



MATEMATIKAI ÉS INFORMATIKAI INTÉZET

Játékfejlesztés Unity keretrendszerben

Készítette

Tápai Árpád

Program tervező informatika

Témavezető

Troll Ede

tanársegéd

EGER, 2024

Tartalomjegyzék

1. Bevezető	4
2. Irodalom feldolgozás, háttér információk	5
2.1. AI	5
2.2. Deep learning	6
2.3. AI Játékokban.	7
2.3.1. Játékokban az Ai fejlődése	7
3. A feladat megoldásához rendelkezésre álló technikák ismertetése.	10
3.1. Mi is az Deep learning ?	10
3.2. Mi is az a Nerural Network (neurális hálózat)?	12
3.3. Mi is az Reinforcement Learning ? (Megerősítő Tanulás)	12
3.3.1. Q-learning	16
3.4. Deep Reinforcement learning (Mély Megerősítő Tanulás)	17
3.5. Unity	17
3.5.1. Deep learning Unityben	18
3.6. Unreal	19
3.6.1. Deep learning Unrealban	20
3.7. Unity vs Unreal	21
3.8. Pytorch	21
3.9. TensorFlow	22
4. Saját teljesítmény előállításához ténylegesen felhasznált eszközök részletes ismertetése ; telepítés, használatba vétel előfeltételei és lépései.	23
4.1. Unity	23
4.1.1. Telepítés	23
4.2. ML Agents(2.0.1)	28
4.3. Telepítés	29
5. Saját munkánk (alkalmazásfejlesztés, mérés, tervezés stb.) részletes leírása, az eredmények szemléletes ismertetése.	32
5.1. Állapottér:	33

5.2.	Akciótér:	33
5.3.	Jutalmi struktúra:	33
5.4.	További fontolóra veendő szempontok:	33
5.5.	Scriptek	36
5.6.	Eredmények	53
5.7.	Hibakeresés	54
5.8.	Verzió követés	56
5.9.	Tesztelés	57
6.	Összegzés	61

1. fejezet

Bevezető

Az informatika világa mindig is rohamos ütemben fejlődött ami meglátszik a játékfejlesztésben és a mesterséges intelligencia (későbbiekben: MI) területén is. Manapság már nem kell saját motort írnuk ahhoz, hogy játékokat fejlesszünk. Számos keretrendszer akad e téren akár a Unity, Unreal, vagy Godot Engin-ről an szó, mind megfelelő választás lehet. Ebben a dinamikus környezetben kiemelkedő szerephez jutott a mélytanulás, amely a MI egyik leg-dinamikusabban fejlődő területe. A mélytanulás a neurális hálózatok speciális esete, amely inspirációt merít az emberi agy működéséből, és lehetővé teszi a játékokban lévő MI számára a komplex feladatok automatizált végrehajtását. A szakdolgozat célja, hogy részletesen megvizsgálja, hogyan lehet a Unity keretrendszer segítségével fejlesztett Tron játékban (mély tanulás alkalmazásával) két MI kiképezését megvalósítani úgy, hogy reagáljanak a környezetre, versenyezzenek egymással, minél tovább éljenek, és megtalálják beavatkozás nélkül a legjobb módszert a túlélésre. A vizsgálatot Unity 3D engine segítségével végzem el, azon belül a Unity Machine Learning Agents szolgáltatja majd a mélytanulási alapokat. A PyTorch segítségével, a videókártya erőforrásaihoz hozzáférve, sokkal gyorsabban lefutási időket tudunk elérni a mesterséges intelligencia oktatására. A játék vezérlése, Visual Studioban, C# nyelven írott scriptekkel készül el. Kisiskolás éveimtől kezdve vonzódom a számítógépes játékok világához. Mindig arról álmodoztam, hogy egyszer saját játékot fogok létrehozni. Amikor felfedeztem, hogy Troll Ede Tanár Úr lehetőséget kínál a Unity témakörében, azonnal felkerestem. Kifejtettem neki az elképzeléseimet, egy teljesen egyedi játékfejlesztő projektről, majd örömmel egyeztettünk a részletekről. Már alig várom, hogy együtt elinduljunk ezen az izgalmas úton, és valóra váltsuk a játékfejlesztés iránti szenvedélyemet!

Az információs technológia térnyerése, az adathálózatok robbanásszerű növekedése, új kihívások elé állította az adatfeldolgozás és az automatizált intelligens döntéshozatal területét. Ebben a gyorsan fejlődő környezetben vált kiemelkedő fontosságúvá a mélytanulás, amely a MI egyik legizgalmasabb és leghatékonyabb ága. A mélytanulás mögött álló kulcsfontosságú elem a neurális hálózatok rendszere.

2. fejezet

Irodalom feldolgozás, háttér információk

Régen, a számítástechnika hajnalán, a számítógépek lassú ütemben és nagy méretben hódították meg a világot. Az ENIAC, az első általános célú elektronikus számítógép, több szobát is elfoglalt, és a működéséhez hatalmas elektroncsövekre és ellenállásokra volt szükség. A számítógépek mérete és súlya korlátozta használatukat, és csak a legnagyobb vállalatok vagy kormányzati intézmények engedhették meg maguknak az ilyen berendezések üzemeltetését.

Az évek során a technológiai fejlődés azonban lenyűgöző ütemben zajlott. A hatalmas elektroncsövek és ellenállások helyét korszerű és kicsiny méretű tranzisztorok vették át. A számítógépek kompakt méretben, de hatalmas teljesítménnyel rendelkezzenek. A tranzisztorsűrűség és a processzorok sebessége olyan szintekre emelkedett, amelyek régen elképzelhetetlenek voltak.

A számítógépek fejlődése nemcsak a méretüket és sebességüket érintette, hanem az interaktivitást és a felhasználói élményt is. Régen, a felhasználók kevés lehetőséggel rendelkeztek, és a számítógépek által kínált élmények egyszerűbbek voltak. Ma viszont a számítógépek és laptopok sőt, még a mobiltelefonok is olyan rendkívül kifinomult grafikus és számítási képességekkel rendelkeznek, amelyek lehetővé teszik számunkra, hogy akár a leg-komplexebb játékokat is élvezhessük, vagy, hogy kreatív projekteken dolgozhassak.

2.1. AI

A mesterséges intelligencia (MI vagy AI) már kiterjedt módon jelen van a minden napjai életünkben. Az okostelefonok személyi asszisztensei, mint például a Siri vagy a Google Assistant, például AI-alapú rendszerek, melyek széles skálán nyújtanak segítséget a felhasználóknak az információkeresésben, emlékeztetők beállításában és egyéb feladatok elvégzésében. Az online keresőmotorok szintén intelligens technológiákat alkalmaznak annak érdekében, hogy személyre szabott és releváns találatokat biztosítsanak.

Az e-kereskedelmi platformok is kihasználják az AI-t a személyre szabott ajánlatok és termékajánlások tervezéséhez, amelyek segítik a vásárlókat a döntéshozatalban. Az autóipar-

ban az autonóm járművek fejlesztése is hangsúlyosan támaszkodik a mesterséges intelligenciára.

Az egészségügy területén is megfigyelhető az AI alkalmazása a diagnózisok támogatásában, a betegadatok elemzésében és a terápiás lehetőségek kifejlesztésében. Ezen kívül a pénzügyi szektorban is széles körben alkalmazzák az AI-t a csalások felderítésére és a befektetési döntések támogatására.

2.2. Deep learning

A deep learning, azaz mélytanulás, az informatikai tudományágak egyik legdinamikusabban fejlődő területe, amely forradalmasítja a gépi tanulás és mesterséges intelligencia (MI) alkalmazását. Az elmúlt években a deep learning számos területen hozott jelentős előrelépéseket, és számos globális szereplőt inspirált arra, hogy fejlessze és alkalmazza ezt az innovatív technológiát.

Az egyik legjelentősebb fejlődési terület a neurális hálózatok bővülése és mélyülése. Az eredeti, sekélyebb architektúrákból, mint például az egyszerű mesterséges neurális hálózatok, a deep learning áttért a mélyebb struktúrákat alkalmazó hálózatokra, amelyek képesek összetettebb és absztraktabb mintázatokat felismerni és megérteni. Ezek a mélyebb hálózatok, például a konvolúciós neurális hálózatok (CNN) és a rekurrens neurális hálózatok (RNN), a képfelismerés, beszédfelismerés és szekvenciális adatok elemzése terén mutattak kiemelkedő teljesítményt.

A deep learning alkalmazása terjed az ipari szektoroktól a minden nap életig. Az autonóm járművek, az egészségügyi diagnosztika, a pénzügyi elemzés, a nyelvi feldolgozás és a kereskedelmi alkalmazások minden olyan területek, ahol a deep learning rendszerek jelentős hatékonyságnövekedést és pontosabb eredményeket hoznak. A képfelismerési algoritmusok például lehetővé teszik azt, hogy pontosan azonosítsák és kategorizálják a képeken szereplő objektumokat, míg a nyelvi feldolgozó algoritmusok révén a gépek egyre jobban érthetik és reprodukálhatják az emberi nyelvet.

A deep learning terjedését támogatja az egyre nagyobb mennyiségű elérhető adat és a folyamatosan fejlődő hardverinfrastruktúra. A GPU-k és TPU-k, vagyis grafikus és tensor processzorok, kifejezetten alkalmasak a deep learning műveletek hatékony végrehajtására. A felhőalapú számítástechnika és az elosztott rendszerek további elősegítő tényezői a deep learning elterjedésének.

Globális szinten az Egyesült Államok, Kína és az Európai Unió versengenek a deep learning kutatások és fejlesztések élén. A vezető technológiai vállalatok, mint például az Alphabet (Google anyavállalata), Microsoft, Amazon, és a kínai tech óriások, mint a Baidu és a Tencent, komoly erőforrásokat fektetnek be a deep learning projektekre. Emellett egyre több egyetem és kutatóintézet vesz részt a deep learning területén folyó kutatásokban és fejlesztésekben.

2.3. AI Játékokban.

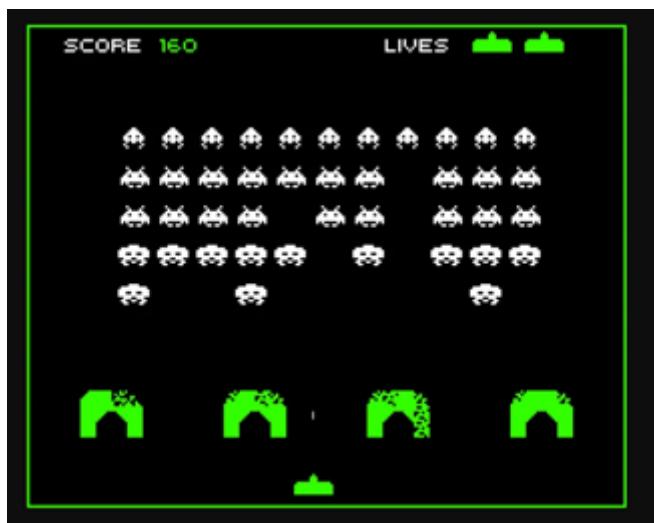
A mesterséges intelligencia (MI) szerepe is kritikusnak bizonyult ezen fejlődésben. A videójáték-iparban valódi áttörést hozott az AI, ami lehetővé teszi a játékoknak, hogy idővel javuljanak képességeikben, intelligenciájukban és társadalmi készségeikben. Ennek eredményeként összetettebb és kihívást jelentő szituációk jönnek létre, főként az emberi tulajdonságok utánzására használva. Ez javítja a játékélményt és a felhasználói élményt.

A nem-játékos karakterek (NPC-k) viselkedésfejlesztéséhez a Game AI-t alkalmazzák, beleértve az NPC útvonaltervezését, különböző terepeken való mozgásukat és a virtuális világban való navigációjukat. Az AI révén ezek az NPC-k ügyesen elkerülhetik az ütközéseket a ellenségekkel vagy megtalálhatják az szövetségeseket.

A mesterséges intelligencia szerepe az egész folyamatban kiemelkedő, hiszen az Mesterséges Intelligencia révén a játékok nem csupán technológiai csodák, hanem interaktív és adaptív élmények is, amelyek folyamatosan formálódnak a játékosok igényei és preferenciái alapján. Az egyre növekvő számú játékos és a piac folyamatos bővülése pedig azt mutatja, hogy a videójáték-ipar még hosszú ideig a fejlődés és innováció főszereplője marad.

2.3.1. Játékokban az Ai fejlődése

Az arcade játékok az egyik korai példája a kereskedelmi játékoknak, amelyek tárolt mintákat használnak az ellenséges mozgás irányítására. A későbbi mikroprocesszorok bevezetése lehetővé tette a véletlenszerű mozgásminták használatát. Ilyen példa az ikonikus Space Invaders (1978). Az arcade játékok korai AI-ja tárolt mintákat használt az idegenek véletlenszerű mozgásának szimulálására, sikeresen megtartva a játékosokat és a gépeikre dobált érméket.



2.1. ábra. Space Invaders (1978)

A Véges Állapot Gépek (FSMs), amelyeket a fejlesztés során alkalmaznak, az AI működési folyamatát mutatják be. Például olyan játékokban, mint a Tekken és a Mortal Kombat

sorozatok. Az FSM-ek az ellenfelek cselekvését minden egyes állapotban vezérlik, például gyógyulás vagy támadás.



2.2. ábra. Mortal Kombat

Mostanában az NPC-k képesek vadászni a játékosokra, a generikus támadás/védekezés mód kibővítésével. Ehhez az AI hangokat és fizikai zavarokat használ, például lábnyomok vagy egy ág törése a lábuk alatt. Ezek a komplexitások jelen vannak lopakodó játékokban, mint a Metal Gear Solid 5 (2015), és egyre élethűbbé teszik az open-world játékokat. Az NPC-k már nemcsak vadászok, hanem túlélők is, hiszen az AI előre beállított jelekkel rendelkezik számukra, hogy javítsák egészségi szintjüket, újra töltsek lőszerüket vagy fedezéket keressenek csaták közben.

Ezek a technikák kulcsfontosságúak a játékélmény javításában. Az online többjátékos játékok uralják a piacot, ahol a játékosok olyan ellenfeleket akarnak, akik impulzív és kiszámíthatatlan viselkedésűek. Az egyjátékos játékoknak folyamatosan fejleszteniük kell az open-world környezeteket és a nem-játékos karaktereket, hogy versenybe szálljanak az emberi játékossal. Az AI az a kulcs ehhez.



2.3. ábra. red dead redemption 2

A rendkívül sikeres GTA5 (2013) az open-world játékokat egy új szintre emelte. A játékban látható karakterek életszerűek és egy valós világ szimulációjában helyezkednek el - Los Santosban. Ez hatalmas előrelépés a kétezres évek elején megjelent madártávlati nézetű játékokhoz képest!



2.4. ábra. GTA V

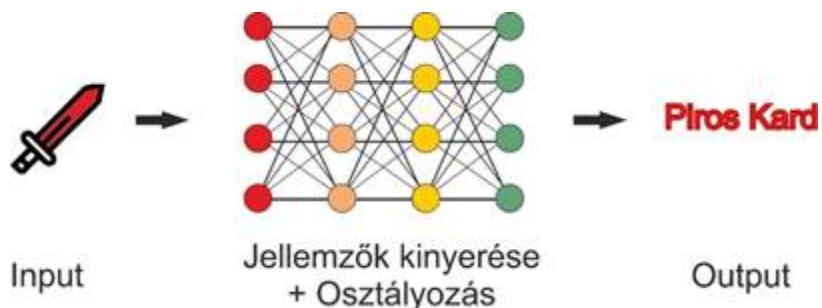
A karakterek életüket élik a játékmenet során. Az interakciók velük rendkívül élethűek. Ez egy fontos tényező a játék körül kialakult vita mögött. Érdemes megemlíteni az AI taxi vezetőket, akik A pontból B-be juttatnak a játékos útvonaltervezése nélkül, vagy éppen egy AI szarvast, amint él az életében a San Andreas világában. Az in-game weboldalak frissülnek a játék során, és a karakterek valósághű beszélgetéseket folytatnak a telefonhívásainon.

De manapság már a játékok olyan információkat rögzítenek, amelyek még soha nem álltak rendelezésünkre arról, hogy mikor és, hogyan játszunk (Big Data). A fejlesztők ezen adatok alapján személyre szabhatják a játékélményt, növelve ezzel a bevételt az egyéni költési szokások iránti trendek felhasználásával. Ezekkel a módszerekkel további fejlett játékélményeket várhatunk.

3. fejezet

A feladat megoldásához rendelkezésre álló technikák ismertetése.

3.1. Mi is az Deep learning?



3.1. ábra. Deep learning

Mélytanulás- Az Intelligencia Fejlődése a Számítógépekben

A mélytanulás, a mesterséges intelligencia (MI) és gépi tanulás egy alcsoportja, melyben mesterséges neurális hálózatokat alkalmaznak, az emberi agy működését utánozva. Ezen technológia lehetővé teszi a számítógépek számára, hogy bonyolult adatokat dolgozzanak fel és értelmezzék, valamint összetett mintázatokat ismerjenek fel különböző adatformákból, mint például képek, szövegek vagy hangok. A mélytanulásnak számos alkalmazási területe van, beleértve az autonóm járművek vezetését, az egészségügyi diagnosztikát és a nyelvi feldolgozást.

A mélytanulási modellek nagy adathalmazokból tanulnak, ezeket könnyen kezelhető komponenseikre bontják, így képesek komplex problémák megoldására. A mélytanulás alkalmazása széleskörű, beleértve digitális asszisztenseket, hang vezérelt távirányítókat, csalásdetektálást, automatikus arcfelismerést, valamint olyan új technológiákat, mint az önvezető autók és a virtuális valóság. A mélytanulási modelleket adattudósok tanítanak algoritmusok segítségével, amelyeket üzleti alkalmazásokban használnak, adatalemzésre és előre

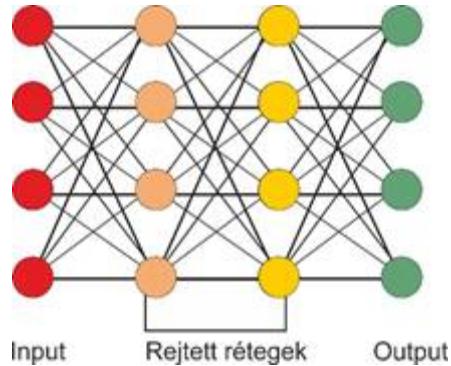
jelzésre. A modellek végtelen rugalmasságuknak köszönhetően számos különböző területen hasznosíthatjuk őket: az önvezető autók képfelismerése, orvosi képfelismerés segítségével a daganatsejtek kiszűrését, a védelmi rendszereket, a műholdképek automatikus azonosítását, az gyártásbiztonság szempontjából fontos helyzetek érzékelését stb. A mélytanulás alkalmazásait az alábbi csoportokba sorolhatjuk, látás, beszédfelismerés, természetes nyelvfeldolgozás (NLP) és ajánlórendszerek.

ajánlórendszerek.

- *Látás* Képekből és videókból információk kinyerése, például tartalommoderáció, arcfelismerés vagy kép osztályozás céljából.
- *Beszédfelismerés* Az emberi beszéd elemzése, például virtuális asszisztensek működtetése vagy beszéd alapú szövegfelismerés.
- *NLP* Szövegek és dokumentumok jelentésének kinyerése, például automatizált virtuális ügynökök vagy dokumentumok összefoglalása.
- *Ajánlórendszerek* Személyre szabott ajánlások készítése az ügyfélpreferenciák elemzésével.

A mélytanulás hatékonyan kezeli az előre nem definiált, strukturálatlan adatokat, felfedezi a rejtett kapcsolatokat és mintákat. Idővel alkalmazkodik az ellenőrizetlen tanuláson keresztül, és kiemelkedő teljesítményt nyújt változékony adatfeldolgozási környezetben. A mélytanulás számára kihívásokat tartalmaz, amikor nagy mennyiségi és változó minőségű adatokat szükséges kezelni és az input adathalmazban a pontatlanságok veszélyének elhárítása és a számítási teljesítmény fokozása az igény.???ennek a mondatnak semmi értelme A mélytanulás felhőalapú futtatása, sok kihívást leküzd. A gyorsabb modellek kifejlesztésére, képzésére, telepítésére nyújt lehetőséget, valamint elősegíti az infrastruktúra könnyű hozzáférését.

3.2. Mi is az a Nerural Network (neurális hálózat)?



3.2. ábra. Nerural Network

A neurális hálózat egy olyan gépi tanulási módszer amelyet az emberi agy összetett működése ihletett. Kifinomult gépi tanulási technikákat használ az adatok elemzéséhez és megalapozott döntések meghozatalához.

A neurális hálózat lényegében mesterséges neuronok, más néven perceptronok egymásra kapcsolódó rétegeiből áll. Ezek a neuronok együtt dolgoznak és továbbítanak információkat. A neurális hálózat általában egy bemeneti rétegből, egy vagy több rejtett rétegből és egy kimeneti rétegből áll.

A bemeneti réteg feladata, hogy a releváns adatokat továbbítsa a rejtett rétegek felé, majd a kimeneti réteg felé. Az adatok továbbítása közben a rejtett rétegekben az adatokon aktivációs függvények, és súlyozott összegek sora dolgozik, így ismeri fel a neurális háló az adatok közötti kapcsolatokat és mintákat.

A rejtett rétegekben lévő neuronok aktivációi továbbítódnak a következő rejtett rétegekbe vagy a kimeneti rétegbe, amely a modell kimenetét generálja. A kimeneti réteg konkrét formátuma az adott probléma természetétől függően változik.

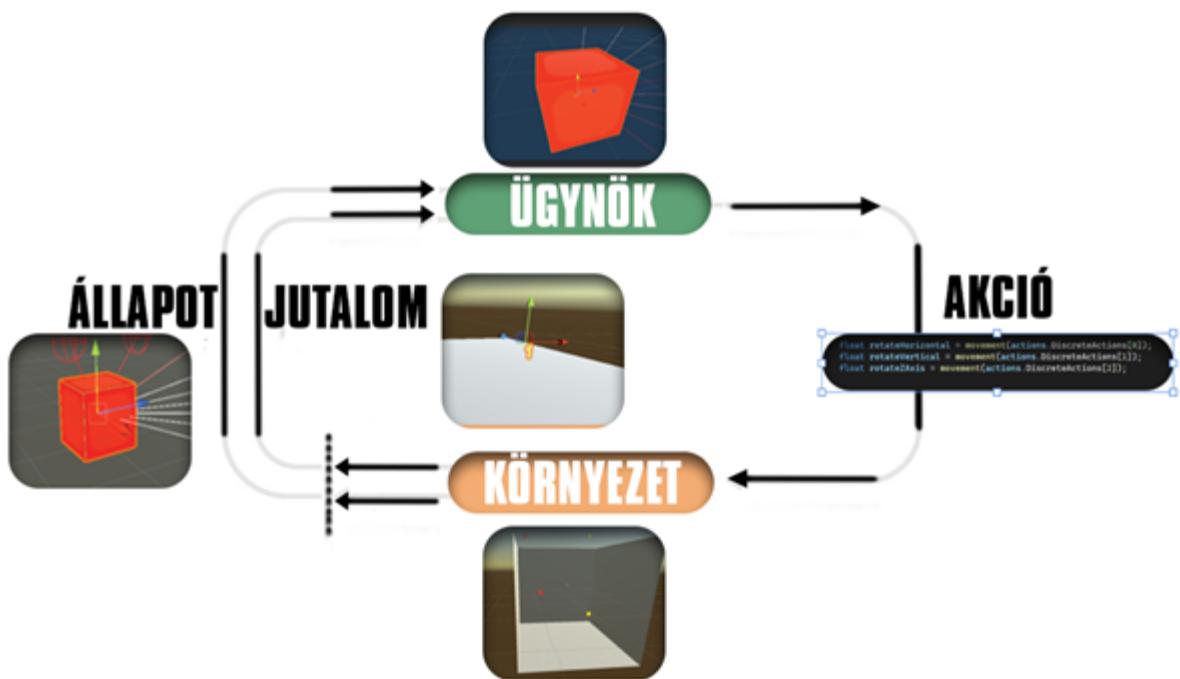
Egy neurális hálózat tanulása valójában azt jelenti, hogy a súlyozásokat és a paramétereket optimalizálják ameddig nem megfelelő a kimenet generálása. Sokféle probléma megoldására alkalmazhatóak a neurális hálózatok mint, képfelismerésre, beszédfelismerésre, természetes nyelvfeldolgozásra, adatbányászatra és még sok más területre.

3.3. Mi is az Reinforcement Learning? (Megerősítő Tanulás)

A Megerősítő Tanulás (Reinforcement Learning - RL) egy izgalmas paradigmája a mesterséges intelligenciában, amely során egy mesterséges ügynök interakcióba lép a környezetével, és kísérletezés és hiba útján tanul a környezettel való kölcsönhatásokból. Az RL módszer mögötti alapötlet az, hogy az ügynök a visszajelzéseket, vagyis pozitív vagy negatív jutalmakat

kapja a végrehajtott cselekedetekért, és ezen keresztül sajátítja el a feladatok megoldását.

Gondoljunk egy gyerekre, aki felfedezi a világot. Meglát egy tál cukorkát, és ha magához veszi, pozitív élményt kap, mivel finom. Ezzel megtanulja, hogy érdemes ismételni ezt a cselekedetet. Ugyanakkor, ha megéríti a forró kávét, negatív élményt tapasztal, és megtanulja, hogy kerülje el ezt a cselekedetet. A Megerősítő Tanulás hasonló módon működik, ahol az ügynök a környezettel való interakció során pozitív vagy negatív visszajelzéseket kapva tanul a helyes döntésekéről. A Mély Megerősítő Tanulás (Deep Reinforcement Learning) mély neurális hálózatokat alkalmaz a Megerősítő Tanulási problémák megoldására - innen ered a "mély" elnevezés.



3.3. ábra. Reinforcement Learning

Erősítéses Tanulás Keretrendszer az RL folyamatot általában négy fő elem alkotja. Állapot (state), cselekvés (action), jutalom (reward) és következő állapot (next state). Az ügynök a jelenlegi állapotban megfigyeli a környezetét, végrehajt egy cselekvést, ami befolyásolja a környezetét, majd megkapja a cselekedetéért járó jutalmat. Ezután a rendszer átvált a következő állapotba, és a folyamat újra kezdődik.

Az állapot (S_0 azaz State) az ügynök érzékeli, hogy a környezet aktuális állapotát, amely lehet például egy játékban a játékos pozíciója vagy egy robot esetében a szenzorok által mért adatok.

Cselekvés (A_0 azaz Action) az ügynök választ egy cselekvést a rendelkezésre álló opciónál, például egy lépést egy játékban vagy egy irányt egy robottal.

Jutalom (R_0 azaz Reward) a végrehajtott cselekvésért az ügynök pozitív vagy negatív jutal-

mat kap a környezettől, ami visszajelzés a cselekvés helyességéről vagy helytelenségéről. Következő Állapot (S1 azaz Next State)a környezet változik az ügynök cselekvése miatt, és az ügynök átvált a következő állapotba, ahol a folyamat újra elkezdődik.

RL Folyamata folyamatot általában Markov Döntési Folyamatok (MDP) keretében írják le, ahol az állapotok, cselekvések és jutalmak időbeli sorrendben követik egymást. Azonban meg kell tennünk egy megkülönböztetést az Observation és a State között. A State a világ állapotának teljes leírása (nincs rejtett információ) teljesen észlelt környezetben. Egy sakk-játékban teljes állapotot kapok a környezettől, mivel hozzáférünk az egész játékhöz. Más szóval a környezet teljesen észlelt. Ezzel szemben az Observation az állapot részleges leírása, részben észlelt környezetben. Tekken-ben csak a játékos közelében lévő pálya részletét látjuk, így megfigyelést kapok. A cselekvések diszkrét vagy folytonos térből származhatnak. Diszkrét tér a lehetséges cselekvések száma véges..Gondoljunk bele a Marió-ban csak 4 lehetséges cselekvésünk van balra, jobbra, fel (ugrás) és le (guggolás). Addig a folytonos térben szinte végtelen számú lehetséges cselekvés található. Érdemes még beszélnünk a Jutalmak és diszkontálásról. A jutalom alapvető fontosságú A Megerősítő Tanulásban (RL), mert ez az egyetlen visszajelzés az ügynök számára. Ennek köszönhetően az ügynök tisztában van azzal, hogy a végrehajtott cselekvés jó volt-e vagy sem. A kumulatív jutalom minden időpillanat t esetén így írható le: A kumulatív jutalom egyenlő a sorozatban szereplő összes jutalom összegével. Ami ekvivalens azzal:

$$R(t) = r(t+1) + r(t+2) + R(t+3) + \dots \quad (3.1)$$

Azonban a valóságban nem adhatjuk egyszerűen össze őket így. Az előbb érkező jutalmak (a játék elején) valószínűbbek, mivel kiszámíthatóbbak, mint a hosszú távú jutalmak. Ezért meghatározunk egy diszkontálási rátát, amit gamma-nak hívunk. Ez 0 és 1 közötti érték lehet, leggyakrabban 0,95 és 0,99 között van. Minél nagyobb a gamma, annál kisebb a diszkont. Ez azt jelenti, hogy az ügynök jobban érdeklik a hosszú távú jutalmak. Másrészről, minél kisebb a gamma, annál nagyobb a diszkont. Ez azt jelenti, hogy az ügynök jobban érdeklik a rövid távú jutalmak (a legközelebbi sajt). Ezután minden jutalmat diszkontálunk gamma a lépései idő exponensével.

$$R(t) = r(t+1) + \gamma \cdot r(t+2) + \gamma^2 \cdot R(t+3) + \dots \quad (3.2)$$

Meg kell említenünk még néhány érdekes módszert, hogy minél effektívebbek legyünk. Ez a felfedezés és a kiszákmányolás (exploration/exploitation). Ebben a kettőben kell megtalálnunk az egyensúlyt különben sohasem találjuk meg az optimális rendszert. A felfedezés a környezet felderítését jelenti a véletlenszerű cselekvések kipróbálásával, annak érdekében, hogy több információt szerezünk a környezetről. A kiszákmányolás pedig a ismert információ kihasználását jelenti a jutalom maximalizálása érdekében.

Nem beszéltünk még az ügynök agyáról a policyről. Ez a Policy függvény megmondja

nekünk, milyen cselekvést kell végrehajtanunk egy adott állapotban. Ezt a Policyt tanulni szeretnénk, a célunk az optimális Policy megtalálása, amely a várható visszatérítést maximálja, amikor az ügynök annak megfelelően cselekszik. Ezt a Policyt a képzés (training) során találjuk meg.

Két megközelítés van az ügynök képzéséhez, hogy megtalálja ezt az optimális Policyt:

Közvetlenül, az ügynöknek tanítjuk meg, hogy megtanulja, milyen cselekvést hajtson végre a jelenlegi állapot alapján: Policy-alapú módszerek. Vagy, közvetetten, az ügynöknek megtanítjuk, hogy tanulja meg, melyik állapot értékesebb, majd válassza azt a cselekvést, amely az értékesebb állapotokhoz vezet. Az egyik közvetlen módszer a Policy-Alapú módszer. Ez a függvény meghatározza az egyes állapotok hozzárendelését a legjobban megfelelő művelethez. Alternatív megoldásként meghatározhat egy valószínűségi eloszlást az adott állapot lehetséges műveletei között.

Két típusú Policy-nk van:

- Determinisztikus: egy adott állapotban a politika minden ugyanazt a cselekvést adja vissza.
- Sztochasztikus: valószínűségi eloszlást ad ki a cselekvések fölött.

Egy kezdeti állapot esetén sztochasztikus politikánk valószínűségi eloszlásokat ad ki a lehetséges cselekvések fölött az adott állapotban. Nem csak Polcy alapú módszerek vannak, hanem Érték-Alapú Módszerek is ami ahelyett, hogy egy Policy függvényt tanulnánk, egy értékfüggvényt tanulunk meg, amely egy állapotot a várható értékhez rendeli. Az állapot értéke az az elvárt diszkontált visszatérítés, amit az ügynök elérhet, ha abban az állapotban kezd, majd az általunk meghatározott politika szerint cselekszik. Érdemes még megemlíteni az Epsilon-greedy stratégiát: Gyakran használt stratégia A Megerősítő Tanulásban, amely a felfedezés és a kiszákmányolás egyensúlyának megtartását célozza. Választja azt a cselekvést, amelynek a legmagasabb várható jutalma van egy 1-epsilon valószínűsséggel. Választ egy véletlenszerű cselekvést epsilon valószínűsséggel. Az epsilon általában idővel csökken, hogy a hangsúlyt a kiszákmányolás felé tolja. Ez a stratégia lehetővé teszi a folyamatos felfedezést és az ismert optimális cselekvések kihasználását.

A Greedy stratégia minden azokat a cselekvéseket választja, amelyek várhatóan a legnagyobb jutalomhoz vezetnek, az aktuális környezet ismeretére építve. (Csak kiszákmányolás) Mindig azt a cselekvést választja, amelynek a legnagyobb várható jutalma van. Nem tartalmaz felfedezést. Hátrányos lehet bizonytalansággal vagy ismeretlen optimális cselekvésekkel rendelkező környezetekben.

Off-policy algoritmusok: A különböző politika használata a tanulási és a következtetési időpontokban. On-policy algoritmusok: Ugyanazt a politikát használja a tanulás és a következtetés során.

Monte Carlo és Temporal Difference tanulási stratégiák: Monte Carlo (MC): Tanulás az epizód végén. A Monte Carlo esetében várunk, amíg az epizód be nem fejeződik, majd

frissítjük az értékfüggvényt (vagy politikafüggvényt) egy teljes epizódból származó adatok alapján.

Temporal Difference (TD): Tanulás minden lépésnél. A Temporal Difference tanulás során minden lépésnél frissítjük az értékfüggvényt (vagy politikafüggvényt) anélkül, hogy teljes epizódra lenne szükség.

3.3.1. Q-learning

Q-Learning a Megerősítő Tanulás (RL) algoritmus, amely tréningel egy Q-függvényt, ami egy akció-érték függvény, amit egy Q-tábla kódol a belső memóriájában, és tartalmazza az összes állapot-akció párhozott értékeit.

		Akciók						Akciók			
		balra	jobbra	fel	le			balra	jobbra	fel	le
Állapotok	1	0	0	0	0	Állapotok	1	0	0.1	0	0
	2	0	0	0	0		2	0.5	0	0	0.9
	3	0	0	0	0		3	0	0	0.6	0
	4	0	0	0	0		4	0	0	0	0.5
	5	0	0	0	0		5	0	0	0	0
	6	0	0	0	0		6	0	0	0	0

3.4. ábra. Q-learning

A Q-függvény megkeresi a Q-táblájában a megfelelő értéket. Amikor a tréning befejeződik, rendelkezünk egy optimális Q-funkcióval, vagy vele ekvivalensen optimális Q-táblával. És ha van egy optimális Q-funkció, akkor egy optimális policyink van, mivel tudjuk, hogy minden állapotban melyik a legjobb akció. 2 stratégia van, hogy megtaláljuk az optimális policyt. Még hozzá a Policy -alapú módszer és az Érték-alapú módszer. A policy általában egy neurális hálózattal tanítják meg, hogy kiválassza, milyen cselekedetet hajtson végre egy adott állapotban. Ebben az esetben a neurális hálózat kimenete az az akció, amelyet az ügynöknek meg kell tennie, ahelyett, hogy egy értékfüggvényt használnánk. Az élménytől függően a neurális hálózatot újra állítják, és jobb cselekedeteket szolgáltat. Az Érték- alapú módszer közvetetten találja meg az optimális policyt egy olyan érték vagy action-value függvény kiképzésével, amely megmondja nekünk minden állapot vagy állapot-cselekvés párra vonatkozó értéket.

3.4. Deep Reinforcement learning (Mély Megerősítő Tanulás)

A DRL abban különbözik a sima Reinforcement Learning-től, hogy itt mély neurális hálózatot (DNN) használunk sima q-learning helyett.

3.5. Unity



3.5. ábra. Unity

A Unity Engine az egyik legelterjedtebb játékfejlesztő motor, amely lehetővé teszi szinte bármely platformon játékok és animációk készítését. Támogatja a külső szoftverekkel készült modellek használatát, és a vezérlést C# nyelven írhatjuk hozzá.

A Unity szoftver fejlődése, hasonlóan más világszerte ismert termékekhez, különleges kezdettel rendelkezik. David Helgason, a Unity társalapítója és vezérigazgatója, egy alagsorban indította el a projektet. Az első verzió 2005-ben jelent meg Mac OS X-re, és 2006-ban elnyerte az Apple Design Awards ezüstérmét a Mac OS X grafika legjobb felhasználása kategóriában.

2007-ben a 2.0-ás verzió több mint 50 új funkciót hozott, beleértve a "Terrain Editor" alkalmazást is.

2008-ban az Apple App Store megjelenése után gyorsan megjelent az iPhone támogatás, annak ellenére, hogy sokáig vitatott volt annak alkalmassága.

2010-ben a 3.0-ás verzióval a Unity kilépett az Apple árnyékából, és már PC-re, konzolokra és Androidra is lehetett játékot és animációt fejleszteni. Bemutatták az automatikus UV-mapping funkciót, amely lehetővé tette 2D képek alapján 3D objektumok létrehozását.

2012-ben a 4.0-ás verzió bemutatásakor már több mint 1 millió fejlesztő használta. Ez a verzió DirectX11 és Flash támogatást, valamint Linux preview-t is tartalmazott.

2013-ban a Facebook integrált egy Unity alapú SDK-t, amellyel a játékokat össze lehetett kötni a felhasználói profilokkal. Ez lehetővé tette a játékosoknak személyre szabott

hirdetések megkapását jáék közben, valamint az eredmények megosztását és a jáék közbeni láthatóságuk növelését.

2015-ben a Unity 5-ös verzió megjelenésével a motor teljes multiplatform fejlesztői palettát célozta meg. Ebben az időszakban elérhetővé vált a WebGL támogatás, Nintendo Switch, Facebook Gameroom, Google Daydream és Vulkan API. A motor továbbá felhőalapú szolgáltatásokat, Nvidia PhysX 3.3-at, új audió motort és valós idejű globális megvilágítás funkciót is kapott.

2016-ra a Unity már elérhető volt PC-s platformon, lehetővé téve a világ szociális hálóján való asztali számítógépes vagy laptopos játékok a weboldalon keresztül, akár egyedül, akár másokkal.

Sajnálatos módon a jövőt illetően talán nem lehetünk túl boldogok mivel 2023 Szepember 12-én A Unity Technologies bejelentette, hogy 2024-től vezetik az új "Runtime Fee"-t amely a fejlesztőktől a jáék minden egyes telepítésénél díjat szedne be. Az ingyenes Unity motor használóit 200,000 letöltés után \$0.20 telepítésenként terhelné, míg a Unity Pro előfizetőknek csak 1,000,000 telepítés után lenne alkalmazva alacsonyabb díj. Korábban A Unity ingyenesen biztosította a telepítéseket.

A döntés széles körű fejlesztői tiltakozást váltott ki. Az új "Runtime Fee"-t sokan A Unity Technologies vezérigazgatójára, John Riccietello-ra fogták, aki korábban az EA vezérigazgatója volt és agresszív monetizációra tett javaslatokat. Az eredeti tervek szerint a díj minden telepítéskor felszámolódott volna, beleértve a jáék újratelepítéseit, Game Pass telepítéseket, demókat és jótékonyiségi csomagok telepítéseit.

A fejlesztők aggodalmukat fejezték ki, különösen az indie piacon, Az Unity Technologies később visszavonta néhány eredeti "Runtime Fee" funkcióját az erős tiltakozások hatására, kijelentve, hogy csak a Unity Pro és Unity Enterprise felhasználókat érinti a változás. Reméljük a jövőben még ezen is változtatni fognak.

3.5.1. Deep learning Unityben

A Unity biztosít egy nagyszerű eszközt a deep learning-hez, ez az ML-Agents, egy rendkívül izgalmas és erőteljes eszközökészlet, amely lehetővé teszi a mély tanulás és mesterséges intelligencia integrálását Unity-alapú játékokba és szimulációkba. Ez a platform lehetőséget kínál a fejlesztőknek arra, hogy intelligens ügynököket, karaktereket és viselkedéseket hozzanak létre, amelyek képesek tanulni és alkalmazkodni a környezetükhez. Az ML-Agents lehetővé teszi a környezet részletes definiálását, beleértve az állapotteret, a cselekvésteret és a jutalmazási struktúrát. Ez lehetővé teszi a fejlesztők számára, hogy testre szabhatnak a szimulációs környezetet a specifikus feladatokhoz. Az ügynökök viselkedését egyedi scriptekkel lehet definiálni. Ezek a scriptek határozzák meg, hogy az ügynökök, hogyan észlelik a környezetüket, milyen cselekvéseket hajtanak végre, és, hogyan reagálnak a jutalmakra vagy büntetésekre. Az ML-Agents támogatja a különböző mély tanulási algoritmusokat, például a

Proximal Policy Optimization (PPO) és a Soft Actor-Critic (SAC). Ezek a fejlett algoritmusok lehetővé teszik az ügynökök hatékony tanítását a környezetükön keresztül. A rendszer lehetőséget ad a felhasználóknak a mély tanulási algoritmusok hiperparamétereinek finomhangolására. Ezáltal a fejlesztők szabályozhatják a tanulási folyamatot és optimalizálhatják a tanulási teljesítményt. Az ML-Agents lehetővé teszi az ügynökök tanítását és értékelését. A tanítási folyamat során az ügynökök kölcsönhatásba lépnek a környezettel, és a mély tanulási algoritmusok segítségével optimalizálják a viselkedésüket. Az eszközök szorosan integrálódik a Unity fejlesztői környezettel, lehetővé téve a fejlesztők számára, hogy könnyedén hozzáférjenek az ML-Agents funkcióihoz és optimalizálják a játékukban vagy szimulációjukban szereplő intelligens ügynökök viselkedését.

3.6. Unreal



3.6. ábra. Unreal

Az Unreal Engine egy olyan szoftverfejlesztői eszközökészlet, amelyet a Epic Games fejlesztett ki, és széles körben használják a videojáték- és egyéb interaktív tartalomfejlesztők. Az Unreal Engine 1998-ban jelent meg először, és azóta több verziója látott napvilágot.

A Unreal Engine erősen fókusztál a 3D-s grafikára és a valós idejű számításokra, és kiválóan alkalmas olyan alkalmazásokhoz, mint videojátékok, szimulációk, virtuális valóság (VR) és augmented reality (AR) projektek, valamint filmszerű animációk.

A motort széles körben használják a játékfejlesztési iparágban, és olyan népszerű címekhez adott alapot, mint a Fortnite, a Gears of War sorozat, az Unreal Tournament és sok más játék. Az Unreal Engine rendelkezik egy erőteljes grafikai motorral, amely támogatja a magas felbontású textúrákat, a valós idejű árnyékolást, az összetett fizikai alapokon nyugvó szimulációkat és más fejlett grafikai funkciókat.

Az Unreal Engine továbbá egy sokoldalú eszközökészletet kínál a fejlesztőknek, amely magában foglalja a Blueprint nevű vizuális szkriptelési rendszert, amellyel a programozási ismeretek nélkül is lehetőség van játékok és interaktív tartalmak létrehozására. Emellett a

fejlesztők az Unreal Engine forráskódjához is hozzáférhetnek, amely további testre szabási lehetőségeket biztosít.

Az Unreal Engine folyamatos fejlődésen megy keresztül, a különböző verziókban új funkciók, optimalizációk fejlesztői eszközök jelennek meg, így segítve a fejlesztőket a magas minőségű és innovatív tartalmak létrehozásában.

3.6.1. Deep learning Unrealban

Természetesen Unrealban is megtalálható az az eszköz amellyel gépi tanulással taníthatjuk a mesterséges intelligenciával. A neve Learning Agents. Ez egy Unreal Engine (UE) bővítmény, amely lehetővé teszi a mesterséges intelligenciával (MI) rendelkező karakterek kiképzését gépi tanulás segítségével. Ez lehetővé teszi a hagyományos játék MI kiegészítését vagy helyettesítését, például azokat, amelyeket viselkedési fák vagy állapotgépek segítségével írtak. Különösen a bővítmény lehetőséget biztosít a Megerősítő Tanulás és az utánzó tanulás módszerek alkalmazására. Hosszú távon a Learning Agents célja az, hogy hasznos legyen különböző alkalmazásokban, beleértve a fizikai alapú animációkat, játékos NPC-eket és automatizált minőségellenőrzési teszteket stb.

A Learning Agents nem egy általános célú GT keretrendszer. A bővítmény minden aspektusa a karakterdöntéshozatal szempontjából lett kialakítva. De meg kell jegyezni, hogy a Learning Agent egy viszonylag új plugin, és még nincs hozzá megfelelő számú dokumentáció.

A Unity és az Unreal Engine mindkettő népszerű játékfejlesztő platform, de a két rendszer különböző megközelítéseket alkalmaz a mesterséges intelligencia (MI) implementálására a játékokban. Az alábbiakban összehasonlítom a Unity által használt ML-Agents rendszert a Deep Learning megközelítésével, ami az Unreal Engine-ben is lehetséges.

A Unity ML-Agents egy sajátos eszközkészlet, amely lehetővé teszi a fejlesztők számára a mesterséges intelligencia és gépi tanulás alkalmazását a játékokban. Az ML-Agents framework segítségével könnyedén implementálhatók agensek, amelyek képesek tanulni és fejlődni. Az Unityben történő ML-Agents használat számos előnnyel jár. Az egyik legfontosabb az, hogy a rendszer könnyen integrálható a Unity fejlesztői környezetébe, így a fejlesztők könnyen vizualizálhatják, finomhangolhatják és tesztelhetik mesterséges intelligencia viselkedését.

Ezenkívül az ML-Agents rendszer támogatja a reinforcement learning-et és a tanulás folyamatának gyorsításához tervezett környezeteket, amelyek segítségével optimalizálni lehet az MI algoritmusokat. A Unity által nyújtott könnyű használhatóság és a kiterjedt dokumentáció segíti a fejlesztőket a hatékony MI implementációban. Másrészt az Unreal Engine lehetővé teszi a fejlesztők számára a saját deep learning alapú rendszerek kialakítását. Ez lehetőséget ad a nagyobb rugalmasságra és testreszabhatóságra, de ugyanakkor nagyobb kihívásokat is jelenthet a fejlesztők számára. A deep learning implementálása a Unreal Engine-

ben magasabb szintű szaktudást igényelhet, és a fejlesztési folyamat komplexebb lehet.

Azonban érdemes megjegyezni, hogy mindenki megközelítésnek megvannak a saját előnyei és alkalmazási területei. Az ML-Agents a könnyű használhatóság és a gyors implementáció révén ideális lehet kevésbé tapasztalt fejlesztők számára, míg a deep learning azoknak lehet vonzó, akik nagyobb kontrollt szeretnének az MI felett, és hajlandók nagyobb kihívásokkal szembenézni. A döntés a konkrét projekt igényeitől és a fejlesztői készségektől függ.

3.7. Unity vs Unreal

Az Unreal Engine és a Unity minden platform kiemelkedő játékfejlesztő eszközök, de egyes projektigényektől függően az egyik vagy a másik lehet jobb választás. Az alábbiakban összehasonlítom a két rendszert, és bemutatom, miért lehet előnyös az egyik a másik felett. Az Unreal Engine valóban rendkívül erős grafikus képességekkel rendelkezik, de átlagos fejlesztő számára ez lehet, hogy túlzottan is robusztusnak tűnik. A Unity grafikus képességei azonban sok esetben még mindig kielégítők és könnyebben kezelhetők. A Unity könnyen elsajátítható C# kódolást támogatja, amit sokan egyszerűbben tanulnak meg, mint a C++. Az egyszerűbb kezelhetőség és a gyorsabb tanulási görbe sok fejlesztő számára előnyt jelent. Továbbá ismereteim a C# nyelvben sokkal mélyebbek mint a C++-ban ami egy nagyon nagy előny volt számomra. A Unity Asset Store-ban széles választékban elérhetőek a kiegészítők, beleértve az ingyeneseket is. Ez lehetővé teszi, hogy könnyen bővítsem projektemet és gyorsabban haladjak a fejlesztés során. A Unity nagyobb felhasználói és fejlesztői táborral rendelkezik, amely elősegíti a közösségi támogatást. A nagyobb közösség gyakran eredményez gyorsabb és hatékonyabb problémamegoldást. Az Unreal Engine 5%-os bevételmegosztást kér a projektek után, míg a Unity teljesen ingyenes a százalékos bevételrig a régebbi verzióknál, sajnos a, hogy fentebb említettem a jövőben már ez nem lesz így. De az Unreal is változtatni fog a licensen a jövőben. A Unity rendelkezik részletes dokumentációkkal, amelyek segítik a fejlesztőket. Emellett, a Unity Machine Learning Agents (ML-Agents) keretrendszer tényleg erős és fejlett, rendelkezik széleskörű dokumentációval és aktív közösséggel. Ezért úgy érzem a Unity ideális választás, hogy megvizsgáljuk képesek vagyunk-e mesterséges intelligenciát tervezni 3d-s környezetben egy Tron játékban.

3.8. Pytorch

PyTorch egy nyílt forráskódú gépi tanulás könyvtár Pythonhoz, amely egy rugalmas és dinamikus számítási gráfot biztosít a mélytanulási modellek hatékony fejlesztéséhez és tanításához. A Facebook mesterséges intelligencia kutatólaboratórium (FAIR) fejlesztette ki, és népszerűsége az egyszerű használat, a dinamikus számítási gráf és a neurális hálózatok fejlesztésének erős támogatása miatt nőtt.

Továbbá a PyTorch dinamikus számítási gráfot használ, ami azt jelenti, hogy a gráf a műveletek végrehajtása során épül fel a helyszínen. Ez nagyobb rugalmasságot biztosít a modellek építése és hibakeresése során. A PyTorch tenziókat használ az neurális hálózatmodellek létrehozásához és munkához. A tenziók hasonlóak a NumPy tömbökhez, de továbbfejlesztett funkciókkal rendelkeznek, amelyeket a mélytanuláshoz optimalizáltak. A PyTorch tartalmaz egy automatikus differenciálás könyvtárat, az Autogradot, amely automatikusan kiszámolja a tenziók gradienseit egy adott veszteségfüggvény szerint. Ez elengedhetetlen a neurális hálózatok gradient alapú optimalizálásához. A PyTorch tartalmazza a "torch.nn" modult, amely egyszerűsíti a neurális hálózatmodellek építését és tanítását. Tartalmaz előre meghatározott rétegeket, veszteségfüggvényeket és optimalizációs algoritmusokat. A PyTorch támogatja a buzgó végrehajtást, amely lehetővé teszi a műveletek végrehajtását egy inkább imperatív, Python-szerű módon. Ez megkönnyíti a kód megértését, hibakeresését és kísérletezését. A PyTorchnak élénk és aktív közössége van, és széles körben használják mind a kutatásban, mind az iparban. Gazdag ökoszisztémával rendelkezik olyan könyvtárak és eszközök formájában, amelyek a PyTorchhoz épültek, megkönnyítve a más népszerű gépi tanulás és mélytanulás keretrendszerrel történő integrációt.

3.9. TensorFlow

TensorFlow egy másik nyílt forráskódú gépi tanulás és mélytanulás könyvtár, amelyet a Google fejlesztett ki. Ahogyan a PyTorch, a TensorFlow is széles körben használt a kutatásban és az iparban a gépi tanulás és neurális hálózatok fejlesztéséhez. Statikus és Dinamikus Számítási Gráfok TensorFlow eredetileg statikus számítási gráfot használt, ami azt jelentette, hogy a gráfot előre definiálták és csak egyszer volt futtatható. Azonban a TensorFlow 2.0-tól kezdve támogatja a dinamikus számítási gráfokat is, ami hasonló rugalmasságot biztosít, mint a PyTorch. Az alacsony erőforrásigényű eszközökön történő futtatáshoz, például mobil eszközökön és beágyazott rendszereken, a TensorFlow Lite egy könnyített változata. A TensorFlow-be épített TensorBoard nevű eszköz segítségével könnyen vizualizálhatók a tanítás során keletkező gráfok, metrikák és egyéb információk. Ez egy a gyártásban való alkalmazáshoz szánt kiterjesztés, amely lehetővé teszi a gépi tanulási modellek bevezetését, értékelését és követését a valós környezetben. Lehetőséget nyújt a TensorFlow modellek JavaScript környezetben történő futtatására, például böngészőben. TensorFlow rendelkezik egy nagy és aktív közösséggel, és sok további eszköz, kiterjesztés és harmadik féltől származó könyvtár áll rendelkezésre, amelyek segítik a modellek fejlesztését és alkalmazását. Ez egy olyan platform, amely lehetővé teszi a modellek újrafelhasználását és megosztását más fejlesztőkkel.

Mivel az ML-Agents közvetlenül támogatja a PyTorch-ot a tanulási algoritmusok implementálásához. A PyTorch kompatibilitás hozzájárulhat a könnyű integrációhoz és optimalizált modellek készítéséhez. Ezért választottam inkább a PyTorchot.

4. fejezet

Saját teljesítmény előállításához ténylegesen felhasznált eszközök részletes ismertetése ; telepítés, használatba vétel előfeltételei és lépései.

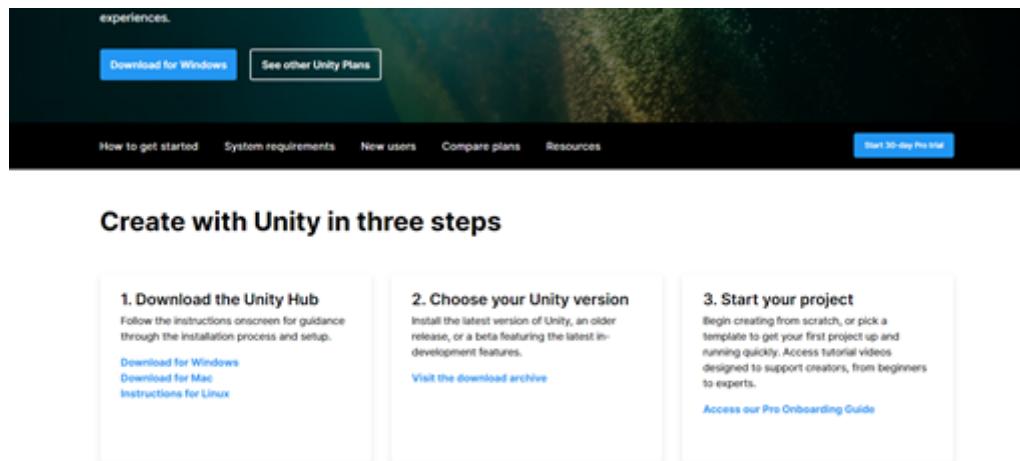
4.1. Unity

Természetesen ahogy fentebb említettem a választásom a motorok terén a Unityre esett. A Unity telepítése rendkívül könnyű és felhasználóbarát folyamat. Az eredeti weboldalról letölthető, ahol a felhasználók egyszerűen követhetik a telepítővarázsló lépéseit. A platform támogatja a legtöbb operációs rendszert, beleértve a Windows-t, a macOS-t és a Linuxot, így a fejlesztők szabadon választhatnak a saját preferenciáiknak megfelelő rendszer mellett. A Unity erős közösségi támogatással rendelkezik, ahol a fejlesztők tapasztalataikat megoszthatják, kérdéseket tehetnek fel, és inspirációt meríthetnek másuktól. Az online dokumentáció és tutorialok segítik a felhasználókat az alkalmazás mélyebb rétegeinek megértésében és hatékonyabb használatában. A Unity a fejlesztők számára többféle licenclehetőséget is kínál, beleértve az ingyenes verziót is, amely sok esetben már kielégíti a kezdeti projektigényeket. Ha diákként vesszük igénybe akkor sokkal több funkció érhető el benne. A fizetős opciónak pedig általában megfizethetők és rugalmasak, így még a kisebb költségvetésű fejlesztők számára is elérhetővé válik a platform teljes funkcionalitása. Ezáltal a Unity a fejlesztők számára könnyen elérhetővé válik, és az alacsony költségekkel kombinálva kiváló választás a kreativitás kibontakoztatásához és különböző digitális projektek létrehozásához.

4.1.1. Telepítés

Látogass el a URL: <https://Unity.com/download> weboldalra,
és kattints a "download for windows" vagy "See other Unity Plans" (Business) verziót

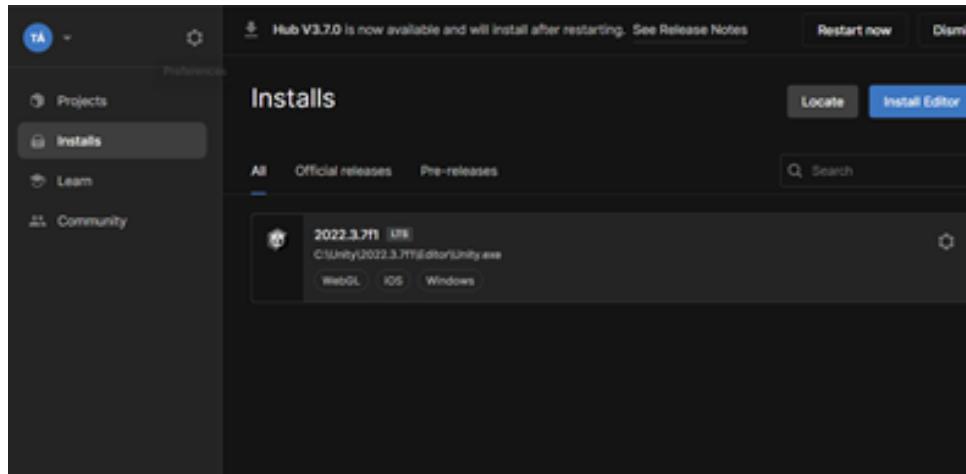
szeretnél.



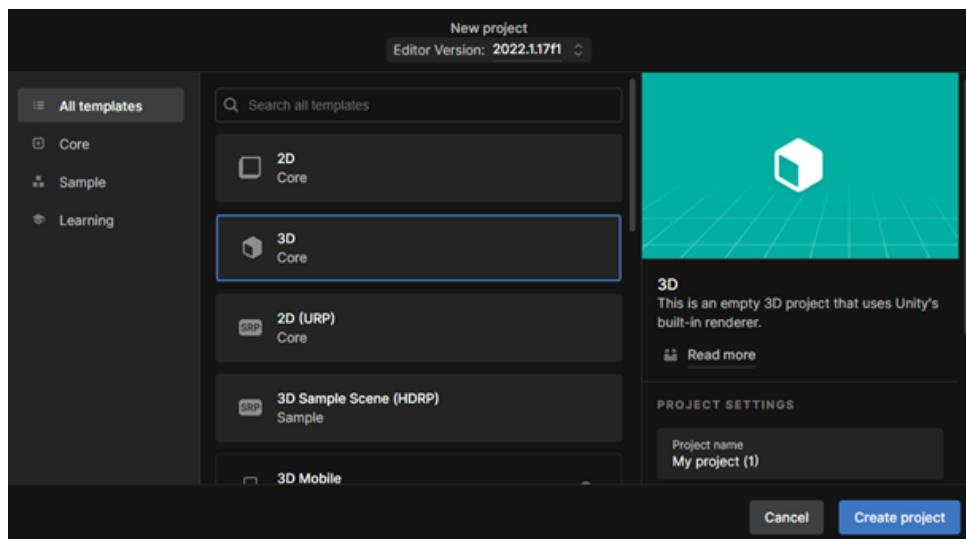
4.1. ábra. Unity Letöltés

Amennyiben már rendelkezel fiókkal, jelentkezz be. Ha még nincs fiókod, hozz létre egyet. A Unity fiókkal lehetőséged lesz projekteket menteni és különböző erőforrásokhoz hozzáérni. Válaszd ki, hogy melyik licencet szeretnéd. Ha ingyenes (Personal) licencet választasz, ügyelj arra, hogy a projektjeid a licenc feltételeivel összhangban legyenek. Kövesd a weboldal utasításait, hogy eljuss a Unity telepítőjének letöltéséhez. A Unity Hub is letöltésre kerül, ami egy kezelőfelület a projektjeid és a Unity verzióid kezeléséhez. A letöltött fájlokat telepítsed a számítógépedre. A Unity Hub segítségével hozz létre egy fiókot vagy jelentkezz be. A Unity Hub főoldalán válaszd ki a "Installs" fület. Kattints az "ADD" gombra, majd válaszd ki a Unity verziót, amelyet telepíteni szeretnél (ez 2022.3.7f1-legyen az ML Agents kompatibilitása miatt.).

Ezután kattints a "Next" gombra. Válaszd ki a szükséges kiegészítőket, például a Visual Studio SDK-t, platformtámogatásokat, nyelveket és dokumentációkat. Kattints a "Next" gombra. Ellenőrizd a kiválasztott beállításokat, majd kattints a "Done" és azután a "Install" gombra a telepítés elindításához. Várj, amíg a telepítés befejeződik. Amikor kész, a Unity Hubban megjelenik a telepített Unity verzió. Nyisd meg a Unity Hubot, válaszd ki a projekt-könyvtáradat, majd kattints az "New projekt" gombra egy új projekt létrehozásához. Az új projekt beállítása után már nekiláthatsz a Unity használatának és projektfejlesztésnek.



4.2. ábra. Unity Telepítés



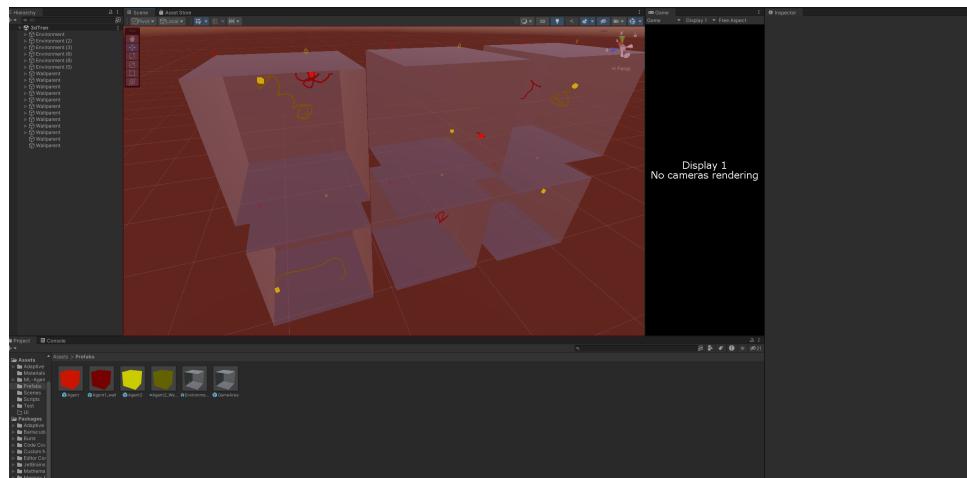
4.3. ábra. Új projekt 3d-ben

Miután a játék vagy jelenet betöltődik, egy viszonylag egyszerű, szürke felület fogad minket. A felső részen található egy menüsor, a bal oldalon az objektum hierarchia, a jobb oldalon az Inspector, amely az objektum tulajdonságainak beállítására szolgál. Az alsó részen pedig láthatjuk a betöltött és betölthető Asset-eket, valamint a Console ablakot.

A látvány szempontjából a középpontban a grafikus ablak található, amelyen egy kamera és egy alapértelmezett fényforrás helyezkedik el, hasonlóan a Blenderhez. Az alkalmazás indításakor négy fül található itt Scene, Game, Asset Store, Animator. Ezek azonban teljesen átrendezhetők, duplikálhatók, és felcserélhetők, hogy megfeleljenek a fejlesztő preferenciáinak.

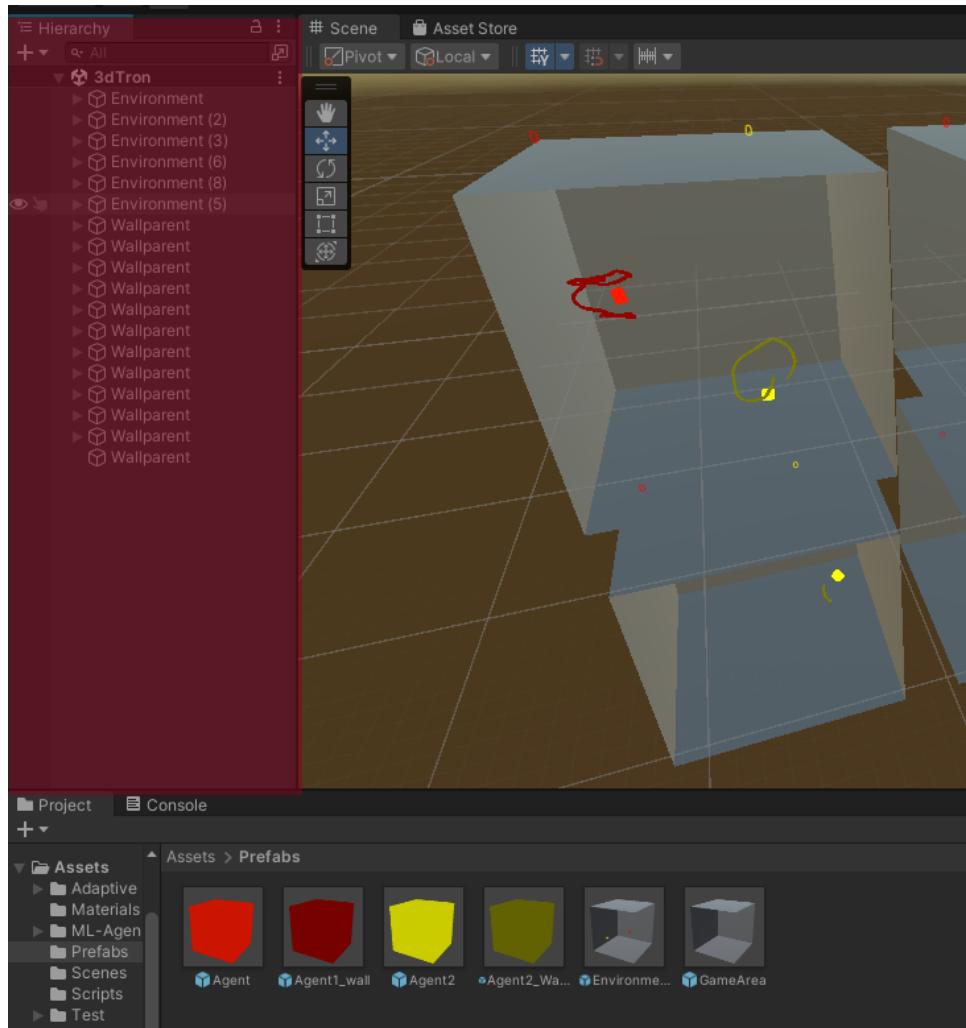
Ez a felépítés lehetőséget biztosít arra, hogy a fejlesztő személyre szabja az ablakokat és a lapfüleket, optimalizálva azokat a munkafolyamatuk és munkastílusuk szerint. A bal oldali hierarchiában megjelenített objektumok és a jobb oldali Inspector tulajdonságokkal

való finomhangolása segíti a fejlesztőt a jelenet és a játék elemek precíz kezelésében.



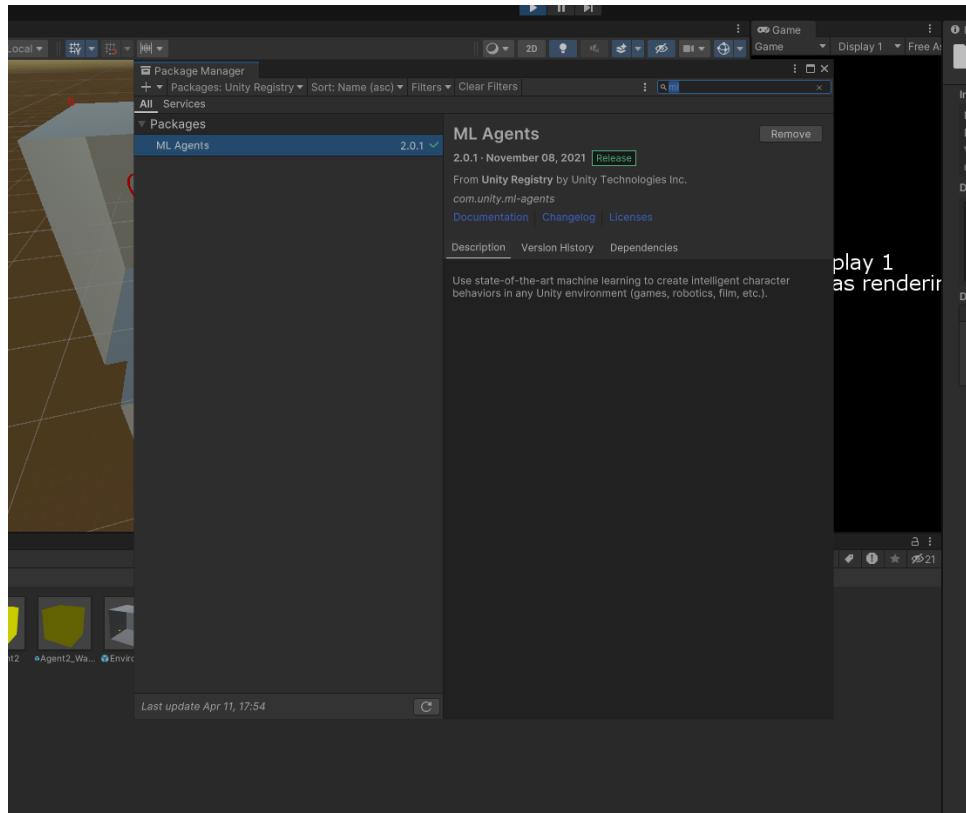
4.4. ábra. Unity Jelenet

A Unityben a "Scene" területen tervezhetjük meg a játékunk jelenetét. minden, a Unityben készült tartalom egy jelenetként funkcionál. Az új objektumokat közvetlenül a Game-Object menüből adhatjuk hozzá, vagy az Asset fülön található elemeket is egyszerűen beilleszthetjük a drag-and-drop funkció segítségével.



4.5. ábra. Assets

Amikor a "Game" fülre váltunk, elindítva a játékot, a jelenet betöltődik, attól függően, hogy éppen mit készítünk. Fontos megjegyezni, hogy ha a kis lejátszás gombbal elindítjuk a játékot vagy a jelenetet, az aktuális állapotot menti, beleérte az objektumokat, tulajdonságokat és C# kódokat. Amíg a futtatás nem fejeződik be, a Scene fülön végzett módosítások nem érvényesülnek az aktív játékon vagy a projekten, mivel a lejátszás leállításával visszaáll az utoljára mentett állapot.



4.6. ábra. Packet manager -> Ml Agents

4.2. ML Agents(2.0.1)

A Unity ML-Agents egy eszköztár a Unity játékmotorhoz, amely lehetővé teszi intelligens ügynökök képzésére szolgáló környezetek létrehozását. Legyen szó előre elkészített vagy saját terem létrehozásáról a Unity-ben, az ML-Agents segít az ügynökök képzésében ezeken a virtuális helyszíneken. A Unity Technologies által fejlesztett eszköztár használata a mai játékokban is egyre elterjedtebb, mint például a Firewatch, Cuphead és a Cities Skylines. A toolkit hat fő komponenst tartalmaz:

- Tanulási Környezet: Tartalmazza A Unity jelenetet (a környezetet) és a környezeti elemeket (játék karaktereket)
- Python Alacsony-szintű API: A környezettel való interakcióhoz és manipulációhoz szolgáló alacsony szintű interfész, amelyet a képzés indításához használunk.
- Külső Kommunikátor: Az Egységen készült Tanulási Környezetet C# összeköti a Python alacsony-szintű API-val.
- Python Trénerek: A PyTorch által készített megerősítési algoritmusok (PPO, SAC stb.).

- PettingZoo Wrapper: A "PettingZoo" a több ügynökös változata a gym wrapper-nek.
- Gym Wrapper: Az RL környezetet egy "gym wrapper"-be burkolózva.

A Tanulási Komponens belsejében két fontos elem található. Az ügynök komponens: A jelenet színésze, amelyet az optimalizált policyt (amely megmondja, milyen cselekvést hajtsunk végre minden állapotban) útján képzünk. A policyt az agynak nevezik. Az Akadémia összetevő irányítja az ügynököket és döntéshozatali folyamataikat. Gondoljunk erre az Akadémiára úgy, mint egy tanárra, aki kezeli a Python API kéréseket. A Tanulási Folyamatot egy hurokmodellként lehet elkezelní, amely állapot, cselekvés, jutalom és következő állapot sorozatokat hoz létre. Az ügynök célja az elvárt kumulatív jutalom maximalizálása. Az Akadémia feladata a következő:

- Megfigyelések gyűjtése.
- Cselekvés kiválasztása a policy segítségével.
- Cselekvés végrehajtása.
- Visszaállítás, ha elértek a maximális lépésszámot, vagy ha végeztünk.

4.3. Telepítés

Megnyitom a Unity Porjectben a Package Manager-t, rákattintok a packages- nél a Unity Registry-re és megkeresem az ML-agent-et, majd jobb oldalt rákattintok az install gombra. Ami 2.0.1-es verzió.

Python 3.10.0rc2

Release Date: Sept. 7, 2021

This is the first release candidate of Python 3.10

This release, **3.10.0rc2**, is the last preview before the final release of Python 3.10.0 on 2021-10-04. Entering the release fixes are allowed between release candidates and the final release. There will be no ABI changes from this point forward as possible.

Call to action

We **strongly encourage** maintainers of third-party Python projects to prepare their projects for 3.10 compatibility during the release cycle. You can track progress in our [issue tracker](#).

4.7. ábra. Pytorch

Ezek után fel kell telepíteni Pythont, méghozzá a python-3.10.0rc2 verziót, mivel ezzel lesz csak kompatibilis az ML-Agents. Méghozzá a Windows 64-bit-es verziót.

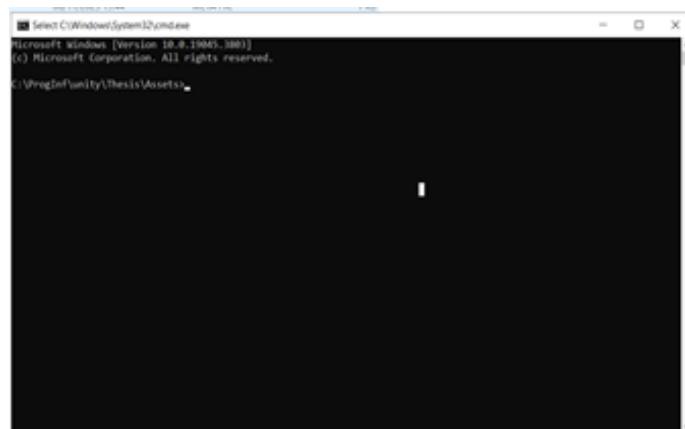


4.8. ábra. Pytorch install 1



4.9. ábra. Pytorch isntall 2

Miután letöltöttem a file-t el, kell távolítani minden másfél Python a gépről. Majd kezdhetjük a telepítést.



4.10. ábra. CMD

Mindenképpen pipáljuk be az add python chekboxot. Különben nem érzékeli, , hogy fel van telepítve.

Ezek után átmegyünk a Unity projekt mappájába és megnyitjuk a Command Prompt-ot létrehozok egy virtuális környezetet ahol dolgozni fog az ML-Agents. Ezt a python -m venv venv parancssal hozzuk létre.



4.11. ábra. Venv mappa

Ezek után aktiválnunk kell ezt a mappát, hogy ebbe dolgozzunk, ezt az 'venv_scripts_activate' parancssal tudjuk megtenni.

Fel kell telepítenünk a python package managert is 'python -m pip install' – upgrade pip

```
venv\ C:\gamedev\unity\ml\agents\venv>python -m pip install --upgrade pip
Requirement already satisfied: pip in c:\gamedev\unity\ml\agents\venv\lib\site-packages (22.0.4)
Collecting pip
  Using cached pip-23.1.2-py3-none-any.whl (2.1 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 22.0.4
    Uninstalling pip-22.0.4:
      Successfully uninstalled pip-22.0.4
Successfully installed pip-23.1.2
```

4.12. ábra. Pytroch PIP install

Ezek után fel kell telepíteni az ML-agentset is az' ip install mlagents' parancssal.

Majd fel kell telepíteni a pytorchot is ami maga a mély tanulási keretrendszer, mely segíteni fogja az mlagents-et.

Ezek után jöhet a PyTorch keretrendszer feltelepítése 'pip3 install torch torchvision torchaudio' parancssal.

A nagy mennyiségű adatátvitel miatt fel kell telepítenünk a protocol buffert 'pip install protobuf== 3.20.3 ' parancssal.

Ha mindenkel megvagyunk megnézhetjük, hogy minden rendben van-e 'mlagents-learn -help' parancssal.

Alapból a PyTorch cpu-val dolgozik, de ha szeretnénk átállíthatjuk gpu-ra is. Ha Nvidia kártyánk van, akkor biztosnak kell lennünk, hogy fel-van e telepítve a gépünkre a CUDA. '--torch-device CUDA parancssal' állíthatjuk ezt be.

5. fejezet

Saját munkánk (alkalmazásfejlesztés, mérés, tervezés stb.) részletes leírása, az eredmények szemléletes ismertetése.

Az alap koncepcióm az, hogy készíték egy Tron (vagy rendes névén Light Cycles) játékhoz hasonló játéket 3 dimenziós térben, ahol 2 AI-t deep reinforcement learningel fogok kiképezni egymás ellen. Majdan a kiképzett agyat vissza implementálva kapok egy ai-t amivel akár mi is megküzdhetünk. Az eredeti játékban az játékosok virtuális motorosokként vesznek részt versenyzésben és stratégiai küzdelmekben a játéktéren. Amint elindul a játék, a játékosok virtuális motorosa egy adott ponton a játéktéren megkezdi mozgását. Ahogy a motoros halad, elhagy egy fényszálat maga mögött, amely ami nyomot hagy a pályán. A játékteret két dimenzióban modellezik, és a játékosok motorosa csak előre és oldalra tud mozogni. A motorosoknak kerülniük kell a saját és az ellenfél fényszálát, valamint azokat az akadályokat, amelyek a játéktér szélén találhatóak. A játék célja, hogy a játékosok minél hosszabb ideig éljenek, miközben megpróbálják megsemmisíteni az ellenfeleket úgy, hogy azok beleütközzenek a fényszálba vagy az akadályokba. A játék célja többdimenziós. Elsősorban is, a játékosnak a leghosszabb ideig kell élnie a játéktéren anélkül, hogy beleütközne a saját vagy az ellenfél fényszálába vagy az akadályokba. Másodsorban, a játékosoknak meg kell próbálniuk megsemmisíteni az ellenfeleiket. Ennek két módja van: vagy az ellenfél beleütközik a játékos által hagyott fényszálba, vagy a játékos képes megfelelően manipulálni a mozgását, hogy az ellenfélbe vezesse, és így az beleütközzön. A játékosoknak taktikusnak kell lenniük a mozgásuk során, megpróbálva megjósolni az ellenfél következő lépését és megfelelően reagálni rá. Én játékos helyett 2 AI-t fogok tréningelni az ML-Agents segítségével a unity keretrendszerben. Először is megtervezem az alapját a Deep Reinforcement Learningnek.

5.1. Állapottér:

- Az állapottér az összes lehetséges konfigurációt jelenti, amelyben a játék környezete lehet. A Tron játék esetében az állapottér tartalmazhatja:
 - Az összes játékos pozícióját és irányát.
 - A falak pozícióit.
 - minden játékos jelenlegi sebességét.
 - Hogy egy játékos felnak vagy másik játékosnak ütközött-e.

5.2. Akciótér:

- Az akciótér meghatározza az összes lehetséges lépést, amelyet az ügynök megtehet. A Tron játék esetében az akciók lehetnek:
 - Előre lépés.
 - Balra fordulás.
 - Jobbra fordulás.

5.3. Jutalmi struktúra:

- A jutalmi struktúra az ügynök által egy lépés után azonnal kapott visszajelzés. A Tron játék esetében a jutalmak alapulhatnak:
 - Túlélés: Az ügynök pozitív jutalmat kap azért, hogy életben marad.
 - Ellenfelek legyőzése: Az ügynök jutalmat kap azért, hogy az ellenfeleket falával ütközésre kényszeríti.
 - Ütközések: Az ügynök negatív jutalmat kap, ha felnak vagy ellenfeleknek ütközik.

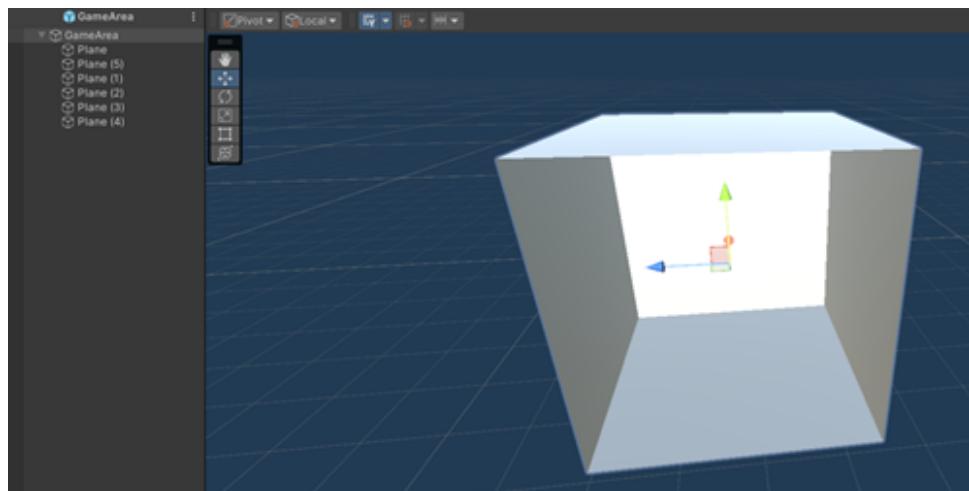
5.4. További fontolóra veendő szempontok:

- Játékállapot reprezentációja: Döntsük el, hogyan képviseljük a játékállapotot olyan formátumban, amely alkalmas a neurális hálózatba való bevezetésre. Ez lehet a játékállapot átalakítása numerikus vagy kategorikus reprezentációvá.
- Diszkretizáció: Attól függően, hogy melyik algoritmust választjuk, lehet, hogy diszkretizálnunk kell az állapottér vagy az akciótér tételeit, hogy könnyebben kezelhetővé váljon a tanítás során.

Most, hogy feltelepítettük az ML-Agentset a Unity Projektben létre kell hoznunk egy környezetet., hogy egyszerre akár több Ügynököt is ki tudjunk képezni, illetve így lokális koordinátákkal tudunk operálni a különböző környezetekben.Jobb klikk a Hierarchy panelre és ott hozzunk létre egy új Empty Objectet. Nevezzük el Environment-nek

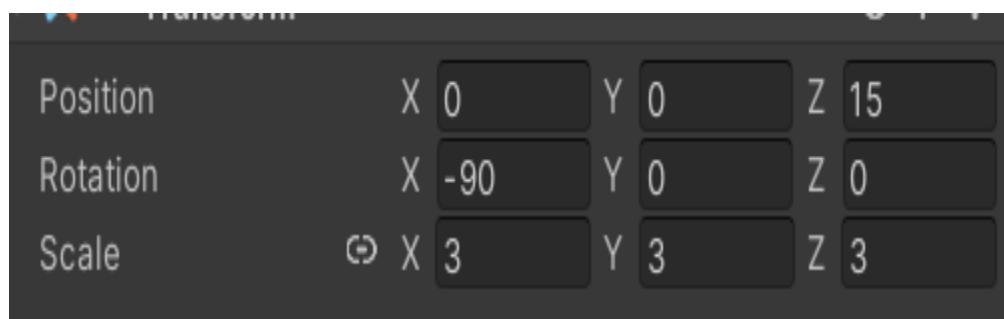


5.1. ábra. Enviroment mappa

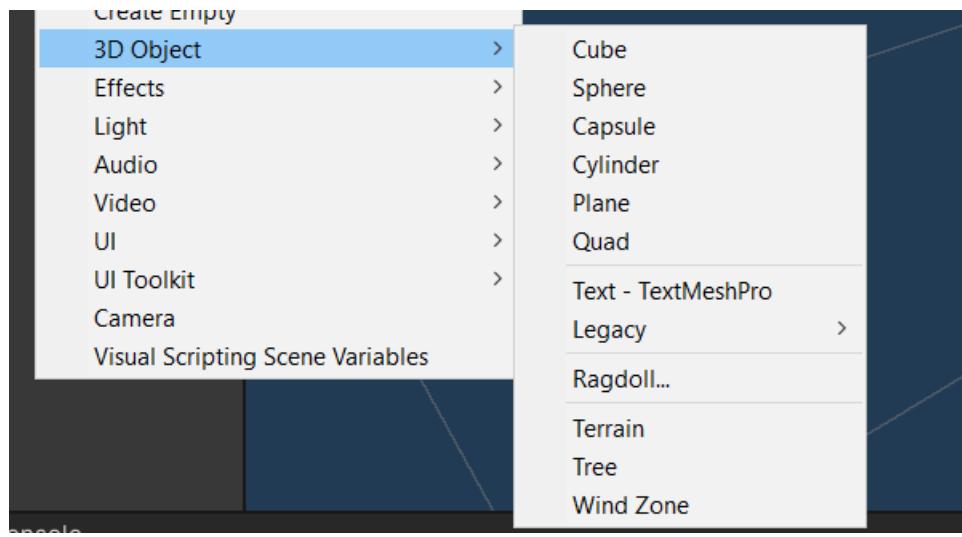


5.2. ábra. Game area

Most hozzunk létre 4 plane-t mint a pálya falait, és helyezzük el az Environment -ben megfelelő koordinátákkal és forgatásokkal. A Plane előnye, hogy átlátunk az egyik oldalon illetve kapok vele egy előre definiált Mesh Renderert és egy Mesh collidert ami segít az ügynökünknek, hogy érzékelje és nekünk, hogy lássuk az objektumot. Ezeket tegyük bele egy külön GameArea objektumba és az egészet belehelyezzük.



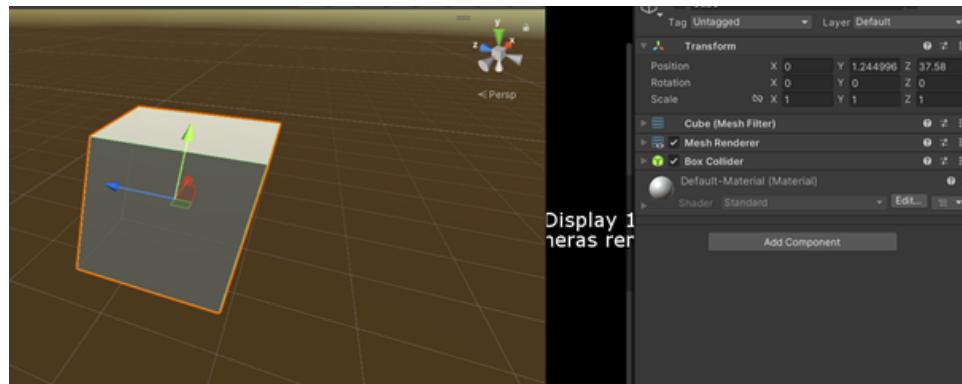
5.3. ábra. Game area



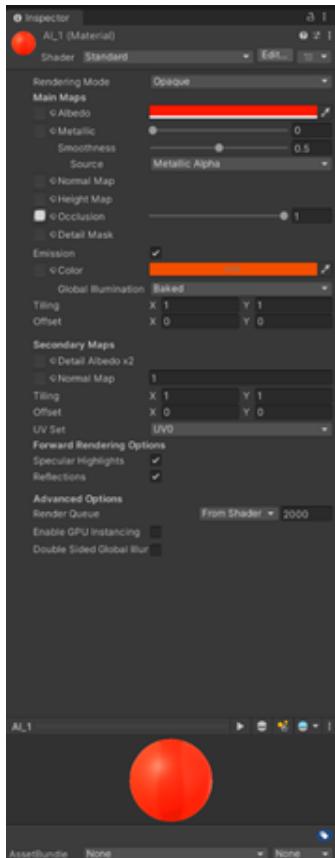
5.4. ábra. Transform

Ezek után létrehozok egy Ügynököt (kockát) a Hierarchi fülre kattintva jobb klikkel, az ott lenyíló menüben rákattintunk a 3D Objectre, azon belül is a Cube-ra,

Majd létrehozok egy Ügynököt (kockát) a Hierarchi fülre kattintva jobb klikkel, az ott lenyíló menüben rákattintunk a 3D Objectre, azon elül is a Cube-ra. kapok egy objectet aminek automatikusan része egy Box Collider, és egy Mesh Renderer. Ez lesz az egyik ügynökünk. evezük is el Agent1-nek,



5.5. ábra. Kocka létrehozás



Ennek az ügynöknek adok egy színt, hogy meg tudjuk különböztetni. Hozzunk létre egy Materialt a projektben, jobbklikk->Create->Material. Erre rákattintva adjunk meg egy színt a Inspectorban, majd ezt a materialt húzzuk rá a Scenen lévő Agent1-re.

Így megvan az első ügynökünk. Ezt másoljuk le, a CTRL-C-vel és CTRL-V -vel. Az új kockának hozzunk létre egy új materialt, és nevezzük el Agent2-nek. Ezt a 2 kockát a Hierarchy fülön tegyük bele az Environment-be. Természetesen kell egy referencia is mint fal amit maga mögött hagy az Agent, így tehát hozzunk létre 2 külön kockát amely a majdani fal lesz, és ezeket is beletesszük az Environment-be

Az eredményeket is szeretném kijelezni, hogy éppen melyik Agent áll nyerésre. Emiatt Objectumot hozunk létre, majd utána az add componentel adok neki egy Text Mesh-t. Ezáltal ide tudjuk majd kiiratni a számokat. ezeket is bele helyezzük az Environment-be.

Most, hogy nagyjából minden benne van az Enviromentben, fogjuk az egészet és lehúzzuk a projektfülbe, hogy prefabot (Prefabricated) hozzunk létre belőle, ezzel biztosítva, hogy ne csak a Scenen legyenek elmentve a dolgok, és ha többet akarunk létrehozni, akkor ne keljen egyesével módosítani az objektumokat mert amikor a prefabot módosítjuk, akkor automatikusan frissül az összes belőle származtatott példány is.

5.5. Scriptek

Most, hogy létrehoztam egy egyszerű Scene-t, elkezdhetjük a scriptek megírását illetve az ML-Agents beállítását.

A scripteknek létrehozok egy külön mappát. Ide jobb kikkel Create C#Script létrehozok egy scriptet és elnevezzük AI-nak.

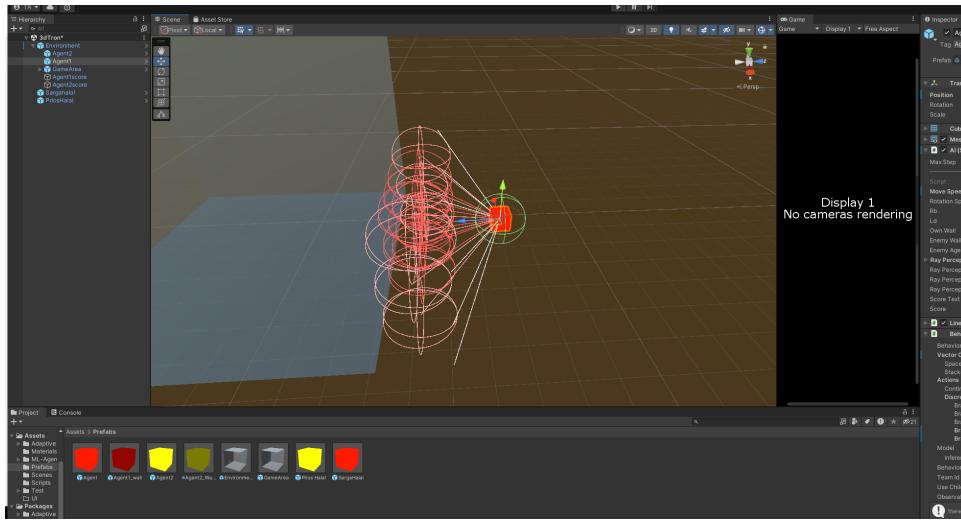
Ahogy megnyitjuk látjuk, hogy kapok egy előre létrehozott osztályt ami a MonoBehaviour-ból származtatok. A MonoBehaviour A UnityEngine névtér része és egy olyan osztály, amelyet a Unity motor használ a játékobjektumok viselkedésének leírására. És kapok 2 funkciót egy Start-ot és Egy Update-t. A Start metódus egy alkalommal hívódik meg, amikor a játékobjektum először aktívvá válik, míg az Update metódus minden képkocka előtt lefut, lehetővé téve az időben változó viselkedések kezelését.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ai1 : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      void Start()
9      {
10      }
11
12      // Update is called once per frame
13      void Update()
14      {
15      }
16
17  }
18
19
```

5.6. ábra. Unity Script

Ezekre most nem lesz szükség, úgy, hogy kitörlöm őket és a MonoBehaviour helyett Agentet írok, hogy abból származzon. Ami által megkapjuk a Behaviour Prarameters-t. Illetve a using Unity.MLAgents;-et beírjuk felülre, hogy hozzáférjünk a névteréből származtatott metódusokhoz. Menjünk vissza a Unitybe és a létrehozott Scriptet húzzuk rá az Agent1-re. Észrevesszük, hogy nem csak az AI Script jelent meg a kockánkon hanem a Behaviour Parameters is. Itt több paraméter is van, amit szabadon változtathatok. Az első legfontosabb, hogy megszabjuk az ügynökünk mennyi információt legyen képes kezelni. Mivel a térben mozog, ezért mindenkorban kellenek a rotation axisok illetve látni is fog, továbbá szerencsére előre autómatikusan halad tőlünk függetlenük, ezért beállítjuk 4-re az Actions-öknél a Continous Action.

A "Continuous Action" (folytonos akció) az ügynökök által végrehajtandó akciók típusát jelöli. A folytonos akciók olyan értékek, amelyek tartományon belül bármilyen valós számok lehetnek, szemben a "Discrete Action" (diszkrét akció) típussal, ahol az akciókat diszkrét értékekre korlátozzák. A gyorsaság kedvéért inkább Discrete Branches-t használjuk, hogy egész értékekkel dolgozhassak. Ügynökünk folyamatosan fog előre haladni, így azzal nem kell foglalkozni, de a forgatást erre bízzuk így beállítok 3-mat. Még egy fontos dolog a Behaviour Type-ban adhatjuk meg azt, hogy manuálisan (heuristic) esetleg scriptel akarjuk megadni az ügynökünk környezettel való interakcióját vagy pedig tanulás(inference) útján. Mi azt szeretnénk hogyha magától tanulna meg minden, ezért a defaulton hadjuk, ami az inference-t jelenti. Ezek után bepipáljuk a Chold Sensors-t, amit majdan hozzá adok scriptként és ez lesz a szeme az agents-nek.



5.7. ábra. raycast

Térjünk vissza az AI scripthez. Először is írjuk meg az ügynökünk irányítását.

```

1
2
3 public override void OnActionReceived(ActionBuffers actions)
4 {
5     float rotateHorizontal = movement(actions.DiscreteActions
6         [0]);
7     float rotateVertical = movement(actions.DiscreteActions
8         [1]);
9     float rotateZAxis = movement(actions.DiscreteActions[2]);
10    Move(rotateHorizontal, rotateVertical, rotateZAxis);
11 }
```

Megírjuk a public override void OnActionReceived(ActionBuffers actions) Ez a függvény felülírja az Agent osztály OnActionReceived függvényét. Az OnActionReceived az egyik legfontosabb callback függvény az ügynökök számára, és ebben határozhat meg az ügynök által végrehajtandó akciókat. float rotateHorizontal = movement(actions.DiscreteActions[0]); A DiscreteActions egy ActionBuffers objektum része, amely tartalmazza a diszkrét (egész értékű) akciókat. Itt az ügynök első diszkrét akcióját (index 0) veszi ki, majd a movement függvényen keresztül kapja meg a hozzárendelt értéket. float rotateVertical = movement(actions.DiscreteActions[1]); Hasonlóan, az ügynök második diszkrét akcióját veszi ki, és a movement függvényen keresztül kapja meg az értéket. float rotateZAxis = movement(actions.DiscreteActions[2]); Az ügynök harmadik diszkrét akcióját veszi ki, és ismét a movement függvényen keresztül kapja meg az értéket. Move(rotateHorizontal, rotateVertical, rotateZAxis); Ez a sor meghívja a Move függvényt, amely a tényleges mozgást végzi a környezetben az ügynök számára. Mivel egyfolytában előre halad csak a forgatást fogja

megtanulni. Az átadott paraméterek (rotateHorizontal, rotateVertical, rotateZAxis) az ügynök által kapott akciók alapján határozzák meg a mozgás irányát. A mozgatási értékekhez társítok 3 értéket.

```
1 private int movement(int input) {  
2     switch (input) {  
3         case 1:  
4             return -1;  
5         case 2:  
6         case 3:  
7             return 0;  
8         default:  
9             break;  
10    }  
11    return 0;  
12 }
```

Ez által a bejövő értékeket transzformálom, hogy a forgatáshoz használható értékeket kapjak vissza.

```
1  
2 public void Move(float rotateHorizontal, float rotateVertical  
, float rotateZAxis)  
3 {  
4  
5     rb.velocity = this.transform.forward * moveSpeed;  
6  
7     yaw = rotateHorizontal * rotationSpeed;  
8  
9     pitch = -rotateVertical * rotationSpeed;  
10  
11    roll = -rotateZAxis * rotationSpeed;  
12  
13    Quaternion deltaRotation = Quaternion.Euler(new Vector3(pitch  
, yaw,  
14 roll) * Time.deltaTime);  
15  
16    rb.MoveRotation(rb.rotation * deltaRotation);  
17  
18 }
```

Amint tovább küldöm ezeket az értékeket az rb.velocity = this.transform.forward * mo-

veSpeed; Ebben a sorban az ügynök sebessége beállításra kerül. Az rb egy Rigidbody komponens, és this.transform.forward az ügynök előre mutató irányát reprezentálja. Ezt szorozzuk meg a moveSpeed értékkel, ami az ügynök sebességét szabályozza. Ezáltal az ügynök a saját előre mutató irányába mozog. yaw = rotateHorizontal * rotationSpeed;, pitch = -rotateVertical * rotationSpeed;, roll = -rotateZAxis * rotationSpeed; Itt az yaw, pitch és roll változókba mentjük az ügynök kívánt forgatási értékeit. Ezek a forgatási értékek a kapott horizontális, vertikális és Z tengely menti értékekből származnak, és szorozva vannak a rotationSpeed változóval, amely az ügynök forgási sebességét szabályozza. Quaternion deltaRotation = Quaternion.Euler(new Vector3(pitch, yaw, roll) * Time.deltaTime); Itt kiszámoljuk a kívánt forgatást egy Quaternion objektumban a pitch, yaw és roll értékekből. A Quaternion.Euler segítségével könnyen készíthetünk egy forgatási quaterniót az Euler szögekből. rb.MoveRotation(rb.rotation * deltaRotation); Ebben a sorban alkalmazzuk az ügynök rotációját a kiszámolt forgatással. rb.MoveRotation segítségével beállíthatjuk az ügynök rotációját a kiszámolt delta rotációval.

```

1
2 public override void CollectObservations(VectorSensor sensor)
3
4 {
5
6 sensor. AddObservation((rb.transform.localPosition));
7
8 sensor. AddObservation(rayPerceptionSensor);
9
10 }
```

Természetesen nem csak forgásra képes az ügynökünk hanem látásra is ezért felülírom a public override void CollectObservations(VectorSensor sensor) metódust. A sensor. AddObservation(rb.transform.localPosition); az ügynök aktuális pozícióját adja hozzá az észlelési vektorhoz. Az rb egy Rigidbody komponens, és transform.localPosition az ügynök aktuális pozícióját reprezentálja a játék világában. Az AddObservation függvény segítségével ezt a pozíciót hozzáadjuk az észlelési vektorhoz, amely az ügynök által a tanulási algoritmusnak átadott megfigyeléseket tartalmazza. sensor. AddObservation(rayPerceptionSensor); egy olyan érzékelő által kapott megfigyeléseket adok hozzá az észlelési vektorhoz, amelyet rayPerceptionSensor változó reprezentál. rayPerceptionSensor- RayCastal képes érzékelni a taggel megjelölt objekteket, és meghatározni azoknak a távolságát, így el tudja kerülni az esetleges ütközéseket az ügynök. Ezeket az észleléseket is hozzáadjuk az észlelési vektorhoz.

Unity MonoBehaviour osztályban található OnEpisodeBegin függvényt, az ml-agents körérendszerben egy tanulási epizód (learning episode) kezdetén hívja meg. A tanulási epizó-

dok olyan időszakokat jelölnek, amikor az ügynök tanulása vagy tréningje zajlik, és a környezettel való interakció során meghatározott feladatokat kell elvégeznie, így ezt felülírva beírjuk a kezdeti állapothoz szükséges adatokat.

```
1  
2  
3 public override void OnEpisodeBegin()  
4 {  
5  
6     rb = GetComponent<$Rigidbody>();  
7     ld = GetComponent<$LineDrawer>();  
8     rb.velocity = Vector3.zero;  
9     ld.besierPontok = new List<$Vector3>();  
10    ld.Wallparent = new GameObject();  
11    ld.Wallparent.name = "Wallparent";  
12    ld.Wallparent.tag = "Wallparent";  
13    rayPerceptionSensors = gameObject.GetComponents<  
        $RayPerceptionSensorComponent3D>();  
14    transform.localPosition = new Vector3(rx, ry, rz);  
15    ld.stopCoroutine = false;  
16 }
```

Az OnEpisodeBegin metódus egy fontos része a Unity játékfejlesztésnek, különösen a Reinforcement Learning (RL) alapú projekteknek. Amikor egy játékmenet (episode) újra kezdődik, ez a metódus felelős azért, hogy az objektumok és a játékállapot inicializálva legyenek a következő játékmenethez. Az első lépés a metódusban a Rigidbody komponens megszerzése és hozzáfűzése az rb változóhoz. Ezután ugyanezt tessük a LineDrawer komponenssel is, ami egy egyéni komponens lehet a játékban. Mindkét komponensre szükségünk van a játékban a viselkedések és műveletek vezérléséhez. A Rigidbody sebességének nullázása biztosítja, hogy a játékmenet újrakezdésekor az objektum ne mozogjon, ami elkerülheti a nem kívánt viselkedést vagy hibákat. A LineDrawer komponenshez beállítók egy szülő objectet ami majd segíteni fog a falak kitörlésében mikor végére ér egy epizódnak. A RayPerceptionSensorComponent3D komponensek lekérem az input raycast sensorokat. stopCoroutine változóját false-ra, hogy újra elkezdhessem a falgenerálást.

Most, hogy megvan a kezdeti függvény valahogyan vége is kell az Episode-nak. Adni kell valamilyen jutalmat az ügynökünknek.

Létrehozok egy EpisodManager scriptet amiben nyomon követem az eredményeket. Ebben lesz egy Winstate enum ami meghatározza, hogy éppen melyik AI kapta a pontot. Először is declaralom az AI-okat a start metodusban majd minden update-kor vizsgálom hogy melyik ai ütközött mivel és, hogy az enum milyen helyzetben van ez által.

```

1
2
3
4 public enum WinState
5 {
6     SargaPalyaFalnakUtkozott,
7     SargaSajatFalnakUtkozott,
8     SargaEllenfelFalnakUtkozott,
9     PirosPalyaFalnakUtkozott,
10    PirosSajatFalnakUtkozott,
11    PirosEllenfelFalnakUtkozott,
12    None
13
14 }

```

A WinState enumban kell egy None állapot is, hogy új episodenál minden vissza tudjam állítani.

```

1
2 public void jutalmazas(WinState winState)
3 {
4     if (winState == WinState.SargaSajatFalnakUtkozott)
5     {
6         AgentSarga.SetReward(-0.25f);
7
8     }
9     else if (winState == WinState.SargaSajatFalnakUtkozott)
10    {
11        AgentSarga.SetReward(-0.45f);
12    }
13    else if (winState == WinState.
14        SargaEllenfelFalnakUtkozott)
15    {
16        AgentSarga.SetReward(-0.75f);
17        AgentPiros.SetReward(1f);
18    }
19    else if (winState == WinState.PirosSajatFalnakUtkozott)
20    {
21        AgentPiros.SetReward(-0.25f);
22    }

```

```

22     else if (winState == WinState.PirosSajatFalnakUtkozott)
23     {
24         AgentPiros.SetReward(-0.45f);
25     }
26     else if (winState == WinState.
27             PirosEllenfelFalnakUtkozott)
28     {
29         AgentPiros.SetReward(-0.75f);
30         AgentSarga.SetReward(1f);
31     }
32     TronWinTrack(winState);
33     EndEpisodes();
34 }
```

Ha az egyik ai ütközik a pályával akkor -0.25 pontot kap, ha a saját falával akkor -0.45-öt, ha az ellenség falával akkor -0.75-öt míg az ellenség 1 pontot. A pontozást természetesen változtattam többször is míg elértem a látszólagos optimálishez amikor is inkább az volt a céljuk hogy az ellenséget valahogy bekerítsék mint-sem hogy minél tovább túléljenek. Ezeket az adatokat rögzíteni szeretném így írok egy switch case-t minden esetre.

```

1
2
3     private static void TronWinTrack(WinState winState)
4     {
5         switch (winState)
6         {
7             case WinState.SargaPalyaFalnakUtkozott:
8                 Academy.Instance.StatsRecorder.Add("Priros/
9                     FalnakUtkozott", 0, StatAggregationMethod.
10                    Average);
11                Academy.Instance.StatsRecorder.Add("Sarga/
12                    FalnakUtkozott", 0, StatAggregationMethod.
13                    Average);
14                Academy.Instance.StatsRecorder.Add("Sarga/
15                    SargaPalyaFalnakUtkozott", 1,
16                    StatAggregationMethod.Average);
17                break;
18
19
20 }
```

```
14     }
15 }
```

A StatsRecorder az Academy.Instance-ból származik. Ez a Unity ML-Agents keretrendszer része, amely lehetővé teszi a játékstatisztikák rögzítését és nyomon követését a tanítási folyamat során. Az Add függvény segítségével új statisztikai adatokat adok a rekorderhez. Az első paraméter egy azonosító vagy név, amelyet az adatokhoz társítok.

Ha elérünk egy episod végére törölök kell minden a pályáról és vissza állítani az eredeti állapotába.

```
1
2
3
4
5 public void Vege()
6 {
7     winState = WinState.None;
8
9
10
11
12     while (AgentPirosAI.ld.linePointsGameobject != null)
13     {
14         foreach (GameObject item in AgentPirosAI.ld.
15             linePointsGameobject)
16         {
17             Destroy(item);
18         }
19     while (AgentSargaAI.ld.linePointsGameobject != null)
20     {
21         foreach (GameObject item in AgentSargaAI.ld.
22             linePointsGameobject)
23         {
24             Destroy(item);
25         }
26
27
28     AgentPirosAI.ld.linePointsGameobject.Clear();
```

```

29     AgentPirosAI.UtkozesEnum1 = UtkozesEnum.start;
30
31     AgentSargaAI.ld.linePointsGameobject.Clear();
32     AgentSargaAI.UtkozesEnum1 = UtkozesEnum.start;
33
34
35
36
37
38     AgentSarga.EndEpisode();
39     AgentPiros.EndEpisode();
40
41 }

```

Az Vege() metodus biztosítja mind-ezt. a winState változó értékét állítjuk WinState.None-ra, ami azt jelzi, hogy a szimuláció egy session végére ért, vagy kezdődött.

Ezt követően két ciklus fut, amelyek mindegyike törli a linePointsGameobject listában tárolt elemeket mind az AgentPirosAI, mind az AgentSargaAI osztályok esetén. Ez a lista valószínűleg a játék során létrehozott pontok vagy objektumok listája.

Majd minden osztályhoz tartozó linePointsGameobject listát töröljük az AgentPirosAI és az AgentSargaAI osztályoknál, hogy felkészüljünk az új játékállapotra.

Ezután újra indítjuk az ütközési állapotot az AgentPirosAI és az AgentSargaAI osztályoknál.

Végül lezárjuk az aktuális epizódokat mind az AgentSarga, mind az AgentPiros esetén. Ez a játék logikájának része, amely jelzi, hogy az aktuális játékfázis véget ért. Leállítok minden az epizód végén. Maga a reward metódus így néz ki.

```

1
2 public void getReward(float increment)
3
4 {
5
6 AddReward(increment);
7
8 score++;
9
10 scoreText.text = score + " ";
11 }

```

a bejövő mennyiséggel hozzáadom a Reward rendszerhez, és kiiratom a játékban.

Minden új Episodnál különböző pozíiókból indítom az Agents-eket. Ezáltal is tréningel-

ve a változó környezethez.

```
float rx = Random.Range(-13, 13); float rz = Random.Range(-13, 13); float ry = Random.Range(-13, 13);
```

Ezeket az értékeket random generálom. Természetesen ahogyan mozog az ügynök valahogyan egy falat is kell húznia mögötte. Mivel-e nélkül nem tudja csapdába csalni az ellenséges ügynököt.

```
1 private void wallmaker(Vector3 position)
2
3 {
4
5 GameObject wall = Instantiate(Wall, position, Quaternion.
identity);
6
7 wall.transform.forward = this.transform.forward;
8
9 linePointsGameobject.Add(wall);
10
11 }
```

A falakat az Instantiate beépített Unity metódussal generálom. Egy előre meghatározott prefab referenciát vár, ami a fal objectumot adja meg. Meg kell adni a pozíóját és a forgatási irányát. Amint létrehoztam a falat elforgatom az irányát, hogy a szülő objektum felé nézzen. Természetesen ezeket a falakat el kell mentenem, hiszen minden epizód végén törölnöm kell őket. Ezért elmentem őket a linePointsGameobject-be.

```
1
2 void FixedUpdate()
3
4 {
5
6 if (!stopCoroutine) {
7
8 timer -= Time.deltaTime;
9
10 if (besierPontok.Count.Equals(4))
11
12 {
13
14 timer = timerdelay;
15
```

```

16 GenerateBezier-gorbe Curve(besierPontok);
17
18 }
19
20 else
21
22 {
23
24 if (timer $$ 0) \{ besierPontok. Add(this.transform.
    position -
25 this.transform.forward * tavolsagSokszorosito); timer =
    timerdelay; \}
26
27 }

```

Magát a falak pozícióját egy Bézier-görbe görbe által adom meg amit a Unity saját függvényében hívok meg. Ez a FixedUpdate metódus a Unity játékmotorban fut minden egyes fizikai frissítési lépés során. if (!stopCoroutine) { Ellenőrzi, hogy a stopCoroutine változó értéke hamis-e. Ha igen, akkor belép a következő blokkba, különben a metódus kilép. A timer -= Time.deltaTime; Csökkenti a timer változót az eltelt idővel (Time.deltaTime). A timer egy időzítő, amely figyeli az idő műlását. Erre azért van szükség, hogy a falat ne az objectben hozza létre. Az if (besierPontok.Count.Equals(4)) Ellenőrzi, hogy a besierPontok lista elemeinek száma 4-e. Ha igen, akkor belép a következő blokkba, különben a következő else blokkot hajtja végre. A timer = timerdelay; Az időzítőt visszaállítja az eredeti értékére (timerdelay) a Bézier-görbe generálásához. GenerateBezier-görbe Curve(besierPontok); Meghívja a GenerateBezier-görbe Curve metódust a besierPontok listával, amely elkészíti a Bézier-görbét. else { Ez az else blokk a besierPontok. Count nem 4-es esetét kezeli. if (timer <= 0) { besierPontok. Add(this.transform.position - this.transform.forward * tavolsagSokszorosito); timer = timerdelay; } Ellenőrzi, hogy a timer lejárt-e. Ha igen, hozzáad egy új pontot a besierPontok listához, amely a jelenlegi objektum pozíciójából kivon egy távolságot a transform.forward irányba, majd visszaállítja a timer értékét a kiindulási értékére (timerdelay). Ez új pontokat ad a Bézier-görbéhez, amikor a besierPontok lista még nem éri el a 4-es számot.

```

1 private void GenerateBezier-gorbe Curve(List<$Vector3>
    besierPontok)
2
3 {
4
5 if (!stopCoroutine) {

```

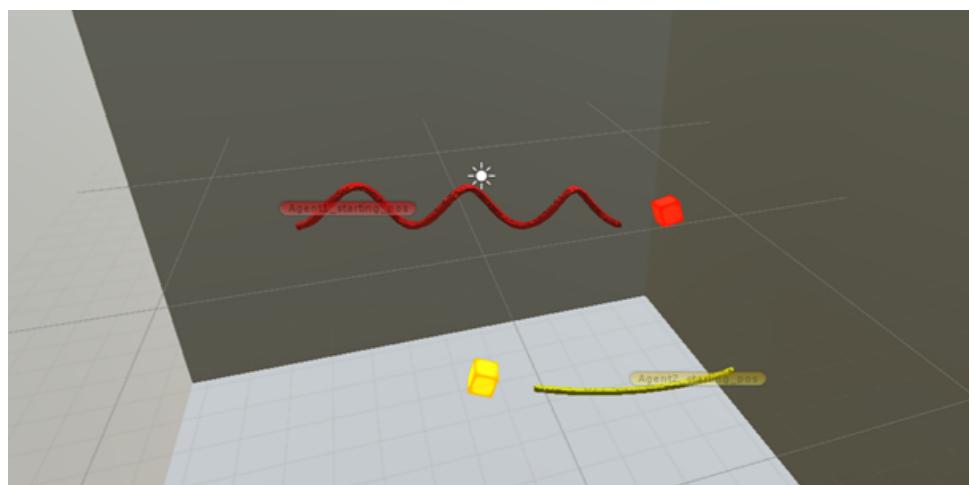
```

6
7 for (float t = 0; t <= 1; t += 0.01f)
8
9 {
10
11 float u = 1 - t;
12
13 float tt = t * t;
14
15 float uu = u * u;
16
17 float uuu = uu * u;
18
19 float ttt = tt * t;
20
21 Vector3 position = uuu * besierPontok[0] + 3 * uu * t *
22 besierPontok[1] + 3 * u * tt * besierPontok[2] + ttt
23
24
25 wallmaker(position);
26
27 }
28
29 p3 = besierPontok[3];
30
31 besierPontok.Clear();
32
33 Debug.Log("clear "+ besierPontok.Count);
34
35 besierPontok. Add(p3);
36
37 }
38
39 }

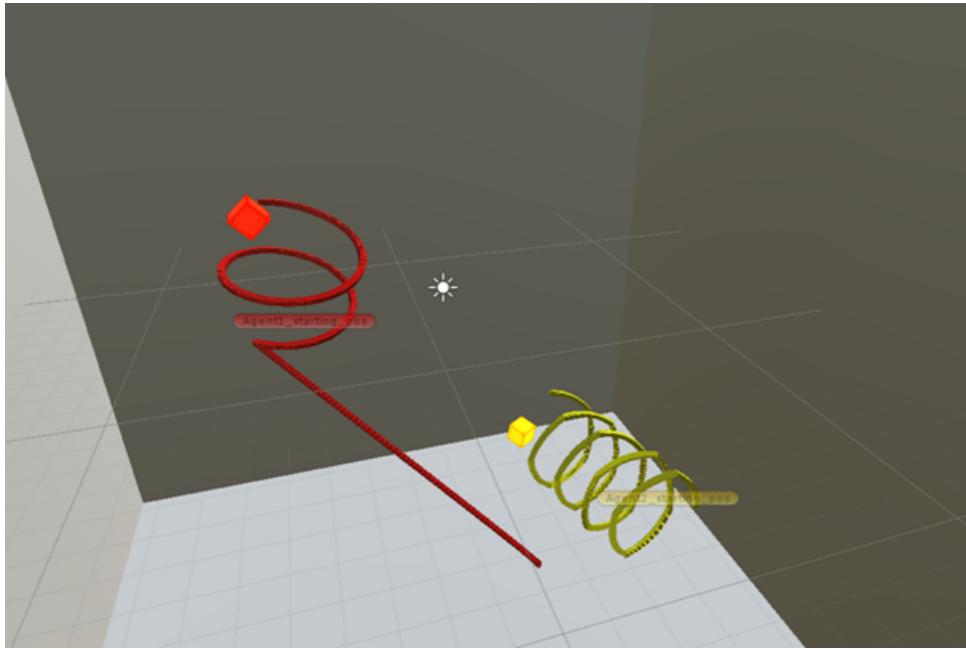
```

private void GenerateBézier-görbe Curve(List < Vector3 > besierPontok) Ez a metódus egy Bézier-görbe pontjait generálja a megadott besierPontok listával. Az if (!stopCoroutine) { Ellenőrzi, hogy a stopCoroutine változó értéke hamis-e. Ha igen, belép a következő blokkba, különben a metódus kilép. A for (float t = 0; t <= 1; t += 0.01f) Egy for ciklus, amely

a Bézier-görbe pontjainak generálását végzi el "t" értékek segítségével 0-tól 1-ig lépkedve 0.01-es inkrementummal. A következő sorokban számos változót számol ki (u, tt, uu, uuu, ttt), amelyek a Bézier-görbe matematikai egyenletében szerepelnek. Vector3 position = uuu * besierPontok[0] + 3 * uu * t * besierPontok[1] + 3 * u * tt * besierPontok[2] + ttt * besierPontok[3]; Ebben a sorban kiszámolja a Bézier-görbe pontjainak aktuális pozícióját a paraméterként kapott besierPontok listából és a t értékekből. Ezután a kapott pozíciót a position változóban tárolja.wallmaker(position); Hívja a wallmaker metódust a kiszámolt pozícióval, amely létrehoz egy falat a megadott pozícióban. A for ciklus végén a p3 változót beállítja a besierPontok negyediek elemének értékével (p3 = besierPontok[3]);.besierPontok.Clear(); Törli a besierPontok listát, hogy felkészüljön az új pontok hozzáadására a következő iterációban.Debug.Log("clear " + besierPontok.Count); Kiírja a konzolra a törölt besierPontok lista elemek számát.besierPontok. Add(p3); Hozzáadja a p3 értékét a besierPontok listához, így az új pontok hozzáadásra kerülnek a következő iterációban.



5.8. ábra. Generációk



5.9. ábra. Generációk

Most, hogy az alapok megvannak, még csak egy dolgot kell beállítani. Jelenleg 1 ügynököt tréningelünk a világ ellen de én azt szeretném hogy önmaga ellen játsszon Pontosabban egy régebbi másolatával, így ösztönözve önmagát a folyamatos fejlődésre. Ehhez a Self_play tud megoldást. Itt az igazából csak az egyik ügynököt tréningelel, a másik az egy statikus másolata a neural networknek míg a másik folymatosan fejlődik. Így meg kell változtatnom az alapértelmezett konfigurációt az ML-Agents-hez illetve a behavior paramteres-nél egy két dolgot. Létrehozok egy .yaml config file-t.t.5.10

Ha már itt vagyok megváltoztatok pár értéket, hogy felgyorsítsam a tanulási folyamatot a tanulási folyamatot. A batchsize-t ami tanítási batch-ek méretét jelenti.2048-ra állítom. A buffersize-t ami a memóriában tárolt adatok maximális mérete legyen 20480. A maxsteps-t ami a tanítási folyamat maximális lépésszáma 80000000, nem akarom végtelenre de minél nagyobbra, hogy sok ideig tartson. A summaryfreq-t ami a TensorBoard összefoglalók gyakorisága.

És most a legfontosabb része a self play paramétereit is beleírom, hogy értelmezni tudja az ML-agents a 2 ügynökök. minden save_steps=20000 lépés pillanatképet készít a tanulóügynök meglévő szabályzatáról. Legfeljebb window=10 pillanatkép kerül tárolásra. Új pillanatfelvétel készítésekor a legrégebbi felvételt eldobja. Önmaguknak ezek a múltbeli változatai válnak az „ellenfelekké”, akik ellen a tanuló ügynök edz. minden swap_steps=10000 lépésnél az ellenfél szabályzata egy másik pillanatképre cserélődik. A pillanatkép mintavételezése play_against_latest_model_ratio=0,5 valószínűsséggel történik, hogy a legújabb szabályzattal (azaz a legerősebb ellenféllel) szemben fog játszani. Ez segít elkerülni, hogy egyetlen ellenfél játékstílusához túlilleszkedjenek. Természetesen ez akkor lenne releváns ha több ügynök is lenne ugyan abban a csapatban.

Team_change=100000 lépés után a tanuló ügynök és az ellenfél felcserélődik. Az aktuális én és az ellenfél elleni játék valószínűsége a készletből a play_against_latest_model_ratio értékkel. A play_against_latest_model_ratio nagyobb értéke azt jelzi, hogy egy ügynök gyakrabban fog játszani az aktuális ellenféllel. Az edzéslépések száma az új ellenfél save_steps paraméterekkel történő mentése előtt. A save_steps nagyobb értéke olyan ellenfeleket eredményez, amelyek a képzettségi szintek és esetleg a játékstílusok szélesebb skáláját fedik le, mivel a szabályzat több képzésben részesül.

```
C:\> ProgInf\unity\Thesis> ! Tron.yaml
1   default_settings: null
2   behaviors:
3     Tron:
4       trainer_type: ppo
5       hyperparameters:
6         batch_size: 2048
7         buffer_size: 20480
8         learning_rate: 0.0002
9         beta: 0.003
10        epsilon: 0.15
11        lambd: 0.93
12        num_epoch: 4
13        learning_rate_schedule: constant
14        network_settings:
15          normalize: true
16          hidden_units: 256
17          num_layers: 2
18          vis_encode_type: simple
19        reward_signals:
20          extrinsic:
21            gamma: 0.96
22            strength: 1.0
23        keep_checkpoints: 5
24        max_steps: 80000000
25        time_horizon: 1000
26        summary_freq: 20000
27        self_play:
28          window: 10
29          play_against_latest_model_ratio: 0.5
30          save_steps: 20000
31          swap_steps: 10000
32          team_change: 100000
33
```

5.10. ábra. Pytorch config

Ezek után a behavior paramteres-nél különböző Teameket választok ki a 2 agnets-nek.



5.11. ábra. Team Id

Visszamegyünk a Command promptba és elindítom az ML-Agentset az 'mlagents-learn -run-id=TeamTest1 Tron.yaml -force -time-scale=10' parancssal amivel id-t is adok neki hogy egyediek legyenek a futások. Illetve az előre definiált konfigurációt szeretném használni amit be is írok ide + jó lenne ha kicsit gyorsabban történne az idő játékon belül, ezzel is felgyorsítva a játékot ezért a time-scale-t átállítom 10-re.

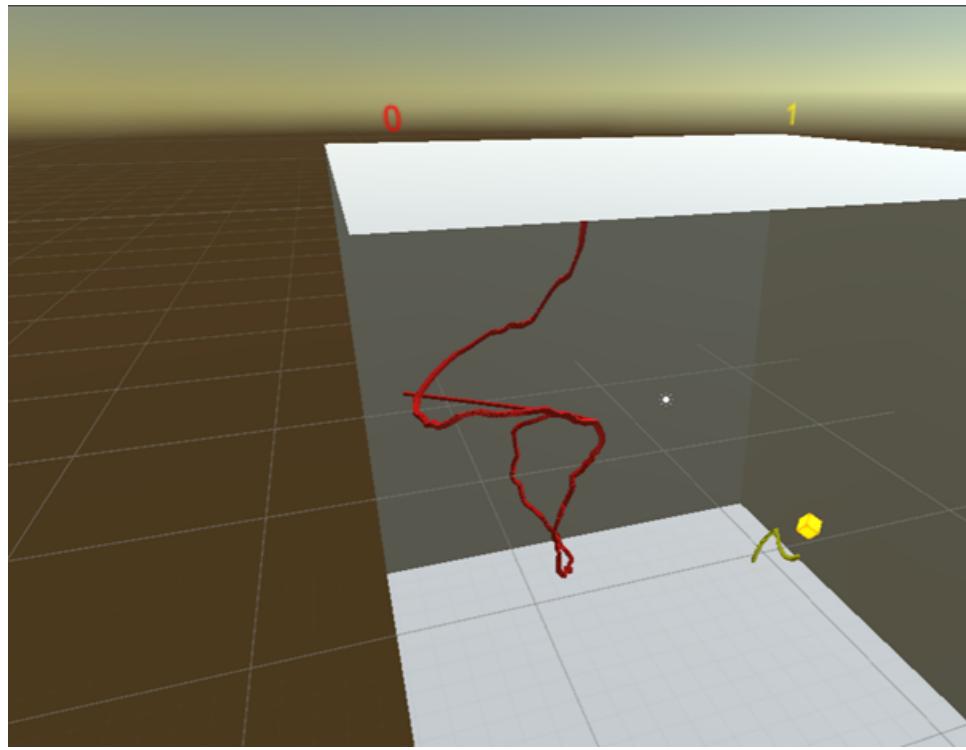
Ha minden jól ment akkor ezt a felületet kell, hogy lássuk és a Unityben a play gomb megnyomásával elkezdhetjük a tréninget.



5.12. ábra. Team Id

látszik, hogy már az egyik ügynök kapott is egy pontot.5.13

Ha újra megnyomjuk a play gombot vagy megvárjuk mind az összes lépést (ami alapértelemezetten 50000) akkor véget ér a tréning.



5.13. ábra. Tréning

Az eredményt egy onnx-ben tárolja.

Ezt a file-t be lehet illeszteni a behavior parametersbe, ami szabályozza majd a actions modelt. Ezáltal minden tesztelésnél külön AI-t hozunk létre, amit majd felhasználhatunk különböző nehézségi szintként a játékunknál.

Organise						New	Open	Select
						Date modified	Type	Size
Name								
My Behavior						14/01/2024 21:03	File folder	
run_logs						14/01/2024 21:03	File folder	
configuration.yaml						14/01/2024 21:03	Yaml Source File	2 KB
My Behavior.onnx						14/01/2024 21:03	ONNX File	103 KB

5.14. ábra. onnx

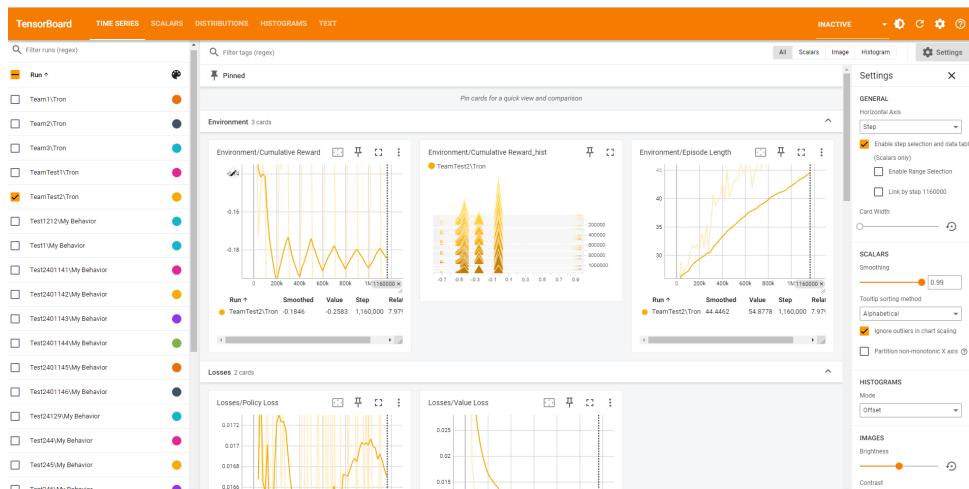
5.6. Eredmények

Szeretném vizualizálni a tanulási folyamatot. Szerencsére a pytorchal van erre lehetőség, méghozzá a TensorBoard ami egy vizualizációs eszköz, amelyet elsősorban a mély tanulási

modellek fejlesztéséhez és monitorozásához használnak. A TensorBoardot a Google fejlesztette a TensorFlow mély tanulási keretrendszerhez, de más mély tanulási keretrendszerrel is kompatibilis, például a PyTorch.

A TensorBoard5.15 segítségével könnyedén megjeleníthetjük a tanítási folyamat során rögzített metrikákat, például a veszteséget és a pontosságot, valamint azok változását az időfüggvényében. Emellett lehetőségünk van megjeleníteni a modell architektúráját, például a rétegek és azok kapcsolatainak diagramját. Más hasznos funkciók közé tartozik a gráfok megjelenítése, az adatok eloszlásának vizsgálata, a különböző rétegek aktivációinak vizualizálása és a hiper paraméterek kísérletezése.

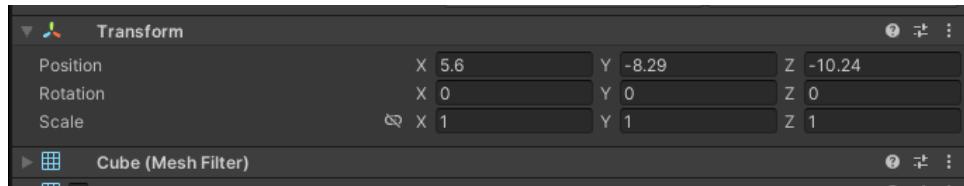
A TensorBoard webes felülete intuitív és könnyen kezelhető, így lehetővé teszi a fejlesztők számára, hogy hatékonyan vizsgálják és elemezzék a mély tanulási modelleket, valamint nyomon kövessék a tanítási folyamatot. Ezáltal segíti a modellek fejlesztését és finomhangolását, valamint segítséget nyújt a hibakeresésben és a modellek teljesítményének optimálizálásában.



5.15. ábra. Tensoboard

5.7. Hibakeresés

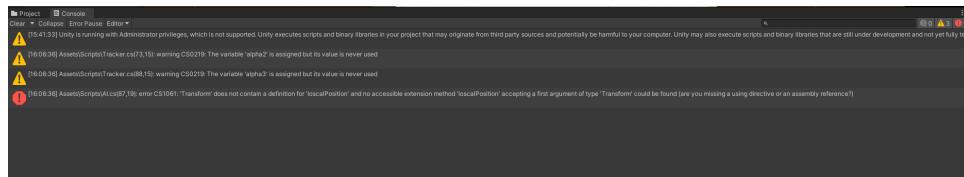
Szerencsére a Unity keretrendszer több hibakezelési lehetőséget is biztosít. Az egyik ilyen a legkézenfekvőbb maga az Inspector5.16. Az Inspector általa jelenetben lévő objektumok kiválasztásával meg tudom tekinteni az azokon lévő sscriptek tulajdonságait ami lehetővé teszi az ott található változók értékeinek futás előtt vagy közbeni módosítását és ellenőrzését. Természetesen itt megjelennek azok a változók amik publikusan lettek deklarálva a scriptben. Így figyelmen tudjuk követni akár futás közben is a változásokat és könnyebb kiszűrni, ha esetleg valamelyik változó nem nekünk megfelelő értéket vess fel. Ez A jelenlegi projektben sem működik másképpen, és sok esetben van a segítségemre.



5.16. ábra. Inspector

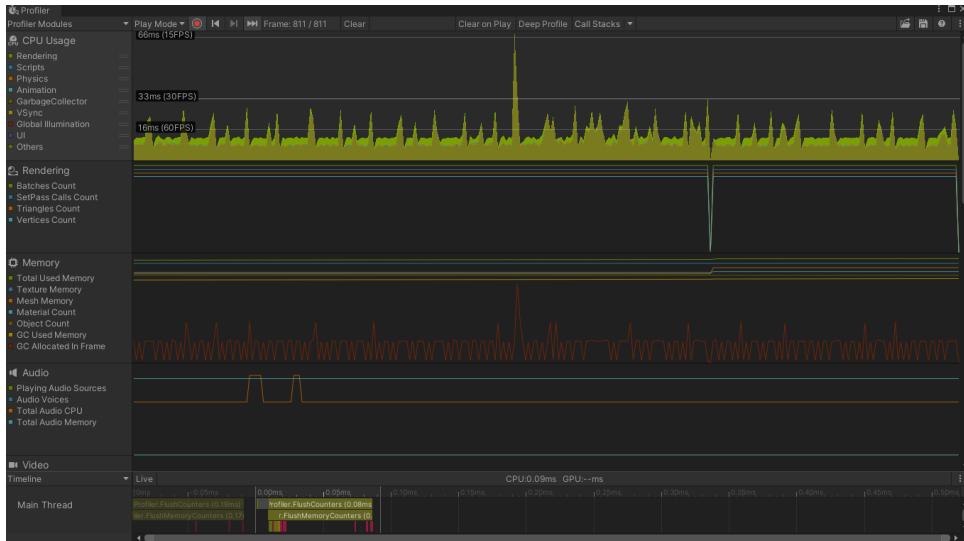
A másik hasznos dolg amit használtam a tesztelésre a konzol5.17 a Unityban elengedhetetlen eszköz a minden nap munkában. Ahogy dolgoztam a projekten, gyakran előfordul, hogy szembesülök különböző hibaüzenetekkel vagy figyelmeztetésekkel. Ebben az esetben a Konzol segítségével könnyen és gyorsan azonosíthatom ezeket a problémákat, és elvégezem a szükséges javításokat. Ebbe bele tartoznak mindenek, amiket én írok Debug.Log() parancssal és azok is amik a motor hibaüzenetei. Itt be tudom kapcsolni az Error Pause lehetőséget, ami arra szolgál, hogy ha hiba történik futás közben akkor megáll a kód, így könnyebben tudom azonosítani a hiba okát. Természetesen az is előfordul, hogy a kód rosszul van megírva, így fordítási időben is kaphatunk hibát. Ilyenkor nem is tudjuk elindítani a játékot. Amikor a konzolban lévő hibára kétszer rákattintunk megnyitja a scriptet és a megfelelő sorra navigál.

Az is rendkívül hasznos, hogy a Konzolon keresztül láthatom a motor és a felhasználi kód által generált hibakeresési naplókat. Ezáltal könnyebben követhetővé válnak a kód működésével kapcsolatos részletek, és gyorsabban reagálhatok az esetleges problémákra vagy hibákra.



5.17. ábra. console

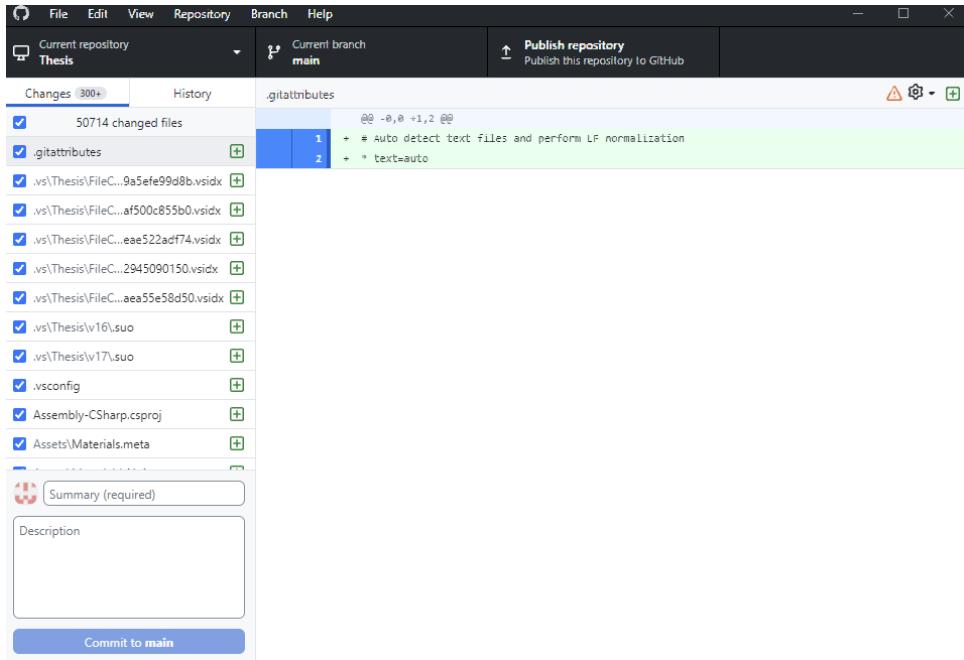
Futás közben szerettem volna gyorsítani a teszteléseken, de nem tudtam, hogy a program meglyik része nyúl hozzá leginkább az erőforrásaimhoz. Szerencsére a unity ebben is nyújt segítséget a Profiler5.18 formájában. A Profiler egy teljesítményelemző eszköz ami szépen mutatja, hogy mi a leginkább teljesítmény igényes. Itt láthatjuk, hogy az általam használt raycasting mennyi energiát fogyaszt, lehetne e kevesebb vagy több, illetve, hogy a falak, amiket futás közben hoz létre a program mennyire terhelik és lassítják le, vagy akár az egyes scriptek mennyire lassítják le a dolgokat, és még azt is, hogy mi mennyi memóriát használ.



5.18. ábra. Profiler

5.8. Verzió követés

A verzió követésre GitHubot használok6.2. De mi is az a GitHub? A GitHub egy olyan webes platform és git alapú verziókezelő rendszer, amelynek segítségével én, mint fejlesztő, könnyen kezelhetem a verziókezelési feladataimat. Saját projektjeim teszteléséhez és hibakezeléséhez is remekül használható. A GitHub lehetővé teszi számomra, hogy megoszthassam a kódomat más fejlesztőkkel, követhetem az előrehaladást, kezeljem a hibákat és fejlesztési javaslatokat, valamint együttműködjem a csapattársaimmal. Az Issue Tracker funkciója segítségével könnyen nyomon követhetem a projekt feladatait és hibáit, míg a Pull Requestek segítenek a változtatások összefoglalásában és a kódellenőrzésben. Így létre is hozok egy új repositoryt a projektemnek és ide töltöm fel minden változásomat. Illetve feltelepítem a Github Desktopot-ot amivel így könnyebben elértem a projektemet és kezelní tudom a változásokat.



5.19. ábra. Github Desktop

5.9. Tesztelés

A HumbleObject tervezési mintát követve fogjuk átalakítani az AI scriptünket. Azért a Humble Object Pattern mert ez egy oylan tervezési minta a szoftvertesztelésben, amelynek célja, hogy külön vállassza a bonyolult logikát az alacsony szintű platformspecifikus műveletektől, amelyeket nehéz tesztelni. ezáltal a bonyolult logika elválik az alacsony szintű műveletektől, és az alacsony szintű műveleteket kiszolgáló osztályokat általában könnyebb tesztelni. Lehetőlegük hogy az AI-ok tényleg megfelelő irányban mozognak-e. Először is létrehozok az Assets mappánkban egy Test mappát, azon belül egyEditMode és egy PlayMode mappát. Majd a PlayMode-ba létrehozok egy alternatív Control-scriptet ControlTest néven. Először is létrehozok egy IInputProvider interfész, amit egy UnityInputProvider osztályban implementálok. Majd ezután átalakítjuk a ControlTest scriptet.

A Move metódusnál már nem kell bemeneti paraméterek a forgatáshoz, mivel az inputokat az IInputProvider interfész segítségével kapja meg. Ehelyett a Move metódus az inputProvider változó segítségével kérdezi le az inputokat az IInputProvider implementációjától, például az UnityInputProvider-tól.

```

public void Move(float rotateHorizontal, float rotateVertical, float rotateZAxis)
{
    // Apply movement to the Rigidbody velocity
    rb.velocity = this.transform.forward * moveSpeed;

    // Get input for rotation

    // Calculate rotation angles
    yaw = rotateHorizontal * rotationSpeed;
    pitch = -rotateVertical * rotationSpeed;
    roll = -rotateZAxis * rotationSpeed;
    // Create a rotation based on the calculated angles
    Quaternion deltaRotation = Quaternion.Euler(new Vector3(pitch, yaw, roll) * Time.deltaTime);

    // Apply rotation to the Rigidbody rotation
    rb.MoveRotation(rb.rotation * deltaRotation);
}

```

5.20. ábra. move

Ez a módszer növeli a kód tisztaságát és tesztelhetőségét, mivel az inputProvider könnyen cserélhető más implementációkkal. Írok most egy Setup metódust, mivel a Setup metódus inicializálja a Rigidbody-t és az input kezelőt az IInputProvider interfész használatával, amely így különbözik a Move metódustól, és megkönnyíti a kód karbantarthatóságát és tesztelhetőségét.

A [SetUp] attribútum jelzi, hogy ez a metódus minden osztályon belül megírt teszt előtt le fog futni.

```

private void Setup()
{
    // Rigidbody inicializálása
    rb = GetComponent<Rigidbody>();

    // InputProvider inicializálása (pl. UnityInputProvider használata)
    inputProvider = GetComponent<IInputProvider>();
}

```

5.21. ábra. Setup

a TearDown5.22 metódusban a rb és az inputProvider változók értékét null-ra állítjuk, így felszabadítjuk az erőforrásokat és visszaállítjuk az állapotot. Természetesen a TearDown metódus tartalma attól függően változhat, hogy milyen inicializációs vagy felszabadítási feladatokat szeretnénk végrehajtani.

Ezek után megírjuk a tesztjeinket nézzük meg hogy move metodus5.23 meghívása után el kezd e egyáltalán mozogni az objekt.

```

private void TearDown()
{
    // Felszabadítjuk az erőforrásokat vagy visszaállítjuk az állapotot
    rb = null;
    inputProvider = null;
}

```

5.22. ábra. Teardown

```

[UnityTest]
public IEnumerator Control_MoveTest()
{
    // Arrange
    GameObject aiObject = new GameObject();
    AI aiControl = aiObject.AddComponent<AI>();
    Vector3 initialPosition = aiObject.transform.position;

    // Act
    aiControl.Move();

    // Assert
    // Ellenőrizzük, hogy a Move metódus után az objektum ténylegesen mozogott-e
    Assert.AreNotEqual(initialPosition, aiObject.transform.position, "Az AI objektum nem mozogott");

    // Várunk egy kis időt, hogy a metódus végrehajtódjon
    yield return null;

    // Objektumok felszabadítása
    GameObject.Destroy(aiObject);
}

```

5.23. ábra. Move Test

Az automatikus buildelési és tesztelési folyamathoz létrehoztam egy github nevű map-pát a Thesis Unity projekt gyökérkönyvtárában. Ezután ebben a mappában további egy workflows nevű almappát hoztam létre. Ebbe a mappában másoltam be a naprakész GameCI dokumentációból az aktiváláshoz szükséges YAML 5.24 kódot, majd elmentettem activation.yml néven. Ezt követően commitoltam a változásokat, majd pusholtam azokat az origin-re. A GitHub felületén a repository kiválasztását követően a Settings/Secrets menüpontban létrehoztam egy új secretet a New repository secret gomb segítségével. A secret neve UNITY - LICENSE lett, és értékként az .ulf kiterjesztésű licenc fájl tartalmát adtam meg.

Ezután létrehoztam egy olyan workflow-t, amely automatikusan végrehajtja a szükséges buildelési és tesztelési folyamatokat. Ehhez a GameCI dokumentáció Test Runner és Builder fejezeteiben található kész példakódokat használtam. Ahhoz, hogy a test runner megfelelően fusson, engedélyeztem a read-write jogosultságot a workflow-k számára a Settings / Code and Automation / Actions / General menüpont alatt a Workflow Permissions bekezdés alatti

Read and Write permissions opció kiválasztásával. Végül összefésültettem a két kész példakódot (buildeléshez és teszteléshez), majd elkészítettem a main.yml fájlt a .github/workflows mappában.

```
on: [push, pull_request]

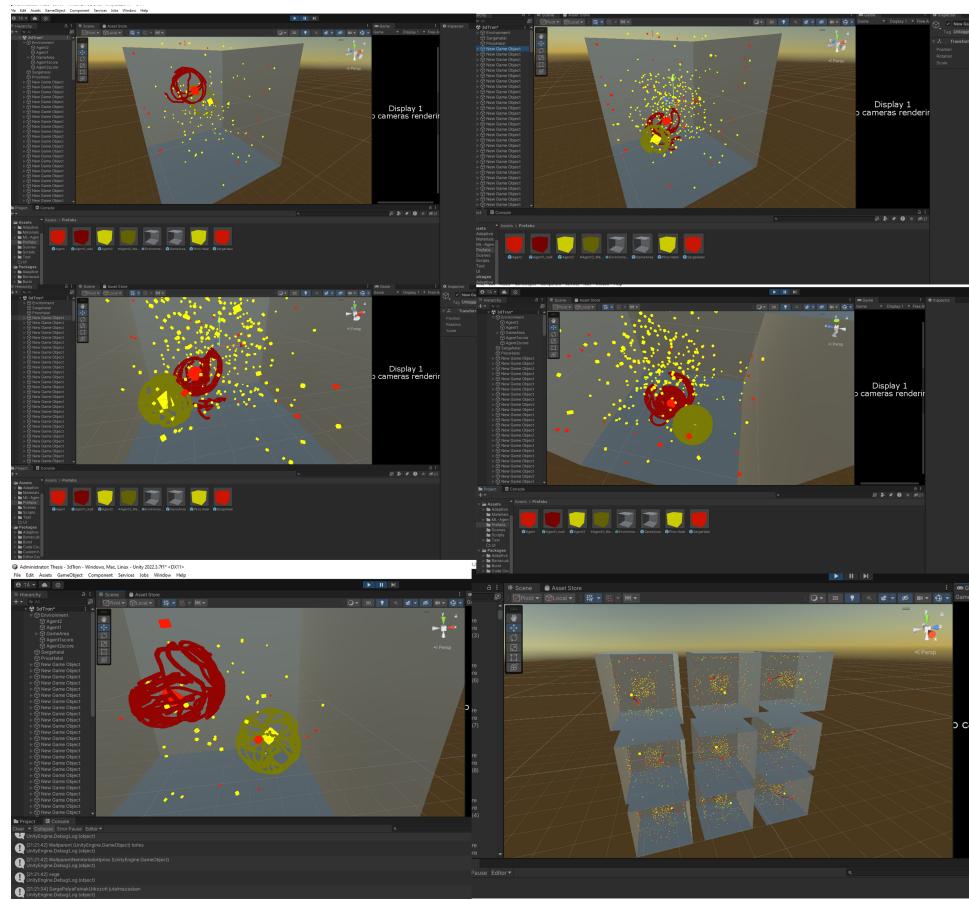
jobs:
  testAllModes:
    name: Test in ${ matrix.testMode }
    runs-on: ubuntu-latest
    strategy:
      fail-fast: false
    matrix:
      projectPath:
        - test-package
      unityVersion: '2020.3.0f1' # some version must be included for package testing
      testMode:
        - playmode
        - editmode
    steps:
      - uses: actions/checkout@v4
        with:
          lfs: true
      - uses: game-ci/unity-test-runner@v4
        id: tests
        env:
          UNITY_LICENSE: ${{ secrets.UNITY_LICENSE }}
          UNITY_EMAIL: ${{ secrets.UNITY_EMAIL }}
          UNITY_PASSWORD: ${{ secrets.UNITY_PASSWORD }}
```

5.24. ábra. Autamtitálás

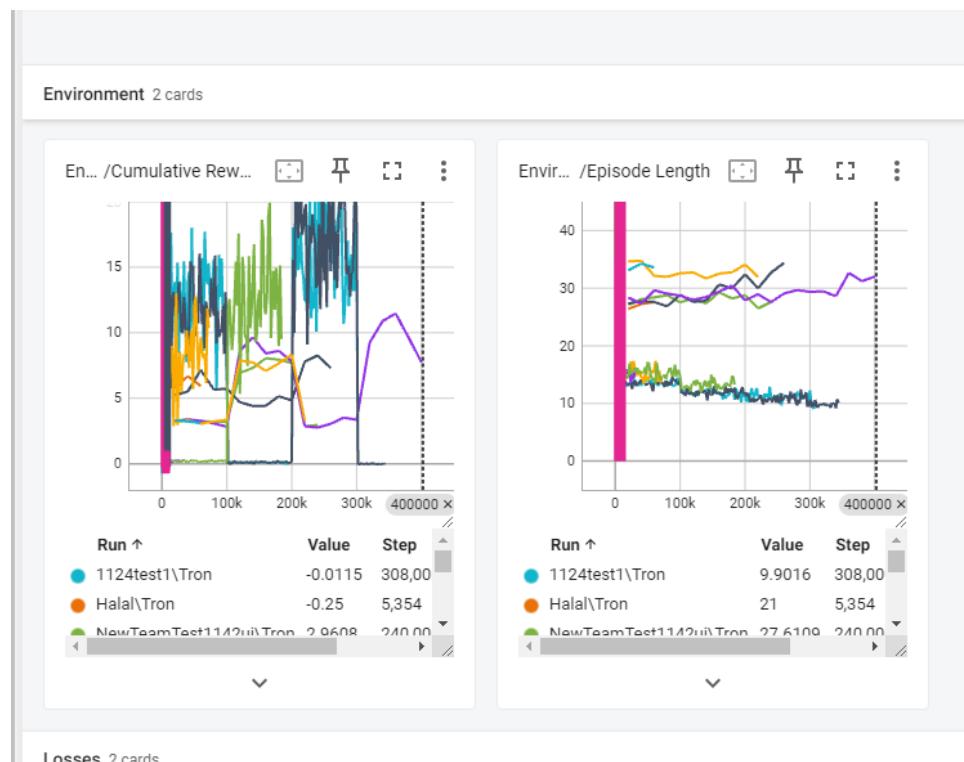
6. fejezet

Összegzés

A futások során sajnos beleakadtam egy két problémába. Ugyan az ügynökök látták az el-lenséges falakat magát az ügynököt nem, így sokszor figyelmen kívül hagyták mint cél. célt. Illetve a raycast csak 1 síkban előre látott, úgy éreztem, hogy ha hozzá adok még 2 síkot finomabb mozgást tudnak eszközölni, így hozzáadtam +2 raycast síkot az ügynökökhöz, hogy szélesebb körben lássonak, illetve most már látják a másik ügynök helyét is. A neurál netwörknek változtatnom kellett a layeryét, mivel változott a bemeneti szám a plusz raycast miatt. Így növelteim 6 layerre még lecsökkentettem a hidden layeryét 6-ra, ezzel tesztelve, hogy gyorsabban tanul-e. Most már látják a másik agentet is. Megváltoztattam a reward rendszert is, mivel a súlyozásnál, nem volt elég motiváló ha csak 1f jutalmat kapnak ha a másik az ellenség falának ütközik. Hozzá adtam a rewardhoz +10-t akkor is ha az ellenfél meghal, illetve megnövelteim azt a mennyiséget amit eredetileg kaptak +-ban ha a másik neki ütközik az ellenséges felnak. De még mindig az elején vagyunk a tanításnak, és fel szeretném gyorsítani a tanulásukat, ezért időbeli jutalmakat is kapnak. Tanuljanak meg először túlélni, aztán majd taktikázni, hogy a másikat bekerítsék, ezért időbeli jutalmazásokat is kapnak. A végeredmény, hogy megpróbálják kijátszani a mozágis mozgási beállításaimat, és minél gyorsabban irányt változtatni, így a folyamatos előre haladás átvált egyhelybeni mozgásra. Változtatnom kellett ezért a mozgási értékeken, illte illetve vizualizáltam a halálokat, hogy biztos legyek benne működnek-e rendesen a collider rendszerek. Majd ana azokat is megváltoztattam trigger colliderekre, hogy ne ne akadjanak bele egymásba, ezzel is hibát generálva. A futások során próbáltam ugyan arra fokusálni, hogy minél gyorsabban számítsák ki az adatokat, és használunk fel semmi féle felesleges erőforrást. Ezért ahogy tudtam optimalizáltam a programot a profiler adatait elemezve. Rájöttem, hogy a renderelés nagyon nagy erőforrásigénnel rendelkezik, így a materiálokat átalakítottam egyszerű színekre, illetve a fényforrásokat is levettem. A kockáknak és a falaknak megváltoztattam a collider-jét mivel nagy számításigényűek, és adtam nekik egy kör collidert, ami sokkal könnyebben ki lehet számolni. Pluszban játék kamerára sem volt szükségem a tesztek futtatásához.



6.1. ábra. Tanulások



6.2. ábra. Jutalmazás grafikon

Irodalomjegyzék

- [1] Deep RL Course. Hugging Face. <https://huggingface.co/learn/deep-rl-course/unit0/introduction>
- [2] Practical Deep Learning. Fast.ai. <https://course.fast.ai>
- [3] Practical Deep Learning for Coders. Fast.ai. <https://course.fast.ai/Lessons/lesson1.html>
- [4] Convolutional neural network. Wikipedia. https://en.wikipedia.org/wiki/Convolutional_neural_network
- [5] How to use Unity ML Agents in 2023! ML Agents 2.0.1. YouTube. <https://www.youtube.com/watch?v=RANRz9oyzko>
- [6] Create a Basic Neural Network Model - Deep Learning with PyTorch 5. YouTube. <https://www.youtube.com/watch?v=RANRz9oyzko>
- [7] Learning Agents Introduction in Unreal Engine. Epic Games. <https://dev.epicgames.com/community/learning/tutorials/8OWY/unreal-engine-learning-agents-introduction>
- [8] Unreal Engine 5.3's New Plug-in Lets You Train NPCs via Machine Learning. 80.lv. <https://80.lv/articles/unreal-engine-5-3-s-new-plug-in-lets-you-train-npcs-via-machine-lea>
- [9] Unity. <https://unity.com>
- [10] Unity Machine Learning Agents. <https://unity.com/products/machine-learning-agents>
- [11] Watching Neural Networks Learn. YouTube. https://www.youtube.com/watch?v=TkwXa7Cvfr8&list=PLchH1h_00r_lfr-aYDvBVPJ8T_aXFG60h&index=22&t=247s&themeRefresh=1
- [12] What is Deep Learning? Amazon Web Services. <https://aws.amazon.com/what-is/deep-learning/>

- [13] Mi is az a neurális hálózat? | 1. fejezet, Gépi tanulás. YouTube. https://www.youtube.com/watch?v=aircAruvnKk&list=PLchH1h_O0r_1fr-aYDvBVPJ8T_aXFG60h&index=16
- [14] Updated Disney Tron Arcade Video Game Free! Light Cycles and more! YouTube. https://www.youtube.com/watch?v=XEp8G2HtDJM&list=PLchH1h_O0r_1fr-aYDvBVPJ8T_aXFG60h&index=10
- [15] Using a neural network as AI in my game in Python - a warm up. YouTube. https://www.youtube.com/watch?v=7GQEV7disi4&list=PLchH1h_O0r_1fr-aYDvBVPJ8T_aXFG60h&index=11
- [16] How to use Machine Learning AI in Unity! (ML-Agents). YouTube. https://www.youtube.com/watch?v=zPFU30tbyKs&list=PLzDRvYVwl53vehwiN_odYJkPBzcqFw110
- [17] AI Learns to play Flappy Bird! YouTube. <https://www.youtube.com/watch?v=fz8D0OZkQGQ&list=PL22-qG2MGHhAUR4cYXQBmLSCisi-OMUpB>
- [18] Hibakeresés és tesztelés Unity keretrendszerben. <https://xdepot.uni-eszterhazy.hu/index.php/s/i66uTfLh5XjkSqc?path=%2FJÃ¡tÃ©l'kfejlesztÃ©l's#pdfviewer>
- [19] Unity ML-Agents 1.0+ - Self Play explained. YouTube. <https://www.youtube.com/watch?v=zAtcRbYdvuw>
- [20] Training intelligent adversaries using self-play with ML-Agents. Unity Blog. <https://blog.unity.com/engine-platform/training-intelligent-adversaries-using-self-play-with-ml-agents>
- [21] ML-Agents Toolkit Overview. GitHub. <https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/ML-Agents-Overview.md>
- [22] My AI played millions of games against itself | Self Play (Unity Devlog 05). YouTube. https://www.youtube.com/watch?v=gMe85hVwC1M&ab_channel=NeuralBreakdownwithAVB
- [23] Unity-Technologies. GitHub. <https://github.com/Unity-Technologies/ml-agents/blob/develop/project/Assets/ML-Agents/Examples/SharedAssets/Prefabs/Logo-PlaneMesh-GRAY.prefab>
- [24] Imitation Learning. GitHub. <https://github.com/gzrjzcx/ML-agents/blob/master/docs/Training-Imitation-Learning.md>

- [25] Training with Imitation Learning. GitHub. <https://github.com/Unity-Technologies/ml-agents/blob/0.15.0/docs/Training-Imitation-Learning.md>
- [26] Ray Perception Sensor Component Tutorial. Immersive Limit. <https://www.immersivelimit.com/tutorials/rayperceptionsensorcomponent-tutorial>
- [27] 155 - How many hidden layers and neurons do you need in your artificial neural network? YouTube. https://www.youtube.com/watch?v=bqBRET7tbiQ&ab_channel=DigitalSreeni
- [28] Deep RL Course. Hugging Face. <https://huggingface.co/learn/deep-rl-course/unit0/introduction>
- [29] 156 - How to limit GPU memory usage for TensorFlow? YouTube. https://www.youtube.com/watch?v=cTrAlg0OWUo&ab_channel=DigitalSreeni
- [30] Deep RL Course. Hugging Face. <https://huggingface.co/learn/deep-rl-course/unit0/introduction>

NYILATKOZAT

Alulírott Tápai Árpád büntetőjogi felelősségem tudatában kijelentem, hogy az általam benyújtott, Játékfejlesztés Unity keretrendszerben című szakdolgozat (diplomamunka) önálló szellemi termékem. Amennyiben mások munkáját felhasználtam, azokra megfelelően hivatkozom, beleértve a nyomtatott és az internetes forrásokat is.

Tudomásul veszem, hogy a szakdolgozat elektronikus példánya a védés után az Eszterházy Károly Katolikus Egyetem könyvtárába kerül elhelyezésre, ahol a könyvtár olvasói hozzájuthatnak.

Kelt: Szeptember, 2024. év 04.... hó 03... nap.

Tápai Árpád

aláírás