TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Carl Custav Mäeorg 142761IAPB

2D ARVUTIMÄNG UNREAL ENGINE 4 MÄNGUMOOTORIL

Bakalaureusetöö

Juhendaja: Jaagup Irve
Tehnikateaduste magister

Tallinn 2018

Autorideklaratsioon

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

Autor: Carl Custav Mäeorg

10.01.2018

Annotatsioon

Bakalaureusetöö eesmärkideks on anda ülevaade arvutimängude ajaloost ning mängulisusega seotud teemadest ja võrrelda mängumootorite Unreal Engine 4, Unity ja GameMaker võimekusi 2D mängude arendamisel ning luua neist sobivaimaga 2D arvutimängu üks tase. Mängutasandi arendusprotsessi käigus kavandatakse taseme ülesehitus, luuakse mängija poolt juhitav karakter ning arvuti poolt kontrollitavad karakterid, arendatakse tasandi läbimist toetavad mänguelemendid, luuakse relvasüsteem ning tehakse visuaal- ja heliefektid. Valminud tasemel saab mängija liikuda, tulistada ja interakteeruda erinevate tasandi elementidega nagu köied, redelid, varisevad trepid, arvuti poolt kontrollitavad vastased. Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 30 leheküljel, 3 peatükki, 9 joonist, 2 tabelit.

Abstract

A 2D Computer Game in Unreal Engine 4

The aim of this thesis is to give a brief overview of the history of computer games, cover important topics related to gameplay, compare three popular game engines, Unreal Engine 4, Unity and GameMaker to analyze their capabilities for developing 2D games and develop a level of a 2D computer game using the most suitable game engine. The development of the game level consisted of creating a level design, developing a controllable character and computer controlled AI characters, designing various obstacles and platforms used in level design, developing a weapon system and creating visual and sound effects. The resulting level allows the player to move, fire bullets and interact with game level elements such as swinging ropes, ladders, collapsing stairs, enemy characters in a 2D environment. The thesis is in Estonian and contains 30 pages of text, 3 chapters, 9 figures, 2 tables.

Lühendite ja mõistete sõnastik

sprite	2D kujutis standardses rastergraafika pildifailivormingus (ingl *bitmap*), mida saab mängus kasutada, ilma et seda oleks vaja mingil viisil muuta. Neid kasutatakse 2D mängudes tavaliselt selleks, et kuvada mängutegelase elusid, mänguskoori, staatilisi taustaelemente. Selle abil saab luua ka tegelaste animatsioone, kus üks sprite on võrdeline ühe kaadriga animatsioonist [1].
tile	väike pilt, tavaliselt ristkülikukujuline või isomeetriline (kahedimensioonilist ruumi näidatakse teatud nurga alt, tekitades illusiooni kolmemõõtmelisest maailmast [2]). See on justkui pusletükk, mille abil saab ehitada suuremaid pilte ehk antud kontekstis mängumaailmasid [3].
tilemap	*tile*'de kogum, mida kasutatakse selleks, et luua mängumaailm [4].
tileset	erinevate korduvkasutatavate *tile*'de hulk, mida *tilemap* kasutada saab. Näiteks võib mängus olla vajadus kasutada eraldi *tileset*'i majade või koobaste jaoks [5].
spritesheet	standardses rastergraafika pildifailivormingus (ingl *birma*) fail, mis sisaldab mitmeid väiksemaid järjestikuseid pilte või *sprite*'e, mida kasutatakse selleks, et vähendada mängu mälukasutust ja parandada mängu jõudlust [6], [7].
flipbook	*spritesheet*'i ekvivalent, mida kasutatakse Unreal Engine 4 mängumootoris 2D animatsiooni loomisel [8], [9].
parallaks efekt	efekt, kus mängumaailma esiplaani ja tagaplaani osad liiguvad vaatleja suhtes erineva kiirusega, tekitades sellega ruumilisuse illusiooni [10]
build	protsess, mille käigus teisendatakse projektiga seotud failid tarkvara tooteks. Selle protsessi käigus võidakse lähtefailid kompileerida, kompileeritud failid pakkida kokkusurutud formaadiga failidesse, luua tarkvara paigaldamise fail ja luua, lisada või muuta andmebaasi skeemi või andmeid [11]

Sisukord

Jooniste loetelu

Tabelite loetelu

Sissejuhatus

Arvutimängud kui meelelahutusmeedium on olnud avalikkusele teada juba peaaegu pool sajandit. Mängutööstuse areng on olnud kiire ning selles valdkonnas tehakse pidevalt uusi läbimurdeid, pakkudes mängijale üha ehedamat kogemust. Arvutimängude arendusprotsess on töömahukas ning aeganõudev ja võib kesta aastaid ka siis, kui mänguga tegelevad suured ettevõtted või meeskonnad. Mängude arendamine nõuab palju teadmisi erinevatest valdkondadest, näiteks programmeerimine, graafiline ja heliline disain, füüsika, matemaatika, loo vestmisoskus. Valides arenduseks sobiva mängumootori, mis neil teemadel töötamist lihtsustab, aitab see kiirendada mängu arendusprotsessi.

Bakalaureusetöö eesmärkideks on anda lühike ülevaade arvutimängude ajaloost ning mängulisusega seotud teemadest, võrrelda enamkasutatavate mängumootorite Unreal Engine 4, Unity ja GameMaker võimekusi 2D mängude arendamisel ning luua 2D arvutimängu üks tase. Mängutaseme loomiseks on vaja lahendada järgmised alamülesanded:

1. Sobiva mängumootori valimine;
2. Taseme ülesehituse kavandamine;
3. Juhitava karakteri loomine;
4. Vastaste loomine;
5. Tasandi läbimist toetavate mänguelementide loomine;
6. Relvasüsteemi väljaarendamine;
7. Visuaal- ja heliefektide tegemine.

Töö esimeses osas tutvustatakse arvutimängude tausta. Teises osas analüüsitakse kolme enamkasutatavat mängumootorit, mis võimaldavad luua 2D arvutimänge. Kolmandas osas antakse ülevaade mängutaseme arendusprotsessist ja analüüsitakse saadud tulemusi.

1 Arvutimängud

Esimene teadaolev mänguks kasutatav masin toodi avalikkuse ette 1940. aastal New York'is ülemaailmsel tehnoloogia ja teaduse messil Edward Uhler Condon'i poolt. Mängu, mis põhines matemaatilisel strateegiamängul *Nim*, mängis kuue kuu jooksul messil ligikaudu 50 000 inimest. Mängud lõppesid teadaolevate andmete kohaselt 90% mängija kaotusega [12].

Esimene kodus kasutatav mängusüsteem loodi peaaegu 30 aastat hiljem 1967. aastal, kui Ralph Baer koos oma meeskonnaga avaldas mängusüsteemi prototüübi *Brown Box*. Seda sai ühendada televiisoriga, lubades kahel mängijal mängida, et abstraktseid kuupe kontrollides üksteist ekraanil taga ajada. *Brown Box*'i mängude hulgas olid ka ping-pong, kabe ning neli sportmängu. Konsooliga oli kaasas valguspüstol ja lisaseade golfi mängimiseks. *Brown Box* litsenseeriti Ameerika elektroonikafirma Magnavox poolt. 1972. aastal alustati mängukonsooli müüki Magnavox Odyssey nime all. Kolm aastat hiljem selle tootmine lõpetati, sest Mängukonsooli müüki saatis vähene edu, kuid see oli digitaalsete mängude ajastu algus [12].

1.1 Mis on arvutimäng?

Cristopher Crawford, paljude mängudisaini raamatute autor ning GDC (ingl *Game Developers Conference* ehk mänguarendajate konverents) asutajaliige, defineerib arvutimängu järgmiselt [13]. Arvutimäng loob lihtsustatud esituse reaalsusest, kuid peab olema reaalne ulatuses, et toetada mängija fantaasiat. Mängu eesmärgiks ei ole rõhutada detaile, erinedes nii simulatsioonist, vaid anda edasi mängudisaineri sõnumit. Raamatutega võrreldes on enamasti võimalik mänge korduvalt erinevatel viisidel läbi proovida, kuni suur osa mängu harudest on läbitud. Mängus on oluline ka konflikt, erinevad viisid seda lahendada ja võimalus reaalsust ohutul viisil kogeda [14].

Mänguarendaja ja ulmekirjanik Greg Costikyan defineerib arvutimängu aga nii. Arvutimäng muudab mängijaga suhestudes olekut ning on seotud kindla eesmärgiga.

Mängud peavad olema pingutust nõudvad, kuid raskuse osas peab olema tasakaal. Mäng põhineb kindlal struktuuril, mille määravad mängureeglid, kuid need peavad toetama mängija soovi erinevate strateegiate ja lähenemistega eksperimenteerida [15].

1.2 Arvutimängu neli põhielementi

Arvutimängu põhilisteks elementideks on mängumehaanika, taustalugu, esteetika ja tehnoloogia. Kõik need on mängu loomisel võrdselt vajalikud. Iga elemendi kirjeldamisel tuuakse näide arvutimängust *Space Invaders*, mille autor on Toshihiro Nishikado ning avaldati *Taito Corporation* poolt 1978. aastal [16].

Mehaanika on tegevuste hulk, mida kasutatakse mängu oleku muutmiseks [17]. *Space Invaders* mängus sai mängija tulistada vastutulistavaid tulnukate kosmoselaevu, mida oli 48, ning peita end kaitsekilpide taha, mille vastased said hävitada. Mäng kaotati kui mängija kosmoselaev hävis või tulnukad jõudsid mängija koduplaneedile. Mängus oli võimalus koguda ka punkte. Rohkem punkte andis kaugemal asuvate tulnukate tabamine. Boonuspunkte sai müstilist lendavat taldrikut tabades. Lisaks liikusid tulnukate laevad seda kiiremini, mida vähem neid alles oli [16].

Taustalugu on mängus avanev sündmuste jada, mis võib olla lineaarne või hargnev ja vastavalt reeglitele muutuv. *Space Invaders* mängu algidee põhines sellel, et mängija tulistas inimsõduritest koosneva armee pihta. *Taito Corporation*'is saadi aru, et selline taustalugu saadab mängijatele halva sõnumi ning seejärel seda muudeti [16].

Esteetika on element, mis on seotud mängija aistingutega. See on mängudisaini osa, millel on mängija kogemusega kõige otsesem seos. *Space Invaders*'i nüüdseks primitiivsena tunduv graafika, oli tol ajal novaatorlik. Kolme tüüpi tulnukalaevade kuvamiseks kasutati kahekaadrilist „marssimise" animatsiooni. Kuna ekraan värve ei kuvanud, kleebiti sellele värvilised kiled nii, et mängija laev ja kilbid olid rohelised, tulnukad valged ja lendav taldrik punane. Pinge tõstmiseks saatis tulnukaid südametukse laadne heliefekt, mis liikumise kiirenedes sagenes. Mängu jaoks kujundati spetsiaalsed kabiinid, et võimendada mängulugu ja saadavat emotsiooni [16].

Tehnoloogia on element, millel mängumehaanika baseerub, esteetika väljendub ja lugu jutustatakse. Valitud tehnoloogia seab piirid, mida mängus teha saab. *Space Invaders* oli

esimene videomäng, kus mängija sai võidelda tulnukate kosmoselaevade vastu ja seda võimaldas mängu jaoks eraldi loodud emaplaat [16].

1.3 Mängutegevused

Mänguarendaja Jesse Schell peab mängumehaanika oluliseks osaks mängus kasutatavaid tegevusi ja jagab need kaheks tüübiks. Neid tüüpe kirjeldades kõrvutatakse neid kabemänguga. Esimesed neist on operatiivsed tegevused, mis on mängutegelase põhitegevusteks. Kabes saab mängija sooritada kolme tegevust: liigutada kabenuppu edasi, hüpata üle vastase kabenupu, liikuda tammiga tagasi. Teist tüüpi tegevused on resultatiivsed, olles olulised suures plaanis ning vajalikud eesmärgini jõudmiseks. Seda tüüpi tegevusi on palju rohkem. Kabe puhul on mõned neist näiteks: kaitsta nuppu äravõtmise eest, liigutades teise kabenupu selle taha; survestada vastast tegema mittevajalikku hüpet; ohverdada kabenupp vastase petmiseks; liigutada kabenupp väljaku lõppu, et see tammiks teha. Jesse Schell kirjeldab, kuidas tegevused võivad mängu mõjutada [16]:

- Tegevuste rohkus – lisades mängule operatiivseid tegevusi, tekib interakteerumiseks rohkem võimalusi. Näiteks mäng, kus saab joosta, hüpata, osta, müüa, sõita ja ehitada, pakub rohkem põnevust võrreldes mänguga, kus saab ainult hüpata ja joosta. Kui operatiivsed tegevused omavahel hästi ei suhestu ja neid on liiga palju, muudab see aga mängu segaseks ja ebahuvitavaks;
- Tegevused, mida rakendada mitmel objektil – mängijal on põnevuse tundmiseks rohkem võimalusi kui näiteks relva saab mängus kasutada toidu hankimiseks, ukseluku või aknaklaasi purustamiseks, autorehve tühjaks laskmiseks või seintele kirjutamiseks;
- Eesmärkide saavutamine mitmel erineval viisil – kui eesmärgini saab jõuda vaid ühel viisil, kaotab mängija huvi ebatavaliste tegevuste ja strateegiate vastu ka siis kui kasutada on palju esemeid ja tegevusi. Näiteks kui võideldes koletisega, on palju erinevaid viise tema võitmiseks, loob see mängu jaoks lisaväärtuse;
- Objektide rohkus – tegevuste ja objektide rohkusest sõltub resultatiivsete tegevuste suurus. Näiteks kabemängu teeb huvitavaks asjaolu, et mängulaual on palju nuppe, mida saab liigutada ja ohverdada.

1.4 Premeerimise ja karistamise viisid arvutimängudes

Mängijat tunnustatakse premeerides, karistuse eesmärgiks on aga väljakutsete loomine. Selleks on erinevaid võimalusi ning neid võib omavahel kombineerida. Paremaks vahendiks mängulisuse suurendamiseks peetakse aga preemiat. Näide sellest on mängus *Diablo*, kus mängijad pidid oma võimete säilitamiseks toitu koguma. Mängija edukus sõltus üsna igavast toimingust, mille tegematajätmisel olid nad nõrgestatud. Mänguarendajad muutsid süsteemi nii, et mängija karakter ei oleks kunagi näljane, kuid süües võimenduksid ajutiselt tema võimed. Karistuse preemiaks muutmine meeldis mängijatele palju rohkem. Järgnevalt on välja toodud põhilised viisid mängija premeerimiseks ning karistamiseks [16]:

- Ülistus ja alandus – sel viisil antakse mängijale teada, kas ta on olnud edukas või mitte, tehes seda näiteks mõne heliefekti, animatsiooni või fraasiga;
- Punktide võitmine ja kaotus – punktide võitmine võib olla tingimuseks mõne teise preemia saamisel, kuid tihti on see juba piisav preemia, näiteks kui punktid kajastuvad tulemustabelis. Punktide kaotus on aga üks raskemaid karistusi;
- Pikendatud ja lühendatud mängu kestus – preemiana antakse mängijale lisaelusid või pikendatakse mänguaega, kuid karistusena vähendatakse mängija elude hulka või mänguaega;
- Võimete saamine ja kaotamine – preemiana võib sel viisil näiteks mängus *Super Mario World* suuremaks kasvada või mängus *Quake* spetsiaalseid relvi omandada. Vastupidiselt pidi kaotaja mängus *Ultima Online* leidma elluärkamiseks tee teatud altarini, kuid sellele liigselt aega kulutades, kaotas mängija väärtuslikke oskuspunkte;
- Ressursside andmine ja äravõtmine – mänguks vajalikke ressursse nagu raha, esemeid või elusid saab mängijale anda või ära võtta;
- Ligipääs uude asukohta ja tagasilöök – preemiana antakse mängijale juurdepääs mõnda uude tasemesse või ka võti, millega uude asukohta pääseda. Karistusena võidakse mängija saata tagasi taseme algusesse.

2 Mängumootorite analüüs ja võrdlus kahemõõtmeliste mängude arendamisel

Tänapäeval on saadaval palju mängumootoreid kahemõõtmeliste mängude arendamiseks, millel on erinevad võimekused, litsentsid ning lisatöövahendid. Järgnevalt analüüsitakse kolme enimkasutatavat mängumootorit, Unity, Unreal Engine 4 ja GameMaker, mille abil saab luua 2D arvutimänge. Peatüki lõpus esitatakse mängumootorite võrdlus kokkuvõtlikult kahe tabelina.

2.1 Unity

Unity on *Unity Technologies* poolt loodud mängumootor, mis loodi 2005. aastal [18]. See võimaldab arendada nii kahe- kui ka kolmemõõtmelisi mänge, toetades mängude ja rakendusprogrammide arendamist mitmel platvormil – mobiilidel, mängukonsoolidel ning arvutitel. Lisaks pakub Unity mänguarenduse produktiivsuse tõstmiseks ning mängude publitseerimiseks ka muid teenuseid – *Unity Asset store, Unity Cloud Build, Unity Game Performance Reporting, Unity Ads ja Unity Everyplay*. Unity mängumootorit kasutab ühes kuus ligikaudu 700 000 kasutajat [19].

2.1.1 Maailma loomise võimalused

Unity mängumootor võimaldab kasutada *tilemap*'e ning animeeritavaid *tile*'e. *Tilemap*'ide jaoks on loodud baasfunktsionaalsus tile'de kokkusobitamiseks ning *tile*'i põhiste pintslite loomiseks. Kasutada saab ka töövahendit *sprite mask*, et teatud *sprite*'i osasid peita, ja *9-Slicing* tehnikat, et *sprite*'i suurust muuta nii, et selle tekstuurimuster korduks [20], [21].

2.1.2 Animatsioonid

2D animatsioone saab luua otse mängumootoris, kasutades tasuta Unity skeletilise animatsiooni töövahendit *Unity Anima2D* [22]. 2D animatsioonide loomiseks on

mängumootoril ka tasulised 2D skeletilise animatsiooni lisatöövahendid *Puppet2D* ja *Spine*, mis on leitavad *Unity Asset Store* veebikeskkonnast [23], [24].

2.1.3 Füüsika

Unity mängumootoril on 2D füüsikamootor, mille lamendatud (ingl *flattened*) komponendid on enamasti 3D füüsikakomponentide ekvivalendid [25]. *Rigidbody 2D* komponent muudab mänguobjekti füüsikamootori poolt kontrollitavaks ning võrreldes *RigidBody 3D* tüübiga piiratakse objekti liikumine xy-tasandile. Komponendil saab seadistada füüsikalisi omadusi, näiteks mass, mõjutatavus gravitatsioonist [26].

Collider'ite abil defineeritakse objektile ääristav kujund, mille abil tuvastatakse objektide vahelisi kokkupõrkeid. *Collider*'i tüüpe, mida saab *Rigidbody 2D* komponendiga siduda, on kuus. Ringi, nelinurga ja kapsli loovad vastavalt tüübid *Circle Collider 2D, Box Collider 2D* ja *Capsule Collider 2D***.** Keerukamad neist on *Edge Collider 2D, Polygon Collider 2D*, mille põrkekujunditeks on vastavalt keerukam vabas vormis loodud avatud või suletud kujund. *Composite Collider 2D* põrkekujundil puudub defineeritud kuju ning selle saab luua *Edge Collider 2D* ja *Polygon Collider 2D* ühenditena [27].

Mängumootoris kasutatakse objektide omavaheliseks sidumiseks järgmisi liigendeid: *Distance Joint 2D, Fixed Joint 2D, Friction Joint 2D, Hinge Joint 2D, Relative Joint 2D, Slider Joint 2D, Spring Joint 2D, Target Joint 2D, Wheel Joint 2D* [28].

Põrkekujunditele füüsikaliste jõudude avaldamiseks on mängumootoris kasutusel mitmeid füüsikalisi komponente. *Surface Effector 2D* rakendab objektile mõõda tasapinda jõudu. *Point Effector 2D* simuleerib objektidele teatud asukohas külgetõmbe- või tõukejõudu. *Buoyancy Effector 2D* abil saab luua vedeliku simulatsioone. *Area Effector 2D* rakendab jõudu teatud alas mistahes suunas ja suuruses. *Platform Effector 2D* seadistab platvormide käitumist, näiteks karakteri hüppamist enda kohal olevale platvormile, ilma et selle põrkekujund teda takistaks [29], [30].

2.1.4 Visuaalefektid ja heli

Visuaalefekte saab mängumootoris luua osakeste süsteemi komponendiga. See komponent koosneb paljudest osakestest, mille hulka saab määrata. Lisaks on seadistatavad osakeste parameetrid, näiteks värvus, suurus, kiirus, mis võivad ka ajas

muutuda [31]. Efektide loomiseks saab kasutada lisatöövahendeid *2DxFX: 2D Sprite FX* ja *2D Dynamic Lights And Shadow*, mis on leitavad *Unity Asset Store* veebikeskkonnast [32], [33].

Unity mängumootoris saab luua ruumilisi 3D helisid, miksida helisid reaalajas ja luua helide hierarhiaid. Mängumootor toetab AIFF, WAV, MP3 ja Ogg helifailivorminguid ning *tracker* mooduli failivorminguid nagu it, s3m ja xm. *Tracker* moodulid on helinäidiste failid, mis on eelnevalt modelleeritud ning programmiliselt järjestatud. Need sarnanevad MIDI failidega, sest on mahult väikesed ning sisaldavad heliridu, milles on informatsioon, millal instrument mängib, kui tugevalt ning mis kõrgusega. Erinevalt MIDI failidest sisaldavad need kõrgekvaliteedilisi helinäidiseid. Kasutades *Audio Mixer* töövahendit, saab helisid miksida, grupeerida ja neile efekte rakendada. *Audio Mixer*'i muutujate seadistusi saab erinevate olekutena (ingl *snapshot*) salvestada, et näiteks luua sujuv üleminek ühest muusikateemast teise [34].

2.2 Unreal Engine 4

Unreal Engine mängumootori esimene versioon loodi 1995. aastal *Epic Games* poolt. Seda on kasutatud sadade mängude loomiseks erinevatele platvormidele [35]. Mängumootor toetab platvorme nagu Windows PC, PlayStation 4, Xbox One, Mac OS X, iOS, Android, Linux, SteamOS, HTML5 ning virtuaalreaalsuse seadmed (SteamVR, HTC Vive, Oculus Rift, PlayStation VR, Google VR, Daydream, OSVR ning Samsung Gear VR) [36]. 2D ja 2D-3D hübriidmängude arendamiseks kasutab Unreal Engine 4 *sprite*'de põhist süsteemi *Paper 2D* [37].

2.2.1 Maailma loomise võimalused

2D mängumaailma saab disainida kasutades *tilemap*'e ning seda saab teha mitmekihiliselt [38]. Mängutasandi loomiseks saab kasutada ka kontrollpunktide abil kõverjoonelise maastiku moodustamist. *Tilemap*'id kui ka kõverjoonelise maastiku moodustamise vahendid on mõlemad eksperimentaalses faasis [39]. Samas alustati 2016. aastal mängu *The Siege and the Sandfox* arendamist, mille maailma loomisel kasutatakse *tilemap*'e [40].

2.2.2 Animatsioonid

Mängumootoris kasutatakse 2D animatsioonide loomiseks *flipbook*'e. Need on pildiseeriatest koosnevad objektid, kus iga kaader on *sprite*, mis tuleb eelnevalt mängumootorisse importida [9]. Skeletiliste animatsioonide loomiseks saab lisaks kasutada tarkvara *Spine* ja *Creature*, millel mõlemal on Unreal Engine 4 mootori tugi [24], [41].

2.2.3 Füüsika

Füüsikat on võimalik mängumootoris seadistada füüsikateegil *Box2D*, kuid selle kasutamine on eksperimentaalses faasis. Seepärast kasutatakse 2D mängudes füüsika seadistamiseks 3D füüsikamootorit *PhysX* [37]. UE4 kasutab füüsika simuleerimiseks 3D võrekujundeid, *Physics Body*'sid, mida saab kasutada näiteks karakteri liikumise piiramiseks kindlale tasandile [42].

Sprite'le defineeritakse teiste mänguobjektidega kokkupõrke tuvastamiseks põrkekujund. Objektile saab seadistada põrkekujundi 2D füüsika põhjal, kasutades eksperimentaalfaasis *Box2D* teeki ning kasutades *PhysX* 3D füüsikamootorit, mille käigus luuakse 3D põrkekujund, mis lamendatakse 2D põrkekujundiks. Põrkekujundi loomiseks saab kasutada järgmisi viise: *Source Bounding Box*, kus põrkekujundiks on nelinurk; *Tight Bounding Box*, kus nelinurkne põrkekujund tekitatakse võimalikult *sprite*'i lähedale; *Shrink Wrapped*, kus põrkekujundiks on hulknurk, mis järgib *sprite*'i kontuure ning *Fully Custom*, kus põrkekujundit saab seadistada käsitsi ümber *sprite*'i hulknurka luues [37]. Lisaks saab objektile vastavalt kasti-, kapsli- ja kerakujulist põrkekujundit luua ka 3D komponentide *Box Component*, *Capsule Component*, *Sphere Component* abil [43].

UE4 kasutab füüsika seadistamiseks ka järgmisi komponente: *PhysicsConstraintComponent* – kahe objekti füüsikaliseks sidumiseks; *PhysicsHandleComponent* – objektide ülestõstmiseks ja liigutamiseks; *PhysicsThrusterComponent* – ettemääratud jõuga objektide liigutamiseks teatud suunas ja *RadialForceComponent* – objektile pideva või konstantse jõu avaldamiseks [42].

2.2.4 Visuaalefektid ja heli

Visuaalefektide loomiseks kasutatakse mängumootoris osakeste süsteemi redaktorit *Cascade*. Redaktoris saab määrata, millist efekti versiooni sõltuvalt kaamera kaugusest kuvatakse (ingl *level of detail*). Efektide loomine on moodulipõhine, kus iga moodul mõjutab spetsiifilist osakeste käitumise aspekti, võimaldades ajas muuta osakeste parameetreid, näiteks loomise asukohta, liikumisviisi, suurust, värvust. Efekte saab luua ka graafikaprotsessori (ingl *GPU*) põhiselt, et korraga efektiivselt sadu tuhandeid osakesi simuleerida. Osakestele saab määrata valgusallika ja kasutada osakesena ka 3D mudeleid. *Cascade* redaktorit saab kasutada vektorväljapõhiste efektide loomiseks. Vektorväljad on ühtlased vektorite võrgustikud, mis mõjutavad osakeste liikumist [44]. Lisaks on võimalik efekti põhjal luua mitmeid instantse, et dünaamiliselt mängus näiteks nende värvust muuta [45].

Mängumootor toetab helifaili vorminguna 16-bitiseid WAV faile. Importides heliefekti mängumootorisse, saab sellel muuta helivaljust ja -kõrgust ning sumbuvuse omadusi. Liithelide seadistamiseks kasutatakse mängumootoris redaktorit *Sound Cue*. Selles on võimalik heliefekte omavahel kombineerida ning modifitseerida. Töövahendite *Dialogue Voice* ja *Dialogue Wave* abil saab luua mängusiseseid dialooge, määrata neile subtiitreid ning tõlkeid ja neid mängumootori abil esitada [46].

2.3 GameMaker

GameMaker on mängumootor, mis loodi 1999. aastal arvutiteadlase Marc Overmars poolt. Algselt oli see kavandatud 2D animatsioonide loomise programmina Animo, kuid järk-järgult arenes sellest tööriistakomplekt mängude arendamiseks [47]. Selle mängumootoriga saab luua mänge platvormidele nagu Windows desktop, Mac OS X, Ubuntu, Android, iOS, fireTV, Android TV, Microsoft UWP, HTML5, PlayStation 4 ja Xbox One. Mänge saab arendada C keelel põhineval GML programmeerimiskeeles ja ka koodi kirjutamata, kasutades visuaalset lohistamiskeskset süsteemi [48].

2.3.1 Maailma loomise võimalused

Kõik mängumootori mängumaailmad disainitakse ruumide loomise redaktoris, kus mänguobjektid, *sprite*'d, *tile*'d ja taustad võivad olla jaotatud eri kihtidesse. Nii *sprite*'e kui ka *tile*'e saab joonistada otse mängumootori redaktoris *Image Editor*.

Mängumootoris on võimalik kasutada *tileset*'e, luua animeeritavaid *tile*'e ning *tile*'dest komplekteerida pintsleid, et *tile*'dest koosnevaid suuremaid objekte luua. Lisaks saab kasutada töövahendit *Auto Tile,* et eri *tile*'ed omavahel tasandit luues sobituks [49].

2.3.2 Animatsioonid

Mängumootoris saab animatsioone luua redaktoris *Image Editor* ka ajaliini (ingl *timeline*) töötades. Selliselt kandub joonistus kõikidesse animatsiooni kaadritesse [50]. Lisaks toetab mängumootor skeletiliste animatsioonide loomise tarkvara *Spine* [24].

2.3.3 Füüsika

Mängumootori füüsika põhineb avatud lähtekoodiga füüsikateekidel *Box2D* ja *Liquid Fun* [51]. Füüsikasüsteem on vaikimisi selline, kus iga üksiku objekti kokkupõrget tuleb kontrollida ja sellele reageerida. Kui mängus on palju objekte, kasutatakse selleks füüsikamaailma, mis määratakse mängu ruumile, seadistades kõikidele selles asuvatele objektidele füüsikamaailma reeglid. Objektile füüsikaliste parameetrite seadistamiseks peab see olema seotud kinnitusrakisega (ingl *Fixture*). Kinnitusrakis seob kuju või vormi objekti instantsiga ja seadistab sellele füüsikalised omadused. Kinnitusrakised võivad olla nelinurksed, ringikujulised, hulknurksed või defineeritavad punkt-punkti haaval [52].

Füüsikamaailmas saab defineerida erinevaid jõude nagu gravitatsioon, impulss, pidevad jõud, väändejõud ja nurkimpulss. Mängumaailma objektide maailmaga või omavaheliseks sidumiseks kasutatakse liigendeid (ingl *Joints*): *distance joint, revolute joint, prismatic joint, pulley joint, gear joint, rope joint, wheel joint, weld joint* ja *friction joint* [52].

Liquid Fun füüsikateegi abil saab kasutada osakesi, mis võimaldavad simuleerida pehmeid kehasid. Pehme keha on kujund, mis moodustatakse suure hulga osakeste abil, mis säilitavad ühtekuuluvuse ja osakeste omavahelise koosmõjuna tekitavad pehme keha illusiooni [52].

2.3.4 Visuaalefektid ja heli

Mängumootoris efektide kasutamiseks tuleb luua osakeste süsteem, kus erinevaid kasutatavaid osakeste tüüpe hoida. Efektide jaoks peavad olema selleks eelnevalt defineeritud osakestele tüübid, millel saab määrata värvust, läbipaistvust, suurust ning

liikumist. Neid tekitatakse osakeste tüübi põhjal efekti jaoks teatud juhuslikkusega. Võimalik on kontrollida ka osakeste voo sagedust [53].

GameMaker mängumootoris saab helisid seadistada redaktoris *Sound Editor*. Mängumootor toetab helifailivorminguid nagu WAV, MP3 ja Ogg. Helisid saab jagada gruppidesse [49]. Gruppi kuuluvaid helisid saab lisaks *Sound Mixer* töövahendit kasutades eraldiseisvalt mängida ning liitheli jaoks nende helitugevust koos seadistada [54]. Helide esitamiseks 3D ruumis kasutatakse *Audio Emitter* ja *Audio Listener* vahendeid. *Audio Emitter* võimaldab helidel reaalajas muuta helikõrgust ning määrata neile efekte nagu Doppleri efekt [55].

2.4 Mängumootorite võrdlus

Eelnevates peatükkides analüüsiti mängumootorite Unreal Engine 4, Unity ja GameMaker võimekusi ning lisatöövahendeid, lähtudes mängumootorite 2D mängude loomise võimalustest. Saadud analüüsist tulenevad andmed esitatakse kompaktsel kujul (Tabel 2). Lisanduvalt võrreldakse eelnimetatud mängumootorite finantsilisi näitajaid, lähtekoodi kasutamisvõimalust ning antakse ülevaade nende poolt kasutatavatest programmeerimiskeeltest (Tabel 1).

Tabel 1. Mängumootorite võrdlus finantsiliste näitajate, lähtekoodi kasutamisvõimaluse ning kasutatavate programmeerimiskeelte kohta.

	Litsents	Litsentsi hind	Max aasta tulu	Kasutusmaks	Lähtekood	Keeled
Unreal Engine 4	Unreal Engine 4 License [57]	Tasuta [36]	Piiramatu [36]	5% kvartalis ületades 3000$ aastas [36]	Avatud [36]	C++, *Blueprint Visual Scripting* [57]
Unity	Unity Plus [58]	Tasuta [58]	100 000 $ [58]	Ei [58]	- [58]	C#, UnityScript [59]
	Unity Personal [60]	32€ kuus [60]	200 000 $ [60]	Ei [60]	- [60]	
	Unity Pro [61]	115€ kuus [61]	Piiramatu [61]	Ei [61]	Tasuline [61]	
GameMaker	Trial [62]	Tasuta, püsiv [62]	- [63]	Ei [64]	Kinnine [65]	Visuaalne lohistamiskeskne süsteem, GML programmeerimiskeel [66]
	Windows	39$, 12 kuud [62]	Piiramatu [67]			
	Mac					
	Desktop	99$, püsiv [62]				
	Fire (Amazon Appstore)					
	Web	149$, püsiv [62]				
	Mobile					
	Universal Windows Platform	399$, püsiv [62]				
	Playstation 4	799$, 12 kuud [62]				
	Xbox One					
	Ultimate	1500$, 12 kuud [62]				

Tabel 2. Mängumootorite võrdlus mängumaailma, animatsioonide, visuaalefektide ja heli loomise ning füüsika kasutamise võimaluste kohta.

Mängumootor	Unreal Engine 4	Unity	GameMaker
Maailma loomine	*Sprite*'d, eksperimentaalses faasis *tilemap*'id, kõverjoonelise maastiku loomine	*Sprite*'d, *tilemap*'id, animeeritavad *tile*'d, *tile*'de pintslid, tile'de sobitamine, *sprite mask*, *9-Slicing* tehnika	*Sprite*'d, *tileset*'id ruumide loomiseks, animeeritavad *tile*'d, *tile*'de pintslid, *tile*'de sobitamine.
	Sprite'd, *tile*'d tuleb eraldi tarkvaraga luua		Saab mängumootoris luua
Animatsioonid	Skeletilise animatsiooni lisatöövahendid *Spine* ja *Creature*	Tasuta skeletilise animatsiooni töövahend *Unity Anima2D*, skeletilise animatsiooni lisatöövahendid *Puppet2D* ja *Spine*	Animatsioonide joonistamine mängumootoris, skeletilise animatsiooni lisatöövahend *Spine*
Füüsika	2D mängule soovitatav 3D füüsikamootor	Eraldi 2D füüsika komponendid	2D füüsikateegid Box2D ja Liquid Fun
Visuaalefektid	Osakeste süsteemi redaktor *Cascade*, moodulipõhine, graafikaprotsessori põhised efektid, valgusallikaga osakesed, vektorväljapõhised efektid, dünaamiline muutmine mängus	Osakeste süsteemi komponent, lisatöövahendid *2DxFX: 2D Sprite FX* ja *2D Dynamic Lights And Shadow*	Efektide jaoks programmikoodis osakeste tüüpide loomine, et genereerida efekti osakesi
	Kohene muudatuste nägemine		Muudatuste nägemiseks koodi kompileerimine
Heli	Liithelide loomiseks redaktor *Sound Cue*, dialoogi töövahendid *Dialogue Voice* ja *Dialogue Wav*e, tõlgete süsteem	Grupeerimine, *Audio Mixer*'iga helide miksimine, efektide lisamine. *tracker* moodulid, *snapshot*'id	Helide grupeerimine, *Sound Mixer*'iga helitugevuse reguleerimine, 3D helide ja efektide jaoks *Audio Emitter* ja *Audio Listener* vahendid
	16-bitine WAV	AIFF, WAV, MP3 ja Ogg, tracker mooduli vormingud it, s3m ja xm	WAV, MP3 ja Ogg.

3 Mängutaseme arendusprotsess

Mängutaseme ja seda toetava funktsioonaalsuse arendamiseks seati järgmised eesmärgid:

- mängutaseme ülesehitamiseks tuleb joonistada erinevad taseme osad nagu sambad, tasapinnad, trepid, väiksemad detailid, suurema plaani taustad;
- mängijale väljakutsete pakkumiseks tuleb taseme keerukamaks läbimiseks luua erinevaid takistusi ning mänguelemente nagu redel ja köis, staatilised ja kukkuvad kivid, vedru, patrulliv tasand ja rippumise punkt;
- takistuste ületamiseks peab karakter saama lisaks ronida, rippuda ja kiikuda;
- mängutasemele tuleb luua kolm arvuti poolt kontrollitavat vastast nagu tulistav vastane, patrulliv vastane ning keerukama funktsionaalsusega peavastane;
- mängukarakteri vahendiks vastastega võitlemiseks on relvad ning nende kasutamiseks tuleb välja arendada relvasüsteem koos kolme kasutatava relvaga;
- mängu esteetilisemaks muutmiseks tuleb luua heliefektid, muusika ning mängutegelaste jaoks animatsioonid.

Lõputöö raames valmiva mängutaseme kohta on loodud videodemonstratsioon (Lisa 4 – Arvutimängu taseme demonstratsioon) Arendusprotsessi käigus esines mitmeid probleeme ning suuremad neist on mainitud lisas (Lisa 3 – Arendusprotsessi suurimad küsimused, takistused ning probleemid). Alljärgnevalt tutvustatakse mängu realiseerimisel kasutatud lahendusi ning põhjendatakse nende valikut.

3.1 Mängumootori valimine

Loodava mängu arendamiseks oli valida kolme mängumootori vahel: Unity, Unreal Engine 4 (UE4) ja GameMaker. Kõik need on laialdaselt kasutusel ning võimaldavad luua 2D arvutimänge, pakkudes arvukaid näiteid olemasolevate mängudena.

Sobiva mängumootori valikul oli esimeseks kriteeriumiks, et mängumootori litsents ei oleks tasuline. Tähtsuselt järgmiseks kriteeriumiks oli, et mängumootoris oleks lihtne ja mugav luua visuaal- ja heliefekte. Kolmandaks määravaks kriteeriumiks sobiva mängumootori valimisel oli, et mängumootori lähtekood oleks avatud ning seda oleks võimalik vajadusel modifitseerida. Eelnevate kriteeriumite põhjal osutus kõige sobivamaks mängumootoriks Unreal Engine 4. Lisaks on UE4 mängumootori puhul eeliseks, et mänge saab arendada kahel viisil, kasutades C++ programmeerimiskeelt või *Blueprint* visuaalse skriptimissüsteemi.

3.2 Mängu baasprojekt

Unreal Engine 4 (UE4) mängumootor võimaldab luua baasprojekte erinevatele mängužanritele nagu näiteks esimeses isikus tulistamismängud (ingl *First Person Shooter*), pealtvaates mängud (ingl *top down*), virtuaalreaalsuse mängud ja 2D mängud. Baasprojekt seadistab vastava mängužanri jaoks kasutatavad teegid ja lisamoodulid ning loob tavaliselt baasfunktsionaalsusega mängutegelase, seadistades ka kaamera. 2D mängu jaoks seadistatakse UE4 versiooni 4.17.2 kohaselt järgmised osad:

- **GameMode klass** – klass, mida kasutatakse põhiliselt matšipõhiste mitmikmängude reeglite määramiseks. Sellega määratakse vaikimisi näiteks mängija alustamispunktide valimise ja matši oleku loogika [69];
- **Karakteri klass** – klass, milles piiratakse karakteri liikumine kahele teljele, määratakse tema põhilised liikumise kiirused ja näiteks sõltuvus gravitatsioonist, seadistatakse mängutegelase põrkekujund ja kaamera komponent. Lisaks luuakse minimaalne animatsioonide olekumasin ning seadistatakse kasutaja sisendile vastavad funktsioonid nagu hüppamine ja horisontaalselt liikumine;
- **Projekti klass** – klass, kus mängu projekt implementeeritakse moodulina, võimaldades seda UE4 mängumootori abil arendada [70];
- **Projekti mooduli build.cs konfiguratsioonifail** – fail, kus määratakse, kuidas projekti moodul *build*'itakse defineerides selleks vajalikud moodulisõltuvused ja määrates kasutatavad lisateegid [71];
- **Target.cs konfiguratsioonifailid** – failid, mis määravad, kas *build* protsessi käigus luuakse mäng, klient, server, redaktor või programm. Failis saab määrata

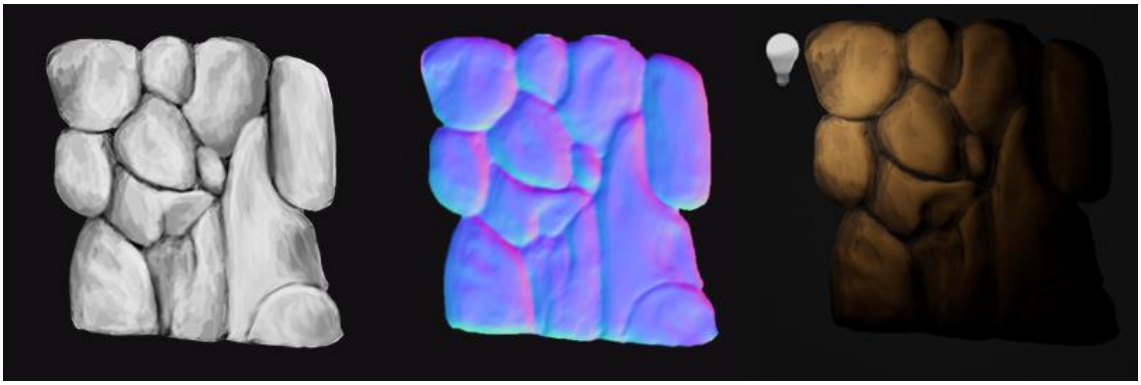
erinevad seadistused, näiteks kasutatav platvorm või kas build'i lisada ka erinevaid lisateeke [71].

3.3 Maailma loomine

Mängutasandi graafilise osa kujundamiseks kasutatakse tarkvara Adobe Photoshop, mille PSD-vormingus faile saab otse mängumootoris kasutada [73]. Mängumootor suudab faili sisu muudatusi koheselt kuvada. See võimaldab mängudisaineril elemente joonistades neid koheselt ka mängumootoris näha, säästes sellega palju aega.

3.3.1 Maailma elementide kuvamine

Ruumilisuse illusiooni saavutamiseks kasutatakse töös osade elementide kuvamisel normaalikaarti, kus iga piksel hoiab informatsiooni tasapinna kalde kohta. Seda kasutatakse kõrgeresolutsiooniliste detailide simuleerimiseks madala resolutsiooniga 3D-mudelitel. Selliselt luuakse illusioon palju detailsemast mudelist, mudeli geomeetriat muutmata [74]. Normaalikaartide abil saab illusiooni kolmemõõtmelisusest tekitada ka kahemõõtmelistel piltidel.



Joonis 1. Loodud kiviseina tekstuur (vasakul), selle normaalikaart (keskel) ja nende põhjal loodud materjaliga *sprite* element Unreal Engine 4 mängumootoris (paremal)

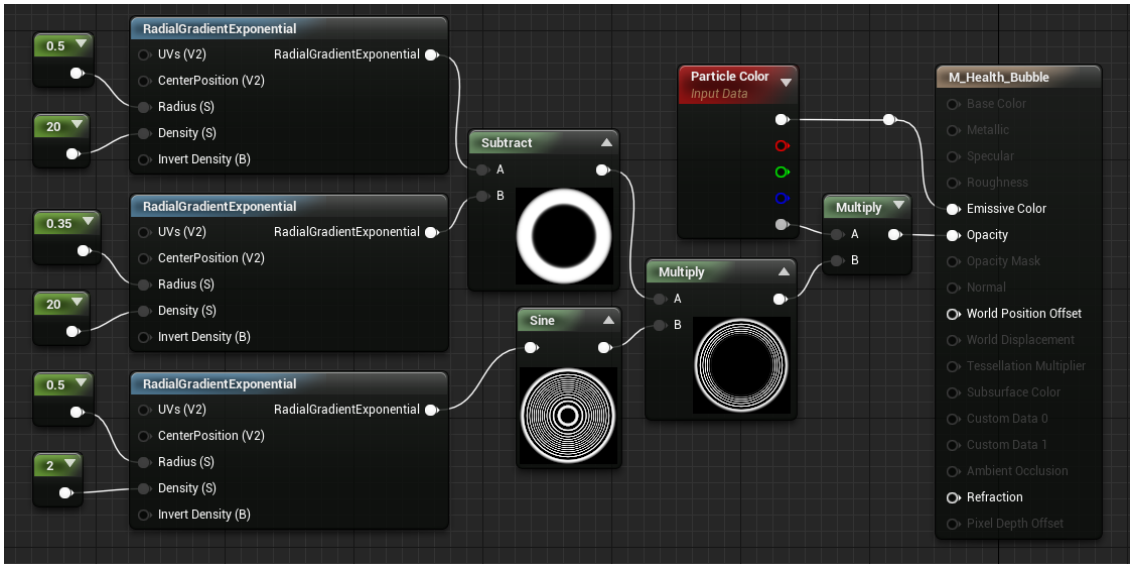
Joonisel (Joonis 1) kujutatakse kiviseina tekstuuri ning sellele vastavat normaalikaarti. Nende abil saab luua Unreal Engine 4 mängumootoris materjali, mis simuleerib kolmemõõtmelist mudelit.

3.3.2 Materjalid

Mänguelementide disainimiseks, saab sõlmedepõhises redaktoris *Material Editor* luua erinevaid varjundajaid (ingl *shader*), mida mängumootoris nimetatakse materjalideks.

Neid saab määrata efekti osakestele ning 2D kui ka 3D mänguobjektidele. Materjalid võivad olla protseduurilised, tekstuuridel põhinevad või kasutada mõlemat viisi korraga. Protseduuriliselt loodud materjalid on mahult väiksemad, kuid kulutavad seejuures rohkem arvutusressurssi. Tekstuuridel põhinevad materjalid ei vaja suurt arvutusvõimsust, kuid on mahult suuremad. Materjalide põhjal saab luua ka dünaamilisi mängus muudetavaid materjalide instantse, et sama materjali omadusi algmaterjalist sõltumatult muuta. Selleks peavad materjali vastavad omadused olema seadistatud parameetritena.

Mängutaseme jaoks loodi udu ja taustakuma materjalid ning nende põhjal erinevate seadistustega materjalide instantsid. Enamus mänguelementide materjalidest on tekstuuripõhised ning mõned neist põhinevad lisanduvalt ka normaalikaartidel. Nii tekstuurid kui ka normaalikaartid on eelnevalt joonistatud ning genereeritud. Suurem osa efekti osakeste materjalidest on protseduurilised, välja arvatud tule efekt, mis vajab efekti loomiseks erinevaid leegi kujude tekstuure.



Joonis 2. Elude korjamise efekti osakese materjali seadistamine mängumootori sõlmedepõhises materjalide loomise redaktoris

Joonisel (Joonis 2) näidatakse elude korjamise efekti jaoks loodud osakese materjali. Materjal on kettakujuline ning see saavutatakse lahutades üksteisest kaks eri suurusega ringi. Järgnevalt, kasutades siinusfunktsiooni, lisatakse kettakujulisele materjalile laineline efekt. Materjali värvus seadistatakse osakeste süsteemi redaktoris.

3.3.3 Elementide kihtide organiseerimine

2D mängudes kasutatakse tihti parallaks efekti, kus mängumaailma esi- ja tagaplaani osad liiguvad vaatleja suhtes erineva kiirusega, tekitades sellega ruumilisuse illusiooni [10]. Mängutasandi loomiseks organiseeriti tasandi elemendid kihiti eraldi kataloogidesse, et kihtide liigutamine ja modifitseerimine oleks lihtsam. Lisaks on võimalik nii iga kihiga eraldi töötades teised kihid peita. Mängumootoris saab mänguelemente grupeerida ka selliselt, et need ei pea ühes kataloogis asuma. Valides grupist ühe elemendi, märgitakse ära kõik gruppi kuuluvad elemendid, mida saab teisendada ka üheks suureks *sprite*'ks.

Elementide kihilisuse kuvamiseks määratakse kõikidel tasandi *sprite*'del kuvamise järjekorra muutuja *Translucency Sort Priority* (TSP) väärtus. Kui elemendi TSP väärtus on väiksem, kuvatakse seda suurema TSP väärtusega elemendi taga [75]. Lõputööks tehtava taseme ülesehitamisel järgitakse järgmisi reegleid:

- kihid organiseeritakse eraldi kataloogidesse;
- kataloogisiseselt grupeeritakse mõned paljuarvulised korduvad elemendid;
- kihtidele määratakse TSP vahemikud;
- mänguelemendid organiseeritakse mitmesse suuremasse kihti.

3.3.4 Kaamera

Töö käigus loodava taseme jaoks on oluline, et kaamera ei oleks seotud ainult juhitava karakteriga, vaid selle liikumist saaks juhtida karakterist sõltumatult. Juhul kui kaamera oleks seotud karakteri klassiga, hävineks karakteri hävimisel ka kaamera, kuna see on karakteri alamkomponent. Dünaamilisema kaamerasüsteemi loomiseks oleks üheks lahenduseks iga fookuspunkti jaoks luua eraldi kaamera, mis erinevates olukordades käivituks. Sel viisil oleks paljud mängus kasutavad kaamerad suurema osa mänguajast tegevuseta. Sellest lähtuvalt kasutatakse loodaval mängutasemel vaid ühte kaamerat ning palju erinevaid fookuspunkte, mida kaamera saab jälgida. Kaamera vahetab fookuspunkte sõltuvalt sellest kui mängija liigub fookuse vahetamise alasse. Igal fookuse vahetamise alal saab mängumootoris määrata, kas kaamera edaspidiseks jälgitavaks on karakter või uus fookuspunkt. Kaamera on seadistatud tegelasele järgnema ajalise viitega.

Mängumootor võimaldab mängus kasutada kahte tüüpi kaamerat: ortograafilise vaatega ja perspektiivvaatega kaamera. Perspektiivvaatega kaamerat kasutatakse 3D mängudes ja ortograafilist kaamerat tavaliselt 2D mängude puhul, projitseerides kõik kolmemõõtmelised objektid kahemõõtmelistena. Lõputöö käigus loodava taseme kuvamiseks kasutatakse perspektiivvaatega kaamerat, kuna see võimaldab parallaks efekti lihtsamalt luua.

3.4 Juhitava karakteri loomine

Mängudes esineb tavaliselt mitmeid erinevaid karaktereid, kelleks võivad olla näiteks humanoidid, loomad või masinad. Mängu karaktereid juhitakse läbi kontrollerite, mille sisendid võivad tulla näiteks klaviatuurilt, mängupuldilt või ka tehisintellektilt. Karakterid saavad ka erinevatel viisidel liikuda ning selle seadistamiseks on neil liikumise komponent. Järgnevalt kirjeldatakse mängija poolt juhitava karakteri loomist.

3.4.1 Kontroller

Karakteri kontrollimiseks on vajalik siduda mängija sisend nagu näiteks klahvivajutus mängutegelase tegevustega. Seda ülesannet täidab kontroller. Mängutaseme loomisel lähtuti tegelase kontrolleri seadistamisel sellest, et karakterite unikaalsed tegevused seotakse klahvidega karakterite klassis ja nende ühised tegevused karakterite kontrolleri klassis. Selliselt toimides saab tegevuste lisades paremini kavandada, millised neist võiksid olla seotud karakteri- ning millised kontrollerisiseselt.

Mängija kontrollitavaid karaktereid on loodavas mängus kaks: mängutegelane ja kontrollitav teleporteerimiskuul. Mängutegelase klassis seotakse klahvidega tema unikaalsed tegevused nagu relva valimine ja kasutamine ning hüppamine. Teleporteerimiskuuli klassis seadistatakse klahv tegevusele, mille eesmärgiks on mängija teleportimine kuuli asukohta. Mõlemale karakterile on omane eri suundades liikumine, seepärast seadistatakse need tegevused klahvidega karakterite kontrolleris.

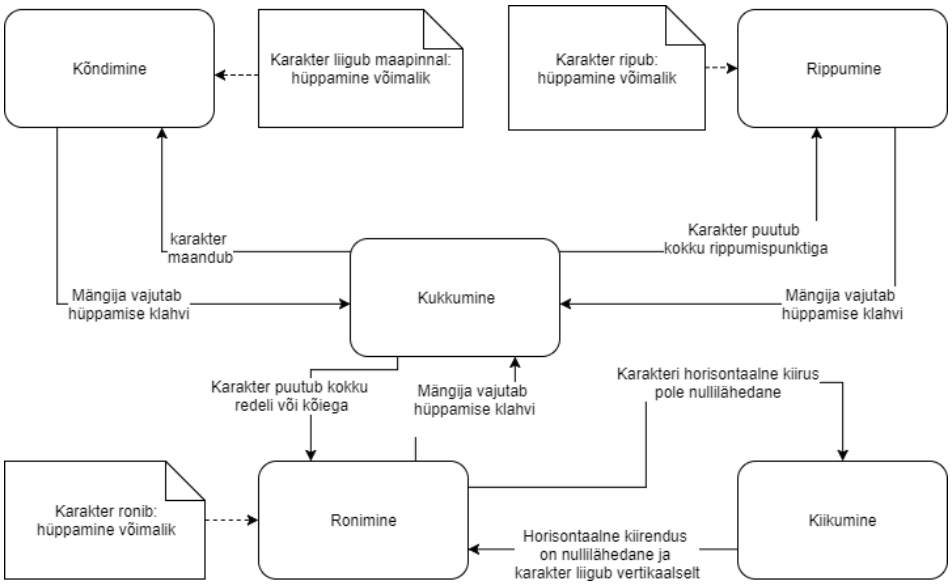
Joonis 3. Karakteri tegevuste sidumine klaviatuuriklahvidega Unreal Engine 4 mängumootoris.

Joonisel (Joonis 3) näidatakse, kuidas seotakse mängumootoris klaviatuuriklahvid konkreetsete tegevuste nimetustega, mis salvestatakse mänguprojekti konfiguratsiooni kausta DefaultInput.ini faili. Seejärel seotakse karakteri tegevused eelnevalt konfigureeritud tegevuste nimetuste ja klahvi vajutuste sündmustega (Lisa 1 – karakteri tegevuste sidumine eelnevalt konfigureeritud tegevuste nimetuste ja klahvi vajutuste sündmustega).

3.4.2 Liikumise komponent

Liikumise komponent seadistab karakterile mängufüüsika põhjal tingimused liikumiseks ning põhilised liikumise olekud nagu kõndimine, lendamine, ujumine, tehisintellekti kontrollitava karakteri navigatsioonipõhine liikumine ning kukkumine, kuhu kuulub ka hüppamine. Mängutegelasele liikumisolekute juurde lisamiseks on liikumise komponendiga seotud ka üks funktsionaalsuseta liikumisolek *Custom*, mille abil saab defineerida veel 255 erinevat viisi karakteri liikumiseks.

Karakterile luuakse lisanduvalt liikumisviisid nagu ronimine, kiikumine ja rippumine. Ronimine ja kiikumine on seotud köie ja redeliga, rippumine aga rippumispunktiga. Ronides piiratakse karakteri liikumine ainult vertikaalteljele, võimaldades liikuda vaid üles ja alla. Kiikudes saab karakter liikuda horisontaalselt koos köie lüliga, millele avaldatakse tema kiiruse kaudu jõudu. Rippumise olek on kolmest liikumisolekust kõige lihtsam, kus karakterile määratakse rippudes eraldi tõeväärtus.



Joonis 4. Mängus kasutavaid liikumisolekud ning karakteri liikumisolekute üleminekud.

Joonisel (Joonis 4) näidatakse mängus kasutatavaid liikumisolekuid ning karakteri tingimusi üleminekuks järgmistesse liikumisolekutesse.

3.5 Vastaste loomine

Vastaste loomisel kasutatakse mängumootoris kahte põhivahendit: andmete tahvel (ingl *blackboard*) ja käitumisreeglite puu (ingl *behavior tree*). Käitumisreeglite puu on reeglite kogum, mis määrab vastase käitumise. Käitumise määramiseks käitumisreeglite puus kasutatakse sisendandmeid nagu näiteks elude hulk või mängija asukoht ning neid hoitakse andmete tahvlil [76].

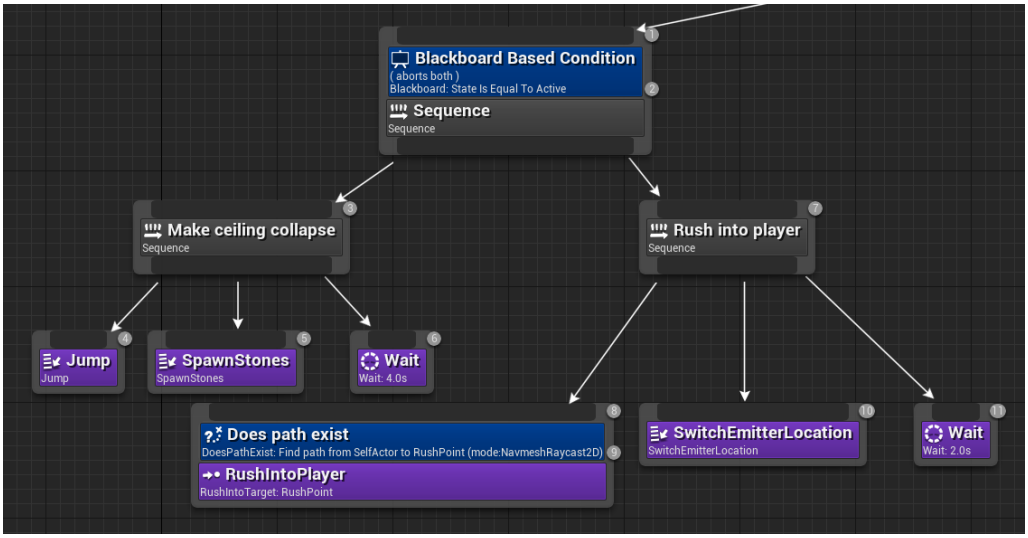
Käitumisreeglite puu loomiseks kasutatakse selektor (ingl *selector*) ja jada (ingl *sequence*) komposiitsõlmi. Need on puu juursõlmed, mis määravad puu läbimise tingimused ning võivad koosneda mitmest alampuust. Selektor juursõlmega puus lõpetatakse töö õnnestunult, kui vähemalt ühe sõlme tegevus õnnestub. Jada juursõlmega puus lõpetatakse töö õnnestunult, kui kõikide sõlmede tegevused õnnestuvad. Nii selektor kui ka jada juursõlmega puudes läbitakse sõlmi vasakult paremale [77].

Selleks, et vastased saaks mängus liikuda, peab platvorm, millel liikumine toimub, asuma objekti *NavMeshBoundsVolume* alas. 2D mängude puhul on vajalik seadistada ka *sprite*'i põrkekujundi paksus (ingl *collision thickness*).

Järgnevas alampeatükis käsitletakse, kuidas on ülesehitatud mängu peavastane. Lisaks peavastasele kasutatakse mängumaailmas ka tulistavaid ning patrullivaid vastaseid.

3.5.1 Peavastane

Lõputöö mängutaseme peavastasel on kolm erinevat käitumist: patrullimine, kivide allaraputamine ja lüüasaamine. Joonisel (Joonis 5) näidatakse vastase käitumisreeglite puu osa, mida läbitakse, kui vastane on aktiveerunud.

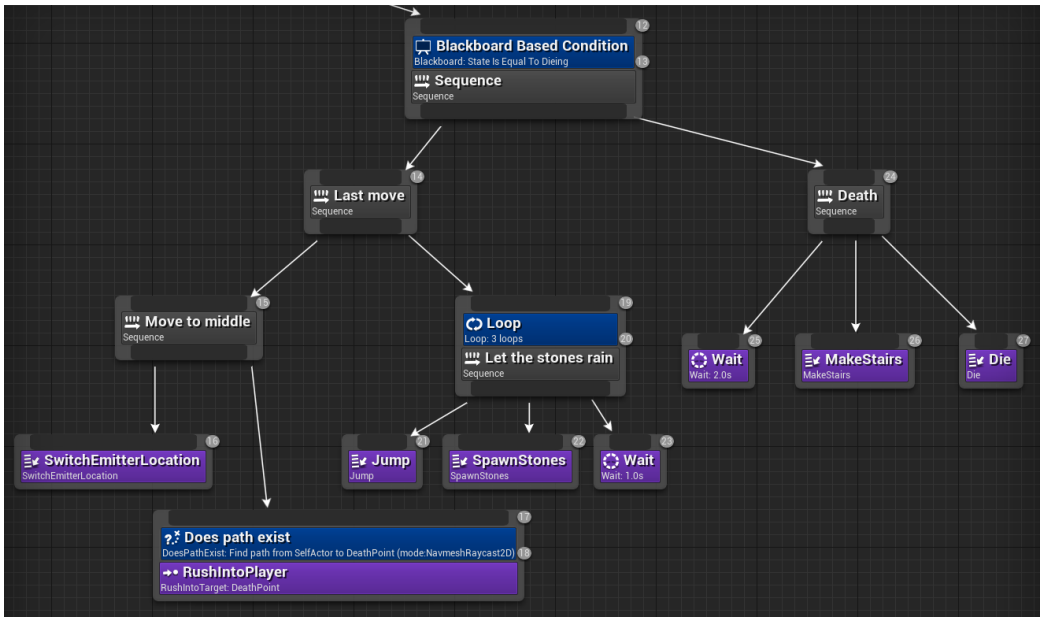


Joonis 5. Peavastase käitumine, kui vastane on aktiivne.

Puus sisaldub kaks tegevust: kivide laest alla raputamine ning mängija suunas liikumine. Selle juursõlmes kontrollitakse, kas vastasel on veel elusid. Kontrollimine toimub ka alamsõlmede töötades. Kui vastasel elusid ei ole, katkestatakse puu läbimine ning alustatakse lüüasaamise alampuu läbimist.

Make Ceiling collapse sõlme töötamisel hüppab vastane õhku ning tema maandudes langevad laest kivid. Maapinnaga kokkupuutumisel hävinevad kõik kivid peale ühe. Allesjäänud kivi on vajalik, et mängija saaks sellelt üle vastase hüpata. Selleks, et mängija jõuaks õigeaegselt kivile, ootab vastane neli sekundit.

Rush into player sõlme töötamisel liigub vastane kiirenedes ettemääratud asukohta, kontrollides eelnevalt, kas teekond sinna eksisteerib. Kui vastane jõuab määratud asukohta, liigutatakse kivide langetamise objekt teisele poole võitlusala ning oodatakse kaks sekundit.



Joonis 6. Peavastase käitumine tema elude puudumisel.

Joonisel (Joonis 6) näidatakse vastase käitumist, kui tal puuduvad elud. Selle puu töötades, ei kaota mängija vastasega kokkupuutel elusid.

Last Move sõlme töötamisel viiakse kive langetava objekti asukoht võitlusala keskele, kuhu liigub ka vastane. Seejärel hüppab vastane tugevamalt ja kiiremini kolm korda vastu maad, raputades kive kogu võitlusala laiuselt.

Death sõlme töötamisel moodustub kukkuvatest kividest trepp, et mängutegelane saaks tasemel edasi liikuda.

3.6 Tasandi läbimist toetavate mänguelementide loomine

Mängija jaoks huvitavamaks ja mitmekesisemaks taseme läbimiseks loodi mitmeid mänguelemente: lülidest koosnev füüsikaline rippredel ja köis, varisevad trepid, staatiline kinnituspunkt, liikuv platvorm, kivide langetamise element ning vedru.

3.6.1 Rippredel ja köis

Need mänguelemendid on loodud sarnasel tööpõhimõttel, koosnedes vastavalt redeli või köie lülidest. Rippredeli ja köie klassis hoitakse andmeid redelil asuva karakteri ning tema poolt hõivatud lüli kohta. Selliselt saab vajadusel redelit kasutada mitu karakterit üheaegselt ning on võimalik kontrollida, kas lüli on mõne karakteri poolt juba hõivatud. Rippredeli ja köie klassis on seadistatud ka lülide genereerimine nii, et sõltuvalt redeli

pikkusest luuakse vajalik arv lülisid. Lüliga kokkupuutumisel viiakse karakter ronimise liikumisolekusse ning lisatakse ta vastava lüli külge. Nii redeli kui ka köie jaoks loodi uus liikumisolek – ronimine. Selleks loodi uus liikumiskomponent, mis määrati vaikimisi karakteri liikumiskomponendiks. Köiel on erinevalt rippredelist lisaks ka kiikumise liikumisolek.

3.6.2 Varisevad trepid ja liikuvad platvormid

Varisevate treppide loomiseks kasutati ajaliini (ingl *Timeline*) komponenti, mis võimaldab kõvera abil ajas muuta objektiga seotud muutujate väärtuseid. Komponendiga seotud kõverad võivad olla erinevad ning seetõttu saab ka objekti muutujate väärtuseid erineval viisil muuta. Varisevate treppide puhul on muudetavaks suuruseks trepi transformatsioon (suurus, asukoht ja kalle). Mängutegelase hüppamisel trepile, registreeritakse kokkupõrge ning käivitatakse ajaliini komponent. Ajaliini komponent lähendab vastavalt kõverale trepi asukohta ja kallet ajas lõppväärtusele.

Liikuvad platvormid kasutavad samuti ajaliini komponenti, kuid võrreldes varisevate treppidega on ajaliini komponendil seadistatud ka funktsioon, mis käivitub kui kõver on läbitud. Pärast kõvera läbimist ootab platvorm teatud aja ning liigub tagasi algpunkti.

3.6.3 Kive langetav mänguelement, staatiline rippumispunkt ja vedru

Kive langetav mänguelement loob juhuslikult teatud ajavahemiku järel kukkuva kivi, mis karakterile langedes tema elusid vähendab. Kivi langetamise viise on kaks: kivide langemine korrapäratult ja kivide langemine korrapäraselt, kus igast kivi loomispunktist kukub ainult üks kivi.

Staatiline rippumispunkt võimaldab karakteril läbida mängutasandit olukordades, kus kõndimine seda ei võimalda. Juhul, kui karakter puutub kokku rippumispunktiga, muudetakse tema liikumisolek rippumiseks. Selle elemendi kasutamise toetamiseks loodi ka rippumise liikumisolek ning täiendati karakteri hüppamise kontrollfunktsiooni selliselt, et rippumispunktilt saaks hüpata.

Vedru eesmärgiks on karakterit ülespoole tõugata. Vedru kokkupuudet karakteriga kontrollitakse vedru peal asuva alaga, et takistada karakteri õhku paiskamist vedru külgedelt.

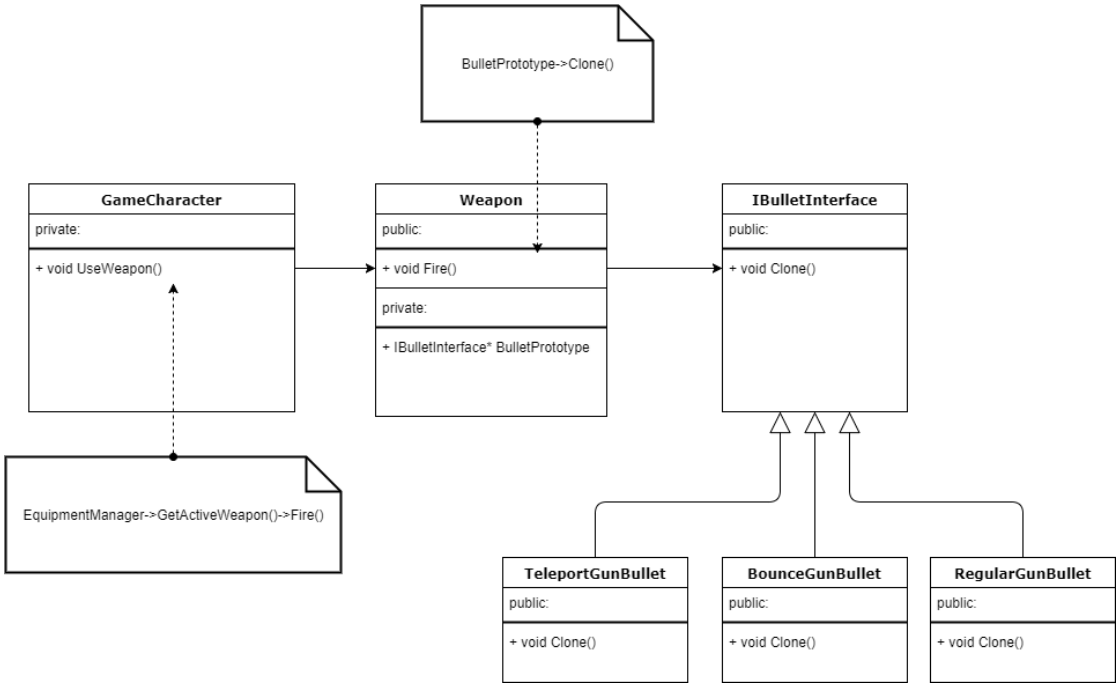
3.7 Relvasüsteemi väljaarendamine

Mängutasemel vastastega interakteerumiseks saab karakter kasutada erinevaid relvi ning lõputöö raames luuakse selleks kolm relva. Relvade haldamiseks ning kergesti lisamiseks kasutatakse *Prototype* disainimustrit. *Prototype* mustrit kasutatakse sageli arvutimängude arenduses, näiteks vastaste genereerimisel [76].

Disainimustri põhiidee seisneb selles, et üks objekt võimaldab näidise ehk prototüübi põhjal teisi sarnaseid objekte luua, näiteks konkreetse kuuli prototüübi põhjal saab relv genereerida just seda tüüpi kuule. *Prototype* mustrit kasutades ei pea arendaja uute relvade lisamisel looma uut relva klassi, tuleb luua vaid uus kuuli prototüübi klass. Kui mängu relvad on erineva funktsionaalsusega, tuleb laiendada relva baasklassi.

Prototype disainimustri implementeerimiseks tuleb luua kuuli baasklass ja sellele lisada abstraktne kloonimise meetod. Iga konkreetne kuuli klass peab seda meetodit implementeerima. Kuulide genereerimiseks on vajalik luua relva klass, mis kasutab kuuli baasklassi laiendavaid konkreetseid kuule. Relva klassis luuakse ka tulistamise funktsioon, mis loob uusi kuule, kasutades selleks prototüübi kloonimise meetodit.

Lõputöös loodava relvasüsteemi jaoks loodi kuulide baasklass *BulletInterface* ning kolm konkreetset alamklassi: *Bullet*, *BouncingBullet*, *TeleportGunBullet*. Kuulide genereerija klassina loodi relva klass *Weapon*, millele määrati *BulletInterface* baasklassi põhjal loodud kuuli prototüüp (Lisa 2 – Mängutegelase relvasüsteem *Prototype* disainimustri põhjal).



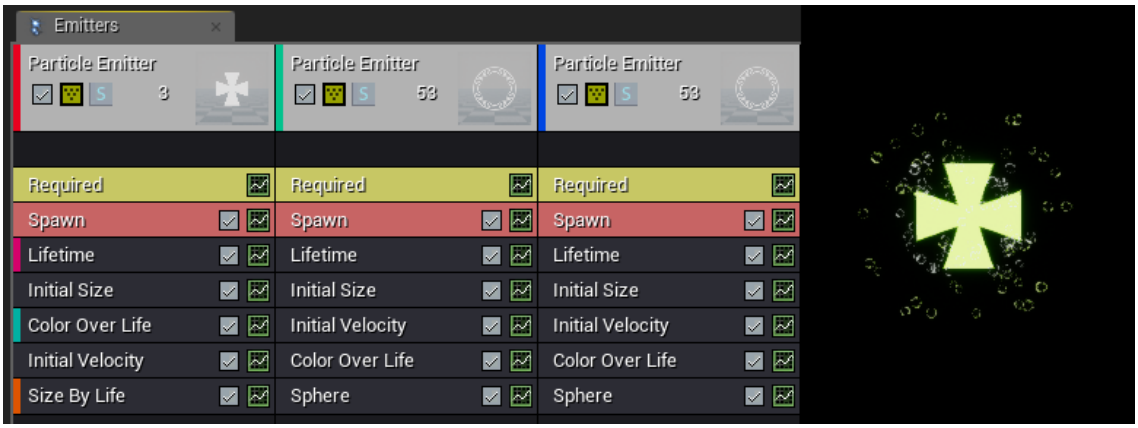
Joonis 7. Mängutegelase relvasüsteem *Prototype* disainimustri põhjal

Joonisel (Joonis 7) näidatakse, milline on *Prototype* disainimustri põhjal loodud relvasüsteem. Järgnevalt kirjeldatakse kuidas mängija poolt antud sisendi põhjal relva kasutatakse:

- mängija sisendi põhjal käivitatakse mängutegelase relva kasutamise meetod;
- relva kasutamise meetodis käivitatakse mängutegelase aktiivse relva tulistamise meetod;
- tulistamise meetodis käivitatakse vastava kuuli kloonimise meetod, mille käigus luuakse uus kuul.

3.8 Visuaal- ja heliefektide tegemine

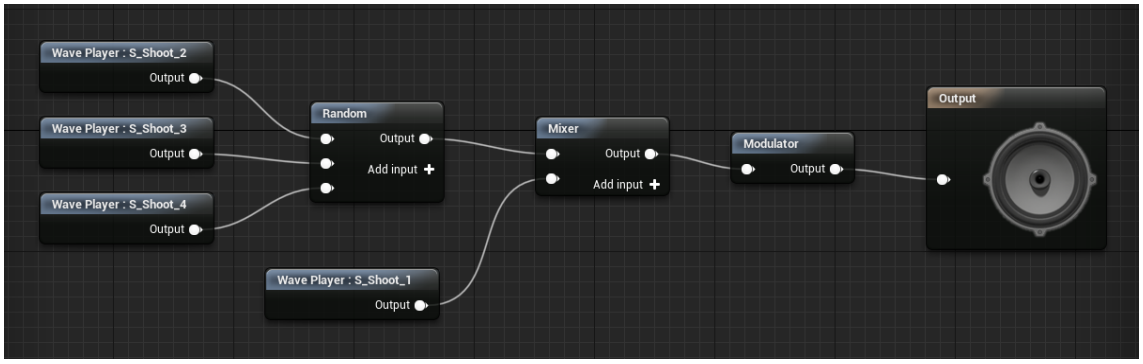
Arvutimängude esteetikas on olulisteks osadeks mängu visuaalne esitusviis ja helikujundus, suurt tähtsust omavad mängus esinevad heli- ja visuaalefektid.



Joonis 8. Elude korjamise efekti seadistus (vasakul) ja efekti lõplik välimus (paremal).

Joonisel (Joonis 8) näidatakse redaktoris seadistatavaid kategooriaid, mida kasutatakse elude korjamise efekti loomisel. Efekti moodustavad kolm osakeste õhkupaiskajat (ingl *Emitter*), mis loovad kindlaid osakesi nagu efekti keskmes olevad ristikujulised osakesed, seda ümbritsevad rohelised ringikujulised osakesed ning ringikujulised helerohelised osakesed. Iga õhkupaiskaja on seotud erinevate moodulitega, mis kirjeldavad osakeste erinevaid omadusi, näiteks osakese algsuurus ja -kiirus, eluaeg, värv või loomisviis (ingl *Spawn*). Ringikujuliste osakeste õhkupaiskajale on lisaks määratud ringikujuline osakeste mõjuala.

Heliefekte seadistatakse Unreal Engine 4 mängumootoris redaktoris *Sound Cue*. Helid loodi lõputöö jaoks kasutades vabavaralist tarkvara Audacity ning Bfxr.



Joonis 9. Tulistamise heli seadistamine UE4 mängumootori *Sound Cue* redaktoris

Joonisel (Joonis 9) näidatakse kuidas mängumootoris seadistatakse tulistamise heliefekt. Heliefekt kasutab nelja varasemalt loodud heliklippi, mis üksteisest vähesel määral erinevad. Relva tulistamisel valitakse neist juhuslikult üks, mida miksitakse teise tulistamise heliklipiga. Seejärel rakendatakse efektile modulaatori funktsiooni, mis muudab igal tulistamisel teatud vahemikus vähehaaval helikõrgust ja helivaljust.

4 Kokkuvõte

Käesolevas töös loodi 2D arvutimängu üks tase. Selle jaoks uuriti arvutimängude ajalugu, anti lühike ülevaade mängulisusega seotud teemadest ning analüüsiti ja võrreldi kolme enamkasutatavat kahemõõtmeliste mängude loomise võimekusega mängumootorit Unreal Engine 4, Unity ja GameMaker. Võrdluse põhjal valiti 2D mängu arendamiseks sobiv mängumootor, lähtudes järgmistest kriteeriumitest: mängumootori litsentsi sobivus, visuaal- ja heliefektide loomise lihtsus ja mängumootori lähtekoodi avatus. Püstitatud kriteeriumite põhjal osutus võrreldavatest mängumootoritest sobivaimaks Unreal Engine 4, mida kasutati 2D arvutimängu ühe taseme loomisel.

Mänguloogika kirjeldamiseks kasutati C++ programmeerimiskeelt. Mängutaseme jaoks vajalikud visuaalefektid ja materjalid loodi mängumootori redaktorites. Mängumaailma elementide kujundamiseks kasutati tarkvara Adobe Photoshop ning helid loodi kasutades vabavaralist tarkvara Audacity ning Bfxr. Komposiithelid loodi ning seadistati mängumootori redaktoris.

Mängutaseme jaoks loodi mänguloogika juhitava peakarakteri, vastaste, tasandi läbimist toetavate mänguelementide, relvasüsteemi ning kaamera seadistamise kohta. Mängija poolt kontrollitavale karakterile loodi kolm lisa liikumisolekut ning lisaks seoti mängukarakteri erinevad tegevused mängija sisendiga. Mängutaseme jaoks loodi kolm arvuti poolt kontrollitavat vastast ning seitse tasandi läbimist toetavat elementi. Mängu peategelase jaoks loodi relvasüsteem koos kolme kasutatava relvaga. Kaamera seadistamise jaoks loodi kaamera poolt kontrollitavad fookuspunktid ning fookuse vahetamise alad.

Töö tulemusena valmis mängutase koos seda toetava loogikaga, mis on alamosaks arvutimängule. Töö käigus esines mitmeid probleeme, mängutaseme arendusprotsess oli töömahukas ning tulemuseni jõudmiseks tuli otsida erinevaid lahendusviise ja palju katsetada.

Tulevikus on lõputöö autoril plaanis luua valmis arvutimäng, laiendades valminud mängutaset ning disainides veel lisanduvalt mitmeid erinevate võimalustega tasemeid.

5 Kasutatud kirjandus

[1] Microsoft, „What Is a Sprite? - MSDN - Microsoft,“ [Võrgumaterjal]. Available: https://msdn.microsoft.com/en-us/library/bb203919.aspx?f=255&MSPPError=-2147217396. [Kasutatud 01 12 2017].

[2] J. Bose, „Creating Isometric Worlds: A Primer for Game Developers,“ Envato Pty Ltd., 27 05 2013. [Võrgumaterjal]. Available: https://gamedevelopment.tutsplus.com/tutorials/creating-isometric-worlds-a-primer-for-game-developers--gamedev-6511. [Kasutatud 01 12 2017].

[3] CaptainKraft, „An Introduction to Creating a Tile Map Engine - Game Development,“ Envato Pty Ltd., 10 09 2013. [Võrgumaterjal]. Available: https://gamedevelopment.tutsplus.com/tutorials/an-introduction-to-creating-a-tile-map-engine--gamedev-10900. [Kasutatud 01 12 2017].

[4] Mozilla and individual contributors., „Tiles and tilemaps overview - Game development | MDN,“ [Võrgumaterjal]. Available: https://developer.mozilla.org/en-US/docs/Games/Techniques/Tilemaps. [Kasutatud 01 12 2017].

[5] Christopho, Solarus, „Solarus quests: Tileset definition file,“ 16 02 2017. [Võrgumaterjal]. Available: http://www.solarus-games.org/doc/latest/quest_tileset_data_file.html. [Kasutatud 01 12 2017].

[6] CodeAndWeb GmbH, „What is a sprite sheet? - The Movie - Introduction - CodeAndWeb,“ [Võrgumaterjal]. Available: https://www.codeandweb.com/what-is-a-sprite-sheet. [Kasutatud 01 12 2017].

[7] GameFromScratch.com, „LibGDX Tutorial 3B: Simple Animation - Game From Scratch,“ 09 12 2013. [Võrgumaterjal]. Available: http://www.gamefromscratch.com/post/2013/12/09/LibGDX-Tutorial-3B-Simple-Animation.aspx. [Kasutatud 01 12 2017].

[8] GameFromScratch.com, „Unreal Engine Tutorial Part Four: Sprite Animation (Flipbooks),“ 29 04 2015. [Võrgumaterjal]. Available: http://www.gamefromscratch.com/post/2015/04/29/Unreal-Engine-Tutorial-Part-Four-Sprite-Animation-(Flipbooks).aspx. [Kasutatud 01 12 2017].

[9] Epic Games, Inc., „Paper 2D Flipbooks | Unreal Engine,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Paper2D/Flipbooks/. [Kasutatud 27 11 2017].

[10] S. Bone, „Parallax Scrolling: A Simple, Effective Way to Add Depth to a 2D Game,“ Envato, 17 07 2014. [Võrgumaterjal]. Available: https://gamedevelopment.tutsplus.com/tutorials/parallax-scrolling-a-simple-effective-way-to-add-depth-to-a-2d-game--cms-21510. [Kasutatud 27 11 2017].

[11] Agile Alliance, „What is Build Automation / Automated Build? | Agile Alliance,“ [Võrgumaterjal]. Available: https://www.agilealliance.org/glossary/automated-build/. [Kasutatud 30 11 2017].

[12] R. Chikhani, „The History Of Gaming: An Evolving Community,“ techcrunch, 31

10 2015. [Võrgumaterjal]. Available: https://techcrunch.com/2015/10/31/the-history-of-gaming-an-evolving-community/. [Kasutatud 26 11 2017].

[13] A. Krotoski, „Chris Crawford: The interview,“ The Guardian, 02 02 2005. [Võrgumaterjal]. Available: https://www.theguardian.com/technology/gamesblog/2005/feb/02/chriscrawford. [Kasutatud 26 11 2017].

[14] C. Crawford, The Art of Computer Game Design. Washington State University, Washington: McGraw-Hill/Osborne Media, 1984.

[15] G. Costikyan, „I Have No Words & I Must Design: Toward a Critical Vocabulary for Games,“ Costikyan, Greg, 2002. [Võrgumaterjal]. Available: http://www.costik.com/nowords2002.pdf. [Kasutatud 28 11 2017].

[16] J. Schell, The Art of Game Design, Burlington: Morgan Kaufmann Publishers, 2008.

[17] M. Sicart, „Defining Game Mechanics,“ *Game Studies,* kd. 8, nr 2, 2008.

[18] J. Brodkin, „How Unity3D Became a Game-Development Beast,“ 03 06 2013. [Võrgumaterjal]. Available: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. [Kasutatud 27 11 2017].

[19] Unity Technologies, „Unity Technologies Releases Whitepaper on Best Practices for Adopting Third-Party Game Development Technology,“ 03 08 2015. [Võrgumaterjal]. Available: https://unity3d.com/company/public-relations/news/unity-technologies-releases-whitepaper-best-practices-adopting-third. [Kasutatud 27 11 2017].

[20] Unity Technologies, „Unite Austin 2017 - 2D World Building in Unity,“ YouTube, LLC, 27 10 2017. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=RkaEh--qUAY. [Kasutatud 09 12 2017].

[21] Unity Technologies, „2D,“ [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Unity2D.html. [Kasutatud 09 12 2017].

[22] A. Golovko, „Unity expands 2D offerings with Anima2D,“ Unity Technologies, 02 12 2016. [Võrgumaterjal]. Available: https://blogs.unity3d.com/2016/12/02/unity-expands-2d-offerings-with-anima2d/. [Kasutatud 09 12 2017].

[23] J. Niman, „Puppet2D,“ [Võrgumaterjal]. Available: http://www.puppet2d.com/. [Kasutatud 27 11 2017].

[24] Esoteric Software, „Spine: Runtimes,“ Esoteric Software, [Võrgumaterjal]. Available: http://esotericsoftware.com/spine-runtimes. [Kasutatud 10 12 2017].

[25] Unity Technologies, „Unity - Manual: Gameplay in 2D,“ Unity Technologies, [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Overview2D.html. [Kasutatud 27 11 2017].

[26] Unity Technologies, „Unity - Manual: Rigidbody 2D,“ [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/class-Rigidbody2D.html. [Kasutatud 27 11 2017].

[27] Unity Technologies, „Unity - Manual: Collider 2D,“ [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Collider2D.html. [Kasutatud 09 12 2017].

[28] Unity Technologies, „Unity - Manual: 2D Joints,“ [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Joints2D.html. [Kasutatud 09 12 2017].

[29] Unity Technologies, „Unity 5 - 2D Physics: Surface Effector 2D and Platform

Effector 2D," YouTube, LLC, 29 04 2015. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=Ojw0pVTGUMA. [Kasutatud 10 12 2017].

[30] Unity Technologies, „Unity - Manual: Effectors 2D," [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Effectors2D.html. [Kasutatud 10 12 2017].

[31] Unity Technologies, „Unity - Manual: Particle Systems," [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/ParticleSystems.html. [Kasutatud 10 12 2017].

[32] Unity Technologies, „2DxFX: 2D Sprite FX - Asset Store," Unity Technologies, [Võrgumaterjal]. Available: https://www.assetstore.unity3d.com/en/#!/content/42566. [Kasutatud 27 11 2017].

[33] Unity Technologies, „2D Dynamic Lights and Shadows - Asset Store," Unity Technologies, [Võrgumaterjal]. Available: https://www.assetstore.unity3d.com/en/#!/content/24083. [Kasutatud 27 11 2017].

[34] Unity Technologies, „Unity - Manual: Audio," [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Audio.html. [Kasutatud 10 12 2017].

[35] B. Steiner, „How the Unreal Engine Became a Real Gaming Powerhouse," Popular Mechanics, 24 06 2013. [Võrgumaterjal]. Available: http://www.popularmechanics.com/culture/gaming/a9178/how-the-unreal-engine-became-a-real-gaming-powerhouse-15625586/. [Kasutatud 10 12 2017].

[36] Epic Games, Inc., „FAQs - Unreal Engine," [Võrgumaterjal]. Available: https://www.unrealengine.com/en-US/faq. [Kasutatud 10 12 2017].

[37] Epic Games, Inc., „Paper 2D | Unreal Engine," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Paper2D/. [Kasutatud 10 12 2017].

[38] Epic Games, Inc., „Paper 2D Tile Sets / Tile Maps | Unreal Engine," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Paper2D/TileMaps/. [Kasutatud 10 12 2017].

[39] M. Noland, „Paper 2D Documentation for 4.3 Preview - Unreal Engine Forums," Epic Games, Inc., 07 01 2015. [Võrgumaterjal]. Available: https://forums.unrealengine.com/development-discussion/content-creation/10210-paper-2d-documentation-for-4-3-preview. [Kasutatud 27 11 2017].

[40] Cardboard Sword Limited, „The Siege and The Sandfox," Cardboard Sword Limited, [Võrgumaterjal]. Available: https://siegeandsandfox.com/. [Kasutatud 10 12 2017].

[41] Kestrel Moon Studios Pte Ltd., „Unreal Engine C++ Runtimes - Creature," 26 11 2017. [Võrgumaterjal]. Available: https://www.kestrelmoon.com/creaturedocs/Game_Engine_Runtimes_And_Integration/Unreal_Engine_C++_Runtimes.html. [Kasutatud 10 12 2017].

[42] Epic Games, Inc., „Physics Simulation - Unreal Engine 4 Documentation," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Physics/index.html. [Kasutatud 10 12 2017].

[43] Epic Games, Inc., „Components - Unreal Engine 4 Documentation," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Components/index.html. [Kasutatud 10 12 2017].

[44] Epic Games, Inc., „Cascade Particle Systems | Unreal Engine,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Rendering/ParticleSystems/. [Kasutatud 10 12 2017].

[45] Epic Games, Inc., „Particle Instance Parameters Tutorial - Epic Wiki,“ [Võrgumaterjal]. Available: https://wiki.unrealengine.com/Particle_Instance_Parameters_Tutorial. [Kasutatud 10 12 2017].

[46] Epic Games, Inc., „Audio and Sound - Unreal Engine 4 Documentation,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/Audio/index.html. [Kasutatud 10 12 2017].

[47] R. Lane, „GameMaker: inside gaming's most user-friendly design tool | PC Gamer,“ PC Gamer, 9 10 2015. [Võrgumaterjal]. Available: http://www.pcgamer.com/gamemaker-the-making-of-gamings-most-user-friendly-design-tool/. [Kasutatud 10 12 2017].

[48] YoYo Games Ltd., „Features | YoYo Games,“ [Võrgumaterjal]. Available: https://www.yoyogames.com/gamemaker/features. [Kasutatud 27 11 2017].

[49] YoYo Games Ltd., „Editors Index,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/2_interface/1_editors/index.html. [Kasutatud 10 12 2017].

[50] M. Alexander, „Using The GameMaker Studio 2 Image Editor – YoYo Games,“ YoYo Games Ltd., 18 06 2017. [Võrgumaterjal]. Available: https://help.yoyogames.com/hc/en-us/articles/216757438-Using-The-GameMaker-Studio-2-Image-Editor. [Kasutatud 10 12 2017].

[51] YoYo Games Ltd., „Soft Body Particles - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/3_scripting/4_gml_reference/physics/soft%20body%20particles/index.html. [Kasutatud 27 11 2017].

[52] YoYo Games Ltd., „Physics Functions - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/3_scripting/4_gml_reference/physics/index.html. [Kasutatud 10 12 2017].

[53] YoYo Games Ltd., „Particles - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/3_scripting/4_gml_reference/drawing/particles/index.html. [Kasutatud 10 12 2017].

[54] YoYo Games Ltd., „Extras - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/2_interface/2_extras/index.html. [Kasutatud 10 12 2017].

[55] YoYo Games Ltd., „Audio - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/3_scripting/4_gml_reference/audio/index.html. [Kasutatud 10 12 2017].

[56] Epic Games, Inc., „EULA - Unreal Engine,“ Epic Games, Inc., [Võrgumaterjal]. Available: https://www.unrealengine.com/en-US/eula. [Kasutatud 18 12 2017].

[57] Epic Games, Inc., „Programming Guide - Unreal Engine 4 Documentation,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Programming/. [Kasutatud 01 12 2017].

[58] Unity Technologies, „Unity - Unity Personal,“ [Võrgumaterjal]. Available: https://store.unity.com/products/unity-personal. [Kasutatud 27 11 2017].

[59] Unity Technologies, „Unity - Manual: Creating and Using Scripts,“ [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [Kasutatud 27 11 2017].

[60] Unity Technologies, „Unity - Unity Plus,“ [Võrgumaterjal]. Available: https://store.unity.com/products/unity-plus?currency=EUR. [Kasutatud 27 11 2017].

[61] Unity Technologies, „Unity - Unity Pro,“ [Võrgumaterjal]. Available: https://store.unity.com/products/unity-pro?currency=EUR. [Kasutatud 27 11 2017].

[62] YoYo Games Ltd., „Get GameMaker | YoYo Games,“ [Võrgumaterjal]. Available: https://www.yoyogames.com/get. [Kasutatud 27 11 2017].

[63] M. Alexander, „GameMaker Studio 2 Trial - Limitations – YoYo Games,“ YoYo Games Ltd., 10 11 2017. [Võrgumaterjal]. Available: https://help.yoyogames.com/hc/en-us/articles/230407528-GameMaker-Studio-2-Trial-Limitations. [Kasutatud 27 11 2017].

[64] M. Alexander, „GameMaker Studio 2 FAQ – YoYo Games,“ 17 11 2017. [Võrgumaterjal]. Available: https://help.yoyogames.com/hc/en-us/articles/230330328-GameMaker-Studio-2-FAQ. [Kasutatud 27 11 2017].

[65] YoYo Games Ltd., „YoYo Games Platforms Publisher Agreement,“ [Võrgumaterjal]. Available: https://www.yoyogames.com/legal/pub-eula. [Kasutatud 01 12 2017].

[66] YoYo Games Ltd., „Scripting - GameMaker Studio 2 - YoYo Games,“ [Võrgumaterjal]. Available: https://docs2.yoyogames.com/source/_build/3_scripting/index.html. [Kasutatud 27 11 2017].

[67] YoYo Games Ltd., „YoYo Games Platforms Publisher Agreement,“ [Võrgumaterjal]. Available: https://www.yoyogames.com/legal/publishers. [Kasutatud 27 11 2017].

[68] Epic Games, Inc., „AGameMode - Unreal Engine 4 Documentation,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/GameFramework/AGameMode/. [Kasutatud 29 11 2017].

[69] Epic Games, Inc., „Gameplay Modules | Unreal Engine - Unreal Engine 4 Documentation,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Programming/Modules/Gameplay/index.html. [Kasutatud 30 11 2017].

[70] Epic Games, Inc., „Unreal Build System - Unreal Engine 4 Documentation,“ [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Programming/UnrealBuildSystem/. [Kasutatud 30 11 2017].

[71] Epic Games, Inc., „Texture Import Guide | Unreal Engine,“ [Võrgumaterjal].

Available: https://docs.unrealengine.com/latest/INT/Engine/Content/Types/Textures/Importing/. [Kasutatud 30 11 2017].

[72] Pluralsight Creative, „CG101: What is a Normal Map?," YouTube, LLC, 20 03 2014. [Võrgumaterjal]. Available: https://www.youtube.com/watch?v=yHzIx41eiD4. [Kasutatud 28 11 2017].

[73] Epic Games, Inc., „TranslucencySortPriority | Unreal Engine," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/API/Runtime/Engine/Components/UPrimitiveComponent/TranslucencySortPriority/. [Kasutatud 01 12 2017].

[74] Epic Games, Inc., „How Unreal Engine 4 Behavior Trees Differ | Unreal Engine," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/HowUE4BehaviorTreesDiffer/index.html. [Kasutatud 06 12 2017].

[75] Epic Games, Inc., „Behavior Trees Nodes Reference | Unreal Engine," [Võrgumaterjal]. Available: https://docs.unrealengine.com/latest/INT/Engine/AI/BehaviorTrees/NodeReference/Composites/index.html. [Kasutatud 06 12 2017].

[76] R. Nystrom, „Game Programming Patterns," Genever Benning, 2014.

[77] Epic Games, Inc., „UE-38054 - Unreal Engine Issues," [Võrgumaterjal]. Available: https://issues.unrealengine.com/issue/UE-38054. [Kasutatud 07 12 2017].

Lisa 1 – karakteri tegevuste sidumine eelnevalt konfigureeritud tegevuste nimetuste ja klahvi vajutuste sündmustega

```
/* Both characters have those actions available. */
void AGameCharacterController::SetupInputComponent()
{
	Super::SetupInputComponent();
	if (InputComponent)
	{
		/* Axis bindings. */
		InputComponent->BindAxis("Move right", this,
		&AGameCharacterController::MoveRight);
		InputComponent->BindAxis("Move left", this,
		&AGameCharacterController::MoveLeft);
		InputComponent->BindAxis("Move up", this,
		&AGameCharacterController::MoveUp);
		InputComponent->BindAxis("Move down", this,
		&AGameCharacterController::MoveDown);
	}
}
void AGameCharacter::SetupPlayerInputComponent(UInputComponent *
PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);
	PlayerInputComponent->BindAction("Jump", IE_Pressed, this,
	&ACharacter::Jump);
	PlayerInputComponent->BindAction("Jump", IE_Released, this,
	&ACharacter::StopJumping);
	PlayerInputComponent->BindAction("Use weapon", IE_Pressed, this,
	&AGameCharacter::UseWeapon);
	PlayerInputComponent->BindAction("Basic Gun", IE_Pressed, this,
	&AGameCharacter::GetBasicGun);
	PlayerInputComponent->BindAction("Teleport Gun", IE_Pressed, this,
	&AGameCharacter::GetTeleportGun);
	PlayerInputComponent->BindAction("Bounce Gun", IE_Pressed, this,
	&AGameCharacter::GetBounceGun);
}
void ATeleportGunBullet::SetupPlayerInputComponent(UInputComponent *
PlayerInputComponent)
{
	Super::SetupPlayerInputComponent(PlayerInputComponent);
	InputComponent->BindAction("Teleport", IE_Pressed, this,
	&ATeleportGunBullet::TeleportPlayer);
}
```

Lisa 2 – Mängutegelase relvasüsteem *Prototype* disainimustri põhjal

```
/* SETUP PHASE */

/* Bullet prototypes are created. */
IBulletInterface* BasicBullet = NewObject<ABullet>(this);
IBulletInterface* TeleportGunBullet = NewObject<ATeleportGunBullet>(this);
IBulletInterface* BounceGunBullet = NewObject<ABouncingBullet>(this);

/* Main Weapon (basic bullets emitter) is created for the character. */
BasicWeapon = Cast<ABasicWeapon>(GetWorld()-
>SpawnActor(ABasicWeapon::StaticClass()));
BasicWeapon->Initialize(RegularWeaponSprite, BasicBullet, Fire_Sound,
EWeaponType::BasicGun);
BasicWeapon->AttachToComponent(GetSprite(),
FAttachmentTransformRules::KeepRelativeTransform, FName("Weapon"));
BasicWeapon->SetActorRelativeRotation(FRotator(0.f, 0.f, 0.f));
/* Two remaining weapons are created... */


/* SHOOTING */
/* Player input activates UseWeapon function. */
void AGameCharacter::UseWeapon()
{
	EquipmentManager->ActiveWeapon->fire();
}

/* Active weapon’s fire function calls prototype clone function. */
void ABasicWeapon::fire()
{
	this->BulletPrototype->Clone();
}

/* Prototype clone function creates bullet instance. */
void ABullet::Clone()
{
	/* Spawn logic... */
}
```

Lisa 3 – Arendusprotsessi suurimad küsimused, takistused ning probleemid

Kas mängus kasutatav relv luua komponendipõhiselt? – kui relv luua komponendipõhiselt, ei saa seda mängumaailmas asetada. Relva asetamiseks maailma, peab see olema seotud mõne Unreal Engine 4 (UE4) *AActor* tüüpi objektiga. Relva *AActor* tüübina luues, saab seda asetada maailma ning lisada ka mängija külge. Lisaks tuleb relva ja karakterit vastavalt seadistada, et nende omavahelisi kokkupõrkeid ignoreeritaks. Vastasel juhul on karakter relva küljes kinni.

Relva sidumiseks konkreetse kuuli prototüübiga ei saa kasutada geneerilisi tüüpe ega parameetritega konstruktoreid – kui relv põhineb *AActor* tüübil, on selle kõige kõrgema taseme baasklassiks *UObject*, mis ei võimalda selle põhjal geneerilisi klasse luua. Lisaks ei saa UE4 mängumootoris kasutada parameetritega konstruktoreid, kuna *AActor* tüüpi objekte luuakse spetsiaalsete funktsioonide abil ning need ei luba parameetrite kasutamist. Selleks, et kuuli prototüüpi relvaga siduda, käivitatakse pärast konstruktori kasutamist funktsiooni, et initsialiseerida kuuli prototüüp. Alternatiiviks on kasutada funktsiooni *UWorld::SpawnActorDeferred*, mis ei käivitada objekti luues selle konstruktorit, et objektile saaks eelnevalt vajalikud väärtused määrata. Seejärel tuleb objekti manuaalseks konstrueerimiseks kasutada funktsiooni *UGameplayStatics:: FinishSpawningActor*.

Ülekirjutatud funktsioonide ülemklassi implementatsiooni tuleb alati käivitada – ülemklasside funktsioonide ülekirjutamisel, tuleb alati käivitada ka selle funktsiooni ülemklassi implementatsiooni, näiteks *AActor::BeginPlay* funktsiooni üle kirjutades, tuleb selle kehandis käivitada funktsiooni *Super::BeginPlay()*. Lõputöö arendusprotsessis kulus selle vea tõttu mitmel korral asjatult palju aega – ühel korral ei töötanud relvasüsteem, teisel korral ei töötanud mängu tehisintellekti loogika. Vea tuvastamiseks ei aidanud ka erinevate loogikaosade välistamine, kuna oluline koodirida oli jäänud kirjutamata.

Kas kaamera peaks olema karakteri alamkomponent või eraldiseisev objekt? – kui mängu karakter mängumootori poolt hävitatakse, hävineks koos karakteriga ka kaamera alamkomponent. Kaamera eraldiseisva objektina aga koos karakteriga ei hävineks, vaid sellele oleks võimalik tegelase hävides seadistada vastav käitumine.

UE4 kasutajaliidese töö kadumine seoses mängumootoris esineva veaga [78] – kui luua UE4's kasutajaliidest, on seda mugavam teha mängumootoris, kasutades selleks *Blueprint* visuaalse skriptimise süsteemi. Sel juhul on kasutajaliideses muudatuste tegemine kiirem ja need on koheselt näha. Lõputöö käigus loodi need samuti sel viisil ning kasutajaliides loodi hierarhiliselt. Peamenüüle lisati seadete alammenüü ning seadete menüüle klahvide seadistamise ja graafika seadete alammenüüd. Kõik alammenüüd sisaldasid omakorda erinevaid vaate jaoks vajalikke komponente. Selliselt kasutajaliidest üles ehitades oli lihtsam seadistada selle välimust ning loogikat. Kasutajaliidest sel viisil UE4 mängumootoris siiski teha ei saa, sest selliselt toimides hiljem mängumootori redaktori avades, ei avane enam ükski selliselt hierarhiliselt loodud menüü.

Objektile komponentide loomisel kaovad need ära – selline probleem esines redelile lülisid genereerides. Objekti jaoks komponentide loomisel tuleb need alati ka funktsiooni *USceneComponent::SetupAttachment* kasutades objekti vastava komponendiga siduda. Vastasel juhul seatakse komponentide asukohaks vaikimisi nullpunkt, mis suurte mängutasemete puhul võib asetseda tasandiga töötamise alast kaugel.

Füüsikalise lülidest koosneva redeli loomise probleemid – füüsika simuleerimiseks peab *sprite*'l olema põrkekujund. Füüsika simuleerides määratakse objekti mõne komponendiga seotud alamkomponendi asukohaks nullpunkt, kuna kõik komponendid peavad üksteisest sõltumatud olema – need saavad olla üksteisega seotud vaid spetsiaalse *UPhysicsConstraintComponent* komponendi abil. Analoogiana peavad nii laud kui ka laual olev õun olema üksteisest sõltumatud, vastasel juhul liiguks õun nagu liimitult koos lauaga kaasa ega kukuks selle pealt maha.

Lisa 4 – Arvutimängu taseme demonstratsioon

Veebiaadress: https://www.youtube.com/watch?v=TIbyAQpJyCg&feature=youtu.be