



ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

MESTERSÉGES INTELLIGENCIÁVAL TÁMOGATOTT LÉFIGORGALOM IRÁNYÍTÁS

Szerző(k):

Gombár Egon

mérnök-informatikus BSc. szak, III. évf.

Konzulens(ek):

Dr. Kertész Gábor

egyetemi docens

Budapest, 2023.

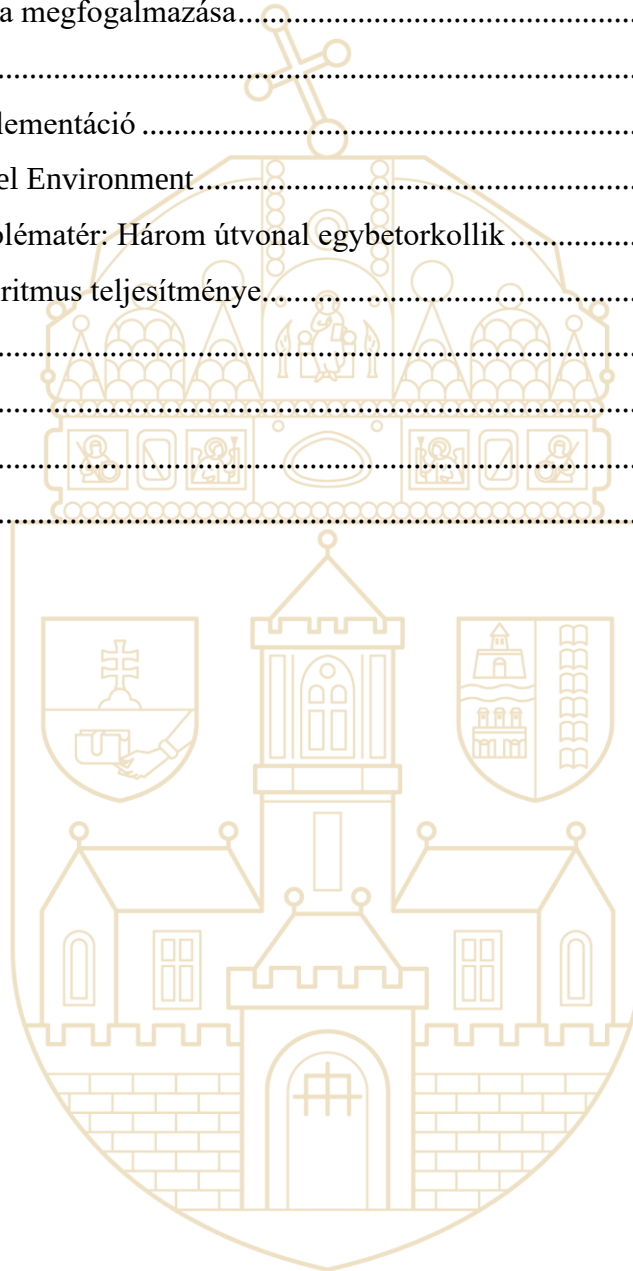


Tartalom

1. Bevezetés	4
2. A feladat.....	5
3. A kutatás célja.....	5
4. Mesterséges intelligencia (MI)	6
4.1. A mesterséges intelligencia ágai	9
4.2. Logikai MI.....	9
4.3. A mesterséges intelligencia alkalmazási területei.....	11
4.4. Mesterséges intelligencia felhasználása repülőtereken/repülésben.....	13
5. A légiforgalom irányítás legfontosabb elemei, működési elve.....	14
6. Autonóm légiforgalom irányítás rendszerek megvalósítására irányuló legújabb módszerek, modellek.....	15
6.1. Deep MARL	15
6.2. Deep Ensemble MARL	18
6.3. Embedding ML in ATC	20
7. Légiforgalmi szimulátorok.....	23
7.1. Cassiopeia.....	23
7.2. Elsa.....	24
7.3. BlueSky – The Open Air Traffic Simulator	25
8. Tervezés	29
8.1. Légiforgalom szimulátor.....	29
8.2. MT keretrendszer	29
8.3. Open AI Gym.....	29
8.4. Programnyelv	29
9. Egy-ágensű architektúra.....	30
10. Egy-ágensű probléma megfogalmazása.....	31
10.1. Célkitűzés	31
10.2. Egy-ágensű implementáció	31
10.3. Szimulációs beállítások	31
10.4. A BlueSky kiterjesztése saját plug-in-nal	32



10.5. Open AI Gym környezet létrehozása	32
10.6. Egy-ágensű problémátér: Három útvonal egy gócponttal.....	36
10.7. Egy-ágensű algoritmus teljesítménye.....	37
11. Több-ágensű architektúra.....	38
12. Több-ágensű probléma megfogalmazása.....	39
12.1. Célkitűzés	39
12.2. Több-ágensű implementáció	39
12.3. PettingZoo Parallel Environment	39
12.4. Több-ágensű problémátér: Három útvonal egybetorkollik	42
12.5. Több-ágensű algoritmus teljesítménye.....	43
13. Összegzés	44
14. Summary	45
15. Irodalomjegyzék	46
16. Ábrajegyzék	48



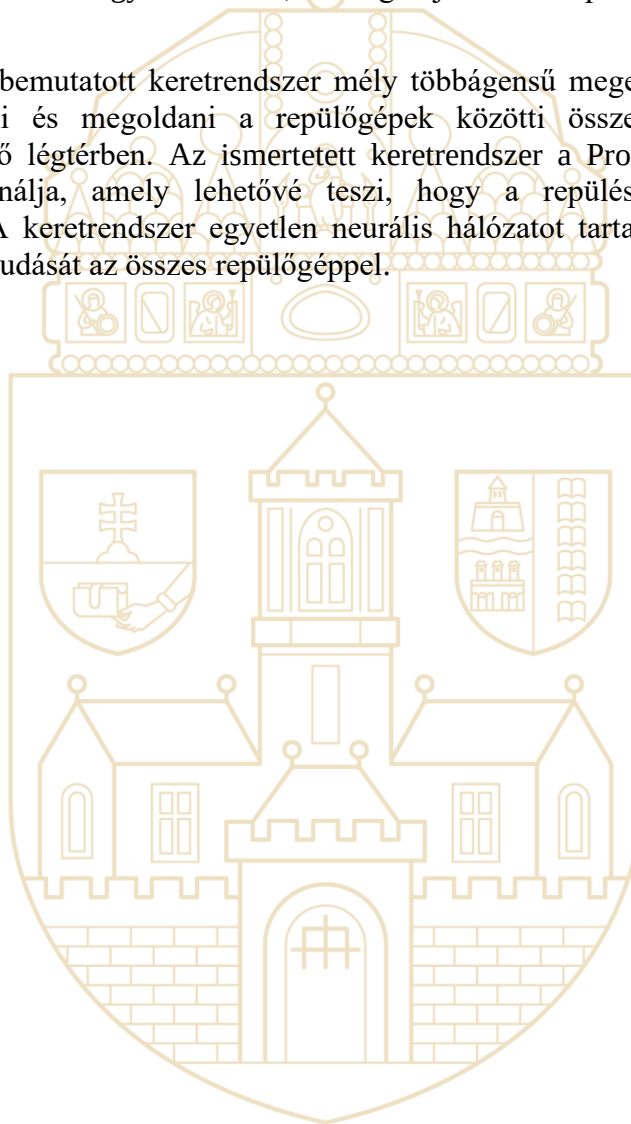


1. Bevezetés

Napjainkban, az egyre növekvő légiforgalom miatt a repülőgépek légtérben való közlekedésének biztonsága és hatékonysága egyre nagyobb kihívást jelent a pilótáknak, így szükségessé válik az autonóm légiforgalom irányítási rendszerek fejlesztése. Az ilyen rendszerek fejlesztésének egyik lehetséges módja a megerősítéses tanulás alkalmazása.

A megerősítéses tanulás [1], a gépi tanulás olyan ága, melynek célja, hogy lehetővé tegye az ágensek (repülőgépek) számára az optimális stratégia megtanulását azáltal, hogy kölcsönhatásba lép egy környezettel. Az ágensek a cél elérése érdekében először feltérképezik a környezet állapotát, majd ennek megfelelően választanak egy műveletet, és végül jutalmat kapnak az állapot és a művelet alapján.

A szakdolgozat által bemutatott keretrendszer mély többágensű megerősítéses tanuláson alapul, amely képes azonosítani és megoldani a repülőgépek közötti összeütközéseket a forgalmas, gócpontokkal rendelkező légtérben. Az ismertetett keretrendszer a Proximal Policy Optimization (PPO) algoritmust használja, amely lehetővé teszi, hogy a repülésirányítás hatékonyabb és biztonságosabb legyen. A keretrendszer egyetlen neurális hálózatot tartalmazó centralizált tanulást valósít meg megosztva a tudását az összes repülőgéppel.





2. A feladat

A mesterséges intelligenciát (MI) felhasználva egy légiforgalom irányítás implementálása és szimulációja a feladat. A szimulált környezet az MI-vel támogatott légiforgalom irányítás szimulációjára, valamint az MI tesztelésére széles körben elterjedt BlueSky keretrendszer. Az egyes repülőgépek azonosítószámokkal (járatszám) és egyéb, a szimulációhoz szükséges, adatokkal (pl. magasság, sebesség) ellátott pontokként jelennének meg.

A repülésirányításban rengeteg megoldandó helyzet alakulhat ki, ami vészhelyzeti protokoll alkalmazását teszi szükségessé, pl. meghibásodik egy repülőgép vagy elfogy az üzemanyaga. Ezen problémakörök közül az ütközés elkerülést (collision avoidance) választottam. Az összeütközések megelőzése érdekében a megvalósítandó MI ágenseknek az alábbiak szerint kell viselkedniük:

Az ágenseknek szükséges tartaniuk az egymás közötti legalább minimális vertikális távolságot annak érdekében, hogy biztonságosan tudjanak közlekedni a levegőben. Amennyiben két vagy több repülőgép egymás felé halad, és következtethető az ütközés valószínűsége, akkor az MI-nek szükséges beavatkoznia úgy, hogy az adott ágens repülési magasságát csökkenti, vagy növeli oly mértékben, hogy legalább a minimális vertikális szeparációt tartsa az ágens, mire a horizontális távolság a minimum alá megy.

3. A kutatás célja

A kutatás célja egy olyan program tervezése, amely képes kezelni minden a valóságban is előfordulható légi eseményt (pl.: le- és felszállás igény, vészhelyzeti protokoll, rossz időjárási viszonyok), és erre optimális megoldást javasolni az ágens számára. A program képes a párhuzamos működésre, azaz a repülőgépek egyszerre mozognak a radar képernyőn, ezáltal a repülőgépek hatással lehetnek egymás útvonalára, amelyet a program szintén kezel annak érdekében, hogy két vagy több repülőgép ne ütközzön össze.



4. Mesterséges intelligencia (MI)

Az MI [2] az intelligens gépek, különösen az intelligens számítógépes programok készítésének tudománya és mérnöki tudománya. Számítógépeket használnak az emberi intelligencia megértéséhez, viszont a mesterséges intelligenciának nem kell biológiai módszerekre hagyatkoznia. [3], „A mesterséges intelligencia a hagyományos rendszerek működését olyan módon képes kiegészíteni, hogy az önállóan tanulásra képes, ezáltal – külső beavatkozás nélkül fejleszt egy adott problémára megoldást – majd ezt a jövőbeni problémák megoldásakor is felhasználja. A feladatok, részfeladatok végrehajtásának hatékonyságát növelve és a hibalehetőségek kockázatát csökkentve a technológia képes több forrásból származó és az eddigieknél nagyobb mennyiségű információ feldolgozására is.”

Összefoglalva tehát az MI-t [4] használó rendszerek tudásfeldolgozó rendszerek. A tudásreprezentáció, tudásszerzés és a következtetés, beleértve a keresést és a vezérlést, három alapvető technika az MI-ben.

Az MI két részre osztható a filozófiája alapján. Az egyik megközelítés az erős MI, ami mögött meghúzódó elv az, hogy a gépek lehetnek gondolkodásra fejlesztve, vagy más szóval a jövőben az emberi elmét képviselhetik. Így az erős MI azt állítja, hogy a közeljövőben olyan gépekkel leszünk körülvéve, amelyek teljesen úgy működnek, mint az emberek, és a gép, ember szintű intelligenciával rendelkezhet. A jelenlegi kutatások közel sem hoznak létre erős MI-t, és élénk vita folyik arról, hogy ez egyáltalán lehetséges-e. A másik elképzelés a gyenge MI, mely azokon a tényeken alapszik, hogy a gépeket igenis lehet úgy fejleszteni, mintha intelligensek lennének. A gyenge MI kijelenti, hogy a gondolkodáshoz hasonló funkciók könnyedén hozzáadhatók a számítógéphez, hogy hasznosabb eszközök legyenek. Ez a folyamat már el is kezdődött. Erre egy példa, amikor egy ember sakkozik egy számítógéppel, úgy érezheti, mintha a számítógép valóban lenyűgöző lépéseket tenne meg a győzelem érdekében, de a sakkprogram nem gondolkodik és nem is tervez előre. Az összes lépését előre betáplálták az alkalmazásba, és ezért biztosított, hogy a szoftver a megfelelő időben a megfelelő lépést hozza meg.

A filozófia felosztáshoz hasonlóan az MI kutatásának is két fő irányvonala van [2]. Az egyik irányvonal a biológiai alapokra épít, azon gondolat alapján, hogy mivel az emberek intelligensek, ezért az MI-nek tanulmányoznia és utánoznia szükséges az emberek pszichológiáját vagy fiziológiáját. A másik irányzat ezzel szemben a gondolkodást veszi alapul, azon belül is a gondolkodásnak azt a működési mechanizmusát, ahogy az leképezi a környező világot, valamint a környező világ által felvetett problémákat igyekszik tanulmányozni és formalizálni, hogy ilyen módon érje el a célját. Mindkét megközelítés kölcsönhatásban van egymással bizonyos mértékig, és végül valószínűleg mindkettőnek sikerülnie kell. Ez egy verseny, de mindkét versenyző „jární” látszik.



A második világháború után, számos ember függetlenül elkezdett dolgozni intelligens gépek készítésében. Alan Turing angol matematikus lehetett az első [2], aki feltárta a mesterséges intelligencia matematikai lehetőségét [5], majd később 1947-ben előadást is tartott a mesterséges intelligenciáról. Szintén ő lehetett az első [2], aki kimondta, hogy a mesterséges intelligenciát leginkább számítógépek programozásával lehetett kutatni, mint gépek építésével. 1950-es évek végére sok ember kutatót a mesterséges intelligencia után és legtöbbjük a számítógépek programozására alapozta munkáját.

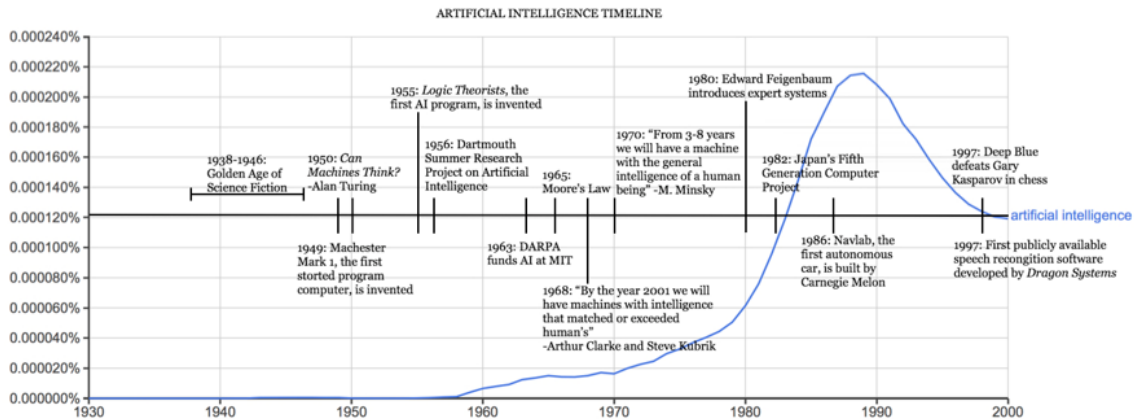
Számos dolog akadályozta Turingot, hogy akkor és azonnal munkába álljon. Először is a számítógépeken alapvető változtatásokat kellett végezni, mivel 1949 előtt a számítógépekből még hiányzott az „intelligencia” egyik kulcsfontosságú előfeltétele: nem voltak képesek ugyanis parancsokat tárolni, csak és kizárólag végrehajtani tudták az adott parancsokat. Más szóval, a számítógépeknek meg lehetett mondani, hogy mit tegyenek, de nem emlékeztek arra, hogy mit csináltak. Továbbá a számítás rendkívül költséges volt, mivel az 1950-es évek elején a számítógép lízingköltsége elérte a 200.000 dollárt havonta.

Öt évvel később megjelent az első MI-vel támogatott program a Logic Theorists, ami Allen Newell, Cliff Shaw és Herbert Simon nevéhez fűződik. A program, amit John McCarthy és Marvin Minsky 1956-ban mutatott be egy konferencián (nevezetesen a „Dartmouth Summer Research Project on Artificial Intelligence” (DSRPAI) nevűn), az emberi problémamegoldó képességek utánzását tűzte ki célul. A konferencia sikeressége kétségtelenül meghatározta az elkövetkezendő húsz év MI utáni kutatását.

1957 és 1974 között az MI kutatása virágzásnak indult, ugyanis a számítógépek egyre több információt tudtak tárolni, gyorsabbak és hozzáférhetőbbek lettek. A gépi tanulási algoritmusok javultak és a fejlesztők is jobban felismerték, hogy melyik algoritmust kell, hogy használják az adott probléma megoldására.

Az olyan korai bemutatók, mint Newell és Simon általános problémamegoldója és Joseph Weizenbaum ELIZA program [5] (Az ELIZA egy olyan program, amely biztosítja a természetes nyelvi beszélgetés lehetőségét az ember és számítógép között [6]), ígéretes lépéseket tettek a problémamegoldás és a beszélt nyelv értelmezése felé. Ezek a sikerek, valamint a vezető kutatók (nevezetesen a DSRPAI résztvevői) támogatása meggyőzte az olyan kormányzati szerveket, mint a „Defence Advance Research Projects Agency” (DARPA), hogy finanszírozzák az MI kutatást számos intézményben. 1960-ban Marvin Minsky azt mondta a Life Magazine-nak, hogy három-nyolc év múlva olyan gépünk lesz, amely egy átlagos emberi lény általános intelligenciájával rendelkezik majd.

1965-ben Moore kimondta híres törvényét, hogy a számítógépek memóriája és sebessége minden évben megduplázódik.



1. ábra
A Mesterséges Intelligencia idővonala

Az MI kezdeti kódének áttörése akadályok hegyeit tárt fel. A legnagyobb a számítási teljesítmény hiánya volt, bármilyen lényeges probléma elvégzéséről is volt szó: a számítógépek egyszerűen nem tudtak elegendő információt tárolni, illetve elég gyorsan feldolgozni adatokat. Ennek következtében ahogy fogyott a türelem, úgy fogyott a finanszírozás is, ezáltal a kutatás is csak lassú ütemben haladt tíz évig.

Az 1980-as években az MI az algoritmikus eszköztárak és az alapok bővítésének köszönhetően indult újra. John Hopfield és David Rumelhart népszerűsítette a „deep learning” technikákat, amelyek lehetővé tették a számítógépek számára, hogy a tapasztalatok felhasználásával tanuljanak. Másrészt Edward Feigenbaum olyan szakértői rendszereket mutatott be, amelyek a humán szakértő döntéshozatali folyamatát utánozták. A szakértői rendszereket széles körben kezdték el használni az iparban. A japán kormány jelentős mértékben finanszírozott szakértői rendszereket és más MI-vel kapcsolatos erőfeszítéseket a „Fifth Generation Computer Project” részeként.

Az 1990-es és 2000-es években az MI számos mérföldkönek számító célja megvalósult. 1997-ben Gary Kasparov akkori sakkvilágbajnok és nagymester vereséget szenvedett az IBM Deep Blue nevű számítógépes sakkprogramjától. Ugyanebben az évben a „Dragon System” által kifejlesztett beszédelismerő szoftvert implementálták Windowsra. Úgy tűnt, nincs olyan probléma, amit a gépek ne tudnának kezelni. Még az emberi érzelmek utánzása sem jelentett gondot, ugyanis Cynthia Breazeal kifejlesztett egy robotot, amely képes felismerni és megjeleníteni az érzelmeket.



Jelenleg a Big Data korát éljük, egy olyan korban, amelyben képesek vagyunk hatalmas mennyiségű információt gyűjteni, ami túl nehézkes ahhoz, hogy az ember feldolgozza. A Big Data lehetővé teszi az MI számára, hogy „brute force” algoritmus által tanuljon. Az MI alkalmazása ebben a tekintetben már igen gyümölcsözőnek bizonyult számos iparágban, mint például a technológia, bankszektor, marketing és a szórakoztatás. [5]

E rövid történelmi áttekintés után sokakban felmerülhet az a kérdés, hogy az MI célja-e egyáltalán az emberi intelligencia szintjének elérése. Erre a válasz az, hogy igen, mivel olyan számítógépes programok készítése a cél, amelyek képesek problémákat megoldani, célokat elérni csakúgy, mint az emberek, [2] és amelyek intelligenciája eléri vagy akár felülmúlja az emberi intelligenciát. [4] Viszont, ahogy említettem már, nem lehet előre megmondani, hogy mikor következik be az az idő, amikor az emberi intelligenciát eléri a mesterséges intelligencia, illetve kérdéses, hogy egyáltalán ez bekövetkezik-e. [2]

4.1. A mesterséges intelligencia ágai

Ebben a fejezetben az MI ágait [2] szeretném ismertetni, viszont a dokumentált ágak közül néhány ág biztosan hiányzik, mivel még senki sem azonosította őket. Ezek közül néhány inkább fogalomnak vagy témának tekinthető, mintsem teljes ágnak.

4.2. Logikai MI

Amit a program a világról tud, amelyben cselekednie kell, az általában az adott helyzet tényei, és minden elérendő célja valamilyen matematikai logikai nyelv „mondataival” van leképezve. A program annak függvényében dönt, hogy a következtetése alkalmas-e a céljai eléréséhez, vagy sem.

4.2.1. Keresés

Az MI-vel támogatott programok gyakran a nagyszámú lehetőségek elemzésein alapulnak, mint például egy sakkjátékban, ahol az összes bábú lépéseinek lehetőségeit megvizsgálva az optimális döntést hozza meg a program.

4.2.2. Minta-felismerés

Amikor egy program valamilyen megfigyelést végez, gyakran úgy van programozva, hogy összehasonlítsa a látottakat egy mintával, mint például egy kamera program, ami megpróbálja összeegyeztetni a szem és az orr mintáját egy színpadon, hogy arcot találjon. Bonyolultabb mintákat is tanítanak, mint például sakkpozíciókat vagy természetes írást, amelyek jóval különbözőbb metódusokat igényelnek, mint az egyszerű minták tanulása.



4.2.3. Ábrázolás

Általában a matematikai logika nyelveit használják a reprezentálásra.

4.2.4. Következtetés

Az MI-t felhasználó program következtetni képes bizonyos tényekből, mint például egy futball mérkőzés program, ami évekre visszamenőleg eltárolja a csapatok győzelmi rátáját, és ebből következtethet, hogy a következő mérkőzés kimenete mi lesz.

4.2.5. Gondolkodás, tudás és érvelés

Ez az a terület, melynél az MI a legmesszebb van az emberi szinttől, annak ellenére, hogy már 1950-től aktív kutatás folyik, és bizonyos részfeladatokban, mint pl. a nem monoton érvelés, cselekvésemélet jelentős előrehaladás mutatkozott.

4.2.6. Tapasztalatokon alapú tanulás

Neurális hálók segítségével képes a tapasztalatokból tanulni a program, mint például egy sakkjátékban, ahol az idő elteltével a program megfontoltabb lépéseket tesz a győzelem érdekében.

4.2.7. Tervezés

A programok képesek tervezni, amelyet egy adott helyzetre való tény felállításával kezdenek, majd pedig a cél kitűzésével. Ezután stratégiát tervel ki a program a cél elérése érdekében. Leggyakoribb esetekben a stratégia csak műveletek sorozata.

4.2.8. Ismeretelmélet

Ez a terület a megismerés fajtáinak tanulmányozásáról szól, amelyek ahhoz szükségesek, hogy a világban előforduló problémákat megoldjuk.

4.2.9. Ontológia

Az ontológia a valóságban előforduló különböző dolgok fajtáit tanulmányozza. Az MI-ben a programok különféle fajtájú objektumokat jelenítenek meg és eközben arra vagyunk kíváncsiak, hogy melyek ezek a fajták és mik az alapvető tulajdonságaik.

4.2.10. Heurisztika

A heurisztika [7] azt a folyamatot jelöli, mely során nem szigorúan szabatos logikai következtetésekkel jutunk el a feltételezéstől a végkövetkeztetésig, az eredmény mégis helyes lesz. [2]A kifejezést többféle értelmezésben is használják az MI világában. Pl. heurisztikus függvényeket használnak bizonyos megközelítésekben arra, hogy megállapítsák a mértékét annak, hogy egy adott csomópont milyen távol van a keresőfában a céltól. Míg a heurisztikus predikátum során két csomópontot hasonlítunk össze egy keresőfában, hogy eldöntsük melyik számít jobbnak a cél elérése szempontjából.



4.2.11. Genetikus programozás

A genetikus programozás [8] a genetikus algoritmus egy speciális alkalmazási területe, amikor is megadott feladatokat automatikusan végrehajtó számítógépes programok kifejlesztése a cél. A genetikus programozás során a program az aktuális populációból iteratív módon új populációt hoz létre, úgy, hogy a megadott szelekciós eljárás által kiválasztott legalkalmasabb egyedeken alkalmazza a rekombinációs és mutációs operátorokat, avagy a genetikus operátorokat. Az informatikában előforduló problémák széles osztályára alkalmazható, ugyanis, akkor is képes megoldást adni, ha nem ismert a pontos célérték, vagy a sok lehetséges megoldás közül az optimális megoldás megtalálása a cél. A genetikus operátorokat [9] a mutáció, szaporodás, génduplikáció, géntörlesztés tartalmazza.

4.2.12. Megerősítéssel-tanulás (MT)

A standard Megerősítéssel-tanulás [10] modellben az ágens érzékelés és cselekvés révén kapcsolódik a környezetéhez. [11] Minden egyes időlépésben az ágens állapotot kap az állapottérben és kiválaszt egy műveletet a művelettérből egy meghatározott irányelvet követve. Állapotról cselekvésre leképezve az ágens skaláris jutalmat kap és a környezet dinamikájának vagy modelljének megfelelően átvált a következő állapotba. Ez a folyamat addig folytatódik, amíg az ágens el nem éri a terminális állapotot, majd újraindul. A visszatérés a diszkontált felhalmozott jutalom a diszkonttényezővel együtt.

4.3. A mesterséges intelligencia alkalmazási területei

Az MI-t már napjainkban is az informatika és a mindennapi élet számos területén alkalmazzák és ez a fejlődés előrehaladtával csak bővülni fog. Ezek közül most csak a leggyakoribbakat emelném ki [4]:

4.3.1. Játékipar

A játékiparban az egyik legismertebb program, amely MI-t használ a sakkjáték. Annak ellenére, hogy ezek a programok nem olyan intelligensek, mint az emberek, „brute force”, vagyis nyers erő algoritmust használnak, ami alapján több százezer pozíciót másodpercenként megvizsgálva határozzák meg a következő lépést. Példának felhozhatom az „AI Deep Blue II” sakkprogramot, mely legyőzte a sakkmester Garry Kasparov-ot, készítői így próbálták bebizonyítani, hogy az MI okosabb, mint az ember. Emellett megemlíteném még a Microsoft Xbox 360-at is, ahol az MI-t mozgásérzékelőként használták fel.

4.3.2. Nehézipar

A nehéziparban az MI-t robotok működtetésére használják olyan munkakörökben, amelyek veszélyesek lehetnek az emberek számára. Ezek a robotok hatékonyabban látják el a feladatukat, mint az emberek, mivel nincs szükségük szünetre.

4.3.3. Időjárás előrejelzés

A meteorológiai intézetben neurális hálózatok segítségével jósolják meg az időjárási viszonyokat. A múltbéli adatok rendelkezésre állnak a neurális hálózatnak, amely azután analizálja az adatokat a minták alapján és megjósolja a jövőbeni időjárási viszonyokat.



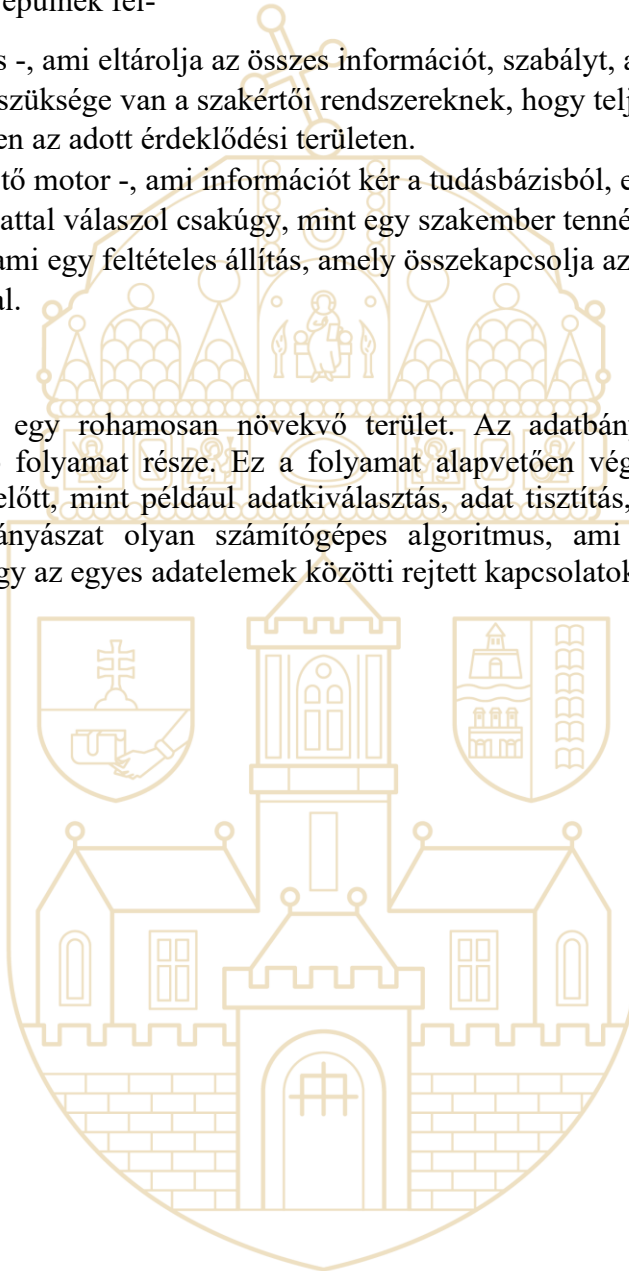
4.3.4. Szakértői rendszerek

A szakértői rendszerek olyan gépek, amelyek arra lettek kifejlesztve, hogy bizonyos érdeklődési területeken teljes szakértői készségük legyen, ezáltal feladatokat legyenek képesek megoldani. Ezek a rendszerek statisztikai elemzést és adatbányászatot használnak a problémák megoldására, amely során eldöntendő kérdések segítségével vezetik le a megoldást. A szakértői rendszerek három részből épülnek fel-

- Tudásbázis -, ami eltárolja az összes információt, szabályt, adatot és kapcsolatot, amelyekre szüksége van a szakértői rendszereknek, hogy teljes szakértelemmel rendelkezzen az adott érdeklődési területen.
- Következtető motor -, ami információt kér a tudásbázisból, elemzi azt és megoldással vagy javaslattal válaszol csakúgy, mint egy szakember tenné.
- Szabály -, ami egy feltételes állítás, amely összekapcsolja az adott feltételeket a végső megoldással.

4.3.5. Adatbányászat

Az adatbányászat egy rohamosan növekvő terület. Az adatbányászat a „tudásfeltárás az adatbázisban” néven futó folyamat része. Ez a folyamat alapvetően végrehajtott lépésekből áll az adatbányászat elvégzése előtt, mint például adatkiválasztás, adat tisztítás, adatok előfeldolgozása és adatátalakítás. Az adatbányászat olyan számítógépes algoritmus, ami egy nagy adathalmazban mintázatok után kutat és így az egyes adatelemek közötti rejtett kapcsolatokat tárja fel.





4.4. Mesterséges intelligencia felhasználása repülőtereken/repülésben

Ezen alkalmazási területet külön kiemelném az előzőekben felsoroltakhoz képest, hisz a kutatási feladatom is ide tartozik. Az MI-t már jelenleg is nagyon változatos módon használják a repülőtereken, illetve a repülésben. Ezek közül az alábbi pár példát említeném meg:

Londonban a Heathrow repülőtér 5-ös terminálján MI-vel támogatott kamerákat állítottak fel [12], amely figyelmeztetést küld a gép fordulójáért felelős dolgozó okosórájára, abban az esetben, ha olyan hibát észlel, amely késést idézhet elő. Ezáltal a felelős személy intézkedni tud a késés megelőzése érdekében.

A pittsburgh-i repülőtéren [13] szintén kamerák segítségével alkalmazzák az MI-t, viszont ebben az esetben a program a biztonsági ellenőrzésre várható várakozási időt számolja ki és ossza meg az utasokkal.

A frankfurti reptéren [14] MI-vel támogatott robot, FRAnny segíti az utasokat az eligazodásban Voice User Interface, azaz hang alapú közvetítés segítségével. Eme asszisztens 9 nyelven képes felvenni a kapcsolatot az utasokkal, amely az általános kérdésekre válaszol, viszont ezen felül csevegésre is alkalmas.

Az MI-t fel lehet használni másodpilótaként a repülésben [3], mivel hatalmas mennyiségű adatot képes úgy feldolgozni rövid idő alatt, hogy a végső döntéshozót, azaz a pilótát az optimális döntés meghozatalában elősegítse.

Végül, de nem utolsó sorban jöjjön a kutatási témám, azaz a légi forgalmi irányítók döntéstámogató rendszerei, ahol szintén alkalmaznak MI-t. Ennek [3],kifejlesztésekor elemezték a munkafolyamatok teljes spektrumát, az alkalmazandó szabályokat, az adott légtérstruktúrát, szektorkiosztási konfigurációkat, a jellemző és uralkodó időjárási jelenségek okozta korlátozásokat, amelyek-ben a mesterséges intelligencia szerepe abban áll, hogy képes elemezni az aktuális helyzetet, figyelembe veszi azokat a faktorokat, amelyek befolyásolják a repülés biztonságát, és döntési javaslatot adnak az légiforgalom-irányítónak. Az irányító akár elfogadja vagy módosításokat tesz a végső döntésében, ezeket a mesterséges intelligencia elemezi és eltárolja, mint alkalmazható lehetőséget.”



5. A légiforgalom irányítás legfontosabb elemei, működési elve

A légiforgalom irányítást (Air Traffic Control, röviden ATC) a légiforgalom menedzsment (ATM) egyik alappillérenek tekintik. A légiforgalom irányító rendszerek biztosítják, hogy a repülőgépek biztonságosan el legyenek választva mind a levegőben, mind a fel- és leszálláskor. A légiforgalom irányító rendszer kezeli az összes földön- és a légiútvonalon közlekedő repülőgépek tevékenységeit az ütközések elkerülése és a forgalomfolyam megszervezése érdekében. A légiforgalom irányítás úgy tekinthető, mint egy valós idejű rendszer, amely összehangoltan működik az emberekkel, az MI segítségével. A légiforgalom irányító rendszer legfontosabb feladata tehát, hogy meggátolja a repülőgépek összeütközését, elősegítve ezzel a biztonságos közlekedést a légi és földi úton egyaránt.

Az ATC-ben [15] a légtér számos szektorra oszlik, ami zónákba van sorolva. Minden szektorban vannak ATC vezérlők, amelyek navigálják a repülőgépeket a repülés, valamint a fel- és leszállás során. Amikor a repülőgép az egyik szektorból a másikba repül át, a vezérlés a repülési terv szerinti útvonal következő szektorához tartozó következő vezérlőhöz kerül. A vezérlésnek az ily módon szektorról szektorra történő átadása az egyes vezérlők legalapvetőbb koordináló funkciója valamennyi szektorra vonatkozóan. A vezérlőnek nem szabad engednie, hogy a légijármű elhagyja a szektort addig, amíg a következő szektorban lévő vezérlő át nem veszi az irányítást a járműre vonatkozóan. Ez az ATC vezérlő egyik leglényegesebb része. Emellett természetesen a pilótával való kapcsolat elvesztése sem megengedhető, ugyanis műszaki problémák, vagy akár egészségügyi vészhelyzet is kialakulhat a repülés közben. Az ATC rendszerbe történő belépéshez minden felszállás előtti repülőgépnek egy repülési tervet kell regisztrálnia az ATC vezérlőnél, amely tartalmazni fogja a repülés alatt követendő útvonal részleteit. Az útvonal az időjárási viszonyok és egyéb fontos körülmény előzetes feltételezése alapján kerül meghatározásra. Az útvonalkészítés, a repülési terv létrehozása a pilóta feladata. A repülési terv meggenerálása után, az ATC rendszer ellenőrzi az útvonalat, jóváhagyja és megadja az engedélyt a felszállásra. A felszállás után a pilóta várhatóan az egyeztetett útvonal mentén navigál, viszont abban az esetben, ha a pilóta mégis eltér az útvonaltól, akkor az ATC vezérlő veszi át az irányítást és visszahelyezi a gépet a korábban jóváhagyott repülési terv szerinti útvonalra. Különböző helyzetekben a vezérlő alternatív útvonalat is javasolhat. A vezérlők fontos szerepet játszanak a repülési magasság meghatározásában is. A pilótáknak engedélyt kell kérniük a vezérlőtől a magasság változtatására, és csak az engedély megadása után változtathatnak a magasságon. A pilótáknak biztonságos távolságot kell tartaniuk a repülőgépek között, mind horizontálisan, mind vertikálisan. Erre az ATC radarokat alkalmaz a járatok nyomon követésére, a szektorokon és zónákon belül.



6. Autonóm légiforgalom irányítás rendszerek megvalósítására irányuló legújabb módszerek, modellek

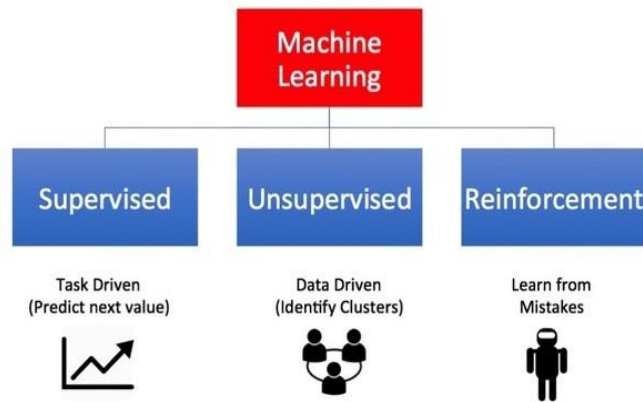
Ezen fejezetben röviden ismertetnék három olyan konkrét kutatási irányvonalat, működő módszert, mely a feladatomhoz hasonlóan egy autonóm légiforgalom irányítási rendszer megvalósítását tűzte ki célul, MI felhasználásával. Az ezen publikációkból kinyerhető információk, eljárások, modellek tanulmányozása révén új ismeretekre tehetek szert, továbbá ötleteket meríthetek arra vonatkozóan, hogy milyen irányban érdemes további tanulmányokat folytatnom. Mindez nagyban segíti kutatási célom elérését.

6.1. Deep MARL

2019-ben publikálták Marc Brittain és Peng Wei a: „Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach” [1] című munkásságukat, amelyben egy mély többágensű megerősítéses tanuláson alapuló megközelítést alkalmaztak az autonóm légiforgalom irányítás problémára vonatkozóan.

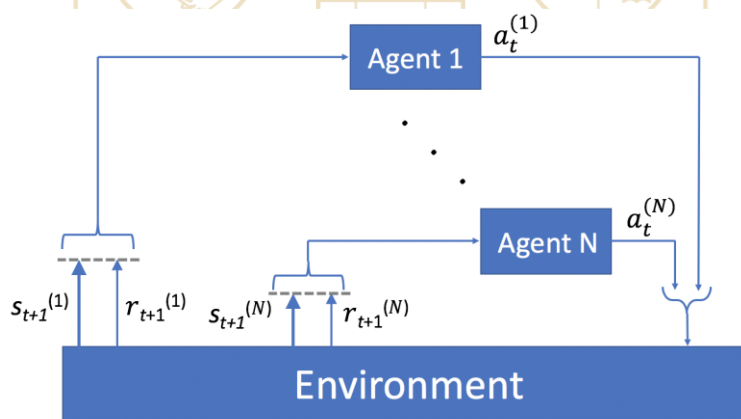
A globális légiforgalomban észlelhető rohamos növekedés nehéz feladattá tette a hatékony és biztonságos légiközlekedést. A légiforgalom irányításban a repülőgépek biztonságos elkülönítése már 50 éve emberek döntésein alapszik. Heinz Erzberger és a NASA munkatársai autonóm légiforgalom irányítás rendszert javasoltak az üzembiztonság és a légtér kapacitás növelésének érdekében. Ez inspirálta Marc Brittain és Peng Wei munkásságának elkezdését, amely során egy teljesen automatikus légiforgalom irányítás rendszer elkészítését tűzték ki célul, ami szerintük végső megoldást ad a komplex, dinamikus és nagy forgalmú légiforgalom irányítás kezelésre. A legnagyobb feladatot abban látták, hogy egy olyan autonóm légiforgalom irányító rendszert hozzanak létre, amely képes valós idejű tanácsokkal ellátni a pilótát a biztonságos közlekedés érdekében, mind a légiútvonal mentén, mind a légiútvonalak kereszteződésében. Továbbá szükségesnek tartották, hogy a rendszer kezelje mind a többszörös légikereszteződés problematikáját, mind a légiforgalom irányítás során bekövetkező eseményekből, azaz környezetből adódó bizonytalanságokat. Úgy gondolták, hogy a megerősítéses tanulás, ami a gépi tanulás egyik ága, képes megoldani a fentebb említett problémákat.

Types of Machine Learning



2. ábra
A gépi tanulás ágai

Egy mély többágensű megerősítéssel tanuló keretrendszert dolgoztak ki, mely az úgynevezett irányelv alapú (policy-based) tanulási algoritmust használja, mely irányelveket generál minden egyes végrehajtandó ágens számára. Az érték alapú (value-based) tanulással szemben ugyanis az irányelv alapú tanulási algoritmus képes elsajátítani sztochasztikus irányelveket is. Modelljükben minden egyes ágens egy repülőgépet reprezentál. Egy adott ágens mindenkor állapota tartalmazza az N darab hozzá legközelebb eső ágens állapotára vonatkozó információkat is. Így ebben a multi ágensű környezetben, az egymásközi kommunikáció révén, minden ágens tisztában van az aktuális légiforgalommal -és online szekvenciális döntéseket hoz, mely során valós időben sebesség javaslatok közül választ, annak érdekében, hogy elkerülje az összeütközéseket a kereszteződésekben, a leszálló pályáknál és az útvonala mentén.

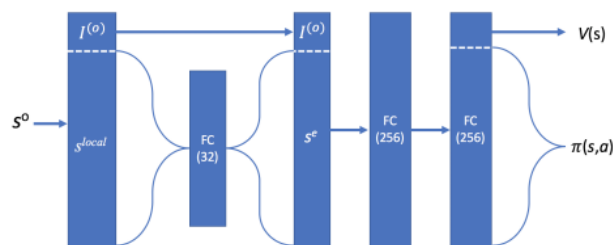


3. ábra
A MARL probléma előrehaladása



Az általuk javasolt keretrendszer egyetlen neurális hálózatot tartalmazó centralizált tanulást valósít meg, decentralizált végrehajtási sémák alkalmazása mellett, ami ebben a formában hatékonyan kezeli a már említett bizonytalanságot is. Ily módon a neurális hálózat központosított „tanulóként” működik, megosztva a tudását az összes repülőgéppel. Azaz minden ágensnek egy közös neurális hálózata van, amin osztozik a többi ágenssel, de mivel különböző állapotokat fejlesztenek ki az egyes ágensok, így a cselekvéseik is különbözőek lehetnek.

A keretrendszer további előnye, hogy magába foglalja, adaptálva az „actor-critic” (A2C) algoritmust, a „Proximal Policy Optimization” (PPO) veszteségfüggvény használatát, amely segítségével stabilizálható a tanulási folyamat.



4. ábra

Az A2C neurális hálózati architektúrájának illusztrációja

A tanulmány szerzői fontosnak tartották megemlíteni, hogy mivel a megközelítésük nem függ a repülőgépek számától, ezért a keretrendszerük skálázható és hatékony tetszőleges számban érkező repülőgépek biztonságos forgalomirányítására. Modelljüket a BlueSky környezet széleskörű szimulációin keresztül értékelték, melynek eredményeképpen bebizonyosodott, hogy a megalkotott keretrendszerük képes 99,97%-os megbízhatósággal kizárni a repülőgépek összeütközésének a lehetőségét a légi kereszteződésekre vonatkozóan, még extrém nagy légiforgalom esetében is. Ez az érték a landoló repülőgépek esetében 100% volt.



6.2. Deep Ensemble MARL

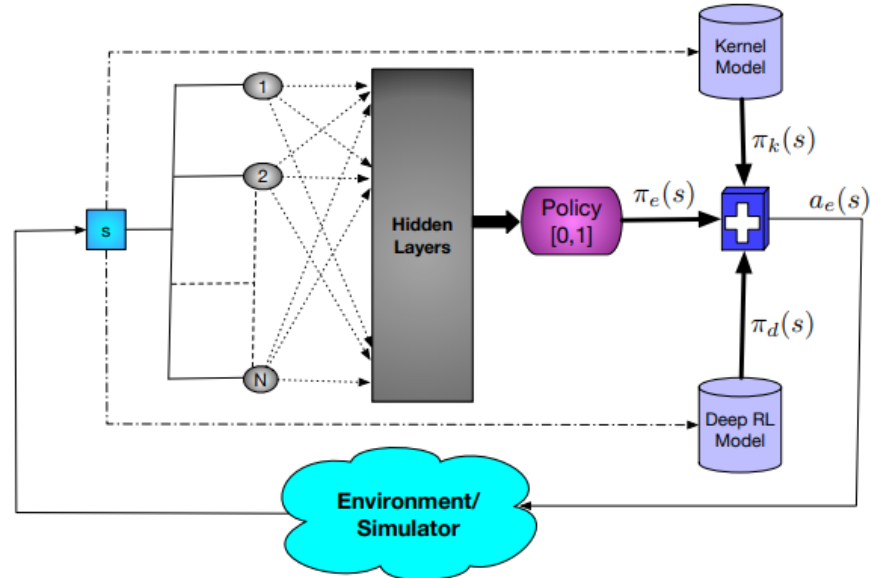
2021-ben publikálták az IBM és az Amazon munkatársai a: „A Deep Ensemble Method for Multi-Agent Reinforcement Learning: A Case Study on Air Traffic Control” [16] című munkásságukat, amelyben egy olyan keretrendszert dolgoztak ki, mely sikerrel ötvözi a modell alapú kernel Megerősítéses Tanulás (MT) és a modell nélküli mély MT megközelítés előnyeit. A tanulmányukban javasolt „deep ensemble” módszerüket a légiforgalom irányítás nagy kihívást jelentő problematikáján tesztelték.

Az MT-t, hasonlóan az előzőekben már ismertetett kutatási anyagban leírtakhoz, ezen publikáció szerzői is ígéretes módszernek tekintették az adat-vezérelt döntéshozatalra egy olyan bizonytalan környezetben, mint a légiforgalom irányítása. Kijelentésüket arra alapozták, hogy az MT-t már sikeresen alkalmazták számos valós műveletben és ellenőrzési problémákban. Ennek ellenére úgy látták, hogy az MT alkalmazása nagy kihívást jelent egy nagyméretű decentralizált többágensű környezetben, az ágensek közötti korlátozott kommunikáció, illetve a részleges megfigyelhetőség miatt. Írásukban emellett még arra is rámutattak, hogy egy kooperatív MARL (Multi-Agent Reinforcement Learning, azaz Többágensű Megerősítéses Tanulás) probléma megoldhatatlan, amennyiben azt egy központosított MT problémaként kezeljük, mivel a probléma nehézsége exponenciálisan nő az ágensek számával, a közös cselekvés, vagy állapotter miatt.

Kutatásuk során olyan, két egymást kiegészítő MT módszertant tartalmazó rendszert fejlesztettek ki, mely egy közös irányelv (shared policy) megtanulására törekszik, az összes ágensről begyűjtött tapasztalati minták felhasználásával. A rendszer egyik komponense, a modell-alapú kernel MT (model-base kernel RL) megközelítés, mely ágens-központú állapotábrázolást alkalmaz, rögzítve a helyi információkat az adott ágens egy bizonyos sugarú környezetében. Míg a másik rész, a modell-nélküli mély MT (modell-free deep RL) modell további szélesebb körű állapotinformációkat is figyelembe vesz.

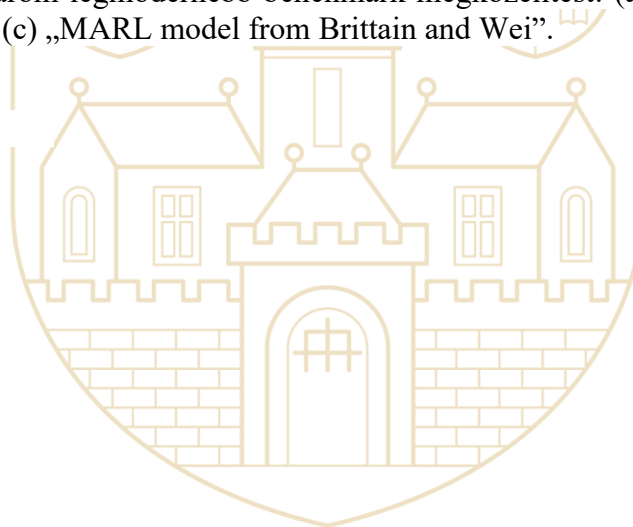
A kernel bázisú MT-től azt várták, hogy jól teljesít a végrehajtás során, amennyiben egy adott állapotnak megfelelően sűrű a környezete a tanítást elősegítő tapasztalati minták vonatkozásában, de meglehetősen rosszul extrapolál. Ezzel szemben a mély MARL módszer sokkal rugalmasabb és jól általánosítható, ugyanakkor hajlamos különféle kóros viselkedéseket produkálni vagy gyengén szerepelni még sűrű tréning adatokkal rendelkező régiókban is.

Mindezen okokból kifolyólag -egy újszerű együttes tanulási megközelítést javasoltak, melynek során mindkét módszer erősségeit kihasználhatják. Az általuk Mély Együttes Többágensű Megerősítéses Tanulásnak (deep ensemble MARL) nevezett módszer lényege, hogy egy különálló mély neurális hálózatot tanítanak, mely így hatékonyan megtanul dönteni az előtréningezett kernel-alapú MT és a szélesebb körű mély MT irányelvei között.



5. ábra
A Mély Együttes MARL módszer architektúrája

Javasolt többágensű módszerüket egy valós légiforgalom irányítási problémán tesztelték, az Eurocontrol által fejlesztett nyílt forráskódú légiforgalmi szimulátoron. A cél a légi járművek hatékony irányítása, a légiforgalmi torlódások, konfliktushelyzetek elkerülése, az érkezési idő, valamint az üzemanyagköltség javítása volt a repülőgép sebességének valós idejű módosításának dinamikus ajánlásával. Az eredmények szempontjából a Mély Együttes Többágensű MT módszerük jelentősen felülmúlja a három legmodernebb benchmark megközelítést: (a) „Baseline approach”; (b) „Local search approach”; (c) „MARL model from Brittain and Wei”.





6.3. Embedding ML in ATC

2019-ben publikálták Mukesh Madanan, Norlaila Hussain, Nitha C Velayudhan és Biju T Sayed a: „Embedding Machine Learning in Air Traffic Control Systems to generate effective route plans for aircrafts in order to avoid collisions” [15] című publikációjukat, amelyben az előbbiekhez hasonlóan a Gépi Tanulás (GT) segítségével adtak megoldást a légiforgalom irányítás problematikájára, azonban ezen belül is kifejezetten a hatékony útvonaltervezésre és a légijárművek összeütközésének elkerülésére helyezve a hangsúlyt.

A legtöbb útvonaltervezés emberekkel együttesen történik és a döntéshozatal az emberi intelligenciától függ, ami néha időigényes és hibaérzékeny. Az MI képességeinek a vezérlőkbe történő beépítésével gyors döntések születhetnek emberi beavatkozás nélkül. A kutatásuk során ezért olyan MI-vel támogatott légiforgalom irányító (ATC) rendszer tervezésére összpontosítottak, amely képes megtalálni a legjobb útvonalat a járatok számára, ezáltal minimálisra csökkentve az ütközések esélyét a levegőben és a földön, mindezt úgy, hogy minimalizálja az emberi beavatkozás szükségességét és ennek következtében az emberi hiba előfordulását.

Az MI számos elérhető technikája közül egy olyan GT alapú rendszer ATC-ben történő megvalósítására fókuszáltak a kutatók, amely előre betáplált adatokból, adatstatisztikák felhasználásával önmagát oktatja, képes hasonló mintázatok felismerésére, érvelési képességeken alapuló döntések meghozatalára, hogy aztán végül hatékony útvonaltervet készítsen. Az ATC vezérlővel szembeni funkcionális követelmény az, hogy az MI segítségével képes legyen nagy hatékonysággal előre megajánlani egy tökéletes útvonaltervet. Az előre megajánlás azért fontos, mert mint már korábban említettem, minden felszállás előtti repülőgépnek egy repülési tervet kell regisztrálnia az ATC vezérlőnél, amely tartalmazni fogja a repülés alatt követendő útvonal részleteit. A repülési útvonal megajánlása a rendszerbe betáplált korábbi adatok alapján történik.

Ami a repülőgépek ütközés-elkerülési rendszer döntéshozatali logikájának megvalósítását illeti a szerzők szerint egy lehetséges megközelítés, ha a problémát Markov-döntési folyamatként (Markov decision process) fogalmazzuk meg és a rendszert dinamikus programozással optimalizáljuk.

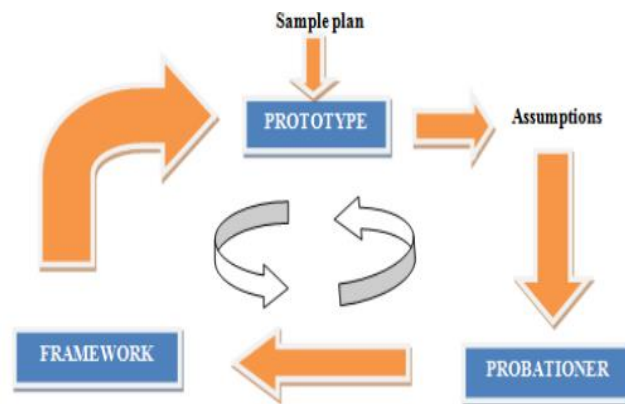
Az ATC GT modulja három részből áll: (i) Prototípus (ii) Keretrendszer (iii) Próbálkozó egység.

A rendszer Prototípus része feltételezéseket és előrejelzéseket tesz a repülési útvonalat illetően olyan tényezők alapján, mint pl.: időjárási viszonyok, sebesség, magasság stb.

A Keretrendszer tartalmazza azokat a tényezőket, amelyeket a prototípus használ a következtetések levonásához és a döntések meghozatalához.

Végül a Próbálkozó egység az a rész, amely módosítja a kerettényezőket és ezáltal a prototípust, összehasonlító vizsgálatot végezve a feltételezéseken és a tényleges kimenetelen. Emellett felhasználja a képzési adatokat, hogy önmagát tanítsa egy jobb prototípus létrehozása érdekében.

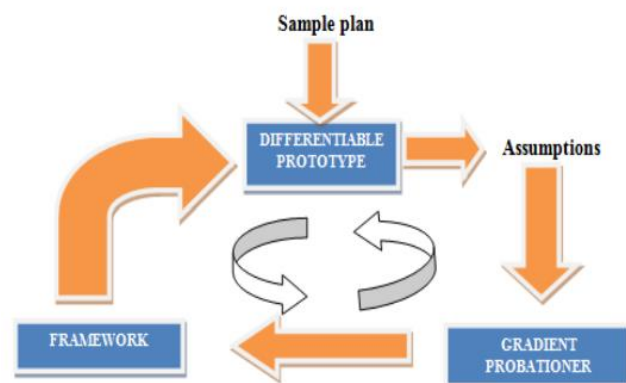
A rendszer leglényegesebb része a Prototípus és az egész rendszer a Prototípus feltételezéseivel kezdődik.



6. ábra

Az ATC gépi tanulási rendszerének modellje

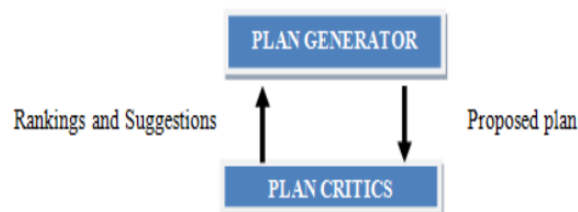
A rendszer GT része matematikai egyenletet alkalmaz, amit egy egyedi grafikonon vezet le, hogy kifejezzon egy általános, elvárt eredményt.



7. ábra

Az ATC gépi tanulási rendszerének átdolgozott modellje

A szoftver architektúrája két részből áll. Terv Generátor, ami valójában maga a Prototípus, valamint Terv Kritikus, amit a Keretrendszer és a Próbálkozó egység kombinációja alkot.



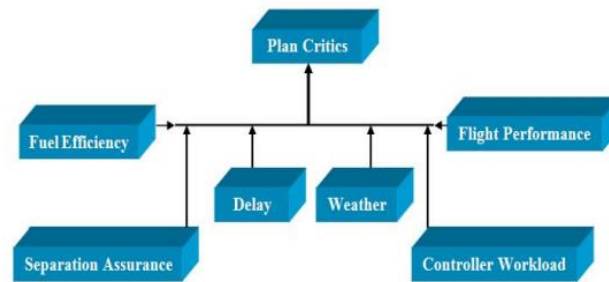
8. ábra

MI architektúra az ATC számára

A Terv Generátor az ATM minden szektorában használatos, annak érdekében, hogy útvonalterveket készítsen a repülőgépek irányítására az adott szektorban. Az elkészült terv

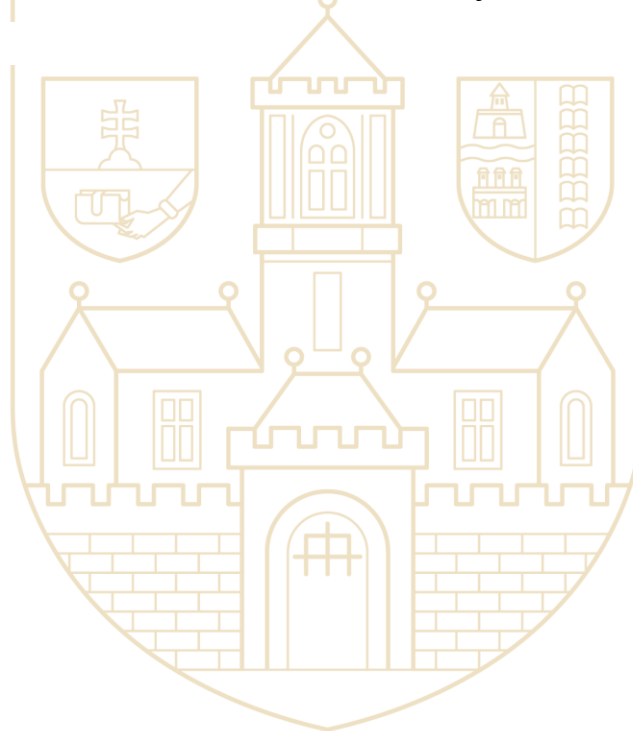


tartalmazza majd minden egyes repülőgép útvonalát, ezáltal a jövőbeni horizontális és vertikális pozícióját és azokat az engedélyeket, amelyek elengedhetetlenek ezekben a konkrét útvonalakban. Az engedélyek a sebesség és a magasság változtatására vonatkozhatnak, amelyet a légi járműnek a repülése során be kell tartania a cél repülőtérig.



9. ábra
Terv kritikus modul

Az elkészült terv ezután átkerül a Terv Kritikus modulba, amely függetlenül működve megkeresi a létrejött terv következményeit. A terv ezután értékelésre és rangsorolásra kerül a tervben szereplő nemkívánatos jellemzők szerint. A rangsorolás empirikus elemzés alapján is történhet. Minél magasabb a rang, annál nagyobb a tervben szereplő problémák súlyossága. Majd a Terv Kritikus által meghatározott rang visszakérül a Terv Generátorhoz különféle javaslatokkal.



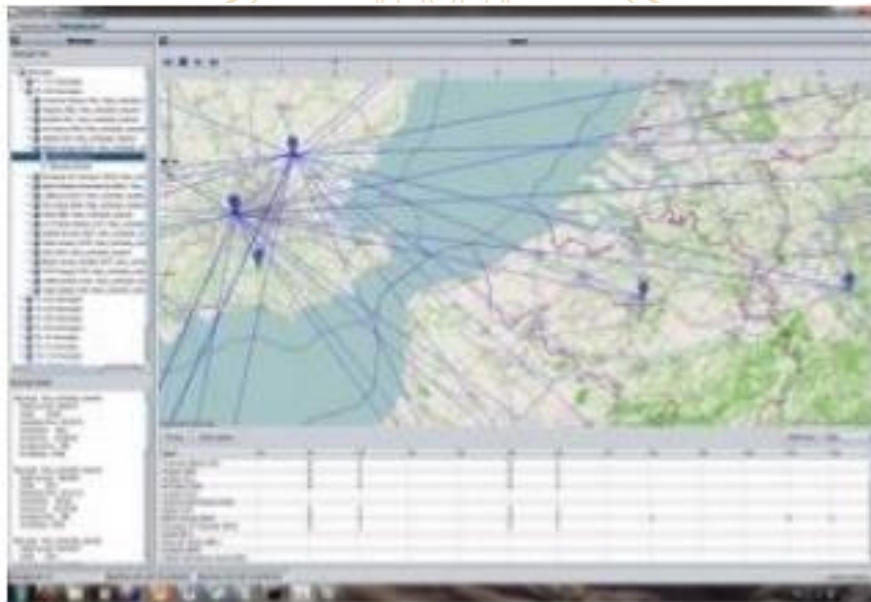


7. Légiforgalmi szimulátorok

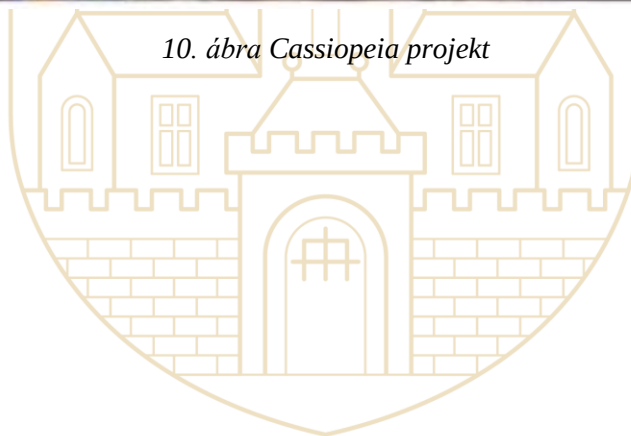
A következő fejezetben a BlueSky mellett a két legismertebb légiforgalmi szimulátort ismertetem. [17]

7.1. Cassiopeia

A Cassiopeia célja egy olyan online szimulációs szolgáltatás biztosítása, amely légiforgalmi komplexitás kutatását segíti elő. A használatához a felhasználóknak le kell fordítani egy használati esetet Auto-lead Data Format-ra, ami egy open XML alapú nyelv. Ez a megközelítés megköveteli a szimulátor némi ismeretét, ami sajnos nem teljesen nyílt forráskódú, ugyanis az alapötlet nem a szimulátor letöltése, hanem az online futtatása. Mivel ez egy szerveralapú zárt forráskódú szimulátor, így a program kibővítése saját konfliktusfelismerő, konfliktusmegoldó algoritmussal nem lehetséges.



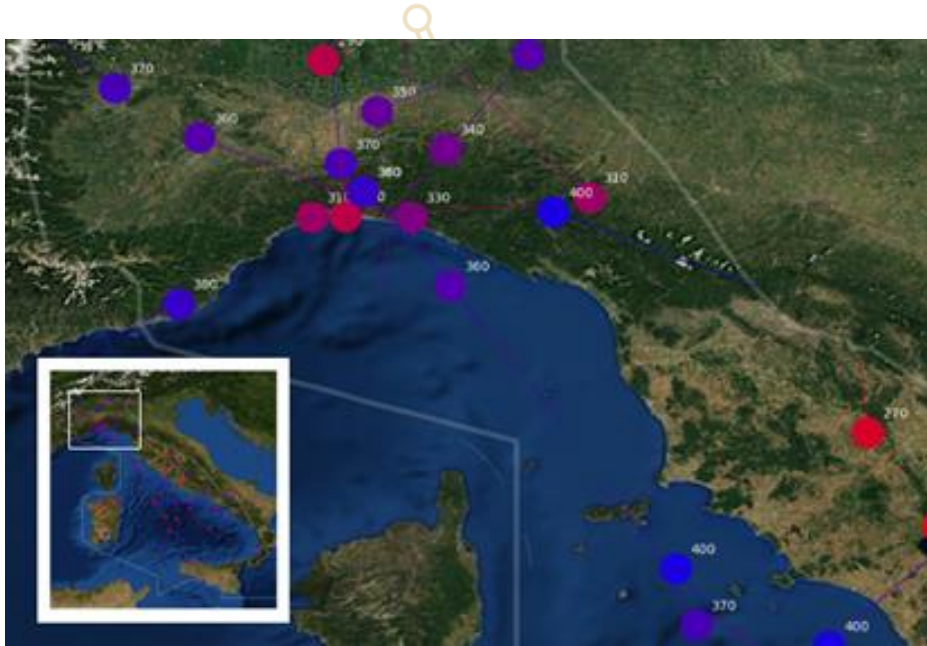
10. ábra Cassiopeia projekt





7.2. Elsa

Az Elsa egy nyílt forráskódú ágens alapú légiforgalmi szimulátor. Célja a forgalom komplexitás vizsgálata olyan scenáriók segítségével, amelyben repülési tervek kerülnek definiálásra. A program nyílt forráskódjának biztosítása egyszerűbbé teszi a program megértését, használatát, nem úgy, mint a Cassiopeia-nál. Az Elsa platformfüggető, mivel lefordított C kódot használ a Python kód mellett.



11. ábra Elsa projekt



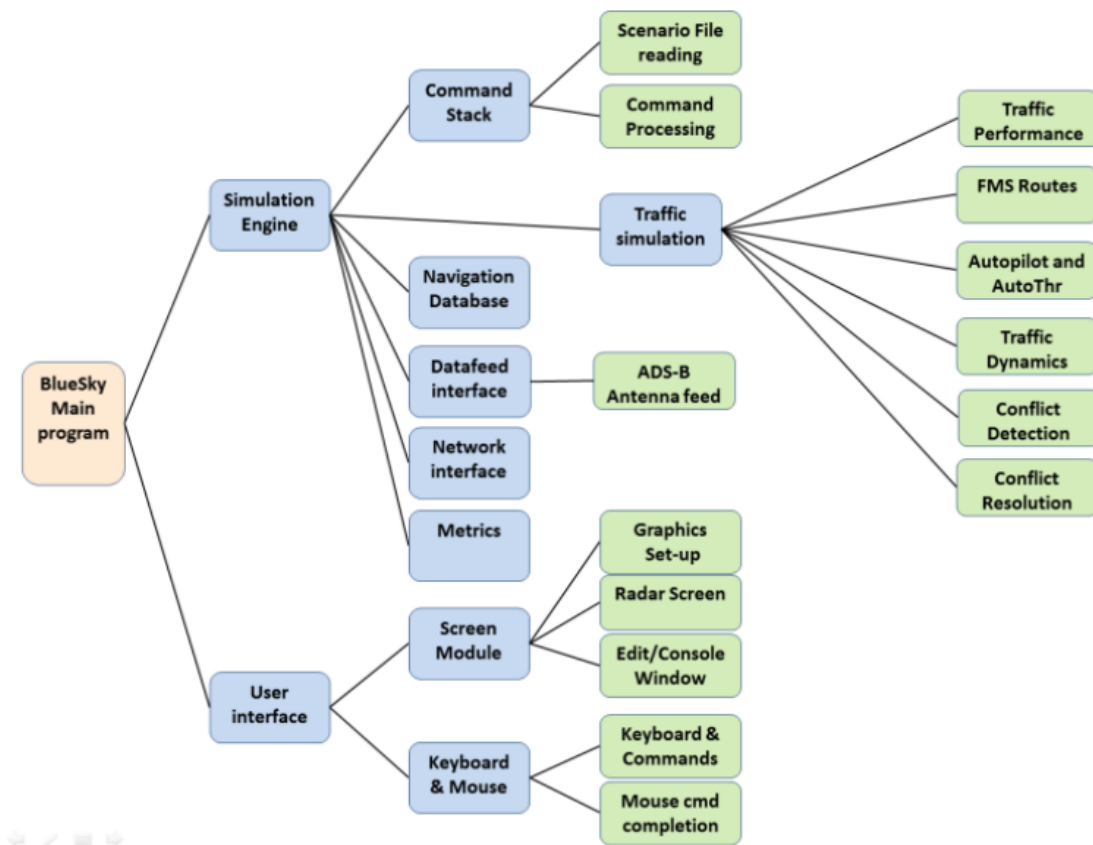


7.3. BlueSky – The Open Air Traffic Simulator

A BlueSky egy magas szintű programozási nyelv használatával készült (Python). Egyszerű szerkezeti felépítése elősegíti a kód bővíthetőségét. [18] A BlueSky célja egy olyan eszköz biztosítása, amivel bárki képes a légi forgalmat vizualizálni, elemezni, szimulálni megkötések, engedélyek és korlátozások nélkül.

7.3.1. A BlueSky moduláris szerkezete

A 10. ábra azt mutatja, hogy mely modulok különböztethetők meg a BlueSky legmagasabb szintjén. [17] A BlueSky moduláris felépítése lehetővé teszi a funkciók egyszerű bővítését anélkül, hogy szükséges lenne a forráskód teljes megértése. A két modul, amelyek a program végrehajtását irányítják, a Szimuláció Vezérlő, valamint a Parancs Verem. A tényleges forgalom szimulációt a Forgalom modul tartalmazza.



12. ábra
BlueSky modulok áttekintése



7.3.2. Szimuláció Vezérlő

A szimuláció vezérlő modul vezérli az ütemezést a fő szimulációs ciklus és a szimulációs mód között. Inicializálja a többi modult, egy vagy több számban. Ennek a modulnak két verziója van: egyszálas és többszálas verzió. A felhasználói felület választásától függően a megfelelő verzió kerül meghívásra a fő programból. A felhasználó választhat interaktív módon, vagy a konfigurációs fájl szerkesztésével vagy törlésével, hogy a klasszikus vagy a haladó felhasználói felületet szeretné használni.

7.3.3. Parancs Verem

A parancs verem modul a szimulációs parancsok elemzésére szolgál. Úgy tekinthető, mint a BlueSky program központi feldolgozó egysége. A verem megkapja a felhasználó által beírt parancssorokat a grafikus felhasználói felület konzolablakát használva. Továbbá más modulok is küldhetnek parancsokat a veremnek a stack metódus meghívásán keresztül. A fogadott parancsok FIFO elven kerülnek feldolgozásra a veremfrissítési ciklus során.

A parancsok egyszerű szöveges karakterláncok. A szöveges karakterláncok sokoldalúsága azt jelenti, hogy a különböző platformok és programok közötti kommunikáció nincs akadályozva interfész vagy adatstruktúra definíciók által. A parancsok a TrafScript nevű ATM szcenárió-vezérlő nyelvet használják.

7.3.4. Forgalom modul

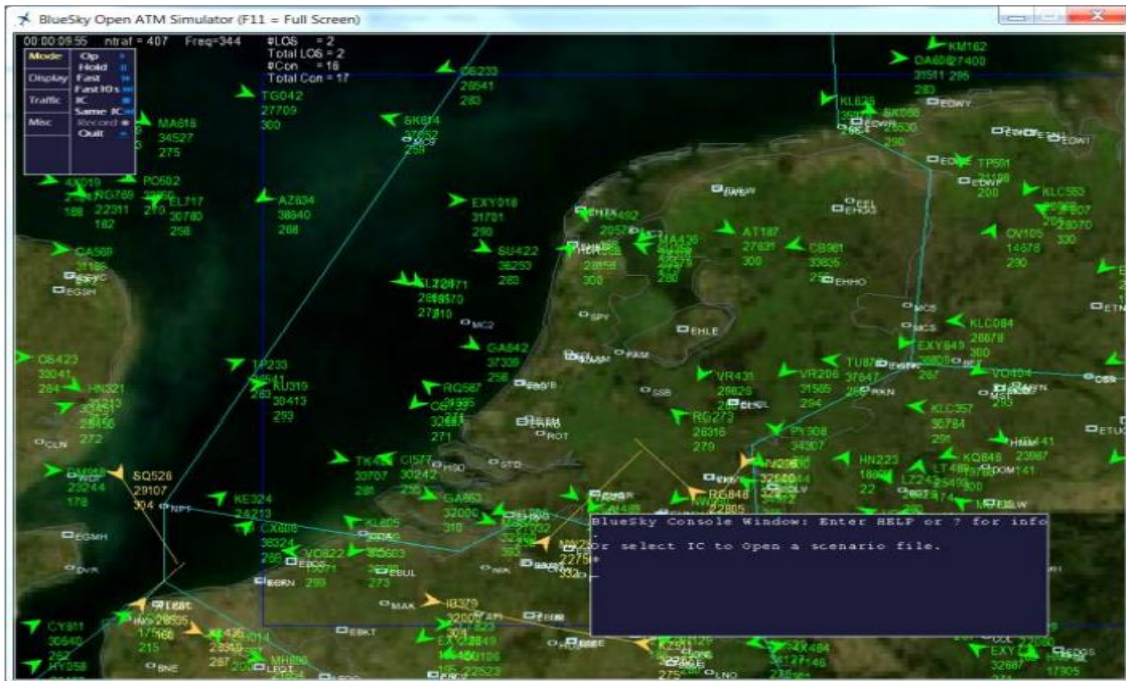
Ez az objektum a tényleges szimulált forgalomhoz kapcsolódó adatokat tartalmazza: az összes repülőgép állapota, teljesítményadatbázis, valamint a navigációs adatbázis.

7.3.5. Felhasználói felület

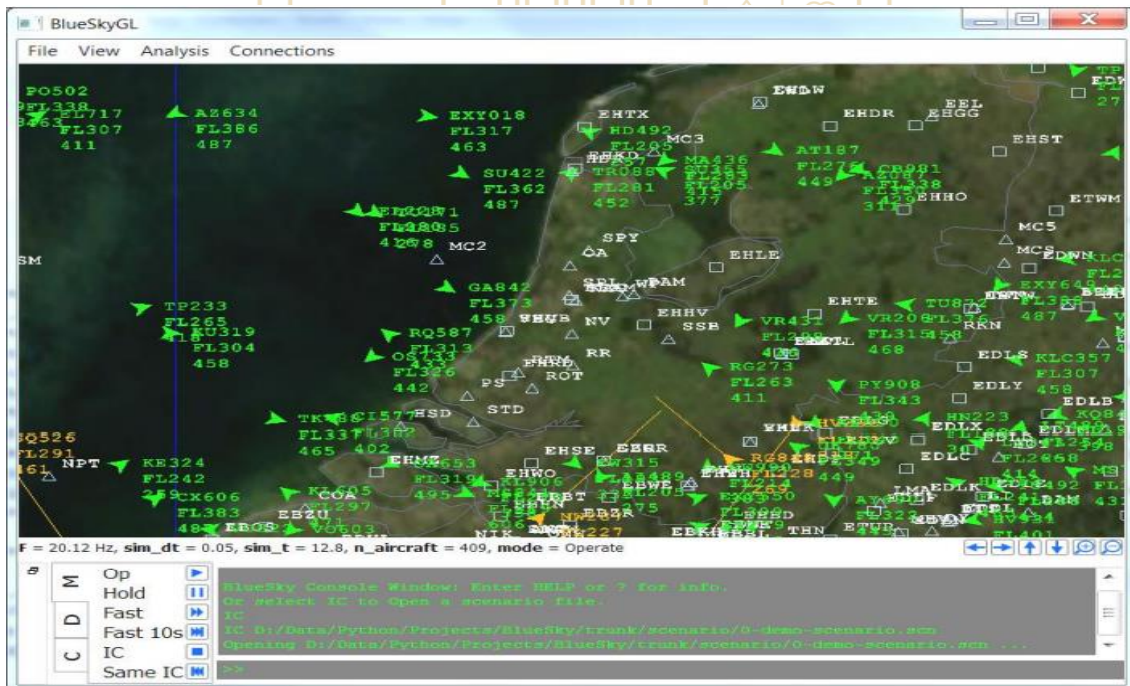
A felhasználói felületet több ablakból áll, mint például a térkép a repülőgép forgalommal, a konzol ablak, amelyben a parancsok szerkeszthetők, a menük stb. Ez a modul a billentyűzet, egér bemeneteit is feldolgozza. A BlueSky két felhasználói felülettel rendelkezik, klasszikus, valamint haladó felhasználói felülettel (lásd 11. és 12. ábra)



7.3.6. Felhasználói felületek



13. ábra
Klasszikus felhasználói felület



14. ábra
Haladó felhasználói felület



7.3.7. TrafScript nyelv

A TrafScript-nek nevezett nyelv segítségével történik a kommunikáció a BlueSky és az ágens között. Röviden ismertetem a következő táblázatban.

Command	Meaning
TrafScript	
CRE acid,type,lat,lon,hdg,alt,spd	Create an aircraft
HDG acid,hdg	Given an aircraft a heading command (use heading in degrees)
LEFT acid,hdg	Given an aircraft a 'turn left' command (use target heading in degrees)
RIGHT acid,hdg	Given an aircraft a 'turn right' command (use target heading in degrees)
ALT acid,alt	Altitude command for autopilot, accepts feet ('30000') or flight levels ('FL300')
SPD acid, spd	Speed command, accepts Mach numbers (speeds <1 optionally prefix by an M) and CAS in kts
DEST acid, airport/lat,lon	Add a destination as the end waypoint of the route
ORIG acid, airport/lat,lon	Add an origin as the begin of the route
ADDWPT acid,wpid,[alt],[spd],[afterwp],[rta]	Add waypoint to route in Flight Management System (FMS), optionally
DELWPT acid, wpid	Delete a waypoint from the FMS route
LNAV acid/*,ON/OFF	Switch lateral navigation mode of FMS on or off
VNAV acid/*,ON/OFF	Switch vertical navigation mode of FMS on or off
DIST lat,lon,lat,lon	Calculate the distance between two positions (e.g. with mouse)
MOVE acid,lat,lon,[alt]	Reposition aircraft
DEL acid	Delete an aircraft
MCRE n,type/*,alt/*	Create n aircraft in area visible on display, optionally for specific type or altitude
MDEL lat,lon,lat,lon	Delete all aircraft within given area, specified by two corners
Simulation Control commands	
IC scenfile	Reset simulation and optionally load a new scenario file
SAVEIC scenfil	Save current situation as the start of a new simulation in scenario file
HOLD	Pause simulation
OP	Goto to OPeration mode: resumes simulation
FF [dt]	Fastforward mode optional time duration in simulated time
When an aircraft id is used as argument, it is allowed to switch it with the command, So: ALT KL204,FL070 is synonymous with KL204 ALT FL070	
Many arguments such as positions, aircraft, headings, waypoints etc can be clicked into the edit window	
When an argument is left blank, you can use commas to add later arguments	

15. ábra

A TrafScript és a Szimulációvezérlő parancsok magyarázata

Minden parancssor a következő szerkezettel rendelkezik:
Command [separator] **argument1** [separator] **argument2**



8. Tervezés

8.1. Légiforgalom szimulátor

Az elérhető légiforgalom szimulátorok közül az MI tesztelésére széles körben elterjedt BlueSky keretrendszert választottam, mert egyszerű a repülőgépek és a hozzájuk kapcsolódó tulajdonságok létrehozása, lehetővé teszi a valós légiforgalmi események generálását (pl.: Turbulencia) és a hasonló problémákat megoldó kutatók nagy része használja ezt a szimulátort.

8.2. MT keretrendszer

Az elérhető MT keretrendszerek közül a Stable-Baselines3 [19] keretrendszert használtam fel, amely nyílt forráskódú mély MT algoritmusok implementációját biztosítja Pythonban. Az algoritmusok egy konzisztens interfészt követnek. A kiterjedt dokumentáció egyszerűvé teszi a különböző MT algoritmusok tanítását, összehasonlítását.

8.3. Open AI Gym

Ahhoz, hogy a tanulási folyamat hatékony legyen, egy olyan környezet definiálása szükséges, amely lehetővé teszi a kapcsolatot a MT algoritmusok és a szimulátor között. Ez a környezet biztosítja az interfészt, amely révén az MI beavatkozhat a szimulációs környezetbe. [20]Az Open AI Gym környezet számos MT keretrendszerrel, így az általam választott Stable-Baselines3-mal is kompatibilis, ezáltal biztosítva a hatékony integrációt a környezet és az MT keretrendszer között.

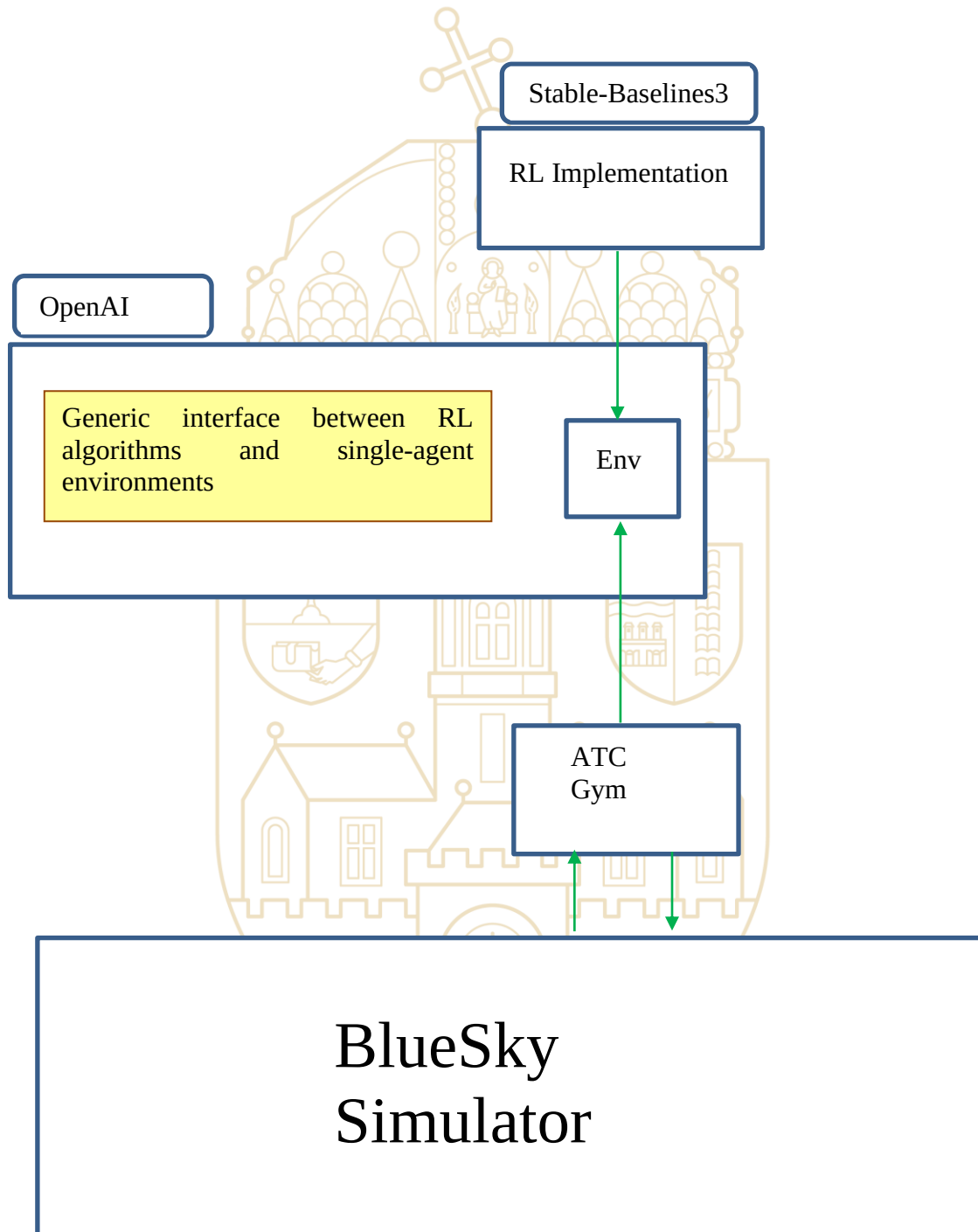
8.4. Programnyelv

A választott programnyelv a Python, mivel az említett keretrendszerek mindegyike Python nyelven íródott. Ennek köszönhetően egyszerűen összehangolhatóak a különböző komponensek.



9. Egy-ágensű architektúra

Ahhoz, hogy el tudjam érni a célokat és sikeresen meg tudjam oldani a feladatot, szükséges volt egyszerűsíteni a problémát először egyetlen ágensre vonatkozó problémává. Ennek érdekében a repülőgépek közül egyet kiválasztottam, amelyre a tanulás és a szimuláció során a figyelem összpontosul.





10. Egy-ágensű probléma megfogalmazása

10.1. Célkitűzés

[1]A cél a biztonságos vertikális távolság fenntartása az ágens és az N-legközelebbi repülőgép között úgy, hogy az ágens magasságváltoztatási tanácsokat kap a Mesterséges Intelligenciától. A jelenlegi problémátérben a repülőgépek három különböző irányból haladnak a cél felé úgy, hogy az útvonalai a repülés során keresztezik egymást.

10.2. Egy-ágensű implementáció

A program implementálása során Marc Brittain és Peng Wei megoldását vettem alapul [1].

Míg a kutatók a repülőgépek közötti biztonságos horizontális szeparációt tűzték ki célul úgy, hogy az MI sebességváltoztatási tanácsokkal látja el az ágenseket egy előre definiált három elemű sebességkonstansokat tartalmazó akcióteréből, addig én a repülőgépek közötti biztonságos vertikális szeparáció megoldását szándékoztam teljesíteni úgy, hogy az MI magasságváltoztatási tanácsokkal látja el az adott ágens.

10.3. Szimulációs beállítások

Minden szimuláció futtatás során előre definiált maximális repülőgépek száma került meghatározásra. Ez lehetővé teszi a szimulációs futtatások összehasonlítható teljesítményét és a modell végső teljesítményének kiértékelését. A BlueSky-ban minden egyes típusú repülőgép különböző sebességtartománnyal rendelkezik, ezért az összes repülőgépet ugyanolyan A320 típusú repülőgépként hoztam létre. Beállításuk során előírtam, hogy nem térhetnek el a repülőgépek az eredeti útvonalaiktól. A problémátér végső beállítása az egyes repülőgépek kezdeti magasságának meghatározása. Ezt 26000 és 28000 láb között véletlenszerűen határoztam meg.



10.4. A BlueSky kiterjesztése saját plug-in-nal

Lehetőség van a BlueSky [21] keretrendszer kibővíteni saját forráskóddal, ami lehetővé teszi a közvetlen hozzáférést a „Traffic”, „Simulation”, „Screen” és „Navdb” szimulációs objektumokhoz. Emellett elérhetővé válik a „Stack” modul is, amely lehetővé teszi a szimulációval való kommunikációt. A „Tools” modul is elérhetővé válik, így hozzáférhetünk a különböző BlueSky funkciókhoz, mint például az aerodinamikai vagy földrajzi funkciókhoz, adatrögzítéshez és még sok más funkcióhoz.

A BlueSky plug-in-ként ismeri fel azt a Python szkriptet, amelynek van egy `init_plugin()` metódusa, valamint definiál és visszatér egy konfigurációs szótárral. A szótárnak tartalmaznia kell a plug-in nevét és típusát. További opcionális paraméterek megadására is van lehetőség, például, hogy milyen gyakran történjen frissítés a szimulációban, illetve mely metódust hívja meg a frissítés során.

10.5. Open AI Gym környezet létrehozása

Miután sikerült parancsokkal vezérelnem a szimulátort, ezt követően szükséges volt létrehozni egy Open AI Gym specifikus környezetet, ugyanis a Stable-baselines3 MT keretrendszer szükséges feltétele, hogy a definiált környezet az Open AI Gym Environment osztályából származzon le.

A környezet definiálása során az `Ösosztály` kötelező változói és metódusai mellett a definiált problémához szükséges változók és metódusok kerültek implementálására [22].

Ösosztály kötelező változói:

- `action_space`, `observation_space`

Ösosztály kötelező metódusai:

- `step(action)`, `reset()`

Ösosztály opcionális metódusai:

- `render()`, `close()`



10.5.1. Egy-ágensű állapotér

Az állapotér az ágens számára szükséges információkat tartalmazza a döntéshozatalhoz. Ez magában foglalja az N-legközelebbi repülőgépek állapotinformációit is. Az ágens állapotát a repülőgép és a cél közötti távolság, a repülőgép magassága, irányszöge, vertikális sebessége, valamint az ágens és az N-legközelebbi repülőgépek közötti távolság határozza meg. Az állapotér tehát az összes lehetséges állapotot tartalmazza, amelyeket az ágensnek figyelembe kell vennie a döntéshozatal során.

```
S = {  
  'agent': [distance_to_goal, actual_FL_level, heading, vertical_speed, distance_to_agent],  
  'intruder-1': [distance_to_goal, actual_FL_level, heading, vertical_speed, distance_to_agent],  
  'intruder-2': [distance_to_goal, actual_FL_level, heading, vertical_speed, distance_to_agent]  
}
```

16. ábra
Egy-ágensű állapotér

10.5.2. Egy-ágensű művelettér

A művelettér a lehetséges cselekvések halmazát foglalja magába, amelyek az adott probléma megoldásához szükségesek. Az ágens a szimuláció során 12 másodpercenként dönti el, hogy a magasságát mely értékre szeretné állítani. A művelettér a következők szerint lett meghatározva:

$A = [25000, 28000, 31000]$

17. ábra
Egy-ágensű művelettér



10.5.3. Egy-ágensű terminált állapot

Az epizód véget ér, amikor az ágens sikeresen kilép a szektorból, vagy ha összeütközik bármely más repülőgéppel.

```
def check_that_ac_should_terminated(self, _idx):
    terminal = False
    terminal_type = 0

    distance, v_separation = self.nearest_ac(self.get_distance_matrix_ac(), _idx)
    goal_d = self.dist_goal(_idx)

    if distance <= 1 and v_separation / ft < 1500:
        terminal = True
        terminal_type = 1
    if goal_d < 5 and terminal == False:
        terminal = True
        terminal_type = 2

    return terminal, terminal_type
```



18. ábra

Egy-ágensű terminálódás

10.5.4. Egy-ágensű step metódus

A step() [22] metódus tartalmazza a környezet logikájának nagy részét. Paraméterként vár egy műveletet, kiszámolja a környezet állapotát a művelet alkalmazása után, majd visszaad egy négy elemű objektumot (állapottér, jutalom, terminálódás, infó).

A metódus megvizsgálja, hogy az ágens terminálódott-e, amennyiben nem, akkor végrehajtja a paraméterként kapott műveletet. Ezt követően kiszámolja az ágens és a legközelebbi repülőgép közötti horizontális és vertikális távolságot. Ennek segítségével meghatározza az adott iterációban lévő jutalmat és frissíti a szimulációt.

```
def step(self, action):
    agent_treminated, agent_termination_type = self.delete_if_terminated()
    self.done = (len(traf.id) == 0 or agent_treminated)
    if not self.done:
        stack.stack('ALT {}, {}'.format(agent_acid, self.possible_alts[action]))
        distance, v_separation = self.nearest_ac(self.get_distance_matrix_ac(), self.current_agent_idx)
        self.reward = self.compute_reward(distance, v_separation, self.prev_v_separation)
        self.prev_v_separation = v_separation

    self.info = {}

    if self.train_mode:
        Timer.update_timers()
        bs.sim.update()
    if not self.done:
        self.state = self.get_agent_and_nearest_ac_intruders_states(self.get_distance_matrix_ac(), self.current_agent_idx)

    return self.state, self.reward, self.done, self.info
```

19. ábra

Egy-ágensű step metódus



10.5.5. Egy-ágensű jutalom függvény

A jutalom függvény [23] egy olyan matematikai függvény, amely jutalmazással és büntetéssel közli az ágenssel, mi helyes és mi helytelen.

A jutalom függvény definiálása során két különböző ágot vizsgáltam.

- Amikor az ágens és a hozzá legközelebbi repülőgép távolsága kisebb, mint 12 tengeri mérföld
- Amikor az ágens és a hozzá legközelebbi repülőgép távolsága nagyobb, mint 12 tengeri mérföld

```
if distance < 12
    if v_separation >= 2000:
        reward = 5
    else if v_separation > prev_v_separation:
        reward = 2
    else:
        reward = -1
else:
    reward = 0
```

20. ábra
Egy-ágensű jutalom függvény

ahol a distance az ágens és a hozzá legközelebbi repülőgép közötti távolság tengeri mérföldben, v_separation az ágens és a hozzá legközelebbi repülőgép közötti vertikális szeparáció, prev_v_separation az előző iterációban lévő ágens és a hozzá legközelebbi repülőgép közötti vertikális szeparáció.

10.5.6. Egy-ágensű reset metódus

A reset() [22] metódus alaphelyzetbe állítja a környezetet, és visszaállítja a környezeti változók kezdőértékeit. Ebben az esetben újra létrehozza a repülőgépeket és visszatér az állapotterrel.

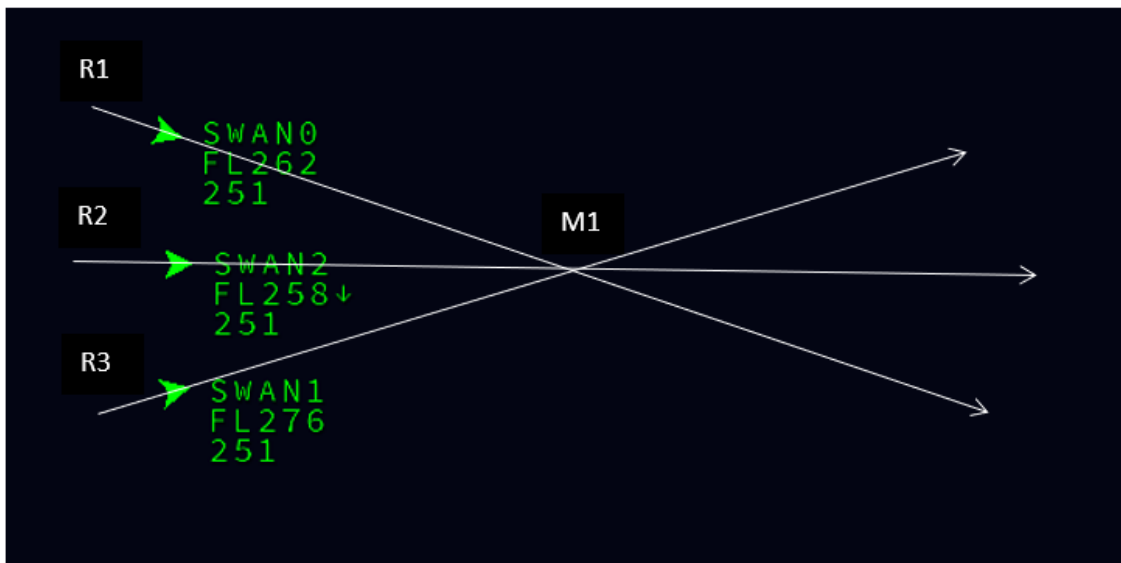
```
def reset(self):
    self.active_ac = 0
    self.total_ac = 0
    self.ac_routes = np.zeros(self.max_ac)
    self.update_timer = 0
    self.spawn_queue = random.choices(self.times, k=self.positions.shape[0])
    self.done = False
    self.spawn_ac_with_delay()
    if len(traf.id) != 0:
        self.reset_states()
    if self.train_mode:
        bs.sim.update()

    return self._states
```

21. ábra
Egy-ágensű reset metódus

10.6. Egy-ágensű problémátér: Három útvonal egy gócponttal

Ebben a problémátérben három különböző útvonalat definiáltam, ahol az útvonalak egy gócpont felé haladnak. Ez a problémátér igazán bonyolult kihívás az ágens számára, ugyanis szükséges elérnie a biztonságos vertikális szeparációt a gócpont elérését megelőzően, majd tartani a gócpontból való kilépést követően.



22. ábra
Egy-ágensű problémátér

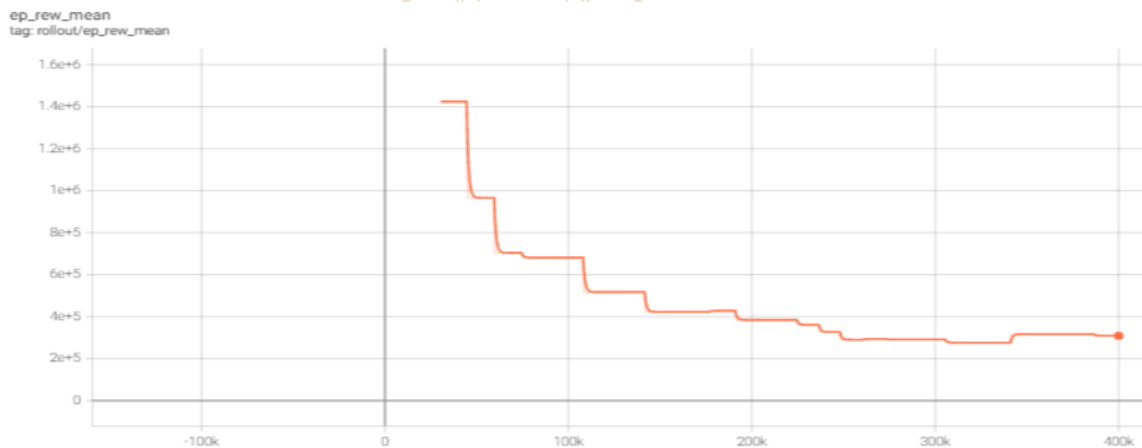
Az epizód akkor ér véget, ha az ágens sikeresen elhagyja a szektort, vagy ha összeütközik más repülőgéppel. Mivel jelenleg csak az adott ágens viselkedését figyelembe véve oldom meg a problémát, ezért a másik két repülőgép összeütközése nem eredményezi az epizód végét. Az optimális megoldás az, ha az ágens nem ütközik össze más repülőgépekkel.



10.7. Egy-ágensű algoritmus teljesítménye

A Stable-Baselines3 MT keretrendszer megírt algoritmusai közül a PPO algoritmussal tanítottam az ágens, amely kedvező eredményt nyújtott a tanítási folyamat során. A paramétereket az alapvető értékeken hagytam.

Az MI ágens 400 000 lépésen keresztül volt tanítva. Az ágens teljesítménye folyamatosan csökkent, ahogy az a mellékelt ábrán látható, de a tanítás végére is pozitív jutalmat kapott. Ezt követően teszteltem az ágens teljesítményét a problématerben, ahol az összeütközések száma 0 volt, ami azt jelenti, hogy az ágens minden esetben képes volt elkerülni a más repülőgépekkel való ütközést.



23. ábra
Egy-ágensű epizódonként átlag jutalom

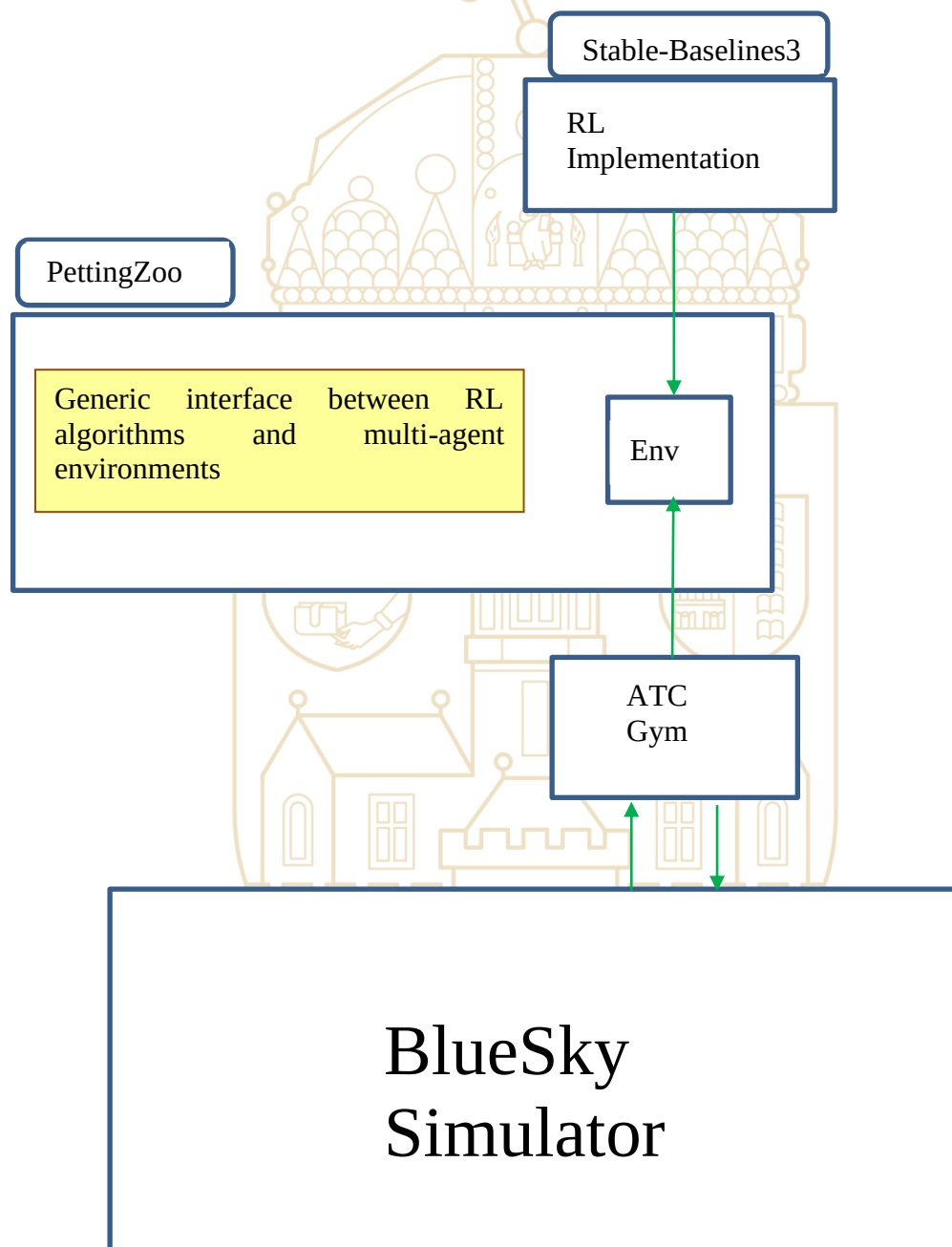


24. ábra
Egy-ágensű átlag epizód hossz



11. Több-ágensű architektúra

Miután sikeresen megoldottam az egy-ágensű problémát, most már az eredeti célomra összpontosítok, hogy az összes repülőgép, mint MT ágens részt vegyen a tanításban. Azonban ehhez fontos volt, hogy az architektúrát lényegesen átalakítsam, mivel az Open AI Gym nem támogatja alapból a több-ágensű környezet implementálását, és a Stable-Baselines3 sem alkalmas több-ágensű tanításra. Így bevezetésre került a PettingZoo [24], amely egy nyílt forráskódú Python könyvtár és API, melyet a több-ágensű rendszerekkel kapcsolatos kutatások támogatására hoztak létre és a Gym több-ágensű verziójaként szolgál.





12. Több-ágensű probléma megfogalmazása

12.1. Célkitűzés

[1]A cél továbbra is a biztonságos vertikális távolság fenntartása az adott ágens és az N-legközelebbi repülőgép között. Azonban, míg korábban csak egy repülőgép vett részt tanításban, és egy repülőgép irányítását végezte az MT, mostantól az összes repülőgép részt vesz a tanításban, így az adott repülőgép cselekedetei az N-legközelebbi repülőgépek cselekedeteitől is függenek. A jelenlegi problématerben a repülőgépek három különböző irányból haladnak a cél felé úgy, hogy az útvonalaik a repülés során egybetorkollanak.

12.2. Több-ágensű implementáció

Amennyiben a probléma definiálása során több ágensre összpontosul a figyelem, akkor a tanításhoz szükséges változók dimenziószámának egyeznie kell a tanítani kívánt ágensek számával.

12.3. PettingZoo Parallel Environment

Az Open AI Gym Environment osztálya helyett a PettingZoo Parallel Environment osztályát volt szükséges paraméterként átadnom a definiált környezetnek. A Parallel Environment minden step metódus hívás során egyszerre lépteti a szimulációban résztvevő ágenseket.

A PettingZoo állapot- és művelettér egy szótár, melynek kulcsai az ágensek azonosítói.

12.3.1. Több-ágensű állapotter

Több-ágensű állapotter definiálása során az ágens állapotát a repülőgép és a hozzá legközelebbi repülőgép közötti távolság, a vertikális szeparáció, az irányszögbeli különbség, a legközelebbi repülőgép vertikális sebessége, valamint a legközelebbi repülőgép előző művelete határozza meg.

```
S = {  
    agent: {  
        'intruder-traf': [distance_to_agent, vertical_separation, heading_difference, vertical_speed],  
        'intruder-act': previous_action  
    } for agent in self.possible_agents  
}
```

25. ábra
Több-ágensű állapotter

12.3.2. Több-ágensű művelettér

```
A = {agent: [25000, 28000, 31000] for agent in possible_agents}
```

26. ábra
Több-ágensű művelettér



12.3.3. Több-ágensű terminált állapot

Az epizód véget ér, amikor az ágensek sikeresen kilépnek a szektorból, vagy ha bármely két repülőgép összeütközik egymással.

12.3.4. Több-ágensű step metódus

A metódus megvizsgálja, hogy az ágensek terminálódtak-e, amennyiben igen, akkor bünteti az összeütköző repülőgépeket, ellenkező esetben a paraméterként kapott művelet szótáron végig iterál, melynek kulcsai megegyeznek az ágensek azonosítóival, és értékei az adott ágenshez tartozó művelettel. Ezt követően kiszámolja az ágens és a legközelebbi repülőgép közötti horizontális és vertikális távolságot. Ennek segítségével meghatározza az adott iterációban lévő jutalmat, végrehajtja az adott ágenshez tartozó műveletet és frissíti a szimulációt.

```
def step(self, action_dict):
    terminal, terminal_type, agent_term, nearest_agent_term = self.delete_if_terminated()
    self.truncations = self.terminations
    self.done = (len(traf.id) == 0 or any(self.terminations.values()))
    if not self.done:
        for agent, action in action_dict.items():
            ac_idx = int(agent)
            distance, v_separation, hdg_diff, nearest_agent = self.nearest_ac(self.get_distance_matrix_ac(), ac_idx)
            nearest_ac_vs = traf.vs[int(nearest_agent)]

            if distance > 10:
                self.rewards[agent] = 0
            elif abs(v_separation) > 2000:
                if action == self.agents_prev_action[agent]:
                    self.rewards[agent] = 5
                else:
                    self.rewards[agent] = 0
            elif (action == action_dict[nearest_agent]):
                self.rewards[agent] = -1
            elif v_separation > 0 and action > action_dict[nearest_agent]:
                self.rewards[agent] = 0.9*(v_separation/2000)
            elif v_separation < 0 and action < action_dict[nearest_agent]:
                self.rewards[agent] = 0.9*(v_separation/2000)
            else:
                self.rewards[agent] = -1 + 0.9*(v_separation / 2000)

            self.agents_prev_action[agent] = action

            self._states[agent]["intruder-traf"] = np.array([distance, v_separation, hdg_diff, nearest_ac_vs], dtype=np.float32)
            self._states[agent]["intruder-act"] = action_dict[nearest_agent]

            stack.stack('ALT { }, { }'.format(traf.id[ac_idx], self.possible_alts[action]))

        if self.train_mode:
            Timer.update_timers()
            bs.sim.update()

    if terminal:
        if terminal_type == 1:
            self.rewards[agent_term] = -1000
            self.rewards[nearest_agent_term] = -1000
        if terminal_type == 2:
            self.rewards = {agent: 1000 for agent in self.possible_agents}

    self.infos = {agent: { } for agent in self.possible_agents}

    return self._states, self.rewards, self.terminations, self.truncations, self.infos
```



12.3.5. Több-ágensű jutalom függvény

A jutalom függvény definiálása során a célom az volt, hogy ha az ágensek közötti horizontális távolság nem biztonságos, akkor a magasságukat úgy módosítsák, hogy az meghaladja a két repülőgép közötti minimum megengedett vertikális távolságot. Amennyiben ez nem teljesül, viszont a repülőgépek távolodnak egymástól, akkor súlyozva jutalmaztam az ágenseket úgy, hogy annál nagyobb jutalmat kapnak, minél távolabb kerülnek egymástól. Ha a repülőgépek közelednek egymáshoz, akkor súlyozva kapnak büntetést, a büntetés mértéke pedig a távolság alapján változik. Továbbá, ha a repülőgépek azonos lábra szeretnék beállítani a magasságukat, ez is büntetést von maga után.

```
if distance > 10:
    self.rewards[agent] = 0
elif abs(v_separation) > 2000:
    if action == self.agents_prev_action[agent]:
        self.rewards[agent] = 5
    else:
        self.rewards[agent] = 0
elif (action == action_dict[nearest_agent]):
    self.rewards[agent] = -1
elif v_separation > 0 and action > action_dict[nearest_agent]:
    self.rewards[agent] = 0.9*(v_separation/2000)
elif v_separation < 0 and action < action_dict[nearest_agent]:
    self.rewards[agent] = 0.9*(v_separation/2000)
else:
    self.rewards[agent] = -1 + 0.9*(v_separation / 2000)
if terminal:
    if terminal_type == 1:
        self.rewards[agent_term] = -1000
        self.rewards[nearest_agent_term] = -1000
    if terminal_type == 2:
        self.rewards = {agent: 1000 for agent in self.possible_agents}
```

28. ábra
Több-ágensű jutalom függvény

ahol distance az ágens és a hozzá legközelebbi repülőgép közötti horizontális távolság tengeri mérföldben, v_separation az ágens és a hozzá legközelebbi repülőgép közötti vertikális szeparáció lábban, action az ágens aktuálisan választott művelete, action_dict az ágensekhez tartozó műveletek szótára, terminal egy logikai érték, amely igazat ad vissza, ha véget ér az epizód, terminal_type melynek értéke alapján meghatározható, hogy összeütköztek-e a repülőgépek, vagy sikeresen kiléptek a szektorból.



12.3.6. Több-ágensű reset metódus

```
def reset(self, seed=None, options=None):
    self.active_ac = 0
    self.total_ac = 0
    self.ac_routes = np.zeros(self.max_ac)
    self.update_timer = 0
    self.spawn_queue = random.choices(self.times, k=self.positions.shape[0])
    self.done = False
    self.agents = self.possible_agents[:]
    self.agents_prev_action = {agent: 0 for agent in self.possible_agents}
    self.rewards = {agent: 0 for agent in self.agents}
    self.terminations = {agent: False for agent in self.agents}
    self.truncations = {agent: False for agent in self.agents}
    self.infos = {agent: {} for agent in self.agents}

    self.spawn_ac_with_delay()

    if len(traf.id) != 0:
        self.reset_states()

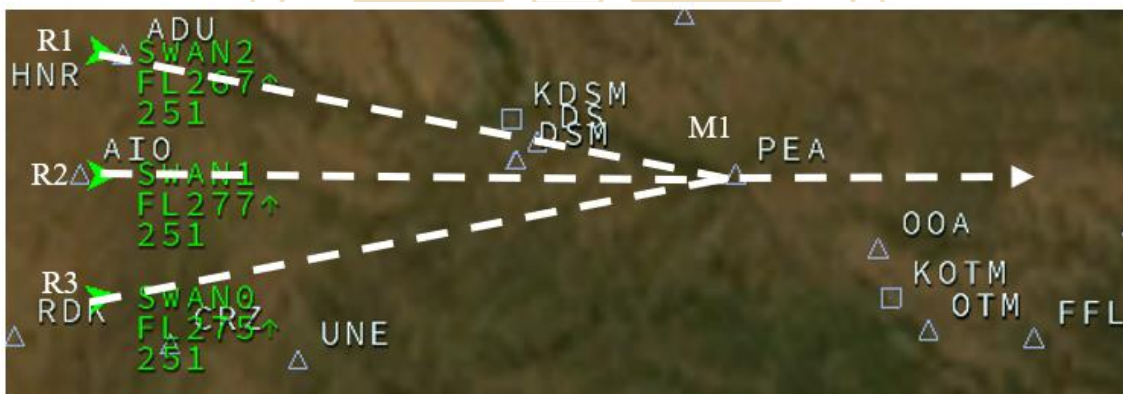
    if self.train_mode:
        bs.sim.update()

    return self._states
```

29. ábra
Több-ágensű reset metódus

12.4. Több-ágensű problémátér: Három útvonal egybetorkollik

Ebben a problémátérben három különböző útvonalat definiáltam, ahol az útvonalak egybetorkollanak, majd egyetlen útvonalként folytatódnak. Ez a problémátér egy jóval bonyolultabb kihívás az ágensek számára, mint az egy-ágensű problémátér, ugyanis az ágenseknek szükséges elérnie a biztonságos vertikális szeparációt a gócpont elérését megelőzően, majd tartani egészen a szektor elhagyásáig.

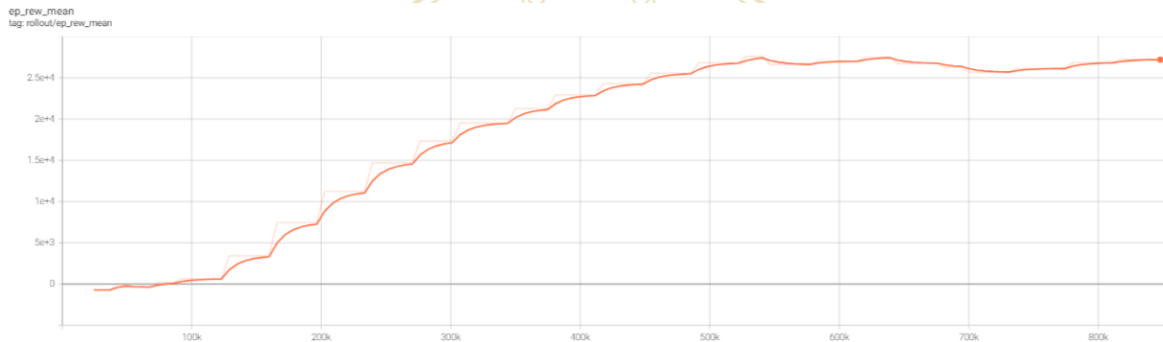


30. ábra
Több-ágensű problémátér

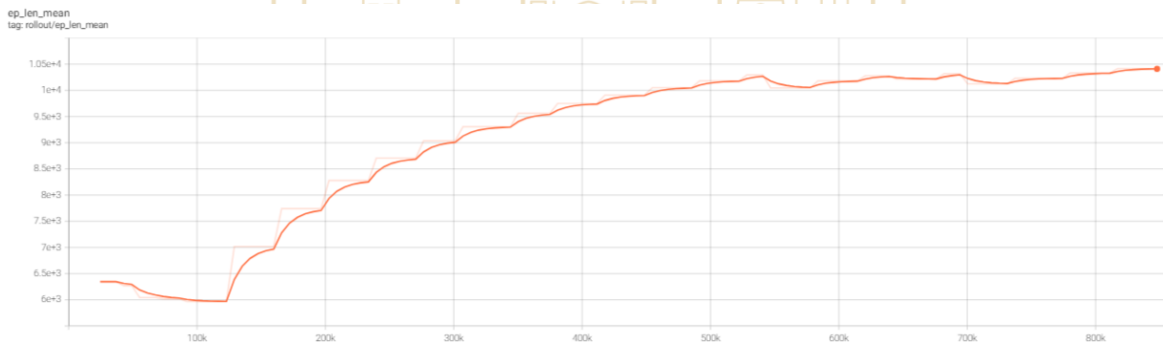


12.5. Több-ágensű algoritmus teljesítménye

A PPO algoritmus több-ágensű tanítás esetén jól működött. A PPO hiperparaméterei közül az entrópia együtthatót az alapértelmezett 0-ról 0.001-re állítottam. Ez a hiperparaméter az ágenseket arra ösztönzi, hogy fedezzék fel a környezetet, ezáltal egy jobb teljesítményű modellt vártam el a tanítás végére. Ahogy az ábrákon is látható az ágensek teljesítménye folyamatosan javult a tanítás folyamán, olyannyira, hogy a kezdeti összeütközések után, a tanítás végére az ágensek sikeresen teljesítették a célként kitűzött feladatot. Ezt követően teszteltem az ágensek teljesítményét a problématerben, ahol az összeütközések száma az egy-ágensű problémához hasonlóan 0 volt, ami azt jelenti, hogy az ágensek minden esetben képesek voltak elkerülni az egymással való ütközést.



31. ábra
Több-ágensű epizódoként átlag jutalom



32. ábra
Több-ágensű átlag epizód hossz

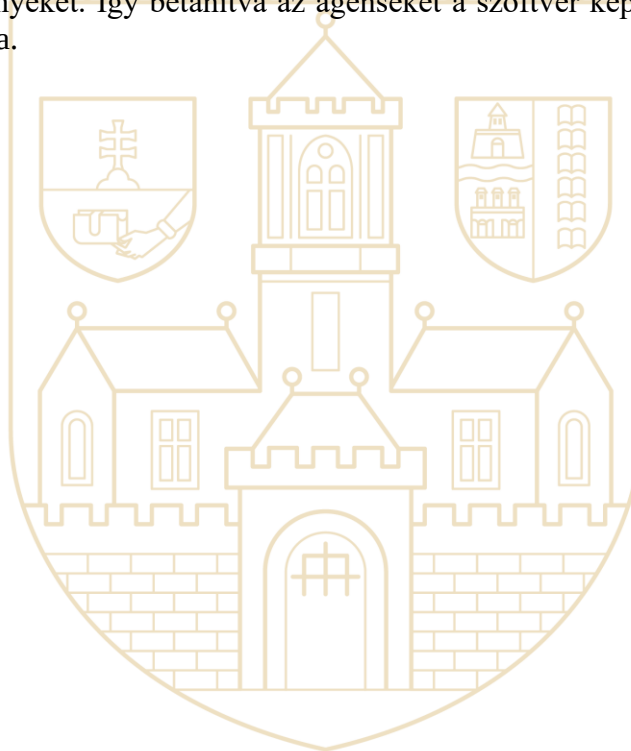


13. Összegzés

Sikerült egy működő szoftvert megalkotni a repülőgépek közötti biztonságos vertikális szeparáció megoldására. A feladat egy több-ágensű megerősítéses tanulás problémaként került meghatározásra, amely során a repülőgépeknek az optimális magasságát volt szükséges meghatározni, annak érdekében, hogy biztonságosan tudjanak közlekedni a légtérben. A megoldás során a Stable-Baselines3 MT keretrendszer algoritmusai közül az Proximal Policy Optimization (PPO) bizonyult optimális választásnak, amely egy-ágensű és több-ágensű tanítás esetén is pozitív átlag jutalmat volt képes produkálni.

A program továbbfejlesztése érdekében ajánlott a különböző MT keretrendszerek összehasonlítása, kipróbálása. Szükséges az alaposabb tanulmányozása az elérhető MT algoritmusok felépítésének, azok hiperparamétereinek. Az algoritmusok és hiperparaméterek megértése segíthet az optimális algoritmus kiválasztásában és a megfelelő értékek beállításában. Megfelelő tanítási lépésszám esetén létrehozható egy optimális MT modell, amely képes a valós légiforgalmi események során javaslatokat tenni a repülőgépek optimális magasságára vonatkozóan.

A BlueSky keretrendszer számos valós légiforgalmi esemény szimulálását teszi lehetővé, például turbulencia generálását, vagy az adott repülőgép aktuális üzemanyagának eltárolását. Lehetőség van a szimuláció során új repülőgépek létrehozására, a szimuláción belül a repülőgépek tulajdonságainak változtatására. Ezek segítségével a probléma oly módon bonyolítható, hogy az ténylegesen tükrözze a valós légiforgalmi eseményeket. Így betanítva az ágenseket a szoftver képes lehet a pilótáknak valós életben a segítségnyújtásra.



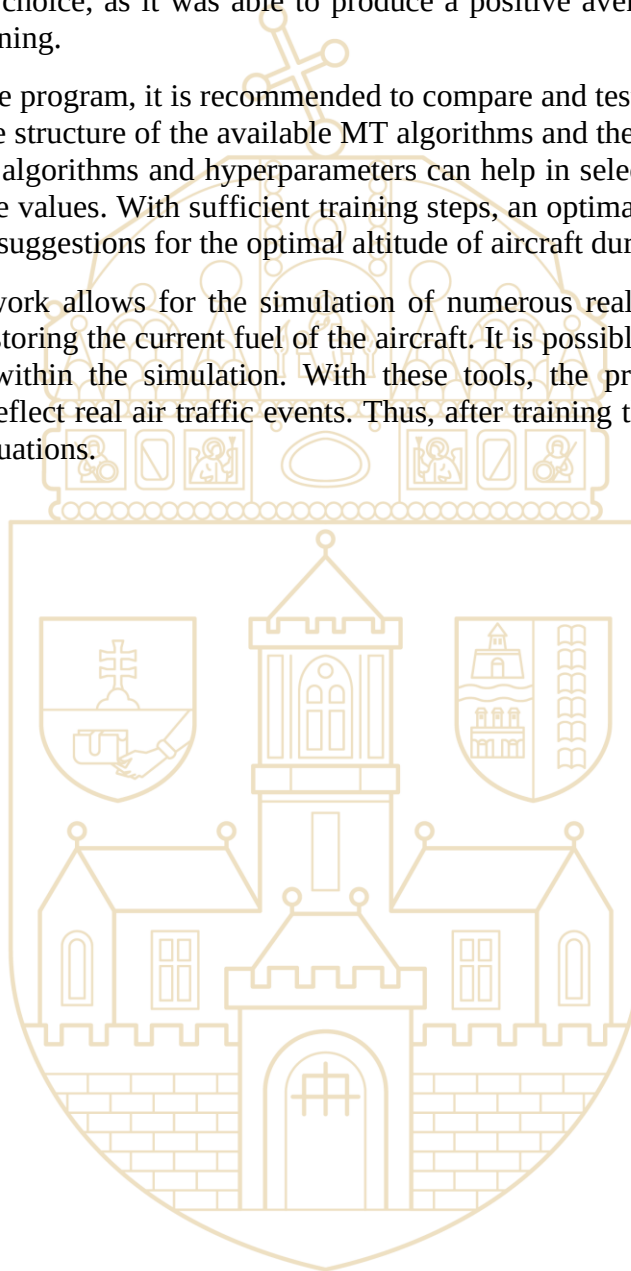


14. Summary

I have successfully developed a working software solution for safe vertical separation between aircraft. The task was defined as a deep multi-agent reinforcement learning problem, where the optimal altitude of aircraft needed to be determined in order to safely navigate the airspace. Among the algorithms of the Stable-Baselines3 MT framework, the Proximal Policy Optimization (PPO) proved to be the optimal choice, as it was able to produce a positive average reward in both single-agent and multi-agent training.

To further develop the program, it is recommended to compare and test different MT frameworks. It is necessary to study the structure of the available MT algorithms and their hyperparameters in more detail. Understanding the algorithms and hyperparameters can help in selecting the optimal algorithm and setting the appropriate values. With sufficient training steps, an optimal MT model can be created that is capable of making suggestions for the optimal altitude of aircraft during real air traffic events.

The BlueSky framework allows for the simulation of numerous real air traffic events, such as turbulence generation or storing the current fuel of the aircraft. It is possible to create new aircraft and change their properties within the simulation. With these tools, the problem can be made more complex and accurately reflect real air traffic events. Thus, after training the agents, the software can assist pilots in real-life situations.





15. Irodalomjegyzék

- [1] B. Marc és W. Peng, „Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement,” 2019.
- [2] M. John, „WHAT IS ARTIFICIAL INTELLIGENCE?,” 2007.
- [3] T. Vas, B. Békési, J. Sári, K. Kele, M. Magyar és R. Szabó, „A mesterséges intelligencia alkalmazhatósága,” 2021.
- [4] B. Jatin, „Applications of Artificial Intelligence & Associated Technologies,” 2016.
- [5] A. Rockwell, „The History of Artificial Intelligence,” 2017.
- [6] W. Joseph, „ELIZA—a computer program for the study of natural language communication between man and machine,” 1966.
- [7] „Wikipédia,” [Online]. Available: <https://hu.wikipedia.org/wiki/Heurisztika>.
- [8] J. Mark, „Genetikus algoritmusok,” 1999.
- [9] P. Riccardo és K. John R., „Genetic programming,” 2005.
- [10] P. K. Leslie, L. L. Michael és W. M. Andrew, „Reinforcement Learning: A Survey,” 1996.
- [11] L. Yuxi, „DEEP REINFORCEMENT LEARNING: AN OVERVIEW,” 2017.
- [12] Á. Szilágyi, „Mesterséges intelligencia segítségével növelné pontosságát a British Airways,” AIRPortal, 2019.
- [13] A. Ambrus, „Mesterséges intelligencia számolja ki a várakozási időt Pittsburgh repterén,” AIRPortal, 2019.
- [14] A. Ambrus, „Beszélő robotfej segít eligazodni a frankfurti reptéren,” AIRPortal, 2019.
- [15] M. Mukesh, H. Norlaila , V. Nitha C és S. Biju T , „EMBEDDING MACHINE LEARNING IN AIR TRAFFIC CONTROL SYSTEMS TO GENERATE EFFECTIVE ROUTE PLANS FOR AIRCRAFTS IN ORDER TO AVOID COLLISIONS,” *Journal of Theoretical and Applied Information Technology*, 2019.
- [16] G. Supriyo, L. Sean, H. L. Shiao, W. Laura és P. Hasan, „A Deep Ensemble Method for Multi-Agent Reinforcement Learning,” in *Proceedings of the Thirty-First International Conference on Automated Planning and Scheduling*, 2021.



- [17] M. H. Jacco és E. Joost, „BlueSky ATC Simulator Project: an Open Data and Open Source Approach,” 2016.
- [18] „BlueSky Homepage,” [Online]. Available: <http://homepage.tudelft.nl/7p97s/bluesky/>.
- [19] Ashley Hill, Antonin Raffin, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus és Noah Dormann, „Stable-Baselines3: Reliable Reinforcement Learning,” 2021.
- [20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang és W. Zaremba, „OpenAI Gym,” 2016.
- [21] J. Ellerbroek, „GitHub,” 02 2023. [Online]. Available: <https://github.com/TUDeft-CNS-ATM/bluesky/wiki/plugin>.
- [22] A. Open, „Gym Documentation,” [Online]. Available: https://www.gymnasium.dev/content/environment_creation/.
- [23] A. Zhang, „How to Design a Reinforcement Learning Reward Function for a Lunar Lander,” 2021. [Online]. Available: <https://towardsdatascience.com/how-to-design-reinforcement-learning-reward-function-for-a-lunar-lander-562a24c393f6>.
- [24] T. J. K, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. L. Williams, Y. Lokesh és P. Ravi, „PettingZoo: A Standard API for Multi-Agent”.



16. Ábrajegyzék

1. ábra A Mesterséges Intelligencia idővonala	8
2. ábra A gépi tanulás ágai	16
3. ábra A MARL probléma előrehaladása	16
4. ábra Az A2C neurális hálózati architektúrájának illusztrációja	17
5. ábra A Mély Együttes MARL módszer architektúrája	19
6. ábra Az ATC gépi tanulási rendszerének modellje	21
7. ábra Az ATC gépi tanulási rendszerének átdolgozott modellje	21
8. ábra MI architektúra az ATC számára	21
9. ábra Terv kritikus modul	22
10. ábra Cassiopeia projekt	23
11. ábra Elsa projekt	24
12. ábra BlueSky modulok áttekintése	25
13. ábra Klasszikus felhasználói felület	27
14. ábra Haladó felhasználói felület	27
15. ábra A TraFScript és a Szimulációvezérlő parancsok magyarázata	28
16. ábra Egy-ágensű állapotter	33
17. ábra Egy-ágensű művelettér	33
18. ábra Egy-ágensű terminálódás	34
19. ábra Egy-ágensű step metódus	34
20. ábra Egy-ágensű jutalom függvény	35
21. ábra Egy-ágensű reset metódus	35
22. ábra Egy-ágensű problémater	36
23. ábra Egy-ágensű epizódonként átlag jutalom	37
24. ábra Egy-ágensű átlag epizód hossz	37
25. ábra Több-ágensű állapotter	39
26. ábra Több-ágensű művelettér	39
27. ábra Több-ágensű step metódus	40
28. ábra Több-ágensű jutalom függvény	41



29. ábra Több-ágensű reset módszer	42
30. ábra Több-ágensű problémater	42
31. ábra Több-ágensű epizódonként átlag jutalom	43
32. ábra Több-ágensű átlag epizód hossz	43

