* mängu sees satub tema piirialale või sisse (*Collision Box)*. *Leaves Spawnbounds* on antud *actor* elemendi 3D *Collision Box* nimetus. *LeafSpawnerBox* puhul on tegemist visuaalselt läbipaistva kastiga, millest saab mängija läbi liikuda, kuid mille ruumilisi parameetreid saab ära kasutada lehtede mänguala ruumi genereerimiseks (Joonis 27).

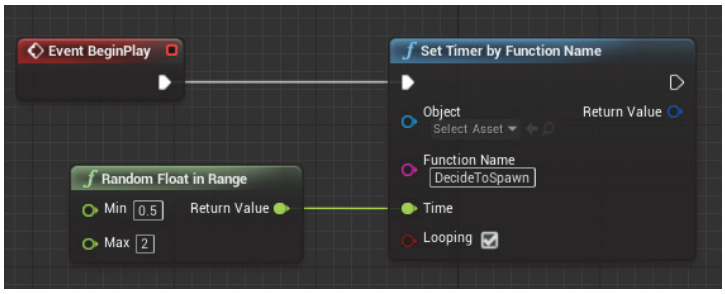


Joonis 27. *LeafSpawnerBox* hierarhia ja visuaalne representatsioon

Tammepuu tüve ümber loodi täiendavalt *Collision Box*, mille ruumilisi parameetreid saab kasutada lehtede ja müntide tüve sisse genereerimise keelamiseks (*ItemlessBox*) (Joonis 28).

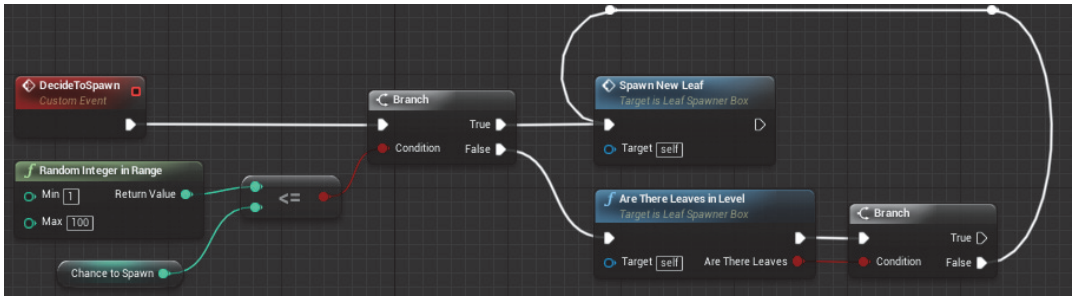
Joonis 28. *ItemlessBox* visuaalne representatsioon ja näide mängumaailma asetusest.

Lehtede genereerimiseks loodi sündmuse *Event BeginPlay* külge sündmus *DecideToSpawn*, mida kutsutakse välja iga 0,5 kuni 2 sekundi tagant. Vahemik loodi, sest aja varieeruvus tekitab mängijas ootamatust ning seega muudab mängu huvitavamaks (Joonis 29).



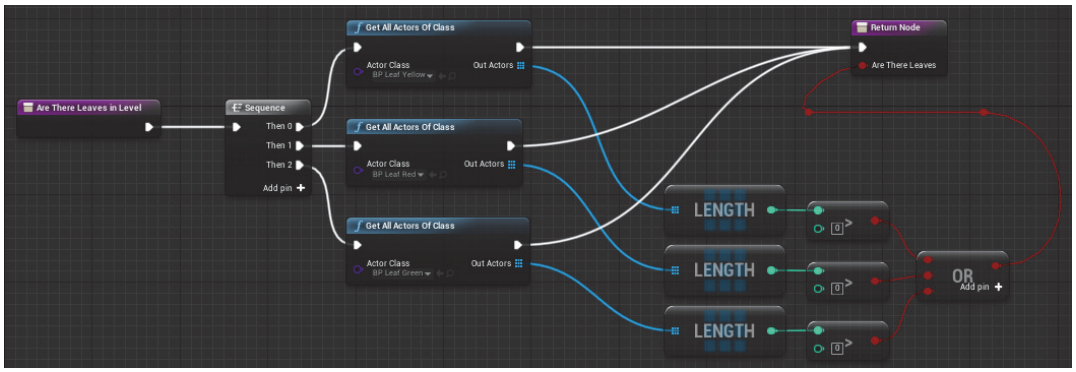
Joonis 29. Sündmuse *DecideToSpawn* väljakutsumine kasutades funktsiooni *SetTimerByFunctionName*.

DecideToSpawn ülesanne on välja kutsuda etteantud tõenäosusega (*ChanceToSpawn*) lehe genereerimise sündmus *SpawnNewLeaf*. Juhul kui muutuja *ChanceToSpawn* ei ole väiksem või võrdne genereeritud täisarvuga vahemikus 1 kuni 100, kontrollitakse funktsiooniga *AreThereLeavesInLevel* kas mängualal lehti on. Juhul kui lehti pole, genereeritakse kindlasti uus leht (Joonis 30).



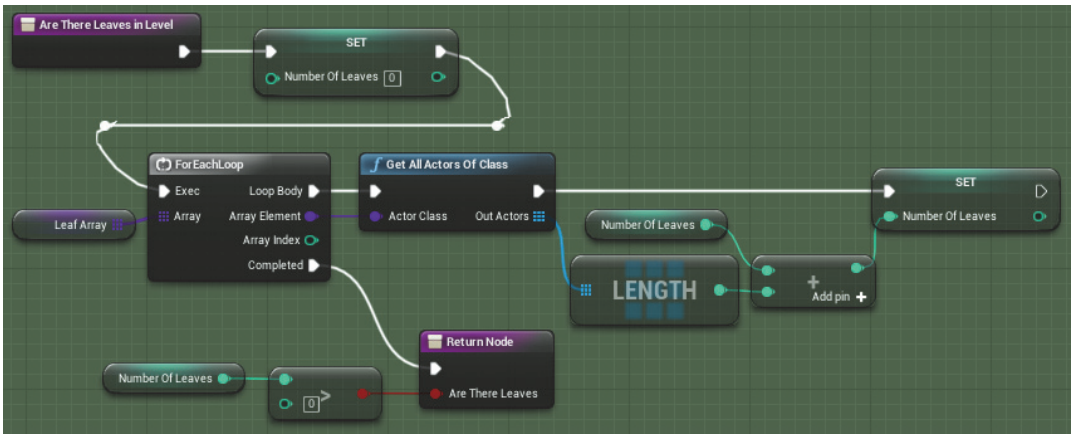
Joonis 30. Sündmuse *DecideToSpawn* loomine.

Lehtede olemasolu kontroll lahendati esialgu teisiti. Autor pidas oluliseks esialgset lahendust kirjeldada, et oleks paremini mõistetav lõplik lahendus. Esialgne *AreThereLeavesInLevel* pärib kõik mängutasemel olevad lehed. Selleks kasutatakse funktsiooni *GetAllActorsOfClass*. Tagastatakse lehtede *actor* elementide massiivid. Kui massiivi suurus on vähemalt ühel suurem kui 0, tagastatakse disjunktsiooni tulemusena tõene väärtus, mis tähendab lehtede olemasolu. Eitav väärtus väljastatakse juhul, kui kõik massiivid ei sisalda ühtegi lehe *actor* elementi ehk teisisõnu siis, kui lehed puuduvad (Joonis 31).



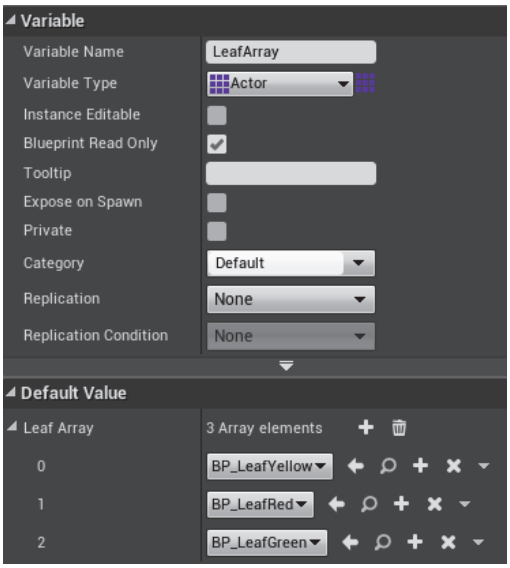
Joonis 31. Funktsiooni *AreThereLeavesInLevel* esialgne versioon.

Hilisemal kaalutlusel leidis autor, et kirjeldatud funktsiooni saab lahendada efektiivsemalt ilma iga *actor* elemendi poole pöördumist eraldi välja kutsumata. Selleks lahendati sama funktsionaalsus tsüklit (*ForEachLoop)* kasutades. Tsükli klemmi *Loop Body* sees korratakse iga klassi poole pöördumist. Lisaks loodi uus muutuja *NumberOfLeaves*, mis loendab massiivis olevate lehtede arvu ning tagastab tõese tõeväärtuse kui massiivis lehed leiti (Joonis 32).



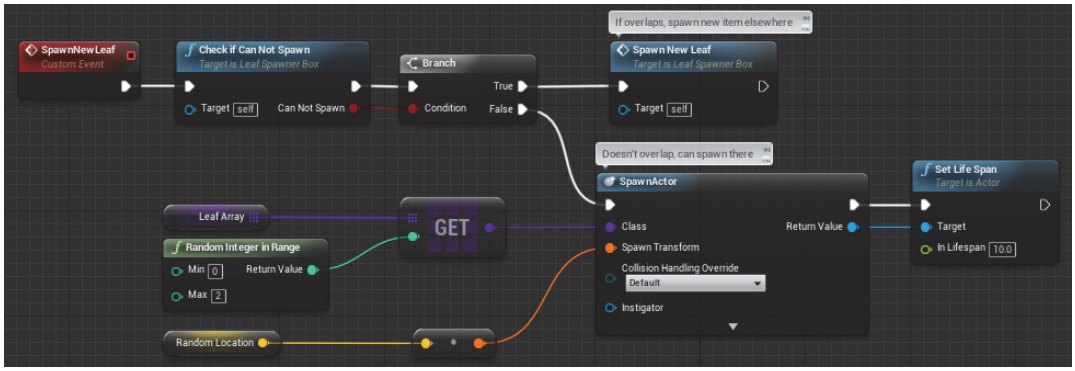
Joonis 32. Funktsiooni *AreThereLeavesInLevel* uuendatud versioon. Tsükkel kasutab sisendiks massiivi *LeafArray*, kus pöördutakse iga massiivi elemendi poole tsükli sees ning päritakse esindavate objektide massiivi suurus(*length*), mis liidetakse *NumberOfLeaves* muutujale.

LeafArray on konstantne (*Blueprint Read Only*) massiiv erinevatest mängus olevatest leheobjekti *actor* klassidest, mille defineerimine näeb välja järgnev (Joonis 33).



Joonis 33. Massiivi *LeafArray* defineerimine. *Blueprint Read Only* teeb muutuja konstantseks. *Default Value* all defineeritakse *blueprint* klassid.

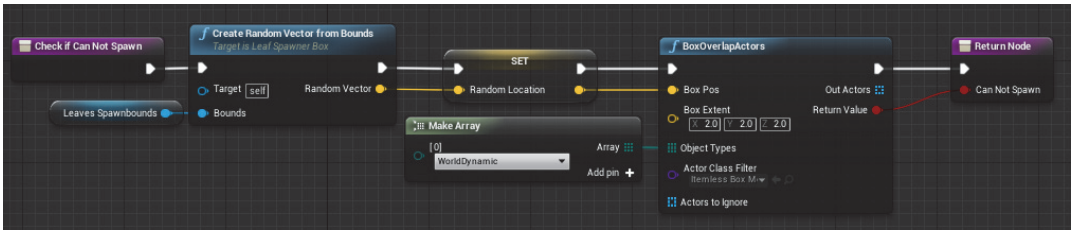
Lehe genereerimise sündmus *SpawnNewLeaf* küsib funktsiooni *CheckIfCanNotSpawn* käest, kas uut lehte on võimalik genereerida või mitte. Juhul kui uut lehte ei ole võimalik genereerida, väljastatakse *branch* tõeses klemmis uuesti sama sündmus, et otsida uut asukohta. Vastasel juhul käivitatakse eituse klemmis *actor*'i genereerimise *blueprint* (*SpawnActor*), mis kasutab juba eelnevalt tutvustatud massiivi (*LeafArray)*. Funktsiooniga *Random* valitakse massiivist juhuslik element, mis genereeritakse *LeafSpawnerBox* sisse. Lisaks seatakse lehele eluiga 10 sekundit funktsiooniga *SetLifeSpan*. Eluiga määrati põhjusel, et objekt ei jääks terve mängu vältel mänguruumi alles (Joonis 34).



Joonis 34. Sündmuse *SpawnNewLeaf* loomine.

CheckIfCanNotSpawn tööks küsib funktsioon *CreateRandomVectorFromBounds* alapeatüki alguses tutvustatud *LeavesSpawnBounds* mõõtmeid ning genereerib

juhusliku asukohavektori nende mõõtmete piiride sees (*RandomLocation*). Funktsioon *BoxOverLapActors* teeb loodud genereeritud asukohavektori ümber kasti, mis on suurima lehe mõõt. Juhul kui kast, mis tekitatakse mängumaailma, satub *ItemlessBox* mõõtmete sisse, ei saa lehte keelualale genereerida. Juhul kui keelatud alale kasti ei tekitata, saab lehe alale genereerida. Sõltuvalt tulemusest tagastab funktsioon tõese või eitava vastuse (Joonis 35).



Joonis 35. Funktsiooni *CheckIfCanNotSpawn* loomine. Funktsioonis *BoxOverLapActors* on määratud *Box Extent* mõõtmeks suurima lehe suurus.

Sarnastel põhimõtetel loodi ja töötab ka mündi genereerimise *actor CoinSpawnerBox*. Antud *actor* otsustati autori poolt teha eraldi, sest münt on mängus haruldasem objekt ja seda ei pea mängualas alati eksisteerima. Ideaalis oleks võinud luua *actor*'i, mis sisaldaks kõiki elemente, kuid selle arendus oleks eeldanud paljude lisatingimuste loomist ning teinud *blueprint* algoritmi keeruliseks.

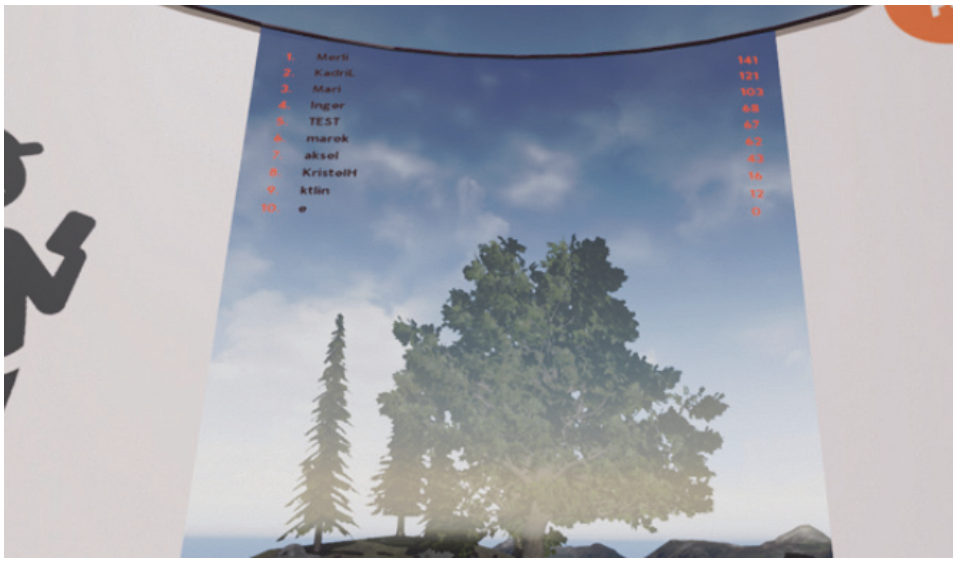
CoinSpawnerBox erinevuseks *LeafSpawnerBox* elemendist on näiteks genereerimise ajastamine. Mündi genereerimise funktsiooni kutsutakse välja harvem, iga 3 kuni 5 sekundi tagant. Mündil puudub kontroll, kas seda leidub mängumaailmas juba või mitte. Sellega on tagatud mündi unikaalsus.

6 Õpetustoa loomine

Alguses oli eesmärgiks teha ainult VR mäng, kus kasutaja pidi aja jooksul koguma punkte. Töö käigus selgus, et mängu tegemisest ainult ei piisanud. Virtuaalreaalsuse esmakordne kasutaja ei oska üldjuhul VR maailmas ringi vaadata ja pulte kasutada oma hüvanguks. Õpetustoa loomise eesmärgiks on kasutajale tutvustada, kuidas mängu mängida ja sisend-väljund liideseid kasutada. Enne ülesande lahendamist tuleb kasutajale selgitada, et virtuaalreaalsuses on pultidel erinevad funktsionaalsused ja HMD abil on võimalik igale poole vaadata. Sarnaseid probleeme on ilmnenud ka teistel arendajatel, kus kasutajad on enne programmi kasutamist soovinud põhjalikumalt tutvuda virtuaalreaalsust edasiandva tehnikaga, et paremini keskenduda programmi sisule [25]. Seega tegi probleemide lahendamiseks projekti autor koos oma tiimiga valmis ka virtuaalreaalsuse õpetamistoa.

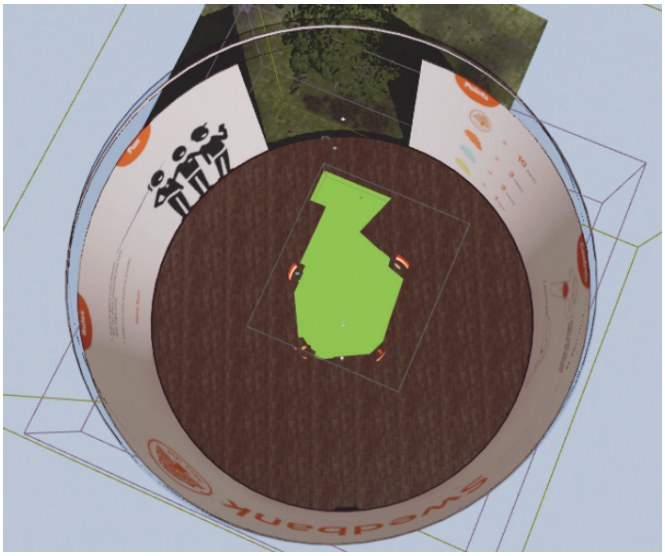
Ühe tiimikaaslase poolt loodi 3D mudel kupliga hoonest, mille siseküljele lisati õpetused. Eesmärgiks oli tutvustada virtuaalreaalsuses kasutatavaid pulte ja nende nuppe, millega kasutaja saab ruumis oma asukohta muuta ning objekte maailmas korjata. Juhised seinal on esitatud Lisa 1 juures. Teadmiste rakendamiseks loodi õpetustoas ülesanne korjata peatükis 5.3 kirjeldatud interaktiivseid objekte, mida kasutatakse ka põhimängus.

Kuppelhoone üks seinaosa jäeti tühjaks. Lõputöö autoril tuli idee kuvada punktitabel tühjale seinale, et mängijal tekiks võistlushasart vaadates parimaid tulemusi. Selleks pöörduti tiimikaaslase poolt loodud punktitabeli *actor*'i poole ning asetati see kuppelhoone külge. Punktitabel oli pooleldi läbipaistev, seega tuli idee jätta põhimängu elemendid taustale nähtavaks, et mängijal tekiks mängust eelvaade (Joonis 36).



Joonis 36. Punktitabel õpetustoas. Kuvatud on mängijate positsioonid, nimed, ja punktitulemused üksteise alla.

Kuppelhoone sees piiritleti sarnaselt põhimänguga ala, kus mängija tohib liikuda. Antud ala pidi mitmeid kordi muutma, sest selgus, et inimesed, kes ei saanud *teleportation* nupu kasutamisega hakkama, reisisid piirialadele ning ei leidnud vajalikke objekte üles. Seega tuli piiriala tekitada täpselt objektide taha, et mängijal õnnestuks *teleportation* käsk. Lisaks otsustas autor jätta ka liikumisala punktitabeli juurde, et mängija võiks näha parimaid tulemusi lähemalt. Seega esialgselt oli piirialal tegemist ruuduga, mida tuli hiljem töödelda (Joonis 37).



Joonis 37. Õpetustoa piiritletud mänguala. Erkrohelisega on tähistatud piirkond, kuhu mängija tohib liikuda.

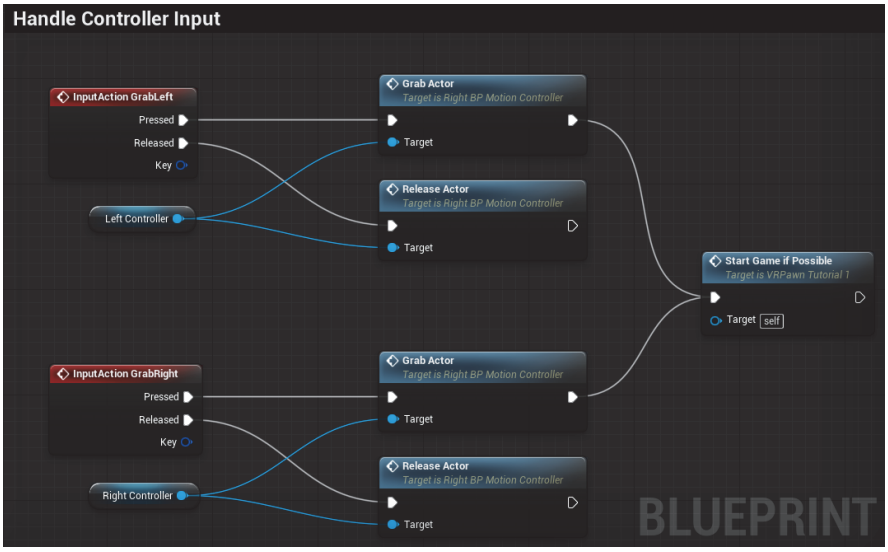
Õpetustoa ruumi keskosas asus mitu elementi:

- õpetustoa mängukarakter;
- kollane, roheline ja punane leht, münt ning neid jälgivad *actor* elemendid;
- tutvustav tekst.

Järgnevates alapeatükkides on kirjeldatud õpetustoa mängukarakteri ning objekte jälgivate *actor* elementide loomet.

6.1 Õpetustoa mängukarakteri arendus

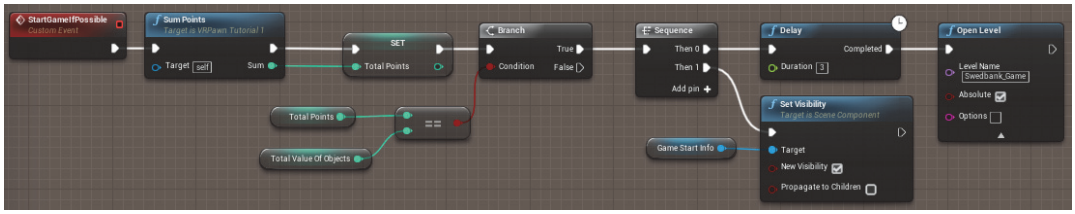
Õpetustoa mängukarakterit võiks kirjeldada kui tavalist mängumootori poolt pakutud *VRPawn* elementi, millele on juurde lisatud üks lisasündmus ja punktide liitmise funktsioon. Sündmuse *StartGameIfPossible* tingimust kutsutakse välja siis, kui on toimunud lehe korjamise sündmus (*GrabActor*). Selle jaoks lisati kontroll lehe korjamise tegevuse lõppu. Seega arvutatakse kogusumma peale iga objekti korjamist (Joonis 38).



Joonis 38. Funktsiooni *StartGameIfPossible* väljakutsumine (paremal).

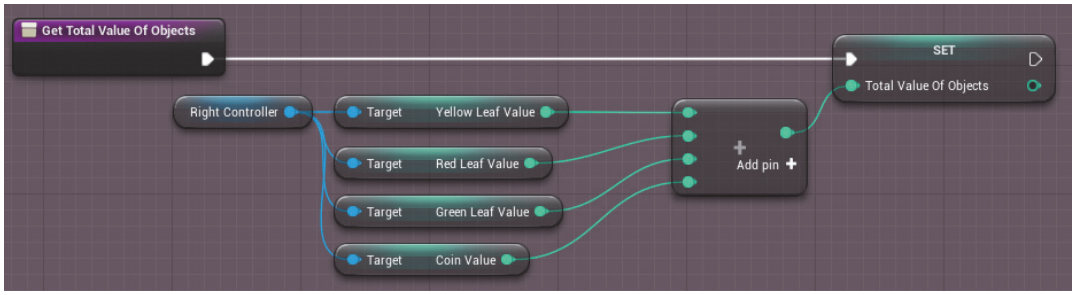
Iga mängus olev interaktiivne objekt asetses õpetustoas üksikult, et mängija saaks iga objektiga enne mängu algust tutvuda. Õpetustoa eesmärk oli ruumis kõik erinevad objektid üles korjata ning seega oli korjatud objektide kogusumma võrdne iga erineva objekti kogusummaga.

Sündmus *StartGameIfPossible* kasutab punktide arvestamiseks parema ja vasaku käe jaoks eelpool mainitud *SumPoints* funktsiooni (vt Joonis 15, lk 33). Kui muutuja *Total Points* on võrdne olemasolevate interaktiivsete objektide kogusummaga (*TotalValueOfObjects*), siis kuvatakse 3 sekundiks (*Delay*) mängijale info (*Game Start Info*) ning minnakse põhimängualale (*OpenLevel*), mille loomist tutvustati peatükis 5.2 (Joonis 39).



Joonis 39. Sündmuse *StartGameIfPossible* loomine.

Esialgse punktide kogumise versioonist (vt Peatükk 5.5.1) lähtudes pöörduti ühe *MotionController* poole ning päriti sealt eriobjektide täisarvväärtused, mis liideti kokku ning määrati muutuja *TotalValueOfObjects* väärtuseks (Joonis 40).



Joonis 40. Muutuja TotalValueOfObjects määramise esialgne versioon.

Teisest punktide kogumise versioonist (vt Peatükk 5.5.2) lähtudes pöörduti muutuja *ValueMap* väärtuste poole funktsiooniga *Values*, mis tagastab väärtuste massiivi. Massiivis olevad elemendid summeeritakse kokku tsükli *ForEachLoop* väljakutsega ning liidetakse muutuja *TotalValueOfObjects* väärtusele. Tulemus võimaldab lisada *ValueMap* sisse elemente juurde ilma algoritmis muudatusi tegemata (joonis 41).

Joonis 41. Muutuja *TotalValueOfObjects* määramise hilisem versioon.

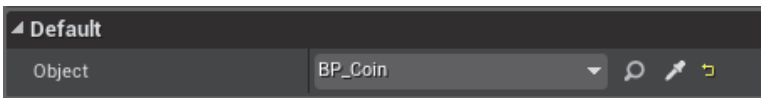
6.2 Interaktiivseid objekte jälgivate elementide loomine

Korjatavate objektide kasutajale atraktiivsemaks muutmise nimel otsustati tiimikaaslaste poolt õpetustuppa luua 3D seina meenutav erksat värvi objekt. Lõputöö autor lisas sellele põhiliselt kaks funktsionaalsust:

- Korjatud objektile reageerimine.
- Materjali ajas nähtamatuks tegemine.

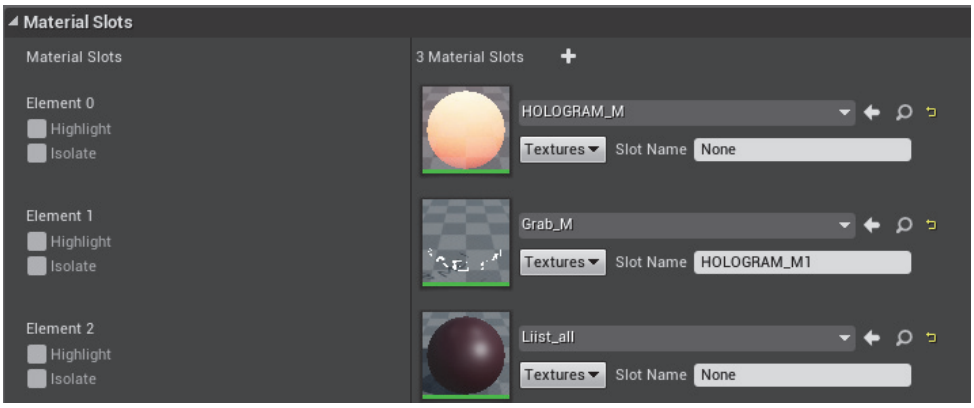
Selle jaoks, et jälgida *Level Editori* interaktiivseid *actor* objekte, loodi *blueprint* nimega *BP_HologramWall.* Korjatud objektile reageerimiseks oodatakse objektilt sündmust *OnDestroyed,* mis käivitatakse siis, kui objekt on mängust eemaldatud. *Level Editori* sees olevate objektide jälgimiseks loodi *BP_HologramWall* sisse avalik *actor* muutuja, millel puudus vaikeväärtus (*Default Value*). Avalikuks määrati see põhjusel, et oleks võimalik *Level Editori* sees soovitud *actor* objekt määrata.

Jälgitava *actor*'i sai määrata õpetustoas kasutades nuppu, mis mängutasemel objektile vajutades tundis ära objekti *blueprint* klassi kuuluvuse (*Pick Actor From Scene)* või manuaalselt vajaliku *blueprint* klassi väljakirjutamisel (Joonis 42).



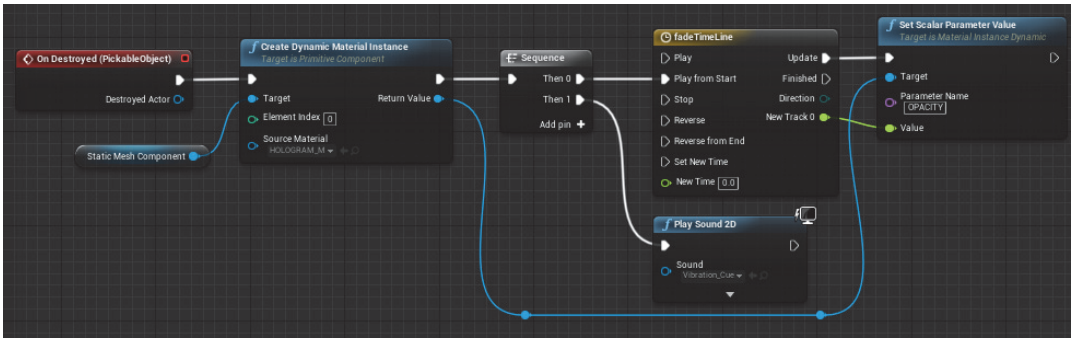
Joonis 42. Vaikeväärtuse määramise väli *Level Editoris*. Antud näites on *Object* elemendi vaikeväärtuseks valitud *BP_Coin*.

BP_HologramWall 3D objekti ümbris (*Static Mesh*), millest *blueprint* loodud on, koosneb kolmest eri materjalist (*material*). Esimese elemendi muutujaid hakkas autor muutma, et saavutada objekti ajas nähtamatuks tegemise efekt (Joonis 43).



Joonis 43. *BP_HologramWall* materjalid (*Material Slots*), millest *blueprint* visuaalselt koosneb. Antud pildil on 3 elementi, mille loendamine algab indeksist 0.

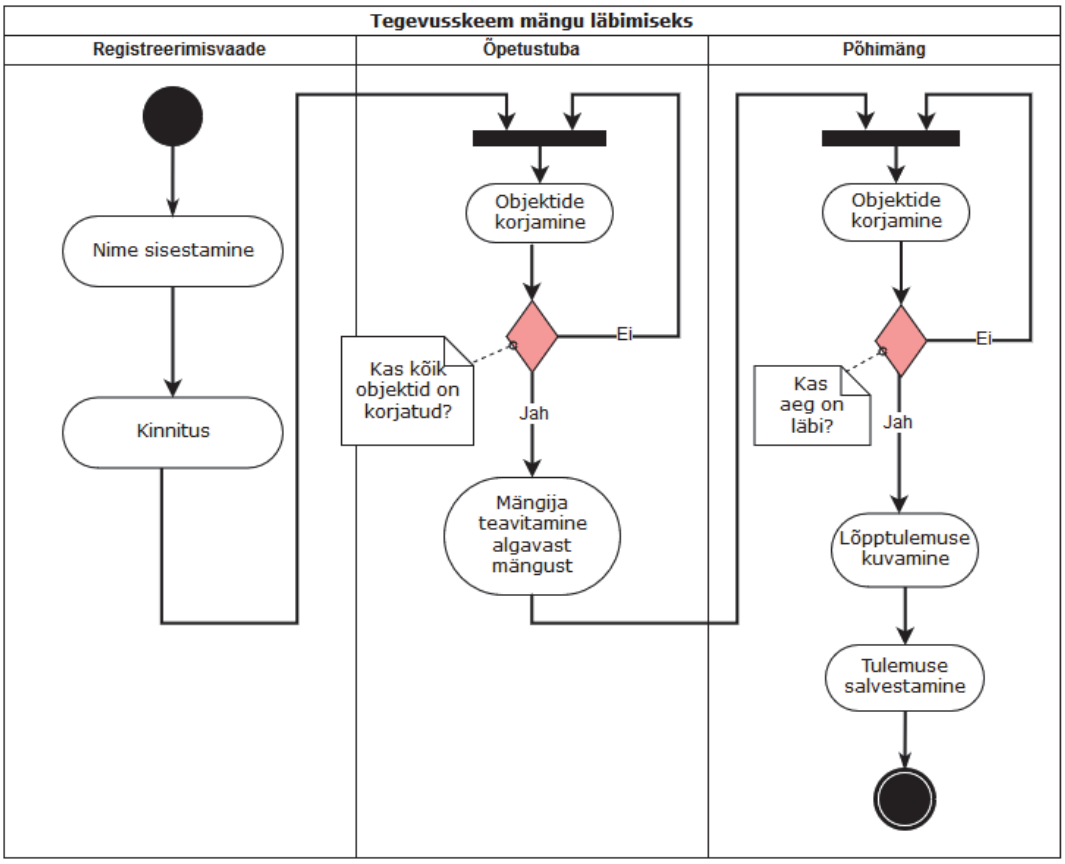
BP_HologramWall ajas nähtamatuks muutmiseks kasutas autor funktsiooni *CreateDynamicMaterialInstance.* Ajajoone jaoks kasutati muutujat *fadeTimeLine*, mis kahe sekundi jooksul muutis oma väärtust. Objekti korjamise järel lisati sellele ka heli funktsiooniga *PlaySound2D* (Joonis 44).



Joonis 44. Sündmuse *OnDestroyed(PickableObject)* loomine. Sisendiks on *PickableObject*, mis valitakse *Level Editori* sees. Muutuja *fadeTimeLine* muutis materjali elemendi parameetri OPACITY (*Parameter Name*) väärtust.

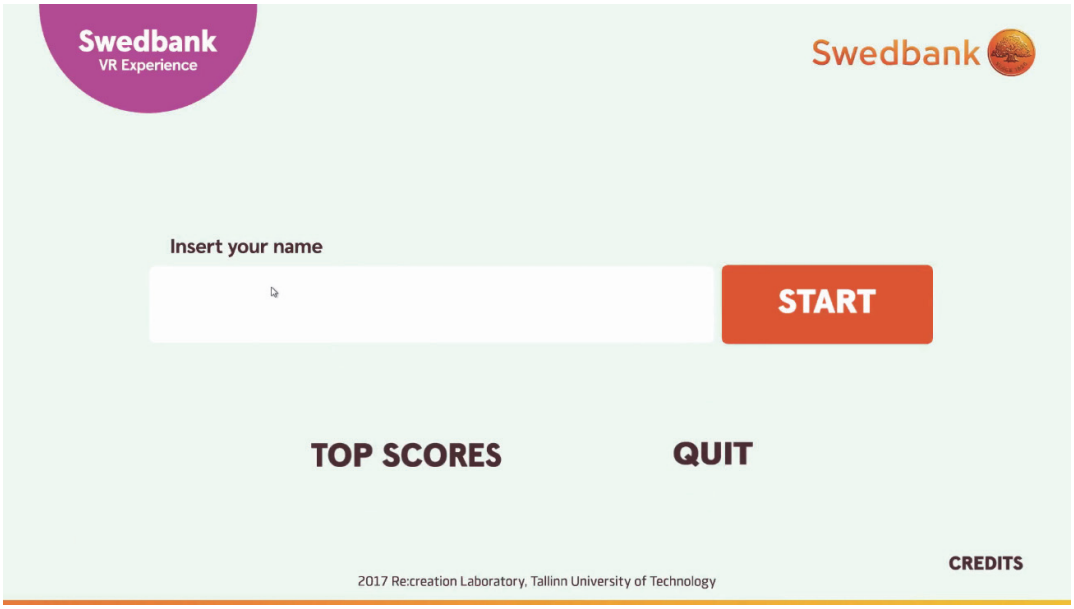
7 Tulemused

Tulemuseks oli VR mäng, mis koosnes kolmest mängutasemest: registreerimisvaade, õpetustuba, põhimäng. Järgnevalt on toodud tegevusskeem mängu edukaks läbimiseks (Joonis 45).



Joonis 45. Tegevusskeem mängu edukaks läbimiseks.

Registreerimisvaate *map* oli loodud tiimikaaslaste poolt ning kasutaja nimi salvestati pärast kinnitamist (*start*) objekti *GameInstance* muutujasse. Mängu lõppedes lisati *GameInstance* muutujasse punktitulemus ning uuendati punktitabelit uue sissekandega. Registreerimisvaadet võib nimetada ka peamenüüks, sest sealt vaatest on võimalik näha tulemusi (*Top Scores*), autoreid (*Credits*) ning vajadusel mängust lahkuda (*Quit*) (Joonis 46).



Joonis 46. Registreerimisvaade.

Registreerimisvaatest edasi avaneb õpetamistuba, mis vajab virtuaalreaalsuse kogemuse ning ülesande läbimise jaoks peakomplekti HTC Vive kasutamist. Õpetamistoas näeb samuti parimaid tulemusi ühel seinaosal (vt Joonis 36, lk 47). Info mängu kohta on kuvatud siseseintele, mis on esitatud Lisas 1. Põhimängu liikumiseks tuleb pultide abil üles korjata õpetamistoas olevad objektid. Kasutajale antakse tagasisidet korjatud objekti kohta nii heli kui ka objekti jälgiva seina värvuse kadumisega (Joonis 47).

Joonis 47. Objekti jälgiva seina *BP_HologramWall* visuaalne representatsioon. Korjatavaks objektiks on kollane leht.

Pärast objektide edukat korjamist ilmub kasutajale info mängu algamise kohta (*Game Starts*). Mängija saadetakse põhilisse mängutasemesse. Seal on mängija ülesandeks korjata piiratud aja jooksul võimalikult palju tammelehti ja münte (Joonis 48).

Joonis 48. Põhiline mängutase koos genereeritud objektidega mängumaailmas.

Kasutaja saab parema käe randmelt vaadata soovi korral järelejäänud aega ja kogutud punkte sõrme juures (vt Joonis 4, lk 28). Lisaks on mängu alguses ning keskel helid, mis tekitavad mängijale ajalise raamistiku.

Mängu lõppedes liidetakse punktisumma ning kuvatakse kasutajale info mängu lõppemise kohta (*Game Over*) koos kogutud punktisummaga (*Score*) (Joonis 49).

Joonis 49. Informatsioon mängu lõpptulemusest mängija vaates põhimängu sees.

Pärast mängu lõppu antakse kasutajale 15 sekundit aega, eesmärgiga puldid tagasi vahendajale anda ning mäng alustab taas registreerimisvaatest.

8 Tulemuste analüüs ja edasiarendamise võimalused

Mäng koosnes mitmest eri osast, seetõttu tekkis mängu arendamisel ning testimisel mitmeid probleeme, mida sai parendada. Samuti on edasiarendamiseks palju võimalusi. Järgnevalt on alapeatükkides välja toodud mängu loomisel tekkinud tähelepanekud ning nende analüüs. Samuti on pakutud võimalikke lahendusi antud projekti edasiarenduseks või ideid parema VR mängu loomiseks.

8.1 Põhilise mänguala kirjeldus ja analüüs

Antud lahendus on olemuselt uudne ning visuaalset poolt arvestades oma ajast ees. Projekti tegi keeruliseks asjaolu, et prooviti imiteerida pärisloodust. Seega ei saanud näiteks kõiki varjusid jõudluse parendamiseks välja lülitada. Tavaliselt on soovituslik kasutada kaadrisageduse huvides lihtsustatud kunstilist stiili, nagu näiteks VR mängus Job Simulator. Optimaalne kaadrisagedus peaks VR mängul olema vähemalt 60-90Hz või isegi 120Hz [26]. Antud projekt vajas seetõttu uuemat riistvara ning tuli teha järeleandmisi graafilise representatsiooni osas.

Mängu jõudlus (*performance*) sõltub arvuti võimekusest (graafikakaardist, protsessorist, jne) ning mängu oma seadistustest [26, lk. 282]. Esialgu VR kogemuse realistlikumaks tegemisel asetati mängumaailma palju erinevaid objekte, näiteks palju muru, põõsa ja lilleobjekte ning jäeti alles paljud objektivarjud, mis kokkuvõttes viisid kaadrisagedust madalamale (45Hz).

Laboris kasutati arenduses 900. seeria videokaardina peatükis 3 nimetatud GeForce GTX 980Ti. Sealsete katsetuste tulemusena ei saavutatud 45Hz puhul soovitud kaadrisagedust. Seega pidi autor otsima lisaks tehnilist ressurssi. Mektory VR laborisse saabus soovitud videokaart alles hiljem.

Võrdluseks 1000. seeria graafikakaardiga (GeForce GTX 1080) ning sama protsessorit kasutades jäi kaadrisagedus 90Hz juurde. Kliendi põhiliseks mängu jooksutavaks graafikakaardiks sai GeForce GTX 1070. Laboriarvutitega töötamisel pidi seetõttu

vähendama objektide hulka mängumaailmas, et tagada kindlam töötamine kõikides arvutites. Mängu jaoks otsustati igaks juhuks kasutada siiski vähemate objektidega versiooni (Joonis 50).

Joonis 50. Esialgne (üleval) ja uuendatud (all) mängumaailm.

Antud probleem vajaks veel täpsemat kontrolli erinevatel HTC Vive kasutavatel arvutitel. Tulevikus on vajadusel võimalik mänguala realistlikumaks muuta. Kliendi arvutis töötab programm korrektselt ning seega on kõige olulisem tagatud.

Mängija põhieesmärk on otsida objekte puu all, seega piirialadest väljaspool asuva ümbruse jälgimine ei ole antud mängus nii oluline. Antud mäng ei ole seiklusliku sisuga nagu on Mektory VR laboris loodud Martin Luige projekt [15]. Piirialadest väljaspool olevast ümbrusest ei sõltunud kuidagi mängu põhiülesande täitmine ning see on peamiseks põhjuseks, miks mängijad üldiselt mängutaseme visuaalsele poolele ei keskendunud. Seetõttu sobis uuendatud visuaalne lahendus, sest mängija saab lahendada oma põhilist ülesannet soovitud kaadrisagedusega.

Antud mängus töötab lehtede ja müntide genereerimine tõenäosusväärtustel. Üks variant oleks mäng jätta selliseks, sest paljudes mängudes kasutataksegi juhuslikkust mänguprotsessi huvitavamaks tegemisel. Juhuslikkus võib tekitada mängijas soovi mängu korduvalt mängida, sest iga kord on objektide mängu ilmumine erinev. Teine variant oleks luua lahendus, kus igal mängijal oleks võrdne võimalus punkte koguda, sest antud juhul genereeritakse mõnesse maailma rohkem objekte, kui teise. Selline variant eeldaks aga keerulisemat mänguloogikat ning jäi antud lõputöö ulatusest välja. Ideaalis võiks mängule teha lisatasemete valiku, kus keerukama mängu puhul tõstetakse genereeritavate objektide hulga tõenäosust suuremaks ning lihtsama mängu puhul tõenäosust väiksemaks.

Antud juhul on aga arendajate poolt leitud keskmine mängutase. Autori poolt on mängus loogika, mis kiirematel mängijatel annab võimaluse rohkem lehti korjata, sest kontrollitakse alati lehtede puudumist (vt Joonis 30, lk 42). Vähemkogenud mängijatel ei teki samuti üleküllust, sest objektidele on alati määratud teatud aeg, kui kaua need mängumaailmas püsivad (vt Joonis 34, lk 35). Lahendust on tiimiliikmete poolt korduvalt kontrollitud ning leitud on sobilikud parameetrid antud lahenduse realiseerimiseks.

Tulevikus võiks mõelda mängija pikkuse arvestamisele. Antud juhul on võetud arvesse autori enda pikkus (164 sentimeetrit), kus objekti kõrgeim punkt võib asuda. Seega pikematel mängijatel on eelis objektide korjamisel. Lühematel mängijatel on esinenud probleeme kõrgel asuvate objektide kättesaamisega. Sellepärast võiks tulevikus leida lahenduse, kus päritakse peakomplekti HMD kõrgus ruumis, ning luuakse lehti genereeriv *LeafSpawnerBox* mängija pikkust arvestades temale sobilikule kõrgusele. Pakutud lahendus looks võrdse võimaluse kõigile mängijatele objekte korjata.

8.2 Mängukarakteri arenduse analüüs

Mängukarakteri arenduse puhul oli oluline otsus kellaaja kuvamine *MotionController* randmele (vt Joonis 3, lk 27). VR mängude puhul on soovitatud lähtuda mõistmisest, et tegemist ei ole traditsiooniliste mängudega, kus liides kuvatakse mängija ekraanile. Kasutajaliidesed virtuaalreaalsuses peaksid eksisteerima mängumaailma osana [27]. Seega tundus antud lahendus sobiliku otsusena, sest rakendas eeltoodud põhimõtet.

Esialgu kuvati kell mängijale kaameravaatesse, mis segas mängule keskendumist, sest pidevalt muutusid sekundid ja minutid kasutaja silme ees. Üldiselt mängu läbimise mõttes ei ole oluline aega pidevalt jälgida, küll aga motiveerib kellaajast teadlik olemine mängijat kiiremini punkte koguma. Seetõttu on tegemist mängule olulise funktsionaalsusega, mida ei saa eemaldada. Randmele informatsiooni kuvamine tähendas, et mängijale anti valik aega jälgida. Kogemus on näidanud, et kogenumad mängijad vaatavad kella, kuid esmakordsed mängijad on tavaliselt unustanud seda teha. Seega pidi looma käekellale lisaks veel meeldetuletusi, et mängija, isegi kella vaatamata, saaks aru, palju aega mängu lõppemiseni on. Selle jaoks lisati mängu algusesse (kümneks sekundiks) ja keskele heli, mis imiteeris kella tiksumist. Selliselt lahendati aja näitamise probleem.

Viimase kümne sekundi meeldetuletuseks kuvatakse sekundid siiski kaameravaatesse ehk mängija silmade ette. Eesmärgiks on, et mängija oleks kindlasti teadlik lõppevast mängust. Viimaste sekundite kuvamine ekraanile ei ole arvatavasti kasutajale kõige vähemsegavam lahendus, kuid arendajad pidasid väga oluliseks mängijale nii konkreetselt kui võimalik selgeks teha, et mänguaeg hakkab lõppema. Võrreldes seda lahendusega, kus terve mänguaja vältel oli kasutajal kellaaeg silme ees, on kümme sekundit vähem häiriv. Info kuvamiseks on valitud ekraani ülaosa, seetõttu mängu terviku jälgimist see olulisel määral ei sega.

Tulevikus võiks mõelda kasutajale vähemhäiriva lahenduse suunas. Üheks lahenduseks võiks olla mängija ekraanilt sekundite kuvamine eemaldada ning viimase 10 sekundi kohta märguannete andmine täpsetelt ajastatud helidega. Lahenduse vastu on probleem, et tellijal ei ole alati olnud võimalust kõlareid kasutada. Lisaks on olnud mõnel rahvarohkel üritusel väline heli nii suur, et on summutanud mängusisesed helid. Seetõttu ainult helisid mängu lõpetamisel kasutada ei saa, sest mõni mängija ei pruugi neid kuulda. Ideaalse lahenduse puhul võiks ära kasutada ruumi, milles mängija on, ning mängija kaamerapositsiooni arvestades kuvada viimased sekundid mängija vaatevälja, kuid mitte täpselt silmade ette.

8.3 Õpetustoa analüüs

Mängu õpitavuse mõttes oli õpetustoa loomine kasulik otsus, sest tänu sellele said enamik mängijaid põhimängu ülesannetega hakkama. Tuli arvestada, et iga mängija

õpib erineva tempoga. Oli mängijaid, kelle jaoks ülesanded tundusid lihtsad ning õpetustoas viibimine ei kestnud üle minuti. Leidus ka mängijaid, kes ei saanud õpetustoas hakkama ning vajasid välist abi. Üldiselt oli probleem just esmase VR kogemusega inimestel.

Hilisemate testide läbiviimisel on nii projekti autor kui ka tiim nõustunud, et informatsiooni siseseintele kuvamine (vt Õpetustoa juhiseid Lisas 1) ei olnud kõige efektiivsem lahendus kasutajale informatsiooni omandamiseks. Võttes arvesse kasutajat, kes ei tea virtuaalreaalsuse seadmetest midagi, ei ole esmakordsed kasutajad võimelised seintelt juhiseid lugema. Kasutajale esitati korraga liiga palju informatsiooni. Üldiselt kasutajad ei loe seintel olevat infot ning soovivad kohe objektidega suhelda ilma funktsionaalsusi mõistmata.

Üheks probleemiks oli näiteks õpetamistoas ringi liikumine, mis tulenes pultide ebaefektiivsest kasutamisest. Tekkisid olukorrad, kus objekti juurde liikumise asemel mindi objektist mööda või ei saadud aru, kuhu objekt kadus. Seetõttu pidi suures ruumis inimese liikumisala piirama. Piiramine vajab planeerimist, kuhu inimest lubada ja kuhu mitte. Antud juhul sai probleem lahendatud (vt Joonis 37, lk 47), kuid võib vajada veel mängijate tagasisidet. Piiratud ala on vähendanud mängijate eksimise võimalust. Järeldus on, et kuigi virtuaalreaalsuses on liikumisele antud vabadust, peab arendaja oskama kasutajat suunata õigetele ülesannetele. Lisaks tuleks suunamist teha loomulikul viisil, nii et kasutaja ei tunneks ennast sunnitult teatud tegevusi tegema. Antud juhul ei ole suunatud esmakordsed kasutajad piisavalt edukalt juhiseid jälgima ning juhistelt õppima. Seetõttu tuleb esmakordsetele kasutajatele tutvustada pulte enne VR kogemust vahendaja abil. Ideaalis võiks programm suuta ise kasutajat õpetada sisend-väljund liideseid kasutama.

Üks kasutaja pakkus välja, et pultide kasutamiseks võiksid olla enne VR maailma sisenemist juhised. Selle jaoks loodi mängu hilisemas versioonis pärast registreerimisvaadet teiste tiimiliikmete poolt vaade, kus oli kuvatud kuppelhoones esitatud informatsioon monitori ekraanile. Informatsiooni kuvamine 2D kuvaril võis anda mängust parema ülevaate, kuid ei lahendanud ära õpetustoa kehva selgitava ruumi probleemi. Parema õpetuskogemuse loomiseks võiks kasutada näiteks passiivse virtuaalreaalsuse ideed, kus inimesele näidatakse enne, kuidas asjad õigesti töötavad ning alles siis palutakse kasutajal endal sama järele proovida [11, lk 2].

Seetõttu antud juhul ei oleks pidanud kasutama jooniseid seinal, vaid võinud virtuaalreaalsuses:

1. näidata animatsiooni või videot pultide kasutamisest;
2. anda mängijale ülesanne tegevus korrata kuni tegevus õnnestub;
3. täita õpetustoa objektide korjamise ülesanne;
4. suunata mängija edasi tegelikule mängualale.

Sellisel juhul oleks kasutaja keskendunud ühele ülesandele korraga, alguses pultide oskuslikule kasutamisele ning alles siis ülesandele. Näiteks oleks võinud luua eraldi mängutaseme ainult pultide kasutama õppimiseks ning antud õpetustuba kasutada mängu tundmaõppimiseks.

Õpetustoa lisaeesmärgiks oli mängus kasutavate objektide ja nende punktiväärtuste tundmaõppimine. Kasutajad, kes õppisid kiirelt või kellel oli varasem kogemus pultidega olemas, said edukalt õpetustoas objektide korjamisega hakkama, sest nad suutsid vajalikku informatsiooni ruumist otsida. Iga objekti korjamise õnnestumine oli kasutajale antud edasi nii heli kui helendava tausta muutumisega. Interaktiivset virtuaalreaalsust täiustabki tagasiside süsteem, et virtuaalreaalsust õppiv kasutaja teaks, kas tegi midagi õigesti või valesti [11, lk 3]. Antud lõputöös on tagasiside süsteemi eelkõige kasutatud kasutajale positiivse tagasiside andmiseks. Töös kastutatud helendava seina taust (vt Joonis 47, lk 53) oli piisavalt silmatorkav, seega enamik kasutajaid märkas suures ruumis objekti ning sai objektile vajutada. Edasi sai kasutaja teatud punktitulemuse, mida kuvati tema sõrme juures oleva numbriga ning objekti kadumist maailmast edastati heliga. Samuti kadus helendav taust koos heliga, et kasutaja saaks keskenduda järgmise objekti otsimisele. Seega otsene tagasiside lahendati edukalt.

Kasutaja sai punktide korjamise hetkel tagasisidet, kuid õpetamistuba oleks võinud objektide leidmisel kasutajaga rohkem suhelda. Eelkõige aitaks suhtlemine esmase kogemusega kasutajaid. Üheks lihtsamaks lahenduseks oleks pärast objekti korjamist helendavale taustale lisada informatsioon järgnevate tegevuste kohta. Teiseks võimaluseks oleks kasutajale kuvada objektide asukoht või teekond objektini pärast mõne aja möödumist, kui pole suudetud ülesannet täita. Kõige parem, aga keeruliselt

realiseeritav, oleks kasutada mõnda tehisintellektist kaaslast, kes liiguks mängijaga õpetamistoas kaasas ning ütleks kasutajale, mida ta järgmisena tegema peaks. Uuringutes on leitud, et passiivse virtuaalreaalsuse kasutamine inimese õpetamiseks on olnud positiivsete tulemustega, sest õpitakse juhendajat jälgides. Lisaks on järeldatud, et kui on olemas kaaslane, siis võib see õppimisel olla efektiivsem kui ainult interaktsioonidele tuginev rakendus [11, lk. 11]. Kolmanda variandi peale võiks kindlasti mõelda mõne keerulisema VR mängu loomisel, kus on rohkem erinevaid ülesandeid kui antud juhul ainult objektide otsimine ja korjamine.

8.4 Järeldus mängust

Valdavalt on mäng saanud head tagasisidet kasutajate poolt. Virtuaalreaalsuse kui ka teiste infotehnoloogiliste vahenditega loodud kogemuste kvaliteet sõltuvad ülesandest, mis nende vahenditega tehakse. Seega hea VR kogemuse jaoks on vaja selgelt püstitatud ülesannet [28]. Järelikult korralikust graafilisest lahendusest ning võimalusest objekte ise liigutada mängumaailmas ei piisa, vaid peab olema lisaks mõni täidetav ülesanne. Antud mängu puhul on punktide korjamise ülesanne olemas ja mäng töötab ilma vigadeta. Pultide oskuslikul kasutamisel on õpetustoast võimalik mängu edasi liikuda, et aja peale ülesannet täita. Need on põhjused, miks mäng on kasutajatele meeldinud.

Loodud virtuaalreaalsuse mäng on hetkeseisuga kasutuses Eestis Swedbanki Liivalaia kontoris ja Mektory hoone Swedbanki toas ning erinevatel Swedbanki üritustel. Huvi on tundnud ka Läti ja Leedu Swedbanki kontorid ning on kasutusel ka sealsetel erinevatel noorteüritustel. Eestis on rahvarohkematel üritustel mängu kasutatud näiteks Robotexi Tallinna Tehnikaülikooli esinduspunktis, Teadlaste öö Mektory VR labori esinduspunktis ning Võti tulevikku Swedbanki esinduspunktis.

9 Kokkuvõte

Töö põhieesmärgiks oli UE4 mängumootoris arendada VR mäng. Sisult on mäng võistlusliku eesmärgiga. Mängu loomisel kasutati mängumootori näidist, mis lahendas ära sisend- ja väljundliideste probleemi ning võimaldas tegeleda mängu sisu arendamisega. Arenduses kasutati visuaalset skriptimiskeelt (*blueprint*).

Kirjeldatud on *blueprint* kasutamise arendusprotsess, mis on kasulik õppematerjal ka teistele huvilistele. Arendusprotsessis on toodud palju näiteid, kuidas struktureerida lõputöös *blueprint* elemente: iga funktsionaalsus kirjutati eraldi sündmuse või funktsiooni sisse, otsiti viise algoritme esitada lühemalt ja efektiivsemalt, jälgiti selgust funktsioonikastikeste vedamisel. Esialgsega võrreldes kutsutakse vähem kordi välja erinevaid funktsioone ning on piisavalt arusaadav, et võimalusel arendada mängu edasi ka tulevikus.

Põhimängu vajalike liideste tundmaõppimiseks loodi tutvustav õpetustoa mängutase, kus kasutaja sai ilma ajaliste piiranguteta ülesannet täita. Õpetustuba täitis eesmärki, kuid informatsiooni edastamise puudujäägid selgusid programmi hilisematel katsetel kasutajatega. Lõputöös on välja pakutud ettepanekuid edasise parenduse suunas. Näiteks suunata kasutaja tähelepanu vajalikule informatsioonile kasutades passiivset virtuaalreaalsust või anda interaktiivseid juhiseid.

Virtuaalreaalsusega seoses sai kinnitust tähelepanek, et oma erisuse tõttu 2D rakendustest tuleb kasutajaliidesed 3D virtuaalreaalsuses siduda ruumi ja teiste meeltega, et kasutajale jääks realistlikum mulje teda ümbritsevast keskkonnast. Selleks loodi aja näitamiseks käekell. Valik aega jälgida anti mängijale, kuid meeldetuletuseks kasutati helisid ning kuvati viimaseid sekundeid mängijale ekraanile. Tulevikus võiks viimaste sekundite kuvamiseks samuti ruumi ja kasutaja positsiooni arvestada.

Käesolev VR lahendus on uudne, sest jäljendati realistlikku loodust. Seetõttu esinevad mängu virtuaalkeskkonnal tehnilised piirangud. Autor leidis, et kui mängijal on vaja keskenduda ülesandele piiriala sees, siis väljaspool piiriala olevale ümbrusele

pööratakse vähem tähelepanu. Seega võis kasutada graafiliselt lihtsamat mänguversiooni. Antud mängus on ka tulevikus edasiarendamise võimalused.

Antud VR mäng täidab edukalt oma ülesannet. Loodud on töötav süsteem, mis võimaldab mängijal registreeruda, õppida ning mäng edukalt läbida. Need on põhjused, miks mäng on kliendile ja kasutajatele meeldinud ning on aktiivselt kasutuses virtuaalreaalsuse tutvustamiseks Balti riikides.

Kasutatud kirjandus

[1] "Class Blueprint," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Blueprints/UserGuide/Types/ClassBlueprint. (Kasutatud 3.11.2018)

[2] "Blueprints," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-us/Engine/Blueprints. (Kasutatud 3.11.2018)

[3] "UGameInstance," Epic Games, 2004-2019. [Online]. Available: https://api.unrealengine.com/INT/API/Runtime/Engine/Engine/UGameInstance/index.html. (Kasutatud 4.11.2018)

[4] "Using Motion Controllers," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-us/Platforms/VR/UsingTouchControllers. (Kasutatud 12.10.2018)

[5] "Levels," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-us/Engine/Levels. (Kasutatud 10.11.2018)

[6] "Level Editor," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/UI/LevelEditor. (Kasutatud 6.11.2018)

[7] "Virtual Reality (VR) Market Size, Share - Segmented by Product Type, VR Technology, Applications, and Region - Trends, and Forecast (2019 - 2024)," Mordor Intelligence Report, 2018.

[8] M. Vasser, "VR dictionary (ENG - EST)," Estonian VR & AR Community, 2015. [Online]. Available: http://www.eevr.ee/?p=242. (Kasutatud 12.11.2017)

[9] I. Stojšić, A. Ivkov-Dzigurski, L.Ivanović Bibić, O. Maričić and S. Đukičin Vučković, "Possible Application of Virtual Reality in Geography Teaching," *Journal of Subject Didactics,* vol. I, no. 2, pp. 83-96, 2016.

[10] "Vive Tracker," HTC Corporation, 2011-2019. [Online]. Available: https://www.vive.com/eu/vive-tracker/. (Kasutatud 6.11.2018)

[11] M. Roussou and M. Slater, "Comparison of the Effect of Interactive versus Passive Virtual Reality Learning Activities in Evoking and Sustaining Conceptual Change," *IEEE Transactions on Emerging Topics in Computing,* pp. 1-12, 2017.

[12] N. Pares and R. Pares, "Interaction-Driven Virtual Reality Application Design," *Presence,* vol. 10, no. 2, pp. 236-245, 2001.

[13] G. Haud, "Development of Virtual Reality Environment in CryEngine on the Example of Tallinn Town Hall Square," Tallinn: Tallinna Tehnikaülikool, 2017.

[14] N. Burke, "Unity Blogs," Unity, 2018. [Online]. Available: https://blogs.unity3d.com/2018/04/26/how-to-get-the-most-out-of-the-new-unity-project-templates-in-2018-1/. (Kasutatud 10.11.2018)

[15] M. Luik, "Kolmemõõtmelise maailma loomine virtuaalprillidele," Tallinn: Tallinna Tehnikaülikool, 2015.

[16] T.-B. Toodo, "Õppematerjal mängu loomiseks mängumootoriga Unreal Engine 4," Tartu: Tartu Ülikool, 2016.

[17] "Material Editor UI," Epic Games, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-us/Engine/Rendering/Materials/Editor/Interface. (Kasutatud 6.11.2018).

[18] "Blueprint Editor Reference," Epic Games, Inc, 2004-2019. [Online]. Available: https://docs.unrealengine.com/en-US/Engine/Blueprints/Editor. (Kasutatud 7.11.2018)

[19] R. C. Martin, "Clean Code," Boston: Pearson Education, Inc, 2008.

[20] "The Flying Spaghetti-Code Monster," The Undead Dev, 2016. [Online]. Available: http://www.undeaddev.com/the-flying-spaghetti-code-monster/. (Kasutatud 20.10.2018)

[21] "Custom Events," Epic Games, Inc, 2004-2019. [Online]. Available: http://api.unrealengine.com/INT/Engine/Blueprints/UserGuide/Events/Custom/index.html. (Kasutatud 3.11.2018)

[22] "Virtual reality software market size worldwide from 2016 to 2019 (in billion U.S. dollars)*," Statista, 2018. [Online]. Available: https://www.statista.com/statistics/562428/global-virtual-reality-software-market/. (Kasutatud 4.12.2018)

[23] T. Looman, "VR Template," Epic Games, 2004-2019. [Online]. Available: https://wiki.unrealengine.com/VR_Template. (Kasutatud 15.11.2018)

[24] "GameMode," Epic Games, Inc, 2004-2019. [Online]. Available: https://api.unrealengine.com/INT/Gameplay/Framework/GameMode/index.html. (Kasutatud 19.11.2018)

[25] K. Stepan, J. Zeiger, S. Hanchuk, A. Del Signore, R. Shrivastava, S. Govindaraj and A. Iloreta, "Immersive virtual reality as a teacing tool for neuroanatomy," *International Forum of Allergy & Rhinology,* vol. 7, no. 10, pp. 1006-1013, 2017.

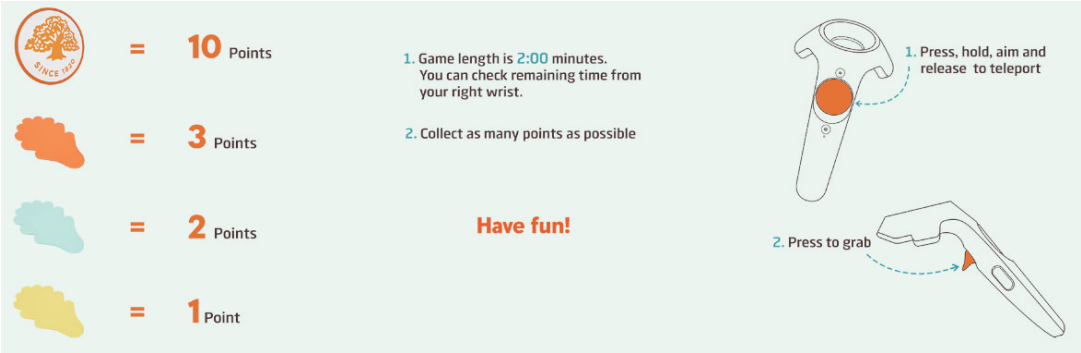
[26] M. McCaffrey, "Unreal Engine VR Cookbook: Developing Virtual Reality with UE4," United States: Pearson Education, Inc, 2017.

[27] D. Allen, "The Fundamentals of User Experience in Virtual Reality," Block Interval, 15 11 2015. [Online]. Available: http://www.blockinterval.com/project-updates/2015/10/15/user-experience-in-virtual-reality. (Kasutatud 11 10 2017)

[28] X. Zhanga, S. Jianga, P. Ordóñez de Pablosb, M. D. Lytrasc and Y. Sund, "How virtual reality affects perceived learning effectiveness: a task–technology fit perspective," *Behaviour & Information Technology,* vol. 36, no. 5, pp. 548-556, 2017.

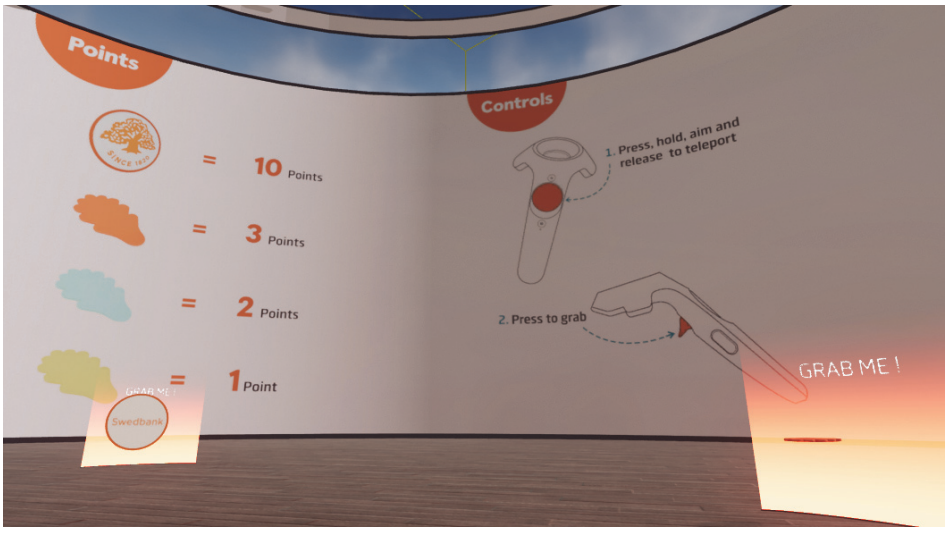
Lisa 1 – Õpetustoa juhised

Järgnevalt on tiimikaaslase poolt loodud joonis õpetustoa juhistest. Juhised on ingliskeelsed ning kirjeldavad objektide punktiväärtuseid, mängu ülesannet ning mängu jaoks kasutatavaid puldinuppe (Joonis 1-1).



Joonis 1-1 Õpetustoa juhised 2D kujul.

Õpetustoa juhiseid kuvati kuppelhoone siseseinal. Näide, kuidas õpetustoa siseseinal juhised välja nägid (Joonis 1-2).



Joonis 1-2. Õpetustoa juhised siseseinal.