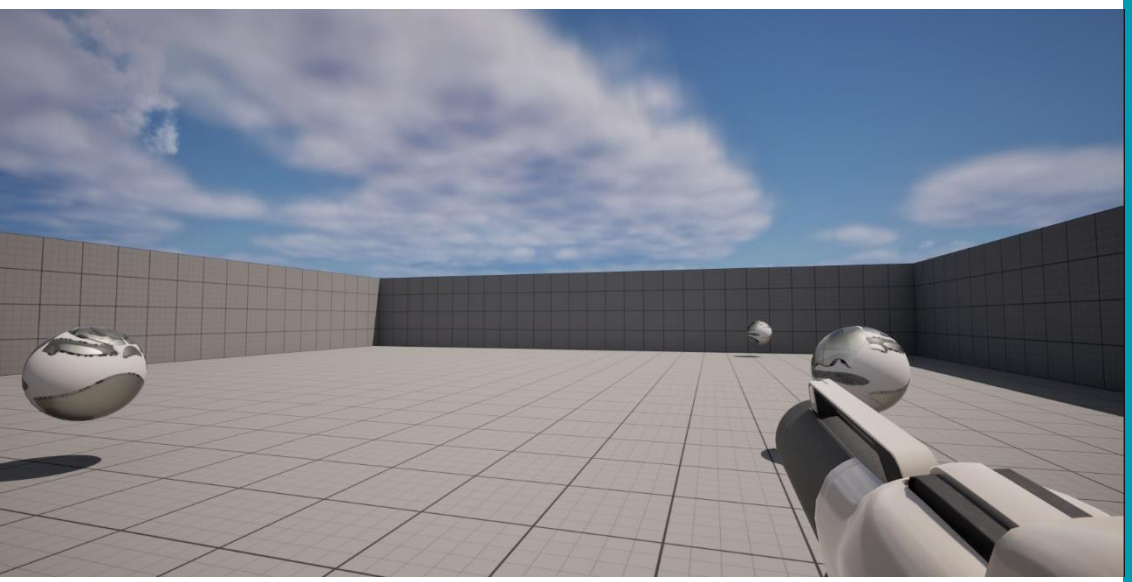
Kauhanen Jani

Automaattisen tähtäyksen tunnistaminen pelaajan liikkeitä seuraamalla

Tradenomi (AMK)
Tietojenkäsittely

Kevät 2024



KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Kauhanen Jani

Työn nimi: Automaattisen tähtäyksen tunnistaminen pelaajan liikkeitä seuraamalla

Tutkintonimike: Tradenomi (AMK), Tietojenkäsittely

Asiasanat: Aimbotin tunnistus, anti-cheat, peliteknologia, reilun pelaamisen edistäminen, videopelit

Opinnäytetyön aihe oli selvittää, voidaanko ampumispeleissä huijareiden käyttämää automaattista tähtäystä tunnistaa ja siinä kehitettiin ohjelma, joka tarkkailee pelaajan kääntymisen muutoksia ruudunpäivityksien välissä. Tavoitteena oli edistää reilua pelaamista tunnistamalla huijareita. Työssä pyrittiin selvittämään millä tavoin huijausohjelma toimii ja miten pelaajan pelikäyttäytyminen eroaa rehellisestä pelaajasta.

Ensimmäinen vaihe oli kerätä tietoa automaattisen tähtäyksen toimintaperiaatteista ja nykyisistä tunnistamismenetelmistä. Tämä edellytti perehtymistä aimbotin tekniseen toiminnallisuuteen siltä osin, jossa tapahtuu pelaajan kääntyminen kohteeseen yksittäisen ruudunpäivityksen välissä. Kerätyillä tiedoilla toteutetaan toimivuudeltaan samanlainen aimbot testiympäristöön. Testiympäristön luominen toteutettiin Unreal Engine 5:n ensimmäisen persoonan ammuntapelipohjaa hyödyntäen. Testiympäristöön luodaan maalitaulut sekä ohjelma, joka tarkkailee ja tallentaa pelaajan katselukulmien muutokset niinä hetkinä, kun pelaaja ampuu.

Pelitestaukseen pyrittiin valitsemaan pelaajia, joilla on jo aikaisempaa kokemusta ampumispeleistä sen vuoksi, että saadaan laadukasta dataa kerättyä. Pelaajat pääsivät ampumaan maalitauluja ilman koneavustusta ja koneavustuksen kanssa. Dataa kerättiin minimissään 0,5 sekuntia ampumista edeltävältä ajalta ja 1,5 sekuntia ampumisen jälkeiseltä ajalta. Testauksesta kerätyt tiedot laitetaan samaan Excel-taulukkoon vertailua varten. Tuloksia analysoimalla saatiin selville, että rehellisillä pelaajilla katselukulmien muutokset rajoittuvat hyvin pieniin muutoksiin ruudunpäivityksien välillä ja epärehellisillä muutokset olivat verrattain erityisen suuria.

Testituloksien perusteella menetelmä toimii tietyn tyyppisissä huijausohjelmissa, mutta se edellyttää vielä laajamittausta testaamista ja datan analysointia. Tutkimusta voidaan kuitenkin hyödyntää reilun pelaamisen edistämisessä. Menetelmää voidaan myös jatkokehittää ottamalla huomioon muitakin muutoksia pelaajien käyttäytymisessä sekä tutkimalla erilaisten huijausohjelmien teknistä toiminnallisuutta.

Abstract

Author: Kauhanen Jani

Title of the Publication: Identifying Automatic Targeting through Player Motion Analysis

Degree Title: Bachelor of Business Administration, Business Information Technology

Keywords: Aimbot detection, anti-cheat, game technology, promotion of fair play, video games

The topic of the thesis was to investigate whether automatic aiming used by cheaters in shooting games can be identified, and a program was developed to monitor changes in player rotation between screen updates. The aim is to promote fair play by detecting cheaters. The work aimed to determine how cheating software works and how player behavior differs from honest players.

The first stage involved gathering information on the principles of automatic aiming and current detection methods. This required understanding the technical functionality of aimbots concerning the player's rotation towards the target between invidual screen updates. The collected information was used to implement a similarly functional Aimbot in a test environment. The creating of the test environment was done utilizing Unreal Engine 5's first-person shooter game template. Targets and a program were set up in the test environment to monitor and record the player's changes in viewing angles during shooting.

For the game testing, players with previous experience in shooting games were selected to ensure high-quality data collection. Players were allowed to shoot targets both without and with assistance from the program. Data was collected for a minimum of 0.5 seconds before shooting and 1.5 seconds after shooting. The collected data from the tests was compiled into the same Excel spreadsheet for comparison. Analyzing the results revealed that honest players exhibited very small changes in viewing angles between screen updates, whereas dishonest players showed relatively significant changes.

Based on the rest results, the method works for certain types of cheating software, but it still requires extensive testing and data analysis. Nevertheless, the research can be utilized to promote fair play. The method can also be further developed by considering other changes in player and examining the technical functionality off various cheating programs, thereby enhancing its application in promoting fair play.

Sanasto

Aimbot: Ohjelmisto, joka automaattisesti tähtää ja ampuu pelaajan puolesta.

Actori: Peliobjekti Unreal Engine 5:ssä.

Anti-cheat: Ohjelmisto, jolla tunnistetaan huijausohjelmia.

BluePrint: Visuaalinen ohjelmointi.

CS: GO: Valven julkaisema FPS-peli.

Collision sphere: Pallon muotoinen törmäystarkistus.

DMA: Suoraa muistinohjausta laitteesta.

Fps: Frames per second eli ruudunpäivitys nopeus.

Key-bind: Tietty toiminto tai komento on liitetty tiettyyn näppäimeen tai näppäinyhdistelmään.

Koodaja: Pelaaja, joka käyttää huijausohjelmia.

Lag-spike: Tilanne, jossa ruudunpäivitysnopeus (FPS) tipahtaa erityisen pieneksi, mikä aiheuttaa pelin hidastelua.

MMORPG: Massively multiplayer online game, massiivinen monen pelaajan verkkoroolipeli.

Online: Verkkopeli, jossa on useita pelaajia.

Overwatch: Paikka, jossa pelaajat voivat päättää pelivideoita katsomalla, käyttääkö toinen huijausohjelmia.

Pay to win: Erikseen maksetusta sisällöstä saadusta hyödystä muita pelaajia kohtaan.

Punkbuster: Ohjelmisto, jolla tunnistetaan huijausohjelmia.

Riot, Valve: Pelialan yrityksiä.

Skripta: Pienimuotoinen ohjelma, joka tekee yksinkertaisia asioita.Unreal Engine: Pelimoottori, jolla voidaan kehittää pelejä tai muita ohjelmistoja.

FPS-peli: First person shooter game, ensimmäisen persoonan ammuntapeli.

Sisällys

1 Johdanto

Tämä opinnäytetyö keskittyy tutkimaan automaattisen tähtäyksen, eli aimbotin tunnistamista seuraamalla pelaajan liikkeitä seuraamalla. Työ liittyy kiinnostukseeni pelienkehittämiseen ja haluun edistää reilua pelaamista. Tarkoituksena on tarjota syvempää ymmärrystä aimbotin tunnistamisen haasteisiin sekä tuoda esiin käytännön ratkaisuja, jotka voivat auttaa pelienkehittäjiä taistelussa epäreilua pelaamista vastaan.

Työ rajataan siihen, että voidaan tunnistaa aimbot, joka tekniseltä toteutukseltaan kääntää pelaajan framen aikana vastustajaan. Tarkoituksena on saada uusia näkökulmia sitä varten, jotta saadaan edistettyä reilua pelaamista ja vähintään että epäreiluilla toimijoilla olisi haastavampaa kehittää huijausohjelmia. Toivon, että tämä tutkimus voi tuoda lisää ymmärrystä huijausohjelmien tunnistamiseen ja ehkäisyyn, ja näin olla avuksi peliteollisuudessa ympäri maailmaa.

Pelimaailma, joka tarjosi alussa pelaajille tasa-arvoiset mahdollisuudet, muuttui yhdessä hetkessä. Vastustajajoukkueet, jotka olivat samantasoisia kilpakumppaneita, saivat käyttöönsä uusia kiellettyjä ohjelmia, joilla he kykenevät näkemään seinien lävitse ja heidän tähtäämisensä on täydellistä, aivan kuin se ei olisi enää ihmisille mahdollista [1]. Tällainen äkillinen muutos heitti pelaamisen maailman kaaokseen ja haastoi sen, mitä pidettiin ennen normaalina.

Pelaajat ovat joutuneet sopeutumaan tähän valtavaan haasteeseen kilpakumppaneidensa voittamiseksi. Vieläkin merkittävämpää on ollut, miten pelinkehittäjät ovat vastanneet tähän mittavaan muutokseen.

Kuinka säilyy peliympäristön tasapaino, joka oli alun perin perustunut pelaajien taitoihin ja reiluuteen? Kuinka estetään väärinkäytökset ja varmistetaan, että kaikki pelaajat voisivat nauttia pelaamisesta? Nämä kysymykset heräsivät pelienkehittäjillä, kun huijausohjelmien käyttäjät pelasivat verkkopelejä. Kun huijaa verkkopeleissä, se vaikuttaa suoraan – enimmäkseen usein negatiivisesti – kaikkien muiden kokemuksiin. [2.]

Artikkelin mukaan 9436 vastaajasta 60 %:lla oli negatiivisa kokemuksia peleistä, joissa kanssapelaaja käytti huijausohjelmia. Kiinan, Saksan, Japanin, Etelä-Korean, Iso-Britannian ja Yhdysvaltojen kuluttajista 77 % vastaajista lopettaa todennäköisesti verkkopelaamisen, mikäli he epäilevät muiden pelaajien huijaavan, ja 48 % kertoi, että he ostaisivat todennäköisesti vähemmän pelin sisäistä sisältöä. [2.]

2 Huijaamisen historia

Yksi videopelihistorian merkittävimmän huijauksen on tehnyt Kazuhisa Hashimoto, joka itse oli pelintekijä. Hän teki sen huomattuaan pelin olevan liian haastava vianmääritys- ja testausvaiheessa. [2.] Huijaus on niin sanottu ”Konami Code”, joka on mahdollisesti historian tunnetuin koodi. Vaikkakin kyseinen koodi oli tehty Gradius-peliä varten, kuitenkin toiset pelinkehittäjät ottivat samaisen koodin käyttöönsä ja se jäikin vahingossa julkaistavaan peliin. Samankaltaisen huijausohjelman tekivät myös Solar Jetmanin kehittäjät, jotta testaajat pääsivät kokeilemaan testattavia asioita nopeammin, ja kehittäjille kävi samoin myös julkaisussa. [3.] Tästä voidaan päätellä, että huijausohjelmat olivat alun perin tarkoitettu nopeuttamaan pelienkehitystä eikä siihen, että niitä käytettäisiin huijaamiseen, kuitenkin vastaavista tapahtumista tuli ongelma vasta, kun internet ja verkkopelit alkoivat yleistymään. Vastareaktiona alkoikin huijausohjelmien kehitys [4, s. 5].

Artikkelin mukaan kehittäjien tekemiä huijausohjelmistoja ei enää jätetty online-peleihin ja kehittäjät huomasivat samaan aikaan, että he voivat rahastaa pay to win -mallilla, jolloin he voivat rahastaa myymällä ohjeita, koska pelaajat olivat valmiita maksamaan siitä, että saisivat pelin helpommaksi [5]. 2000-luvun alussa alkoikin online-peleissä tulemaan ongelma, koska huijarit näkivät seinien lävitse ja ampuivat täydellisellä tähtäyksellä, jolloin alkoi huijauksenesto järjestelmien kehittäminen.

Vaikkakin koodeja käytetään yleisimmin saamaan etulyöntiasema peleissä, joissa hyöty jää vain kyseisen pelin voittamiseksi, koodaamista on kuitenkin tapahtunut turnauksissa, joissa on suuriakin palkintoja. Esimerkiksi OpTic-tiimin pelaaja nimeltään Nikhil ”forsaken” jäi kiinni huijauksesta CS:GO-turnauksessa kiinni, jossa oli 100 000 dollarin palkinto voittajille. [6.]

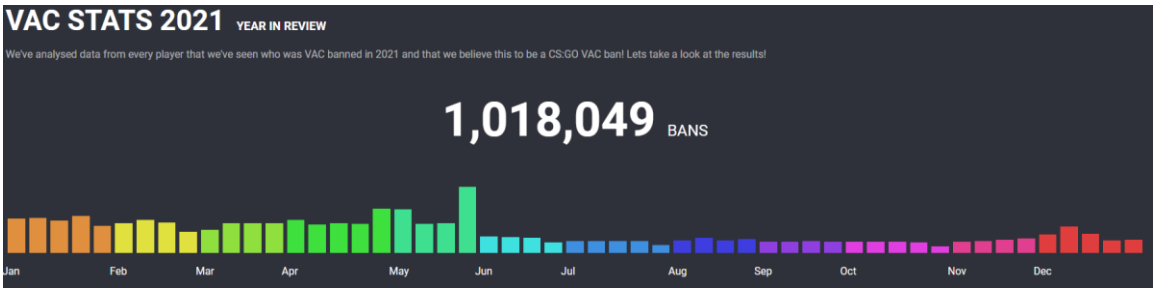
2.1 Huijauksenestonkehittyminen

Ensimmäisten huijauksenestojärjestelmien eli huijauksenesto-ohjelmien ajankohtaa on haastava määritellä, mutta niitä on ollut jo 1980 vuosikymmenellä verkkopeleissä, kuten Ultimate-Onlinessa on jo ollut huijauksenesto toiminnallisuuksia. Ne on tosin julkaistu vasta 1997. Tietoa huijauksenestontoiminallisuudesta ei julkisesti ole kuitenkaan kovinkaan paljon tietoa. Toiminnallisuudet olivat kuitenkin vielä niin alkeellisia, että jo julkaisuvuotena pelaajat keksivät, miten voidaan ohittaa huijauksenesto ja käyttää aukkoa hyväkseen. Väitöskirjassa viitataan myös erääseen mieheen, joka olisi tienannut elantonsa kaappaamalla muiden pelaajien virtuaalisia taloja, jotka

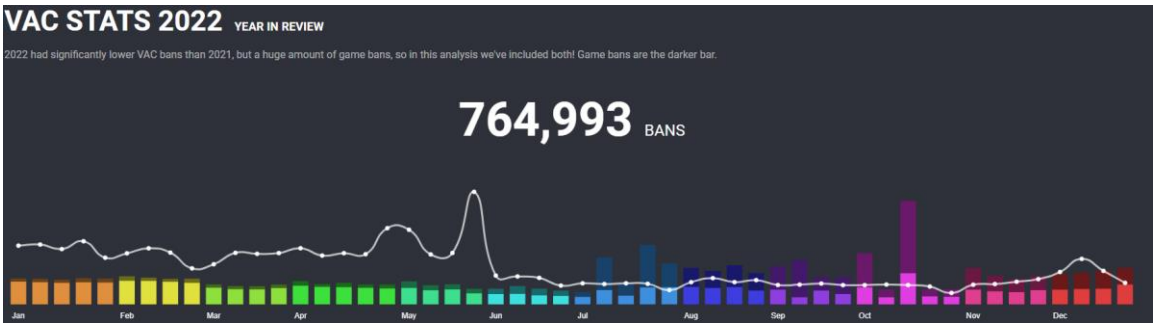
olivat menneet kaupaksi eBayssa ja kiinalaisilla markkinapaikoilla noin 2000 dollarin hinnalla. [4, s. 9.]

Todennäköisesti kuuluisin huijauksenesto-ohjelma, Punkbuster, on yksi ensimmäisiä koodaajien tunnistamiseen tarkoitettu ohjelma, joka teki debyyttinsä Half-Lifelle vuonna 2000. Ohjelma skannasi pelaajan laitteen muistin tunnettujen koodien varalta, skriptoilta ja key-bindeiltä [2]. Punkbusteri levisikin laajalle ja sitä on käytetty useissa peleissä, kuten The Battlefield, Ghost Recon, Rainbow Six, Medal of Honor ja Call of Dutyssä. Kuitenkin hyvin moni isoimmista peliyhtiöistä on tehnyt oman versionsa anti-cheat järjestelmistä, kuten Valve.

Julkisesti saatavilla olevista graafeista (kuvat 1 ja 2) nähdään selvästi, että huijareita jää jatkuvasti kiinni. Kyseessä on vain yksittäisen pelin tilasto (CS: GO), joten voidaan havaita, että huijaaminen online-peleissä on hyvinkin suuri ongelma, johon tarvitaan ratkaisuja sekä jatkuvaa kehitystä. Vaikkakin kolmannen osapuolien tekemät tilastot eivät ole täydellisiä, koska kaikkia käyttäjätilejä ei pääse suoraan analysoimaan, siltikin otanta käyttäjistä antaa jo tarpeeksi tietoa tämänhetkisestä tilanteesta. [7.]



Kuva 1. Statistiikkaa vuoden 2021 VAC tilastoista [7].



Kuva 1. Statistiikka vuoden 2022 VAC tilastoista [7].

Valven huijauksenestonpäivitys tunnisti historiallisesti suurimman määrän pelaajia tammikuussa 2019. Kuitenkin ratkaisevassa roolissa on ollut koneoppiminen, joka vaatii massiivisen määrän dataa, että siitä saa luotettavan. Ilman koneoppimista taistelu huijareita vastaan vaatii jatkuvaa anti-cheat ohjelmistojen päivittämistä. [8.] Yksinkertaistettuna, kun pelinkehittäjät päivittää suojausta, niin huijausohjelmien tekijät reagoivat siihen, ja päinvastoin pelinkehittäjien täytyy reagoida, kuten kuuluu sanonta ”Every action has a reaction”.

Koneoppimisen käyttämisen ajatuksena oli pysäyttää tuulimylly, mutta kaikilla kehittäjistä ei ole tarvittavaa tiedonmäärää siihen. Esimerkiksi Valven kehittämä Vacnet vaatii heidän omasta Overwatchistaan dataa, jossa muut pelaajat päättävät peliä katsomalla, onko kyseinen pelaaja koodari. Syyllisiksi todetut pelaajat ja niiden käyttäytyminen pelissä laitetaan yhteen koneoppimisen kanssa, jolloin kone pystyy päättelemään suurin piirtein samalla tavalla kuin ihminen päättelisi syyllisyyttä. Kun John McDonald (Valve) ryhtyi valmistelemaan koneoppimista kyseiseen käyttötarkoitukseen, hän mietti asiaa ”If a human can do it, why can’t a machine?”, minkä lopputulos nähdäänkin Valven statistiikkagraafista kuvassa 3. [8.]

Kuva 2. Statistiikkaa tunnistetuista koodareista [8].

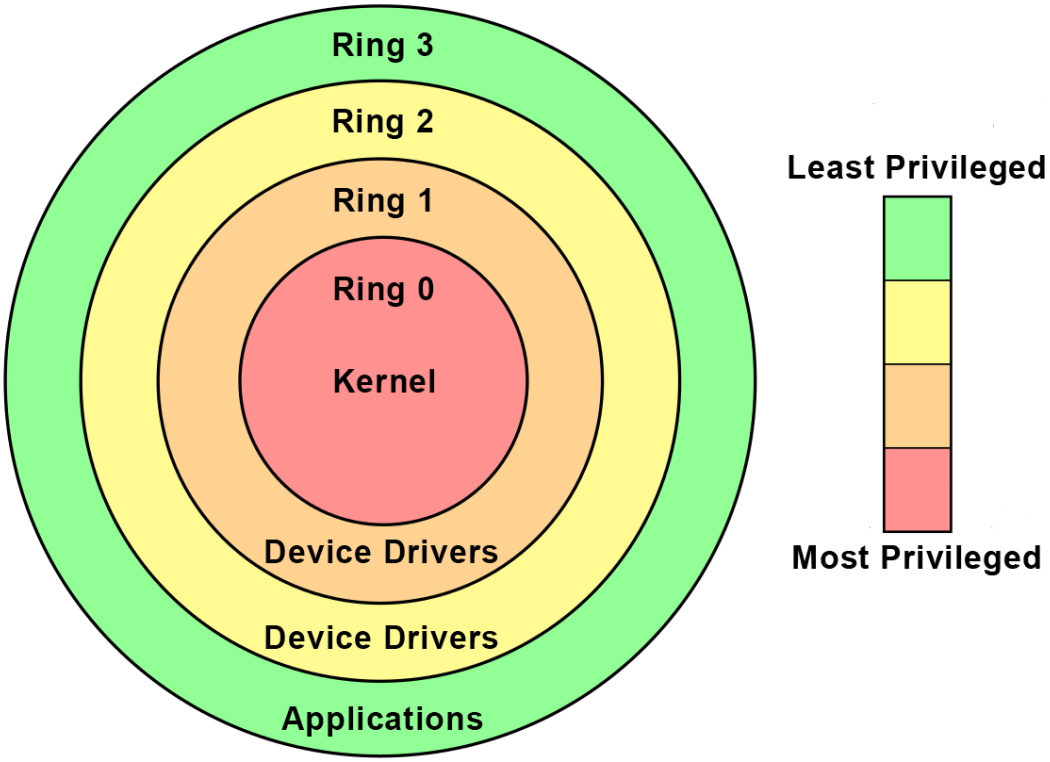
Ensimmäinen asia, johon Valve halusi puuttua oli juuri automaattinen tähtäys, jonka ihminen pystyy havaitsemaan. Heräsi kuitenkin kysymyksiä siitä, minkälaista dataa tarvitaan, ja lopulta se olikin vain pelaajan katselukulma X-horisontaali ja Y-pystysuora. Kyseisiä arvoja tallennetaan 0,5 sekunnin ajalta jokaiselta framelta, kun pelaaja ampuu, sekä 0,25 sekunnin ajalta ampumisen jälkeen. Saatu tieto laitetaan koneelle käsiteltäväksi 140 tapahtuman kappaleina, jolloin saadaan automaattisesti pääteltyä, käyttääkö pelaaja koodeja vai ei. [9.]

2.2 Nykytilanne

Koneoppimisen lähestymistapa on siis erittäin toimiva, mutta ei kuitenkaan täysin luotettava. Miten ne pelinkehittäjät, joilla ei ole tarvittavaa dataa tai edes mahdollisuutta kerätä sitä tarpeeksi? Mahdollisia ratkaisuja on useita, koska kuitenkin pelaajan käyttäytymisen seuranta voi mahdollisesti olla ratkaisevassa roolissa tuulimyllyn pysäyttämiseksi. Vaikkakin on otettu käyttöön koneoppiminen, joka on koulutettu ihmisten tekemillä arvioilla, siltikin osa huijareista ei jää kiinni. [10.]

Viime vuosina huijausohjelmistojen kehittäjät ovat alkaneet hyödyntää haavoittuvuuksia tai vioittaa Windowsin allekirjoituksen varmistusta suorittaakseen sovelluksiaan tai niiden osia ytimen tasolla. Ongelma syntyy siitä, että kernel-tilassa suoritettava koodi voi kytkeä itseensä juuri ne järjestelmäkutsut, jotka pitäisi olla luotettavia tietojen hakemiseksi. Se Muuttaa tuloksia näyttämään laillisilta tavalla, jota voisi olla vaikea havaita. Riot Gamesilla on jopa havaittu, että huijausohjelmistojen kehittäjät käyttävät erikoistunutta laitteistoa, joka hyödyntää DMA1:stä lukeakseen ja käsitelläkseen järjestelmän muistia. Mikäli se on täydellisesti tehty, sitä on mahdoton havaita. Häiritsevän suuri määrä käyttäjiä on osoittanut olevansa kiinnostuneita liittymään toisen henkilön niin sanottuun bottiverkkoon saadakseen suorittaa erilaisia toimintoja peleissä. Tämän takia monien huijausohjelmien käyttöoikeudet ovat jopa korkeammalla tasolla kuin osa huijauksenestojärjestelmistä. [11.]

Käyttöoikeustasojen visuaalisessa ympyrässä nähdään tasot, joihin on pääsy ja muokkausoikeudet. Viitaten aikaisempaan tekstiin, jossa käsiteltiin sitä, että huijausohjelmilla on jopa suuremmat käyttöoikeudet kuin huijauksenestolla niin ring 0-tasolla toimivan ohjelman havaitseminen on liki mahdotonta, mikäli huijauksenesto toimii vähemmillä käyttöoikeuksilla. Tästä syystä esimerkiksi Riotin huijauksentunnistus järjestelmä toimii kernel-tasolla, käyttöoikeuksien tasot visualisoituna kuvassa 4. [11.]



Kuva 3. Päätelaitteen käyttöoikeustasot visualisoituna [11].

Kuitenkaan sekään ei vaikuta toimivan täydellisesti, koska artikkelin mukaan huijaaminen on lähtenyt käsistä ja Riotin huijauksenestojärjestelmää pidetään jopa vitsinä [12]. Ei toivottujen ohjelmien seuranta ja muistin muokkauksen tunnistamisessa tulee usein se ongelma, että huijauksenestoa joutuu päivittämään sitä mukaa, milloin huijaus ohjelmistojen kehittäjät päivittämät omia ohjelmiansa.

Vuosikymmenien ajan huijaamista on pyritty estämään monella eri tavalla, kuitenkin vaikuttaisi siltä, että aina löytyy jokin keino, jolla pelaajat pääsevät käyttämään ohjelmistoja, jotka ovat kiellettyjä. Vaihtoehdot pysäyttää huijaaminen kokonaan on hyvin epätodennäköistä, mutta huijausohjelmien työstäminen on muuttunut haastavammaksi.

Valven Vacnet, joka kehittyy jatkuvasti sitä mukaa, kun käyttäjät silmämääräisesti tulkitsevat, huijaako pelaaja vai ei, vaikuttaisi olevan tähän mennessä paras huijauksentunnistusjärjestelmä. Siinäkin tulee ongelmia silloin, kun ihminen ei enää pysty tunnistamaan peliä katsomalla (Valven Overwatch), huijaako pelaaja vai ei. Esimerkiksi automaattisen tähtäämisen tunnistaminen on joissain tapauksissa erittäin helppoa, mutta silloin kun ohjelmaa muokataan tarpeeksi, niin siitä

voi tehdä sellaisen, jota ihminen ei välttämättä huomaa. Kuten esimerkkikoodissa vaihdellaan satunnaisesti tähtäämisen nopeutta. [12.]

Näin ollen opinnäytetyössä tarkastellaan sitä lähestymistapaa, jolla katsotaan, mikä on ihmiselle mahdollista tähtäyksen suhteen. Nimenomaan katselukulmien muutoksien seuranta, joka voidaan ohjelmassa tarkistaa katselukulman radiaaneilla, asteilla tai yksikkövektorilla. Mikäli ihminen pystyy silmämääräisesti katsomalla peliä sen huomaamaan, miksi ei tietokone voisi sitä myös huomata ja voisiko se mahdollisesti tehdä sen paremmin kuin ihminen?

3 Aimbotin tunnistaminen

Tunnistaminen edellyttää datan keräämistä testiympäristöstä, jossa pelaaja käyttää aimbottia. Ei ole merkitystä mistä FPS-pelistä ottaa teknisen toteutuksen aimbottiin, koska sen toiminta periaate on samankaltainen eri peleissä: Tähdätä tarkasti vastustajan pelihahmon ruumiinosaan ja ampua [14]. Kuitenkin tässä pitää huomioida se, että siitäkin on useita erilaisia versioita ja täten tutkimus kohdistetaan yksittäiseen versioon, jossa tähtäys tapahtuu yhden ruudunpäivityksen yhteydessä, joka on varsin yleinen.

Toimintaperiaatteen samaisuuden vuoksi ympäristönä käytetään Unreal Enginen valmista aloitusmateriaalia, jossa on pelihahmo, jolla voi liikkua sekä ampua. Näin ollen kyseiseen materiaaliin voidaan lisätä maalitauluja, jotka toimivat vastustajina. Ohjelmointipuolelta pelaajahahmolle tuotetaan samankaltainen aimbot kuin nykypeleissä olemassa olevat.

Ohjelmointitasolla koodista voidaan päätellä, että haetaan 2 pisteen sijainnit, oma sijainti sekä vastustajan sijainti, jolloin voidaan suoraan tietokonetta käskeä kääntämään katselukulma suoraan vastustajaan. Näin ollen käännös tapahtuu ensimmäisen framen aikana täydellisesti kohteeseen [15.] Ihmispelaajan kääntyminen ei ole niin nopeaa.

Mikä on se raja, johon ihmispelaaja pystyy kääntymään ensimmäisen framen aikana? Sen selvittämiseksi tarvitaan erittäin suuri otanta, tosin tässä työssä otanta tehdään pienemmästä ryhmästä, josta voidaan selvittää ihmispelaajan kääntymisnopeus ensimmäisen framen aikana suuntaa antavaksi arvoksi. Jolloin muutokseen voidaan lisätä turvaväliksi tietty astemäärä.

```
if (closestTarget.first.has_value()) {
                normalise(closestTarget.second);
                vec3 AimPunch = localPlayer->m_aimPunchAngle;

                AimPunch *= 2.f;

                closestTarget.second -= AimPunch;

                normalise(closestTarget.second);

                clampAngle(closestTarget.second);

                *myViewAngle = closestTarget.second;}
```

Testiversiosta kerätään dataa talteen niiltä hetkiltä, kun hahmo ampuu, kuitenkin maksimissaan 0,5 sekunnin ajalta ennen ampumista ja 1,5 sekuntia tapahtuman jälkeen, mikä on jo todettu riittäväksi ajaksi Valven anti-cheat järjestelmän pohjalta [15]. Tieto, joka kerätään, on X- ja Y-katselukulmat, joko yksikkövektorina pelaajahahmon eteenpäin suunnasta, kulmana tai radiaaneina.

4 Pelitestaus

Luodusta testiympäristöstä järjestetään pelitestaus, jossa pelaajat voivat käyttää huijausohjelmaa sekä pelata ilman sitä. Fyysistä testauspaikkaa ei järjestetä, vaan testaajat voivat ladata ohjelmiston ja testata sitä.

4.1 Tiedon tallentaminen

Testauksesta saatu tieto kerätään talteen Excel-taulukkoon. Mikäli tiedon määrä kasvaa niin suureksi, ettei sitä ole enää käytännöllistä tarkistaa manuaalisesti, tiedot syötetään C++-ohjelmaan. Ohjelmalla tietokone tarkistaa tallennetusta tiedosta suurimmat katselukulmien muutokset, ja kyseinen toimenpide suoritetaan testausvaiheessa.

4.2 Tiedon analysointi

Huijaavasta pelaajasta kerätystä datasta tarkastellaan, kuinka suuren käännöksen pelihahmo tekee maalitauluun osuessaan juuri ennen ampumista. On todennäköistä, että tällä arvolla ei ole muuta rajaa kuin suurin mahdollinen kääntyminen eli 180 astetta horisontaalisessa- ja vertikaalisessa. Tästä voidaan päätellä, onko koodeja käyttävällä pelaajalla mitään rajaa kääntymisen suhteen.

Kun tarkastelussa on taas pelaaja, joka ei käytä koodeja, selvitetään otannasta maksimaalinen käännös, joka on tapahtunut ennen maalitauluun osumista. Saatua arvoa voidaan näin ollen käyttää vertailukohteena, mikäli pelaajan käännökset ylittävät saadun arvon voidaan päätellä, että kyseinen pelaaja käyttää huijausohjelmaa. Huomioiden kuitenkin, että otannasta ei voida suoraan määrittää ihmisen maksimaalista käännösnopeutta framen aikana, mutta se toimii kuitenkin suuntaa antavana arvona tässä työssä.

5 Koodinmuokkauksen vaikutus tunnistamiseen

Huijausohjelmien kuten aimbotin tunnistaminen on jatkuva taistelu peliyhteisöissä, missä huijausten kehittäjät pyrkivät jatkuvasti löytämään uusia tapoja välttää havaitsemista. Erityisesti koodinmuokkauksen osalta on näin, ja siksi sitä sanotaankin kissa ja hiiri -tyyliseksi. Kun uusi koodi julkaistaan, pelienkehittäjät reagoivat siihen ja tunnistavat sen ja vastaavasti huijausohjelmaa taas kehitetään. [16.]

Kuten tässäkin lähestymistavassa, huijausohjelmien kehittäjät todennäköisesti reagoisivat ja päivittävät koodiansa siten, ettei sitä enää havaittaisi. Kuitenkin erilaisia tarkastusmenetelmiä siihen, mihin ihminen kykenee ja mihin ei, on useita.

6 Yksityisyys ja oikeudenmukaisuus

Pelaajien liikkeiden tarkka seuranta ja tietojen tallentaminen herättää yksityisyyden suojan kysymyksiä. Tietoa pelaajien pelitottumuksista ja liikkeistä voidaan käyttää monin eri tavoin, ja pelaajilla tulisi olla oikeus tietää, miten heidän tietojaan käytetään ja säilytetään [17]. Kuitenkin tässä työssä kerätty tieto on täysin anonyymiä, joka sisältää vain katselukulmien muutokset, joita ei voida yhdistää keneenkään tiettyyn pelaajaan. Käyttäjänimiä tai vastaavia arkaluonteisia tietoja ei tallenneta. Ohjelmistojen kehittäjät kuitenkin huomauttavat, ettei ole olemassa anonyymiä tietoa, kaikki voidaan yhdistää toiseen tietokantaan [18].

Järjestelmän kyvyttömyys täysin ymmärtää kääntymisen nopeutta ja pienen otannan vuoksi herättää haasteita tarkkuudessa ja luotettavuudessa. Pieni otanta voi johtaa tilanteisiin, joissa järjestelmä ei pysty kattavasti huomioimaan erilaisten pelaajien kääntymisnopeutta [19].

Tämä haaste liittyy olennaisesti siihen, että pelaajien liikkeiden analysointiin käytetään rajallinen otanta tilanteista. Kääntymisen nopeuden ymmärtämisen vaikeus voi johtaa virheellisiin tuloksiin, erityisesti kun tässä työssä tehty järjestelmä ei kykene ottamaan huomioon erilaisia pelitilanteita ja pelaajatyylejä. Kuten mahdollista tietokoneen jäätymistä on niin sanottu lag spike, jolloin pelaajan kääntyminen voi olla huomattavankin nopeaa yhden framen aikana.

Pieni otanta voi vääristää järjestelmän käsitystä normaalista pelaamiskäyttäytymisestä ja tehdä siitä alttiin virheellisille tunnistuksille [19]. Lisäksi kääntymisen nopeuden tulkitseminen yksinään voi jäädä puutteelliseksi, jos järjestelmä ei pysty ottamaan huomioon muita tekijöitä, kuten pelitilannetta, ympäristöä tai pelaajan taitotasoa.

7 Automaattisen tähtäyksen luominen

Toimivan aimbotin työstäminen oikeaan peliin vaatii useita eri asioita: oman ohjelman injektointi pelin ohjelmaan, jotta voi muokata tiettyä muistipaikkaa, koodin kääntämistä tietokoneen muistista ja löytämään muistipaikasta vaadittavat tiedot. Vaadittavia tietoja ovat muun muassa vastustaja pelaajien määrä, sijainnit, tietyn ruumiinosan sijainti, mikäli haluaa tähdätä tiettyyn ruumiinosaan, onko pelaaja elossa sekä ottamaan myös huomioon mahdolliset huijauksenestojärjestelmät. [20.]

Yksinkertaisuudessansa automaattiseen tähtäykseen vaaditaan oman pelaajan sekä maalitaulun sijainti, jolloin voidaan matemaattisella laskukaavalla laskea suunta, johon halutaan tähdätä. Se voidaan laskea joko kulmana, suuntavektorina tai radiaaneina. Kun saatu suunta on laskettu, käskytetään pelaajahahmoa kääntymään kyseiseen suuntaan, jolloin voidaan laukaista. [20.]

Tässä työssä voidaan kuitenkin ohittaa suurin osa vaadituista toimenpiteistä, koska testiympäristössä kaikki tieto on saatavilla eikä tarvitse huomioida huijauksenestojärjestelmiä.

7.1 Suunnan laskeminen

Vektorin pituus T määritellään kaavalla [21] seuraavasti:

$$T = \sqrt{(X2 - X1)^2 (Y2 - Y1)^2 (Z2 - Z1)^2} \quad (1)$$

Missä

X2 on kohteen X-sijainti 3D-ympäristössä

X1 on lähtöpisteen X-sijainti 3D-ympäristössä

Y2 on kohteen Y-sijainti 3D-ympäristössä

Y1 on lähtöpisteen Y-sijainti 3D-ympäristössä

Z2 on kohteen Z-sijainti 3D-ympäristössä

Z1 on lähtöpisteen Z-sijainti 3D-ympäristössä

Suuntavektorin laskemisessa tarvitaan oman pelihahmon X1, Y1, Z1 ja vastustajan X2, Y2, Z2 sijainnit 3D-maailmassa. Suuntavektori $A\vec{B}$ on erotus pelihahmojen sijainnista eli X2 pelaajan sijainnista miinustetaan X1 pelaajan sijainti. Suuntavektori saadaan myös järkevämpään muotoon, kun se muutetaan yksikkövektoriksi. Yksikkövektori saadaan näin ollen jakamalla suuntavektori $A\vec{B}$ sen pituudella. Vektorin pituus lasketaan kaavalla (1).

7.2 Matematiikan soveltaminen C++-ohjelmistoon

Suuntavektorin laskemiseen ohjelmiston puolella tarvitaan ensimmäisenä kohteen sijainti sekä oma sijainti. Kohteen sijainti voidaan selvittää useilla eri tavoilla, mutta tässä tapauksessa se haetaan lisäämällä tagi maalitauluun Unreal Engine 5 editorin puolelta, pelihahmon CPP-tiedostoon lisätään array, johon tallennetaan jokainen objekti, jolla on kyseinen tagi. Näin ollen voidaan hakea kohteen sijainti, joka tallennetaan muuttujaan sekä tehdään tarkistus, että jotain on tallennettu.

Kun sijainnit ovat selvillä, luodaan vektorimuuttuja, johon tallennetaan arvo miinustamalla kohteen sijainnista oma sijainti. Lasketaan matka kohteisiin ja lopuksi käännetään pelaajahahmo lähimpään kohteeseen. OpenAI:n ChatGPT-kielimallia hyödynnettiin syntaksin suhteen sillä tavalla, että haetaan muun muassa oma sijainti sekä kohteen sijainti ohjelmoinnin osuudessa. Kuvassa 5 toteutettuna kohteiden sijainti sekä lasketaan oikea suunta ja käskytetään pelaaja kääntymään kohteeseen.

```cpp
void AsavefileCharacter::AutoAim() {
    FName EnemyTag = FName("AimTarget");
    TArray<AActor*> FoundActors;
    UGameplayStatics::GetAllActorsWithTag(GetWorld(), EnemyTag, FoundActors);
    FVector PlayerLocation = GetActorLocation();
    if (FoundActors.Num() > 0) {
        AActor* ClosestTarget = nullptr;
        float MinDistanceSquared = MAX_FLT;
        for (AActor* TargetActor : FoundActors) {
            if (TargetActor) {
                float DistanceSquared = FVector::DistSquared(TargetActor->GetActorLocation(), PlayerLocation);
                if (DistanceSquared < MinDistanceSquared) {
                    MinDistanceSquared = DistanceSquared;
                    ClosestTarget = TargetActor;
                }
            }
        }
        if (ClosestTarget) {
            FVector TargetLocation = ClosestTarget->GetActorLocation();
            FVector DirectionToTarget = TargetLocation - PlayerLocation;

            FRotator TargetRotation = FRotationMatrix::MakeFromX(DirectionToTarget).Rotator();
            APlayerController* PlayerController = UGameplayStatics::GetPlayerController(GetWorld(), 0);
            if (PlayerController) {
                PlayerController->SetControlRotation(TargetRotation);
            }
        }
    }
}
```

Kuva 5. C++-toteutus automaattisen tähtäyksen toiminnallisuudesta.

8 Unreal Enginen ensimmäisen persoonan mallipohja

Unreal Engine 5:ssa oleva mallipohja (engl. First person template) sisältää testiympäristön luomiseen sopivan mallipohjan, joka sisältää seuraavat asiat:

- pelattavan ensimmäisen persoonan hahmo, jolla voi liikkua,
- ase, joka voidaan ottaa haltuun, sekä ammusprojektiilin luominen sekä sen käyttäytyminen,
- pelikenttä, jossa on perusgeometriaa, kuten ramppeja sekä alusta, ja
- kuutioita, jotka reagoivat, mikäli ammus törmää niihin.

Mallipohja saadaan käyttöön luomalla uusi projekti Epic Gamesin käyttöliittymän kautta. [21.]

9 Tiedon tallentaminen testiympäristössä

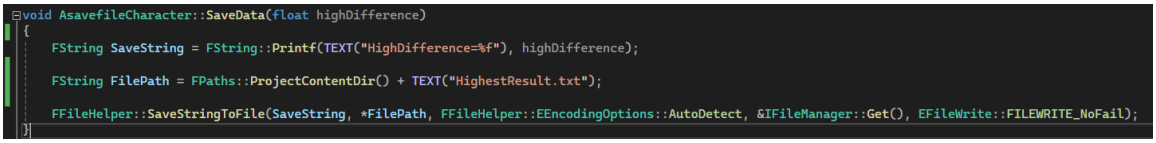
Tick-funktiossa tallennetaan vertikaalisen ja horisontaalisen ero jokaisen framen välillä 150 frameen asti. Mikäli 200 muutosta on tallennettu, niin vanhin poistetaan. Maksimi ruudunpäivitysnopeus asetetaan Unreal Enginen asetuksista 100, jolloin tallennetut muutokset ovat minimissään 1,5 sekunnilta ennen ampumista ja 0,5 sekuntia ampumisen jälkeen. Toimenpiteen jälkeen tehdään vertailu siitä, oliko uudessa vertailussa tapahtunut suurempaa muutosta aikaisempaan ja suurin tallennetaan. Lopuksi arvot asetetaan lähtöarvoihin ja nollataan AngleChanges-array. Kuvassa 6 tallennetaan välimuistiin ampumisen ajankohdalta tiedot ja suoritetaan vertailu aikaisempaan arvoon kulmien muutoksista.

```
void AsavefileCharacter::Tick(float DeltaTime)
{
    Super::Tick(DeltaTime);

    APlayerController* PlayerController = GetWorld()->GetFirstPlayerController();

    if (PlayerController)
    {
        FRotator CurrentRotation = PlayerController->GetControlRotation();
        float VerticalAngleDifference = FMath::Abs(FRotator::NormalizeAxis(CurrentRotation.Pitch - LastFrameRotation.Pitch));
        float HorizontalAngleDifference = FMath::Abs(FRotator::NormalizeAxis(CurrentRotation.Yaw - LastFrameRotation.Yaw));

        TimeAccumulator += DeltaTime;
        if (TimeAccumulator >= -DurationBeforeFiring && TimeAccumulator <= DurationAfterFiring)
        {
            AngleChanges.Add(VerticalAngleDifference + HorizontalAngleDifference);
            if (AngleChanges.Num() > 200)
            {
                AngleChanges.RemoveAt(0);
            }
        }
        if (bIsFiring && TimeAccumulator >= DurationBeforeFiring + DurationAfterFiring)
        {
            for (float AngleChange : AngleChanges)
            {
                MaxCombinedChange = FMath::Max(MaxCombinedChange, AngleChange);
            }
            bIsFiring = false;
            TimeAccumulator = 0.0f;
            AngleChanges.Empty();
        }
        LastFrameRotation = CurrentRotation;
    }
}
```

Kuva 6. C++-toteutus katselukulmien vertailusta.

Testauksen aikana tallennetaan suurin katselukulmien muutos muuttujaan, joka pelin sulkeutuessa tallennetaan .txt-tiedostoon.

Kuvassa 12 funktion ”SaveData”-toteutus, jossa tallennetaan lokaalisti käyttäjän laitteelle .txt-tiedostoon pelisession suurin framejen välinen katselukulman muutos. Funktion ”SaveData” luomisessa hyödynnettiin OpenAI:n ChatGPT-kielimallia, jonka vuoksi kyseisen funktion toteutus onnistui hyvinkin nopealla aikataululla.

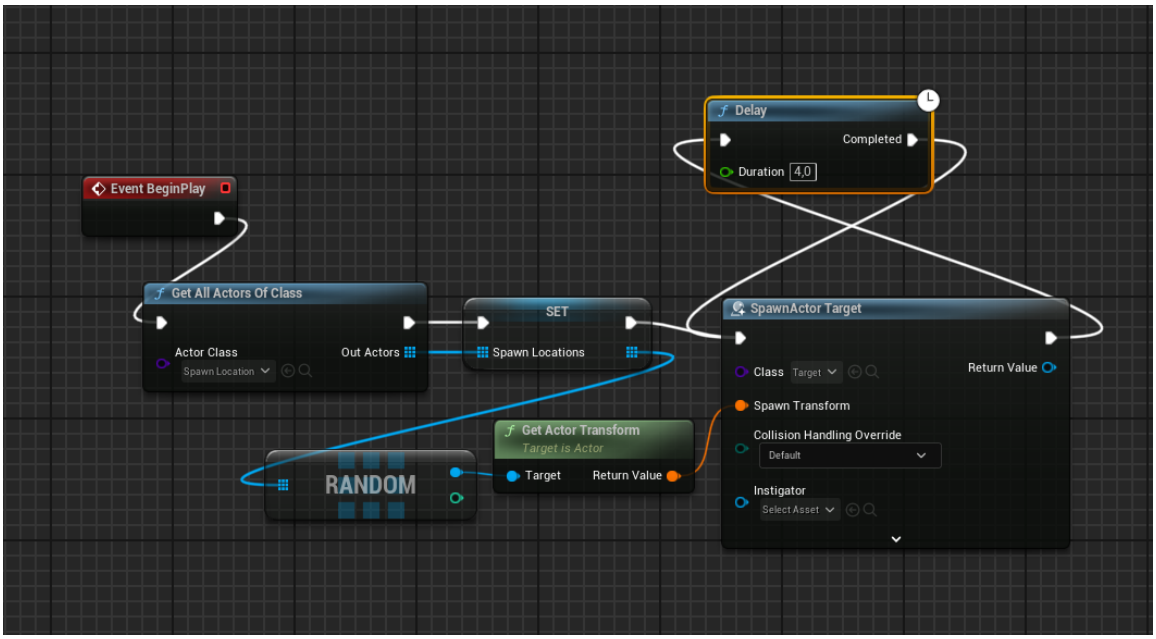


```
void AsavefileCharacter::SaveData(float highDifference)
{
    FString SaveString = FString::Printf(TEXT("HighDifference=%f"), highDifference);

    FString FilePath = FPaths::ProjectContentDir() + TEXT("HighestResult.txt");

    FFileHelper::SaveStringToFile(SaveString, *FilePath, FFileHelper::EEncodingOptions::AutoDetect, &IFileManager::Get(), EFileWrite::FILEWRITE_NoFail);
}
```

Kuva 7. C++-toteutus tiedon tallentamisesta päätelaitteelle.

10 Maalitaulujen luominen ja toiminnallisuus

Maalitauluksi luodaan actori, johon laitetaan collision sphere sekä mesh, jotta maalitaulu näkyy pelaajalle. Lisätään pelikentälle useita tyhjiä objekteja, jotta niiden sijainteihin voidaan sattumanvaraisesti luoda maalitaulu. Maalitauluja luodaan 4 sekunnin välein. Kuvassa 8 blueprint-toteutus maalitaulujen luomisesta.



Kuva 8. Blueprint-toteutus maalitaulujen luomisesta.

11 Testituloksien analysointi

Testiympäristössä suoritettiin muutamia testejä, joista kerättiin tiedot talteen. Tarkastelussa voidaan suoraan huomata, että erot ovat erittäin suuret. Joka osoittaa väitteen todeksi, katselukulmia tarkastelemalla pystytään havaitsemaan epärehelliset pelaajat. Huomioon ottaen kuitenkin, että on olemassa erilaisia huijausohjelmia, joihin kyseinen lähestymistapa ei toimi. Käyttöönottoa varten tulisi myös suorittaa erittäin suuri pelitestaus rehellisillä pelaajilla, jolloin voidaan päätellä, mihin raja voidaan vetää katselukulmien muutoksien suhteen.

Kuvassa 9 nähdään selvästi, miten suuret erot rehellisillä ja epärehellisillä pelaajilla on katselukulmien muutokset.

A	B
Player 1	11.2
Player 2	16.625
Player 3	19.775
Player 4	12.6
Player 5	17.85
Player 6	18.725
Player 7	247.866577

Kuva 9. Rehellisten pelaajien 1–6 ja epärehellisen pelaajan 7 testitulokset.

12 Johtopäätökset

Tutkimuksen tulokset vahvistavat, että tietynlaisen huijausohjelman tunnistaminen on mahdollista pelaajien katselukulmien muutoksien perusteella. Rehellisten pelaajien luontaiset liikkeet rajoittuvat pieniin katselukulmamuutoksiin, kun taas huijausohjelmien käyttäjät näyttävät suurempaa ja epäluonnollisempaa liikettä.

Tutkimuksen pohjalta voidaan kehittää edistyneempiä huijausohjelmien tunnistusmenetelmiä, jotka perustuvat katselukulmamuutosten analysointiin. Lisäksi tutkimusta voitaisiin laajentaa muihin huijausohjelmistoihin, jotta voidaan edistää reilua pelaamista.

Menetelmä osoittautui tehokkaaksi vaihtoehdoksi erottaa rehelliset pelaajat epärehellisistä. Näitä tuloksia voidaan hyödyntää pelinkehittäjien toimenpiteissä huijausten torjumiseksi ja pelikokemuksen parantamiseksi.

Tutkimuksessa ei ole kuitenkaan selvitetty sitä, miten koodinmuokkaus ja erilaiset huijausohjelmien versiot voivat vaikuttaa katselukulmamuutosten analyysiin. Aimbot-ohjelmat kehittyvät jatkuvasti sitä mukaa, kun uusia tunnistusmenetelmiä luodaan.

Testauksessa käytettiin erittäin pientä otoskokoa. Tämä voi vaikuttaa tulosten luotettavuuteen. Laajempi otoskoko olisi voinut tarjota monipuolisemman kuvan pelaajien katselukulmien muutoksista eri tilanteissa.

Vaikka opinnäytetyö tarjoaa arvokasta tietoa huijaamisen tunnistamisesta, siinä on tiettyjä rajoituksia, jotka vaativat huomiota. Lisätutkimusta ja kehitystä tarvitaan erilaisten huijausohjelmistojen tunnistamiseen sekä erilaisten pelaajien pelikäyttäytymisen huomioimiseksi.

Lähteet

1. Cheats Could Ruin Online Gaming – CBS News [Internet]. [Viitattu 18.10.2023]. Saatavilla: https://www.cbsnews.com/news/cheats-could-ruin-online-gaming/

2. A brief history of cheating at video games – Engadget [Internet]. 2019 [Viitattu 19.10.2023]. Saatavilla: https://www.engadget.com/2019-06-15-a-brief-history-of-cheating-at-video-games.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAK-OhMagY8paxxoumnMvicjYQdzJ2JizEZNO4RRypIrOLjg2rLYX5s9oxygwAX96zxC06b19qdNgaVdPAXnBcoHjVVmy2eDXbMDvcEho09vFMevfP7S_eOIC_1FXTvmExocnqhZE5QfaJ_7b6SLnEyJU5uiWzDCFs8Ewno1cYzKX

3. Code Red: The history of the cheat – Red Bull [Internet]. R 2022 [Viitattu 19.10.2023]. Saatavilla: https://www.redbull.com/us-en/the-history-of-the-cheat-code

4. Lehtonen S. Comparative Study of Anti-cheat Methods in Video Games [Väitöskirja]. Saatavilla: https://urn.fi/URN:NBN:fi:hulib-202003241639

5. A History of Video Game Cheats. LevelSkip [Internet]. [Viitattu 21.10.2023]. Saatavilla: https://levelskip.com/community/A-History-of-Video-Game-Cheats

6. Good OS. Watch a Counter-strike pro get caught cheating during a major esports tournament [Internet]. Polygon. [Viitattu 21.10.2023]. Saatavilla: https://www.polygon.com/2018/10/21/18006358/counter-strike-esports-cheating-shanghai-video

7. Vac Stats – ESL Gaming Online [Internet]. [Viitattu 21.10.2023]. Available from: https://csstats.gg/fi

8. Biggest VAC Wave Ever - 1 Million CS: GO Cheaters Banned In 2019 [Internet]. [Viitattu 22.10.2023]. Available from https://www.elecspo.com/games/biggest-vac-wave-ever-1-million-csgo-cheaters-banned-in-2019/

9. GDC 2018: John McDonald (Valve) – Using deep learning to combat Cheating in CSGO [YouTube]. [Viitattu 22.10.2023]. Saatavilla: https://www.youtube.com/watch?v=ObhK8lUfIlc&ab_channel=max4warn

10. Counter-Strike2's Cheater Problem – Hackers Everywhere [Internet]. [Viitattu 27.10.2023]. Saatavilla: https://www.gameleap.com/news/counter-strike-2s-cheater-problem-hackers-everywhere

11. Anti-Cheat Kernel Driver – League of legends [Internet]. [Viitattu 27.10.2023]. Saatavilla https://www.leagueoflegends.com/en-us/news/dev/dev-null-anti-cheat-kernel-driver/

12. Valorant fans voice converns as cheaters are “getting out of hand” – Dexerto [Viitattu 28.10.2023]. Saatavilla https://www.dexerto.com/valorant/valorant-fans-voice-concerns-as-cheaters-are-getting-out-of-hand-2213120/

13. GitHub [Internet]. [Viitattu 28.10.2023]. Saatavilla: https://github.com/Jire/Charlatano/blob/main/src/main/kotlin/com/charlatano/scripts/aim/General.kt#L116

14. What Are Aimbots and How Do They Work [Internet] [Viitattu 26.11.2023]. Saatavilla https://chatbotsjournal.com/what-are-aimbots-and-how-do-they-work-4936794e5453

15. CSGO_Aimbot_Only_Simple – Github [Internet]. [Viitattu 29.11.2023]. Saatavilla: https://github.com/MrLiamMcQ/CSGO_Aimbot_Only_Simple/blob/3f3b0d76bfa0524aa4321ce6ba45853e5c267b9a/Aim-Bot%20Only/dllmain_Aimbot.cpp#L120

16. RICOCHET Anti-Cheat Progress Report – Season 04 Update [Internet]. [Viitattu 2.12.2023]. Saatavilla: https://www.citethisforme.com/cite/sources/websiteautociteeval

17. Yleinen tietojasuoja-asetus (GDPR) – Europa EU [Internet]. [Viitattu 22.3.2024]. Saatavilla https://europa.eu/youreurope/business/dealing-with-customers/data-protection/data-protection-gdpr/index_fi.htm

18. The dangers of in-game data collection – Polygon [Internet]. [Viitattu 23.1.2024]. Saatavilla: https://www.polygon.com/features/2019/5/9/18522937/video-game-privacy-player-data-collection

19. Otos ja otantamenetelmät – Otantamenetelmät – KvantiMOTV [Internet]. [Viitattu 23.1.2024]. Saatavilla: https://www.fsd.tuni.fi/menetelmaopetus/otos/otantamenetelmat.html

20. Aimbot – Game Hacking Academy [Internet]. [Viitattu 24.1.2024]. Saatavilla: https://gamehacking.academy/pages/5/06/s

21. Pythagora’s Theorem-in 3D – Math is Fun [Internet]. [Viitattu 22.3.2024]. Saatavilla: https://www.mathsisfun.com/geometry/pythagoras-3d.html

22. First Person Template – Epic Games Documentation [Internet]. [Viitattu 27.2.2024]. Saatavilla: https://docs.unrealengine.com/5.0/en-US/first-person-template-in-unreal-engine