ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Seres Richárd Sándor**

2023 Hallgató törzskönyvi száma: **T/006366/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Seres Richárd Sándor**
Törzskönyvi száma: T/006366/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Forgalomirányító rendszer megerősítéses tanulást használó ágens segítségével
Traffic control system with reinforcement learning agent

Intézményi konzulens: Sipos Miklós László
Külső konzulens:

Beadási határidő: 2021. december 15.

A záróvizsga tárgyai: Számítógép architektúrák
Szoftvertervezés és -fejlesztés specializáció

A feladat

A feladat megerősítéses tanulást használó ágens segítségével közlekedési lámpák irányítása egy szimulációs környezetben. Vizsgálja meg, hogy miként működnek az adaptív forgalomirányítási rendszerek, vonjon le következtetést a jellemzőikből. Vizsgálja meg, hogy voltak-e már próbálkozások a közlekedési lámpák irányítására megerősítéses tanulás használatával. Elemezze ezeket a megközelítéseket. Vizsgálja meg a neurális hálózatok működését és tanulmányozza a megerősítéses tanulás területét a probléma szempontjából releváns irányból. Jelölje ki a megfelelő technológiákat a feladat megvalósítására. Implementálja a szimulációs környezetben a megerősítéses tanulást a közlekedési lámpák irányítására. Legyen képes az implementált megoldásának hatékonyságát bemutatni a hagyományos, eddig ismert forgalomirányítókkal szemben. Elemezze, hogy az implementált megoldás milyen hatással van a közlekedésre a vizsgált szimulációkban.

A dolgozatnak tartalmaznia kell:

- a megoldandó feladat pontos leírását és részletes elemzését,
- készítsen magas szintű szerkezeti mintát az alkalmazásról, melyhez vizsgáljon meg hasonló rendszereket,
- az egyes funkciók megvalósításához gyűjtse össze a lehetséges technológiákat,
- tekintse át a lehetséges technológiákat, majd válassza ki a szükségeseket,
- mutassa be a készített rendszer tervét,
- ismertesse a megvalósítás menetét,
- ismertesse a tesztelési tervet, a tesztelés eredményit, a fejlesztés során felmerült problémákat és azok megoldását,
- mutassa be az elkészült szoftver felhasználási lehetőségeit.

ÓBUDAI EGYETEM * Neumann János Informatikai Kar * 1. Budapest

Ph.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2023. december 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

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

Budapest, 2022. 12. 13.

.....

hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar
Neumann Faculty of Informatics

KONZULTÁCIÓS NAPLÓ

Hallgató neve: SERES RICHÁRD SÁNDOR

Neptun kód: [redacted]

Tagozat: NAPPALI

Telefon: [redacted]

Levelezési cím (pl: lakcím): [redacted]

Szakdolgozat címe magyarul:

FORGALOMIRÁNYÍTÓ RENDSZER MEGERŐSÍTÉSES TANULÁST HASZNÁLÓ ÁGENS SEGÍTSÉGÉVEL

Szakdolgozat címe angolul:

TRAFFIC CONTROL SYSTEM WITH REINFORCEMENT LEARNING AGENT

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: ---

Alk.	Dátum	Tartalom	Aláírás
1.	2021.02.10.	Félév teendőinek ismertetése	Sipos S.
2.	2021.03.18.	Hasonló rendszerek elemzése	Sipos S.
3.	2021.04.22.	Irodalomkutatás összegzése, konzekvenciák meghatározása	Sipos S.
4.	2021.05.06.	Tervezés, követelmény specifikáció megfogalmazása	Sipos S.

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat I. (BSc) tantárgy követelményét teljesítette, beszámolóra bocsátható.

Budapest, 2021.05.10.

Sipos S.

Sipos Miklós

intézményi konzulens

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve: SERES RICHÁRD SÁNDOR

Neptun Kód: [redacted]

Tagozat: NAPPALI TAGOZAT

Telefon: [redacted]

Levelezési cím (pl.: lakcím): [redacted]

Szakdolgozat címe magyarul:

FORGALOMIRÁNYÍTÓ RENDSZER MEGERŐSÍTÉSES TANULÁST HASZNÁLÓ ÁGENS SEGÍTSÉGÉVEL

Szakdolgozat címe angolul:

TRAFFIC CONTROL SYSTEM WITH REINFORCEMENT LEARNING AGENT

Intézményi konzulens: SIPOS MIKLÓS LÁSZLÓ

Külső konzulens: -

Alk.	Dátum	Tartalom	Aláírás
1.	2022.10.02.	Fejlesztés tervezett menetének revíziója, szakmai egyeztetés, fejlesztési teendők megkezdése.	
2.	2022.10.30.	Fejlesztés állapotának és haladásának áttekintése. Technikai problémák és módosítások egyeztetése.	
3.	2022.11.20.	Felmerülő problémák egyeztetése és megoldása. Tesztelési fázis megindítása.	
4.	2022.11.30.	Dokumentáció véglegesítése. Leadással kapcsolatos teendők egyeztetése.	

A Konzultációs naplót összesen 4 alkalommal, az egyes konzultációk alkalmával kell láttamoztatni bármelyik konzulenssel.

A hallgató a Szakdolgozat II. tantárgy követelményét teljesítette, védésre bocsátható.

.....

Intézményi konzulens

Budapest, 2022.12.01.

Tartalomjegyzék

1. BEVEZETÉS **5**

2. IRODALOMKUTATÁS **6**

2.1. Hasonló rendszerek elemzése 6

2.1.1. Konvencionális forgalomirányítási módszerek 6

2.1.2. Adaptív forgalomirányítási módszerek 7

2.1.3. IntelliLight 10

2.1.4. A kísérleti környezet 11

2.1.5. A rendszer felépítése 11

2.1.6. Összehasonlított forgalomirányítási módszerek 12

2.1.7. Felhasznált technológia 13

2.1.8. Konklúzió az IntelliLight projektből 13

2.2. Neurális hálózatok 14

2.2.1. Új számítási módszerek 14

2.2.2. Mi is a neurális háló? 14

2.2.3. Milyen esetekben érdemes neurális hálókat alkalmazni? 15

2.2.4. A neurális hálózatok jellemzői 15

2.2.5. Milyen tanulságot tudunk vonni a természetes idegi hálózatokból? . 16

2.2.6. Neuron formális felépítése és működése 18

2.2.7. Aktivációs függvények szerepe 19

2.2.8. Neurális hálózatok topológiája 19

2.2.9. A neurális hálózat tanítása 20

2.2.10. Neurális hálózatok alkalmazásának menete 22

2.2.11. Aktivációs függvények 22

2.3. Megerősítéses tanulás 25

2.3.1. Markov döntési folyamat 25

2.3.2. Optimalizációs algoritmusok 27

2.4. Összegzés 28

3. KÖVETELMÉNY SPECIFIKÁCIÓ **30**

4. TERVEZÉS **31**

4.1. A megerősítéses tanulást használó ágens rendszerterve 31

4.2. A komponensek egymással való interakciói 32

4.3. Használati esetek 32
4.4. A rendszer funkciólistája 32
4.5. Ágens neurális hálózata 34
4.6. Ágens interakciói 35
4.7. Jutalom függvény 35
4.8. Hálózati kapcsolat a közlekedési csomópontokkal 35
4.9. Eredmények nyomonkövetésének a módja 36
4.10. Külső komponensek jellemzése 36

5. FEJLESZTÉS 38
5.1. Kapcsolódás a SUMO szimulátorhoz és a váz létrehozása 38
5.2. Megerősítéses tanulás integrációja a szimulációs környezettel 39
5.3. Komponens alapú megközelítés a fejlesztésben 41
5.4. Hálózati kapcsolódás a közlekedési csomópont vezérlőihez 42
5.5. Állapot tér 43
5.6. Cselekvési tér 44
5.7. Jutalom függvény 45
5.8. A szimuláció felparaméterezése 45
5.9. Statikus forgalomirányító szimulációja 45
5.10. Kísérletek és eredmények 46
5.10.1. Kísérletek 46
5.10.2. Tanított ágens felhasználása 51
5.10.3. Összehasonlítás a statikus módszerrel 52

6. TESZTELÉS 53
6.1. Tesztelés menete a projektben 53

7. ÖSSZEGZÉS ÉS TOVÁBBFEJLESZTÉS 55

ÁBRAJEGYZÉK 56

TÁBLÁZATOK JEGYZÉKE 57

IRODALOMJEGYZÉK 58

8. MELLÉKLETEK 61

ABSZTRAKT

A dolgozat egy mély-megerősítéses tanulást használó ágens rendszertervét és az implementált rendszer eredményit mutatja be közlekedési lámpák adaptív irányítására. Ehhez először bemutatja a releváns témaköröket az irodalomkutatás fejezetben. Itt olvasható a hasonló rendszerek elemzése ahol bemutatja a konvencionális forgalomirányítási módszereket majd pedig egy konkrét megerősítéses tanulást használó projektet vizsgál meg. Ezután a neurális hálózatok témakörét mutatja és az ehhez kapcsolódó fontosabb aspektusokat. Majd a megerősítéses tanulás témakörének a bemutatása következik, ahol a mély-megerősítéses tanuláshoz szükséges ismereteket írja le. A dolgozat ezután tartalmazza a tervezett mély-megerősítéses tanulást használó ágens implementációját közlekedési lámpák adaptív irányítására. Az ehhez szükséges rendszertervet és rendszer felépítést tárgyalja a tervezés rész. A tervezett rendszer implementációját a fejlesztés részben lehet olvasni, ahol felépített rendszer eredményei láthatók.

ABSTRACT

The paper presents the system design of an agent using deep reinforcement learning and the results of the implemented system for adaptive control of traffic lights. For this purpose, it first presents the relevant topics in the literature survey section. Here an analysis of similar systems is presented where conventional traffic control methods are introduced and then a concrete project using reinforcement learning is discussed. Then the topic of neural networks is presented and the most important aspects related to it. Furthermore the topic of reinforcement learning is presented, where the knowledge needed for deep reinforcement learning is described. The thesis then includes the implementation of the proposed deep reinforcement learning agent for adaptive control of traffic lights. The necessary system design and system architecture are discussed in the design section. The implementation of the designed system can be found in the development section, where the results of the built system are shown.

1. BEVEZETÉS

Az elmúlt évszázad végén jellemző volt a nagyvárosokba költözés vidéki területekről. A megnövekedett népesség igényei, pedig a közlekedésben is megmutatkoztak. Ennek következményeképpen megnövekedett az autók száma ezekben a városokban, viszont a közlekedésirányítási infrastruktúra nem változott. Ez a múlt évszázadi rendszer nincs felkészülve ilyen mennyiségű jármű irányítására és ezt mi a mindennapjainkban tapasztalhatjuk.

Gondoljunk bele, hogy a rengeteg autó, amely a dugóban áll a belvárosban délután négykor, milyen mértékben járul hozzá a légszennyezéshez, ezáltal milyen veszélynek teszi ki a belvárosban lakók egészségét. Láthatjuk, hogy az Európai Unió irányelve is a légszennyezés csökkentése a kontinensen, viszont úgy tűnik abba nem gondoltak bele, hogyha modernizálnánk a közlekedési forgalomirányító rendszereinket, akkor nem csak a károsanyag-kibocsátást és a légszennyezést csökkentenénk városainkban, hanem a klímaváltozás elleni küzdelmet is segítené. Ebben a dolgozatban a cél kideríteni, hogy korunk egyik legígéretesebb irányzata, a neurális hálózatok miként tudnának segíteni a közlekedés modernizálásában. Konkrétan a jelzőlámpák irányításának alkalmazására kerül megvizsgálásra a lehetséges szerepük, mély-megerősítéses tanulást használó ágens implementálásával.

2. IRODALOMKUTATÁS

Az alábbi bekezdésben olyan témakörök kerülnek bemutatásra, amelyek a projekt megvalósításához szükségesek. Először a hasonló rendszerek témakörében, különféle forgalomirányító rendszerek kerülnek megvizsgálásra, ezen belül az online illetve offline rendszerek. Részletesebben pedig az adaptív forgalomirányítás olyan aspektusai, amelyek a projekt szempontjából relevánsak.
Ezek után a megerősítéses tanulás egy konkrét közlekedési lámpák irányítására való implementációja kerül bemutatásra. A további irodalomkutatás során a neurális hálók témaköre illetve a megerősítéses tanulás kerül bemutatásra hosszabb terjedelemben.

2.1. Hasonló rendszerek elemzése

A hasonló rendszerek elemzése egy nehezebb téma. A megerősítéses tanulást alkalmazó ágensnek használata a közlekedési lámpák irányítására egy újonnan felvetett ötlet, ezért konkrét elkészített rendszerből nincs sok. Viszont sok tanulmányt lehet megtalálni amelyben valamilyen implementációját vizsgálják az megerősítéses tanulásnak, legyen szó egy ágens, vagy több ágens használatáról, illetve különféle térreprezentációról amelyet felhasználnak a modell létrehozására.
Mivel jelzőlámpa irányításról van szó, ezért elengedhetetlen különféle forgalomirányítási rendszerek megvizsgálása, annak ellenére, hogy nem egy teljes forgalomirányító rendszer és architektúra kerül implementálásra a való világban.

2.1.1. Konvencionális forgalomirányítási módszerek

Az alapvető megközelítés szerint a közlekedési lámpák irányítására, megkülönböztetünk online és offline rendszereket [1]. Az offline rendszerek a mindenki által megfigyelhetett, fix fázistervet használó lámpák. Ezek általában előre meghatározott időtervek szerint váltogatják a fázistervüket. Ezek a lámpák természetesen semmilyen információt nem gyűjtenek a forgalomból, semmilyen online komponenshez nem kapcsolódnak. Ebből szimplán a belső időzítőiktől függően váltogatják a lámpáikat és fázisaikat.
Egy másik, valamivel intelligensebb változata az offline rendszereknek a programválasztós lámpák, ahol már nem csak külön napszakokra bontva léteznek fázistervek, hanem már, valamivel adaptívabb módon a szenzorok segítségével, a forgalom megfigyelése után módosítja a saját fázistervét a lámpa. Természetesen ezek a módszerek is meglehetősen korlátozott adaptivitást eredményeznek, a komolyabb rugalmasság és adaptivitás

eléréséhez online rendszerek alkalmazására van szükség. A következőkben ezek kerülnek tárgyalásra.

2.1.2. Adaptív forgalomirányítási módszerek

A következő kategória, amely már jobban részletezésre fog kerülni, az online rendszerek. Ezen az úton elindulva jutottam arra a következtetésre, hogy valamilyen, dinamikusabb, adaptívabb forgalomirányítási rendszert lehetne kitalálni. Annak ellenére, hogy a következőkben felsorolt adaptív forgalomirányító rendszerek valamivel intelligensebbek mint az átlagos rendszerek, feltételezhető, hogy még a következőknél is lehet jobb megoldásokat elérni.
Ki lehet emelni, hogy az alábbi rendszerek többsége önmagukban implementál, különféle szenzorokat és detektorokat, ezek ha említésre is kerülnek, a dolgozatban tervezendő és implementálandó megoldás nem fog a különféle valós világba való infrastrukturális megoldásokkal foglalkozni, lehet itt gondolni a különféle szenzorokra vagy hardverekre.
Visszatérve az adaptív forgalomirányító rendszerekre, ezek általában valamilyen online rendszerek amelyek a fázisterveket valós idejű adatok alapján képesek módosítani, vagy pedig valamilyen algoritmus segítségével új tervet tudnak létrehozni. A célja ezeknek az adaptív rendszereknek a forgalom valós idejű állapotához való alkalmazkodás. Általában ezek az adaptív rendszerek különféle szenzorokra, detektorokra támaszkodva gyűjtenek adatokat a forgalom pillanatnyi állapotáról és a beállításuknak megfelelően módosítják a fázistervüket.
Néhány adaptív rendszer [2] és jellemzőjük:

- **SCATS (Sydney Coordinated Adaptive Traffic System)**:
  A SCATS rendszer volt az egyik legelső adaptív rendszer. 1970-ben kezdték el fejleszteni Ausztráliában. Az ott lévő nagy városok forgalmának irányítására alkalmazták. Manapság leginkább szimulációs környezetekben használják, mert jól együtt tud működni egyes szimulációs szoftverekkel, mint pl: PTV VISSIM.
  A SCATS rendszer működésében fellelhető a korai adaptív rendszerek gyerekbetegségei. Szenzorok segítségével, nyomásérzékelőkkel adatokat gyűjt a forgalom "valós" idejű helyzetéről, ezeket feltöltik egy központi számítógépnek, amely a beérkező adatok segítségével kiszámolja az optimális ciklusidőt, illetve ajánlást tesz egy új fázisterv implementálására [2]. Ezután egy emberi munkaerőnek kell eldöntenie, hogy a rendszer által szolgáltatott információk alapján, hogy módosítja a forgalomirányítás működését.

A rendszer viszont kiforratlan volt és voltak hiányosságai. Valós idejű visszajelzés ténylegesen nem volt a közlekedés állapotáról, hiszen egy, a megállás helyét jelző vonalnál elhelyezett nyomásérzékelő nem tud elégséges adatokat szolgáltatni a központi rendszer számára. Másik hiányossága a rendszernek a fentebb említett emberi munkaerő, amelyet alkalmazni kellett. A rendszer nem volt más, mint egy ajánlástevő rendszer, amelynek a számításait és becsléseit egy embernek ki kellett értékelnie. Ez további nem kívánt optimalizációs problémákhoz vezethet. Az emberi komponens tévedhet, amely azzal járhat, hogy a rendszer által ajánlott fázistervek közül nem mindig a megfelelőt választja ki implementációra.

A rendszer működéséből levonható következtetések az alábbiak:

- Ahhoz, hogy a rendszer a lehető legjobban tudjon döntéseket hozni, szükség van a közlekedés valós idejű helyzetének minél nagyon pontosságú és részletességű ismeretére. Minél több és pontosabb információt tudunk gyűjteni a döntéshozó rendszernek, annál inkább optimálisabb döntéseket lesz képes majd hozni.
  Ebben az esetben a kutatáshoz és implementációhoz ebből is adódóan egy szimulációs környezetet kell majd felhasználni, amelyből ennek köszönhetően a közlekedés pillanatnyi helyzetéről az információ rendelkezésre fog állni.
- Nem elegendő egy olyan irányító komponens létrehozása, amely csak ajánlásokat tesz és egy embernek kell ezeket implementálnia. A rendszernek egy konkrét döntéssel kell előállnia az adott közlekedési helyzetre és ennek megfelelően implementálnia kell a döntést. A különféle infrastrukturális kihívásokkal ismét csak a szimulációs környezetnek köszönhetően nem kell foglalkozni.

- **UTOPIA (Urban Traffic Optimization by Integrated Automation)**:
  Az UTOPIA rendszer egy valamivel innovatívabb elképzelés alapján lett megtervezve.

  Azokon a célokon túl, amiket az előbb tárgyalt technológiák kitűztek, egyel tovább ment az UTOPIA. Úgy gondolták, hogyha egyedülálló elsőbbséget adnak a tömegközlekedés járműveinek, azzal nem csak az általános javulásokat fogják a rendszerben megfigyelni, hanem gazdasági előnyökkel is számolhatnak [2].
  Ezen kívül az UTOPIA rendszere egy érdekes megközelítést vet fel. A rendszer a SPOT szoftver segítségével képes arra, hogy azon kívül, hogy egy központi szá-

mítógép fázisterveket határoz meg a csomóponti egységeknek, további lokális döntésekre és megfigyelésekre is képes csomóponti egységeket használ [2]. Ez egy egyedi megközelítés az általános centralizált megoldásokhoz képest.
A rendszerről levonható konklúziók az alábbiak:

  - A fentebb leírt két pont, egyaránt érdekes lehetőségeket vet fel. Egyfelől, a rendszer implementációjában van-e értelme, járműveket megkülönböztetni és ezt figyelembe venni a döntések meghozatalakor. A megerősítéses tanulást használó ágensnek lehetséges ilyen metrikákat átadni, a kérdés, hogy megérné-e egy ilyen megkülönböztetés a járművek között.
  - Másfelől pedig az a szemlélet, hogy a rendszer által csomópontokban is elhelyezett, önmagukban döntésekre, és egymással kommunikáló komponensek használata mennyire lenne indokolt a probléma megerősítéses tanulást használó ágenssel való megközelítésekor. Más tanulmány [3] is vizsgálta a lehetőségét több ágens implementálására a közlekedési lámpák irányítására.

- **INSYNC**:
  Az INSYNC rendszer a piacon lévő forgalomirányító rendszerek közül a leginkább modernebbnek nevezhető. Az eddigi, ezredforduló előtti rendszerek technológiája is tükrözte a korukat. Az INSYNC egy központi számítógép helyett, egy decentralizált plug-and-play rendszer, ami IP kamerák segítségével térképezi fel a forgalmat [2]. Az alap architektúra két részre bontható fel. A rendszer „szeme” maga az IP kamera, amely az elsődleges adatforrása a rendszernek, a feldolgozó egység pedig az „agyként” funkcionál. A kép alapján megszámolják a közlekedési lámpa előtt álló járműveket, meghatározzák, hogy mennyi ideje várnak a zöldre. A központi egység pedig, amely a csomópont mellett van egy dobozban, meghatározza, hogy a saját kamerája, illetve a többi csomópont által szolgáltatott információk alapján a legjobb lokális döntést. A rendszer próbál zöldhullámokat létrehozni, amikkel minimalizálni próbálja a megállások számát. Ha erre nincs lehetőség, akkor pedig mellékutak felé próbálja terelni a forgalmat, hogy a terhelést levegye a már így is túlterhelt útszakaszról.
  Érdekes az a tény, hogy ismét egy olyan rendszert láthatunk, amely a decentralizált, csomópontonkénti megközelítés lehetőségét valósítja meg. Ez az irány a megerősítéses tanulás témakörében is nagyon erősen él, miszerint akár több ágens alkalmazása is egy járható út a probléma megoldására. További releváns információ pedig a kamerák használata az adatgyűjtésre, ez a megoldás is azt mutatja, hogy

a lehető legnagyobb mennyiségű információra van szükség a forgalom megfelelő irányítására. A szimulációs környezetben való implementálás miatt, ezzel nem lesz probléma.

2.1.3. IntelliLight

Az IntelliLight[4] projekt megalkotói ugyancsak a közlekedési lámpák intelligens irányításában látnak javítani valót. A projekt le szeretné váltani a mostanában megszokott fix fázisú közlekedési lámpákat. Megoldásként megerősítéses tanulást használó ágenst alkalmaznak. A tanulmány szerint a legnagyobb kihívást az jelenti, hogy miként lehet a környezetet reprezentálni, illetve a döntések és a környezet közötti kapcsolatot felírni. A projekt próbál a valósághoz közelibb közlekedési helyzeteket teremteni és itt megvizsgálni, a megerősítéses tanulást használó ágens hatékonyságát.
A projekt megközelítése a közlekedési lámpák irányítására a megerősítéses tanulás segítségével, a Deep Q-Learning(DQN) módszer alkalmazása. Későbbiekben mélyebben szeretnék belemenni a módszer részleteibe de röviden a lényege, hogy az ágens figyeli a környezetének állapotát, döntést hoz a megfelelő akcióról, majd ezt a döntést megvalósítja a környezetben. A megerősítéses tanulás megvalósítására mély neurális hálózatot használnak, illetve egy Q függvénnyel megbecsült értékkel számítják ki a lehetséges jutalmat.
A megerősítéses tanulás implementációjához szükség van az ágensre, továbbá három fő aspektus meghatározása. Az állapot, jutalom és akció definiálására.

Az IntelliLight projekt a fentieket az alábbi megközelítés szerint képzeli el [4] :

- **Állapot**: Megvizsgálja a várakozó autósor hosszát, a számát az autóknak az adott sávban illetve az átlagos várakozási időt.
- **Akció**: Az ágens által végrehajtható akciók "a válts lámpát" és a "ne változtass lámpát" állapotokból áll.
- **Jutalom**: A jutalom érték kiszámításához figyelembe vesznek több metrikát is.
  - Megvizsgálják, hogy egy adott akció után hány darab autó tudott átmenni a kereszteződésen
  - A számát a lámpák váltásának
  - Az átlagos várakozási időt

- A hosszát a várakozó autók sorának
- Az egy helyben álló autók számát
- A menetidejét az autóknak miután az ágens egy akciót végrehajtott

Ezen metrikák alapján egy súlyozott összeget definiálnak, amely felhasználásával az adott akció utáni jutalom kiszámítható. Az ágens célja a maximum jutalom elérése hosszútávon, amelyet a Bellman-egyenlet ír le:

$$q(s_t, a, t) = r_{a,t+1} + \gamma \, max \, q(s_t, a, t, a', t + 1)$$

A modell, amelyen a kísérleteket végezték, két részből áll. Az "offline" rész során, adatokat gyűjtenek a még ekkor fix fázisidejű közlekedési lámpákkal. Ezekkel az adatokkal tanítják az ágenst. Ezután az "online" részben az ágens fog minden egyes időpillanatban megfigyeléseket tenni a környezetről, majd pedig döntéseket hoz. A meghozott döntéseket, azaz, hogy váltson-e lámpát, vagy sem, a jutalom függvény alapján kiértékelik. Az adott hármast: (állapot, akció, jutalom) eltárolják későbbi vizsgálatra.

2.1.4. A kísérleti környezet

Az IntelliLight projekt egy szimulációs környezetet használ a kísérletek lebonyolítására. A SUMO (Simulation of Urban Mobility) [1] segítségével könnyen lehet közlekedési hálózatokat szimulálni és tervezni. A környezethez tartozik egy flexibilis API, amely segítségével különféle metrikákat lehet kinyerni az általunk elkészített szimulációból, illetve interakciókat lehet végezni a környezettel. Később még a SUMO működéséről mélyebben is szó fog esni.

2.1.5. A rendszer felépítése

Ezzel együtt meghatározható, hogy az IntelliLight projekt az alábbi módon épül fel [4]. A szimulációs környezetben elkészítenek egy közlekedési helyzetet. Meghatároznak egy úthálózatot, az úthálózatban elhelyeznek autókat, ezeket az járműveket a SUMO környezetben, előre implementált logika irányítja és mozgatja.
Az autók mozgásáról és a környezet adott pillanatban lévő állapotát a SUMO API-on keresztül lehet kinyerni.
A projekt által figyelt metrikák az alábbiak:

[1] https://www.eclipse.org/sumo/

- Várakozó autósor hossza
- Átlagos várakozási időt
- Fázis váltások
- Autók száma

A fenti metrikák azok, amelyeket a megerősítéses tanulást használó ágens figyelembe tud venni az akciók meghozatalához. Ezek a metrikák a jutalom függvényben is szerepelnek. Ennek megfelelően számolható ki az, hogy egy adott döntés, amelyet meghozott az ágens, ténylegesen mennyire volt jó.
A rendszert további optimalizációs technikákkal látták el. Az egyik ilyen az "experience replay" [5]. A módszer lényege, hogy az ágens a saját maga által elkészített memóriából időről-időre kiválaszt egy részt, és az alapján módosítja a hálózatot. Annak érdekében, hogy az ágens megtanulja a ritkábban előforduló közlekedési helyzeteket, különféle memória részekbe fogják tárolni, így a gyakoribb fázis kombinációk nem fogják a tanulás során megzavarni az ágenst a kevésbé gyakori kombinációk megtanulásában, hiszen a memória mintavételben nem lesznek sokszorosan túlsúlyban a gyakoribb esetek.

2.1.6. Összehasonlított forgalomirányítási módszerek

A projekt hatékonyságának validációjára összehasonlítást végeztek különféle eddig használt megoldásokkal, majd összehasonlították a teljesítményüket. A vizsgált metodikák között volt a fix idős, statikus forgalomirányítás (FT) [6] és az önszervező forgalomirányító (SOTL) [7]. Ezeken kívül megvizsgáltak egy másik mély-megerősítéses tanulásos módszert (DRL) [3].
A fentebb említetteken kívül a projektnek három különböző verzióját különböztetik meg:

- Az alap IntelliLight verzió
- A verzió, amelyben használták a memória mintavételt
- Egy verzió, amelyben a memória mintavételt, illetve a "Phase Gate" módszerüket: A "Phase Gate" módszert nem részletezik

A kísérletek elvégzésére többféle konfigurációt vizsgáltak, itt megfigyelték, hogy különféle közlekedési környezetben melyik forgalomirányítási rendszer, hogyan teljesített. Ezen kívül szintetikus, illetve valós adatokon is megvizsgálták a rendszerek hatékonyságát. A valós világból származó adatokhoz, megfigyelték hosszabb ideig, hogy különböző

napokon egyes közlekedési csomópontoknál milyen a jellemző forgalom [4] .
Miután lefuttatták a különböző konfigurációkkal a kísérleteket. A konklúziók az alábbiak:

- A legtöbb esetben az IntelliLight mindegyik verziója közel azonos vagy jobb eredményeket ért el, mint a többi vizsgált módszer.
- A mintavételezés és a memória ("experience replay") néhány esetben növelte a rendszer hatékonyságát, emiatt a szerepe inkonzisztens.
- A "Phase Gate" használata a mintavételezéssel és memória komponenssel nagy javulást hozott a várakozási idők tekintetében, illetve ezzel lehetett a legjobb jutalom értéket elérni.

2.1.7. Felhasznált technológia

A készítői a projekt megvalósításához, az alábbi eszközöket használták [4]:

- A projekt Python 3.6-ban íródott
- A neurális hálózatok implementációjára Keras 2.2.0 és Tensorflow 1.9.0 könyvtárakat alkalmaztak
- A szimulációs környezet a fentebb említett SUMO volt, pontosabban a 0.32-es verzió TraCI komponenssel

A projekt természetéből kifolyólag a Python [8] nyelv használata magától adódó, illetve a neurális hálózatok témaköre is egyértelműen a Python, ezzel együtt pedig a Tensorflow [9], ami magába foglalja a Keras könyvtárat, használatára ösztönzi a fejlesztőt. A Python nyelv kimondottan alkalmas a kutatási munkára, a Tensorflow pedig megkönnyíti a neurális hálózatok implementációját a fejlesztő számára. A SUMO szimulációs környezet API-a jól együtt tud működni a Python nyelvvel, ami pedig ismét csak megkönnyíti a munkáját a fejlesztőknek.

2.1.8. Konklúzió az IntelliLight projektből

Az IntelliLight projekt megerősítette azt a feltevést, hogy a kutatott projektnek van alapja és megvalósítható. Ezzel együtt ez azt is megerősítette, hogy a megerősítéses tanulást használó ágens módszere jó út a közlekedési lámpa probléma megoldására. A projekt

implementációjára a kutatás során többször is a SUMO szimulációs környezet neve bukkant fel, az IntelliLight projekt is igazolta, hogy ez lesz a megfelelő komponens a közlekedés szimulációjára.
A különböző kísérletek eredményét látva, arra lehet következtetni, hogy versenyképes tud lenni, mi több, jobb megoldás tud lenni a megerősítéses tanulást használó ágens a közlekedési lámpák irányítására.

2.2. Neurális hálózatok

Az alábbi részben bemutatásra fog kerülni a neurális hálózatok kialakulása és alapvető működése.

2.2.1. Új számítási módszerek

Korunk egyik legizgalmasabb kutatási és technológiai irányát, egy, a számítástechnika hagyományos szemléletével szembemenő megközelítés adja. A most ismert analitikus, algoritmikus megközelítése az információfeldolgozásnak, amely szerint a probléma megoldására írnunk kell egy utasítás sorozatot, amit a számítógép sorban végrehajt. Annak ellenére, hogy ez a megközelítés vitathatatlanul sikeres és kiválóan alkalmazható az elénk kerülő problémák nagy részén, ezzel párhuzamosan kialakulóban volt a 20. század közepe fele egy másik megközelítése az információfeldolgozásnak, ami sokkal inkább a természetet és a biológiát vette alapul. [10]
Amikor megvizsgáljuk, a körülöttünk élő biológiai lényeket, észrevesszük, hogy ők nem követik a digitális paradigma algoritmikus szabályait. Akárhogy szemléljük meg, mi emberek se így működünk. A kérdés tehát, hogy van-e helye egy új, természetet alapul vevő megközelítésnek. A válasz pedig igen, mégpedig zászlóshajója ennek a „régi-új” technológiának a neurális hálók.

2.2.2. Mi is a neurális háló?

A neurális háló egy olyan számítási módszer, amelynek alapjául az idegrendszert vesszük. Természetesen egy igencsak leegyszerűsített struktúrát kell elképzelni, de mégis a koncepció adott, miszerint az idegrendszer nem más mint, neuronok közti kapcsolatrendszer. A gondolatmenet szerint, ha képesek lennénk lemodellezni, még ha csak leegyszerűsítve is, az idegrendszer kapcsolatrendszerét, illetve a neuronok közti kommunikációt, akkor képesek lehetnénk tanítani ezt a rendszert hasonlóan, mint ahogy mi vagy más élőlények tanulnak. A legelképesztőbb az a tény, hogy már 1958-ban egy Frank Rosenblatt nevű

kutató képest volt egy hasonló rendszert megalkotni. [10]
Ez volt a perceptron. A perceptron képes volt vetített nyomtatott betűket felismerni tanítás alapján, ami egészen elképesztő. A kortársak miután megvizsgálták a rendszert, arra a konklúzióra jutottak, hogy a technológia nem több, mint gyerekjáték és komolyabb problémák megoldására nem alkalmazható, ezzel pedig a technológia jó ideig komolyabb figyelmet nem kapott.[10]
Az egyetlen probléma az volt, hogy a kortársak, akik megvizsgálták a perceptron mögött rejlő technológia lehetőségeit, mint később kiderült, tévedtek és mostanra az egyik leginkább kutatott témává nőtte ki magát az említett technológia.

2.2.3. Milyen esetekben érdemes neurális hálókat alkalmazni?

Ha megvizsgáljuk a neurális hálók alkalmazását, akkor azt látjuk, hogy rengeteg területen képesek jobb eredményt elérni, mint a hagyományos algoritmikus módszerek. Alapvetően olyan területeken szokás alkalmazni, ahol ismeretlenek a probléma megoldásához szükséges szabályok vagy a megismerésük túl költséges vagy időigényes lenne [11]. Esetünkben, nincs egy univerzális forgalomirányító „egyenlet”, amely segítségével minden város forgalmi adottságainak megfelelő paraméterek kiszámíthatók lennének. Emiatt is jó alany a neurális háló alkalmazására.
Továbbá szükséges, hogy a problémával kapcsolatban hatalmas adathalmaz álljon rendelkezésre. Egy város közlekedési rendszere nagyon jó forrás ehhez, hisz folyamatos és sokféle statisztika vonható belőle. Gondolhatunk itt akár a csomópontokon várakozó autókra, vagy az adott csomópontokon áthaladó autók számára, azaz a „flow-rate”-re.

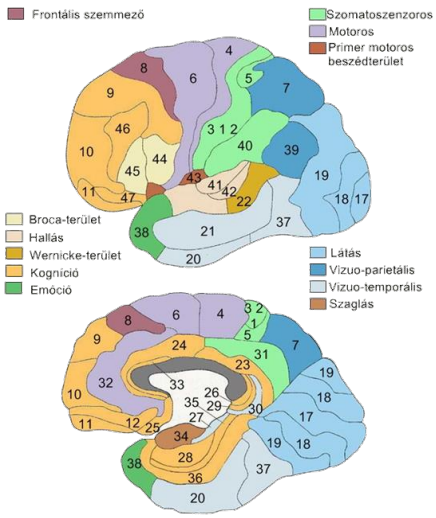
2.2.4. A neurális hálózatok jellemzői

Ahhoz, hogy a neurális hálózatokat jellemezni lehessen, meg kell érteni a felépítésüket, ami a bevezetőben már említve volt, hogy neuronokból állnak. Ezek az alapvető építőelemei a hálózatnak. Tekinthetünk rájuk úgy, mint processzorok, de a továbbiakban csak, mint neuronok lesznek hivatkozva A neuronok egymással összeköttetésben állnak, a köztük lévő kapcsolatot pedig egy változtatható súlytényező adja meg. Ez egy szorzószám, akár csak a súlyozott gráfoknál. Ez a súlytényező adja meg, hogy az adott két neuron mennyire dolgozik együtt, vagy, hogy milyen viszonyban vannak egymással.[11]
Következő fontos jellemzője a neurális hálónak, hogy más megközelítést igényel az alkalmazásuk, emiatt nem is programozzuk a működésüket, hanem inkább tanítjuk őket. Ezt úgy kell érteni, hogy természetesen még mindig kell használnunk egy programnyel-

vet, hogy a számítógépünkön megépítsünk egy neurális hálót, viszont maga a konkrét működését nem mi mondjuk meg. Egy szemlélet szerint, egyfajta fekete dobozként kell gondolni a neurális hálóra, amely attól függően, hogy mi milyen jól választunk meg bizonyos paramétereket, mint például a tanító adatok, illetve az aktivációs függvény vagy a hálózati topológia, fog valamilyen működést produkálni. Ez a működés remélhetőleg a számunkra megfelelő, viszont ahogy, a kutatott irodalom mutatja, közel se ilyen egyszerű a helyzet. Ahhoz, hogy a neurális hálónk azt tegye, amit mi szeretnénk, hosszas próbálkozások szükségesek.[12]
A neurális háló, felépítéséből adódóan egy hibatűrő rendszer [11]. A tárolt információ eloszlik a hálózatban, emiatt is szokták olyan területeken alkalmazni, ahol nem tudni biztosra, hogy a bemeneti adatok mindig megfelelőek, vagy hibátlanok-e. Az elosztott párhuzamos tudásreprezentáció miatt, néhány súlytényező megváltozása a rendszerben, nem fogja a hálózat működését befolyásolni [10] [11]. Ez egy nagyon jó tulajdonság tud lenni egy forgalomirányító rendszerben.
Három fő tényezője van egy neurális hálózatnak. Az első a neuronok aktivációs függvényei, a következő a hálózat összeköttetési sémája, azaz miként kapcsolódnak egymáshoz a neuronok, illetve a tanítási módszer. A tanítási módszer több szempontból is érdekes. Később kifejtésre kerül részletesebben, de elöljáróban annyit, hogy megkülönböztetünk felügyelt és felügyeletmentes tanítású neurális hálózatokat [11]. Az alkalmazási terület, vagy a probléma határozza meg, hogy melyiket választjuk. A tanítási módszeren kívül fontos, hogy a tanító szabályokat alkalmazó algoritmus miként finomhangolja a súlytényezőket a tanítási folyamat során. [13]

2.2.5. Milyen tanulságot tudunk vonni a természetes idegi hálózatokból?

Mint többször is említve volt, a neurális hálózat alapjául az idegrendszert vették alapul. [11] Az emberi idegrendszer vizsgálata emiatt fontos intelmeket tud adni számunkra is, amikor megfogalmazunk elvárásokat a neurális hálóval szemben. Az egyik ilyen fontos dolog, hogy az idegrendszerre nem egy univerzális gépként, hanem sokkal inkább specializált gépként kell tekinteni.



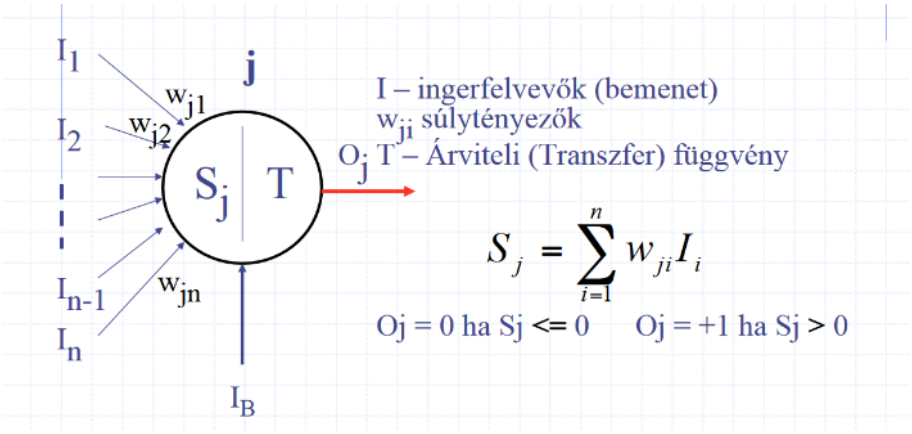
2.1. ábra.
Brodmann-modell (forrás:Dr. Kutor László előadása)

A Brodmann-modell megvizsgálásakor érdekes megfigyeléseket lehet tenni. A különböző agyi területek más-más funkciót látnak el az agyunkban, továbbá ezek, a területek struktúrájukban is különböznek egymástól.
Ez azt jelenti, hogy amikor neurális hálót szeretnénk alkalmazni egy bizonyos problémára, akkor azt fogjuk látni, hogy a probléma jellegétől függően a feladatra kell szabnunk a neurális hálót és a már fentebb említett jellemzőket ennek megfelelően kell megválasztani. Fentebb volt említve, hogy egyes területeken a neurális hálózatok jobb eredményeket tudnak elérni, mint a „hagyományos" analitikus megközelítést alkalmazó módszerek [11]. Ez nem jelenti azt, hogy kimondhatjuk azt, bármelyik is felsőbbrendű lenne a másiktól, viszont van amiben az egyik jobb, valamiben pedig a másik. Például alakfelismerésben a neurális megközelítés messze jobb eredményeket ért el, mint az eddigi „hagyományos" megközelítés [11]. Ennek többek közt az az oka, hogy az idegrendszer egyik fő feladata a természetben is, hogy alakokat és mintákat ismerjen fel. [10]
A minta felismerés viszont további érdekes lehetőségeket vet fel. Olyan területeken, ahol hatalmas mennyiségű adat keletkezik, például a közösségi médiákon, vagy adatfarmokon, szükségünk lehet olyan minták felismerésére, amiről pontosan nem tudjuk, hogy mi, ennek ellenére viszont ott van. Gondolhatunk itt akár a YouTube Recommendations algoritmusára, amely próbálja megtippelni, hogy mit szeretnénk megtekinteni a weboldalon, vagy a hirdetésekre, amelyek mindenhova követnek minket és próbálják kitalálni, hogy mit vásárolnánk meg nagy valószínűséggel. Ilyen esetben nem segít a „hagyományos" módszer, hiszen nem tudjuk pontosan, hogy mit keresünk, illetve az adat mennyisége

olyan nagy, hogy a manuális elemzés lehetetlen.

2.2.6. Neuron formális felépítése és működése

A formális neuron modelljét W. Mc Culloch és W. Pitts uraknak köszönhető, még 1943-ból [14]. Ők voltak az elsők, akik az agyat, számításokat végző szervernek tekintették. Az agy elemi processzorainak tekinthetjük a neuronokat. [10] [11]
A neuron felépítése meglehetősen egyszerű. Vannak ingerfelevők, amiket az egyszerűség kedvéért hívjunk bemeneteknek, ezen kívül a neuron rendelkezik valamennyi számú kimenettel. Ez a hálózat struktúrájától függ. Jelen esetben ez legyen egy. A neuron az adott bemenethez rendel súlytényezőket (w_{ji}), (j neuronba, i neuron irányából) amely súlytényező a két neuron közti kapcsolat „erősségét” határozza meg. Ennek a súlytényezőnek a finomhangolása az egyik fontosabb a neurális háló tanítása során [10].
A neuron működése is meglehetősen egyszerű. Fogadja a többi neurontól érkező bejövő jeleket. Ezeket a jeleket az előbbiekben már említett súlytényezővel szorozzuk, majd ezeket a kiszámított szorzatokat minden egyes bemenetre összeadjuk. Ez a súlyozott összeg S_j [11].



2.2. ábra. Neuron formális modellje (forrás: Dr. Kutor László előadása)

Eleinte úgy gondolták, hogy a neuron kimenete bináris működésű, azaz, ha az S_j szumma értéke egy bizonyos érték felett van, akkor az O_j kimenet 1, ha bizonyos érték alatt, akkor O_j kimenet 0.
Ez az elképzelés biológiai inspirációból ered, mivelhogy messziről nézve úgy tűnik, hogy az idegsejtek is ezt csinálják, hiszen, ha elér egy bizonyos ingerszintet, akkor „tüzel” a neuron, ha ezt viszont nem éri el, akkor pedig nyugalomban marad. Később rájöttek a tudósok, hogy ez a bináris működés nem teljesen fedi le a valóságot [10] [11].

2.2.7. Aktivációs függvények szerepe

Az aktivációs függvények megválasztása kulcsfontosságú a rendszer tanítása és működése szempontjából. A klasszikus aktivációs függvények bináris, kapcsolószerű megközelítését elhagyták a kutatók és helyette, olyan függvényt választottak, amely könnyen differenciálható. A differenciálhatóság szerepe, a későbbiekben sorra kerülő hiba függvény kiszámítása, illetve a hálózat tanítása során kerül tárgyalásra. [11]
Az aktivációs függvény megválasztása során nincsen ökölszabály, ha látjuk, hogy nem úgy működik a rendszer, mint ahogy azt reméltük, egyszerűen lecserélhetjük azt és folytathatjuk a tesztelést. Szerencsére sok ajánlás van különféle aktivációs függvényekre. Ezek közül választhatunk. Megint csak fontos megjegyezni, hogy a neurális hálózat egy specializált rendszer, ezért az adott problémára kell szabni azt.

2.2.8. Neurális hálózatok topológiája

A neurális háló legfontosabb jellemzője a topológiája. A neuronok összeköttetési rendszerét, a bemenetek és kimenetek helyét legjobban irányított gráffal tudjuk bemutatni. A megfeleltetés elég kézenfekvő, a neuronok a gráf csomópontjai a kapcsolatok, a neuronok között pedig az irányított élek, a bemenettől a kimenet felé. A súlytényezőket pedig, mint a súlyozott mátrixnál szokás az élekhez rendelhetjük hozzá [11].
Többféle topológia létezik. A legtöbben nem csatlakozik minden neuron minden neuronhoz, azaz nincsen teljes gráf, ami pedig lehetőséget ad arra, hogy diszjunkt részhalmazokra bontsuk a neuronokat.
Háromféle neuront különböztető meg, a bemeneti, kimeneti és a rejtett neuronok. A bemeneti neuronok típusaikban is különböznek a többi neurontól. Feladatuk az adatok eljuttatása a többi neuron felé, azaz a bemenetük a neurális hálózat bemenete, kimenetük pedig további neuronok bemenete. A kimeneti neuronok a rendszer kimeneteként szolgálnak, továbbá, ha olyan a topológia visszacsatolás során más neuronoknak is adhatnak bemenetet. A rejtett neuronok a belső rétegeit adják a hálózatnak, mind a bemenetük és kimenetük is más neuronokhoz csatlakoznak. Ezek alapján rétegekbe lehet szervezni a hálózat neuronjait. A fenti típusoknak megfelelően, bemeneti réteg, rejtett réteg és kimeneti rétegeket különböztetünk meg. A bemeneti réteg, a már fent megemlített buffer jellegű neuronokat foglalja magába. Ezek a neuronok információfeldolgozást nem végeznek, csak továbbítják az adatokat a következő rétegnek. A hálózatnak ennek megfelelően legalább két rétege van. Egy bemeneti és egy kimeneti. A kettő réteg között pedig elvi síkon bármennyi rejtett réteg lehet. Amikor egy hálózat rétegszámáról beszélünk a megál-

lapodás szerint, olyan rétegeket számolnak bele, amelyek aktív, feldolgozó tevékenységet végeznek [11].
Az összeköttetési rendszer a neuronok között, két nagy csoportba sorolható. Az egyik az előrecsatolt hálózat, a másik pedig a visszacsatolt hálózat. Akkor nevezünk egy hálózatot visszacsatoltnak, ha a hálózatot leíró gráf tartalmaz hurkot, ellenkező esetben előrecsatolt a hálózat. A neurális hálózatok a gráfok mellett, mátrixokkal is leírhatók. Az előrecsatolt hálózatoknál az egyes rétegek neuronjaihoz tartozó súlyok egy mátrixba reprezentálhatók, ahol a mátrix egyes sorai az egyes neuronok súlyaiból képzett vektorok. W mátrix tehát, az i-edik rétegben lévő neuronok súlyaiból, mint sorvektorokból képzett mátrix [11].
Az a tény, hogy a neurális hálók komponensei mátrixokként reprezentálhatóak, nagyban megkönnyíti a műveletek végzését, hiszen a fentebb említett szummázás és a későbbi tanítás mind-mind mátrixműveletekkel elvégezhetőek és szerencsénkre a számítógépek videókártyái meglehetősen jók erre a célra. Megfigyelhető, hogy a videókártyák számítási teljesítményének növekedése jótékony hatással volt a neurális hálózatok alkalmazásának elterjedésére is, hiszen a tanítása a hálózatnak nagyon számítás igényes folyamat.

2.2.9. A neurális hálózat tanítása

Amikor a neurális hálózatok tanulását említjük, akkor arról a folyamatról beszélünk, amely során finomhangoljuk a rendszert, annak érdekében, hogy elérjük a kívánt működést. A tanítási folyamatok közül kétfélét különböztetünk meg. Az egyik a felügyelt, a másik pedig a felügyelet nélküli tanítás. A neurális hálózatok működéséhez alapvető, hogy megértsük, azok miként tanulnak.
A felügyelt tanítás menete, jól bemutatja a neurális hálózat működésének alapjait. A folyamat az alábbi gondolatmenet szerint halad. A hálózatnak bemeneti adatokat adunk, amelyekhez mi ismerjük az elvárt kimeneti adatokat. Ha a rendszer hibázik, akkor a rendszer belsejét módosítjuk, majd megismételjük a folyamatot. Ismét odaadjuk a bemeneti adatokat, megvizsgáljuk a kimenetet, és ha hibázik, akkor ismét állítunk a rendszer belsején. Ezt mind addig megismételjük, amíg egy általunk meghatározott hibaráta alá nem esik a rendszer pontossága. Miután elértük, ezt a szintet a rendszernek olyan bemeneti teszt adatokat adunk, amelyeket a tanulása során még nem látott. Ha ezekre az adatokra is jó kimeneteket kapunk, akkor készen is volnánk. [10]
A kérdés viszont, hogy az a bizonyos lépés, a rendszer módosítása, hogyan történik és mi alapján. Itt vissza kell térni a rendszer aktivációs függvényekhez és a különböző súlyértékekhez. A neuronok közti kapcsolatot a súlyértékek határozzák meg, illetve egy eddig még meg nem említett paraméter, a „bias” érték. A cél, hogy valamilyen módon meghatá-

rozzuk annak a mértékét, hogy hol és mennyivel kellene módosítani ezt a két paramétert, ahhoz, hogy a rendszer az adott bemenetre az elvárt kimenetet adja.
Ahhoz, hogy értsük, hogy mi alapján kell megváltoztatni a neuronok közti súlytényezőket kettő alapvető szabályt meg kell ismertetni.
Az első a Hebb szabály [10]. A Hebb szabály szerint két neuron közti kapcsolat attól függ, hogy együtt aktiválódnak, vagyis együtt dolgoznak-e. A szabály az idegrendszerben lévő neuronok működése alapján fogalmazódott meg. Két neuron közötti $w_{ji}(t+1) = w_{ji} + \alpha * O_i * O_j$ a két neuron $w_{ji}(t+1)$ időpillanatában vett súlyérték, az α pedig tanítási tényező, egy skalár, az Oi és az O_j pedig a kimenete a két neuronnak az adott időpillanatban. A szabály a célértéktől függetlenül, csak a két neuron együttműködése alapján módosítja a súlyértéket.[11]
A következő szabály a delta szabály. A delta szabály figyelembe veszi a célértéket, megnézi, hogy a neuron kimenete mennyiben tér el ettől a célértéktől. Ha a célérték magasabb, akkor a súlytényezőt felfele, ha kisebb, akkor a súlytényezőt lefele kell módosítani. A célérték segítségével felügyelhetően taníthatóvá tettük a rendszert, hisz ennek segítségével szabadon állíthatóak az értékek, amíg a számunkra megfelelő kimenetet nem adja rendszer. Egy problémát viszont még nem tudtak megoldani a kutatók, mégpedig, hogy nem tudták tanítani a rejtett rétegekben lévő neuronokat. Emiatt a neurális hálózatok, akárhány rejtett réteggel is rendelkeztek, jelentős előrelépést nem tudtak kimutatni a teljesítményükben. A cél az volt, hogy valahogy képesek legyenek a belső rejtett rétegeket is tanítani.[10]
A jelentős áttörést [11], az 1986-ban publikált, hiba visszaterjesztéses algoritmus kidolgozása [15] jelentette. A módszer újdonsága az volt, hogy ha az aktivációs függvénynek egy könnyen deriválható függvényt választunk meg, akkor bármely elemnek a hálózatban kilehet számolni a súlytényező változtatásának igényét. A hiba meghatározására szükség van bevezetni egy új fogalmat, a hibafüggvényt. Ennek segítségével lehet meghatározni azt, hogy mekkora a mértéke a hibának. Ezután a módszer az, hogy a hálózat súlytényezőit a hiba létrehozásában játszott szerepükkel arányosan változtatjuk meg. A megváltoztatás mértékét pedig a hibafüggvény parciális deriváltja szerint megváltoztatjuk. A módszer neve onnan ered, hogy visszaterjesztjük ezt a finomhangolást lépésről lépésre az összes elemre.
Miután elkezdték alkalmazni a fent említett technikát a neurális hálózatok tanítására, jelentős eredményeket értek el számos területen [10][11]. Mielőtt azt gondolnánk, hogy minden problémánkra megoldást találtunk, egy fontos felismerést kell tennünk. Mégpedig, hogy a módszer egy grádiens módszer, a grádiens módszerekre pedig jellemző, hogy

„beszorul” a lokális minimumokba [10]. Ez azt jelenti, hogy egy bizonyos lépésszám után, annak ellenére, hogy a globális minimum nem az, amelyet kerestünk, nem tudunk tovább menni, így a hálózat tanítása megakad egy ponton, azaz nem garantált, hogy neurális hálózat megtalálja azt az eredményt, amelyet szeretnénk. Ha ilyenbe ütközünk, akkor, egy valamit tudunk tenni, mégpedig, hogy elkezdjük még egyszer a hálózat tanítását, más kezdeti paraméterekkel [10].

2.2.10. Neurális hálózatok alkalmazásának menete

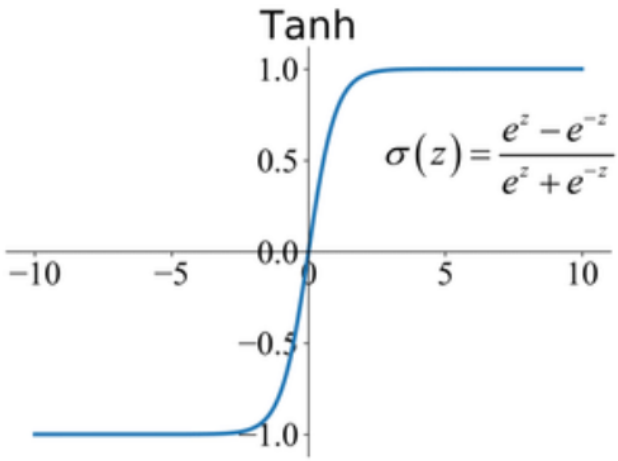
Az említett ismeretek segítségével leírható, hogy mi a neurális hálózatok alkalmazásának a menete. Az első lépésben fontos eldönteni azt, hogy milyen adatok alapján szeretnénk a neurális hálózatunkat tanítani, fontos, hogy rengeteg adat álljon rendelkezésre a minél jobb eredmény érdekében. A következő feladatunk a neurális hálózati paradigma kiválasztása. Itt arra, gondolunk, hogy elkezdünk kutatni, hogy a problémánkra ismert-e már egy olyan jól bevált módszer, amely mentén elindulhatunk mi is.
Ezek után, figyelembe véve az előző pontot, meghatározzuk a hálózat különféle jellemzőit. Mint például megválasztjuk a neuronok számát, rejtett rétegek számát, kiválasztjuk az aktivációs függvényt, és a tanítási módszert és a kezdeti súlymátrix értékeket. Ezek után már csak a tanítás és a tesztelés van hátra, addig, amíg a hálózat a kívánt viselkedést nem mutatja.

2.2.11. Aktivációs függvények

Mint említve volt a neurális hálózatoknál a neuronok közti aktiválódása a két neuron közti aktivációs függvénytől függ. Az aktivációs függvény meghatározza, hogy milyen a bemenet súlyozott szumma érték esetén aktiválódik két neuron között a kapcsolat [16] illetve, hogy ezt az értéket milyen kimenetként továbbítja. Általában az aktivációs függvények kétfélék lehetnek, lineárisak vagy nem lineárisak, a függvény alakjától függően. Attól függ, hogy milyen aktivációs függvényt választunk, hogy milyen feladatkört szeretnénk megoldani. Általában más aktivációs függvényeket használnak objektumdetektálásra mint megerősítéses tanulásra, átfedés természetesen lehetséges. A legjobb módszer a próbálgatás ebben az esetben is. Természetesen vannak tendenciák, ahol látható, hogy az egyik terület, például kép osztályozás során, melyek a bevált aktivációs függvények. Az alábbi függvények felmerültek megerősítéses tanulás témakörében, mint lehetséges aktivációs függvény jelöltek. A megfelelő eredmény érdekében többféle függvényt is érdemes kipróbálni.[11]

A tanh függvény

A tanh függvény vagyis a tangens hiperbolikusz függvény az alábbi ábrán látható.

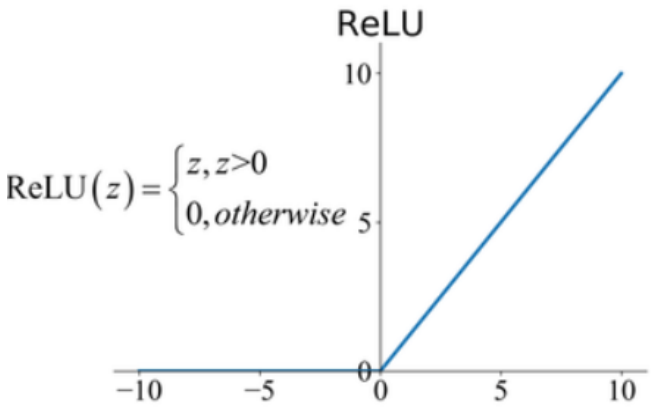


2.3. ábra.
Tanh aktivációs függvény (forrás: [17])

Mint látható a tanh függvényt -1 és 1 közötti értékekre normalizálja a bemeneti értékeket, azaz a bemenet súlyozott szumma értéket. A kutatások azt mutatják, hogy a tanh függvény több rejtett réteggel rendelkező neurális hálózatok esetében jól teljesít, példádul a sima szigmoid függvényhez képest [16]. A tanh aktivációs függvény alkalmazása, rekurrens neurális hálózatoknál releváns lehet [16].

A ReLU függvény

A ReLU függvény az egyik leginkább használtabb aktivációs függvény napjainkban [16]. A ReLU aktivációs függvény az alábbi ábrán látható.

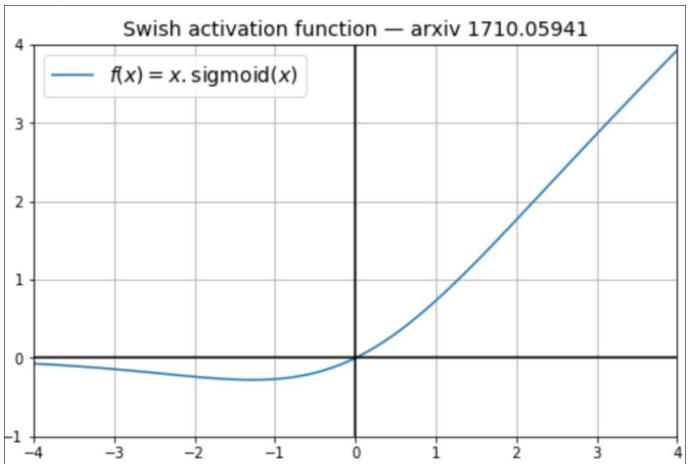


2.4. ábra.
ReLU aktivációs függvény (forrás: [17])

Ahogy az ábrán is látható a ReLU függvény a 0 alatti értékeket ugyancsak 0 ként továbbítja, azaz nem aktiválódik a két neuron közti kapcsolat. Az 0-nál nagyobb értékeket pedig képzi le. A ReLU lineáris leképezése miatt, gyorsabb tanítási gyorsaságot lehet elérni, hiszen a lineáris kimenetet kevesebb számítási igénnyel rendelkezik, mint például a tanh vagy a szigmoid függvény.

A Swish függvény

A Swish függvény a Google Brain csapata által publikált [18] aktivációs függvényt. Céljuk egy a ReLU aktivációs függvényt helyettesítő függvény keresése volt.



2.5. ábra.
Swish aktivációs függvény (forrás: [19])

Ahogy láthatjuk, a Swish 0 alatt, a ReLU-tól eltérően aktiválódik és visz át értéket, mínusz végtelen irányában 0 felé konvergál. 0 érték felett a Swish a ReLU-hoz hasonlóan

szigorúan monoton növekvő, lineárishoz hasonló értékeket visz tovább. A Swish alsó határral rendelkezik, viszont felső határral nem [18].
A Google Brain csapata kísérleteik során azt találta, hogy a Swish a ReLU-val szemben legalább azonos, több feladatban pedig jobban teljesített [18].

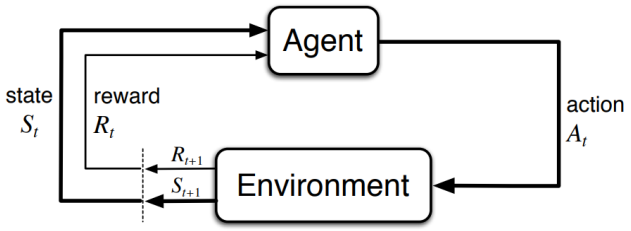
2.3. Megerősítéses tanulás

A megerősítéses tanulás egy rész területe a gépi tanulás témakörének. A következőkben a megerősítéses tanulásról általánosan, illetve a fentebbi feladat megoldásához specifikusan releváns irányból lesz szó.

2.3.1. Markov döntési folyamat

A megerősítéses tanulás probléma lemodellezésére használatos megközelítés a Markov döntési folyamat [20]. A fő komponensek, amelyeket a modell kiemel a következők:

- Állapot tér: a környezet lehetséges állapotaiból álló tér
- Cselekvés tér: az ágens által hozható lehetséges cselekvésekből álló tér
- Jutalom függvény: az ágens által, egy adott állapotban hozott cselekvéséért kapott jósági érték, amellyel egy különböző állapotba került
- Átmeneti függvény: az ágens által, egy adott állapotban hozott cselekvéssel egy különböző állapotba kerülés valószínűsége



2.6. ábra.
Az ágens és környezet interakciója (forrás: [20])

A modell szerint két fő része van a megerősítéses tanulásnak. A környezet és az ágens. Az ágens a cselekvő, vizsgálja a környezetet és döntéseket hoz. Az ágens lehetséges cselekvéseit az "action space" vagy cselekvés tér írja le. Ez definiálja mindazon cselekvéseket,

amelyet az ágens elkövethet. A következő aspektus az "observation space" vagy a megfigyelési tér. Ez a tér definiálja, hogy az adott ágens mit lát a körülötte lévő környezetből. Azaz a környezetnek mik azok a részletei, amelyek rendelkezésre állnak számára az adott döntés meghozatalához. Az ágensnek a célja a jutalom maximalizálása a tevékenysége során [20].
A modell egy rugalmas keretet ad feladatok megoldásához. A cselekvési térben definiált cselekvések lehetnek akár alacsony szintű áramköri interakciók és magasabb szintű összetett funkciók, folyamatok is. Az állapot tér reprezentációja ennek megfelelően lehet egyszerű, illetve összetett. A modell ezzel nem foglalkozik, csak azzal a ténnyel, hogy véges számú legyen az elemszáma a fent említett tereknek [20].
A Markov döntési folyamat (MDP) által biztosított absztrakció segít a feladat felbontásában, és könnyebb értelmezésében. Ezt a sablont ráhúzva a problémánkra egyszerűbben átláthatjuk azt. Egy lehetséges megközelítése ennek megfelelően, a közlekedési lámpák irányításának megerősítéses tanulást használó ágenssel, az alábbi lehet az MDP-modell szerint:

- Állapot tér: a közlekedésben résztvevő autók pozíciója egy adott pillanatban, az ágens alatt operáló közlekedési lámpák, ezen közlekedési lámpák aktuális állapota
- Cselekvési tér: az ágens képes minden egyes közlekedési lámpának a színét megváltoztatni pirosról zöldre,és zöldről pirosra (köztes sárga jelzéssel)
- Jutalom függvény: az állapottér adott pillanatában megvizsgáljuk az egy helyben álló autók számát, majd jutalmazzuk ha ez alacsony
- Átmeneti függvény: az adott döntés valószínűsége

Egy következő szempont pedig a környezetről való ismerete az ágensnek. Lehetséges, hogy az adott ágens mindent tud a környezetről, illetve lehetséges, hogy csak egy rész egységét ismeri annak. A feladat nehézsége, a környezetről való ismeretétől független az ágensnek, gondoljunk itt egy sakk feladat megoldására [20]. A közlekedési lámpák irányítása szempontjából, kecsegtető lehet, a környezetről minden létező információt odaadni az ágensnek. Ez megoldható lenne hiszen egy szimulációs környezetben számtalan metrikát kilehet nyerni a forgalomról. Ennek ellenére preferálandó, hogy a valós világban való esetleges implementálás érdekében, csak olyan információkat szabad a környezetről adni az ágensnek, amelyeket ténylegesen meg is tud kapni arról. Itt konkrétan egy jármű detektáló komponensre gondolok, amely a járművek számát, illetve pozícióját tudja megadni ideálisan az ágensnek [20].

2.3.2. Optimalizációs algoritmusok

Az ágens ahhoz, hogy optimális döntéseket tudjon hozni egy adott állapotban, szükség van valamilyen számértékre, amely meghatározza a legjobb cselekvést egy adott állapotban. A terület kutatása során több módszert is találtak arra, hogy minél megfelelőbb döntéseket tudjanak hozni az ágensek. Általában a szempont ezeknek az algoritmusoknak a megválasztásánál a megoldandó probléma természete. A megerősítéses tanulás területén beszélünk model-free és model-based megközelítésekről. [20]
A model-based megközelítésben a jutalom függvény, illetve az átmeneti függvény ismertek előre. Ez esetben, az ágens a környezet megfigyelése után be tud helyettesíteni a fentebbi két függvénybe és megtudja határozni az optimális döntést.
Egy másik megközelítés a model-free koncepció. Ebben az esetben a legjobb cselekvést tapasztalatból becsüli meg az ágens. [20]
A következőkben megvizsgáljuk a különféle optimalizációs algoritmusokat, amelyek relevánsak lehetnek:

- **Q-Learning**:

  Az optimális döntés meghatározásához az utóbbi időben a Q-Learning megközelítést alkalmazták a megerősítéses tanulás területén[21]. A cselekvés meghatározásához nincs szüksége a környezet ismeretéhez, hanem attól függetlenül próbálja megbecsülni a Q értéket. Az algoritmus hozzárendel minden egyes cselekvés-állapot párhoz egy Q értéket. Az algoritmus a működése során felépít egy táblázatot az adott cselekvés és állapot párokból, majd ezeket folyamatosan frissíti. Mivel előre nem tudható a kiszámításra kerülő érték, ezért a jutalom függvény segítségével közelíti meg a valós értékét ennek a Q értéknek. A Q-Learning algoritmus által becsült érték a végtelenben konvergál a valós érték felé [21].

- **DQN**:

  A Deep Q-Learning egy a Q-Learninget továbbfejlesztő megközelítés, amelyben az eredetileg táblázatot használó Q-Learning algoritmus által előállított Q értéket neurális hálózattal közelítik meg [22]. A diszkrét cselekvés és állapot terek helyett, sok probléma folytonos tereket definiál. Ennek megfelelően a diszkrét cselekvés és állapot párok segítségével létrehozott táblázat helyett, függvény közelítésre van szükség. A függvény közelítésére alkalmazható bármilyen grádiens módszer, viszont a neurális hálózatok nagyon effektívnek bizonyultak ezen a területen is. A függvény közelítő módszerek divergencia problémája a neurális hálózatok alkal-

mazásakor is előjön. Erre viszont a DQN (Deep Q-Learning) algoritmus megalkotásakor egy, már az IntelliLight projektben is említett módszerrel, az experience replay-el álltak elő.
Az experience replay módszerével egy adatbázist állít elő [23] az algoritmus, amely tuple-kként tartalmazza a régebbi, a tanulás során már tapasztalt cselekvés, állapot, jutalom, átmenet és új állapot paramétereket. A tanulás során egy fix méretű, FIFO elvű bufferben ezeket eltárolják és néha újra lefuttatják őket. Így az esetleges oszcilláció vagy divergencia esélyét minimalizálják.

- **PPO**:

  A következő vizsgált módszer a PPO, avagy a Proximal Policy Optimization [24] algoritmus. A PPO ugyancsak egy megerősítéses tanulásra használt algoritmus, amely az utóbbi időben terjedt el. A PPO-t az OpenAI [25] csapata készítette. A Q-Learning algoritmusokkal szemben a Policy Gradient algoritmusok célja nem a Q függvény közelítése és a Q érték becslése, hanem a cselekvési tér "policy-jének", irányelvének az optimalizálása. Policy optimalizálás során az ágens cselekvései által előállított állapot, akció, jutalom, eredmény állapot tuple-t kapjuk. A célunk, a neurális hálózat paramétereinek a finomhangolása olyan módon, hogy a nagyobb jutalommal járó cselekvések nagyobb valószínűséggel következzenek be a jövőben. Az algoritmus különböző hiperparaméterekkel rendelkezik, amelyeket a megfelelő tanulás mérték eléréséhez finomhangolni kell. A Policy Gradient módszer célja természetesen a hosszú távú maximum jutalom elérése, hasonlóan a legtöbb megerősítéses tanulásnál használatos algoritmusnak.

2.4. Összegzés

Az irodalomkutatás fejezetből az alábbi konklúziók kiemelendőek. A hasonló rendszerek fejezet alapján több tervezési döntést is meg lett hozva. Az egyik ilyen, hogy a szimulációs szoftverrel való integráció fontos része lesz a fejlesztésnek. Továbbá, hogy olyan szimulációs környezetet kell választani, amely eléggé rugalmas, megfelelő mennyiségű adattal tud szolgálni a megerősítéses tanulást használó ágens számára.
A neurális hálózatok témakörének a kutatása elengedhetetlen a feladat megoldásához. A fentebb leírt információk fényében előreláthatóan a fejlesztés során sok kísérletezés és paraméter finomhangolás várható. A neurális hálózat modellje, a hálózatban használt aktivációs függvények megválasztása jelentős kihívásnak tűnik.
A megerősítéses tanulás szempontjából az ágens percepciója a környezetéről, illetve szi-

mulációs környezettel való interakciója mindenféleképpen kulcs szempont lesz a fejlesztés során. A környezetről való ismerete az ágensnek, preferálandó, a valós világban is megvalósítható mértékű kell, hogy legyen. Ennek megfelelően a parciális láthatóság elve szerint, nem teljes képet kell adni a szimulációról az ágensnek, hanem részlegeset. Itt lehet gondolni néhány autóra akár, ezeknek a pozíciójára. Továbbá a tanítás során használt megerősítéses tanulás algoritmusa is fontos lesz. Itt a fent említett DQN, PPO és további algoritmusok tesztelése is szempont lesz. Az utóbbi időben a PPO algoritmus ért el nagyon jó eredményeket különböző alkalmazások során. A PPO algoritmus felparaméterezése, illetve a neurális hálózat modelljének meghatározása sok teszteléssel fog járni.

3. KÖVETELMÉNY SPECIFIKÁCIÓ

Az irodalomkutatás során szerzett ismeretek alapján az alábbiakban olvasható a projekt követelmény specifikációja, azaz, hogy mik az elvárások az elkészült rendszertől. A kezdetleges rendszerterv, melynek komponenseit és ezek feladatát, illetve elvárt működését az alábbiakban lesznek kifejtve.

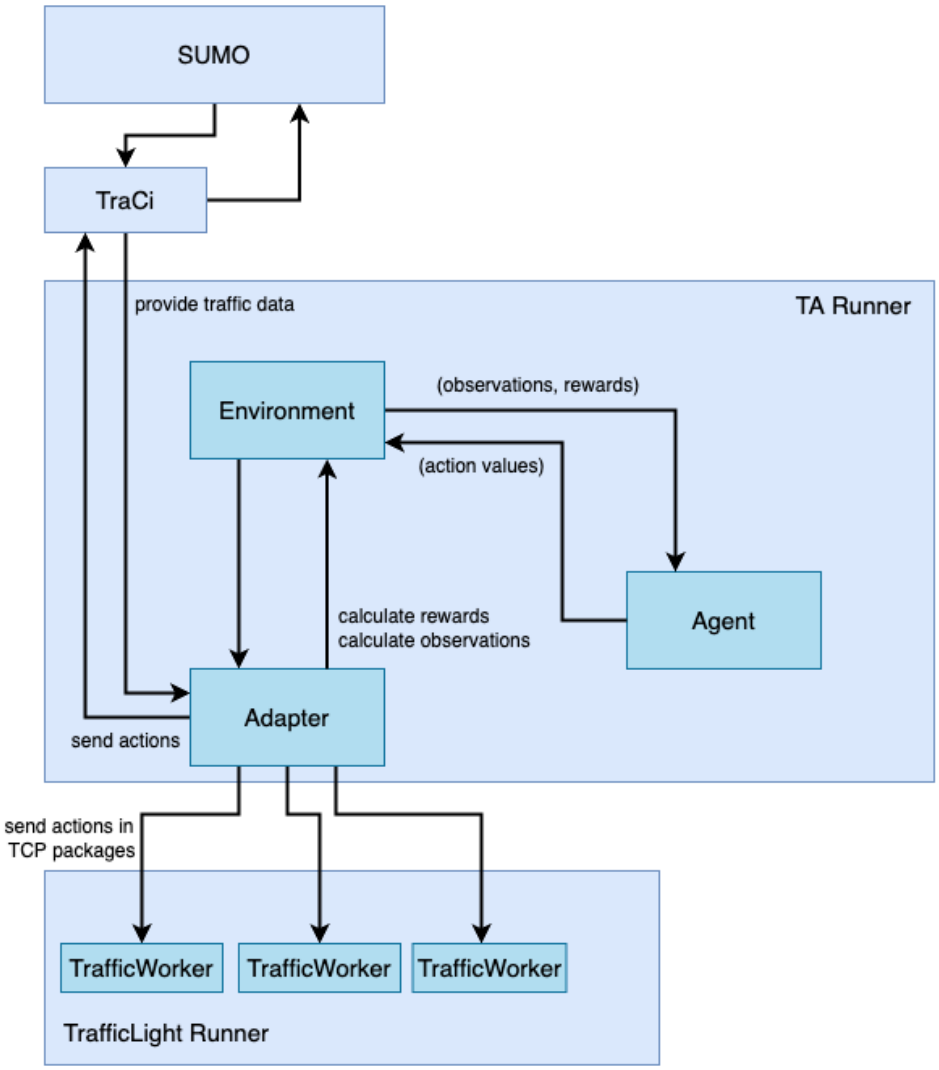
Az ágens feladata, minél jobb eredményt elérni az elé kerülő feladat esetében, ezért a tanítás során az ágens jutalmazásának alapja, egyben a cselekvéseinek jósági metrikájából lesz levezetve. Egy ilyen metrika lehetne például a járműkésés, amivel mérni lehet az ágens által meghozott döntések jóságát. A fentebb említett szimulációs környezet és neurális hálók mind-mind Python környezetben lesznek megvalósítva.

A rendszernek képesnek kell lennie egy szimulációt futtatni, az ágensnek ebben kell tevékenykednie. A szimulációs környezetből különböző metrikákat kell kinyerni, majd az ágensnek ezen metrikák alapján bemenetet formálni. Meg kell valósítani továbbá a megerősítéses tanulást, amely segítségével az ágens tanulni fog. A megerősítéses tanulást paraméterezhetően kell implementálni a rendszerben, annak érdekében, hogy rugalmasan lehessen tesztelni többféle megoldást. Az ágens tanulását a kísérletek végén ki kell tudni értékelni azaz értékelhető eredményekkel kell szolgálnia a rendszernek.

4. TERVEZÉS

4.1. A megerősítéses tanulást használó ágens rendszerterve

Ebben a fejezetben kerül tárgyalásra az összeállítandó rendszer terve. Megnevezésre kerül a szimulációs környezet, a megerősítéses tanulás implementációjához használt könyvtárak és ezek egymással való interakciójának a bemutatása fog megtörténni.
Az alábbi ábrán látható a megerősítéses tanulást használó ágens rendszerterve. Az egyes komponensek egymással való interakciói részletesebben kerülnek tárgyalásra a további albekezdésekben.



4.1. ábra.
A projekt rendszerterve. (Saját készítésű ábra.)

4.2. A komponensek egymással való interakciói

A fenti terven látható, ahogy több komponens egymással interakcióba lép. A SUMO szimulátor fogja futtatni a szimulációt melyből, a TraCI interfész segítsével lesz lehetőség kinyerni adatokat a közlekedés állapotáról. Ezeket az adatokat az Adapter komponens fogja feldolgozni majd a Jutalom és Állapot tér értékeket a szolgáltatni az OpenAI Gym specifkikus Environment részére. Az Adapter azon túl, hogy a SUMO szimulátornak képes parancsokat küldeni az ágens aktuális döntéseiről, képesnek kell lennie hálózati kapcsolaton keresztül adatokat továbbítania a bárhol futó lámpa csomópont folyamatok számára.
Az ágens és a megerősítéses tanulás implementációja az OpenAI Gym [26] és a stable-baselines3 [27] csomag segítségével lesz megvalósítva. Az ágens számára az OpenAI Gym fogja adni az absztrakciót a megerősítéses tanuláshoz. Ezek az absztrakciók például az Observation space vagy Állapot tér, az Action space azaz Cselekvési tér vagy éppen, hogy mely függvényt kell meghívnia a modellnek a Jutalom érték kiszámításához.

4.3. Használati esetek

Az alábbi használati eseteket kell megfontolni a tervezés, illetve fejlesztés során:

- A SUMO szimulációs környezet segítségével új modellt tanítunk be.
- A SUMO szimulációs környezet segítségével már meglévő modell tanítását folytatjuk.
- A SUMO szimulációs környezetet, mint bemeneti adatforrás és kimeneti cél használva, de már meglévő modellt, annak betöltése után használunk.
- A SUMO szimulációs környezetet, mint bemeneti adatforrás és kimeneti cél használva, továbbá hálózati közlekedési csomópontokat, mint kimeneti cél, de már meglévő modellt, annak betöltése után használunk.

4.4. A rendszer funkciólistája

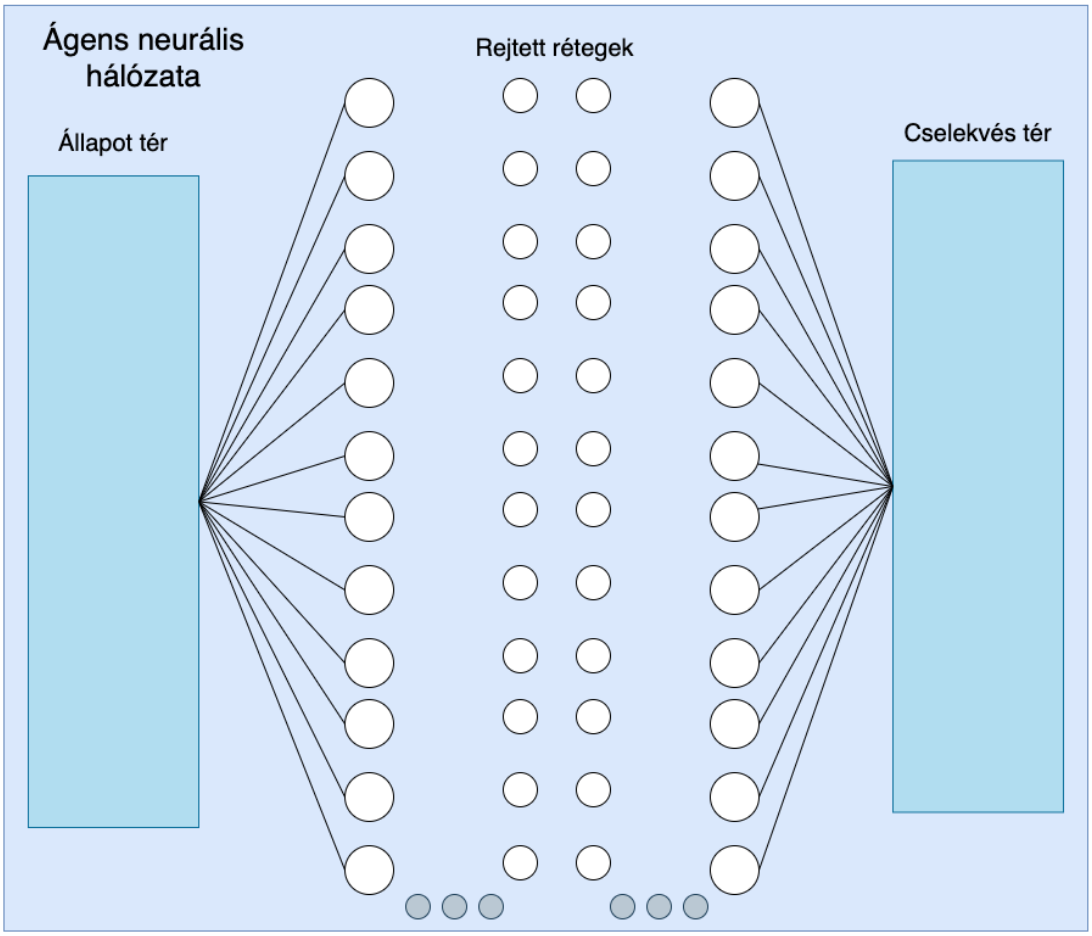
Az alábbi felsorolásban látható a funkciólista melyet az elkészítendő rendszer céloz kielégíteni:

- Szimuláció és a modell felparaméterezése és elindítása.

- Az ágens interakciója, a szimulációs környezettel.
- Szimuláció vizuális megjelenítése.
- Megerősítéses tanulás ágens interakciója, a szimulációs környezettel.
- Közlekedési adatok kinyerése a szimulációból és azok felhasználása.
- Közlekedési csomópontok workereinek elindítása aszinkronos módon.
- Tanított modell elmentése, majd újonnani betöltése.
- Mentett modell kiértékelése, cselekvések továbbítása a közlekedési csomópontok felé TCP kapcsolat segítségével.
- Kísérlet eredményeinek elmentése és megjelenítése.

A rendszernek képesnek kell lennie a SUMO környezettel interakciókat végezni, mint például a kívánt úthálózattal és járművekkel, azok viselkedésével elindítani kísérleteket. Erre azért van szükség, hogy az ágens a tanítása során a paramétereknek megfelelően el-tudja indítani a kísérletet, letudja azt állítani, a szükséges adatokat képes legyen kinyerni, majd a cselekvéseit végrehajtani. Ezekre a TraCI könyvtár lesz segítségül, ezen keresztül fogják a rendszer különféle komponensei elérni a szimulációs környezetet.
Annak érdekében, hogy a könnyedén lehessen váltani a szimulációs környezettel és a közlekedési csomópontokkal való interakciók között a tanított modell használata esetében szükséges, hogy az implementáció során laza kötés legyen alkalmazva. Ez a kód ismétlés elkerülése érdekében is fontos, továbbá, hogy könnyedén lehessen bővíteni másféle cél implementációkkal a közlekedési lámpák komponensét.

4.5. Ágens neurális hálózata



4.2. ábra.
Az ágens neurális hálójának felépítése. (Saját készítésű ábra.)

Az fenti ábrán látható az ágens neurális hálózatának felépítésének a terve. Az állapot tér egy mátrix vagy vektor, amely tartalmazza a környezetről az információkat, normalizálva számértékként. Ilyen adat lehet például az autók száma, bizonyos autók helyzete [28]. Ez a bemenete a neurális hálózatnak. A rejtett rétegek és a neuronok száma bizonytalan de rugalmasan változtatható paraméter. A használandó aktivációs függvény is fontos paraméter lesz a tanítás során. A cselekvési tér az a kimenet, amely alapján a módosítva lesz később a szimulációs környezet [20]. Ez ugyancsak egy vektor/mátrix lesz, amely tartalmazza, hogy melyik közlekedési lámpának milyen állapota legyen.

4.6. Ágens interakciói

Ahogy, a nagy rendszertervet bemutató ábárn látható 4.1, hogy a SUMO környezet szolgáltatja a közlekedési metrikákat az ágens számára. A TraCI könyvtár segítségével Python nyelven lekérhetőek olyan segédadatok, mint az autók száma, autók pozíciója. Természetesen az úthálózattól függően másképp kell interakciót folytatni a környezettel. Illetve a kísérlettől függően változtatni kell a metrikákat, amelyeket felhasználunk.
Az ágens, miután meghatározta a következő lépését, a SUMO számára átadja a végrehajtandó cselekvéseket adja. Ez implementáció szinten azt jelenti, hogy a neurális hálózat kimeneti vektorát értelmezni kell és a közlekedési lámpák állapotát kell megváltoztatni az értékeknek megfelelően.
Továbbá a tanuláshoz szükséges lépések közé tartozik, hogy az ágens a stable-baselines3 felé szolgáltatja a jutalom értéket és azt az állapot teret, amelyet kiszámolt a környezet megvizsgálása után. A stable-baselines3 ezután a meghatározott neurális hálózat modell és optimalizáló algoritmusnak megfelelően visszaad cselekvés kimeneteket, illetve a kalkulált jutalom értékeket [27], továbbá ha esedékes elvégzi a neurális hálózat értékein való finomhangolási műveleteket.

4.7. Jutalom függvény

A jutalom függvény definíciója a TraCI és a SUMO komponensek segítségével történik meg. A SUMO környezet futtatja a szimulációt és szolgáltatja a TraCi segítségével az adatokat. A TraCI [29] könyvtár tartalmazza, azokat a segédfüggvényeket, amelyek hozzáférést biztosítanak a kísérletben résztvevő objektumok állapotához (például járművek). Ezen osztályok segítségével könnyedén lehetőségünk van különféle metrikákat kinyerni az éppen futó kísérletünkből. Majd ezeket felhasználva jutalom függvényeket tudunk definiálni. A kísérletben résztvevő autók állapotának felhasználásával olyan metrikákat lehet kiszámítani mint szumma késés, egy helyben álló autók száma, vagy akár az aktuális sebességük.

4.8. Hálózati kapcsolat a közlekedési csomópontokkal

Mivel a hossszútávú cél az, hogy ne csak szimulációs környezettel tudjon az ágens kommunikálni, ezért szükséges felfedezni a lehetőségét esetleges hálózaton elérhető távoli közlekedési csomópontokkal való interakcióknak.
Ennek megfelelően szükséges lesz implementálni a közlekedési csomópont szolgáltatáso-

kat. A követelmény az, hogy folyamatos kapcsolatot tartson fent a közlekedési csomópont és a modell döntéseit továbbító Adapter komponens között. A dizájn döntés ettől fogva a socket alapú hálózati kommunikációval kézenfekvő. Ezt természetesen mindkét oldalon támogatni szükséges.

4.9. Eredmények nyomonkövetésének a módja

A kísérlet kiértékelése fontos szempont. A stable-baselines3 segítségével futás közben is tudjuk a kísérlet eredményeit figyelni, illetve miután vége a kísérletnek Tensorboard [9] segítségével megtudjuk nézni az eredményeket. Itt a tanítás szempontjából több fontos metrika is lehet, például a jutalom függvény értékének a változása.

4.10. Külső komponensek jellemzése

- **SUMO: Szimulációs környezet**:
  A SUMO [30] egy nyílt forráskódú szoftver, amely segítségével a közlekedés különböző aspektusait lehet modellezni és vizsgálni. A SUMO alkalmas nagy úthálózatok elkészítésére és ebben különböző közlekedési helyzetek lemodellezni. A szimulációs szoftverben a közlekedés legtöbb szereplőjét lehetőségünk van szimulálni, például: járművek, gyalogosok, közlekedési lámpák. A SUMO alapértelmezetten tartalmaz többféle támogató modult, mint például a károsanyag-kibocsátás nyomon követése, beépített jármú irányítás és jelzőlámpa irányítás. A jármű vezérlés, illetve a jelzőlámpa irányítása teljes mértékben módosítható és állítható. Mind a járművek sebessége, kezdeti pozíciója paraméterezhető. A jelzőlámpák irányítása ugyancsak a kezünkben lehet, a fázisváltás és a fázisterv módosítására is lehetőséget ad a SUMO. A szimulátor modularitását mutatja, hogy akár többféle szimuláció futtatására is alkalmas például: gyalogos, biciklis, vasutas és vízi közlekedésre. A szoftver ezen túl beépített GUI interfésszel rendelkezik, amely segítségével az elkészített szimulációnkat a szemünk előtt láthatjuk.

- **stable-baselines3**:
  A stable-baselines3 egy könyvtár a megerősítéses tanulás implementálásához [27]. Alapvetően a megerősítéses tanuláshoz szükséges optimalizációs algoritmusok implementációját tartalmazza a csomag. Ilyen algoritmus például a PPO vagy a DQN. A könyvtár fejlesztői az implementáció során, ahol lehetséges, a hivatalos publikációt alapulvéve cselekedtek. A stable-baselines3 csomag emellett az alapvető abszt-

rakciókat is megvalósítja, mint modell mentésével járó IO műveletek, illetve logolás alapvető funkcionalitásai. Ezentúl az OpenAI Gym Environment standardizált definícióját is támogatja. A megerősítéses tanuláshoz szükséges hiperparaméter-finomhangolás, illetve a modell kísérletre szabása is könnyedén véghezvihető.

- **OpenAI Gym**:
  A OpenAI Gym egy csomag, amelyet az OpenAI csapata implementált. A célja, hogy egy standardizált módszert adjon a megerősítéses tanulás implementálásához. Erre azért van szükség, hogy elkülöníthessük a megerősítéses tanulás Állapot tér, Cselekvési tér definícióját, amit a Gym környezetben leírunk, illetve az ettől független tényleges megerősítéses tanuláshoz alkalmazott optimalizáló algoritmusokat. Ezzel a megközelítéssel könnyedén lehet cserélni az optimalizáló algoritmusokat megvalósító könyvtárakat és csomagokat. A legtöbb esetben ezek a könyvtárak már alapvetően támogatják a Gym Environment implementációt.

- **TraCI: Traffic Control Interface**:
  A TraCI [29] egy, a SUMO-hoz készült interfész, amely segítségével a SUMO környezettel tudunk interakciókat végezni. A SUMO-ban elkészített szimulációinkkal a TraCI segítségével adatokat lehet kinyerni, különböző interakciókat végezni. A TraCI TCP, csomagkapcsolt kommunikációval tud a SUMO környezettel interakciókat végezni, így akár a több, párhuzamosan futó szimulációval is kapcsolatot tud tartani. A TraCI segítségével "kintről" tudunk olyan parancsokat adni a szimulációnak, mint a: válts lámpát, válts sávot. Emellett a TraCI segítségével tudunk fontos metrikákat kapni a szimulációnkból, mint például: sebesség, mind-mind nélkülözhetetlenek lesznek ahhoz, hogy saját vezérlést tudjunk kapcsolni a jelzőlámpáknak, továbbá, hogy a megerősítéses tanuláshoz megfelelő mennyiségű adatot tudjunk az ágens számára biztosítani.

- **Conda**:
  A Conda egy nyílt forráskódú virtuális környezet menedzselő szoftver [31]. A Conda segítségével képesek vagyunk Python csomagjainkat és függőségeinket letölteni, kezelni, frissíteni és ami még fontosabb, különálló virtuális környezetet tudunk létrehozni. Ez azért fontos, hogy a különböző Python verziókat, az azokkal járó csomag verziókat könnyedén lehessen elkülöníteni a projektjeink szükségleteinek megfelelően.

5. FEJLESZTÉS

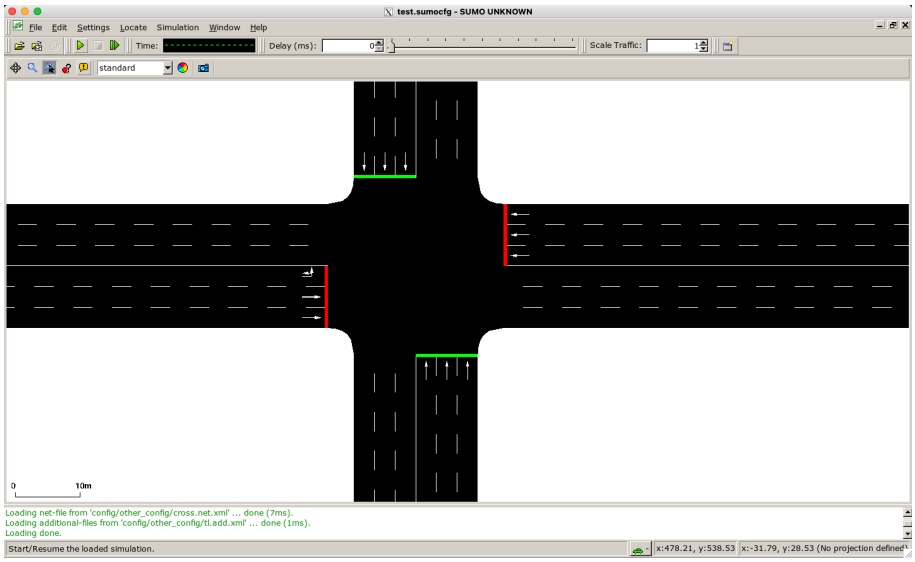
5.1. Kapcsolódás a SUMO szimulátorhoz és a váz létrehozása

Ezután a Python virtuális környezet létrehozása volt a feladat. Erre alapvetően a függőségek könnyebb kezelésére van szükség. A projekt a Python 3.10.7-es verziójával lett implementálva és validálva.

Csomag:verzió	Leírás
stable-baselines3:1.4.0	Megerősítéses tanulás
torch:1.13.0.dev20220622	Neurális hálózatok
sumolib:1.13.0	A SUMO szimulációs környezettel való kommunikációhoz szükséges
traci:1.13.0	A SUMO szimulációs környezettel való kommunikációhoz szükséges
tensorboard:2.9.1	A tanítás megjelenítéséhez szükséges csomag
numpy:1.22.4	Különféle matematikai műveletekhez szükséges csomag

5.1. táblázat.
Táblázat a csomagokról

A függőségek feltelepítése után a következő lépés az volt, hogy programkód segítségével lehessen a SUMO környezetet elindítani. Ehhez szükséges volt valamiféle úthálózat összeállítása, amelyet a SUMO-nak átadhatunk. Általánosan a cél, hogy egy egyszerű kereszteződésben tudjunk tevékenykedni az ágens segítségével.



5.1. ábra.
Az általunk használt csomópont

A fenti konfiguráció nem csak a vizuális paramétereit határozza meg az úthálózatnak, hanem az útsávokat, hogy melyik sávból melyikbe lehet kanyarodni, hogy melyik csomóponton hány lámpa van, azon lámpák lehetséges fázisai egymáshoz képest. Az úthálózat elemein kívül a járművek viselkedését is itt kell, hogy meghatározzuk. A szimulátor a konfigurációtól függően fogja elhelyezni, a jelenlegi konfigurációban valószínűségek alapján, hogy melyik sávra, melyik irányból miként helyezzen el járműveket. Az járművek vezérlésére egy egymást követő vezérlő lett választva, ahol az autók nem térnek le az útvonalukról. Természetesen a lámpák jelzését figyelik, a piros jelzésnél megállnak, a zöldnél elindulnak.
Ahhoz, hogy ellehessen indítani a szimulációkat, a konfigurációs fájlokon kívül a SUMO futtatható binárisára van szükség. A rendszer egyaránt fut MacOS és Windows környezetben is, illetve a SUMO szimulátor is egyaránt létezik mindkettő platformon, így megfelelő elérési út megadása után futtatható a projekt.
Miután ez megvan, attól függően, hogy grafikus vagy háttérben futó verzióban indult el a SUMO, vár a következő parancsokra. Ennek megfelelően egységnyi idő alatt ezek a lépések lesznek a megerősítéses tanulás támogatására:

1. Ha van, akkor a megkapott cselekvések végrehajtása
2. Következő lépés utasítás
3. Állapot tér lekérése
4. Állapot tér kiértékelése
5. Jutalom értékek kiszámítása
6. Cselekvések küldése

Ezek a lépések természetesen addig ismétlődnek ciklikusan, amennyi ideig szeretnénk, hogy egy epizód tartson. Például egy epizód 1000 lépésből áll, egy tanítási folyamat pedig 1000 epizód futtatásából áll. Ezeket a viselkedéseket két ciklussal könnyedén lehet implementálni. Természetesen, hogy a kód átlátható és moduláris legyen megfelelő osztályokba szükséges, a fenti funkciókat is implementálni.
A fenti folyamat megadja az alap szerkezetét a megerősítéses tanulás implementálásához.

5.2. Megerősítéses tanulás integrációja a szimulációs környezettel

A következő kihívás maga a megerősítéses tanítás implementációja volt. Itt fel kellett ismerni, hogy használati esetei lehetnek egy ilyen programnak. Egyfelől szükséges, hogy

szimulációt tudjon elindítani és egy új modellt tanítani, másfelől szükséges, hogy a már meglévő betanított modellt betudja tölteni és mint egy irányító logikát használni. Ez a két különböző használati eset, amelyet támogatni kell alapvetően.
Egy, a megerősítéses tanulás támogatására széles körben használt csomag és absztrakció az OpenAI csapat által fejlesztett OpenAI Gym. A Gym segítségével standardizált interfésszel rendelkezünk, amelyet sok megerősítéses tanulást implementáló csomag, többek között a stable-baselines3 is támogat.
Az OpenAI Gym csomag egy Environment osztályt ad, amelyből saját osztályunkkal szükséges leszármazni. Itt felülírva egyes kulcs metódusokat, tudunk ténylegesen a saját szükségleteinknek megfelelően implementálni olyan paramétereket és jellemzőket, mint például:

- Milyen dimenziókkal és értékekkel rendelkezzen az Állapot tér és a Cselekvési tér?
- Mi történjen, ha vége egy epizódnak?
- Milyen jutalom függvény hívódjon meg?
- Mi történjen egy lépés során?

Ezek természetesen azok a paraméterek is, amelyeket általában finomhangolni kell a megerősítéses tanulás során. Különféle állapot tér definíciók, különféle jutalom függvények kipróbálása szükséges a kísérletek során.
Alapvető kérdésekre már most is tudunk viszont válaszolni. Tudjuk, hogy a jutalom értéket szeretnénk nullára állítani a következő epizódra, illetve, hogy majd az Állapot teret implementáló mátrixot is szeretnénk kinullázni. Továbbá tudjuk, hogy minden egyes lépés során szükséges a fentebb már leírt lépések véghezvitele. Ennek az implementációja alább látható:

```
def step(self, action):

    self.make_action(action)
    self.traci.simulationStep()

    observation = self.observe()

    additional = self._calculate_reward_(self.traci.simulation.getTime())
    self.reward = additional

    self.done = False
    info={}
    return observation, self.reward, self.done, info
```

5.2. ábra.
Egy lépés során végzett műveletek (Saját készítésű ábra.)

Itt természetesen a részletei nem láthatóak a Jutalom érték számításának illetve annak se, hogy miként lett az Állapot tér felépítve. Ezek a későbbi fejezetben lesznek tárgyalva.
Mint már említve volt a stable-baselines3 támogatja az OpenAI Gym interfészét, ami segítségével a modell a számára fontos értékeket, mint az Állapot tér, Jutalom értékek megtudja szerezni. Egy modell objektumot többféleképpen lehetséges létrehozni, de minden esetben szükséges az adott optimalizációs algoritmus kijelölése, például: PPO, DQN, illetve a Gym Environment objektum átadása. Az egyéb, általában algoritmus függő hiperparaméterek többnyire rendelkeznek alaphelyzetbe konfigurált értékekkel. Az algoritmus hiperparamétereiről később fog szó esni.
Továbbá kimeneti könyvtárat is tudunk állítani ahová a Tensorboard számára, ahova tanítás során menteni fogja a modell a metrikákat. A modellnek továbbá átadhatjuk, hogy hány darab lépésből állnak az epizódok, így nekünk már csak epizódok ciklikus elindítását kell implementálni.
A tanítás során lehetőségünk van egyfajta mérföldkőről mentéseket készíteni a modell aktuális állapotáról. Erre azért van szükség mert a megerősítéses tanulás egy hosszú, időigényes, nem determinisztikus folyamat, ahol könnyedén előfordulhat, hogy a tanítás az 1500. epizódjától kezdve mélyrepülésbe kezdtek menni a jutalom értékek és mi a csúcspontot elérő állapotról szeretnénk kapni a modellről egy verziót.

5.3. Komponens alapú megközelítés a fejlesztésben

Annak érdekében, hogy a könnyedén lehessen a megerősítéses tanulással kísérletezni és dinamikusan fenntartható legyen a kódbázis, szükséges előrelátónak lenni az implementáció során. Ez az oka annak is, hogy a Jutalom függvény kiszámítás, a Cselekvések végrehajtása, az Állapot tér felmérése más helyen történik, nem pedig a Gym Environment osztályában.
A fenti, teljes rendszer ábrán látható az Adapter komponens. Az Adapter komponens, mint egy központi egység áll a különböző egységek között. Az Adapter az komponens, amely meghatározza, hogy milyen implementáció-specikus módon kell az Állapot teret felépíteni a Gym Environment-nek, hogy az, a számára értelmezhető legyen. Ez azt jelenti, hogy ha leszeretnénk cserélni a SUMO szimulációs környezetet egy másik szimulátorra, akkor elég lenne csupán egy másik implementációt készíteni számára az Adapter osztályból, a Gym Environment és maga a modell ebből pedig semmit nem érzékelne. Ez azért is fontos mivel nem nehéz elképzelni, hogy egy másik szimulátor az másféle interfészekkel rendelkezne, esetleg hosszabb, terjedelmesebb kóddal lehetne csak ugyanazt a

metrikát kinyerni belőle, mint a jelenlegi SUMO és TraCI párossal.
Továbbá egy másik komponens, ami fel is használja ezt a lehetőséget az a TrafficLight Runner. Ez a hálózaton elérhető közlekedési csomópont futtató alkalmazás, amely segítségével távoli független közlekedési lámpa csomópontokat tudunk imitálni és bemutatni vele, hogy miként lehetséges hálózaton keresztül megkapni a közlekedési parancsokat az Ágenstől. Erről a következő bekezdésben többet is meglehet tudni.

5.4. Hálózati kapcsolódás a közlekedési csomópont vezérlőihez

Fentebb már említve volt, az a használati eset, amikor már van egy meglévő modell és szeretnénk azt "élesben" használni. Azon túl, hogy ebben az esetben már nem tanítjuk az ágenst, hanem csak kiértékeléseket végzünk a segítségével, egy valós helyzetben már nem elégszünk meg azzal, hogy egy szimulátorban tudunk lámpákat irányítani vele. Ennek érdekében további cél lett egy olyan külső alkalmazás létrehozása, amely egy távoli hálózati végpontként működik. Ezzel a távoli végponttal az alap alkalmazás, socket alapú kommunikáció segítségével tud kommunikálni és továbbítani információkat az ágens aktuális cselekvés parancsairól. Ezeket a parancsokat feldolgozza az aktuális csomópont és az ottani konfigurációnak megfelelően változtatni a lámpák fázisait.

```
class TrafficCenter():
    def __init__(self, connections):
        self.connections = connections
        self.sockets = []
        for i in connections:
            self.sockets.append(socket.socket(socket.AF_INET, socket.SOCK_STREAM))

    def make_connection(self):
        counter = 0
        for i in self.connections:
            print(i)
            self.sockets[counter].connect(('localhost', i))
            print('connected to port: ', i)
            counter += 1

    def send_phase(self, action):
        counter = 0
        for i in self.connections:
            self.send_action(counter,str(action))
            counter += 1

    def send_action(self,counter, message):
        self.sockets[counter].sendall(bytes(message, 'utf-8'))
```

5.3. ábra.
Kapcsolódás a workerekhez és cselekvések küldése. (Saját készítésű ábra.)

Annak érdekében, hogy ez megvalósítható legyen, ennek a funkcionalitásnak mindkét ol-

dalon támogatva kell lennie. A komponens alapú fejlesztésnél már említve volt az Adapter osztály használata, amely annak érdekében lett létrehozva, hogy jobban szeparálódjon a használati eset és platform attól, hogy miként van tanítva az Ágens. Egy ilyen funkciónál egyértelműen erre van szükség. Egy tanított modellt kell használni, amely minden egyes cselekvésénél továbbítja a konfigurált végpontokra a modell cselekvéseit.
A közlekedési lámpákat futtató alkalmazás implementációnál az volt a cél, hogy könnyedén lehessen skálázni a szimulációban meghatározott számú lámpák számával. Ez azt jelenti, hogy ha 4 darab lámpacsoportot futtató szimulációval értékeltetjük ki a modellt, a másik oldalon 4 darab aszinkronos worker lesz, ami várni fogja a cselekvéseket. Négy darab worker esetén a lehetséges port számok például: 8000,8001,8002,8004.

```
class TrafficWorker():
    def __init__(self, traffic_light, port):
        self.traffic_light = traffic_light
        self.loop = asyncio.get_event_loop()
        self.port = port
        self.coro = asyncio.start_server(self.handle_client, 'localhost', port, loop=self.loop)
        self.server = self.loop.run_until_complete(self.coro)

    @asyncio.coroutine
    def handle_client(self,reader, writer):
        while True:
                data = yield from reader.read(100)
                if len(data) > 0:
                    message = data.decode()
                    addr = writer.get_extra_info('peername')
                    print("Worker %r:Received %r from %r" % (self.port,message, addr))
                else:
                    print("Close the client socket")
                    writer.close()
                    break
```

5.4. ábra.
Aszinkronos worker kódrészlete. (Saját készítésű ábra.)

A kódban lehet látni, hogy a worker 100 bájt adatot vár beolvasásra. Ezt az adatot beolvassa majd kiírja konzol felületre. A konzol felületre való kiírás mellett a traffic light objektum logikáját is fel lehet használni, a bejövő adatot kiértékelni és ennek megfelelően a jelzőlámpákat változtatni.

5.5. Állapot tér

Mint fentebb említve volt, ez a környezeti definíció adja meg, hogy az ágens mit lát a térből. A cél a környezet valamilyen feldolgozása és bemeneti adattá való formálása annak érdekében, hogy a neurális hálózat ezzel dolgozni tudjon [20]. Továbbá kívánt cél lenne, olyan metrikákat kell felhasználni, amelyet egy járműdetektáló komponens is

tud szolgáltatni. Ennek megfelelően olyan metrikák jönnek szóba, mint például az autók pozíciói, a lámpától való távolságuk, esetleg a sebességük.
Az állapot tér egy az OpenAI Gym-ben definiált típussal, a Box-al történt, a minimum értéke -5000 a maximum értéke pedig 5000, lebegőpontos értékek. Az adatszerkezet 2 oszlopból és 80 sorból áll. A 2 oszlop a járművek X és Y koordinátát hivatott jelölni, a 80 sor pedig a 80 járművet, amelyet a csomópontba érkező négy lámpával ellátott úton láthat az ágens. Ez lámpákként maximum 20 járművet jelent.

5.6. Cselekvési tér

A cselekvési tér, a neurális hálózat kimeneti rétege, ezt felhasználva van lehetőség megváltoztatni a környezetet, amelyben az ágens tevékenykedik. Ebben az esetben, az úthálózat jelzőlámpák állapotának a megváltoztatása a cél, a kimeneti cselekvési vektor alapján. A közlekedési lámpákat csomópontonkénti csoportokba érdemes rendezni, hiszen egy csomópontba lényeges a lámpák szinkronizált működése. Azaz nem lehet olyan eset, hogy két szomszédos lámpa egyszerre váltson zöldre, hiszen ez balesethez vezetne. Ennek megfelelően a csomópontokban a forgalom iránya egyszerre fog változni, az adott csomóponthoz tartozó vektor érték feldolgozásának megfelelően. A cselekvést meghatározó vektor ebből adódóan a csomópontok számának nagyságának felel meg.
Az implementált megoldás szerint adott egy csomópont, ahol több sávos utak vannak. Ezekhez a sávokhoz tartoznak jelzőlámpák, amikhez pedig különböző fázistervek tartoznak. Ezeket a fázisterveket a konfigurációs fájlokban lehet definiálni.

```
<additional>
  <tlLogic id="0" type="static" programID="my_program" offset="0">
    <phase duration="7" state="GGGrrrGGGrrrrrr"/>
    <phase duration="7" state="yyyrrryyyrrrrrr"/>
    <phase duration="7" state="rrrGGGrrrrrrrrr"/>
    <phase duration="7" state="rrryyyrrrrrrrrr"/>
    <phase duration="7" state="rrrrrrrrrGGGGGG"/>
    <phase duration="7" state="rrrrrrrrryyyyyy"/>
  </tlLogic>
</additional>
```

5.5. ábra.
Csomópont konfiguráció. (Saját készítésű ábra.)

A fenti ábrán látható, hogy egy adott fázis milyen állapotokból áll össze és mennyi ideig tartson egy adott állapot. Erre úgy lehet tekinteni, mint egy egységnyi idő amire minimum

szeretnénk, hogy fázis váltás legyen. Erre azért van szükség, hogy ne kezdjen el lépésenként váltogatni a különböző fázisok között. Az Adapternél ezt a működést támogatni szükséges, azaz számolni kell, hogy az adott fázis mennyi ideig tart és addig nem kell változtatni a fázison. Aztán ha a modell ismét úgy gondolja, hogy ez a fázis a preferált állapot, akkor ismét azt a fázist fogja választani.

5.7. Jutalom függvény

A jutalom függvény adja meg azt a metrikát, amely alapján a tanítás során kiértékelődnek az ágens egyes döntései. Megvizsgálni érdemes jutalom függvények például az összes késés, illetve az egy helyben álló autók száma alapján kalkulált büntető érték vagy pedig a járművek átlagsebessége [32]. A kísérleteknél az éppen használt jutalomfüggvények jelezve lesznek.

5.8. A szimuláció felparaméterezése

A következő és egyben egyik legfontosabb szempont pedig a neurális hálózat modelljének az összeállítása, illetve a megerősítéses tanulás során használt algoritmus paramétereinek a meghatározása. A kísérletek során PPO algoritmus volt felhasználva. Az stable-baselines3 segítségével belehet importálni és fel lehet paraméterezni az algoritmust.
A környezet regisztrációja során az OpenAI Gym segítségével az stable-baselines3 ráilleszti a neurális hálózat bemeneteként az állapot teret [27], (itt Observation space), illetve a kimeneteként a cselekvési teret (itt Action space). Lehetőség van minden egyes iterációban menteni az ágens állapotát, vagy akár csak a legvégén [27]. A mentési pontok segítségével egy adott tanított ágenst van lehetőség betölteni és tovább tanítani. A kísérletet ezután ellehet indítani. A Tensorboard-nak köszönhetően lehet látni futásidőben a hatékonyságát az ágensnek. Iterációnkként kapunk visszajelzést arról, hogy milyen jutalom értéknél jár az ágens, illetve, hogy a változó paraméterek éppen milyen értékeket vesznek fel. Miután a kísérlet lefut a különböző metrikákat a futásról elmentjük egy célkönyvárba a kísérlet nevével. Az eredményeket Tensorboard segítségével lehet megtekinteni. Itt az egyik legfontosabb adat a jutalom érték.

5.9. Statikus forgalomirányító szimulációja

A kísérletekben az ágens eredményei fontosak, ennek ellenére ez csak arról ad képe, hogy relatíve az ágens alaphelyzetéhez, fejlődik-e a forgalomirányítási készsége a megadott ju-

talom függvény, mint "bíráló" metrika, alapján. Ez ugyan fontos tény, hiszen hosszútávon a rendszer továbbfejlesztésére és egyre jobb értékek elérése csak olyan esetben lehetséges, ha van legalább egy olyan konfiguráció az ágens számára, ahol tanuló tendenciát tud mutatni. Ezt a lokális teljesítményt, ami relatív önmagához az ágenshez, hasznos lenne összehasonlítani egy másik forgalomirányítási rendszerrel. Kézenfekvő a leggyakrabban használt rendszert, a statikus forgalomirányítókat alapul venni, nem csak azért, mert ez a legelterjedtebb és ez a világ minden táján ugyanazt a forgalomirányítási logikát jelenti, hanem ez a képzeletbeli teljesítmény-létrán a legalsó kihívó és logikus ezzel kezdeni.
Ahhoz, hogy az összehasonlítás mértékadó legyen, fontos, hogy a két szimulációnál csak a forgalomirányító logika legyen különböző, a többi paraméter maradjon azonos. Ilyen például, hogy a szimuláció ugyanazon az úthálózaton fusson, illetve, hogy az autók beáramlása hasonló módon történjen mint, ahogy az ágensnél futtatott kísérletekben történt. Ennek értelmében a statikus szimuláció során hasonlóan, folyamatos inflow-al legyen tesztelve a környezet. Ez azt jelenti, hogy az úthálózat bemeneteként folyamatosan érkezni fognak az újabb és újabb autók.
További cél az autók viselkedésének felkonfigurálása. Természetesen ez is az ágens során teszttelttel kell, hogy egyezzen.
Az összehasonlíthatósághoz a szimulációt ugyan azon az elkészített környezeten érdemes futtatni. A cél a megerősítéses tanulás során használt jutalom függvény alkalmazása a statikus irányító tesztelésekor. Mivel ugyanazt a környezet lesz használva, így a jutalom függvény hasonlóan tudja majd a környezeti leírásokból a pillanatnyi autók helyzetét kinyerni, ezzel pedig valós összehasonlítási alapot tud képezni a két módszer teljesítményéről. A jutalom érték a környezet pillanatnyi állapota alapján van kiszámolva.

5.10. Kísérletek és eredmények

Ebben a fejezetben a különféle kísérletek és ezeknek az eredményei lesznek bemutatva. Minden kísérletnél az aktuális futás paraméterei fel lesznek tüntetve, ilyen paraméterek például a modell algoritmusának hiperparaméterei, az Állapot tér, Cselekvési tér és a Jutalom függvény. Természetesen a futás végén a releváns diagramok is helyet kapnak az eredmények mellett.

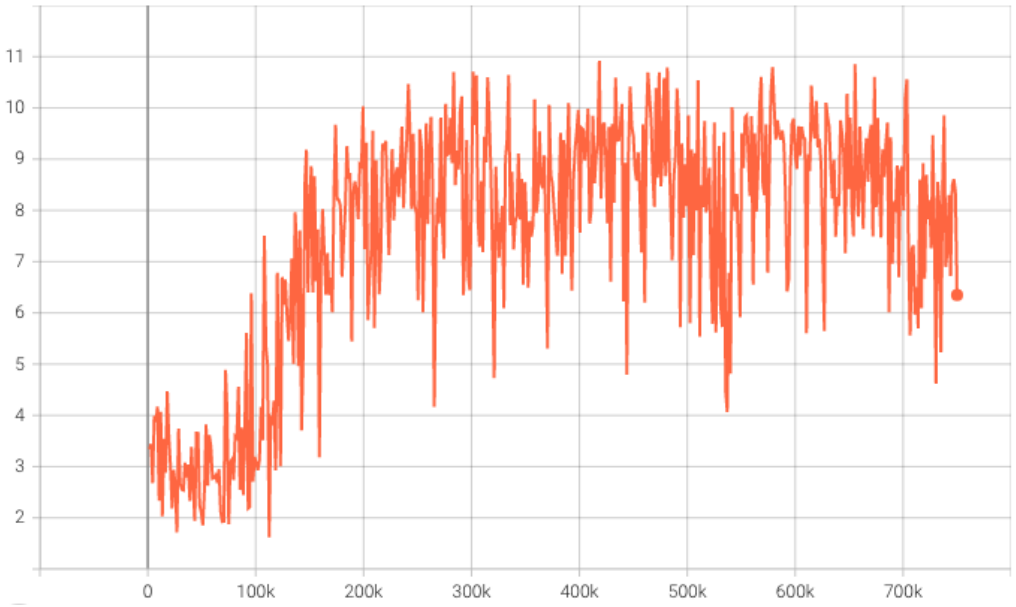
5.10.1. Kísérletek

A következő kísérletek PPO optimalizációs függvényt használva lettek elvégezve és az alábbi paramétereket használták.

Hiperparaméter	Érték
learning rate	0.0003
batch size	futás ideje
n epochs	1000
gamma	0.999
gae lambda	0.99
clip range	0.2
ent coef	0.0
vf coef	0.5
max grad norm	0.5
target kl	0.01

5.2. táblázat.
PPO Hiperparaméterek

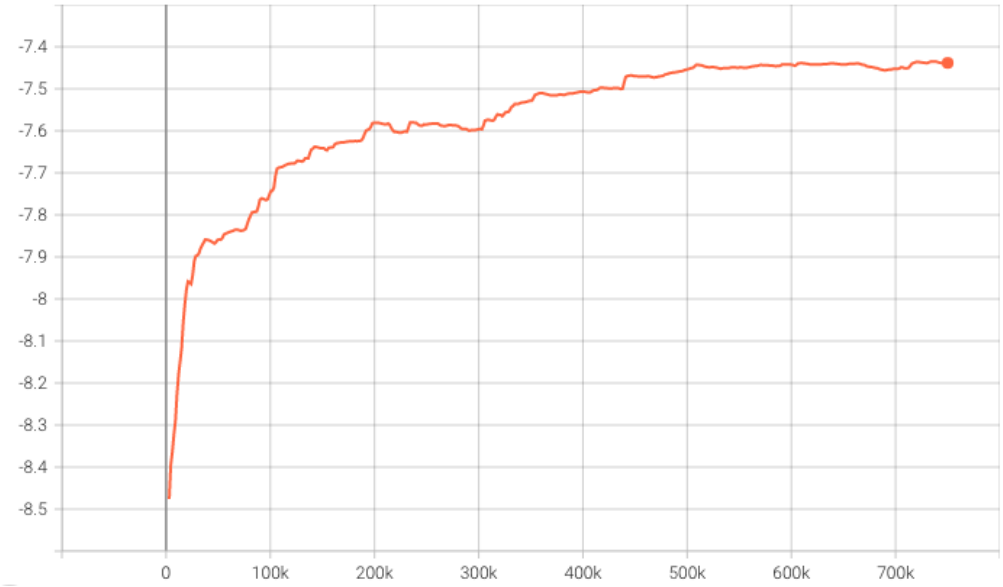
Az alábbi kísérlet a fenti paramétereket használva lett elvégezve és összesen 500 iterációból és iterációnként 1500 lépésből állt. Ez volt az első olyan kísérlet, ahol az ágens többnyire felfelé mutató tendenciát adott a jutalom értékek terén. A használt jutalom függvény a járművek átlagsebességét vizsgálta. Minél magasabb a járművek átlagsebessége az adott pillanatban annál nagyobb jutalom értéket kap az adott lépés során az ágens. A magasabb átlagsebesség alapján feltételezhetjük, hogy kevesebbet kellett lassítania és várnia a járműveknek az útjuk során.
Az ágens Állapot tér-ként a járművek X és Y koordinátáját kapja meg, Cselekvés tere pedig 6 darab lebegőpontos értéket tartalmazó vektor. A 6 egység az a jelenlegi kísérlet csomópontjában megtalálható közlekedési lámpák összes lehetséges állását jelöli. A maximális értéket jelölő fázis kerül kiválasztásra a vektorból és az ahhoz tartozó fázisidő is beállításra kerül. Ez azt jelenti, hogy másik fázis nem fogja megszakítani a már kiválasztott fázist. A modell 2 darab rejtett réteggel rendelkezik, rétegenként 64 darab neuronnal. Az aktivációs függvény Tanh.
Az alábbi diagramon látható a tanítás eredménye.



5.6. ábra.

Tanítás eredménye (1). (Saját készítésű ábra.)

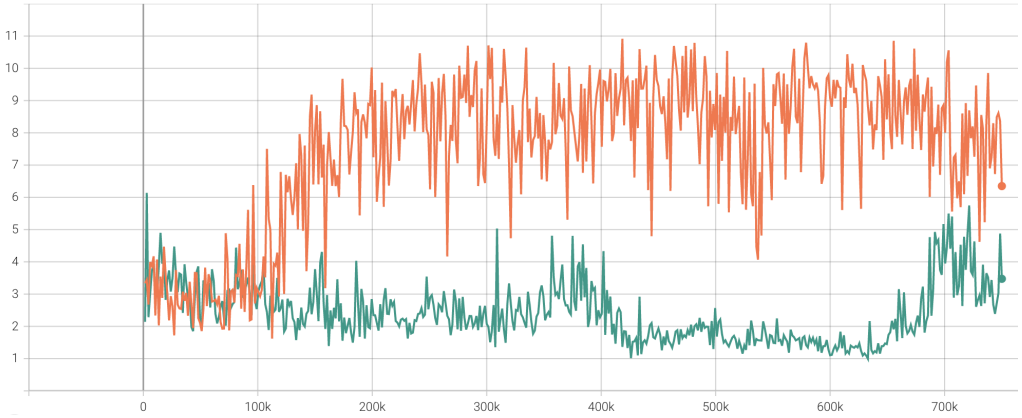
Az ábrán látható, hogy az ágens egyre jobb eredményeket ér el a tanulás során, azaz a járművek átlagsebessége folyamatosan nő. A maximum elért jutalom érték a futás során 11.15. További releváns metrikákat, bizonyos paraméterek alakulásáról az alábbi diagramokon láthatunk.



5.7. ábra.

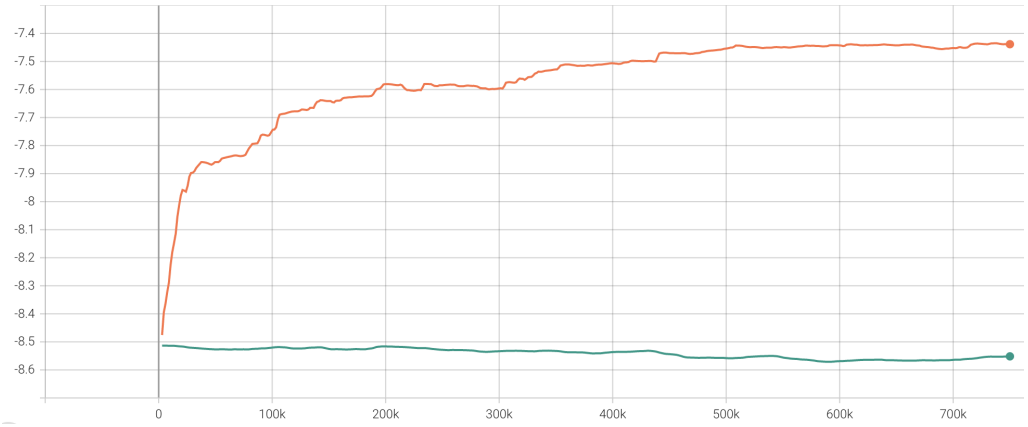
Entrópia diagram (1). (Saját készítésű ábra.)

Az entrópia érték azt mutatja meg, hogy az ágens a tanulás során, mennyire megy el a felfedezés irányába, azaz mennyire hoz véletlenszerű döntéseket. Itt láthatóan az ágens ahogy elkezdett egyre jobb értékeket elérni csökkent a véletlenszerű döntések száma és inkább a megtanult minta alapján hozott döntéseket.
Mivel ez volt az első olyan kísérlet, amely bizakodásra adott okot, így ezeknek a paramétereknek a további finomhangolásával történő próbálkozások lesznek bemutatva. Első sorban a hiperparaméterek lettek felfedezve, próbálgatva a jobb eredmény reményében. Ez leginkább empirikus módon történt, szinte véletlenszerűen változtatva a paramétereket, különböző eredményeket kapunk. A hiperparamétereken kívül a modell neurális hálózatának topológiájával is történtek kísérletek, illetve az Állapot tér megváltoztatása is vizsgálva lett.
A következő esetben a learning-rate paraméter lett módosítva 0.0003-ról 0.001-re, illetve az n_epochs paraméter lett lentebb véve 10-re. A többi paraméter változatlan maradt. Az ezzel elért eredmény alábbi ábrán látható.



5.8. ábra.
Tanítás eredménye, összehasonlítva (2). (Saját készítésű ábra.)

Zölddel látható az módosított paraméterekkel végrehajtott kísérlet diagramja. Az eredmények láthatóan nem közelítik meg az elsőjét. Legjobb esetben is minimális növekedés figyelhető meg a legvégén a kísérletnek, itt a legjobb, amit tenni lehet, hogy a maximum jutalom értéknél lementett modellt használnánk a mint független logika. Erre nincs szükség hiszen az előző kísérletnél már jobb értéket is el tudtunk érni.



5.9. ábra.

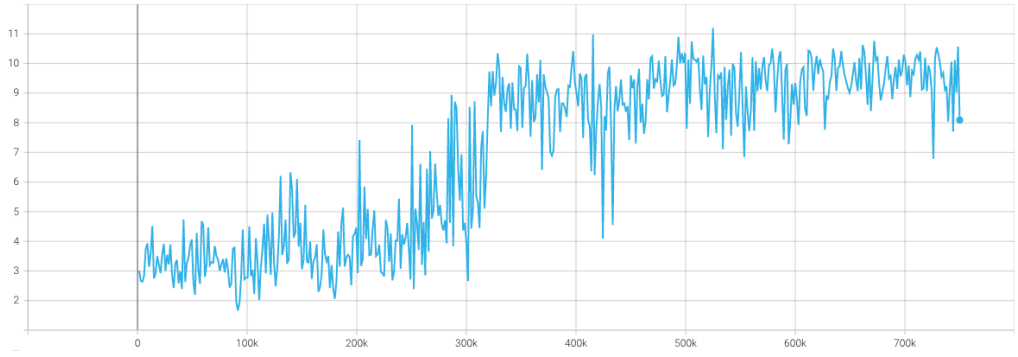
Entrópia diagramok összehasonlítása (2). (Saját készítésű ábra.)

Itt is megfigyelhető, hogy az entrópia értékek abban az esetben csökkennek, ahol ténylegesen a jutalom értékek is növekednek, azaz a számunkra preferált irányba tanul.

Továbbiakban az epochs és a learning-rate paraméterek vizsgálata zajlott, az 1000 körüli értékek megválasztása az epochs számára, javított az eredményeken valamelyest, de a learning-rate értékek megváltoztatása hozta általában a legjelentősebb eredményeket kísérletek között.

Amint az előbb említett paramétereket a legjobb eredményhez hasonlóvá korrigáltuk, annál jobb eredményeket produkált az ágens.

Többi próbálkozás után az alábbi kísérlet hozta a legjobb eredményt. Ennek a futásnak a maximális jutalom értéke 11.2. A target_kl érték módosult 0.001-re, a learning-rate 0.001, illetve az epochs paraméter 1500-ra lett módosítva.

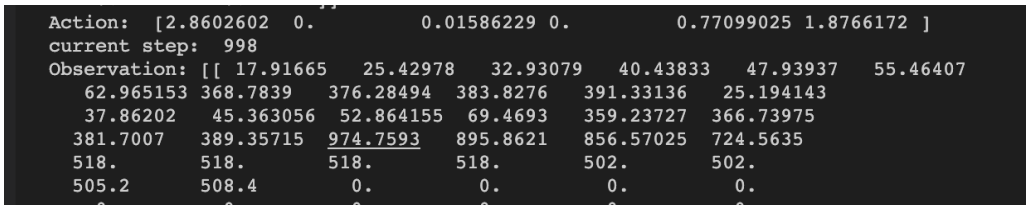


5.10. ábra.

Az eddigi legjobb futás grafikonja. (1). (Saját készítésű ábra.)

5.10.2. Tanított ágens felhasználása

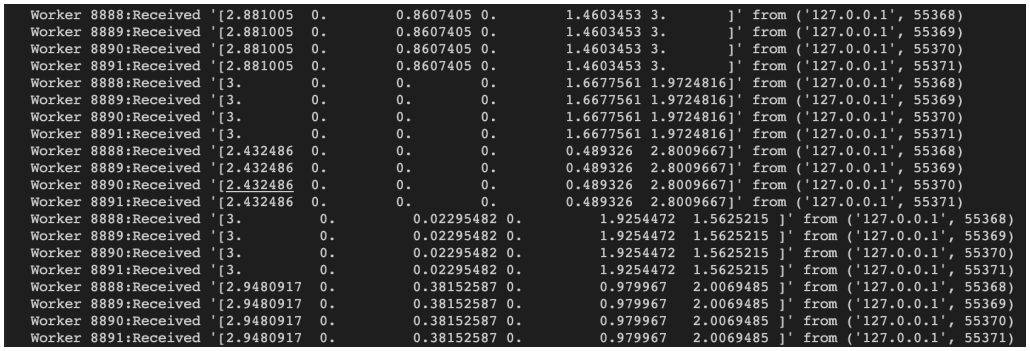
Az egyik kimondott cél az volt, hogy lehetőség legyen felhasználni a betanított modellt valamilyen más módon is, a szimulációs implementáción kívül. Erre a közlekedési csomópontokat szimulálni hivatott TrafficWorker lett elkészítve, amellyel socket alapú kommunikációt végez az ágens.
A legjobban teljesítő ágenst ki kell választani, majd pedig betölteni. A legjobban teljesítő ágensnek pedig a legjobb értéket elérő mentését kell kiválasztani, a fenti ágens esetében ez a 350. A szimuláció ebben az esetben is szükséges lesz, hiszen bemenetet kell szolgáltatni az ágens számára. Kimenete az ágensnek két irányba van ebben az esetben. Az egyik a szimuláció maga, a másik pedig a közlekedési csomópontokat szimuláló TrafficWorker-ek. Ezt az Adapter osztály segítségével lett megvalósítva. Itt a specifikus implementáció gondoskodik arról, hogy csatlakozzon a megfelelő végpontokhoz, illetve továbbítsa a cselekvéseket.
Az alábbi ábrán látható, a betöltött modell munka közben.



5.11. ábra.

Ágens kimeneti értékei, forgalomirányítás közben. (Saját készítésű ábra.)

A konzol kimeneten látni lehet a modell számára bemenetként eljuttatott Állapot teret, illetve a kimeneti Cselekvési mátrix egy részét.
A másik "oldalon" a TrafficWorker-ek az alábbi konzol kimeneteket generálják.



5.12. ábra.

A hálózati végpontok konzol kimenete futás közben. (Saját készítésű ábra.)

5.10.3. Összehasonlítás a statikus módszerrel

A statikus módszerrel való összehasonlításhoz, szükséges a hasonló kísérleti környezet megteremtése, illetve, hogy közös metrikák szerint legyen a két módszer felmérve. Ez a metrika jelen esetben a járművek átlagsebességének megvizsgálása a két módszer használata során. A kísérletek során a legjobb elért eredmény, azaz a maximum jutalom érték 11.2. A statikus módszer esetében ez az érték 3.474. Ezek szerint a megerősítéses tanulást használó módszer jobb eredményt volt képes elérni a vizsgált metrika szerint.

6. TESZTELÉS

6.1. Tesztelés menete a projektben

A projekt fejlesztése során a tesztelés két részben történt meg. Egyfelől manuális tesztelés másfelől automatizált tesztelés történt.
Automatizált tesztelésre a Python beépített unittest könyvtára lett felhasználva. Ennek segítségével egység tesztek lettek implementálva, amelyek az atomi metódusok szintjén ellenőrizték a helyes működést. Mivel voltak külső könyvtárakra mutató függőségek ezért ezek nem lettek tesztelve. Abban az esetben, amikor pedig szükséges volt egy-egy metódusnál külső könyvtárra való kihivatkozásra, a unittest [33] könyvtár Mock osztálya lett felhasználva ezzel is biztosítva az egység teszteknél szükséges izolációt. Az automatizált tesztelés során többek között a jutalom értékek kiszámítását, az Állapot tér mátrixának a felépítését, illetve a Cselekvés mátrix döntéseinek a végrehajtási logikája lett tesztelve. A legfőbb tesztelt funkciók listája az alábbi.

Funckiócsoport	**Példa tesztleírás**	**Eredmény**
A jutalom értékek helyes kiszámítása.	Átlagos sebesség kiszámítása helyes értékekkel.	OK
A SumoAdapter Állapot tér felépítő függvény helyes működése.	Egy elhelyezett autónak a pozíciójának az elhelyezése megfelel.	OK
A SumoAdapter Állapot tér felépítő függvény helyes működése.	Nincsenek autók a szimulációban és helyes a felépített tér.	OK
A SumoAdapter Cselekvés mátrix végrehajtásának az ellenőrzése.	A Cselekvési mátrixnak megfelelő számú függvényhívás illetve a paraméterek ellenőrzése.	OK
A SumoAdapter Cselekvés mátrix végrehajtásának az ellenőrzése.	A választott válaszidő hátralévő idejének az ellenőrzése .	OK

6.3. táblázat.
Táblázat az automatizált tesztekről

Manuális tesztelés leginkább nehezen tesztelhető funkciókat fedett le, ilyen például az úthálózat betöltése a SUMO szimulátorba, annak elindítása, cselekvések végrehajtásának validációja grafikus felületen. SUMO szimulátorban elindított kísérlet hosszának validációja, hiperparaméterek paraméterezésének ellenőrzése. Kísérlet konfigurációjának validációja a grafikus felületen.

Követelmény	**Tesztleírás**	**Eredmény**
A betöltött úthálózat a kívánt típusú.	A úthálózatot betöltése majd szimulációban való ellenőrzése.	OK
Az elhelyezni kívánt autók megjelennek a szimulációban.	A járművek betöltése a szimulációba, szimuláció elindítása és járművek vizsgálata.	OK
Szimuláció grafikus megjelenítése.	A render paraméter beállítása és a szimuláció elindítása, megjelenítés.	OK
Szimuláció grafikus elrejtése.	A render paraméter beállítása és a szimuláció elindítása, háttérben fut a szimuláció.	OK
A kísérlet a kívánt számú iterációszor fut le.	Az iteráció beállítása és a szimulációban való ellenőrzése.	OK
Az iterációk hosszát lehetséges állítani.	Az iteráció hosszának felparaméterezése után a kísérlet az kívánt ideig fut.	OK
Hiperparaméterek konfigurálása a preferált értékekre.	A hiperparaméterek a kimeneti log-ok alapján megegyeznek a konfigurált értékekkel.	OK
Az állapot tér a kívánt méretű.	Állapot tér paramétereinek beállítása, futás közben megegyezik az elvárt értékkel.	OK
A cselekvési tér a kívánt méretű.	Cselekvési tér paramétereinek beállítása, futás közben megegyezik az elvárt értékkel.	OK
Fix fázisidejű szimuláció indítása a környezetben.	Az elkészített szimulációban fix fázisidejű logika kerül és ez megfigyelhető a kísérletben.	OK
Megerősítéses tanulást használó ágenssel való szimuláció indítása.	Az elkészített szimulációba a kijelölt ágens kerül és a környezettel interakcióba lép a kísérletben.	OK

6.4. táblázat.
Táblázat az elvégzett manuális tesztekről

7. ÖSSZEGZÉS ÉS TOVÁBBFEJLESZTÉS

A kísérletek még korai fázisban járnak, szükséges további közlekedési szcenáriókban letesztelni különböző neurális modellek teljesítményét. Az elvégzett kísérletek rámutatnak arra, hogy a megerősítéses tanulást használó ágensnek jó eredményeket tudnak elérni a közlekedési lámpák irányításának a területén. A vizsgált esetekben a legjobb jutalom értéket elérő modell, 11.2 értéket tudott felmutatni a statikus forgalomirányító 3.474-gyel szemben. A jutalom érték a közlekedésben résztvevő járművek átlagsebessége alapján lett kiszámítva.

A következő lépés lenne az ágens tesztelése, nagyobb úthálózatok csomópontjainak az irányítására, illetve több optimalizáló algoritmus kipróbálása is preferált. A különböző neurális modellek feltérképezése, illetve a hiperparaméterek finomhangolása is jobb eredményekhez juttathatnak a továbbiakban.

Ezzel együtt a projekt a saját, mértékében, a kijelölt követelményeket teljesítette, a statikus forgalomirányítónál pedig még jobb eredményt is képes volt elérni, a vizsgált metrika alapján.

ÁBRAJEGYZÉK

2.1. Brodmann-modell (forrás:Dr. Kutor László előadása) 17
2.2. Neuron formális modellje (forrás: Dr. Kutor László előadása) 18
2.3. Tanh aktivációs függvény (forrás: [17]) 23
2.4. ReLU aktivációs függvény (forrás: [17]) 24
2.5. Swish aktivációs függvény (forrás: [19]) 24
2.6. Az ágens és környezet interakciója (forrás: [20]) 25
4.1. A projekt rendszerterve. (Saját készítésű ábra.) 31
4.2. Az ágens neurális hálójának felépítése. (Saját készítésű ábra.) 34
5.1. Az általunk használt csomópont 38
5.2. Egy lépés során végzett műveletek (Saját készítésű ábra.) 40
5.3. Kapcsolódás a workerekhez és cselekvések küldése. (Saját készítésű ábra.) 42
5.4. Aszinkronos worker kódrészlete. (Saját készítésű ábra.) 43
5.5. Csomópont konfiguráció. (Saját készítésű ábra.) 44
5.6. Tanítás eredménye (1). (Saját készítésű ábra.) 48
5.7. Entrópia diagram (1). (Saját készítésű ábra.) 48
5.8. Tanítás eredménye, összehasonlítva (2). (Saját készítésű ábra.) 49
5.9. Entrópia diagramok összehasonlítása (2). (Saját készítésű ábra.) 50
5.10. Az eddigi legjobb futás grafikonja. (1). (Saját készítésű ábra.) 50
5.11. Ágens kimeneti értékei, forgalomirányítás közben. (Saját készítésű ábra.) 51
5.12. A hálózati végpontok konzol kimenete futás közben. (Saját készítésű ábra.) 51

TÁBLÁZATOK JEGYZÉKE

5.1. Táblázat a csomagokról 38
5.2. PPO Hiperparaméterek 47
6.3. Táblázat az automatizált tesztekről 53
6.4. Táblázat az elvégzett manuális tesztekről 54

IRODALOMJEGYZÉK

[1] Tettamanti, T.: A közúti közlekedés irányítása előadás, (https://docplayer.hu/51685893-A-kozuti-kozlekedes-iranyitasa-dr-tettamanti-tamas-bm.html), utoljára megtekintve: 2022.11.23

[2] Studer, L., Ketabdar, i. M., Marchionni: Analysis of Adaptive Traffic Control Systems Design of a Decision Support System for Better Choices.

[3] van der Pol, E., Oliehoek, F. A.: Coordinated Deep Reinforcement Learners for Traffic Light Control. NIPS. 2016

[4] Wei, H., Zheng, G., Yao, H., Li, Z.: IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control. *In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2018, pp. 2496–2505.

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G.: Human-level control through deep reinforcement learning. 2015 Feb 26, pp. 518–529.

[6] Miller, A. J.: Settings for fixed-cycle traffic signals. *In Journal of the Operational Research Society*, 1963, pp. 373–386.

[7] Cools, S.-B., Gershenson, C., D'Hooghe, B.: Self-organizing traffic lights: A realistic simulation. *In Advances in applied self-organizing systems. Springer*, 2013, pp. 45–55.

[8] Python Website, (https://www.python.org/), utoljára megtekintve: 2022.11.18

[9] Tensorflow Website, (https://www.tensorflow.org/tensorboard), utoljára megtekintve: 2021.05.13

[10] Dr. Kutor, L.: Intelligens rendszerek elmélete előadás, (http://uni-obuda.hu/users/kutor/IS%202020/IS%202020-4.pdf), utoljára megtekintve: 2021.05.13

[11] Altrichter, M., Horváth, G., Pataki, B., Strausz, G., Takács, G., Valyon, J.: Neurális hálózatok

[12] Kurenkov, A.: A Brief History of Neural Nets and Deep Learning. *Skynet Today*, 2020

[13] Schmidhuber, J.: Deep Learning in Neural Networks: An Overview. 2014

[14] McCulloch, W. S., Pitts, W.: A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY. 1943

[15] Rumelhart, D. E., Hinton, G. E., Williams, R. J.: Learning representations by back-propagating errors. 1986

[16] Nwankpa, C. E., Ijomah, W., Gachagan, A., Marshall, S.: Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. 2018

[17] Feng, J.: Reconstruction of porous media from extremely limited information using conditional generative adversarial networks

[18] Ramachandran, P., Zoph, B., Le, Q. V.: Searching for activation functions. 2017

[19] Swish activation function, (https://twitter.com/aureliengeron), utoljára megtekintve: 2021.05.13

[20] Sutton, R. S., Barto, A. G.: Reinforcement Learning An Introduction

[21] Melo, F. S.: Convergence of Q-Learning: A simple proof

[22] Fan, J., Wang, Z., Xie, Y., Yang, Z.: A Theoretical Analysis of Deep Q-Learning. *In A. M. Bayen, A. Jadbabaie, G. Pappas, P. A. Parrilo, B. Recht, C. Tomlin, M. Zeilinger, editors, Proceedings of the 2nd Conference on Learning for Dynamics and Control*, Vol. 120 of Proceedings of Machine Learning Research, 2020, pp. 486–489.

[23] Torres, J.: Deep Q-Network (DQN)-II

[24] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal Policy Optimization Algorithms

[25] OpenAI, (https://openai.com/blog/openai-baselines-ppo/#ppo), utoljára megtekintve: 2021.05.13

[26] OpenAI Gym Website, (https://www.gymlibrary.dev/), utoljára megtekintve: 2022.11.18

[27] Stable-Baselines3, (https://stable-baselines3.readthedocs.io/en/master/), utoljára megtekintve: 2022.11.18

[28] Raeis, M., Leon-Garcia, A.: A Deep Reinforcement Learning Approach for Fair Traffic Signal Control

[29] TraCI Documentation, (https://sumo.dlr.de/docs/TraCI.html), utoljára megtekintve: 2021.05.13

[30] SUMO Documentation, (https://sumo.dlr.de/docs/index.html), utoljára megtekintve: 2021.05.13

[31] Conda Website, (https://docs.conda.io/en/latest/), utoljára megtekintve: 2021.05.13

[32] Egea, A. C., Howell, S., Knutins, M., Connaughton, C.: Assessment of Reward Functions for Reinforcement Learning Traffic Signal Control under Real-World Limitations. *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020. (http://dx.doi.org/10.1109/SMC42975.2020.9283498)

[33] Unittest Python, (https://docs.python.org/3/library/unittest.html), utoljára megtekintve: 2022.11.18

8. MELLÉKLETEK

A kísérletek futtatásához az alábbi lépéseket szükséges megtenni:

- SUMO [30] szimulációs szoftver telepítése.
- Conda [31] virtuális környezet telepítése.
- Python 3.10.7 [8] telepítése.
- Az 5.1 táblázatban felsorolt csomagok telepítése (némelyik része a Python alapkönyvtárnak).
- A kísérlet futtatásához szükséges konfigurálni a SUMO szimulátor elérési útját:
  - A tester.py állományban, futtatási platformtól függően, sumoBinary/sumoBinaryMac változó értékének módosítása az elérési útra, például:
    "C:/Program Files (x86)/Eclipse/Sumo/bin/sumo-gui.exe"
    (konkrét kísérletnél ajánlott "sumo-gui.exe" helyett "sumo.exe"-t futtatni, hogy ne kapjunk grafikus felületet)
- A kísérlet felparaméterezése és elindítása:
  - Attól függően, hogy mit szeretnénk futtatni az alábbi opciók vannak.
    * experiment(kísérlet neve, iteráció számok, lépések száma): tanítás futtatása
    * test_static(kísérlet neve, iteráció számok, lépések száma): kísérlet futtatása statikus forgalomirányítóval (ekkora a config/tl.add.xml határozza meg a logikát)
    * test_model(modell neve, n): betanított modell, n-dik mentésének a betöltése SUMO szimulációval
    * test_model_realife(betanított modell, n): betanított model, n-dik mentésének a betöltése távoli workerekek együttműködésével és SUMO szimulátorral
  - Itt a fenti felsoroálsból kívánt függvényt kell meghívni, például: experiment("teszt", 500, 1500)
- Abban az esetben ha távoli workereket is szeretnénk kapcsolni, szükséges először elindítani a traffictester.py scriptet.

- Projekt állományainak a felépítése:
  - config mappa: SUMO kísérlet konfigurációs állományai (úthálózat, forgalom, fázisdefiníciók)
  - logs mappa: a különböző kísérletek auditjai kerülnek ide
  - models mappa: a kísérletek során mentett modellek kerülnek ide
  - tests mappa: teszt állományok
  - traffic_lights mappa: TrafficLight workerekkel kapcsolatos állományok
  - adapter.py: Az Adapter implementációk találhatók itt: SumoAdapter illetve RealLifeAdapter
  - custom_callback.py: Hozzáadott auditálás a kísérleteknél
  - environment.py: OpenAI Gym implementáció a kísérlet futtatásához
  - static_environment.py: Statikus kísérletekhez külön OpenAI Gym implementáció
  - neural_model.py: Neurális hálózat modell osztálya, személyre szabásért
  - policy.py: Neurális hálózat modell osztálya, személyre szabásért
  - rewards.py: Jutalom definíciók
  - runner.py: A kísérletek futtatásáért felelős osztály definíciója
  - tester.py: Különböző szcenáriók futtatása, felparaméterezése, elindításáért felelős metódusok helye