TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Marek Pihel 220883IAIB

Stealth põhise videomängu arendus koos tehisintellekti loomisega kasutades Unity mängumootorit

Bakalaureusetöö

Juhendaja: Inna Švartsman
Magistrikraad

Tallinn 2022

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: Marek Pihel

22.02.2022

Annotatsioon

Töö eesmärk, on anda ülevaade video mängu arendusega seotud faktorites ning arendada stealth žanri põhine mäng. Töö alguses, räägitakse ajaloost ja sellest, kus ning kuidas on arenenud stealth mängud. Seejärel on käsitletakse erinevaid faktoreid, mis teevad mängust stealth mängu, ning mis on probleemid, mis esinevad seoses stealth video mängu arendusega. Teemat on lahatud nii tarkvaralise poole pealt kui ka tehnilisema ja mängumehaanilise poole pealt. Nii on püstitatud probleemid seoses mängija liikumise, tehisintellekti kui ka sellega kuidas mängu klassifitseerida stealth kategoorias. Seejärel on räägitud potentsiaalsetest lahendustest eelnevalt püstitatud probleemidele. Peale mida leiab töö nõuded, mis selle projekti raames piiritlesid ja seadsid eesmärgid mängu loomisele. Peale seda leiab ülevaatlikku osa arendusest ning Unity ehitusest, koos projekti raames loodud funktsionaalsuse kirjeldusega. Lõpuks käsitleme mängides testimise protsessi (*inkl. playtesting)* ja selle mõju arendusele ja projektile.

Projekti repositoorium kloonituna giti on leitav aadressilt: https://github.com/marekpihel/Marek_Pihel_bakalaureuset66

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 37 leheküljel, 9 peatükki, 8 joonist, 1 tabelit.

Abstract

Creating stealth genre video game and artificial intelligence using Unity game engine

This thesis covers the video game creation process in the stealth genre, with the addition of artificial intelligence creation for it. The thesis starts with a brief coverage and introduction to the history of stealth genre, along with the root concepts that define it. In addition, the historical section provides overview, of the evolution of those concepts. Following that is a summary, of what makes video game, a stealth game and the problems that need to be addressed. Problems like player movement, artificial intelligence and factors that go into classifying the game, into the stealth genre subgenres. Analysis chapter covers the potential fixes to the problems, that were proposed. Along with some solutions, for the software side of the game development process. After solutions, we delve into why Unity is used for this project, as the game engine. Software for development and the language usage is covered thereafter. When all that is figured out, we outline the requirements for our game, that are derived from the solutions section. Moving forward, we analyze these requirements and how we accomplish them during our project. Which is followed by section about developing a game, with intention to demonstrate some of the solutions that were outlined in the solutions section. In addition, this section also covers the setup of the project, in the game engine and the version control system. Subsequently we introduce the various game engine possibilities, that were used in the project, along with what they were used for. When all the required important parts are introduced, an overview is done about the various parts of the game development, that was focused on for this project. To round out the thesis, playtesting session draws conclusions to usefulness and methodology used while also outlining the purpose for our project along with the analysis into if the requirements were filled.

Project repository cloned into git can be found at: https://github.com/marekpihel/Marek_Pihel_bakalaureuset66

The thesis is in Estonian and contains 37 pages of text, 9 chapters, 8 figures, 1 table.

Sisukord

Lühendite ja mõistete sõnastik

MGS	Metal Gear Solid
MGS2	Metal Gear Solid 2: Sons Of Liberty
PMC	PlayerMovementController skript
UI	User Interface / Kasutajaliides
GameObject	Mänguobjekt
Playtesting	Mängides testimine

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Lõputööks oli soov kirjutada videomängu arendusest. See tulenes faktist, et olen ise olnud seotud videomängu tööstusega väga pikalt ning sellest, et kui lõputööks võtta projekt, mis laskub enda huviorbiiti, siis on motivatsiooni töö kallal rohkem vaeva näha ja seega parem tulem saavutada. Videomängu arendamisel on rida probleeme ja küsimusi, mida on vaja lahendada, seega valisin täielikule hiilimisele keskenduva stealth žanri mängu ning kasutasin seda, et anda ülevaade mängu arenduse protsessist. Töö eesmärk on anda lugejale terviklik komplekt teadmisi, mida kasutada video mängu arendamiseks ning täpsemalt anda ülevaade stealth žanrile selleks, et tööd kasutades oleks võimalik, valmistada otsast lõpuni stealth mäng. Töö raames käsitletakse küsimusi nagu:

1. Kuidas salvestada mängu arendusel faile, mis ei ole ainult koodi failid, vaid ka mudelid ning muud mänguga seonduvad failid?

2. Kuidas luua mäng, mis klassifitseerub stealth žanri alla?

3. Mis on mängumootor ning mis on Unity eelis teiste mängumootorite osas?

4. Mida peaks arvestama ja mis lahendused eksisteerivad erinevate stealth mängude probleemidele?

5. Kuidas testida oma mängu?

Projekti ülesehitus on kronoloogilises järjekorras, arenduse protsessi läbiviimise vaatest. Kus alustame minevikust ning analüüsist, seejärel püstitame probleemid, mis eksisteerivad ja millega peaks arvestama, peale mida leiame neile lahenduse. Seejärel püstitame nõuded ning valmistasime projekti valmis, et näidata kuidas rakendada kirjeldatud lahendusi. Samas on kirjeldatud ka mängu mootori enda osad, vastavalt kasutatud tähtsamale funktsionaalsusele. Lõpetuseks käsiteleme mängu testimist ning meetodit, mida kasutatakse väga laialdaselt mängude arenduses, milleks on mängides testimine.

2 Stealth žanr

Stealth žanr tõlkes nähtamatus, hiilimise, vargsi või märkamatuks jäämise žanri mängud, on mängud, mis kuuluvad märuli mängude alamkategooriasse. Stealth žanri puhul, on mängija peamiseks ülesandeks, kavaldada üle või kõrvaldada tema vastane, kasutades talle saadaval olevat ressurssi ja hiilimist. Stealth mänge, saab omakorda jagada üldistatult kahte kategooriasse. Hiilimisele ja märkamatuks jäämisele keskenduvad mängud ja võitlusele orienteeritud mängude vahel. Esimese puhul, on tegu mänguga, kus mängijale jäetakse mulje kindlalt kaitstud alast, millest mängija saab läbi hiilida, kasutades talle antud ressurssi ja oskusi, ekspluateerimaks auke turvalisuses. Sellisel juhul, on läbi kukkumise eelduseks, mängija avastamine. Teise puhul on tegu mänguga, kus mängijal on võimalus ka vastupanu limiteeritud kujul osutada. Sellist stiili mängudes, on läbikukkumise kriteeriumiks, mängija elude jõudmine nulli ehk mängija suremine. [1]

2.1 Stealth mängude ajalugu

Ajaloos esimeseks videomänguks, mis kasutas lihtsal kujul stealth mehaanikat, loetakse mängu nimega 005. See mäng loodi 1981. aastal mängukonsoolile SEGA. Mängu eesmärgiks, oli salajaste dokumentide portfelli toimetamine lähedal olevasse helikopterisse. Mängu mehaanikatest on paljud ka kasutusel tänapäeval. Kasutades vastase valgusvihku, et anda teada nende vaateväljast ning kasutades pimedaid alasid ja kaste, et ennast vastase eest peita. [2] [3]

Järgnes sellele samal aastal loodud mäng Castle Wolfenstein. Mängu missiooniks on mängijal ülesse leida natside salajased sõjaplaanid ja põgeneda lossist. Algselt oli võimalik läbida mäng ilma kedagi kõrvaldamata, kõrvaldamise vajadus tekkis alates 3D mängudest. Kus oli vaja mängu läbimiseks ka kõrvaldada vastaseid. [4]

Rääkides 3D mängudest, siis esimene 3D mäng mis tuli arvutile, oli Thief: The Dark Project mis ilmus 1998. aastal [5]. Thief oli mäng, mis esimest korda ajaloos, kasutas ära valgust ja varje, et mängija saaks ennast peita. Lisaks eelnevale, oli see ka esimene mäng, mis kasutas ära heli millel oli mänguline tähtsus. Kasutades ära samme ja pealtkuulamist,

et anda mängule mängumehaanika poolest sügavust. Vaatamata sellele, et mäng sisaldas minimaalselt kaklemist, oli mäng väga lõbus ja samal ajal ka väljakutseid pakkuv. [3] [4]

Samal aastal kui Thief: The Dark Project, tuli välja ka MGS (Metal Gear Solid). MGS ei olnud aga esimene mäng Metal Gear seerias, sellele eelnes 1987. aastal välja tulnud Metal Gear. Mängus on mängija erioperatsioonide liige, kelle ülesandeks hävitada kahel jalal kõndiv tank, mis on võimeline laskma välja tuumarakette. Kui mängija jääb vahele turvamehele, siis vastased lähevad valvsasse olekusse (*inkl. alert state*). Mängijal on võimalik olekut muuta, minnes ekraanist välja. Kui aga mängija jääb kaamerale vahele või kasutab summutamata relva, siis vastased tulevad suuremate jõududega mängijat kõrvaldama, mis juhul mängija peab põgenema lifti või kõik vastased elimineerima. MGS kasutas sellele eelneva mängu mängumehaanikaid. Sarnaselt Thiefile näitas ka MGS, et selleks, et mängijal lõbus oleks, ei pea ilmtingimata ringi jooksma ja kõrvaldama kõike mis talle ette jääb. Kuigi nii Thief kui ka MGS mõlemad sisaldasid suure koguse märkamatuks jäämise mehaanikaid, loetakse Stealth žanri populariseerivaks MGS mängu. Tähtis selleks oli mängu narratiiv, mida loetakse senimaani parimaks mängude seas. [3]

Metal Gear seerias on iga järgnev mäng toonud sisse uusi mängumehaanikaid, mida kasutada ja olnud väga innovaatiline. MGS'ile järgnev mäng, Metal Gear seerias, oli MGS2(Metal Gear Solid 2: Sons Of Liberty), mis tuli välja 2001. aastal. Mängu teeb eriliseks paar fakti. Esiteks on ta kõige müüdum stealth žanri mäng ajaloos, seda 23. Märtsi seisuga 2017 aastal, kus oli tema müüginumbriteks 6.05 miljonit müüdud eksemplari [6]. Teiseks mis teeb MGS2 eriliseks on fakt, et ta oli esimene mäng mis kasutas kollektiivselt tehisintellekti. [7]

Liikude edasi erinevate mängudega, siis Hitmani seerias on ka väga tähtsal kohal igasugune märkamatuks jäämine. Hitmani mängu ülesandeks, on antud mängijale ülesanne kõrvaldada mingi kindel sihtmärk, väga suures mängumaailmas. Hitmani seerias märkamatuks jäämisel, on kõige tähtsam mängija oskus kõrvaldada isikuid vaikselt ning kasutada erinevaid kostüüme selleks, et üle kavaldada vastaseid ja ringi liikuda. [3]

Ubisofti toodetud Splinter Cell seerias, on mängumehaanika sarnasem eelnevalt mainitud Thiefi ja MGS mängudega. Spliter Cell mängudes julgustatakse kõrvaldama vastaseid

vaikselt. Splinter Cell kasutab oma mängus märkamatuks jäämisel sarnast süsteemi Thiefile, kus valgus mängib suurt rolli. Mängijal on infrapuna ja termo prillid, mida ta saab kasutada ära pimedas navigeerimiseks. Mängijal on ka valgusmeeter, mis näitab kui hästi vastased mängijat näevad, seega julgustatakse mängijat liikuma pimedas. [3]

Assassin's Creed seeria on üle aastate pakkunud eri kujul ja raskusastmel märkamatuse mängumehaanikat. Kõige viimane sellisele mängustiilile keskenduv mäng selles seerias, oli Assassin's Creed Unity, mille puhul erinevalt hilisematele mängudele, on soovituslik ja ka lõbus mängida hiilides. Assassin's Creed Unity's on hiilimisele ja märkamatuks jäämise eeliseks see, et mängija ei pea kaklema ülekaaluka vastase vastu, kui ta suudab jääda märkamatuks, samas aga on jäetud võimalus märuli põhiselt mängu läbida. Assassin's Creed Unity puhul, kasutatakse eri viise mängija varjamiseks. Vargsi ringi liikumise soodustamiseks, on loodud mehaanika kus liikumine on tehtud võimalikult sujuvaks, aitamaks kaasa parkuurimisele, mida saab kasutada mängumaailmas ringi liikumiseks, kaasaarvatud katuseid mõõda. Selle eesmärk on varjata ennast potentsiaalse vastase eest. Kui aga on aeg tulla katuselt alla, rakendatakse hiilimise saavutamiseks sotsiaalse märkamatuse süsteemi, kus mängija saab sulanduda rahvamassiga ühte, mis aitab osade missioonide puhul eesmärki täita. Samuti on maa peal võimalik mängijal ennast peita erinevate objektide sisse, nagu näiteks tünnid, heinahunnikud või kapid. Samuti on antud vastastele avastamismeeter ja käitumismallid, mis annab mängijale aimu sellest, mida nad plaanivad või mis on nende järgmine käik. [8]

2.2 Mis teeb mängust stealth mängu ning nende probleemid

Stealth põhise mängu loomisel, on mängu eesmärk luua mängijale kogemus, mille põhiliseks fookuseks on vähemal või rohkem määral mängija suutlikkus jõuda punktist A punkti B, kasutades selleks ära talle kättesaadavaid ressursse ja oskusi. Sellega seoses tuleb disainida palju mängumehaanikaga seotud süsteeme ja võtta mängu disainimisel vastu otsuseid mida, kui täpselt ja keeruliselt peaks looma. Põhilised tuumikkomponendid stealth põhise mängu loomisel on, liikumine, katte või peidukohad, turvamehed või turvameetmed, alarmi staatus, algsed sensoorsed süsteemid ja hästi disainitud mängu tasemed. Kõigi eelneva eesmärk, on luua illusioon vastavalt peatükis 2 püstitatud mõttele „kus mängijale jäetakse mulje kindlalt kaitstud alast, millest mängija saab läbi hiilida, kasutades talle antud ressurssi ja oskusi, ekspluateerimaks auke turvalisuses." (Pt. 2). [1]

Selle peatüki sees käsitleme probleeme, mis saavad tekkida tuumikkomponentide loomisel, stealth põhise mängu jaoks.

Paljudes stealth mängudes on käsitletud liikumise süsteemi suhteliselt sarnaselt. Mängijal on võimalus liikuda eri kiirustel ning seoses sellega, on mängija poolt genereeritud heli, kas valjem või vaiksem, mis omakorda suurendab või vähendab tehisintellekti reageerimis kaugust. [1] Siinkohal saab probleemiks süsteemi toimimine ning kuidas luua liikumine, mis sobib hiilimiseks, kuid on ka paindlik kui hiilida pole vaja. Liikumisega seoses on vaja otsustada, kuidas süsteemi siduda mängumaailmaga ja tehisintellektiga ning sellega tuleb arvestada, liikumise mehaanika disainimisel.

Katte ja peidukohad on süsteem, millega mängijal on võimalus jälgida ümber toimuvat ja hoiduda vastase pilgu alla sattumast [1]. Seega kui mängijale on disainitud peitmise koht mängus, kus valgus annab tema asukoha ära, siis ei ole mõtet seda ala valgustada, sest mängija ei varju sinna, isegi kui disainerina tead, et see on turvaline koht. See tuleb faktis, et mängija jaoks, ei ole selle koha eesmärk koheselt arusaadav. [1] [9] Katte ja peidukohtade disain, peab lahendama taseme läbitavuse, keerukuse ja kiiruse. Ometi peab olema disain lihtne ja arusaadav. Samas ei tohiks valmistada see disain, mängijale segadust oma eesmärgi suhtes.

Alarmi olek määrab ära põhilise viisi, mängijale infot edastada. Seda just märkamatuse stiiliga mängul, kus see annab edasi infot, kuidas vastane käitub. Mängija jaoks on siinkohal tähtis saada infot vastase oleku kohta. Lisaks eelnevale sunnib ta mängijat vastu võtma otsuseid nagu näiteks kas põgeneda või võidelda ning üldisemalt annab edasi mängus toimuvat läbi vastase käitumise. [1] Siinkohal on tähtis, et mängija jaoks oleks arusaadav, millises olekus vastased on seda selleks, et ta saaks arvestada nende käitumisega ja võtta seeläbi vastu õigeid otsuseid.

Sensoorsete süsteemide loomine on väga tähtis osa tehisintellekti kujundamise protsessist. Kuna tehisintellekt ei tohiks kunagi omada, stealth žanri mängu puhul, terviklikku pilti olukorrast, peab tal olema võimalus koguda informatsiooni. [1] Sensoorsed süsteemid ja kuidas neid arendada, on üks põhiline keerukus arendamisel. Selle tulemusel määratakse ära kui keeruline on vastane ja millist informatsiooni ta saab koguda maailmast. Lisaks on vaja veel arvestada faktoriga, et loodud süsteemil oleks

mõte ja seos tehisintellekti käitumisega või alternatiivselt, peab see abistama tehisintellekti informatsiooni kogumise võimekust.

Tasemete loomisel tuleb lisada mängule väärtust, luues tasemed selliselt, et need ei langeks liiga lihtsa ega võimatu piiridesse. Tasemed peaks olema loodud selliselt, et mängu jaoks arendatud tehisintellekt, suudaks kontrollida ala tema ümber, seda juhul kui muid väliseid mõjutegureid ei ole. Juhul aga kui mängijal on vaja läbida mingi ala, peab olema tal selleks võimalus, kasutades talle kättesaadavaid vahendeid või taseme disaini selleks, et üle kavaldada oma vastane. [1]

Turvameeste ja turvameetmete disainimisel, tuleb arvestada ka tehisintellekti enda keerukusega. Selle puhul on tähtis otsustada, kui palju informatsiooni tehisintellekt omab. See saab olema üheks suurimaks probleemiks, kuna kui tehisintellekt teab liiga palju, siis mängija jaoks muutub taseme läbimine võimatuks ning kui tehisintellekt teab liiga vähe, siis on tasemed liiga kerged ja vastased jätavad väga ebanaturaalse mulje. Raskuseks saab siin olema ka loogika loomine, ehk kuidas peaks tehisintellekt ning teised tema ümber, reageerima mingile stiimulile. Tehisintellekt tuleb luua arvestades eelnevalt arutatud nõuet, kus tal peavad olema sensorid ning talle peab olema loodud ümber vastav keskkond, kus ta saab iseseisvalt hakkama. Põhilise tähtsusega, jääb vastase naturaalse ja ennustatava käitumise loomine selleks, et mängija ei saaks üllatatud iga kord kui ta vastast kohtab. [1]

Tavaliselt on vaja mängijal mingisugust eelist, kui talle on vastu mitu vastast [10]. Seega tuleb juba varajases staadiumis arvestada, kuidas luua mängijale eelis. See sõltub osaliselt sellest, mis stiilis mängu stealth mängu arendatakse. Nagu peatükis 2 sai käsitletud, on kaks võimalust mis stiilis stealth mänge arendada. Märulile keskenduvas mängus on põhiline probleem mängija elud. Mängija peab suutma võitlusmomendid edukalt seljatada ning seejärel peab tal olema võimalus, järgmiseks võitluseks uuesti ette valmistada. Teisalt on märkamatuks jäämisele keskenduvad mängud, kus mängija põhiliseks eesmärgiks on jääda nähtamatuks. Sellistel juhtudel on vaja mingisugust muud süsteemi, millega mängijale eelis anda, mis ei ole seotud mängija eludega.

3 Lahendused

Hakates disainima stealth mängu, tuleb arvesse võtta kõiki peatükis 2.2 määratletud probleeme ning piiranguid. Faktoritega nagu, kuidas disainida tehisintellekti ja mängija mängumehaanika, peab olema arvestatud juba varajases staadiumis ning pidevalt meeles hoidma. See on vajalik selleks, et mängus oleks igal asjal mõte, et vältida tühise funktsionaalsuse lisamist.

Mängija liikumine on üks probleemidest, mida käsitlesime peatükis 2.2. Üldjuhul on erinevates mängudes loodud mängija liikumiseks mitu erinevat viisi, see kujuneb välja hiilimise süsteemile lähenemisest [1]. Sellest kui põhjalikult liikumine on loodud, sõltub mängu kiirus ja tehisintellekti keerukus. Alustame kõige lihtsamast, milleks on staatiline kiirus. Selle lähenemise puhul, oleks mängijale omistatud ainult üks kiirus, sellega seoses ei ole mängijal väga palju vabadust, kuidas läheneda situatsioonidele loovalt, mis seab omakorda piirangud tehisintellekti loomisele. Teisalt saab luua süsteemi, kus liikumine on dünaamilisem. Kaasaegsetes hiilimis mängudes kasutatakse just säärast süsteemi, kus mängija saab liikuda ringi eri kiirustel ning vahetada oma kiirust, vastavalt vajadusele [1]. Selle puhul on võimalik luua erinevaid viise, kuidas hallata liikumist. Üks viis on kõige laialdasemalt levinud süsteem, kolme erineva kiirusega. Selle puhul, liigub mängija tava tingimustes standardsel kiirusel ning heli tugevus on keskmine. Sellele lisaks on mängijal võimalik kiiremaks ringi liikumiseks, kasutada spurtimist, mille puhul on küll kiirus suurem, kuid sama kehtib ka heliga mis mängija genereerib. Viimaseks on mängijale lisatud hiilides liikumine. Selle puhul on tehtav heli väga väike või isegi puudu, kuid tasemes ringi liikumise kiirus, on oluliselt väiksem, kui joostes või spurtides. Kuid täpne piiritlemine, pole ainus viis kuidas saab mõjutada liikumist. Võimalus on sama laadi süsteem luua aga üleminekud on sujuvad, ühest olekust teise, või anda mängijale sootuks valik, mis kiirusel ta tahab liikuda. Viimase puhul piiritledes ainult maksimaalse ja minimaalse liikumise kiirused. Kolmandaks on juurde arendus eelmisele, kus lisatakse ka parkuurimis võimalus, tasemes ringi liikumiseks. See annab mängijale võimaluse kasutada kolmest mõõtmest kõiki kolme selleks, et vastasest mööduda või oma missiooni/eesmärki saavutada.

Järgnevaks käsitleme kahte probleemi ning nende lahendusi. Taseme loomine ning sellega seoses ka katte ja peidukohtade loomine, on väga tähtis osa mängu disainist. Kui võitlus põhistel mängudel, saab luua taseme väga raske ja eeldada, et mängija kogeb mingil tasemel eksimust, siis hiilimis põhistel mängudel ei tohi eeldada, et mängija teeb eksimuse. Seega on vajalik iga hiilimise mängu kohtumisel, anda mängijale võimalus mitte läbi kukkuda, kuid samas ei tohiks liiga silmnähtav olla, kuidas neid võimalusi esitletakse. [9]

Maailma loomisel tuleb arvestada sellega, et olukorrad ei tohiks olla võimatud, kasutades hiilimist ega liiga lihtsad, sest siis ei ole vaja hiilimist. Seega tuleb maailma loomisel, arvestada asjade paigutamisega ja mängija ootuste haldamisega. Tasemed peaksid võimaldama, läbida neid vältides läbi kukkumist ning seda saab testida sellega, kui palju on hiilimist soodustavaid faktoreid ning palju on pärissvaid faktoreid mängijale saadaval. Soodustavad faktorid on siinkohal katte ja peidu kohad ning alad mis võimaldavad mängijal ennast peita. Need peavad olema loogiliselt paigutatud ning nende abil, saab hinnata mängija käitumist olukorra läbimisel. Katte ja peidukohtasid luua on võimalik erinevalt. Esimene nendest on disainida mängumaailm selliselt, kus mingid alad on just kui naturaalsed peitmise kohad. Nii võib kohata, mängudes tasemeid kus on loodud pimedad nurgad ja nurgatagused, kus mängija saab ennast ajutisel peita, kuni tehisintellekt temast möödub ning seejärel jätkata edasi liikumist. Teine võimalus on anda mängijale valik ennast asjade sisse peita, kasutades mängu taseme disainis, igasuguseid ruumis igapäevaselt leiduvaid esemeid. Nii näiteks on võimalik luua kapid, kastid ja tünnid ning muu sarnane, mida mängija saab kasutada. Takistavateks faktoriteks on maailma alad, kus mängija on avatud või kus tal pole võimalust olukorda kontrollida ning kohad mis reedavad ta asukoha, nagu näiteks valjem põrand või mööbliesemed mis teevad häält. Liiga palju soodustavaid faktoreid ja tasemed on kerged, liiga palju pärssivaid faktoreid ning tasemed muutvad võimatuks. Seega on taseme disainil tähtis leida balanss nii pärssivatest kui ka abistavatest faktoritest selleks, et mängijal jääks mulje, kindlalt kaitstud alast millest ta saab läbi, ainult kasutades oma oskusi [9]. Lisaks saab eelnevaid abitegureid ning pärssivaid faktoreid, kasutada ära taseme analüüsil. [9]

Selleks, et tasemele disainida igapäevaseid esemeid, on vajalik valida ka sobivad tööriistad modelleerimiseks. Kuna selle töö raames käsitletakse 3D mängu arendust, siis peame valima tarkvara, mis võimaldab meil mängu erinevaid komponente disainida kolmemõõtmeliselt. Töös kasutame selleks Blenderi tarkvara, kuna ta on tasuta ning väga

võimas ja sobib täpselt 3d mudelite loomiseks. Blenderil on ka lisa võimalusi, nagu animatsioonide loomine, UV mappide loomine[1], skulpteerimis võimalus ja palju muud, mida saab kasutada, kuid selle töö raames piirdume ainult modelleerimisega ning tekstuuride loomisega oma mudelile. Värvi piksli põhjad tekstuuride jaoks kujundasime GIMP programmi kasutades.

Kuna hiilimis mängudes on väga väike ebaõnnestumise piir, siis on vaja mängijale selgesti edasi anda, mis olekus ta vastased on. Seda saab edasi anda tehisintellekti järjepideva käitumisega, kus mängija saab oodata kindlat reaktsiooni tehisintellekti poolt. See on vajalik, sest tihtipeale on hiilimis mängudes läbikukkumiseks mängija avastamine, seega peame andma talle infot, kas teda otsitakse või mitte. Lisaks eelnevale, on vajalik selge sündmuste käik, mille saab välja selgitada vaatluse käigus. Seega peab olema tehisintellekti käitumine ettearvatav. [9]

Tehisintellekti sensoorsete süsteemide loomisel, on tähtis pidada meeles asjaolu, et see on tema viis koguda informatsiooni, teda ümbritsevast keskkonnast. Väga vähestel juhtudel on vajalik tehisintellektile luua maitse, puudutamise või lõhnataju, seega piirdutakse enamasti nägemise või kuulmise taju loomisega. [11]

Vaadates eelmistes lõikudes väljatoodud punkte selgub, et kokkuvõtvalt on mängija eeliseks enamus mängudes, tema oskused ja informatsioon. Nagu T.Francis oma artiklis ütles „Information is power“ [10] seega on väga tähtis mängija jaoks, just informatsioon. Selle põhjal saab ta formuleerida plaani, mida teha, kuidas teha ja millal teha. Seega peab mängijal olema võimalus rohkem teada, kui tehisintellektil. Seda siis läbi mängumehaanika, heli või mängu maailma disaini abil. Informatsiooni kogumiseks, on eri lahendusi võimalik rakendada. Näiteks saab luua tasemes kohad, mida R.Smith kirjeldas kui skautimise kohtadeks. Nende mõte on see, et mängija saab turvaliselt jälgida ümbrust, kartmata, et ta avastatakse. [9]

Üheks probleemseks kohaks veel mängude arendamisel, on versiooni haldus. Seda ei käsitletud eelnevas peatükis, kuna see probleem on üldisem mängude arenduses ning ei ole otseselt stealth žanrile unikaalne. Vaadates eelnevaid bakalaureuse töid, Tallinna

[1] UV mappide loomine – 3D kujundi projitseerimine 2D pinnale selleks, et talle saaks kaardistada tekstuuri [39]

Tehnikaülikoolis, mis kasutasid Unity mängumootorit, on versioonihaldusest räägitud ainult ühe lausega, Unity cloudist, K.Romuluse poolt. [12] [13] [14] [15] Seega käsitleme selle töö raames, ka versiooni haldust mängu arenduse jaoks. Versiooni halduse tarkvara sõltub mängumootorist mida kasutatakse. Erinevatel mängumootoritel on erinevad ühilduvused versioonihaldus tarkvaradega. Mängu arendusel on nii koodijuppe, kui ka palju disaini faile nagu mängija, vastaste ja mängumaailma esemete mudelid, mängu maailm ise, graafilised moodulid nagu kasutajaliides, tekstuurid erinevatele mudelitele, animatsioonid ning veel palju muud, mis ei ole seotud koodiga, kuid mängu arenduse käigus on vajalik seda hallata. Seega on vaja lahendust, mis suudaks hallata nii suuri faile kui ka koodi. Plastic SCM võimaldab hallata kõiki faile, mis tekivad mängu arenduse käigus, ilma probleemideta. Gitis on selle jaoks vaja ümbernurga lahendusi, nagu „Large File Storage“. [16] Etteruttavalt selle töö raames kasutame Unity mängumootorit, lähemalt sellest peatükis 4.1. Arvestades fakti, et Unity omab Plastic SCM tarkvara [17], siis teeb see selle tarkvara kasutamise väga mugavaks ja lihtsaks, mängu arendamisel Unity mängumootoriga.

4 Mängumootor

Mängumootor on tarkvara, mis võimaldab tema kasutajal, teha valmis video mäng. See aga ei ole päris täpne kirjeldus, selle mis on mängumootor. Põhiline mängumootori eesmärk, on abstraheerida üldised video mängu funktsioonid, võimaldamaks seeläbi taaskasutada koodi ja mängu vara teistes mängudes. Tüüpiliselt on mängumootoritel järgnev funktsionaalsus:

- Renderdamis mootor 2D ja 3D graafika jaoks
- Sisendite käsitsemine (klaviatuuri ja hiire, puutetundlike, teise riistvara jne)
- Mängu loop (sisemine rutiin mis kalkuleerib iga kaader mängus toimuvat)
- Füüsika mootor, millel on ka kokkupõrke tuvastus ja sellele reageerimine
- Heli
- Stseeni graaf (haldab graafiliseid elemente ekraanil)
- Animatsioonid (2D graafikale ja 3D mudelitele)
- Mälu haldus
- Protsesside threading (lubab mitut paralleelselt töötavat protsessi)

Lisa funktsionaalsusena võib olla lisatud:

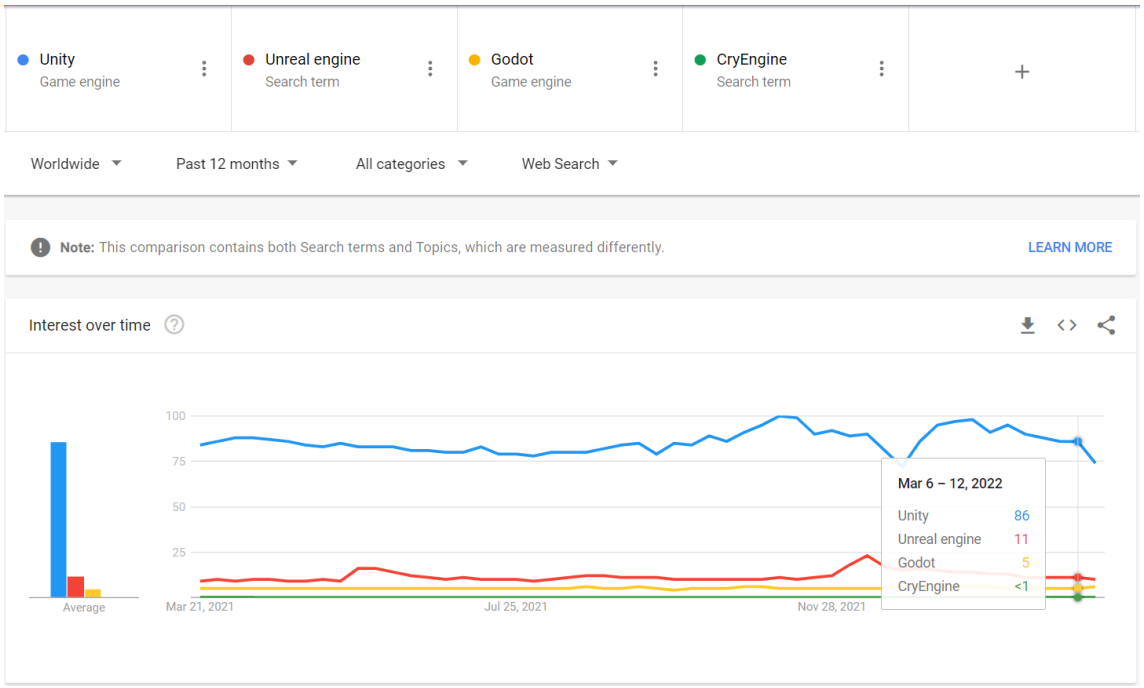
- Skriptimine
- Tehisintellekt
- Võrgundus
- Striimimine
- Lokalisatsiooni tugi

- Mitmele platvormile publitseerimine

[18]

4.1 Miks Unity?

Unity on 2005. aastal välja tulnud mängumootor, mis oli algselt MACile loodud. Alguses oli Unity mõte, pakkuda kättesaadava hinna eest, professionaalseid tööriistu algajatele mänguloojatele. 2008. aastal kui Apple tuli välja App Store'ga, siis oli Unity esimene mängumootor, kes toetas seda platvormi ning samal aastal välja tulnud mäng nimega „FusionFall", aitas lisada mängumootorile populaarsust. Hiljem on erinevad suure firmad, nagu Electronic Arts, Ubisoft ja Microsoft olnud nende kliendid. [19] Nagu eelnevast tekstist näha, siis on Unity päris kaua olemas olnud ning ka suuremate firmade tähelepanu äratanud, seega on mängumootor vägagi võimekas. Samas kui vaadata erinevate mängumootorite trendi Google otsingumootoris, on selgelt näha, et Unity mängumootorit otsitakse kõige rohkem. (Joonis 1)



Joonis 1. Mängumootorite esinemise populaarsus Google otsingumootoris.

Sellest saab järeldada, et see on kõige populaarsem neist ning seega on olemas selle kohta rohkelt lisa materjali, mida probleemide esinemisel saab kasutada. Samuti on näha, et Unity toetab väga palju erinevaid platvorme, millest kõiki teised mängumootorid ei toeta. (

Tabel 1)

Tabel 1. Mängumootorite platvormide tugi.

	Unity	Unreal	Godot	CryEngine
iOS	X	X	X	
Android	X	X	X	
Windows PC	X	X	X	X
Universal Windows Platform	X		X	
Linux	X	X	X	X
WebGL	X			
Playstation 4	X	X	X	X
Playstation 5	X	X		
Xbox Series X	X	X		
Xbox Series S	X	X		
Xbox One	X	X	X	X
Oculus	X	X		X
androidtv	X			
tvOS	X			
Nintendo Switch	X	X	X	
ARCore	X	X		
Stadia	X	X		
Microsoft HoloLens	X	X		
Magic Leap	X	X		
HTML5		X	X	
BSD			X	

Unity kasutamine on ka hea sellepärast, et see võimaldab meil kasutada Plastic SCM tarkvara, mis kuulub Unityle [17]. Teiste mängumootorite puhul, kasutavad nad kolmandate osapoolte lahendusi versiooni halduseks. Sellest tulenevalt, üks põhjuseid miks valime Unity, on ka fakt, et sellega saama kogu projekti arenduse seonduva korraga ühest kohast ning saame eeldada, et saame süsteemi mis juba oma eos, on disainitud koos töötama. Kõik võrreldud mängumootorid on kättesaadavad tasuta mis on boonuseks neile.

4.2 Arendus tarkvara ja C# keel

Mängu skriptide programmeerimiseks, kasutatakse Microsoft Visual Studio Community 2019 tarkvara ning C# programmeerimiskeelt. Microsoft Visual Studio Community 2019 on võimalik panna kiiresti peale Unity editori installeerimisel, lisa moodulina ning ta on tasuta tarkvara, millega on võimalus programmeerida C# keeles. Programmeerimis keeltest on Unityl valikus ainult C# [20] seega seda ka kasutame.

5 Nõuded enda mängule

1. Kasutame Unity 3D mängumootorit mängu loomiseks.
2. Kasutame Blenderi tarkvara mudelite loomiseks.
3. Programmeerimiseks kasutatakse keelt C#.
4. Versioonihaldus toimub kasutades Plastic SCM tarkvara.
5. Ülesannete ning aja logimiseks kasutame Trello keskkonda.
6. Mäng peab vastama stealth žanri klassifikatsioonile ehk vastama nõuetele mis teevad mängust stealth mängu.
7. Mäng peab olema mängitav ilma suuremate vigadeta (*inkl. bugs*).
8. Tehisintellekt peab suutma reageerida stiimulile. (Erinevad hääled)
9. Tehisintellekt peab olema võimeline mängus iseseisvalt hakkama saama olukordadega.
10. Tehisintellekt peab suutma avastatud mängija kõrvaldada ehk läbi kukutada mängija eesmärk ja seega tekitada kaotatud mäng.
11. Tehisintellekt peab suutma koos töötada mängija avastamiseks.
12. Mängijal peab olema võimalik mängu mängida ehk ringi liikuda erinevatel kiirustel.
13. Mängija peab saama asju maast võtta.
14. Mängija peab saama loopida asju, mida on maast korjanud.
15. Mängijal peab olema selge eesmärk kuidas mängu võita/läbida.
16. Mängija ei tohiks kõrvaldada saada tehisintellekti.

17. Mängijal peab olema võimalus tehisintellekt üle kavaldada enamus olukordades, kus mängijal on vaja temast mööduda.

18. Maailma peab olema läbitav, see tähendab ükski koht ei tohi olla läbikukutav või läbimatu väljaarvatud mängija avastamine tehisintellekti poolt.

5.1 Nõuete analüüs

Kõik järgnevad nõuded mis on lahti analüüsitud, on peatükist 5, kui on mainitud järgnevates lõikudes punkti, siis on mõeldud peatükis 5 nummerdatud loetelu all olevat ühte punkti. Nõuded võib väga jagada selle töö raames kolmeks. Nõuded on need, mis seavad piirangud korraldusele ja mängu arendusele endale. Nõuded 1–5 on puhtal kujul, nõuded tarkvarale ning selle kasutamisele, seega nende nõuete täitmisega ei tohiks probleemiks tulla. Need tulenevad eelnevalt põhjendatud otsustest. Punktis 6 toodud nõude täitmiseks, on vaja natuke analüüsida mängu ja kuidas ta on mängitav. Seda nõuet arvestame kogu arenduse vältel ning üritame mängu arendada, alati selle kaalutlusega, et see punkt saaks täidetud. Punktis 7 toodud nõuet, saame testida alles mängu mängides testimise faasis. Seda sellepärast, et suuremate vigade mitte esinemist saab hinnata alles, peale mängu valmis saamist. Punktis 18 toodud nõuet, arvestame mängu maailma disainimisel ja täitmisel erinevate esemetega. Siinkohal peame tegema analüüsi sellele, kas tase on alati läbitav igas olukorras. Siin kohal tulevad appi analüüsimisel peatükis 3 mainitud pärissivad ja soodustavad faktorid.

Järgnevalt on teine alamkategooria, milleks on punktides 8–11 on toodud nõuded, mis on seotud tehisintellektiga. Alustades algusest, siis on punkt 8, mis räägib tehisintellekti reageerimisest stiimulile. Töö jooksul arendame süsteemi, mis võimaldab tehisintellektil reageerida stiimulile, milleks saab olema heli. Asjade loopimisel, tekib heli mis tõmbab tehisintellekti tähelepanu ja võimaldab seeläbi kavaldada üle või isegi suunata tehisintellekti, enda jaoks sobivamasse kohta. Stiimulina ei käsitleta selle töö raames mingeid muud sensoorseid lahendusi, mis on toodud välja peatükis 3. Selle nõude täitmise kohta saame infot sellest, kui tehisintellekti saab mängu mängimise ajal mõjutada. Punkti 9 täitmiseks on vaja tehisintellekt selliselt luua, et ta saab talle ette juhtunud olukorraga hakkama ja suudaks õigesti reageerida mängus toimuvale. Selle kohta on raske piiritleda, mida tähendab õigesti reageerida olukorrale. Peamiselt mõtleme selle all seda, et ta reageerimisel stiimulile, peaks ta käitumine olema ettearvatav ja ei tohi tekkida

anomaaliat, millega üllatatakse mängijat. Punkti 10 täitumist, saame hinnata väga lihtsalt. Kui mängus tehisintellekt suudab mängija avastada ning ta kõrvalda, siis peab ka järgnema mängu kaotus. Punkti 11 täituvust on natuke raskem piiritleda, kuid siinkohal mõtleme info jagamiseks tehisintellektide vahel seda, kui palju on neist keegi midagi kahtlast kuulnud nii lühi, kui pika ajaliselt. Samuti salvestatakse viimase kahtlase heli asukoht ning kutsutakse raadiusesse jäävad droonid abistama otsimisel, seega mida rohkem heli tehakse, seda raskemaks mäng läheb, kuna abijõude tuleb rohkem mängijat otsima.

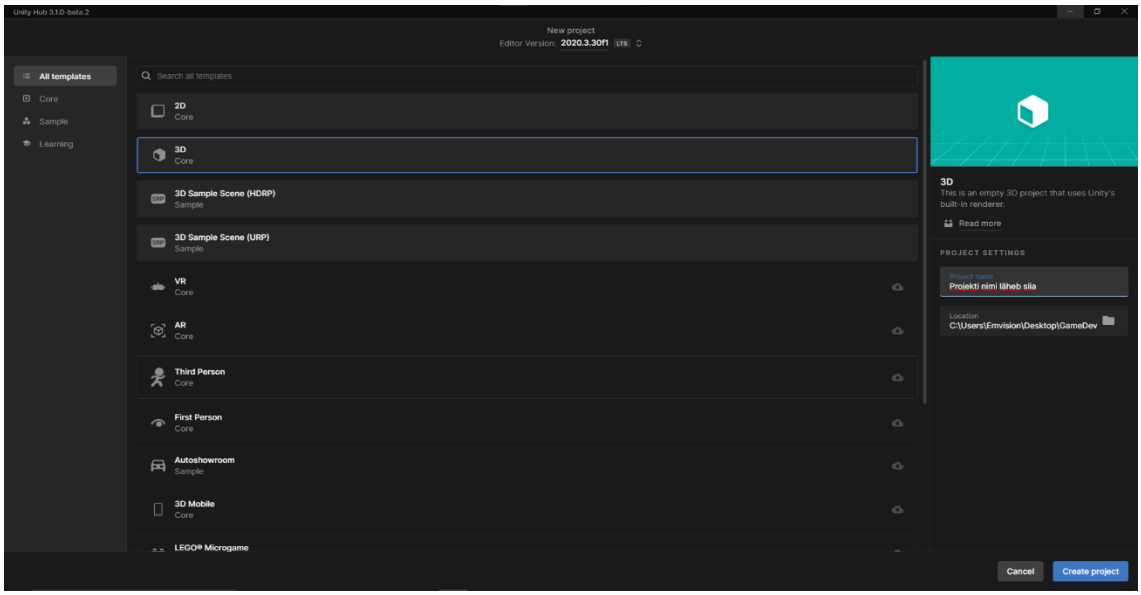
Viimaseks olulisemaks nõuete kohaks on mängijaga seonduvad nõuded, mis kategoriseeruvad punktide 12–17 alla. Alustades jällegi madalamast ja liikudes kõrgema poole, siis 12. punkti testimine ja loogika on lihtne. Siinkohal peab mängija olema võimeline mängus ringi liikuma. Videomängudes saab mängija tavaliselt ringi liikuda nuppudega „wasd“ ning liigutades hiirt, saab mängija mõjutada oma vaatevälja. Sama mängumehaanika, luuakse ka selle punktiga seoses. Punktis 13 on mainitud asjade maast ülesse korjamine, mis nõuab lisa objektide loomist ning mängija ja nende objektide vahelise loogika loomist. Mängija peaks olema piisavalt ligidal asjale, et ta saaks selle üldse ülesse korjata ning lisaks on vaja luua mängijal võimalus talletada eset käes. Nõue 14 on seotud nõudega 8, mille korral mängija peab saama maast korjatud esemeid loopida, et tekitada heli, millele tehisintellekt reageerib. Nõue 15 näeb ette, et mängijal peab olema selge eesmärk, kuidas mängu võita või läbida. Siinkohal tuleb loomise käigus lühike lugu mõelda maailmale, et anda mõte sellele, mida mängija üritab saavutada ning kommunikeerida see läbi kasutajaliidese, mängijale edasi. Nõue 16 tuleb faktist, et kui lisada mängijale võimalus tehisintellekti kõrvaldada, siis on vaja süsteemi, millega mängijal on võimalik reageerida ka tehisintellektiga eraldi. Lisaks sellele, on siin kohal vaja ka laiendada mängija läbikukkumise kriteeriumeid, sellest tulenevalt ei saa enam mängija avastamisel ta kohe läbi kukutada, vaid peab võimalus olema osaliselt võidelda, mis klassifitseeriks mängu juba märuli kategooriasse, mitte puhtalt stealth žanri mänguks. Tulenevalt eelmisest nõudest, kuna mängija ei saa tehisintellekti kõrvaldada, peab mängija suutma teda üle kavaldada oma oskuseid ja mängu tasemest tulevaid võimalusi arvestades, seda juhul kui temast mööda pääsemine, on edasi liikumiseks vältimatu.

6 Mängu valmistamine

Mängu valmistamisel jälgiti peatükis 5 kehtestatud nõudeid. Selle projekti raames, rakendasime arendusel rohkem analüütilist ja piiritletud arendust, mitte loovalt lähenemist. Seega ei hakanud enne mängu arendama kui nõuded olid paigas. Alguses seati ülesse kogu arendus keskkond. Selle järgnes töö ülesse seadmine Trello keskkonnas, kus loodi tehtava töö funktsionaalsust käsitlevad ülesanded. Peale seda hakkas pihta arendus protsess, mille käigus tehti branch Plastic SCM keskkonnas ning seejärel sinna branchi, loodi funktsionaalsus mille kallal töötati.

6.1 Unity ülesse seadmine

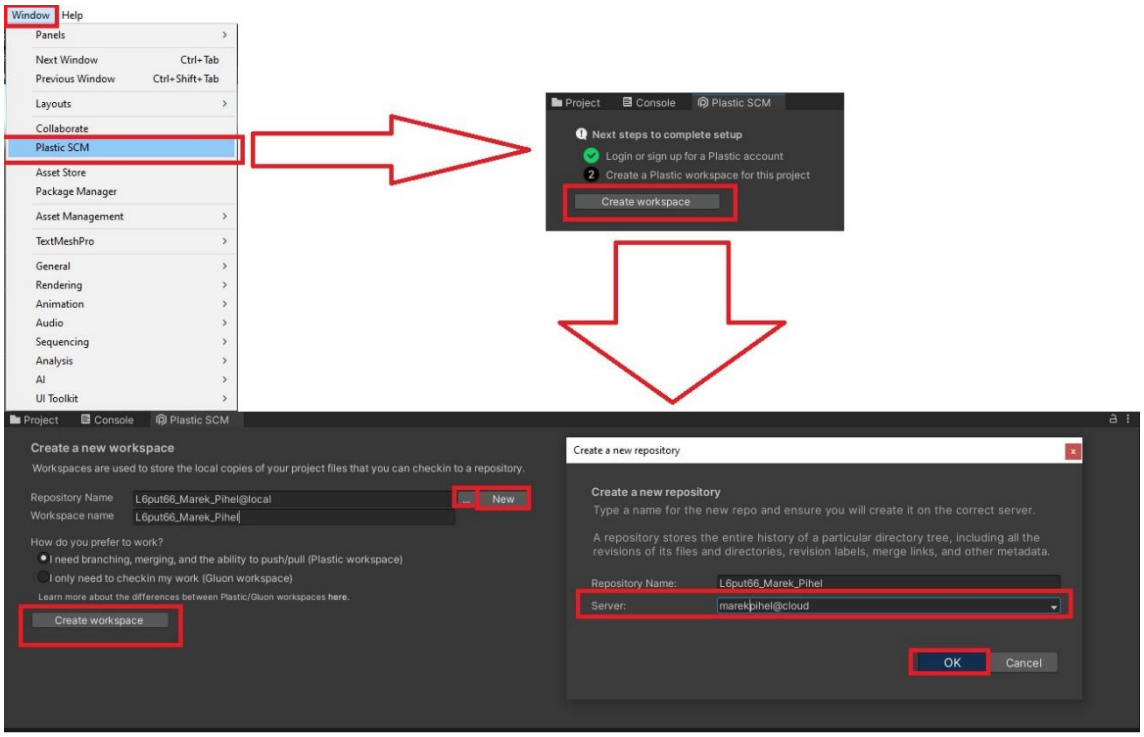
Unity hub on paigaldatud Unity kodulehelt. [21] Seejärel, on kasutades Unity hubi paigaldatud Unity editor programm, kasutades selles olevat „Installs“ vahelehte ja ülevalt võetud „Install Editor“, seejärel paigaldati avanevast aknast „LONG TERM SUPPORT (LTS)“ alt Unity 2020.3.30f1 versioon, vajutades „Install“ nupule. Lisana on pandud „Dev tools“ alt „Microsoft Visual Studio Community 2019“ peale, kuna Unity enda sees olev C# koodi kirjutamise programm ei ole väga võimekas, võrreldes Microsoft Visual Studio Communityga. Moodulitest on pandud peale „Universal Windows Platform Build Support“. Kui Editor oli peal, tuli luua uus projekt „Projects“ menüü alt, kasutades „New Project“ ja seejärel valides 3D ning andes projektile nime. (Joonis 2.)



Joonis 2. Unity 3D projekti loomine.

6.2 Versiooni haldus

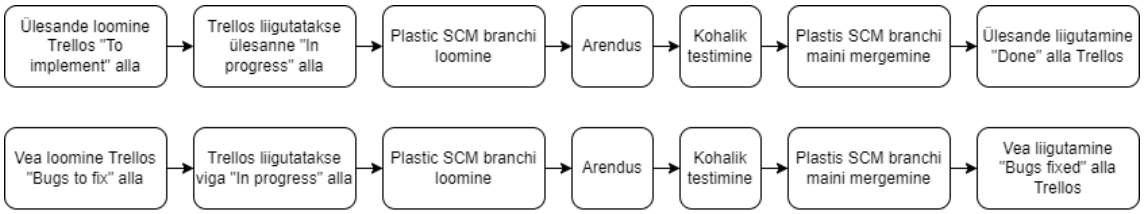
Versiooni haldus tarkvara valikul, otsustasin Plastic SCM kasuks. Selle otsuse taga on kaks erinevat asjaolu. Üks on see, et Unity ostis mõnda aega tagasi ära Plastic SCM tarkvara. [17] Seega ühildub see väga hästi Unity endaga. Teiseks on see, et Plastic SCM võimaldab mängu arendajatel, väga kergesti ning kiiresti hallata kogu mänguga seonduvat projekti, kaasarvatud suuri faile ning võimaldab branchide haldamist paremini kui git, see on tähtis mängu arendamiseks. [16] Plastic SCM endasse sisse logimiseks, kasutasin Unity kasutajat. Ülesse seadmine oli väga lihtne. Olles alla laadinud programmi nende lehelt ja peale installeerinud, tegin Unityt kasutades uue 3D projekti, kus ülevalt „Window“ alt vajutasin „Plastic SCM“ ja sealt avanevast aknast vajutasin „Create workspace nupule“. Seejärel avanevast menüüst, valisin Repository Name tagant „New“ ning locali asemel serveri alt võtsin enda cloudi. Peale „OK“ vajutamist tuleb valida „…“ alt, enda loodud cloudi repositoorium ning vajutada „Create workspace“ ja see tekitab repositooriumi koos branchiga. (Joonis 3)



Joonis 3. Plastic SCM versiooni halduse seadistamine.

6.3 Ülesannete ja aja logimine

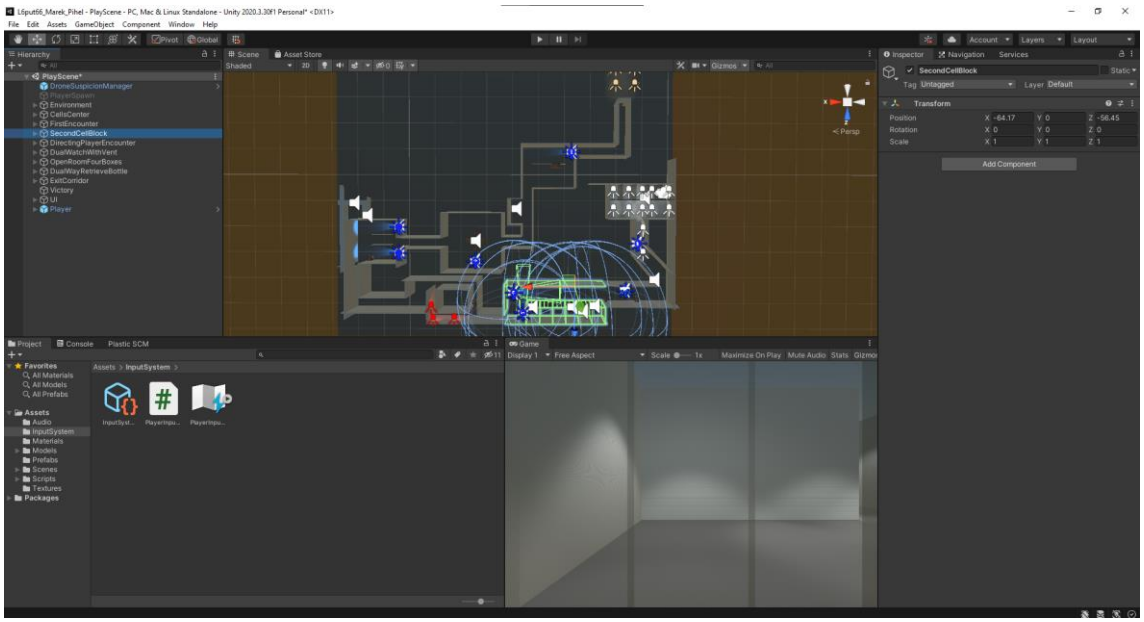
Projekti ülesannete tegemiseks ja jälgimiseks ning ajalise logi tekitamiseks, kasutan Trello keskkonda. Trellot sellepärast, et ta on tasuta ning kiiresti ja lihtsalt ülesse seatav ja sobib sellise skoobiga töö jaoks. Trellos loodi eraldi workspace, selle projekti jaoks ning kogu arendusega seotud ülesannete ja vigade haldamine toimus seal. Lisaks võimaldas Trello, ka aega logida kasutades power-upi, nimega „Time Tracker - Chronos“. See oli hea viis arvestust pidada, mingile ülesandele või veale kulunud ajaga. Samuti peab see arvestust selle kohta, kus listis praegu mingi funktsionaalsus asub seega saab ka ülevaate sellest, mis vajab veel tegemist ning mis on juba valmis. On ka näha, et projekti lõpuks said kõik asjad „Done“ ja „Bugs fixed“ alla, seega said kõik vead lahendatud ning vajalik funktsionaalsus loodud. (Lisa 3) Trellot oli ka hea kasutada sellepärast, et see võimaldas omada ülevaadet arenduse käigust. Ülesannetest, mis on veel vaja teha ning mis juba tehtud on. Lisaks võimaldas see ka saada ülevaadet sellest, mille kallal hetkel töötatakse. Ülesannete ja vigade elutsükli korral, oli Trello koht, kus võeti neid töösse ning samuti liigutati tehtud alla, kui funktsionaalsus oli loodud, või kui viga oli parandatud. Joonis 4 pealt on näha, kuidas erinevate vigade ja ülesannete puhul elutsükkel välja nägi arenduse käigus.



Joonis 4. Ülesande ja vea elutsükkel arenduse käigus.

6.4 Unity programmi osad ning nende kasutamine selle projekti raames

Joonis 5 peal on välja toodud vaade mis esines Unity aknast selle projekti arendamise raames. Vasakult ülevalt leiame „Hierarchy“ akna, kus on praeguses stseenis olevad mänguobjektid toodud. Lisaks sellele on seal ära toodud, mis stseenid on hetkel lahti, sellest räägime täpsemalt stseenide peatükis. Vasakul all avaneb vaade projektile. Siin leiab kaustad ja struktuuri kogu projekti puudutavate failide kohta. Projekti raames jagati kaustad loogilisteks osadeks ning ühtse funktsionaalsuse järgi. Lisaks leiab ka samast paremalt eraldi tabi alt, „Plastic SCM“, akna kus toimus projekti raames versiooni haldus, mis toimus Unity enda seest. Sealt enamasti pushiti muudatusi branchi, mille kallal töötati. Selle kõrval on omakorda „Game“ aken, kus toimus play nupule vajutades, mängu testimine ehk vaade mängija perspektiivist. Edasi liikudes paremale, kus ülevalt alla välja, tuleb „Inspector“ ja „Navigation“ aken. „Inspector“ aknas toimus kogu mänguobjektidega seonduvate osade haldamine, mis oli vajalik erinevate mängu komponentide loomiseks. „Navigation“ akna alt toimus projekti navigatsiooni loomine, maailma jaoks, seal täpsustatakse täpsemalt tehisintellektiga seotud parameetreid selleks, et genereerida kõnnitav pind tema jaoks. Keskelt leiame „Scene“ akna, kus on näha praegune stseen mängu maailmas ja kus toimub mänguobjektide positsioneerimine ning muutmine mängu tarvis, mis on asetatud praegusesse stseeni, vasakul oleva „Hierarchy“ all. Kõige ülevalt leiame nupud Play, Pause ja Step, mis haldavad mängu mängimise edasi liikumist, editoris endas.



Joonis 5. Unity editori vaade selle projekti raames.

6.4.1 Stseen

Stseeni vaade, on interaktiivne vaade loodavasse mängumaailma. Stseenidega manipuleerimine ja asjade asetamine sinna, on üks põhilisi oskusi Unityga töötamisel [22]. Projektis on kasutatud erinevaid stseene selleks, et grupeerida tervik funktsionaalsus või osa. Nii on loodud UI (User Interface) stseen, algne maailma stseen ja põhi mängu stseen, lisaks on ka testimise stseen, kus uue funktsionaalsuse loomisel sai kiiresti testida seda, enne kui lisada põhi mängu stseenile. Üks asi mida selle projekti raames sai kasutatud, on mitme stseeni koostöötamine. Varasemad bakalaureuse tööd, ei ole käsitlenud seda aspekti, kuid see on väga hea asi mida kasutada. Sellega on võimalik luua huvitavaid lahendusi ja eristada tervikuid üksteisest. Nii on loodud UI stseen, kus on ainult UI osa, mis on menüüde jaoks vajalik. Seal asetsevad ainult UI komponendid ning mida laetakse ja eemaldatakse vaatest, vastavalt vajadusele. Seda lisaks siis juba laetud stseenile. Mängu pausi aken ja pea menüü, on loodud selle loogikaga.

6.4.2 Kasutajaliides

Selle projekti raames, kasutati kasutajaliidese loomiseks Unity UI mänguobjektide põhist süsteemi mis on vanem süsteem, võrreldes uue Unity UI toolkitiga. See kasutab komponentide ja mängu vaadet selleks, et määrata stiil ning positsioon UI'le [23]. Kuigi see on vanem, sobis see ideaalselt vajaliku UI loomiseks. Kuna see põhineb mänguobjektide loogikal, oli elementide paigutamine väga lihtne ja loogiline. Seda

kasutades, loodi kasutajaliidese element vasakul „Hierarchy“ aknas ning seejärel „Inspector“ aknas sai kõik vajalik paigutus ja loogika liidetud kasutaja liidese elementidega. Lisaks sai UI suurus pandud sõltuvusse ekraani resolutsioonist.

6.4.3 Mänguobjekt

Unity GameObject (mänguobjekt) klassi kasutatakse kõigeks, mis saab stseenis eksisteerida. Mänguobjektidel on tag ja layer osad. Tagid võimaldavad grupeerida ja vajadusel leida ning layerite puhul saab välistada mänguobjekte, teatud renderdamis protsessist [24]. Nii tag kui ka layer funktsionaalsust kasutati selle töö raames. Tage kasutates, loodi seosed erinevate mänguobjektide vahel ning loogika loomisel. Loodud tagid projektis, aitasid koodis liita funktsionaalsust. „Environment“ tag mida kasutati pudeli katki mineku loogika jaoks. „Cell“ tag mille korral mängija viibib kongis ning on kaitstud tehisintellekti eest ning tehisintellekt ei reageeri talle. „Player“ tagi kasutatakse tehisintellekti ja mängija vahelise loogika puhul, võidu ekraani kuvamiseks ning pudeli ülesse võtmiseks. „Drone“ tagi kasutatakse selleks, et panna tehisintellekt virtuaalsele helile reageerima. „Bottle“ tagi kasutame pudeli ülesvõtmisel. Layeri puhul kasutame „Drone“ layerit. See on vajalik, et skänneri valgusvihust välja võtta drooni enda mudel, varjude tekitamisel. Kuna skänneri ala näitamiseks, kasutame spot valgustit ning selle asukoht jääb drooni seljataha. Valgusti asukoht oli vajalik, kuna vastasel juhul on valgustatud ala algus, kaugemal reaalselt alast, kus droon suudab juba skännida.

6.4.4 Input system

Kõige kiirem viis saada mängija sisendit, on lugeda seadme olekut. Selle lähenemise puudus on aga see, et vaja on iga seadme kohta eraldi teada, kuidas teda lugeda. See võib tekitada muudatuse tegemisel mängus, väga palju peavalu. Selleks, et lahutada spetsiifiline seade ja funktsioon mida ta mängus käivitab, on hea kasutada Actioneid selleks, et vahendada sisendeid. [25] Selleks, et luua see süsteem, lisasime mängijale komponendi Player Input, millelt vajutades create actions, lisatakse algsed parameetrid. Seejärel lisasime avanevasse listi juurde Sneak, Sprint ja ToggleMenu Actionid ning valisime neile nupud. Seejärel Unity assets alt vajutades tekkinud Input Actions peale, valisime „Generate C# Class“ mille korral tekitati meile skript, mida saime kasutada selleks, et kuulata evente, mille korral saame käivitada teatud funktsionaalsust oma koodis.

6.4.5 Skript

Skriptimine on väga tähtis osa Unityga loodud rakenduses. Skriptimisega luuakse seosed sündmustele, mis mängus toimuvad. Nii saab luua skriptimisega sisendite vastuvõtmist mängijalt, graafilisi efekte, mõjutada objektide käitumist maailmas või luua kohandatud loogika tehisintellektile. [26] Selle projekti raames, sai loodud 20 erinevat skripti mis asetsevad kõik „Scripts“ kaustas. Lisaks sellele, sai kasutatud Unity enda poolt loodud „PlayerInputActions“ skripti, mis oli vajalik kasutamaks Unity Input Systemit, mida kasutati mängija poolt sisendite saamiseks.

Drone kaustast leiame States kausta, millest leiame InvestigateState.cs, patrolState.cs, SearchingState.cs. Need defineerivad drooni kolme oleku, sisemist loogikat ning käitumist kui droon on nendes olekutes. Minnes States kaustast tagasi Drone kausta, leiame skriptid nimedega DroneActivator.cs, DroneBehaviourController.cs, ScannerController.cs, mis hoolitsevad drooni käitumise eest. DroneActivator skripti kasutatakse esmase mulje loomisel, kus kõige esimene droon pannakse mängumaailma teatud aja möödumisel. ScannerController on drooni skript, mida kasutab drooni skänner ning millega tuvastatakse mängija ja vaadatakse, kas tegelikult ka droon näeb mängijat. Selleks tekitatakse joon skänneri asukohast, kuni mängijani ja kui midagi on nende vahel, siis droon ei näe mängijat. DroneBehaviourController on kõige suurem skript mis oli drooniga seotud. Käitumise loogika põhines lõplikul olekumasinal, kus droon oli korraga ühes olekus ning teda oli võimalik mõjutada, üle minemiseks teise olekusse. Siin olid ka kõik algväärtustamised drooni jaoks, kus anti vajalik informatsioon, et droon saaks toimida maailmas iseseisvalt. Lisaks oli algväärtustamine vajalik ka selleks, et kui mängija kukkus mängu läbi, oli vaja droonid panna tagasi algolekusse, et välistada fantoomkäitumine, nagu mängu alguses droon on kohe otsingu olekus, mitte patrull olekus nagu vaja.

Managers kausta alt, leiame mängu erinevad halduse skriptid, mis on vajalikud sujuvuse jaoks. Siit leiame seega DroneSuspicionManager.cs, GameManager.cs, UIManager.cs ja WinLossScreenManager.cs skriptid. DroneSuspicionManager haldab droonide info jagamist ning mitmekesi reageerimise koordineerimist, droonide vahel. Samas leiame aga ka siit alt helide haldamise ning droonide otsingu olekusse panemise funktsionaalsuse, sõltuvalt mängija tekitatud lühiajalisest helist. Samuti saavad siit droonid infot, kui suurelt alalt peaksid nad otsima, huvi pakkuva koha ümber, mis on sõltuv sellest kui palju

mängija on heli teinud. Samuti salvestab see viimase tehtud heli asukoha. GameManager alt leiab loogika, mis puudutab stseenide muutmist, sõltuvalt sellest millele mängija vajutab UI peal ning menüü avamine kui escape klahvile vajutada. Menüü avamisel escape klahviga kasutatakse Unity event süsteemi, millega saab määrata skriptis eventi juhtumisel, käivitatav funktsionaalsus. Samuti pannakse mäng kinni, kui vajutatakse Exit nupule. Selle skripti kaudu, saab ka informatsiooni selle kohta, kas menüü on avatud või mitte. See on vajalik, et peatada mängus toimuv tegevus. UIManager skripti alt leiame loogika UI komponentidele ehk kui UI on lahti ja vajutame mingi komponendi peale, siis milline on soovitud käitumine. Kui vaja on stseeni muuta, siis kasutatakse selleks GameManageri. WinLossScreenManager toimib kohaliku UIna mängu stseenis, mis avab või sulgeb võidu või kaotuse UI, sõltuvalt kumb juhtus.

Misc kaustast leiame skriptid, mis on seotud muude protseduuridega, nagu näiteks pudeli või maalima animeerimisega. Bottle kaustast leiame BottleBreakingSound.cs ja BottleController.cs ning Environmental kaustast leiame DoorOpener.cs ja LightDestroyer.cs. BottleController haldab pudeliga seotud loogikat. Selle alla kuuluvad erinevad stsenaariumid, mida võib pudeliga ette juhtuda. Alustades viskamisega ning seejärel kokkupõrge keskkonnaga, millele järgneb lähedal oleva drooni teavitamine läbi simuleeritud heli. Lisaks genereeritakse mängijale heli kuulamiseks, sinna ka eraldi mänguobjekt nimega BottleBreakingSound. Sellel on sama nimeline skript küljes, mis mängib audio faili ära ning kui ta on lõpetanud, hävitab sama objekti. See hoiab kokku mälumahtu ja seeläbi on mäng sujuvam, ka vähem jõudsatel arvutitel.

Player kaustast leiame kõik mängijaga seotud skriptid. Selle projekti raames sai loodud ItemSlot.cs, PlayerItemManagementSystem.cs, PlayerMovementController.cs ja VirtualFootStepSound.cs. Alustame siinkohal PMCga (PlayerMovementControlleriga), mis on selle projekti raames, kõige mahukam ise loodud skript. PMC skriptis kuulatakse mängija liigutamiseks vajalikke evente. Kuulame Move, Sprint, Sneak ja Look evente ning nende esinemisel, käivitame funktsiooni koodis. Viimase puhul kui seda kuuleme, siis muudame mängija pööret maailmas. Ülesse ja alla vaatamise nurk on piiratud seose sellega, et simuleerida natukenegi reaalset vaatevälja. Vastavalt toimunud eventile, väärtustatakse teatud muutuja või siis täidetakse kohe mingi osa koodist. Move, Sprint ning Sneak puhul väärtustame muutujad ning kasutame seda Update funktsioonis, mida Unity jooksutab iga kaader. VirtualFootStepSound on skript mis haldab seda, kui kaugelt kuuleb droon mängija samme. Ta on virtuaalne, sest reaalselt ei mõjuta sammu heli

drooni. Selleks, et droone mõjutada, oli vaja eraldi süsteem luua, kus praegusel hetkel kasutame triggerit, mis drooniga reageerimisel annab droonile mängija asukoha punktina, mida uurida. Selle triggeri suurust muudame just sellest skriptis. Kui mängija on vangikongis, siis ei tekita me virtuaalseid samme, seega droonid ei reageeri mängija liikumisele kongis olles. Lisaks peatükis 6.4.3 räägitud „Cell“ tagi abil tekitatud kaitsest, võimaldab see mängijal skautida ja planeerida mida teha. ItemSlot on skript, mis ainult haldab mängija käes olevat eset. See on ehitatud ülesse iseseisva süsteemina ning ei ole sõltuv pudelist, seega saab süsteemi laiendada ka vajadusel teiste objektide peale. PlayerItemManagementSystem haldab mängija maast asjade ülesse võtmise ja loopimise loogikat. Siit käivitatakse loopimine ning loogika kuidas maast asja korjata. See süsteem kasutab käes hoidmiseks ItemSlot skripti, mille korral ei korjata asju, kui käes juba on midagi ning kui visatakse, siis ka tehakse ItemSlot tühjaks.

Viimase kaustana leiame StateMachine kausta, kust leiame 2 skripti State.cs ja StateMachine.cs. State on abstraktne klass, mida kasutavad kõik kolm Drone all, States kaustas nimetatud oleku skripti. See defineerib mõned üldisemad funktsioonid, mida saavad alam klassid kasutada, kes pärivad need sellest klassist. StateMachine haldab olekut. Ta salvestab oleku, mis talle määratakse ning on võimeline seda muutma või tagastama. Seda kasutame omakorda, et lihtsustada DroneBehaviourController skriptis, drooni käitumise ellu viimist.

SerlializeField on Unity võtmesõna mis lubab läbi editori, määrata teatud muutujatele väärtuse ilma, et nende muutujate skoop oleks public. [27] Seda on kasutatud läbi projekti vastavalt vajadusele, et liita teatud osasid skriptidele juurde selleks, et mitte otsida läbi stseeni neid mänguobjekte või osasid eraldi ülesse.

6.4.6 Valgustus

Spot light on valgustus mida kasutatakse tihti taskulampide, auto tulede ja otsingutulede puhul [28]. Kuigi valgustusi on Unitys veel, siis selle töö raames kasutati ainult spot lighti. Spot light kasutati algse tulede kustumise simuleerimiseks ning drooni skännerite ala demonstreerimiseks. See oli vajalik, kuna kaamera oli first-person perspektiivist ja seega andis valgus hästi ära, seda kuhu droon vaatas. Mis andis võimaluse mängijale kaugelt ja nurkade tagant infot koguda.

6.4.7 NavMesh

NavMeshi loomist taseme geomeetriast, kutsutakse NavMesh bakinguks. Selle protsessi käigus, kogutakse kokku renderdatud meshid ja maastik mis on markeeritud kui „Navigation Static“. Seejärel töödeldakse need läbi ning tehakse navigation mesh, mis väljendab ligikaudselt maailmas kõnnitavat ala. [29] Peale NavMeshi bakemist saab luua „NavMesh Agent“ tegelase, kes saab navigeerida mööda stseeni, millel on loodud NavMesh. Seda saab lisades oma mänguobjektile, „NavMesh Agent“ komponendi. [30] Seda kasutati tehisintellekti loomisel. Kus drooni objektile, lisati juurde „NavMesh Agent“ komponent, mida kasutatakse selleks, et liikuda eri punktide vahel.

6.4.8 Collider

Collider komponent defineerib mänguobjekti kuju, eesmärgiga teha võimalikuks füüsilised kokkupõrked, mänguobjektide puhul. Kõige vähem ressurssi võtvad colliderid on primitiivsed. Primitiivseteks loetakse Box, Sphere ja Capsule collidereid. Kuid on olukordi kus primitiivsete collideritega, ei aja asja ära. Sellisel juhul on võimalik luua mesh colliderid, mille puhul 3D mudelite kuju jäljendatakse täpselt. Need on aga väga ressursimahukad ning seega peaks neid võimalikult vähe kasutama. Samuti on võimalik colliderid seadistada kui triggereid. Sellisel juhul ei toimu füüsiliselt mingit takistamist, kuid kutsutakse välja funktsioon OnTriggerEnter, see järel jääb trigger infot edastama kasutades OnTriggerStay ja lõpuks kui trigger väljub oma kokkupõrkest kutsutakse OnTriggerExit. Siis on olemas Rigidbody colliderid, mille korral füüsika arvutatakse täies mahus objektile ning teda mõjuvad erinevad füüsikalised jõud. Rigibodyl on olemas ka kinematic rigidbody collider, mis puhul saab liigutada objekti, millel on rigidbody ja collider küljes. Erinevus aga tavalise rigidbody collideriga on et mänguobjekt ei reageeri kokkupõrgetele ning teda ei mõjuta füüsika, seega saab seda liigutada skriptist. [31]

Loodud projektis kasutatakse collidereid väga laialdaselt ja nende eri kujudel. Mängija liigutamiseks ja mitte seinadest läbimiseks, on kasutatud capsule colliderit, kuna mängija ei ole loodud eraldi mudelina, vaid on kasutatud capsule mänguobjekti. Mängija virtuaalsete sammude genereerimiseks, on kasutatud box colliderit, mis on pandud kui trigger. Seega füüsiliselt ei takistata see liikumist, aga kui trigger reageerib drooniga, siis kasutatakse seda infot drooni mõjutamiseks. Drooni skänneri jaoks on kasutatud mesh colliderit, mis on convex collider, kuna muidu ei saa seda kasutada kui triggerit ning ta eesmärk on mängija avastamine. Kogu mängumaailma kõikidele objektidele on lisatud

collider, et takistada mängija liikumist ning ka pudeliga reageerimiseks. Droonile on lisatud ka rigidbody, koos capsule collideriga selleks, et nad läbi maailma ja üksteise ei saaks minna. Pudelil endal on kaks trigger colliderit ning lisaks ka rigidbody. Pudelil on trigger mängijaga reageerimiseks, et saaks pudelit ülesse võtta kergemalt ning ei peaks väga täpselt pudeli peale kõndima. Pudel on alguses rigidbody kinematic selleks, et ta läbi mapi ei kukuks ning ülesvõttes ei reageeriks mängijaga. Peale viskamist võetakse kinematic maha selleks, et simuleerida füüsikaga pudeli lendamist ja talle mõjuvaid jõude. Vangikongidele on lisatud trigger tagiga „Cell“, mille puhul eemaldatakse mängija virtuaalsete sammude genereerimine selleks, et mängijal oleks skautimise koht vangikongist, mida ta saab kasutada planeerimaks oma järgnevat tegevust, turvalisest kohast kuna droon ei kõrvalda mängijat vangikongist.

6.4.9 Rigidbody

Rigidbody võimaldab füüsika mõjumise mänguobjektile. Rigidbody on võimalik mõjutada jõu ja pöördemomendiga ning on vajalik, et mänguobjektid käituksid realistlikult. Kui me soovime, et mänguobjekti mõjutaks füüsika, siis ta peab sisaldama rigidbody komponenti. Tavaliselt ei tohiks korraga mõjutada mänguobjekti parameetreid (positsioon, pööre ja suurus), skript ning rigidbody. Kasutama peaks objekti mõjutamiseks ühte või teist. [32] Rigidbody kasutame pudeli füüsikaga mõjutamiseks ning tehisintellektiga kokkupõrgete jaoks, mis on vajalik, et nad üksteisest läbi ei läheks.

6.4.10 Character controller

Character controllerit kasutatakse peamiselt first-person ja third-person mängija kontrollimiseks, mis ei kasuta rigidbody füüsikat. Sellega on võimalik seadistada controlleri height ja radius selleks, et see vastaks mängija mõõtmetega. Skin width on ka lisaks eelnevale, väga tähtis asi mida seadistada. Skin width lubab objektidel natukene siseneda controllerisse, kuid eemaldab ära värisemise ja mängija kinni jäämise. [33] Töö raames on mängija liigutamiseks, kasutatud character controller komponenti. Selleks, et saavutada kükitamine, on modifitseeritud mängija suurust ja sellega ka character controlleri suurust.

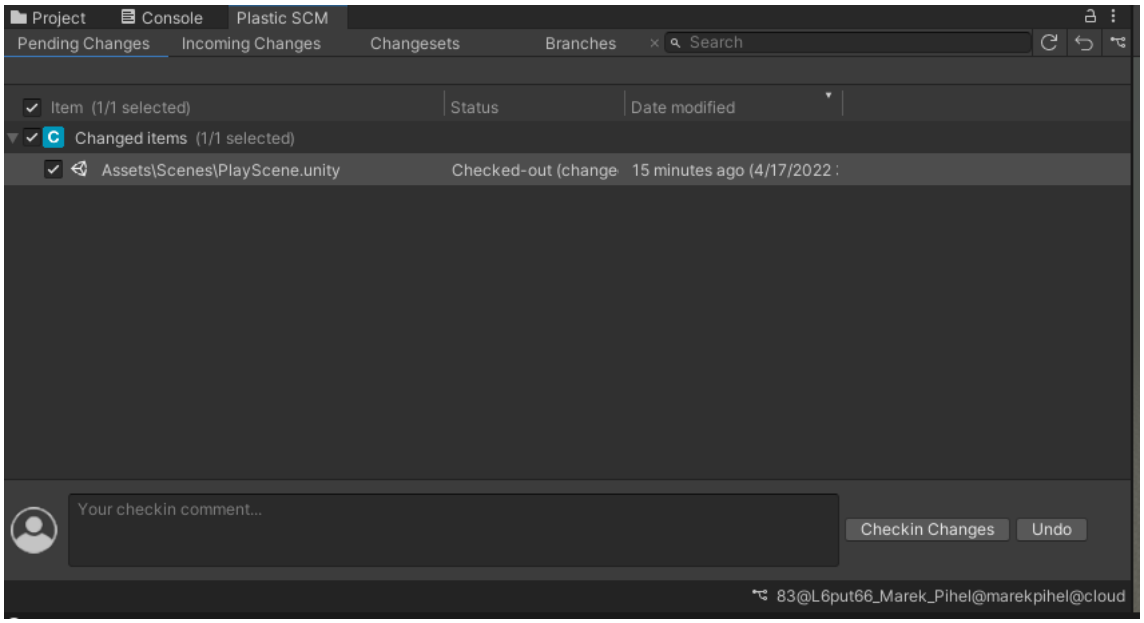
6.4.11 Prefab

Unitys on võimalik kasutada prefab süsteemi, mille korral on võimalik salvestada terviklikke mänguobjekte, koos kogu tema komponentide, väärtuste ja alam

mänguobjektidega, eesmärgiga teha see kergesti taaskasutatavaks. Prefabe on mõistlik kasutada siis, kui sul on vaja kasutada sama mänguobjekti mitmes eri kohas ning teistes stseenides. See on hea selleks, et kui muuta prefabi, siis uuenevad samuti kõik objektid, mis kasutasid seda. Seega saab efektiivselt viia sisse muudatusi, kogu projekti ulatuses. Lisaks on prefabe mõistlik kasutada, kui on vaja mängu objekte dünaamiliselt maailma luua. Samuti on prefabide kasutamine mõistlik tasemeloomisel kasutatavate mänguobjektide jaoks. [34] Selle projekti raames on loodud ning kasutatud järgnevaid prefabe: Bottle, BottleBreakinSound, Cell, Crate, DroneSuspicionManager, KillerDrone, Player ning SecondaryCell. Neid kõiki selleks, et oleks lihtne ja mugav taaskasutada erinevaid mänguobjekte, millel on sama funktsionaalsus.

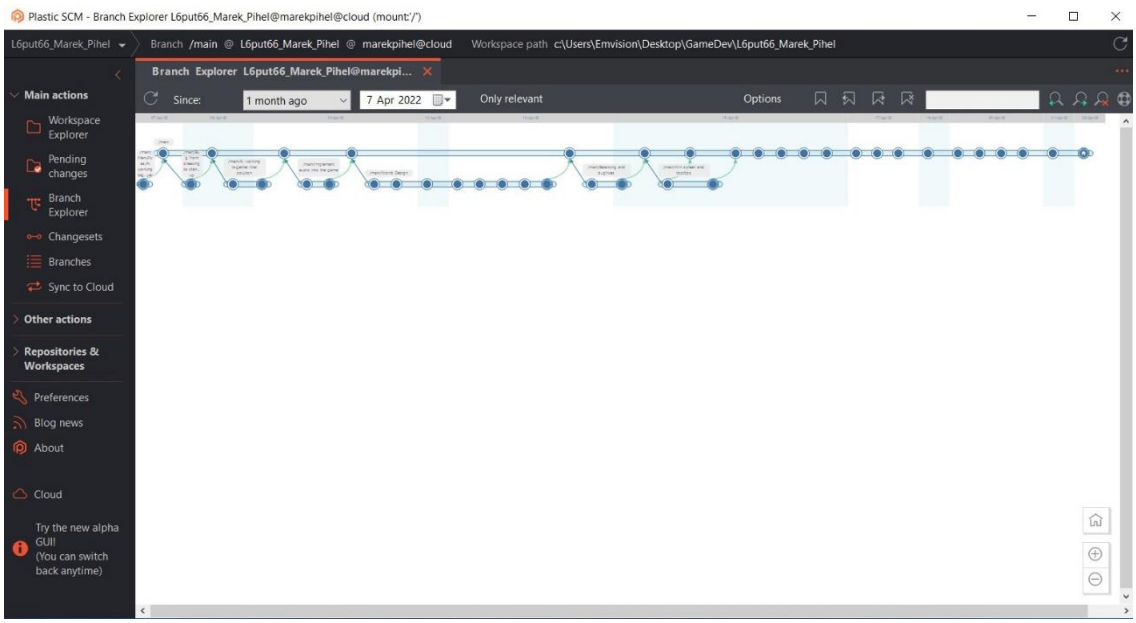
6.5 Plastic SCM verisoonihalduse tutvustus ja kasutatud funktsionaalsus

Vaadates Joonis 6’te näeme vaadet, mis Unity seest avanes Plastic SCM tarkvara kasutamiseks. Üleval on aknad eri toimingute jaoks, nagu praegu muutmist ootavad asjad, mis on erinevad sellest branchist kus me asume, selle kõrval on sissetulevate muudatuste aken, mis näitab muudatusi, mis on erinevad sinu ja branchi vahel. Seega kui vahetad branchi kusagile, kus on teistsugused failid, siis seal all näitab mis faile pead alla laadima, et lokaalne seis oleks sama, nagu branch kuhu läksid. Selle kõrvalt leiame changesettide akna, kus on näha kõik muudatused mis projektis on tehtud. Ning selle kõrvalt branchide akna, kus saab erinevate branchide vahel vahetada. Keskel on näha muudatused, mis on lokaalselt tehtud, mida pole veel ülesse lükatud branchi ning selle all, avaneb aken kommentaari lisamiseks, muudatuse branchi ülesse lükkamise jaoks. Checkin changes lükkab muudatused ülesse ning undo eemaldab tehtud muudatused. Lisaks sellele annab paremal kõige all kirje märku sellest, kus branchi või changeseti peal sa praegu asud.



Joonis 6. Plastic SCM Unity vaade.

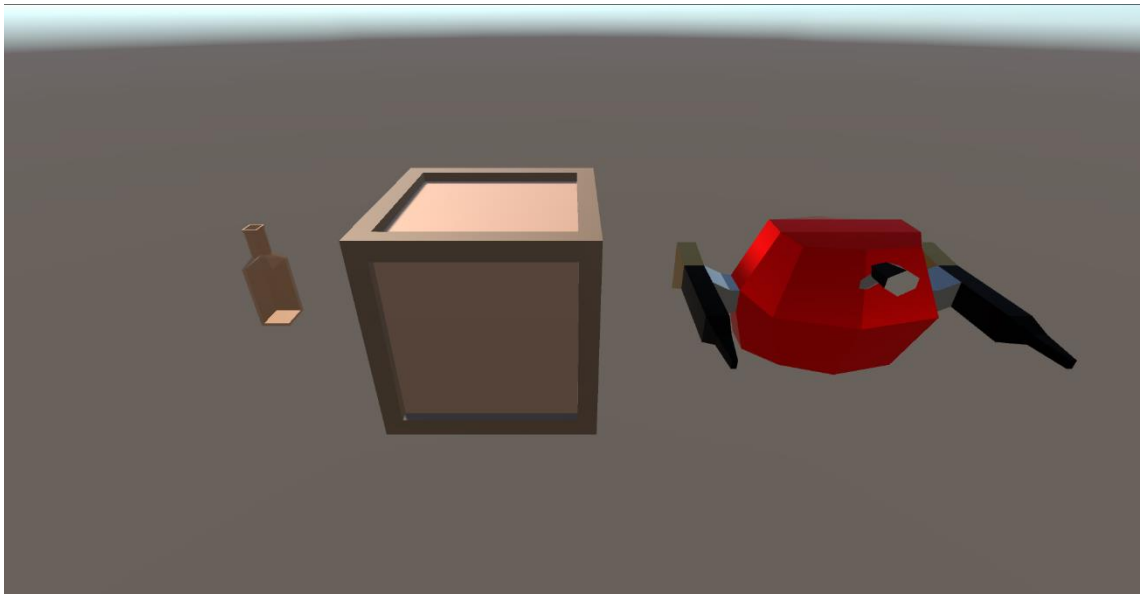
Joonis 7'l on näha vaade, mis avanes Plastic SCM sees, kus toimus branchide mergemine ja sai ülevaadet selle kohta, milline on projekti branchindus üldisemalt. Programmis endast, toimus ka erinevatest branchidest maini tagasi mergemine, mis polnud Unity alt võimalik. Kuna selle töö raames ei olnud tegu, laialdaselt kasutusele mineva mänguga, vaid kohalikult arendatavaga, siis ei ole loodud eraldi development ja live branche.



Joonis 7. Plastic SCM tarkvara vaade programmis sees.

6.6 Mudelite loomine

Blenderis loodi kolm erinevat mudelit (Joonis 8) mis olid vajalikud mängu jaoks ning üks lisa mis oli vajalik skänneri jaoks. Esimene neist oli pudel nimega „Bottle“, mis oli vaja tehisintellekti tähelepanu eemale tõmbamiseks. Teine mudel mis sai loodud, oli tehisintellekti enda mudel ehk droon nimega „KillerDrone“, millel on mõlemal küljel relvastus mängija kõrvaldamiseks. Kolmas mudel oli kast nimega „Crate“, mille taha mängijal oleks võimalik varjuda tehisintellekti eest ning mida saaks kasutada osana maailma disainil. Lisaks on loodud „Scanner“, mudel mida oli vajalik skänneri avastamise ala loomiseks. Tekstuuride värvimiseks kasutati meetodit mille korral, UV mapi osad, kahandatakse üheks punktiks ning seejärel viiakse punkt värvitud piksli peale, et kogu pind korraga värvida. See on väga hea ja kiire lahendus, tekstuuride loomiseks oma mudelitele.



Joonis 8. Loodud mudelid mängu tarvis

6.7 Taust ja missioon

Peale voolukatkestust vanglas on kõik uksed avanenud. Moodsas vanglas aga selliste juhtumite puhul, rakendatakse põgenemise takistamiseks roboteid, mis patrullivad mööda koridore ning kõrvaldavad kõik kes üritavad sellel ajal põgeneda kompleksist. Mängija eesmärgiks, on oma oskusi kasutades põgeneda sellest kompleksist. Selleks peab ta kasutama hiilimist ning oma oskusi ja tarkust, et vajadusel juhatada robotite tähelepanu mingisse kohta, et neist mööda saada.

6.8 Taseme disain

Mängus oleva mängumaailma loomisel, on arvestatud peatükis 5 toodud nõuetega, mis puudutavad maailma loomist. Põhiline millega mängumaailma loomisel tuli arvestada, oli see, et kõik olukorrad kus mängija ennast leidis, oleksid mängija kasuks. Seda on vaja selleks, et säiliks totaalne hiilimise stiil, mille korral mängijal pole ühegi kokkupuute korral, võimalik läbi kukkuda (pt 5 punktid 17 ja 18). Samuti oli mängijal vaja eelist robotite ees. Iga kokkupõrke korral on antud mängijale erinevaid viise, kuidas tollele kokkupuutele läheneda. Osade kokkupuudete korral, oli võimalik läbida ainult kindlat moodi see olukord, ning osade puhul oli võimalus lahendada probleeme loovalt ning võimaldatud erinevalt nendest kohtadest läbi saada. Lisaks on tasemete disainil arvestatud R.Smith'i välja toodud faktiga, et tasemete loomisel, peaks jääma mulje kindlalt kaitstud alast, mille korral mängija peab kasutama oma oskusi, et ekspluateerida auke turvalisuses (peatükk 3). Kokkupuuted olid jagatud maailma loomisel loogilisteks sektsioonideks. Nagu on näha Joonis 5 pealt vasakult „Hierarchy" akna alt, siis võib sealt näha mängumaailma ülesehitamisel kasutatud sektsioone, millest igale on antud loogiline nimetus. Selliselt arendades, on võimalik luua erinevaid sektsioonid kaardile ning hiljem need oma vahel ühendada, vastavalt mängu eesmärgile või sujuvusele.

6.9 Mängija mehaanika loomine

Mängija funktsionaalsusele püstitati nõuded peatükis 5 punktides 12–16. Selle projekti raames, on mängijale analüüsitud mängumehaanika nõuded, kirja pandud lähtuvalt stealth žanrist, kuid mõned nõuded kehtivad üldisemalt mängude kohta ning ei ole eksklusiivsed stealth žanrile. Üldiselt on mängijal vaja liikuda, see kehtib päris paljude mängude kohta, kuid mitte kõikide. On olemas mängud, kus mängija liikumine pole ilmtingimata vajalik, kuid selle projekti raames oli vaja see luua. Lisaks sellele on enamus mängudes väga tähtis anda mängijale selge eesmärk, mida ta tegema peab. Selle projekti raames loodud mängija loogika, on žanri ja selle mängu põhine kuid mida, võib ka teatud olukordades laiendada teistele žanritele. Mängijale sai loodud võimalus asju maast korjata ja neid loopida, seda eesmärgiga tehisintellekti tähelepanu juhtida. Lisaks sellele sai ka rõhku pööratud faktile, et mängust ei saaks märulile keskenduv mäng. Selleks oli vajalik kinni pidada faktist, et tehisintellekt ei tohiks olla kõrvaldatav. Sellega säilib ka R.Smith'i poolt mainitud faktor, kus stealth žanri hiilimisele keskenduvates mängudes, on mängija läbikukkumise kriteeriumiks mängija avastamine [1].

6.10 Tehisintellekti väljatöötamine ja valmistamine

Tehisintellekti loomisele keskenduvad, nõuded leiab peatükist 5 punktidest 8–11. Mängu arendusel kulus tehisintellektile suur osa ajast. Probleem oli keeruline ning välja oli vaja töötada palju erinevaid aspekte, tehisintellekti käitumisega seoses. Tehisintellekti vaateväljast ülevaate andmiseks, kasutati valgusele baseeruvat tehnikat. Alustuseks tuli luua tehisintellektile võimalus mööda mängu maailma ringi navigeerida. Lihtsustamise huvides ei hakatud ise arendama navigeerimise ja teekonna otsingu jaoks eraldi süsteemi, vaid kasutasin siinkohal navigeerimiseks Unity mootoris olevat navigatsiooni süsteemi, millest juttu täpsemalt peatükis 6.4.7. Küll aga oli vaja töötada välja süsteem, millega saame anda „Nav Mesh Agent“ komponendile punkte kuhu robot peaks minema. Tehisintellekti liikumise süsteemi arendamiseks, kasutasime lõpliku olekumasinat, mille korral lõime eraldi olekud mis olid loodud kasutades abstraktset klassi. Sellega sai ühtsustada tehisintellekti käitumise koodis oleku puhul tehtavat toimingut. Olekud jagasin loogilisteks osadeks, mis olid seotud minu mänguga, kuid üldisemalt leiab nad ka stealth žanris olevate olekutega. Oma mängu jaoks sai loodud 3 olekut. Patrull, uurimise ja otsimise olekud. Patrull olekute puhul, on üks või mitu punkti, mille vahel tehisintellekt ringi liigub. Üksiku punkti puhul, on tehisintellekt postil ning ei liigu seega on vaja teada ära sundida sealt selleks, et saavutada mängijana oma eesmärk. Uurimise olekusse satub tehisintellekt heli kuulmisel. Sellisel juhul valib tehisintellekt punkti, heli põhjustanud punkti juures ning uurib ala natuke aega, skännides nii vasakule kui paremale. Kui robot ei leia selle punkti juures midagi huvitavat, läheb ta tagasi oma algsesse patrull olekusse. Otsingu olekusse lähevad kõik robotid korraga, sest selle puhul on, liiga palju müra tehtud lühikese aja jooksul. Otsingu olekusse sattudes on mängijal väga raske mängu läbida, kuna robotid otsivad terve aeg mängijat taga ning käituvad väga ettearvamatult. Punktis 8 käsitletud nõue, on mängu loodud erinevate simuleeritud helide näol.

Kuna Unity ei võimalda mitut audio listener’i korraga stseeni panna [35], siis tuli leidlikult läheneda helile mängus. Nimelt loodi virtuaalse heli süsteem. Nii pudeli kokkupõrge mängumaailmaga kui ka mängija liikumine, põhjustab virtuaalset heli, kaasa arvatud tavalisele helile mida mängija kuuleb, millele tehisintellekt reageerib. Vangikongis olles tehisintellekt mängija sammudele ei reageeri, kuid pudeli purunemisele reageerib endiselt, kuigi mängijat ei kõrvaldata vangikongist. Oleku masina ja nav mesh agent’i koosmõjul, on ka täidetud punktis 9 toodud nõue, mille puhul peaks tehisintellekt saama hakkama, talle püstitatud olukordadega. Loodud on ka mängu

kaotuseks vajalik loogika. Mängija avastamisel kõrvaldab tehisintellekt mängija, peale mida avaneb vastav UI aken. Sellega antakse mängijale teada tema kaotusest ning ülejäänud tegevus peatub. Mängija saab seejärel minna tagasi ainult menüüsse.

Tehisintellekti koostööle keskenduv osa selle projekti raames, sai defineeritud järgnevalt. Tehisintellekt peab suutma omavahel jagada seda, kui palju on kokku heli kuuldud nii lühi- kui pikaajaliselt. Selle vajalikkus seisneb selles, et mida rohkem heli nad kuulevad, seda suuremast raadiusest reageerivad tehisintellektist vastased ning sellega seoses, on järjest rohkem abiväge tulemas. Täiendava tegevusena, panustavad kõik robotid lühiajalise heli kuulmise süsteemi, mille korral kui mängija genereerib liiga palju heli, lühikese aja jooksul, lähevad kõik robotid korraga otsingu olekusse. Selle tulemusel muutub mängu läbimine palju raskemaks. See soodustab omakorda hiilimisele keskenduvat stiili. Lisaks on loodud süsteem, mis annab teatud raadiuses teada tehisintellektidele kes peaksid reageerima, et seda punkti on vaja uurida. Seega teevad robotid koostööd, selle nimel, et mängijat otsida ja panustavad ühisesse info ruumi.

6.11 Heli lisamine

Kuigi eraldi nõudena seda küll välja ei toodud, oli vajalik heli lisamine mängu, eesmärgiga anda tagasisidet mängijale. Sellega seoses sai mängu lisatud kolm erinevat heli. Sammumisel genereeritud heli, pudeli purunemine ning tehisintellekti poolt genereeritud heli, mis aitab mängijal koguda informatsiooni, ka seinte ja nurkade tagant, tehisintellekti asukoha kohta. See aitab omakorda mängijal teha otsuseid selle kohta, millal ja kuidas liikuda ning kus asetseb parajasti tehisintellekt, kes üritab teda kõrvaldada.

Kõikide helifailide autorid on lasknud need helid välja „Creative Commons 0“ litsentsi all ning seega on helifailide modifitseerimine enda tarbeks lubatud ning viitamine pole otseselt vajalik. Kuigi puudub vajadus selle järele, on viited algsetele heli failidele selle peatüki lõpus. Heli faile sai natuke muudetud selle projekti tarbeks, ühtlustatud sai pudeli lõhkumisel tekkiv heli, kuna algselt oli see mängu pannes väga vaikne. Lisaks sellele sai drooni liikumisel tekkiv heli muudetud selleks, et see läheks natuke paremini kokku drooni välimusega ja oleks pikaajalisel kuulamisel meeldivam. Ning viimaseks sai sammude heli failis, lõigatud ära algne ootuse moment ja lõpu tühimiku osa selleks, et

seda heli kasutada paremini nii hiilimisel kui ka spurtimise korral, kus sammude heli kogus on erinev. [36] [37] [38]

7 Testimine

Mängu testimiseks sai kasutatud selle projekti raames, mängides testimise meetodit. Selleks, et mängides testimist saaks läbi viia, oli vaja ehitada mäng valmis. Õnneks võimaldab Unity peale arendust, väga lihtsalt mängu valmis ehitada, kasutades sisse ehitatud „Build“ funktsionaalsust. Mängu testijatest 3 olid Windowsi platvormi põhised ning 1 testija testis MacOS peal. Arendajana testisin ise kogu arendus protsessi vältel Windows platvormil. Mängu ehitamiseks MacOSi platvormile oli Unity mängumootoris ainult vaja vahetada platvorm, millele ta ehitab kui „Build“ funktsionaalsust kasutati. Mängides testimine on mängude puhul väga hea meetod toote testimiseks, kuna mängud iseenesest on väga interaktiivsed ning seega ainult koodi testimisega, ei ole võimalik välistada vigasid, mis võivad tekkida mängides. Mängides tekkivad vead, võivad olla seotud füüsikaga või mingite komponentide omavahelise interaktsiooniga ja sellepärast ei pruugi traditsioonilised testimise meetodid neid avastada. Eelnevat laiendades tulenevalt faktist, et projekti koodis on palju funktsioone, mis käsitlevad mängu jooksmise ajal loogikat ning ei tagasta midagi, oli väga raske leida kohta, kus rakendada traditsioonilist tarkvara testimist, seega viisime läbi ainult mängides testimise.

Mängides testimise puhul rakendati erinevaid inimesi, lisaks arendaja enda poolsele testimisele. Kokkuvõttes saab öelda, et testimise metodoloogia valiti õige, kuna tagasiside mängides testimisest oli väga hea ja võimaldas veel projekti lõpu staadiumis, avastada vigu mida iseseisvalt ei suudetud avastada. Sellega seoses viidi sisse ka tagasisidest avastatud vigade parandused ja see võimaldas väga iteratiivselt, parandada mängu kogemust mängija jaoks. Samas avastati ka vigasid, mis olid tekkinud mingite juhtumitega arvestamata jätmisel. Mängu testimisel enda projekti raames, testisid mängu kokku 5 erinevat inimest, millest üks oli projekti arendaja. Samuti on lisas 2 toodud välja küsimuste vastused, mida paluti täita testijatel, seoses mängu enda nõuetega. Seda selleks, et saaks hinnangut ka nõuetele vastavuse kohta, teistelt osapooltelt. Nagu võib näha tagasisidest, siis said projekti nõuded täidetud. Siinkohal tuleb arvestada mõningate erisustega, kuna iga inimese jaoks on mängimis kogemus ja lähenemine erinev, võib sellest tulenevalt olla tagasisides erisusi.

8 Nõuetele vastavus

Peatükis 5 toodi välja nõuded enda mängule ning peatükis 5.1 analüüsiti ka seda, kuidas saaks analüüsida nende täitmist. Juba arenduse alguse faasis, kui sai Trellosse ülesanded lisatud, lähtuti projektile püstitatud nõuetest ja kogu arendatud funktsionaalsus pandi paika neid järgides. Seega vastas loodud mäng sellele, mis nõuded püstitati ning mida üritati saavutada. Nimelt loodi stealth žanrile vastava klassifikatsiooniga mäng. Seda saab ka kinnitada, vaadates mängides testimise tagasisidet, mis näitab, et nõuded 6–18 said ilusti täidetud ning kõik vastas ootustele. Punktides 1–5 toodud nõuded programmide kasutusele, said seatud varasema analüüsi tulemusel. See tugines faktile, et kõik tarkvara on väga võimas ning ka populaarne, boonusena oli kogu kasutatud tarkvara tasuta kättesaadav ja sobis ideaalselt sellise projekti loomiseks. Kõik programmid töötasid väga hästi koos ning seega oli nende kasutamine lihtne.

9 Kokkuvõte

Töö põhieesmärk, milleks oli anda edasi komplektne pagas teadmisi mängu arenduse kohta, läbi stealth žanri, sai täidetud. Kõik vajalikud probleemid leidsid lahenduse ning said ka arendatud mängus kajastatud, demonstreerimaks lahendusi mis töö jooksul esitati. Arenduse protsessi käigus saime teada, et on olemas lahendused versioonihalduseks, ka video mängu tööstuses, millega on võimalik salvestada suuri faile, mis ei ole ilmtingimata ainult koodi failid. Lisaks sellele saime teada, et mitte ainult ei ole olemas selliseid lahendusi, vaid ka Unity puhul omab sama firma nii mängumootorit kui ka versioonihaldus tarkvara. Seega on põimitud väga hästi mängumootorisse versioonihaldus, mis teeb selle kasutamine lihtsaks ning mugavaks. Mängumootoriga seoses, vaatasime nende populaarsust ning käsitlesime punkte, mis mõjutasid Unityga arendamise kasuks otsustamist. Mängumootori kasutamine kiirendas ja lihtsustas oluliselt arenduse protsessi, võimaldades kiiresti luua mängu loogikat, läbi stseenide ja skriptide ning testida mängu koheselt, kasutades Unity play funktsionaalsust. Stealth žanri puhul, on tähtis alati meeles pidada mängija läbikukkumise kriteeriume ning see defineerib ka osaliselt ära stealth žanri. Käsitlesime ka R.Smith'i poolt pakutud kriteeriume, kuidas klassifitseerida mängu kui tõelist stealth mängu. Sellega seoses vaatasime ka üle probleemid, nagu mängija liikumine, tehisintellekti informatsiooni hulk mida ta omab mängu kohta ning selle, kuidas arendada tehisintellekti kasutades lõpliku olekumasinat. Mis lihtsustas tema haldust ning võimaldas käitumise abstraheerimist. Lõpuks tuli juttu mängu mängides testimisest, kus käsitlesime milleks seda kasutatakse ning mis on selle eesmärk. Samuti käsitlesime võimalusi mäng ehitada erinevatele platvormidel, muutes ainult seda mis platvormile mäng ehitati. Käsitlesime töö kohta saadud tagasisidet, mis tuli juba päriselt läbiviidud testimisest ja selle vajalikkusest ning süvenesime ka viimaste muudatuste sisse, mis viidi läbi seoses tagasisidest tulnud vigade avastamisega.

Kasutatud kirjandus

[1] Y. Khatib, „Examining the Essentials of Stealth Game Design,“ 6 Juuni 2014. [Võrgumaterjal]. Available: https://www.gamecareerguide.com/features/1318/examining_the_essentials_of_.php?print=1. [Kasutatud 28 Veebruar 2022].

[2] Guinessi rekordite raamat, „First stealth videogame | Guinness World Records,“ 1981. [Võrgumaterjal]. Available: https://www.guinnessworldrecords.com/world-records/first-console-game-to-use-stealth. [Kasutatud 28 Veebruar 2022].

[3] A. GILYADOV, „In the Shadows: A Brief History Of Stealth Games,“ 20 Oktoober 2015. [Võrgumaterjal]. Available: https://www.cgmagonline.com/articles/features/in-the-shadows-a-brief-history-of-stealth-games/. [Kasutatud 28 Veebruar 2022].

[4] Shane_Patterson, „The sneaky history of stealth games | GamesRadar+,“ 3 Veebruar 2009. [Võrgumaterjal]. Available: https://www.gamesradar.com/the-sneaky-history-of-stealth-games/. [Kasutatud 28 Veebruar 2022].

[5] Guinessi rekordite raamat, „First 3D stealth videogame on PC | Guinness World Records,“ Juuni 2011. [Võrgumaterjal]. Available: https://www.guinnessworldrecords.com/world-records/98899-first-3d-stealth-videogame%c2%a0on-pc. [Kasutatud 6 Märts 2022].

[6] Guinessi rekordite raamat, „Best-selling stealth videogame | Guinness World Records,“ [Võrgumaterjal]. Available: https://www.guinnessworldrecords.com/world-records/86553-best-selling-stealth-videogame. [Kasutatud 6 Märts 2022].

[7] Guinessi rekordite raamat, „First stealth videogame to feature collective Artificial Intelligence | Guinness World Records,“ [Võrgumaterjal]. Available: https://www.guinnessworldrecords.com/world-records/first-stealth-game-to-feature-collective-artificial-intelligence. [Kasutatud 22 Märts 2022].

[8] M. Milton-Jefferies, „In Praise of Assassin’s Creed Unity’s Stealth Mechanics - JumpCut PLAY,“ 17 Jaanuar 2021. [Võrgumaterjal]. Available: https://play.jumpcutonline.co.uk/2021/01/17/in-praise-of-assassins-creed-unitys-stealth-mechanics/. [Kasutatud 6 Märts 2022].

[9] R. Smith, Esitaja, *Level Building for Stealth Gameplay.* [Helisalvestis]. Game Developers Conference. 2006.

[10] T. Francis, „How stealth games control information | Rock Paper Shotgun,“ 3 Aprill 2018. [Võrgumaterjal]. Available: https://www.rockpapershotgun.com/what-works-and-why-unfair-intel-in-stealth-games. [Kasutatud 13 Märts 2022].

[11] T. Leonard, „Building an AI Sensory System: Examining The Design of Thief: The Dark Project,“ 7 Märts 2003. [Võrgumaterjal]. Available:

https://www.gamasutra.com/view/feature/2888/building_an_ai_sensory_system_.php?print=1. [Kasutatud 18 Märts 2022].

[12] S. J. Rüstern, „Kahemõõtmeline mobiilimäng Unity mängumootori abil," Tallinn, 2014.

[13] S. Saarsen, „3D võrgumängu arendus Unity mängumootoril," Tallinn, 2016.

[14] K. Romulus, „3D tulistamis- ellujäämismängu arendus Unity mängumootoril," Tallinn, 2019.

[15] N. Koikov, „Virtuaalreaalsusmängu arendus Unity mängumootoril," Tallinn, 2018.

[16] P. S. Luaces, „Plastic SCM vs Git — 2018 edition | by Pablo Santos Luaces | Medium," 15 Märts 2018. [Võrgumaterjal]. Available: https://medium.com/@psluaces/plasticscm-vs-git-c17934fad7ed. [Kasutatud 18 Märts 2022].

[17] MVC Staff, „Unity acquires Plastic SCM developer Codice Software - Development News - MCV/DEVELOP," 17 August 2020. [Võrgumaterjal]. Available: https://www.mcvuk.com/development-news/unity-acquires-plastic-scm-developer-codice-software/. [Kasutatud 2022 Märts 18].

[18] A. Andrade, „Game engines: a survey," *EAI Endorsed Transactions on Serious Games*, kd. 2, nr 6, p. 2, 05 November 2015.

[19] J. Brodkin, „How Unity3D Became a Game-Development Beast," 3 Juuni 2013. [Võrgumaterjal]. Available: https://insights.dice.com/2013/06/03/how-unity3d-become-a-game-development-beast/. [Kasutatud 20 Märts 2022].

[20] Unity Technologies, 19 Märts 2018. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. [Kasutatud 24 Märts 2022].

[21] Unity Technologies, „Unity Real-Time Development Platform | 3D, 2D VR & AR Engine," [Võrgumaterjal]. Available: https://unity.com/. [Kasutatud 21 Aprill 2022].

[22] Unity Technologies, „Unity - Manual: The Scene view," 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/UsingTheSceneView.html. [Kasutatud 17 Aprill 2022].

[23] Unity Technologies, „Unity UI: Unity User Interface | Unity UI | 1.0.0," 6 Oktoober 2020. [Võrgumaterjal]. Available: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html. [Kasutatud 17 Aprill 2022].

[24] Unity Technologies, „Unity - Manual: Important Classes - GameObject," 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/class-GameObject.html. [Kasutatud 17 Aprill 2022].

[25] Unity Technologies, „Quick start guide | Input System | 1.0.2," 22 Jaanuar 2021. [Võrgumaterjal]. Available: https://docs.unity3d.com/Packages/com.unity.inputsystem@1.0/manual/QuickStartGuide.html#getting-input-indirectly-through-an-input-action. [Kasutatud 19 Aprill 2022].

[26] Unity Technologies, „Unity - Manual: Scripting," 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/ScriptingSection.html. [Kasutatud 17 Aprill 2022].

[27] Unity Technologies, „Unity - Scripting API: SerializeField,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/ScriptRefeunityrence/SerializeField.html. [Kasutatud 17 Aprill 2022].

[28] Unity Technologies, „Unity - Manual: Types of light,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Lighting.html. [Kasutatud 17 Aprill 2022].

[29] Unity Technologies, „Unity - Manual: Building a NavMesh,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html. [Kasutatud 17 Aprill 2022].

[30] Unity Technologies, „Unity - Manual: Creating a NavMesh Agent,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/nav-CreateNavMeshAgent.html. [Kasutatud 17 Aprill 2022].

[31] Unity Technologies, „Unity - Manual: Colliders,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html. [Kasutatud 17 Aprill 2022].

[32] Unity Technologies, „Unity - Manual: Rigidbody,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/class-Rigidbody.html. [Kasutatud 17 Aprill 2022].

[33] Unity Technologies, „Unity - Manual: Character Controller,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/class-CharacterController.html. [Kasutatud 17 Aprill 2022].

[34] Unity Technologies, „Unity - Manual: Prefabs,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/Prefabs.html. [Kasutatud 17 Aprill 2022].

[35] Unity Technologies, „Unity - Manual: Audio Listener,“ 9 Aprill 2022. [Võrgumaterjal]. Available: https://docs.unity3d.com/Manual/class-AudioListener.html. [Kasutatud 17 Aprill 2022].

[36] mgamabile, „Freesound - "smashing glass.wav" by mgamabile,“ 20 September 2018. [Võrgumaterjal]. Available: https://freesound.org/people/mgamabile/sounds/440773/. [Kasutatud 17 Aprill 2022].

[37] miguelab1998, „Freesound - "17.Step.wav" by miguelab1998,“ 6 November 2017. [Võrgumaterjal]. Available: https://freesound.org/people/miguelab1998/sounds/408564/. [Kasutatud 17 Aprill 2022].

[38] hello_flowers, „Freesound - "HF Computer Hum C.wAv" by hello_flowers,“ 20 Märts 2007. [Võrgumaterjal]. Available: https://freesound.org/people/hello_flowers/sounds/32522/. [Kasutatud 17 Aprill 2022].

[39] Pluralsight, „Understanding UVs - Love Them or Hate Them, They're Essential to Know | Pluralsight,“ 19 Jaanuar 2014. [Võrgumaterjal]. Available: https://www.pluralsight.com/blog/film-games/understanding-uvs-love-them-or-hate-them-theyre-essential-to-know. [Kasutatud 24 Märts 2022].

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Mina, Marek Pihel

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Stealth põhise videomängu arendus koos tehisintellekti loomisega kasutades Unity mängumootorit“, mille juhendaja on Inna Švartsman
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

22.02.2022

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 – Mängides testimise tagasiside testijatelt

Taavi Metsla –

„1) Kas mängu oli võimalik läbida lihtsalt joostes või pidid kusagil kasutama hiilimist ja oskusi et AI mööda saada?

Pidin kasutama kõiki olemas olevaid meetmeid (ka algset lae/katuse puudumist.)

2) Kas tehisintellekt reageeris helile nagu pudeli katki minek ja sammud ?

Jah, vahest isegi liiga hästi.

3) Kas vahele jäädes lõppes mäng ära või oli võimalus edasi kuidagi mängida?

Mäng lõppes ära ja viskas otse menüüsse, ühe korra uuesti alustades ei olnud võimalik ennem liikuda kui ESC'i ei vajutanud.

4) Kas mitme AI läheduses reageerisid mõlemad ja hakkasid otsima koos mängijat või tegutsesid droonid üksi alati ?

Reageerisid kõik kes läheduses olid ja hakkasid otsima.

5) Kas mängijana said ringi liikuda?

Jah

6) Kas mängijana said asju maast korjata ja loopida ?

Ainult pudeleid, tahaks kaste ka loopida.

7) Kas mängu eesmärk oli selge?

Jah, vältima hõljuvaid roboteid ja neist mööda hiilima.

8) Kas said kuidagi kõrvaldada AId?

Ei ole veel viisi leidnud, pudeliga loopides ei saanud.

9) Kas mingis olukorras ei olnud võimalik AId üle kavaldada ?

Sai kõikidest kohtadest mööda.

10) Kas oli olukordi kus maailmas edasi liikumine oli võimatu mingil põhjusel?

Võimatu ei olnud, alati leidis lahenduse.

11) Kas mängus oli suuremaid vigasid? (BUGe. Läbi seina sai joosta või kuidagi muud moodi mängu katki teha mis ei olnud selleks ette nähtud)

Viimaste läbimängimistega BUGe ei täheldanud, kõik paistis toimivat nii nagu ettenähtud.“

Ceisi Peik –

„1) Kas mängu oli võimalik läbida lihtsalt joostes või pidid kusagil kasutama hiilimist ja oskusi et AI mööda saada?

Mängu ei olnud võimalik läbida lihtsalt joostes. Kuna jooksmine tegi kõvemat häält tuvastas AI mängija lihtsamini. Kohtades, kus pidi AI'st lähemalt mööduma pidi kasutama hiilimist, et ta sammude järgi mängijat koheselt ära ei tunneks.

2) Kas tehisintellekt reageeris helile nagu pudeli katki minek ja sammud?

Jah reageeris.

3) Kas vahele jäädes lõppes mäng ära või oli võimalus edasi kuidagi mängida?

Mängu oli võimalik menüü kaudu uuesti alustada.

4) Kas mitme AI läheduses reageerisid mõlemad ja hakkasid otsima koos mängijat või tegutsesid droonid üksi alati?

Ei pannud tähele, et oleks koos mängijat otsinud.

5) Kas mängijana said ringi liikuda?

Jah.

6) Kas mängijana said asju maast korjata ja loopida?

Jah, pudeleid.

7) Kas mängu eesmärk oli selge?

Mängu eesmärk oli pealehel olevas juhendist selgelt loetav.

8) Kas said kuidagi kõrvaldada AId?

Proovisin pudeliga loopida, kuid ei õnnestunud, võibolla seetõttu, et raske oli aru saada kuhu reaalselt pudel lendab ehk kas ma viskasin selle AI pihta või kõrvale maha.

9) Kas mingis olukorras ei olnud võimalik AId üle kavaldada?

Ei oska öelda.

10) Kas oli olukordi kus maailmas edasi liikumine oli võimatu mingil põhjusel?

Ei õnnestunud leida sellist kohta.

11) Kas mängus oli suuremaid vigasid? (BUGe. Läbi seina sai joosta või kuidagi muud moodi mängu katki teha mis ei olnud selleks ette nähtud)

Ei märganud, et oleks.“

Martin Ojamets –

„1) Ei, mängus oli olukordi kus pidi kasutama hiilimist ja/või pudeli viskamist.

2) Jah, reageeris.

3) Jah lõppes, kuigi kui pudel oli veel lennus ja vahele jäid, siis robotid ikkagi keerasid ennast pudeli poole. Kasutaja ise ei saanud edasi mängida ning pidi minema tagasi "main menu"

4) Jah, mõlemad droonid reageerisid.

5) Jah.

6) Jah, kuid kui sul oli pudel käes, seisid uue pudeli peal, viskasid vana pudeli, siis uut pudelit ta kätte ei võtnud. Pidi selle pealt maha liikuma, uuesti peale.

7) Algul ei olnud, samas ei lugenud ma mida tegema pidi ka, vajutasin kohe "play"

8) Pudeliga sai manipuleerida kuhu droonid läksid, et ise saaks edasi liikuda.

9) Ei. Igas olukorras oli võimalik drooni manipuleerida, et edasi liikuda.

10) Ei. Vastus 9.

11) Suuremaid ei. (Kui tahad sisse jätta, kasutaja sai kükitada ja sprintida. Heli tekitas küll sama palju nagu sprintides, aga kasutaja ise kükitas)“8

Jenny Gramdal (MacOS ja inglise keeles)–

„1) Was it possible to just run through the game or did you have to use sneaking to get past AI?

Both, you cant really just run through as the AI will catch you, so you have to combine both, but for me I got caught even if I walked or crawled.

2) Did the AI react to the sounds made by the player like bottle breaking and footsteps?

Yes it did.

3) When getting caught did the game end or was there way to continue playing?

The game ended.

4) When bottle broke near multiple AI did they work together aka both react or did only a single drone always react?

Haven’t come this far.

5) Were you able to move around as a player?

Yes, 360.

6) As a player were you able to pick up and throw items?

Yes

7) Was the objective of the game straight forward?

Yes

8) Were you able to eliminate the AI?

Haven’t come this far

9) Was there a situation where you were unable to outsmart the AI?

Yes, many times

10) Was there situations where moving forward in the world was impossible for some reason?

Nope

11) Was there major bugs in the game? (Were you able to play the game in unintended ways like going through walls or falling through map, not get hit at all by scanners so forth)

Nope“

Lisa 3 – Trello ajalogi

Card name	Board	List	User	Comment	Date started(f)	Time spent
Project setup	L6put66	Done	Marek Pihel		03/16/2022, 11:20 AM	1h 30m
Modelling for the game	L6put66	Done	Marek Pihel	Created necessary models	03/26/2022, 10:00 AM	1h
Textures for models and importing everything to Unity	L6put66	Done	Marek Pihel		03/26/2022, 12:30 PM	3h 30m
AI/enemy design	L6put66	Done	Marek Pihel		03/26/2022, 4:00 PM	1h 30m
AI/Enemy movement	L6put66	Done	Marek Pihel		03/27/2022, 11:06 AM	1h 48m 53s
Player movement	L6put66	Done	Marek Pihel		03/27/2022, 2:44 PM	10m
Player movement	L6put66	Done	Marek Pihel		03/27/2022, 2:55 PM	1h 48m 37s
Player movement	L6put66	Done	Marek Pihel		03/28/2022, 10:38 AM	2h 27m 12s
Create UI	L6put66	Done	Marek Pihel		03/28/2022, 1:56 PM	1h 53m 6s
Create UI	L6put66	Done	Marek Pihel	30 minutes of work without starting timer adjustment	03/28/2022, 4:06 PM	30m
Create UI	L6put66	Done	Marek Pihel		03/29/2022, 9:51 AM	3h 3m 29s
Create UI	L6put66	Done	Marek Pihel		03/30/2022, 9:37 AM	1h 53m 56s
AI discovery and player elimination (loss condition)	L6put66	Done	Marek Pihel		03/30/2022, 12:44 PM	58m 49s
Fix multiple game manager instances	L6put66	Bugs fixed	Marek Pihel		03/30/2022, 2:10 PM	11m 3s
AI/Enemy movement	L6put66	Done	Marek Pihel		03/30/2022, 2:29 PM	1h 23m 24s
AI/Enemy movement	L6put66	Done	Marek Pihel		03/31/2022, 2:20 PM	2h 19m 24s
Picking up items from the ground	L6put66	Done	Marek Pihel		04/02/2022, 6:37 PM	2h 1m 47s
Allow player to throw items	L6put66	Done	Marek Pihel		04/02/2022, 9:12 PM	2h 4m 58s
Bottle creates sound	L6put66	Done	Marek Pihel		04/03/2022, 2:04 PM	2h 13m 27s
Bug: fix double UI opening after failed game state	L6put66	Bugs fixed	Marek Pihel		04/03/2022, 6:15 PM	13m 33s
AI reacting to stimuli	L6put66	Done	Marek Pihel		04/03/2022, 6:40 PM	1h 14m 34s
AI reacting to stimuli	L6put66	Done	Marek Pihel		04/04/2022, 12:25 PM	2h 4m 31s
Bug: Player able to move while in menu	L6put66	Bugs fixed	Marek Pihel		04/04/2022, 2:55 PM	39m 37s
Bug: Player able to move while in menu	L6put66	Bugs fixed	Marek Pihel		04/05/2022, 10:21 AM	5m
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/05/2022, 10:33 AM	1h 40m 30s
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/06/2022, 1:26 PM	1h 45m 12s
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/06/2022, 3:11 PM	5m 15s
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/07/2022, 9:31 AM	1h 22m 45s
Bug:Sneaking to stand up not working as intended	L6put66	Bugs fixed	Marek Pihel		04/08/2022, 11:03 AM	1h 3m 6s
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/08/2022, 12:25 PM	17m 33s
Make AI work together to search for the player	L6put66	Done	Marek Pihel		04/08/2022, 2:56 PM	30m 22s
Make AI work together to search for the player	L6put66	Done	Marek Pihel	Worked on refactoring code and building the system for expanding radius plus testing	04/09/2022, 10:00 AM	2h
Make AI work together to search for the player	L6put66	Done	Marek Pihel	Last refactoring and testing of drone AI working together	04/10/2022, 8:30 AM	1h
Implement sound to the game	L6put66	Done	Marek Pihel		04/10/2022, 9:55 AM	23m
Implement sound to the game	L6put66	Done	Marek Pihel		04/10/2022, 10:19 AM	1h 53m 39s
World/Level design	L6put66	Done	Marek Pihel		04/12/2022, 11:55 AM	45m
World/Level design	L6put66	Done	Marek Pihel		04/12/2022, 12:30 PM	3h 21m 1s
World/Level design	L6put66	Done	Marek Pihel		04/13/2022, 7:56 AM	3h 30m
World/Level design	L6put66	Done	Marek Pihel		04/13/2022, 11:26 AM	3h 31m 12s
Balancing and bug fixes	L6put66	Done	Marek Pihel		04/13/2022, 3:01 PM	26m 19s
Balancing and bug fixes	L6put66	Done	Marek Pihel		04/15/2022, 10:39 AM	11m 4s
Win screen and add keys to UI	L6put66	Done	Marek Pihel		04/15/2022, 10:52 AM	1h 45m 48s
Playtesting and bug fixes according to playtesting	L6put66	Done	Marek Pihel		04/15/2022, 12:57 PM	6h 30m
Playtesting and bug fixes according to playtesting	L6put66	Done	Marek Pihel		04/15/2022, 5:53 PM	4h
						72h 37m

Joonis 9. Trello ajalogi power-upi väljavõte.