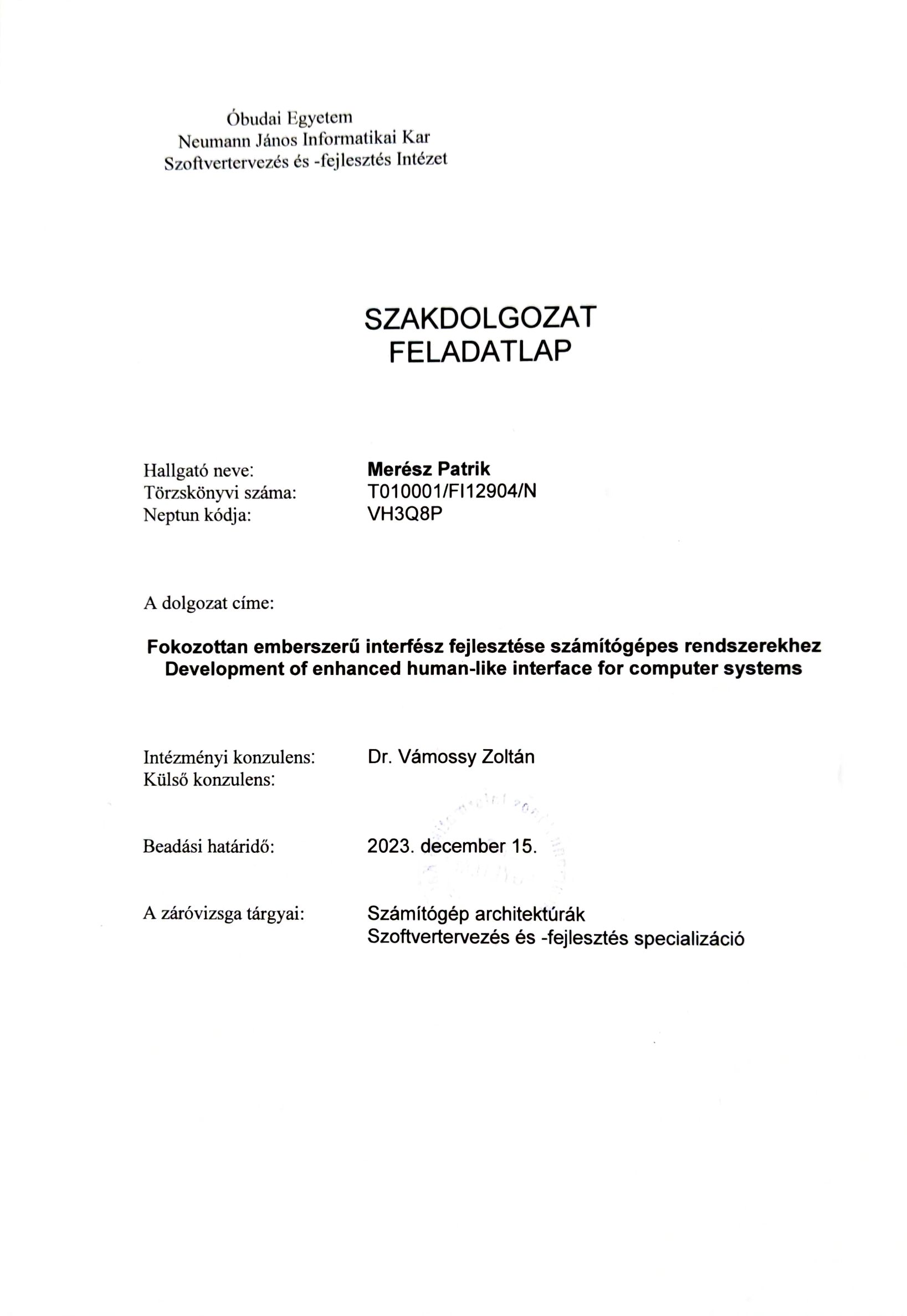
|  |  |  |
| --- | --- | --- |
|  | **NEUMANN JÁNOS**  **INFORMATIKAI KAR** | NIK_cimer.jpg |

|  |  |  |
| --- | --- | --- |
| **OE-NIK**  **2024** | Hallgató neve:  Hallgató törzskönyvi száma: | **Merész Patrik**  **T010001/FI12904/N** |

**SZAKDOLGOZAT**



A képen szöveg, levél, Betűtípus, papír látható

Automatikusan generált leírás

HALLGATÓI NYILATKOZAT

Alulírott hallgató kijelentem, hogy a szakdolgozat / diplomamunka saját munkám eredménye, a felhasznált szakirodalmat és eszközöket azonosíthatóan közöltem. Az elkészült szakdolgozatomban / diplomamunkámban található eredményeket az egyetem és a feladatot kiíró intézmény saját céljára térítés nélkül felhasználhatja.

A képen kézírás, kalligráfia, Betűtípus, kézzel írott látható

Automatikusan generált leírásBudapest, 2024.04.23... ..........................................

|  |  |
| --- | --- |
|  | hallgató aláírása |

A képen szöveg, Betűtípus, papír, levél látható

Automatikusan generált leírás

A képen szöveg, Betűtípus, képernyőkép, kézírás látható

Automatikusan generált leírás

|  |  |
| --- | --- |
|  |  |

Tartalomjegyzék

[1. Absztrakt/Abstract 7](#_Toc165625489)

[2. Bevezetés 9](#_Toc165625490)

[2.1 Motiváció a projekt elkészítéséhez 10](#_Toc165625491)

[2.2 Hasonló projektek 11](#_Toc165625492)

[3. Bot detektálás videójátékokban 12](#_Toc165625493)

[3.1 Viselkedés analízis 13](#_Toc165625494)

[3.2 Heurisztikus módszerek 14](#_Toc165625495)

[3.3 Összegzés 15](#_Toc165625496)

[4. Rendszerterv 16](#_Toc165625497)

[4.1 Követelmények 16](#_Toc165625498)

[4.2 Funkciós lista 16](#_Toc165625499)

[4.3 Felhasznált eszközök 17](#_Toc165625500)

[4.4 Rendszerterv diagram 19](#_Toc165625501)

[5. Implementálás 20](#_Toc165625502)

[5.1 Képfelismerés adatgyűjtés és tanítás 20](#_Toc165625503)

[5.2 Emberszerű egérmozgás 21](#_Toc165625504)

[5.3 Lineáris regresszión alapuló neurális hálózat fejlesztése 24](#_Toc165625505)

[5.4 GAN neurális hálózat fejlesztése 27](#_Toc165625506)

[5.5 GAN neurális hálózat legnagyobb problémája: Mode collapse 34](#_Toc165625507)

[5.6 Program működése 39](#_Toc165625508)

[5.7 Tesztelés és további fejlesztések 40](#_Toc165625509)

[5.8 Egérmozgató rendszerek összehasonlítása 42](#_Toc165625510)

[6. Konklúzió 44](#_Toc165625511)

[7. Köszönetnyilvánítás 45](#_Toc165625512)

[8. Hivatkozások 46](#_Toc165625513)

[9. Ábrajegyzék 49](#_Toc165625514)

# 1. Absztrakt/Abstract

A szakdolgozat tartalmazza azon preventív rendszerek bemutatását, amelyek képesek megkülönböztetni gépi, illetve emberi interakciókat, különös tekintettel a bot detektálás területén a videójátékokban. A dolgozatban kifejtem, hogy milyen következményekkel jár, ha az említett ellenőrzés elmarad, továbbá ezen rendszerek előnyeit és hátrányait fogom taglalni. A szakdolgozat második részében egy olyan szoftver tervezéséről és megvalósításáról lesz szó, amelyről a fentebb említett rendszerek még nem megbízhatóan képesek eldönteni a generált válaszok valódiságát. Az elkészült szoftvert összehasonlítom más ilyen jellegű tanulmányokkal, és egy olyan valós tesztkörnyezetben is ki fogom próbálni, mint például az online számítógépes játékok világa, amelyben a legkorszerűbb bot felismerő rendszereket fogom próbatétel elé állítani. Egy olyan szoftvert fogok fejleszteni, amely úgy mozgatja az egeret, hogy az a lehető legemberszerűbb legyen és amely teljesen automatizálja az egérműveleteket. A program emberi viselkedéséhez egy képfelismerő algoritmust és egy GAN típusú neurális hálózatot fogok használni, ami az egérmozdulatok létrehozásáért fog felelni. A neurális hálózattal a saját egérmozgatási adataimmal feltanítva fogok emberszerű egérmozgást generálni. A dolgozat végén összegzem a fejlesztés menetét és értékelni fogom a projekt sikerességét.

The thesis includes a presentation of preventive systems that can distinguish between machine and human interactions, with a special focus on bot detection in video games. In the thesis, I will explain the consequences of failing to perform such monitoring and the advantages and disadvantages of these preventive systems. In the second part of the thesis, I will discuss the design and implementation of a software for which the aforementioned systems are not yet able to reliably determine the veracity of the generated responses. The software will be compared with other studies of this kind, and will be tested in a real test environment, such as the world of online computer games, in which state-of-the-art bot recognition systems will be put to the test. I will develop a software that moves the mouse in a way that is as human-like as possible and that fully automates mouse actions. For the human behaviour of the program I will use an image recognition algorithm and a GAN-type neural network that will be responsible for generating the mouse movements. I will use the neural network to generate human-like mouse movements by learning from my own mouse movement data. At the end of the thesis, I will summarize the development process and evaluate the success of the project.

# 2. Bevezetés

Az ember-számítógép kapcsolatok (HCI) kutatása egyik kulcsterület napjainkban. Számos olyan alkalmazás jelenik meg, ahol a kommunikációs partner egy számítógép, és az szolgáltat emberszerű válaszokat. Ilyenek például a chatbotok, hangalapú asszisztensek vagy tágabb értelemben vett virtuális ügynökök. Ezek a rendszerek rengeteget fejlődtek az elmúlt pár évben és a korlátjaik terén is jelentős előre lépés történt, köszönhetően az AI és machine learning technológiáknak. Azonban érdemes észben tartani, hogy az alább felsorolt rendszerek még mindig nem tökéletesek és alkalomadtán képesek olyan választ adni, amely messze távol van az elvárt kimenettől. Találkozhatunk olyan esetekkel is amikor ezek a botok rendkívül kártékonyak lehetnek, mivel teljesen megtévesztik a kommunikáló felet és fontos információk kiadását vagy az egyén véleményének befolyásolására használják, illetve olyan előnyre tesznek szert használatával, amivel a felhasználó a bot program nélkül soha nem lett volna önerőből képes.

A bot egy olyan szoftveres alkalmazás, amit arra terveztek, hogy különböző feladatokat oldjanak meg automatikusan, jellemzően azzal a céllal, hogy helyettesítse vagy utánozza az emberi viselkedést. Ezek a feladatok általában ismétlődő jellegűek és a botok ezeket a folyamatokat sokkal hatékonyabban oldják meg, mint mi emberek.

Fontos megjegyezni, hogy a botokat rengeteg területen használják, mind hasznos, illetve kártékony használatra ezért szükségszerű, hogy tisztában legyünk a lehetséges kockázatokkal és sebezhetőségekkel továbbá, hogy megtegyük a megfelelő óvintézkedéseket az okozott károk mérsékléséhez. Az elmúlt években jelentősen megnövekedett a kártékony botok száma ez különösen veszélyes lehet az online közösségi médiában, ahol rengeteg kamu profillal találkozhatunk. Ezek a fiókok úgy vannak tervezve, hogy félre tájékoztassák a felhasználót, amelyekkel befolyásolni tudják a közvéleményt, politikai kampányokat vagy akár a pénzügyi piacokat. A másik szegmens, amely világméretűvé nőtte ki magát és amiről dolgozatom keretein belül fogok foglalkozni az nem más, mint a videó játékok világa. Egyre népszerűbb a játékosok körében, hogy olyan eszközökhöz nyúljanak, amellyel valamilyen előnyre tudnak szert tenni a többi játékossal szemben ezáltal jelentős anyagi károkat okozva a játékfejlesztő cégeknek.

A videójátékokban a legnagyobb kihívás a detektálás és a megelőzés, a nevesebb játékfejlesztő cégek rengeteg pénzt fektetnek abba, hogy olyan rendszereket építsenek ki, hogy azok garantálni tudják a tisztességes játékteret és hogy megőrizhessék játékuk integritását. Ezek a rendszerek úgy működnek, hogy felismerik és megakadályozzák a játékfájlok vagy a memóriában lezajló folyamatok olyan jogosulatlan módosításait, amelyek tisztességtelen előnyhöz juttathatják a játékosokat. Ezen szoftverek, emellett képesek a játékosok viselkedésének megfigyelésére és a csalásra utaló minták, például az aimbotting vagy a wall hacking felismerésére. Kezdetben heurisztikus módszereket alkalmaztak, de ahogy fejlődni kezdtek a különböző csalási metodikák és egyre több adat állt rendelkezésre, úgy egyre jobban megkövetelte az ipar a megfelelő gépi tanulási módszerek bevetését.

A szakdolgozatban lévő kutatásommal arra a kérdésre keresem a választ, hogy mit tudnak kezdeni ezek a rendszerek, akkor, ha a szemben álló program machine learning technológiával támogatva próbálja kijátszani azokat. Röviden összefoglalva a projekt lényegét, egy macska egér játékot kell elképzelni, amiben az egér szerepét én fogom betölteni az általam elkészített szoftverrel, ami videójátékokban fogja automatizálni a célzási műveleteket, mindezt úgy, hogy lehetőleg a macska szerepébe bújt csalásellenes szoftverek vizsgáló módszereit és a többi játékos megfigyeléseinek ellenére észrevétlennek tudjak maradni, annak ellenére, hogy valamilyen segédprogramot használok.

## 2.1 Motiváció a projekt elkészítéséhez

Az elsődleges motivációm a kihívás, olyan program létrehozásán gondolkoztam, amelyet akár több projektben is hasznosítani lehet, amellyel meglehet oldani a repetitív feladatokat és energiát lehet megtakarítani a használatával.

Egy olyan projektet akartam készíteni, amelyben a szoftver specializációban hallgatott tárgyaim közül a lehető legtöbb tudást tudom kamatoztatni. Legyen szó képfeldolgozási alapokról, neurális hálózatok létrehozásáról, használatáról, illetve párhuzamosításról és legutolsó sorban a videókártya által nyújtott gyorsításról. További szempont volt a program megfelelő optimalizálása ugyanis a program teljesítménye kritikus a megfelelő használat szempontjából.

Beszeretném mutatni, hogy AI technológiákat felhasználva képesek vagyunk olyan rendszereket gyártani, amelyekről nem lehet teljes mértékben eldönteni, hogy gép vagy ember gyártotta a megfelelő kimenetet, kifejezetten videójátokra fókuszálva.

A videójáték tesztkörnyezetre azért esett a választásom, ugyanis gyerekkorom óta érdekelt ez a terület és rengeteg időt töltöttem el velük. Ezenfelül rendkívül kíváncsi vagyok, hogy milyen sikereket lehet benne elérni, ha az ember munkáját szoftveresen támogatjuk. Ezen felül kompetitív személyiségűnek tartom magam, gyerekkoromban sok versenyen részt vettem, mind sportban, tanulmányi, illetve e-sport szinten is. Tanulmányaim során nem jut elég idő a gyakorlásra az utóbbi területre, ezért a játékokban való teljesítményem is drasztikusan romlik, ez különösképpen akkor a legbosszantóbb, amikor egy közeli ismerős vagy egykori csapattárs hívja fel a figyelmed arra, hogy mennyire nem vagy formában. Megoldást ebben a projektmunkában látom.

Szeretnék egy olyan programot készíteni, amely valós időben képes detektálni ellenfeleket a képernyőn és a pozíciójukat meghatározva a másodpercek tört része alatt 0.1-0.2 másodpercen belül becélozni azokat. Ezt a gyakorlatban úgy szeretném megvalósítani, hogy a programot számítógépes játékokban, kifejezetten FPS (belső nézetes) típusúakban fogom használni és tesztelni. A mágikus 100-200 közötti ezredmásodperces szám, egy átlagos fizetett profi játékos reakció ideje az ellenfél felismerésétől az első célzott lövés elsütéséig [1]. A projekt másodlagos célja az, hogy mind ezt úgy hajtsam végre, hogy a profilunkat ne érje kitiltás. A projektet két játékon tervezem megvalósítani az egyik a Counter Strike: Global Offensive (CSGO) a másik játék pedig a Valorant lesz. A CSGO-ra az egyszerűség és a kevésbé kifinomult csalásellenes szoftvere miatt esett a választás, a Valorantra pedig a híresen erős csalásellenes szoftvere miatt a választottam. Mind a két játék ingyenes, így a tesztelés közben felmerülő anyagi károk valószínűleg nem lesznek súlyosak.

## 2.2 Hasonló projektek

Az emberi egérmozgás metrikáit már számos területen nagy figyelem övezi, mivel ez az adat jóval több információval tud szolgálni, mint elsőre gondolhatjuk. Ilyen például a felhasználók azonosítása egérmozdulatok alapján [2] vagy demográfia adatokat kinyerése [3] esetleg stressz szint felmérése munkahelyi környezetben [4] ezt mind pusztán az egérmozdulatok vizsgálata alapján megtörténhet.

Az emberszerű egérmozgatás generálásáról kiváló kiinduló pontot nyújt a Sapientia Erdélyi Magyar Tudományegyetem által készített tanulmány, melynek címe SapiAgent: A Bot Based on Deep Learning to Generate Human-Like Mouse Trajectories [5]. Ebben a tanulmányban több autoencoder alapú megoldást készítettek, amivel szintetikus egérmozdulatsorokat generáltak. Azért választották az autoencoder architektúrát, mert a tanító adathalmaz zajosságát akarták csökkenteni ezzel javítva a modell pontosságát és konvergenciájának sebességét.

A videójátékokban való éles tesztelésről nem találtam információt. Az interneten különböző fórumokon kutatva fellelhető pár hasonló megoldás, ami szintén képfelismerést alkalmaz a kívánt objektumok azonosítására, de ezek elég gyengére sikerültek hatékonyság szempontjából. A legalapvetőbb probléma az, hogy az egérmozgatás mechanikája könnyedén detektálható, más részről a teljesítményük is elmarad a kívánttól.

# 3. Bot detektálás videójátékokban

A játékokban használt botok olyan automatizált programok, amely egy adott játék bizonyos részeit vagy a játék teljes egészét játssza egy emberi játékos nevében. Ezek a programok sokkal több játék valutát vagy tárgyat tudnak keresni, mint az emberi felhasználók, mivel az előbbiek szünet nélkül tudnak játszani. Ezen játékrobotok használata illegálisnak tekinthető, használatukért jellemzően felhasználó fiókját éri tiltás, ilyen esetben a felhasználó elveszíti a játék használati jogát és a játékban megszerzett javait. A játékban megszerzett javak egy része jelentős valódi anyagi értékkel bír. A legdrágább digitális tárgy idáig egy bolygó volt, aminek az értéke hatmillió dollárért kelt el a Entropia's universe című játékban [6]. A játékokban lévő botok az emberi felhasználókat is zavarhatják, mivel folyamatosan fogyasztják a játék erőforrásait és ez súlyos játékon belüli inflációt okoznak, továbbá lerövidíthetik a játék életciklusát és rossz hírnevet generálhatnak a játékfejlesztő cégről [7].

A botok detektálásához és felderítéséhez számos tanulmányt mutattak be a tudományos és az ipari életben. Ezek a módszerek a következő kategóriákba sorolhatók: kliensoldali, hálózat, illetve szerveroldali.

A legtöbb játékgyártó cég a játékrobotok elleni elsődleges intézkedésként a kliensoldali észlelési módszereket alkalmazza, amelyek a játékrobotok digitális lenyomatát elemzik. Az ügyféloldali felismerési módszerek a botprogram nevét, a folyamat információit és a memória állapotát vizsgálják elsősorban. Ez a módszer hasonló a számítógépes vírusokat felismerő vírusirtó programokhoz. A kliensoldali észlelési módszereket a tapasztaltabb játékrobotok fejlesztői könnyen megkerülhetik, bizonyos esetekben pedig a számítógép teljesítményét is ronthatják.

A hálózati észlelési módszerek, például a hálózati forgalom figyelése vagy a hálózati protokoll változások elemzése hálózati túlterhelést és késleltetést okozhat a játékokban, ami jelentősen rontja a játékélményt. A kliens- és hálózatoldali észlelési módszerek ezen korlátjainak kiküszöbölésére számos online játékszolgáltató szerveroldali észlelési módszereket alkalmaz.

A szerveroldali észlelési módszerek a játékszerverek naplóadatait elemző adatbányászati technikákon alapulnak. A legtöbb játékszerver eseménynaplókat generál, amikor a felhasználók olyan műveleteket végeznek, mint a vadászat, a gyűjtögetés és a csevegés. Ezért ezek a játékon belüli naplók megkönnyítik az adatelemzést, mint a játékrobotok felderítésének lehetséges módszerét. Az online játékokkal foglalkozó vállalatok a szerveroldalon elemzik a felhasználói viselkedéseket vagy egyéb adatcsomagokat, majd ennek hatására az online játékszolgáltatók szelektíven blokkolhatják azokat a bot felhasználókat, akiket ki akarnak tiltani, anélkül, hogy további programokat telepítenének a kliensoldalon. Ezért a legtöbb online játékszolgáltató a szerveroldali észlelési módszereket részesíti előnyben.

Egyes online játékokkal foglalkozó vállalatok olyan nagy adatelemző rendszer megközelítéseket vezettek be általában gépi tanulási algoritmusok formájában, amelyek adatvezérelt profilalkotást és észlelést alkalmaznak. Az ilyen megközelítések képesek akár, több mint 600 TB játékszerverek által generált naplófájlokat elemezni [7], és nem okoznak semmilyen mellékhatást, például teljesítménycsökkenést vagy konfliktust más programokkal.

## 3.1 Viselkedés analízis

Az egyik leggyakoribb módszer a viselkedés analízis, amely magába foglalja a játékosok viselkedésének elemzését a játékokban, hogy azonosítani lehessen a botok tevékenységére utaló mintákat. Ez a gyakorlatban úgy jelenik meg, hogy azt a tényt feltételezzük, hogy a botok egy előre meghatározott úton közlekednek és ismétlődően ugyan azt az interakciót hajtják végbe, ellenben az emberi felhasználóval, ami kiszámíthatatlanul és változatosabb módon viselkedik. Előfordulhat olyan eset is, amikor anomáliákat keresünk, ilyenkor olyan kirívó adatokat figyelünk, amelyek nem jellemzőek az emberi játékosokra, mint például kiugró erőforrás gyűjtögetési értékek vagy túlzott szint növekedés esetleg több napon keresztül átívelő folyamatos játékmenet [8].

Egy másik módszer a közösségi tevékenységelemzés, amely a közösségi hálózat jellemzőit használja fel az emberi és a játékrobotok megkülönböztetésére [9]. Ebben a módszerben azt a lehetőséget aknázzák ki, hogy a botok jellemzően nem végeznek semmiféle interakciót a többi játékossal, ezáltal rendkívül hatékonyan szétválasztható a két csoport. Az interakciók az alábbiak lehetnek, chatelés, csoport hívások vagy esetleg bármilyen tárgy cseréje más néven üzletelés lebonyolítása.

A fentebb említett módszerekre jellemző, hogy rendkívül nagy mennyiségben kell az adatoknak rendelkezésre állniuk, hogy ezek a technikák hatékonyan működjenek. A legtöbb esetben klaszterező algoritmusokat vagy statisztikai eszközöket használnak. Egy másik kihívás, amellyel az adatelemzési módszerek szembenéznek az a számításigényesség, különösen az óriási mennyiségű adatok kezelésénél. Ezért fontos, hogy a játékfejlesztők optimalizálják az adatelemzési technikákat, hogy azok észszerű idő alatt elvégezhetők legyenek és hogy a megfelelő intézkedés a játékos irányába minél rövidebb idő alatt megtörténjen.

## 3.2 Heurisztikus módszerek

A heurisztikus elemzés a botok észlelésének olyan módszere, amely a gyanús viselkedésminták vagy műveletek azonosítását foglalja magában, amelyeket valószínűleg botok hajtanak végre. Ez a technika előre meghatározott szabályokon vagy heurisztikákon alapul, amelyeket a bottevékenységre utaló viselkedések azonosítására használnak.

Például az egyik heurisztika, amelyet a játékban lévő botok felismerésére lehet használni, a játékosok mozgásának sebessége. Ha egy játékos természetellenesen gyors sebességgel mozog, ami nincs összhangban a játék fizikájával, az jelezheti, hogy valamilyen csalásra alkalmas programot használ. Hasonlóképpen, ha egy játékos ugyanazt a műveletet ismétlődően, tökéletes időzítéssel hajtja végre, az is jelezheti, hogy botot használ [10].

A heurisztikus elemzés hatékony lehet bizonyos típusú botok azonosításában, például azokéban, amelyek bizonyos típusú viselkedési mintákra vagy a játékban található kihasználható résekre támaszkodnak. Ugyanakkor hamis pozitív eredményeket is produkálhat, mivel egyes legitim játékosok hasonló viselkedési mintákat mutathatnak, ez különösen igaz például a Counter-Strike 2 nevű játékra, ahol már azért is kitiltás jár, ha a játékos magas egér dpi beállításokkal túl gyorsan mozgatja az egerét. Ezért fontos, hogy a játékfejlesztők gondosan meghatározzák heurisztikáikat, és alaposan teszteljék azokat a hamis pozitív eredmények minimalizálása érdekében.

A heurisztikus elemzés egyik előnye, hogy valós időben végezhető, ami lehetővé teszi a játékfejlesztők számára, hogy gyorsan azonosítsák és eltávolítsák a botokat a játékból. A heurisztikus elemzés önmagában azonban nem biztos, hogy elegendő a botok minden típusának felismeréséhez, mivel a fejlettebb botokat úgy tervezték, hogy elkerüljék a heurisztikus módszereken alapuló felismerést.

Összefoglalva a heurisztikus elemzés hasznos technika a videójátékokban használt botok észlelésére, mivel előre meghatározott szabályok vagy heurisztikák alapján gyorsan azonosítani tud bizonyos típusú botokat. Ugyanakkor hamis pozitív eredményeket is produkálhat, és nem biztos, hogy hatékony a fejlettebb botok felderítésére. Ezért a játékfejlesztőknek a heurisztikus elemzést más botfelismerési technikákkal együtt kell használniuk annak biztosítására, hogy a botokat felismerjék és eltávolítsák a játékból.

## 3.3 Összegzés

A botok felderítése elengedhetetlen a jelenlegi videójáték iparban, biztosítani kell a tisztességes játékmenet fenntartását, a játékosok elégedettségének növelését, a gazdaság és a monetizációs rendszerek védelmét, a biztonság fokozását, a közösség bizalmának erősítését, valamint a játékfejlesztők pénzügyi javainak érdekét. Egy fejlett botfelismerő rendszer a fentebb említett technikákat ötvözve használják. Mindegyik módszernek meg van a saját előnye és hátránya. A legnagyobb problémát az jelenti, hogy maximalizálják a pontosságot és minimalizálják a fals pozitív eredményeket, továbbá, hogy ne rontsák a játék teljesítményét. A felhasznált bot detektáló rendszerek kombinációját jelentősen meghatározza, hogy milyen a játék felépítése és a botok fejlesztésének fejlődő jellegétől is nagymértékben függ. Például egy modern mélytanuló algoritmuson alapuló bot detektálásához egy teljesen más eljárású rendszerre van szükség, mint a korábbi csalási módszerek kiszűréséhez. Fontos a folyamatos fejlődés hangsúlyozása és az elmúlt évek tapasztalataiból kiindulva elmondható, hogy ez a folyamat egy soha véget nem érő macska-egér játék.

# 4. Rendszerterv

Az alábbi fejezetben fogom bemutatni a program hardveres és szoftveres elemeinek összetételét, valamint a rendszer működési logikáját. Ez magában foglalja a rendszer célját, a felhasználói követelményeket, a rendszer architektúráját, a szoftver és hardver specifikációkat, a tesztelési és implementációs terveket.

## 4.1 Követelmények

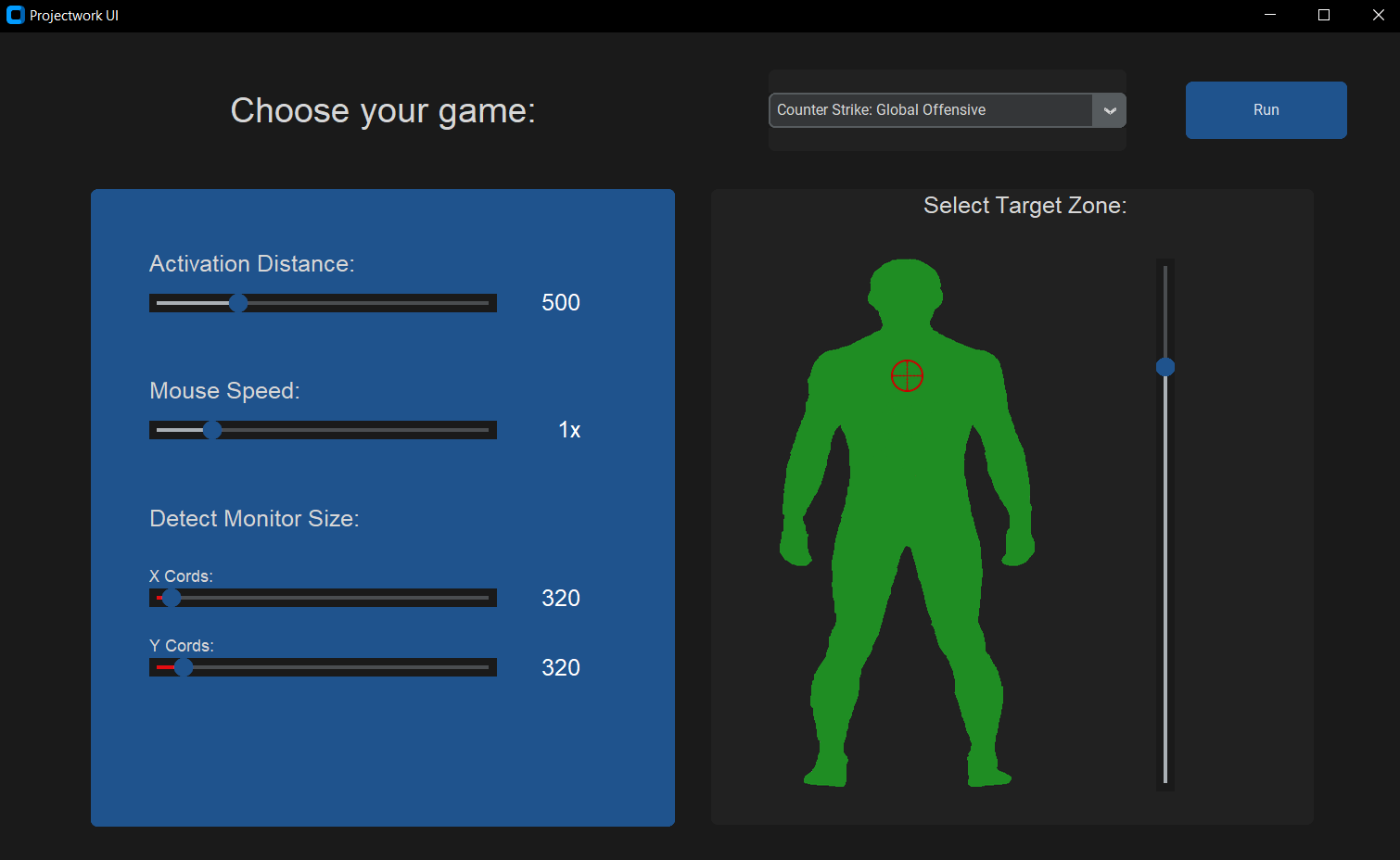
A képfelismerésnek elsősorban gyorsnak és pontosnak kell lennie. Képesnek kell lennie különböző felbontású képek kezelésére is, beleértve a nagy és kis felbontású képeket is. Az algoritmusnak olyan gyorsnak kell lennie, hogy valós idejű alkalmazásokban is tudjon megfelelően működni.

Az egérmozdulat generáló neurális hálózatnak emberszerű egérmozdulattal kell visszatérnie, ami azt jelenti, hogy kerülnie kell az egyenes vonalakat és az irányváltoztatásoknál sem látszódhat semmi gépre utaló jel, továbbá a rendszer az egérmozdulat generálását a másodpercek tört része alatt képes legyen elvégezni. Az emberszerű egérmozgás generálásához a neurális hálózatnak, meg kell tanulnia az egérmozdulat sorokban lévő jellegzetességeket, gyorsulásokat, lassulásokat, pusztán a megfelelő egér pálya lekövetése nem megfelelő.

A tesztkörnyezetben az elkészített szoftvernek valós időben nagy pontossággal fel kell tudnia ismerni a betanított objektumokat a képernyőn. Rendelkeznie kell egy grafikus interfésszel, amellyel a felhasználó személyre szabhatja a program működését. A rendszernek megfelelően le kell kezelnie azt, ha egyszerre több ellenfél is tartózkodik a képernyőn. Az egérmozgató funkciónak emberszerűnek kell tűnnie. A használatáért a játékos fiókját nem érheti kitiltás, kivételt képez ez alól, ha a programot olyan paraméterekkel indítják el, hogy a rendszer a maximális teljesítményét kihasználva működtetik. A rendszer kizárólag Windows környezetben működik. Képesnek kell lennie olyan eszközön is működni, ami nem rendelkezik grafikus gyorsítóval.

## 4.2 Funkciós lista

A szoftverben szabadon paraméterezhetőek az alábbi funkciók: játék kiválasztása, célpont területének kiválasztása, a képernyőn vizsgált terület meghatározása, az egérmozgatási funkció távolsága és sebességének állítása. Ezeket a beállítási lehetőségeket egy grafikus interfészben lehet állítani. A játékot kiválasztva a program betölti az adott játékhoz tartozó képfelismerési modellt. A felhasználónak lehetősége van kiválasztani, hogy a célpont objektumnak melyik részét kívánja célozni. A grafikus interfészben (Lásd 1. ábra) beállítható a képernyőn vizsgált terület mérete, amely a képernyő közepén van elhelyezve. Ebben a rész ablakban számolja ki a képfelismerő algoritmus a releváns értékeket. A program lehetőséget ad a használónak, hogy megváltoztassa az egérmozgatás paramétereit, mint például a távolságot, amelyen belül a program aktiválódik ez az érték megjelöli azt az euklideszi távolságot az egér kurzor és a célpont között, amelyen belül az egérmozgató automatika működésbe lép. További opciót nyújt a program arra, hogy az egér sebességét változtatni tudjuk. Az egérmozgatás sebessége egy szorzó, amely megváltoztatja az egér egy tick alatt maximálisan végrehajtható pixel távolságot a szorzó szorosával. A programban alapértelmezetten ez 10 pixel.



1. ábra Grafikus felhasználói interfész

## 4.3 Felhasznált eszközök

Ebben a fejezetben bemutatom a felsorolt technológiákat és az eszközöket, valamint azt, hogy milyen szerepük és jelentőségük van ebben a projektben.

A megvalósítandó programot Python nyelvben fogom megírni. A Python programozási nyelv 3.9-es verzióját fogom használni, mivel a projekt készítése közben ez a verzió támogatja megfelelően a PyTorch és CUDA könyvtárakat. A programban az egérvezérlést az a Win32Api könyvtárral fogom végrehajtani, ami lehetővé teszi a Windows operációs rendszer API-jának közvetlen hívását. A képernyőmentéseket Mss (Multi Screen Shot) nevű könyvtárral fogom megoldani. A tanítóhalmaz adatkezelését Pandas és Numpy-val valósítom meg.

A Python programozási nyelvre azért esett a választás mert rengeteg hasznos keretrendszerrel van ellátva, továbbá rendkívül sok dokumentáció lelhető fel hozzá. Gyorsan lehet vele dolgozni, nem kell szenvedni az alap dolgok implementálásával és könnyen olvasható a kódja. A Pythont előszeretettel használják machine learning algoritmusok írására és más feladatok automatizációjára, amire pont nekem is szükségem van. Egyetlen egy hátránnyal szembesültem a Python használata alatt az pedig a sebesség, de ez egy kis párhuzamosítással és a beépített adatszerkezetek, vagy az általam megírt függvények helyett Numpy könyvtár használatával jelentős sebesség növekedést tudtam elérni és így már a teljesítmény jelentősen javult.

A neurális hálózatokat PyTorch [11] keretrendszerrel szeretném készíteni, mert használata egyszerű és könnyű benne szerkeszteni a neurális hálózat architektúráját. Nyílt forráskódú ezért a használatáért nem kell fizetni. Rendelkezik videókártya gyorsítással amire a tanítás során és a modell kiértékelésénél vettem nagy hasznát. A Pytorch-nak nagy és aktív felhasználói közössége van, ami az én esetemben azt jelentette, hogy könnyen találtam megfelelő forrásokat, illetve kaptam segítséget, ha elakadtam, további nagy előnye, hogy rendelkezik debug funkcióval, ami megkönnyítette a hiba keresést és azok megtalálását.

Az objektumok detektáláshoz a Pythonhoz megírt Yolov5 [12] algoritmust használom, amit azért választottam ki mert rendkívül magas pontosságot és teljesítményt ígért. A Yolov5 egy nyílt forráskódú mélytanuláson alapuló megközelítés, ami a képeken vagy videókon lévő objektumok felismerését teszi lehetővé, amely különböző számítógépes látási feladatokhoz, például önvezető autókhoz, robotikához és biztonsági rendszerekhez használnak. Az egyik legnagyobb előnye, hogy saját adathalmazra is betanítható és, hogy képes lekezelni a TensorRT [13] modelleket is.

A Mathplotlib nevű könyvtárral az adatvizualizációs problémákat oldom meg. Ezzel a könyvtárral fogom készíteni a tanulási folyamatokat bemutató ábrákat, további az emberi egérmozdulatokat is ezzel a könyvtárral fogom grafikusan megjeleníteni, illetve a feltanított modellek kimenetét.

A Pythonba közvetlenül beépített GUI keretrendszerét a Tkintert fogom használni, hogy legyen egy kézzel fogható interfész, amelyen keresztül belehet állítani a program paramétereit és amelyből egyszerűen kezelhető a program indítása vagy leállítása.

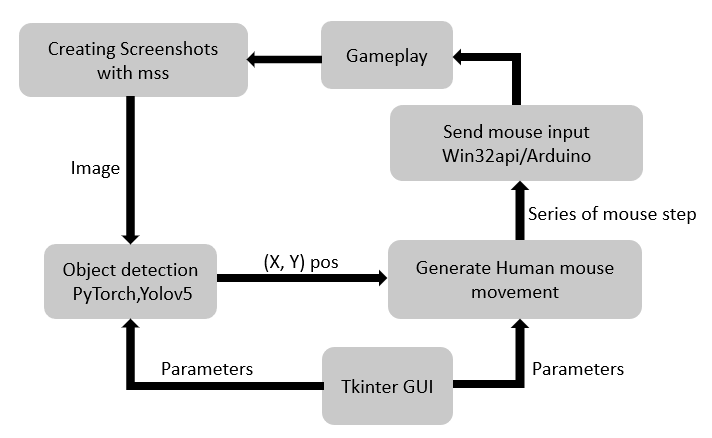
A Win32Api könyvtár azért felel, hogy a szoftverben kiadott egérmozgatási parancsok globálisnak hassanak, magyarán szólva, hogy a játékban is megjelenjenek. Sok könyvtárral ellentétben, mint például a PyautoGui és a PyMouse melyek egérmozgatási parancsai DirectX környezetben már nem működnek. Egyetlen egy gond van vele, hogy már a Valorant nevű játék képes kiszűrni. A másik hátránya, hogy ez a könyvtár csak windows számítógépen működik és így a programom nem tud működni más operációs rendszeren

Az Mss könyvtár egy nagyon gyors platformfüggetlen képfelvevő könyvtár [14], megfelelő hardverrel képes akár a 60fps fölötti gyorsasággal rögzíteni a képernyő tartalmát. Leggyakrabban automatizált tesztek és videófelvételekre használják.

## 4.4 Rendszerterv diagram

Az alábbi rendszerterv diagram (Lásd 2. ábra) ábrázolja a rendszeren belüli komponenseket, kapcsolataikat és az adatáramlást. Segíti a fejlesztés menetét azáltal, hogy egy egyszerű összefoglaló képet ad a rendszer működéséről.

A rendszer kiindulási pontja a grafikus felhasználó interfész, ahol a program paramétereit tudjuk változtatni. A paraméterekkel képesek vagyunk kiválasztani a megfelelő képfelismerő modellt és az egérmozgató algoritmus bizonyos a beállításait módosítani. Ezután a rendszer egy ciklikus folyamatba lép.



2. ábra Rendszerterv diagramm

# 5. Implementálás

## 5.1 Képfelismerés adatgyűjtés és tanítás

A projekt megvalósításának az első és legfontosabb lépése az adatgyűjtés, első teendőm az volt, hogy képeket gyűjtsek a felismerni kívánt objektumokról ezt a problémát úgy oldottam meg, hogy írtam egy scriptet, ami képernyő felvételeket készít a monitor tartalmáról. Kezdetben ezt úgy próbáltam kivitelezni, hogy a program bizonyos időközönként például 5 másodpercenként képet készít. De ezzel a megközelítéssel az volt a gond, hogy nagyon sok irreleváns kép keletkezett. Végül annál a megvalósításnál maradtam, hogy egy gomb lenyomására készítse a felvételeket. Ezt a kis programot játék közben futtattam és a bal egérgombbal aktiváltam a képek készítését. Amikor meguntam a játékot, ugyan ezt a tevékenységet videókon alkalmazva végeztem, mindig, amikor egy ellenfél tűnt fel a képernyőn kattintottam. Két hónap kitartó munkája alatt sikerült 2137 képet gyűjtenem és megfelelően felcímkézni azokat. A képek címkézést LabelImg nevű szoftverrel végeztem (Lásd 3. ábra). Ez a folyamat az adatgyűjtés legidőigényesebb része, de szerencsére csak egyszer kell megcsinálni, illetve annyiszor ahány játékra megszeretnénk írni programot. A képek megcímkézésére azért van szükség, hogy a Yolov5 modellt supervised vagy más néven felügyelt módon tanítsuk.

|  |  |
| --- | --- |
| A képen Számítógépes játék, Videojáték-szoftver, 3D modellezés, képernyőkép látható  Automatikusan generált leírás |  |

3. ábra Címkézés menete

A supervised learning magyarul felügyelt tanulás a gépi tanulás egyik alkategóriája, amiben címkézett adathalmazokat használnak olyan algoritmusok betanítására, amelyek adatok osztályozására vagy eredmények pontos előrejelzésére szolgálnak. Ez úgy történik, hogy a bemeneti adatokat betáplálják egy modellbe, ami addig módosítja a súlyokat, amíg a modell megfelelően nem illeszkedik az elvárt eredményhez [15].

A Yolov5 modellt a dokumentációjában [12] leírt instrukciókat követve sikerült feltanítanom Google Colabon a saját adathalmazommal, mind össze két órán keresztül tartott a tanítás egy P100-as videókártyát használva, mivel nekem a teljesítmény elsődleges szempont volt ezért a legkisebb modellt választottam, ami az „S” névre hallgat. Ez a modell a gyártó szerint egy V100-as videókártyával képes az átadott 1280 pixel széles képen 4.3 milliszekundum körül felismerni a megfelelő objektumot és visszaadni körbe író téglalap pozícióját. Természetesen ez a sebesség nagyban függ attól, hogy az átadott kép mekkora. A legjobb hatékonyságot 320x320-as képeken sikerült elérnem, ennél kisebb kép méretbe már nem férnek bele a felismerni kívánt objektumok. A nagyobb képméretnél a felismerés ideje nőtt drasztikusan. Egy 1920x1080-as képen 50 milliszekundum körüli értékeket kaptam. A képfelismerő algoritmus kimenete alapján könnyen kitudjuk számolni a felismert célpont középpontját, amire a célzást fogom implementálni.

## 5.2 Emberszerű egérmozgás

Szakdolgozatomban az emberszerű egérmozgást több fajta megközelítéssel szeretném megvizsgálni és összehasonlítani, először is a Sapientia Erdélyi Magyar Tudományegyetem által készített tanulmány alapján, melynek címe SapiAgent: A Bot Based on Deep Learning to Generate Human-Like Mouse Trajectories [5] a későbbiekben csak SapiAgent néven fogok hivatkozni rá. Ebben a tanulmányban arra a kérdésre keresték a választ, hogy hogyan lehet emberszerű egérmozdulatsort generálni, illetve kellő pontossággal felismerni azt, kiszűrve a botokat csak az egérmozdulatsorokra alapozva. A tanulmány arra is kitért, hogy az ő általuk készített adathalmaz első sorban azt a célt szolgálja, hogy a jelenlegi, illetve későbbi bot detektáló rendszerek továbbfejlesztését, javítását biztosítsa.

A SapiAgent 128 felhasználó egérmozgatási szokásaiból lett betanítva, fontos kiemelni, hogy ez a tanulmány három és öt perces időintervallumban vizsgálta a résztvevőket. A felhasználók 18 és 53 év közötti korosztályból származnak és a felmérésben részt vettek jobb és balkezes emberek is. A méréseket megközelítőleg 60Hz-es frissítéssel monitorozták és az egyes lépéseket egy fix méretű tömbbe mentették, amely 128 elemet tartalmaz, ha egy mozdulatsor ennél több lépésből állt akkor egyszerűen nem mentették el a felhasználó mozdulatait és ha a résztvevő fél kevesebb lépésből megoldotta a megadott egérműveletet akkor a létrejött tömböt addig növelték 0-s tagokkal ameddig 128 méretű lett. A mérés diverzitását tovább fokozza, hogy több fajta egérmozdulat sort vettek számításba, azon felül, hogy vizsgálták az elmozdulást, külön eseményeket is figyelembe vettek, mint például drag and drop, jobb és bal klikk lenyomásokat. Az összegyűjtött adatokból több neurális hálózatot tesztelve próbálták a legeredményesebb adathalmazt generálni. A legjobb eredményt egy sajátos módszerrel betanított konvolúciós autoenkóderrel érték el [5].

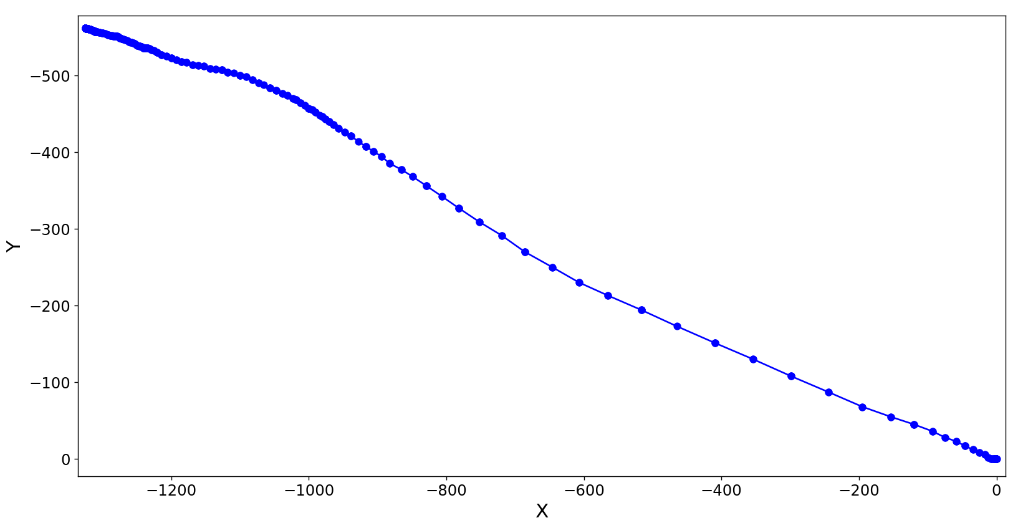
Összegezve a SapiAgent tanulmányt, a SapiAgent eltér a hagyományos tanítási megközelítésektől és komoly sikereket értek el a felismerésekkel összefüggésben. A tanító adathalmaz 120 alanytól származik és a botok észlelését anomália detektálásként fogalmazták meg. A SapiAgent által generált szintetikus egérmozdulatok sokkal valósághűbbek ez eddig ismert Bézier-görbékhez és más hagyományosan feltanított autoenkóderekhez képest.

A másik megközelítést saját magam szeretném megvalósítani, illetve implementálni GAN neurális hálózatokkal. Az adatokat a saját egérmozgatási szokásaimból szeretném felépíteni és bizonyos szempontból egy jóval egyszerűbb, de másrészről sokkal precízebb környezetet szeretnék biztosítani a méréshez, mint az előbb említett SapiAgent tanulmányban, ugyanis a szakdolgozatomban, nekem nem lesz szükségem drag and drop vagy más klikkelési szokásokra szimplán csak az egérmozdulások lesznek mérvadóak.

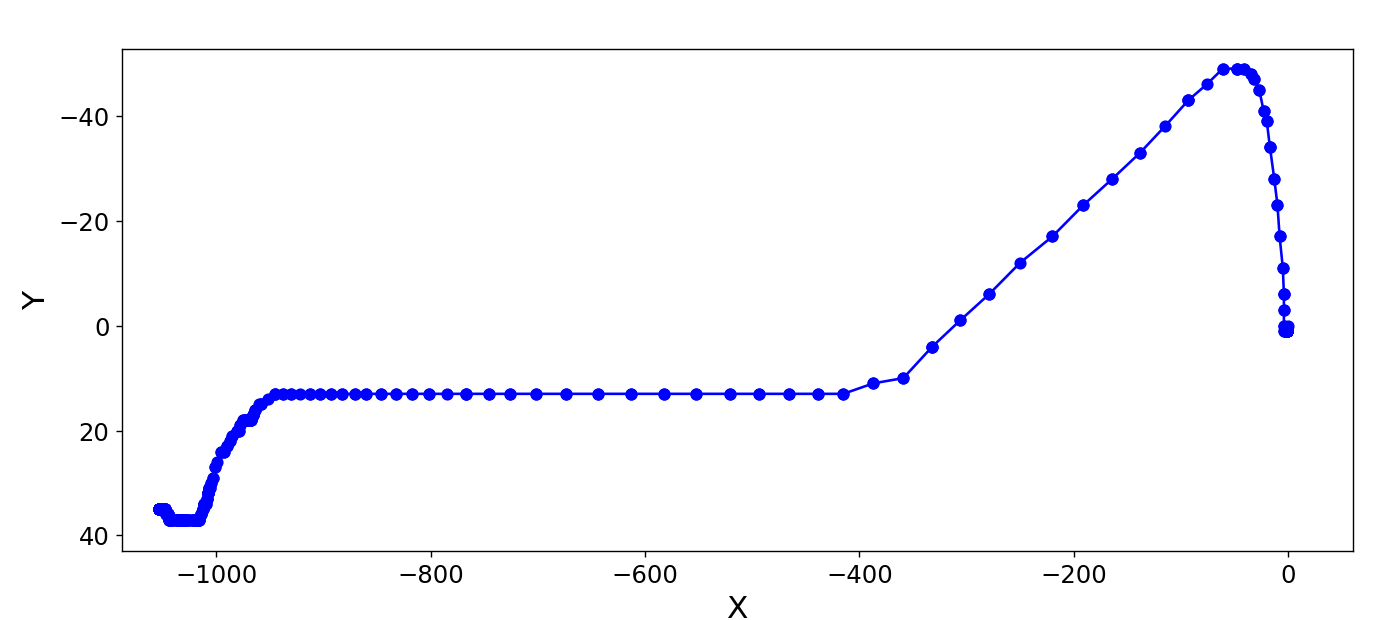
Hogy minél tisztább képet kapjak a valós emberszerű egérmozgásokról az én adathalmaz generáló rendszeremben jóval nagyobb 1000Hz-es frekvencián fogom mérni az egérműveleteket és egy sokkal nagyobb méretű adathalmaz tömbbe fogom eltárolni az adatokat. Erre azért lesz szükségem, hogy az elmozdulások mellett egy részletesebb képet kapjak az elmozdulások időbeliségéről és a sebesség változásairól.

Az adathalmaz generáló rendszerről tudni érdemes, hogy a Python Pygame könyvtárával készítettem és úgy működik, hogy a képernyőn egy véletlenszerűen generált ponton megjelenik egy pont, amelyre kattintva eltűnik majd egy újabb pozícióban megjelenik. Ezen folyamat közben a háttérben elmentjük az összes keletkezett egér elmozdulást és a program bezárásával az összegyűjtött elmozdulásokat kiírjuk egy csv-be. A kiírt elmozdulások tartalmazzák az egér útvonalának kezdő és vég koordinátáit és a köztük végbe menő lépéseket.

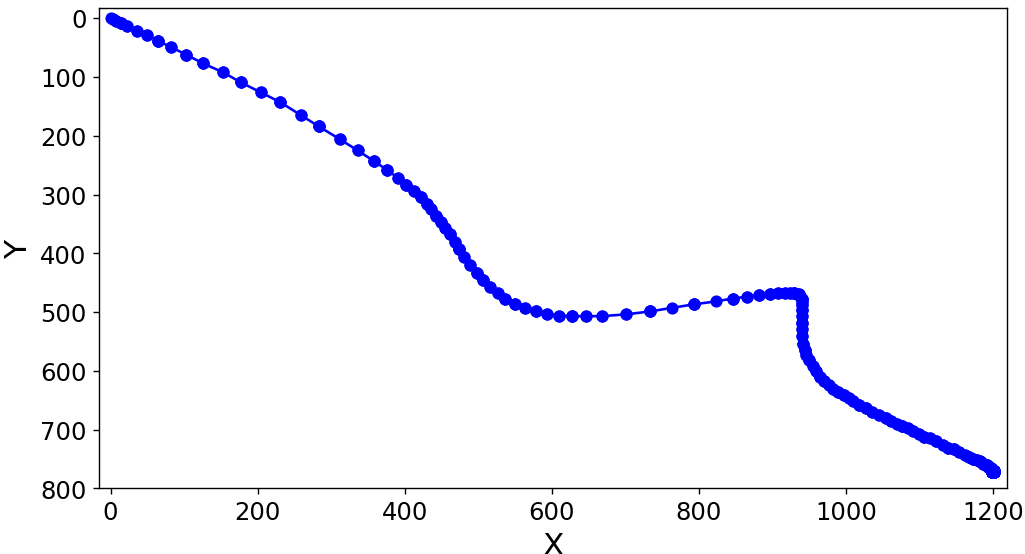
Az összegyűjtött adathalmazból megállapítható, hogy egy átlagos egér elmozdulás 301.14 lépésből áll, ebből az értékből következtetni, tudunk, hogy egy emberszerű egérműveletnek átlagosan 0.3 másodperc alatt meg kell történnie, természetesen ennek a mértéke nagyban függ attól, hogy mekkora távolságot fed le az egér útvonala, de statisztikailag az előbbi állítás tapasztalható. A felmért adathalmazból érdekesség lehet, hogy a legnagyobb lépés 125 pixel két mintavételezés között az átlag pedig 6.64 pixel volt. Ezeket a mért eredményeket később össze tudjuk hasonlítani a neurális hálózat által kapott értékekkel. Az alábbi képeken (Lásd 2. ábra és tovább) egy-egy mérési eredmény látható. A mérések a 0,0 pontból indulnak. Jelenleg az adathalmaz 11030 ehhez hasonló mozdulatsorból áll (Lásd 4-7. ábra).



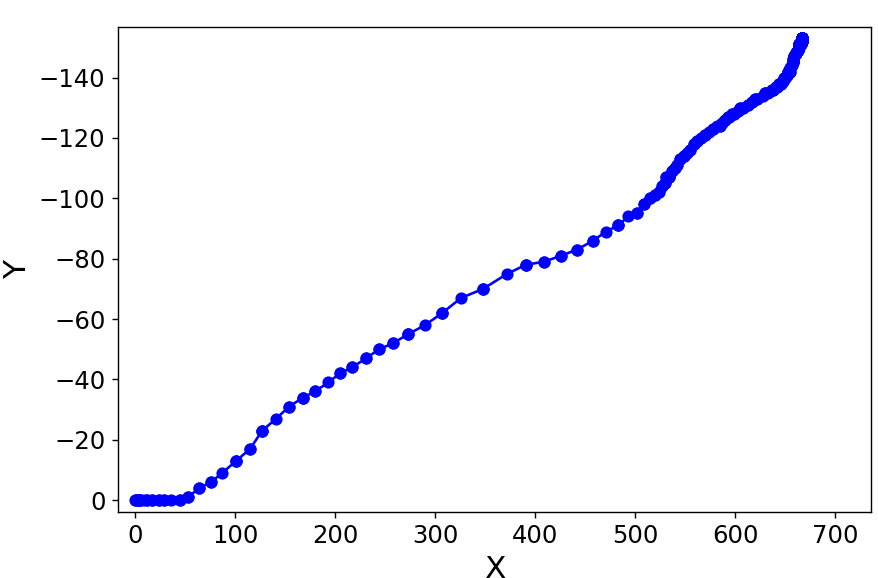
4. ábra Saját egérmozdulat mérési eredménye



5. ábra Saját egérmozdulat mérési eredménye

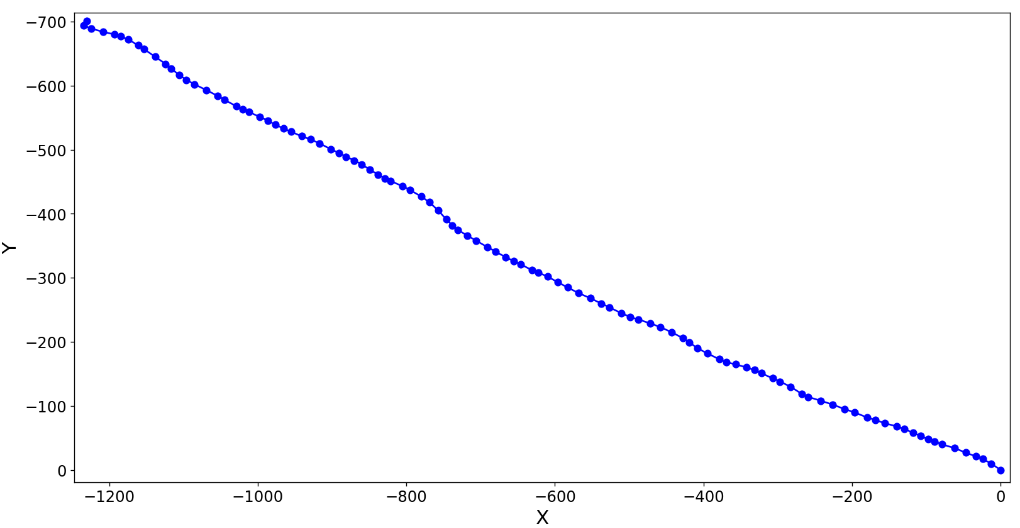


6. ábra Saját egérmozdulat mérési eredménye



7. ábra Saját egérmozdulat mérési eredménye

Mivel a szakdolgozatom írása közben többször felmerült bennem az a gondolat, hogy ágyúval lövök verébre amiért az emberszerű egérmozgás generálását neurális hálózattal oldom meg. Ennek következményeként szeretném a keletkezett eredményeket összehasonlítani egy harmadik módszerrel, ami egy jóval egyszerűbb algoritmussal oldja meg ezt a problémát, ezt az algoritmust a szerzője WindMouse [16] névre keresztelte. Ez a program lényegében egy 50 kódsorból álló függvény, amely középiskolai fizika ismereteket felhasználva random zajjal generál emberszerű egérmozdulatot. Az eredmény az alábbi ábrán látható (Lásd 8. ábra).



8. ábra WindMouse által generált egérmozdulat

Az ábrán megfigyelhető, hogy ez a megoldás ugyan formavilágában életszerű egérmozdulatot generál, de közel sem tartalmazza azokat a tulajdonságokat, amivel egy valódi emberi egérmozdulat rendelkezik, viszont fontos kiemelni, hogy a Counter Strike: Global Offensive nevű videójáték, már ilyen típusú szintetikus egérmozdulatra sem volt képes kitiltani a felhasználót.

## 5.3 Lineáris regresszión alapuló neurális hálózat fejlesztése

Az első neurális hálózat, amivel próbálkoztam, lineáris regresszión alapul, mivel a megvalósítandó problémát úgy képzeltem el, hogy egy adott ponthalmazt kell összekötnöm egy képzeletbeli vonallal csak azzal a megkötéssel, hogy az egyenes ne az egész ponthalmazra illeszkedjen, hanem csak egy kisebb szeletére. Főleg azért választottam ezt a típusú neurális hálózatot, mert az órákon sokat tanultam róla és az implementálása meglehetősen egyszerű.

A modellt úgy implementáltam, hogy inputként egy koordinátapárt várjon (x,y)-t, ami egy irányvektornak feleltethető meg. Kimenetként pedig egy olyan (Δx, Δy) elmozdulást vártam, ami a tanító adathalmazból a legnagyobb fitnesz értékű eredményt adja.

A gyakorlatban a működést úgy kell elképzelni, hogy a feltanított modellnek átadok egy (100,100) koordináta párt, majd a kimenet visszaadja a legjobb elmozdulást, ami (0,0) pontból a (100,100) pontba mutat, ami legyen jelen esetben (9,8) ezt a kimenetet kivonom az előző bemeneti paraméterből. Most a modellnek a (100-9,100-8) pontra kell megkeresni a legjobb elmozdulást és ezt a folyamatot addig folytatom egy ciklusban ameddig a kimeneti elmozdulások összege nem lesz egyenlő a megadott kezdőponttal vagy is a (100,100) ponttal.

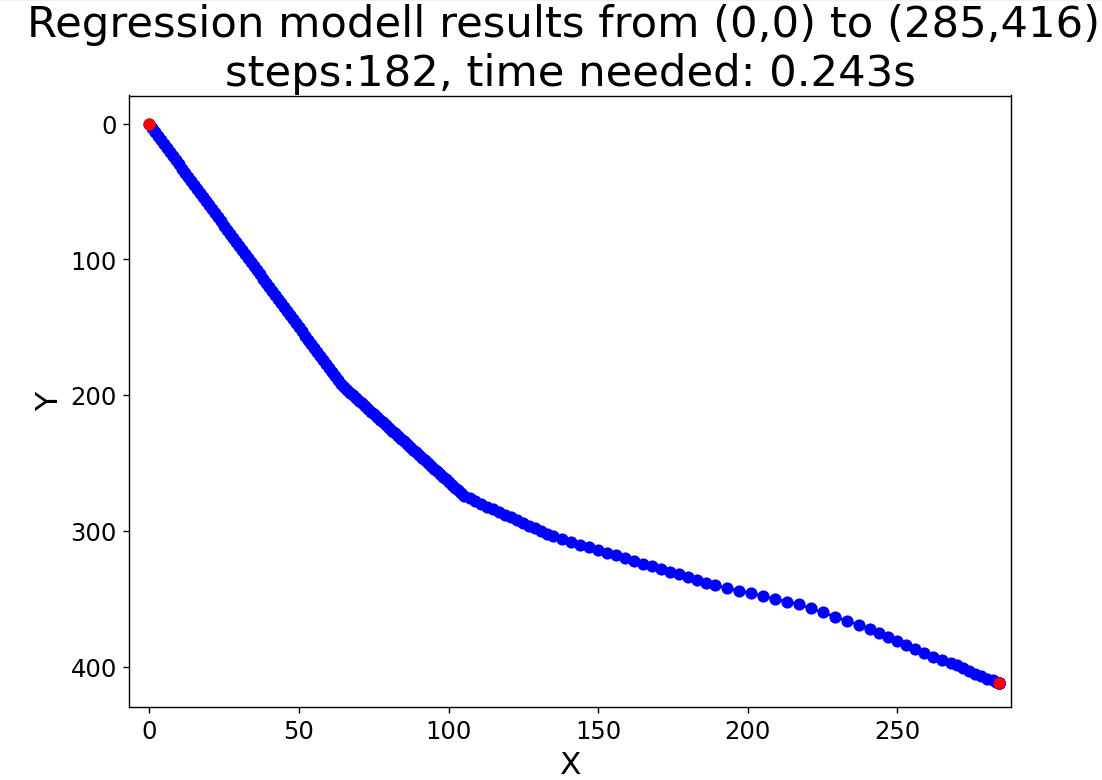
Ennek a megvalósításnak vannak előnyei és hátrányai is. Az előnyök közé felsorolnám azt, hogy a viszonylag kevés tanító adathalmazzal is jól működött a modell és, hogy a generált egérmozdulatok első ránézésre tényleg emberszerűnek hatnak, de ez csak olyan esetekre igaz, amikor a bemenethez létezik hasonló tanító minta. Továbbá kiemelném a többi megvalósítással ellentétben, hogy ez a modell képes olyan mozdulatsor halmazt generálni, amelynek elemszáma viszonyul a megtett távolsághoz, röviden megfogalmazva rövid egérmozdulathoz, kis lépésszámú kimenet tartozik és az eredmény pontosan oda mutat, amelyre a bemeneti paraméter jelzi. Viszont a modell rendkívül érzékeny a túltanulásra és ebből adódóan könnyen megeshet, hogy a kimenet egy szimpla egyenes vonal lesz.

A hátrányok közé továbbá megemlítem azt, hogy a modell elveszíti az emberre jellemző randomitást és a tipikus jellemzőket. A modell egy bemenetre csak egy útvonalat képes generálni. Ezzel szemben, ha egy embert megkérünk arra, hogy mozgassa az egeret 100 pixelen keresztül egy adott irányban, akkor minden próbálkozás során hasonló, de egy másik megoldást fogunk kapni. A modell továbbá elveszti azokat a tulajdonságokat, amit emberi egérmozdulatok során tapasztalhatunk, miszerint a mozdulatsor általában egy gyorsuló szakasszal indul, ahol a lépések közötti távolság folyamatosan növekszik, majd egy állandó sebességű szakasz következik és végül a cél felé közeledve egy lassuló mozgás figyelhető meg. A modell a működéséből adódóan először az adható legnagyobb lépésszámokat generálja és a távolság csökkenésével változik a kimeneti távolságok mértéke is. A legfőbb probléma pedig az, hogy a generált kimenet sok-sok iteráció alapján készül és ez negatív hatással van a rendszer teljesítményére. Általánosságban elmondható, hogy egy egérmozdulatsor 0.3 másodperc körül generálódik, amely az élesben való teszteléshez nagyon lassúnak mondható.

Összefoglalásként azt szeretném kiemelni, hogy a lineáris regresszió egy könnyen implementálható, egyszerű megoldást nyújtott a problémára, de sajnos közel sem optimálisat és inkább csak kísérletezés szempontjából emelném ki ezt a felépítést. A generált eredmények az alábbi felsorolt ábrákon lesznek láthatók (Lásd 9-10. ábra).



9. ábra Lineáris regresszión alapuló modell által generált egérmozdulat



10. ábra Lineáris regresszión alapuló modell által generált egérmozdulat

## 5.4 GAN neurális hálózat fejlesztése

A Generative Adversarial Network röviden GAN nevezetű eljárást 2014-ben Ian J. Goodfellow és a kollégái dolgozták ki. Ez egy olyan neurális hálózatot jelent, amely két neurális hálózat versengéséből alkotja meg az ideális kimenetet. Az egyik alhálózat a generátor, amely előállítja az elvárt megoldást, a másik hálózat a diszkriminátor, ami elbírálja a generátorból származó kimenet valódiságát. A diszkriminátor úgy van feltanítva, hogy megtudja különböztetni a helyes és helytelen adatot. Kiemelten fontos, hogy generátornak fogalma sincs a valódi és mesterséges mintázatokról. Az egyetlen dolog, amiből tanulni tud az a diszkriminátortól származó eredmény. A végeredmény akkor számít jónak, ha a generátor már olyan hiteles eredményt tud produkálni, amelyről a diszkriminátor nem tudja eldönteni, hogy valódi, illetve mesterségesen generált [17] [18].

A GAN típusú neurális hálózatokat főleg azokon a területeken használják, ahol a bementi adatból valamilyen szabály jelleget szeretnénk felismerni és reprodukálni. Ilyen problémák lehetnek az alábbiak, mint például kép alkotás, szöveg generálás [17] vagy esetleg emberszerű egérmozdulat reprodukálás is. A GAN típusú hálózatok egyik fő ismérve, hogy unsupervised módon tanulnak és képesek egy adott kimenetre több megoldást adni, szemben a többi mélytanuláson alapuló módszerrel. Kiváló példa erre egy GAN típusú képgenerátor, ami, ha paraméterül egy macska szót adunk, akkor képes akár több különböző macskát ábrázoló képet generálni, ahelyett, hogy csak egy adott képet tudna visszaadni.

Az unsupervised learning: a supervised learning ellentéte, amelyben címkézetlen adathalmazok elemzését végzik. Ezek az algoritmusok emberi beavatkozás nélkül fedezik fel a mintázatokat és adatcsoportosításokat [19].

A GAN neurális hálózattal az a célom, hogy a generálni tudjak emberszerű egérmozdulatokat az alapján, hogy a modell felismeri az egérmozgatásban lévő mintázatokat. Az egérmozdulatokról többnyire megállapítható az a tulajdonság, hogy van benne egy gyorsulási szakasz, majd ezt követi egy viszonylag lineáris mozgás és az utolsó pillanatban amikor a mozdulatsor a végcélhoz közeleg akkor lassulás figyelhető meg. Előfordulhat olyan eset is, amikor a felhasználó a kívánt célon túlmozgatja az egerét és utána vissza.

A neurális hálózat két komponensből tevődik össze, egyik a generátor a másik a diszkriminátor. A diszkriminátor az a neurális hálózat, amelyet a tanító adathalmazzal feltanítunk és ezáltal képes eldönteni a következő bejövő adatról, hogy valós vagy mesterséges. Szemben a másik alhálózattal, amit generátornak hívnak, ami azért felel, hogy a diszkriminátortól kapott válaszok alapján úgy változtassa a modell súlyait, hogy az általa generált kimenet a legkevesebb veszteséget eredményezze vagy olyan eredményeket adjon, amelyről a diszkriminátor helytelenül állapítja meg a kimenet valódiságát, más szóval becsapva azt. A neurális hálózat implementálását PyTorchban fogom megoldani.

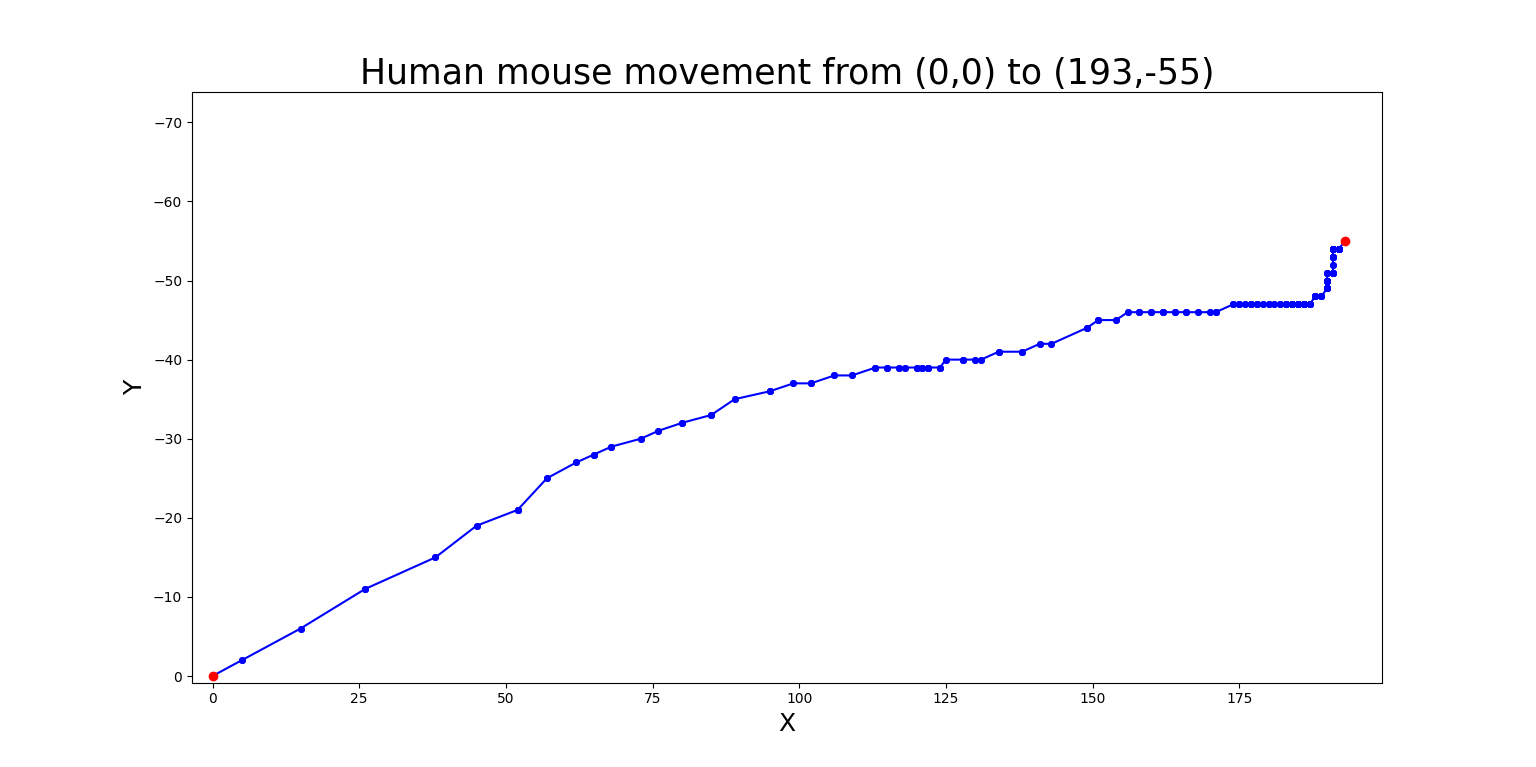
A GAN neurális hálózatok egyik nagy hátránya, hogy rendkívül instabilak. Ez azért van mert a generátor paraméterül egy normál eloszlású zajt kap, jelen esetben egy olyan számsort, ami 0 és 1 érték között tartalmaz olyan számokat, amelynek eloszlása megegyezik a normál eloszlással. A másik indok pedig az, hogy a kimenet folyamatosan az alapján változik, hogy a két versengő neurális hálózat éppen mire tanul rá. Ezekből a versengő viselkedésekből az egyik alhálózat könnyebben rátanulhat bizonyos dolgokra, amelyek akár képesek teljesen rossz irányba vezetni a kimenetek pontosságát.

További problémát jelent, hogy a GAN típusú neurális hálózatok teljesítményének és hatékonyságának kiértékelése a hagyományos metrikákkal, mint például veszteségfüggvényekkel, ugyan lehetséges, de teljesen értelmetlen. A problémát az okozza, hogy a neurális hálózatnak az a feladata, hogy létrehozzon egy generatív modellt, amely képes olyan adatok generálására, amelyek hasonlóak a tanító adathalmazhoz. Ehhez azonban nincs egyértelmű referencia vagy célmutató, mint például a hagyományos osztályozási feladatokban, ahol a pontosság könnyen mérhető. Röviden megfogalmazva a problémát. A GAN-oknál nehéz megállapítani, hogy mi a "jó" generált tartalom.

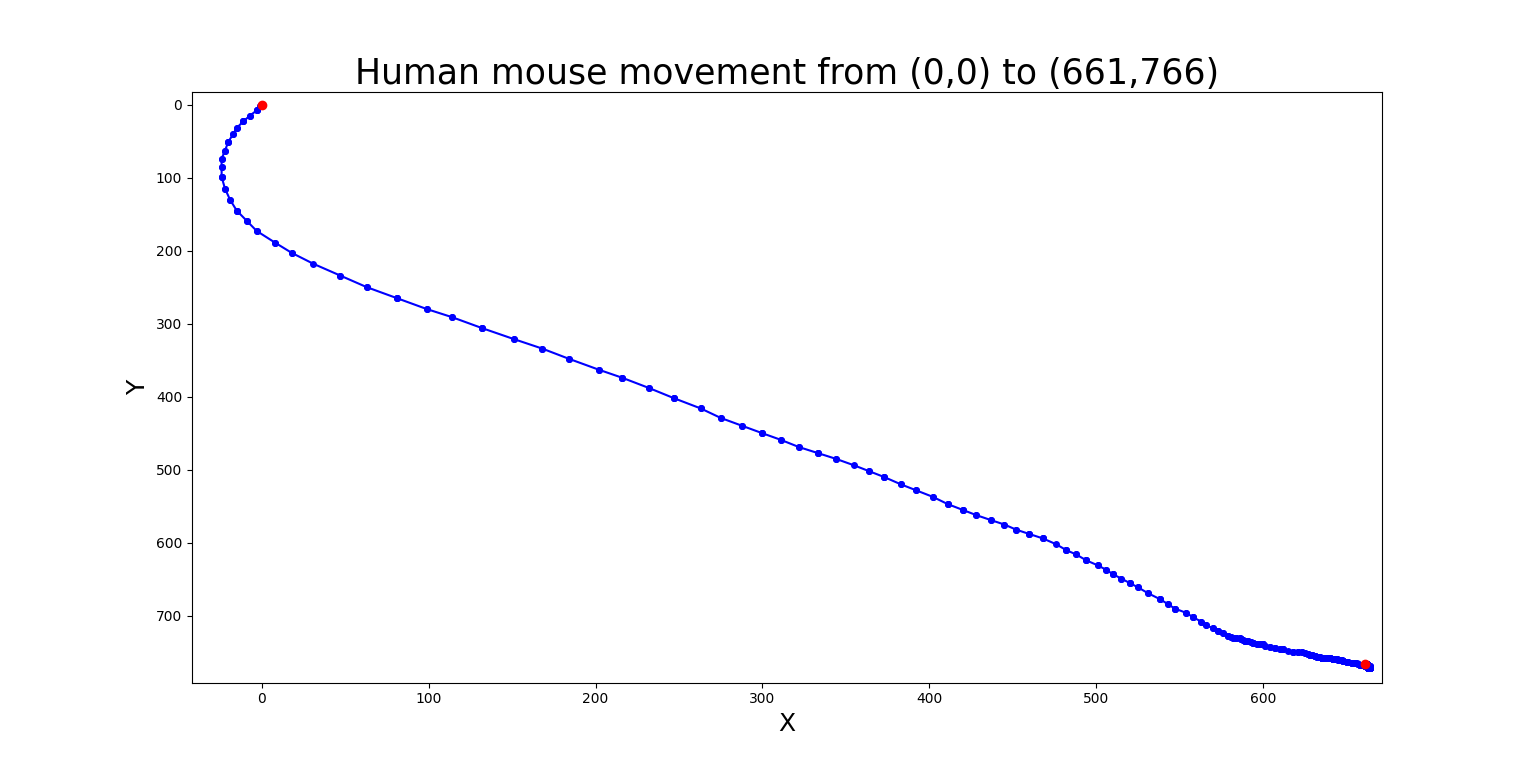
Az egyik lehetséges módszer a neurális hálózat teljesítményének a vizsgálatára az, hogy mi magunk ellenőrizzük le a modell által generált mintát. De ebben az esetben egy másik problémával szembesülhetünk, az pedig a szubjektivitás. Ugyanis előfordulhat, hogy számomra, mint alkotónak tetszik a generált eredmény, de lehet, hogy egy másik szakértő szkeptikus lesz ezzel kapcsolatban.

A neurális hálózat implementálásánál számos akadályba ütköztem. A problémák között szerepel a rossz minőségű tanító adathalmaz, modell megfelelő kiértékelése, stabilitási és túl tanulási nehézségek.

A legelső probléma, amellyel szembesültem, az adatok minősége és mennyisége volt. A GAN hatékony működéséhez nagyszámú, változatos és lehetőleg címkézett adatra van szükség, amely megkönnyíti a hálózatnak a tanulást. Természetesen kezdetben a címkézés számomra nem állt rendelkezésre. Először is azzal az ötlettel álltam elő, hogy az egérmozdulatsor kezdő és végpontjainak különbsége lesz a címkém. A címkékkel ellátott adathalmazon gyorsan beláttam, hogy közel ugyan annyi címkém keletkezett, mint ahány tanító mintám, szóval ez az ötlet nem sikerült túl. A második megoldás erre az volt, hogy az egérmozdulatokat három csoportra osztom fel az alapján, hogy mekkora távolságot írnak le mozdulatsorok. A felosztás a következőképp történt. A 200 pixelnél kevesebb, 200 és 600 közötti pixel távolságot leíró mozdulatok és legvégül a 600 pixelt meghaladó minták kaptak egy-egy külön álló címkét. Így már a tanító adathalmaz eloszlása egyenletes lett. Az adatok előkészítése, formázása és tisztítása időigényes folyamat volt, hát még a gyűjtés. Az alacsony minőségű vagy rosszul címkézett adatok rendkívüli módon tudják torzítani a hálózat tanulását.

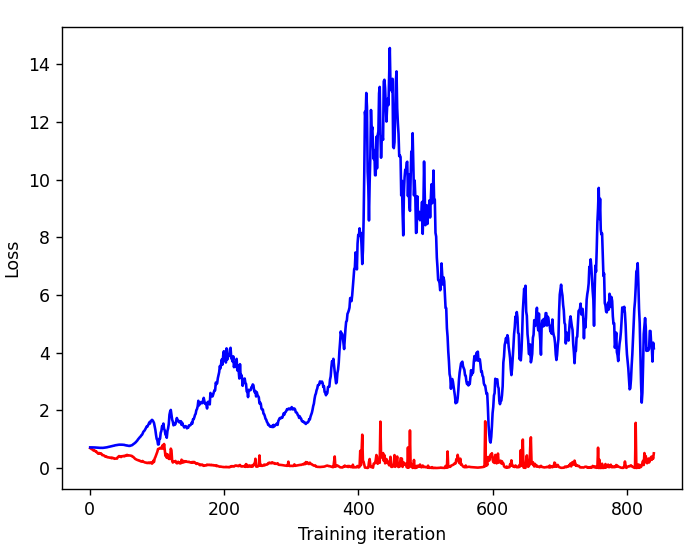
A tanító adathalmaz diverzitásának a növelésére az alábbi lépéseket hajtottam végre. Az egérmozdulatok generálásába több felhasználót bevontam, az adatgyűjtő programot több számítógépen használtam és különböző egér típussal is próbálkoztam. Számomra megdöbbentő volt az egérmozdulatok különbsége, hogy ha azt egy kifejezetten játékosoknak szánt eszközzel vagy egy olcsó irodai egérrel végezzük a mozgást, amelyet az alábbi ábrákon demonstrálok (Lásd 11-12. ábra).

11. ábra irodai egér mintavételezése



12. ábra játékosoknak szánt egér mintavételezése

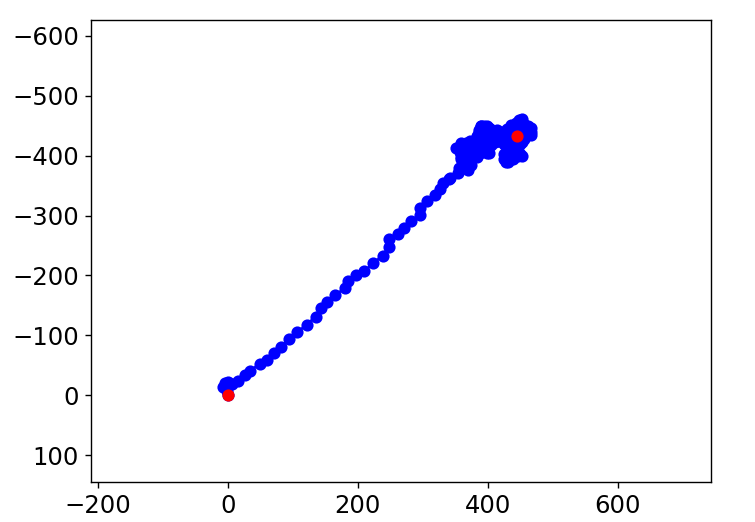
Újabb kihívást jelent a modell megfelelő struktúrájának a kialakítása mivel, ha az egyik alhálózatot túl komplexnek más néven sok réteggel ruházzuk fel akkor az ledominálja a másik neurális hálózatot. Ezáltal instabil működést fogunk elérni és megnehezítjük a modell teljesítmény kiértékelését is. Ideális esetben a két közbelső neurális háló felépítése harmóniában van, képesek a másik kimenete alapján finom hangolni a kívánt eredményt, egymástól tanulva fej-fej mellett versengve haladni a kívánt cél elérése érdekében. Az alábbi képen egy rosszul megválasztott architektúra eredménye látható, illetve a tanítás utáni eredmény (Lásd 13-14. ábra).



Generator

Discriminator

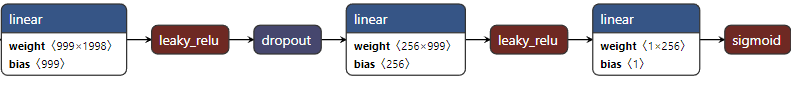
13. ábra GAN neurális hálózat veszteségfüggvényeinek alakulása



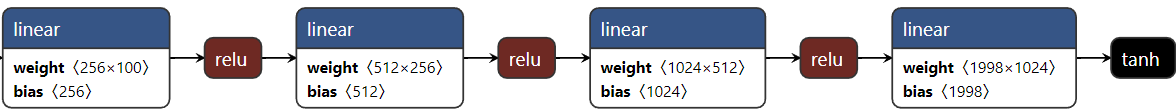
14. ábra kezdetleges GAN neurális hálózat által generált egérmozdulat

A GAN neurális hálózatok fejlesztésénél fontos kiemelni a hiperparaméterek jelentőségét, mivel a hálózat optimális működéséhez elengedhetetlen a megfelelő paraméterek megválasztása. A hiperparaméterek szabályozzák a neurális hálózat struktúráját, képzési dinamikáját és a konvergenciáját. A főbb paraméterek a következőek: tanulási ráta, belső struktúra felépítése, batchek mérete, aktivációs függvények, regularizációs technikák, zaj dimenziói és végül veszteség függvények.

A tanulási ráta angol nevén learning rate meghatározza, hogy a modell a becsült hiba függvényében mekkora változtatást hajtson végre a súlyokon következő tanulási ciklusban. A helyes tanulási ráta kiválasztása nem egyszerű feladat, ugyanis, ha az érték túl nagy akkor a modell konvergenciája nagyon könnyen és gyorsan konvergálhat egy szuboptimális eredményhez, illetve, ha túlságosan alacsony értéket választunk akkor pedig a tanulás sebességének rovására mehet, ami többlet idő és számítási kapacitásba kerülhet, sőt előfordulhat, hogy elakad a tanulási folyamatban a modell [20]. A modellek tanításánál két különböző learning rate-tel dolgoztam, mivel a diszkriminátor jóval hamarabb megtanulta a szintetikus és valódi adat megkülönböztetését, mint ahogy a generátor eltudta volna sajátítani vagy becsapni az általa generált adattal a diszkriminátort. A generátor learning rate-je 0.0001 a diszkriminátor tanulási sebessége pedig 0.00001 lett az általam létrehozott neurális hálózatban.

A modell belső struktúrájának a felépítésénél egyetlen egy tanult módszer sem aratott sikert. A végső állapothoz rengeteg próbálkozás útján jutottam el, fontos kiemelni, hogy a GAN típusú neurális hálózatoknál szinkronban kell tartani a két belső alhálózat struktúráját. Általános szabály, hogy a diszkriminátor a mélyebb hálózat, míg a generátor egyszerűbb, de ez különbözhet a megoldandó feladat körülményeinek függvényében. Az általam megvalósított hálózatok felépítése a következőképpen néz ki (Lásd 15-16. ábra):

15. ábra Diszkriminátor struktúrájának vizuális ábrázolása



16. ábra Generátor struktúrájának vizuális ábrázolása

A batch mérete hatással van a GAN modell tanulásának stabilitására és konvergenciájára. A nagyobb batch méret választása, pontosabb gradiens becsléseket tesz lehetővé, viszont drasztikusan növeli a memória igényt, ha kisebb batch mérettel dolgoztam akkor a sokkal nagyobb jelentősége volt a zajnak, ami gyorsabb konvergenciát jelentett, de sokkal rosszabb minta minőséget a végeredménynél. Az általam legjobbnak választott batch méret 512 lett.

Az aktivációs függvények a megelőző szint neuronjaiból érkező bemenetekhez rendel hozzá egy további értéket [21]. A hozzárendelt érték különbözik az aktivációs függvény típusától. A megfelelő aktivációs függvény kiválasztása döntő szerepet játszik a modell megfelelő viselkedéséhez és teljesítményéhez. Ezek a függvények az úgynevezett nemlinearitást visznek a hálózatba, lehetővé téve számára az összetett leképezéseket a bemenet és a kimenet között. A megfelelő aktivációs függvény kiválasztásához figyelembe kell venni a modell struktúráját és a tanító adathalmaz karakterisztikáit is. Kísérletezéseim alkalmával számomra a generátorban ReLU a diszkriminátorban Leaky ReLU volt a legalkalmasabb. A ReLU más néven Rectified Linear Unit az egyik legelterjedtebb függvény, amit neurális hálózatok létrehozásánál használnak. Működését tekintve is rendkívül egyszerűnek mondható, mivel annyi a szerepe, hogy a negatív értékeket 0-val helyettesítse, illetve, hogy a pozitív értékeket változatlanul hagyja. Matematikailag így írható fel: f(x) = max(0, x). A másik nagy erénye, hogy képes megszüntetni az esetlegesen felmerülő vanishing gradient problémát, ami hatalmas segítség a GAN neurális hálózat feltanításánál. A vanishing gradient probléma főleg a sokrétegű hálózatok tanításánál szokott gondot okozni, ahol jelen van a backpropogation. A probléma abból ered, hogy a veszteség függvények gradiensei a hálózat súlyai tekintetében rendkívül kicsivé válnak a tanulás során, ahogy a kimeneti rétegből a korábbi rétegekbe visszaterjednek. Az utóbbi folyamatot backpropogation-nek nevezi a szakirodalom. Ebben az esetben a modell tanulási képessége lassulhat vagy teljesen elveszítheti [22]. A másik felhasznált aktivációs függvény a Leaky ReLU. Ez a függvény annyival tud többet, mint az előző, hogy a negatív értékeket 0 helyett, egy kis meredekségű gradienst enged meg, amit az α-val szabadon paraméterezhetünk. Matematikailag így írható fel: f(x) = max(αx, x). Jelentősége, hogy képes megakadályozni a neuronok inaktívvá válását.

A modell túltanulásának az elkerüléséhez és a stabilitás megőrzéséhez az alábbi regularizációs technikát alkalmaztam. A diszkriminátor első rétege után rögtön egy dropout réteget használtam, aminek paraméterül 0.3-as értéket adtam meg. Ha túl sok a dropout réteget használok akkor a kimenet eredménye rendkívül zajos marad, ami abban nyilvánul meg, hogy emberi egérmozdulatra egyáltalán nem hasonlító eredményt kapok, továbbá fokozza a mode collapse kialakulását. Droupout réteg nélkül viszont a hálózat csak egy-két mozdulatra tanul rá, szóval megszűnik a kimenetek széles variációja. A dropout réteg lényege, hogy a paraméterül átadott szám jelen esetben legyen ’x’. 1/(1-x) részét a bejövő és kimenő kapcsolatoknak ideiglenesen eltávolítja a hálózatból. A következő ábrák a dropout hatását mutatják meg modell által generált 10 darab egérmozdulatsoron (Lásd 17-18. ábra).

A képen sor, diagram, Diagram, képernyőkép látható

Automatikusan generált leírás

A képen diagram, sor, szöveg, Diagram látható

Automatikusan generált leírás17. ábra túlzott dropout használat hatása a kimenetre

18. ábra dropout nélkül generált kimenet

Bemeneti zaj dimenziója meghatározza a generátorba bejuttatott véletlenszerűség nagyságát, ami lehetővé teszi a változatos minták generálását. A megfelelő zaj struktúrájának kiválasztásánál szintén törekedni kell az arany középút megtalálására. Ha túl komplex vagy túl nagy dimenzió számú a zaj akkor a modell nem lesz képes megfelelően tanulni, ha viszont túlságosan kis szintű akkor a modell befog ragadni bizonyos kimenetekre.

A veszteség függvényeknek a modell konvergenciájának biztosításában van nagy szerepe. Az általam használt modellben a diszkriminátor és a generátor is egy Binary cross-entropy-t használ, ami lényegében egy igen-nem választ ad vissza a generált mintázat valódiságát illetően.

## 5.5 GAN neurális hálózat legnagyobb problémája: Mode collapse

Ahogy az alcím is említi a GAN típusú hálózatok annak ellenére, hogy számos applikációban sikeresen szerepelnek, még is köztudottan szenved egy óriási nagy problémától, ami a további fejlődésében szab gátat és amit úgy hívnak, hogy mode collapse. Ennek a jelenségnek kiemelkedően nagy súlyú hatása van a generátor által kreált kimenet diverzitására, aminek hatását a legjobb esetben is csak enyhíteni tudjuk, de megszüntetni teljesen soha [23]. A probléma abban gyökerezik, hogy a generátor ugyan azt az adatot generálja újra és újra, függetlenül a bemeneti zajtól. Ez a jelenség nagy mértékben korlátozza a GAN-ok használhatóságát, mivel a változatosság hiánya miatt a generált adatok nem tükrözik hűen az eredeti adathalmaz sokféleségét.

A saját szavaimmal megfogalmazva a probléma, úgy képzelhető el, hogy tekintsünk egy festőművészt, aki a pályafutása elején nyilván nem arat túl nagy sikert a közönségnél, illetve a piacon a műveivel, de a folyamatos tanulás és a visszajelzések alapján egyre jobb és jobb képeket készít. Megtanulja, hogy mely stílusjegyek azok, amik megfelelnek az elvárásoknak és ami megfelelő szakmai sikert hoz. A jelen állapot után a képzeletbeli művészünk félhet az újdonságok bevezetésétől és inkább a jól bevált művekhez hasonló darabokat fog készíteni, ami bizonyos idő vagy mennyiség után már nem fogja kielégíteni a vásárló közönséget. A való életben ilyenkor változás szokott bekövetkezni, de visszakanyarodva a GAN típusú hálózatokhoz ott az újítás egy bizonyos szintű siker elérése után biztosan befog ragadni az újításra való hajlam.

Innentől kezdve a modell sikeressége a legjobban abban mérhető, hogy hány darab megfelelő, egymástól különböző kimenet generálására képes a neurális hálózat. Tipikus mode collapse probléma az, amikor egy képgenerátort megkérünk, hogy készítsen képet egy autóról és az autó a generált képen több próbálkozás után is ugyanaz. Esetemben a mode collapse arra utal, hogy az általam készített GAN neurális hálózat tökéletesen megtanult egy fajta egérmozdulatot generálni és teljesen elvesztette, illetve nem tanulta meg a tanító adathalmaz sokszínűségét. Az eredmény a következő ábrán látható (Lásd 19. ábra).

A képen képernyőkép, Diagram, sor, szöveg látható

Automatikusan generált leírás

19. ábra Mode Collapse illusztrálása

A mode collapse előfordulását számos jelenség okozhatja. Az egyik fő probléma az elégtelen modellkapacitás, ami azt jelenti, hogy a generatív modell nem rendelkezik elegendő bonyolultsággal vagy réteg számmal ahhoz, hogy megfelelően leképezze az adatok sokféleségét. Ennek oka lehet a modellkomplexitás hiánya vagy a dimenziómegfelelés hiánya, amikor a modell architektúrája nem illeszkedik az adatok dimenziójához. További tényezők közé tartozik a tanító adathalmaz sokszínűségének és mennyiségének hiánya. Ezen felül a túlreguláció és a képzési instabilitás, amelyek fokozzák az összeomlást a GAN-ok képzése során. A mode collapse kezelése érdekében különböző stratégiák léteznek. A legelterjedtebb a Wasserstein veszteség bevezetése, ami A GAN veszteségfunkcióit úgy alakítja át, hogy közvetlenül képviselje a két valószínűségi eloszlás közötti távolság minimalizálását [24].

Számomra a legnagyobb áttörést a tanítóadat mennyiség megsokszorozása és a batch normalizáció hozta. A kezdeti kettő és háromezer közötti egérmozdulatsor nem nyújtott elegendő diverzitást, bárhogyan is voltak variálva a hiperparaméterek mindig mode collapse lett vége. Később a tanító adathalmaz nagyságát szorgalmas munkával sikerült egészen pontosan 11030-ra emelni. Ebben óriási segítséget nyújtottak a közeli hozzátartozóim, akik serény munkájukkal az adatgyűjtő programomat használták és gyűjtötték az adatot. A másik módszer, ami segített az a batch normalizáció.

A batch normalizáció röviden batch norm egy olyan módszer, amelyet az mesterséges neurális hálózatok gyorsabb és stabilabb képzésére használnak a rétegek bemeneteinek normalizálásával, újraközpontosítással és újraskálázással. Fontos kiemelni, hogy nem a nyers adatokon, hanem a modell rétegei között végzünk normalizálást [25]. A batch norm egy belső rétegnek felel meg amit a rejtett rétegek közé illeszthetünk. Feladata, hogy az első rejtett réteg kimeneteit átvegye és normalizálja, mielőtt továbbadja a következő rejtett réteg bemeneteként.

Lépései a következőképpen írhatóak le, először kiszámítjuk a batch vagy magyarul köteg dimenziónak középértékét ennek képlete a következő [25]:

ahol *m* a köteg mérete és *xi*a bemenet i-edik értéke. Második lépés a kötegek dimenzióinak variancia számítása, aminek képlete [25]:

Harmadik lépésként az előző képletek eredményeinek segítségével normalizáljuk a bemeneteket. Ez a következő módon írható le [25]:

ahol egy tetszőleges kis szám, amelyet stabilitási okokból rendelünk hozzá a képlethez, hogy a nullával való osztást elkerüljük. Az utolsó lépésben pedig skálázást és eltolást kell végezni. Egészen pontosan a normalizált bemeneteket megszorozzuk egy skalárral és eltoljuk egy értékkel. Ezek a változók folyamatosan változnak az eredeti modell paramétereivel a tanítás folyamán és az a szerepük, hogy a visszaállítsák a hálózat reprezentációs tulajdonságait. Ennek a képlete így írható fel [25]:

A felsorolt lépések biztosítják, hogy a neurális hálózat rétegeinek bemenetei hasonló eloszlásúak legyenek, ami segít a hálózat gyorsabb és stabilabb tanulásában. A *γ* és *β* paraméterek lehetővé teszik a hálózat számára, hogy tanulja meg a legjobb skálázást és eltolást a tanulási folyamat során.

A képen diagram, sor, Diagram látható

Automatikusan generált leírásÖsszességében a mode collapse kezelése komplex feladat, amely számos stratégiát igényel, és az alkalmazás és rendelkezésre álló erőforrások függvényében érdemes kiválasztani a megfelelő módszert. A fejezetben felsorolt technikákat alkalmazva a modell eredménye tíz generált egérmozdulatsorra a következő kimeneteket produkálta, ahogy az alábbi képen látható (Lásd 20. ábra).

20. ábra Mode Collapse sikeres visszaszorítása

Az ábrán jól látható, hogy a mode collapse problémát sikeresen orvosoltam, a generált adatok diverzitása megfelelő, ugyan is képes különböző eredményeket produkálni. A második sikeres jel pedig az, hogy a GAN által generált minták illeszkednek a tanító adathalmaz elemeire.

A fentebb említett technikák sikeres alkalmazása után a GAN neurális hálózat az alábbi egérmozdulatsorok generálására volt képes. Megdöbbentő számomra, hogy bizonyos esetekben elsajátította a túlhúzást és ritkán még képes volt akár hurkokat is generálni, amelyből rendkívül csekély számú előfordulás volt a tanító adathalmazban. Az eredmények a következő oldalon láthatóak (Lásd 21. ábra).

A képen szöveg, diagram, képernyőkép, sor látható

Automatikusan generált leírásA képen szöveg, diagram, sor, képernyőkép látható

Automatikusan generált leírásA képen szöveg, sor, diagram, képernyőkép látható

Automatikusan generált leírásA képen szöveg, sor, diagram, Diagram látható

Automatikusan generált leírásA képen szöveg, képernyőkép, sor, diagram látható

Automatikusan generált leírásA képen szöveg, sor, diagram, képernyőkép látható

Automatikusan generált leírás

21. ábra GAN által generált egérmozdulatsorok

## 5.6 Program működése

Ebben a fejezetben azt fogom taglalni, hogy a szakdolgozathoz elkészített rendszereket, hogyan fogom együttesen működtetni. Először is a program kiindulópontja a grafikus felhasználói felület, amelyben a felhasználó beállíthatja, hogy melyik videójátékra szeretné futtatni a programot. A videójáték kiválasztása fontos szempont ugyanis ez a beállítás határozza meg, hogy melyik képfelismerő modellt töltse be a program az elindulásakor. A felhasználónak lehetősége van kiválasztani az egérmozgató algoritmust is, mivel a legjobb teljesítményt a WindMouse algoritmussal és a GAN neurális hálózattal értem el ezért ezt a két lehetséges opciót szabtam meg.

A program elindítása a Start gombbal történik, ilyenkor egy új process vagy más néven folyamat jön létre azért, hogy több fő szállal rendelkezzen a futó program. Az újonnan létrehozott szálban betöltődik először a képfelismerő modell, majd ezt követi a GAN modell betöltése, ha esetleg ezt az egérmozgatási algoritmust választottuk. A modellek betöltése után elindul a képfelvevő szoftver, ami folyamatosan képernyőfelvételeket készít az elsődleges monitorunkról. Ezeket az elkészített képeket, rögtön át is adjuk a képfelismerő algoritmusnak. A képfelismerő algoritmus elvégzi a számításokat és kimenetként egy listát ad vissza, amelyben a küszöbértéket meghaladó objektumok köré írt téglalap csúcspontjait tartalmazza. Ha a lista üres akkor a ciklus elölről folytatódik, viszont, ha a lista nem üres akkor először a téglalapok koordinátából kiszámoljuk annak a közbelső területnek a pozícióját, ahová célozni akarunk. Ez a számolás roppant egyszerű, mindössze megfelezzük a téglalap szélességét, ekkor már az X koordinátát meg is kapjuk, utána az Y-ra van szükség, amit úgy kapunk, hogy a téglalap magasságát elosztjuk a kívánt aránnyal, ha például elfelezzük akkor a célpont alhasi tájékának a pozícióját kapjuk meg, minél nagyobb számmal osztunk annál közelebb kerülünk a téglalap felső részéhez. Ezt a számítást a lista összes elemére elvégezzük. A kiszámolt X és Y párokhoz hozzárendeljük az Euklideszi távolságukat a képernyő középpontjához mérten. A kiszámolt távolság alapján rendezzük a listában lévő elemeket. A legkisebb távolsággal rendelkező fog a lista elejére kerülni. Ezután a legközelebbi elem X és Y koordinátájából kivonom a képernyő közepét leíró X és Y pozíciókat. Az így megkapott ΔX, ΔY pontokat átadom az egérmozdulatsort generáló algoritmusnak, ami rövidesen elő is állítja a kívánt egérmozdulatot.

GAN esetén a generált egérmozdulatsor egy teljesen véletlenszerű irányú és hosszúságú útvonal, amit transzformálni kell ahhoz, hogy pontosan oda mutasson az egérmozdulat, ahová kell. A transzformáció két lépésből áll, először is nyújtani, illetve zsugorítani kell az eredeti útvonalon, majd azt a megfelelő méretre alakítás után, el kell forgatni a szükséges szöggel, hogy az irány ideális legyen. Ezután a generált soron, végig iterálunk, ahol minden egyes elem egy kis elmozdulást tartalmaz, amelyet továbbítunk a win32api egérmozgató függvényének, mivel az iteráció gyorsabban végbe megy, mint kellene, magyarán szólva túl gyors lesz az egér mozgás, ezért a ciklusban sleep-et alkalmazunk, hogy az emberi jellemzőket betartsuk, ha a sleep ideje 0.1 és 0.001 között van, akkor kapjuk a legideálisabb eredményt.

A WindMouse által generált egérmozdulatsornál a transzformációval nem kell foglalkoznunk. Az algoritmus úgy működik, hogy a távolság függvényében, mindig generál egy olyan véletlenszerű kis lépést, amelynek iránya nagyjából a célpont felé mutat. Ez a folyamat addig ismétlődik, ameddig az aktuális egér pozíció és a célpont értéke egy adott küszöbértéken belülre nem esik. A programban ezt 3 pixel távolságra állítottam be. 0 pixel esetén a program már akkor is egérmozgatást végez, amikor a célpont lélegzetvételéből adódóan elmozdulás történik. Röviden fogalmazva túl érzékeny lesz.

## 5.7 Tesztelés és további fejlesztések

A szakdolgozat korai állása alapján a képfelismerést a WindMouse algoritmussal kiegészítve tudtam tesztelni a videójátékokban. Kezdetben a tesztelést offline üzemmódban, hajtottam végre, hogy elkerüljem az esetleges szankciókat. Legelőször azzal a problémával szembesültem, hogy a képfelismerés nagyon lassan futott átlagosan 20 fps környékén. Tudni érdemes, hogy a rendszer teljesítményét jelentősen befolyásolja a képfelismerés sebessége, mivel ez a folyamat frissíti a célpont koordinátákat és csak ezután lehet végre hajtani az egérvezérlést, ha lassan frissül a kiszámolt pozíció akkor szaggatott hatást fogunk tapasztalni az egérműveletek generálása között. A tesztelést főként a Counter Strike: Global Offensive videójátékon végeztem, mivel a többi tesztelni kívánt játék, jóval nagyobb gépigénnyel rendelkezik és a rosszabb minőségű, kevesebb képből feltanított képfelismerő modellek sokkal több álpozitív kimenetet generáltak, ami rendkívül zavaró volt.

A képfelismerő algoritmus gyorsítására az elsődleges megoldás az volt, hogy redukáltam a felvett, vizsgálni való képek méretét, kezdetben 1920x1080-as felbontásban végeztem a tesztelést és most már belátom, hogy jóval kisebb méret is, akár 320x320-as képméret is teljesen megfelel a célnak. Ezzel a módszerrel jelentős teljesítmény növekedést értem el és mivel tapasztalt videójátékosként tudom, hogy ha a célkeresztet megfelelő pontokra, illetve sarokpontokra helyezem akkor egyáltalán nem jelent gondot az, hogy a képfelismerő szoftver kisebb területet vizsgál a rendszer összteljesítményét tekintve.

A második nagy előrelépést az jelentette, hogy sikerült szert tennem egy nagyobb teljesítményű videókártyára egészen pontosan egy RTX 3060-ra, amellyel a kétszeresére tudtam növelni a képfelismerés gyorsaságát. Ezekkel a módosításokkal a képfelismerés már képes volt közel 60fps frissítési értéket produkálni.

Az utolsó nagy áttörést annak köszönhettem, hogy az irodalom kutatásom alatt rátaláltam a TensorRT SDK-ra [13] amit az Nvidia fejlesztett. Ez egy olyan könyvtár, ami optimalizálja a gépi tanuláson alapuló modellek kiértékelő teljesítményét. Az én esetemben miután átkonvertáltam a PyTorch modellemet TensorRT modellre, a képfelismerés a kezdeti 60fps helyett most már 110 és 120 közötti fps értékeket produkál. Hátránya az, hogy csak Nvidia kártyákat támogat.

Miután sikerült a GAN neurális hálózattal generáltatnom az egérmozdulatokat a program futtatása közben egy másik problémával is szembesültem. A kihívás abban rejtőzik, hogy a mozgó célpontokat a rendszer nehezen tudja lekövetni, mivel az objektum aktuális pozícióját meghatározva egy egérmozdulat szimulálása 0.2 – 0.3 másodperc alatt megy végbe és mire az egér a megfelelő koordinátákra kerül, addig már elmozdult a célpont és ilyenkor az egér a célpont 0.2 – 0.3 másodperccel korábbi pozíciójára mozog, ha ez az esemény többször egymás után előfordul akkor ez egy szaggatott hatást fog eredményezni, ami a segédprogramunk detektálását nagyban elősegíti. Természetesen ezt a problémát meglehet oldani úgy, hogy az egérmozgatás sebességét megnöveljük és akkor kevés ideje marad a célpontnak az elmozdulásra, de ebben az esetben a programunk elveszíti az emberszerű jellegét, mivel túlságosan gyorsan fog mozogni az egér és nagy valószínűséggel elveszítjük a fiókunkat is.

A megoldást a fentebb említett problémára a Kálmán-filter alkalmazását látom. A Kálmán-szűrő egy olyan algoritmus, amely optimális becslést ad mozgó, változó rendszerek állapotáról sorozatos mérések alapján, figyelembe véve az előző állapotméréseket és akár még zavaró tényezőket (zajok, bizonytalanságok, pontatlanságok) is. Ezen algoritmus segítségével pontosabb következtetések és információk nyerhetők ki a vizsgált tárgy állapotáról vagy jelen esetben a célpont elhelyezkedéséről.

Az elnevezése Rudolf E. Kálmán (1930–2016) nevéhez fűződik, akinek munkássága főleg Amerikában villamosmérnökként volt számottevő. A Kálmán-szűrőnek számos alkalmazási területe van, gyakran használják navigációs rendszerekben, irányítás vezérlésekben, különösen repülőgépek, űrhajók és robotrepülőgépek esetében. Emellett széles körben alkalmazzák jelfeldolgozó rendszerekben és az ökonometria területén [26].

Az algoritmus két lépésben működik. Az első lépésben a Kálmán-szűrő kiszámolja az aktuális állapotváltozókat a hozzájuk tartozó bizonytalanságokkal együtt. A következő lépésben a sorozatos mérések eredményeit súlyozott átlagolással veszi figyelembe. Ahogy haladunk a valós idejű mérések során, az átlagolás eredményeként egyre pontosabb értékek jelennek meg, miközben a zajok és más zavaró tényezők kiszűrődnek. Az algoritmus rekurzív jellegű, csupán az aktuálisan kalkulált állapotot és az aktuális mérési eredményeket veszi figyelembe, korábban mért adatokat nem használ fel. Elméletileg a Kálmán-szűrő alapfeltevése az, hogy a vizsgált rendszer egy lineárisan dinamikus rendszer, és minden hibafüggvénynek és mérésnek is normális eloszlása van [26] [27].

Sajnos az implementálása során komoly akadályokba ütköztem. A probléma abban gyökerezik, hogy a pontos predikcióhoz többszörös mintavételezésre van szükség, ami időigényes és ameddig elő nem áll a pontos becslés, addig a kurzorunk folyton a kívánt célpont mellé fog mozogni. Tovább súlyosbítja a helyzetet, ha a mozgás irányát gyakran változtatja az objektum, ilyen esetben a rendszer rendkívül lassan tud újra pontosan célozni.

Megoldásként azt tudnám elképzelni, ha pythonban megfelelően lehetne párhuzamosítani a programot és nem ütköznénk folyton a globális interpreter lock által okozott gátakba. Kétségbe esésemben és a teljesítménycentrikus gondolkozásomra hivatkozva az lenne a legoptimálisabb lépés, ha az egész program C++-ban lenne átírva python helyett.

5.8 Egérmozgató rendszerek összehasonlítása

Ebben a fejezetben az emberszerű egérmozgatásért felelős rendszerek tulajdonságait fogom röviden bemutatni, amelyeket az általam készített GAN neurális hálózattal hasonlítottam össze. A tesztelést több szempontból is vizsgáltam. Kezdetben a tanító adathalmaz felépítését, ezt követően a futási időt, majd éles játékban tettem próbára a módszereket. Az összehasonlítás szempontjait egy táblázatban foglaltam össze (Lásd 22. ábra) a könnyebb átláthatóság miatt.

|  |  |  |  |
| --- | --- | --- | --- |
| Eljárás neve: | SapiAgent autoenkóder | Saját GAN | WindMouse |
| Tanuló adathalmaz: | 128 ember, 3 és 5 perces mérések | Általam készített  11030 minta | Nem rendelkezik |
| Monitorozás frekvenciája: | 60Hz | 1000Hz | Nem rendelkezik |
| Szükséges idő egy generált adathalmaz létrehozásához: | Kevesebb mint 1ms | 1ms | 3.6ms |
| Átlagos lépésszám egy mozdulatsorban: | 128 | Változó | Dinamikusan változik a távolság függvényében |
| Újra generáláskor változik a kimenet | Nem | Igen | Igen |
| Használatáért a felhasználót kitiltás érte: | Nem | Nem | Nem |

22. ábra Az összefoglaló táblázat az egérmozdulat generáló módszerekről

Az elsődleges különbség az, hogy az egérmozgató algoritmusok közül a WindMouse algoritmus semmiféle gépi tanulási módszert nem alkalmaz, viszont ennek ellenére megközelítőleg ugyan olyan jó kimeneteket tud nyújtani, mint a másik két megoldás. Kétségtelen, hogy ez az algoritmus implementálható a legegyszerűbben és működését tekintve sem nevezhető bonyolultnak.

Visszakanyarodva a machine learningen alapuló módszerekre jól látható, hogy mind a két modell viszonylag sok tanítóadattal rendelkezik. A tanítóadatok sokszínűségében kiemelkedik a SapiAgent, mivel több ember bevonásával készült és a puszta egérmozdulatokon kívül még a drag and drop és különböző egérgombok lenyomásával létrehozott egérmozdulatsorokat is tartalmaz. Komoly hátránya viszont az alacsony monitorozási frekvencia, amivel pontatlanabb, illetve kevésbe részletes adatokra tehetünk szert. Futásidő tekintetében nincs nagy különbség a leggyorsabb egérmozdulatsor generálást a SapiAgenttel mértem, viszont az éles videójáték tesztben nem volt érezhető különbség a módszerek között. A SapiAgent egyik legnagyobb hátránya, hogy a kimenetei rendkívül statikusak, ami szerintem az emberszerű egérmozdulatra egyáltalán nem jellemző, tovább ront a helyzeten, hogy mindig egy fix lépésszámú eredményt ad.

# 6. Konklúzió

A szakdolgozat végleges állapotát tekintve sikerült létrehozni, egy olyan mélytanuláson alapuló GAN modellt, ami képes kimagaslóan jó minőségű emberszerű egér mozdulatsorok generálására. Az egérmozgató algoritmus teszteléséhez nem csupán diagramokat és egyéb mérőszámokat hasonlítottam össze, hanem valós környezetben is próbára tettem a Counter Strike: Global Offensive, Counter Strike 2 és a Valorant nevű videójátékban. A célom az volt, hogy az egérmozgató algoritmust képfelismeréssel kiegészítve, egy olyan programot tudjak létrehozni, ami akár képes túlszárnyalni az emberi precizitást és reakciót, mindezt úgy, hogy egy valódi személy viselkedését tükrözze. Az elkészült projekt képes a másodpercek tört része alatt akár teljesen autonóm módon célzott lövést leadni, ha ellenfelet talál maga előtt.

A programnak bizonyos szempontból vannak hiányosságai, először is ahhoz, hogy eredményesen tudjon működni szükség van kiemelkedően nagy teljesítményű hardverre. További másik probléma, hogy a rendszer gyorsan mozgó célpontok esetén már jóval többször téveszt célt és ezt semmilyen módon nem sikerült kijavítani.

A program használata mellett a játékok átlagosan 10-15%-os teljesítmény csökkenéssel futnak ez főleg a képfelismerő algoritmus következménye. Felhasználói szinten az alábbi dolgokat tapasztaltam 23 CS: GO mérkőzés adataiból kiindulva. Javult a pontosságom, a fejlövési arány 22%-ról 50%-ra emelkedett, viszont ölési/halálozási arány érdemben nem javult. A fentebb leírt eredményeket egy online profil alapú statisztikai oldal adataiból használtam fel. Továbbá észrevettem magamon, hogy a program használatával jobban izgulok a játék közben és a kognitív készségek, mint például a taktikai döntések, helyezkedések sokkal inkább a háttérbe szorulnak.

Összességében elmondható, hogy a program bizonyos aspektusok alapján növelni tudja a játékos teljesítményét, de egy amatőr játékosból nem fog világklasszist készíteni. A legkiemelkedőbb eredmény az, hogy az elkészített programmal egyik játékfiókomat se érte kitiltás, teljesen észrevétlenül tudtam játszani, úgy, hogy egyszerre voltam figyelve a videójátékokban használt védelmi mechanizmusokkal és a körülöttem lévő játékosok által. Az elkészült munka demonstrálása az alábbi linket tekinthető meg.

<https://youtube.com/playlist?list=PLRSRqGkivma474Tlw7tV-XZ4InfvlAJCt&si=gKr_XplNpCsNvucr>

A projekt biztosan tovább finomítható bizonyos aspektusaiban és kiegészíthető olyan, gépi tanuláson alapuló megoldásokkal, amelyek akár a karakter mozgását is tudnák hatékonyan kezelni és ezáltal létrehozni egy teljesen autonóm programot, ami akár képes lenne megszorongatni a profi játékosokat.

# 7. Köszönetnyilvánítás

Szeretném kifejezni mély hálámat mindazoknak, akik segítettek és támogattak engem a szakdolgozatom elkészítése során. Először is szeretnék köszönetet mondani Dr. Vámossy Zoltán témavezetőmnek, akinek köszönhetően a témám szabad lendületet vehetett és aki nélkül ez a munka nem valósulhatott volna meg. Az iránymutatásai, tanácsai és támogatása nélkül a dolgozat elkészítése kivitelezhetetlen lett volna. Köszönettel tartozom, Prof. Dr. Szénási Sándornak, aki a GAN neurális hálózat optimális kiértékelésében nyújtott segítségéért.

Köszönettel tartozom továbbá az intézménynek, ahol nem csupán tudásra tettem szert, hanem értékes kapcsolatokat és élményeket is gyűjthettem. Az itteni oktatók és gyakorlat vezetők hozzájárultak a megfelelő elméleti és a gyakorlati tudás elmélyítéséhez, amit mind igyekeztem visszatáplálni a szakdolgozatomba.

# 8. Hivatkozások

|  |  |
| --- | --- |
| [1] | Ersin, A. ., Tezeren, H. C. ., Ozunlu Pekyavas, N., Asal, B. ., Atabey, A. ., Diri, A. ., & Gonen, İsmail "THE RELATIONSHIP BETWEEN REACTION TIME AND GAMING TIME IN E-SPORTS PLAYERS", *Kinesiology*,Vol. 54., 36–42., 2022. |
| [2] | C. Feher, Y. Elovici, R. Moskovitch, L. Rokach és A. Schclar, „User identity verification via mouse dynamics,” *Information Sciences, Vol. 201.,*pp. 19-36., 2012. |
| [3] | L. A. Leiva, L. Arapakis és C. Iordanou, „My Mouse, My Rules: Privacy Issues of Behavioral User Profiling via Mouse Tracking,” *Proceedings of the 2021 Conference on Human Information Interaction and Retrieval,Vol. 90.,* pp. 51-61, 2021. |
| [4] | N. Banholzer, S. Feuerriegel, E. Fleisch, G. F. Bauer és T. Kowatsch, „Computer Mouse Movements as an Indicator of Work Stress: Longitudinal Observational Field Study,” *JMIR Publications,Vol. 23.,* pp. 4-23., 2021. |
| [5] | M. Antal, K. Buza és N. Fejer, „SapiAgent: A Bot Based on Deep Learning to Generate Human-Like Mouse Trajectories,” in *IEEE Access*, Vol. 9., pp. 124396-124408, 2021. |
| [6] | S. Ford, „mmorpg.com,” 27 01 2011. [Online]. Available: https://www.mmorpg.com/news/planet-calypso-sold-for-6-million-usd-2000073586. [Hozzáférés dátuma: 24 04 2023]. |
| [7] | A. R. Kang, S. H. Jeong, A. Mohaisen és H. K. Kim, „Multimodal game bot detection using user behavioral characteristics,” *ResearchGate,Vol. 5.,* pp. 3-9, 2016. |
| [8] | Y. Chung, C.-y. Park, N.-r. Kim, H. Cho, T. Yoon, H. Lee és J.-H. Lee, „Game Bot Detection Approach Based on Behavior Analysis and Consideration of Various Play Styles,” in *Etri Journal*,Vol. 35.,pp.1058-1067., Korea, 2013. |
| [9] | Varvello, Matteo & Voelker, Geoffrey, „Second Life: a Social Network of Humans and Bots,” in *NOSSDAV*, Amsterdam, 2010. |
| [10] | Su-Yang Yu, N. Hammerla, J. Yan és P. Andras, „Aimbot Detection in Online FPS Games Using a Heuristic Method Based on Distribution Comparison Matrix,” in *Proceedings of the 19th international conference on Neural Information Processing - Volume Part V*, pp. 654-661,Doha-Qatar, 2012. |
| [11] | „PyTorch,” Open Source, 09 2016. [Online]. Available: https://pytorch.org/. [Hozzáférés dátuma: 27 09 2022]. |
| [12] | G. Jocher, „YOLOv5 documentation,” 18 Május 2020. [Online]. Available: https://docs.ultralytics.com/yolov5/. [Hozzáférés dátuma: 8 12 2022]. |
| [13] | Nvidia, „TensorRT,” Nvidia, 2020. [Online]. Available: https://developer.nvidia.com/tensorrt. [Hozzáférés dátuma: 03 04 2023]. |
| [14] | „Python MSS’s documentation,” [Online]. Available: https://python-mss.readthedocs.io/. [Hozzáférés dátuma: 13 12 2022]. |
| [15] | IBM, „IBM - Supervised Learning,” [Online]. Available: https://www.ibm.com/topics/supervised-learning. [Hozzáférés dátuma: 06 05 2023]. |
| [16] | B. J. Land, „WindMouse, an algorithm for generating human-like mouse motion,” 25 4 2021. [Online]. Available: https://ben.land/post/2021/04/25/windmouse-human-mouse-movement/. [Hozzáférés dátuma: 8 12 2022]. |
| [17] | A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta és A. A. Bharath, „Generative Adversarial Networks: An Overview,” *IEEE Signal Processing Magazine, Vol.* 35., p. 53, 2018. |
| [18] | I. J. Goodfellow és J. Pouget-Abadie, „Generative Adversarial Nets,” in *Departement d’informatique et de recherche opérationnelle*, Universite de Montreal, 2014. |
| [19] | IBM, „IBM - What is unsupervised learning,” International Business Machines, [Online]. Available: https://www.ibm.com/topics/unsupervised-learning. [Hozzáférés dátuma: 19 05 2023]. |
| [20] | J. Brownlee, „Machine Learning Mastery,” 12 09 2020. [Online]. Available: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/. [Hozzáférés dátuma: 22 3 2024]. |
| [21] | Szénási Sándor; Kertész Gábor, „Deep Learning V0.1,” in *Backpropogation*, 2018, p. 29. |
| [22] | Chi-Feng Wang, „Towards Data Science,” 08 01 2019. [Online]. Available: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484. [Hozzáférés dátuma: 22 03 2024]. |
| [23] | Youssef Kossale; Mohammed Airaj; Aziz Darouichi, „Mode Collapse in Generative Adversarial Networks: An Overview,” *IEEE Xplore, Vol. 1.*, pp. 1-2, 2022. |
| [24] | M. Neri, „Spot Intelligence,” [Online]. Available: https://spotintelligence.com/2023/10/11/mode-collapse-in-gans-explained-how-to-detect-it-practical-solutions/. [Hozzáférés dátuma: 5 4 2024]. |
| [25] | C. S. Sergey Ioffe, „Batch Normalization: Accelerating Deep Network Training by Reducing,” in *Google*, 1600 Amphitheatre Pkwy, Mountain View, CA 94043, 2015. |
| [26] | „Kálmán-szűrő - Wikiwand,” [Online]. Available: https://www.wikiwand.com/hu/Kálmán-szűrő. [Hozzáférés dátuma: 18 07 2023]. |
| [27] | E. R. Kalman, „A New Approach to Linear Filtering,” in *Research Institute for Advanced Study*, Baltimore, Md, 1960. |

# 9. Ábrajegyzék

[1. ábra Grafikus felhasználói interfész 17](#_Toc164275049)

[2. ábra Rendszerterv diagram 19](#_Toc164275050)

[3. ábra Címkézés menete 20](#_Toc164275051)

[4. ábra Saját egérmozdulat mérési eredménye 22](#_Toc164275052)

[5. ábra Saját egérmozdulat mérési eredménye 23](#_Toc164275053)

[6. ábra Saját egérmozdulat mérési eredménye 23](#_Toc164275054)

[7. ábra Saját egérmozdulat mérési eredménye 23](#_Toc164275055)

[8. ábra WindMouse által generált egérmozdulat 24](#_Toc164275056)

[9. ábra Lineáris regresszión alapuló modell által generált egérmozdulat 26](#_Toc164275057)

[10. ábra Lineáris regresszión alapuló modell által generált egérmozdulat 26](#_Toc164275058)

[11. ábra irodai egér mintavételezése 29](#_Toc164275059)

[12. ábra játékosoknak szánt egér mintavételezése 29](#_Toc164275060)

[13. ábra GAN neurális hálózat veszteségfüggvényeinek alakulása 30](#_Toc164275061)

[14. ábra kezdetleges GAN neurális hálózat által generált egérmozdulat 30](#_Toc164275062)

[15. ábra Diszkriminátor struktúrájának vizuális ábrázolása 31](#_Toc164275063)

[16. ábra Generátor struktúrájának vizuális ábrázolása 31](#_Toc164275064)

[17. ábra túlzott dropout használat hatása a kimenetre 33](#_Toc164275065)

[18. ábra dropout nélkül generált kimenet 33](#_Toc164275066)

[19. ábra Mode Collapse illusztrálása 35](#_Toc164275067)

[20. ábra Mode Collapse sikeres visszaszorítása 37](#_Toc164275068)

[21. ábra GAN által generált egérmozdulatsorok 38](#_Toc164275069)

[22. ábra Az összefoglaló táblázat az egérmozdulat generáló módszerekről 42](#_Toc164275070)