ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

NEUMANN JÁNOS
INFORMATIKAI KAR

SZAKDOLGOZAT

OE-NIK Hallgató neve: **Fülep Fanni**
2022. Hallgató törzskönyvi száma: **T/005627/FI12904/N**

Óbudai Egyetem
Neumann János Informatikai Kar
Szoftvertervezés és -fejlesztés Intézet

SZAKDOLGOZAT FELADATLAP

Hallgató neve: **Fülep Fanni**
Törzskönyvi száma: T/005627/FI12904/N
Neptun kódja: [redacted]

A dolgozat címe:

Intelligens parkolóház szimulációja virtuális környezetben
Simulation of an intelligent parking garage in a virtual environment

Intézményi konzulens: Kiss Dániel
Külső konzulens:

Beadási határidő: 2022. május 15.

A záróvizsga tárgyai: Számítógép architektúrák
IoT, beágyazott rendszerek és robotika specializáció

A feladat

Az okos parkolási rendszerek fontos szerepet játszanak a városi mobilitás fejlődésében. A szabad parkolóhely keresésével töltött idő nem csak frusztrációt vált ki a sofőrökben, hanem jelenleg nagymértékű üzemanyag pazarlással is jár, így egy hatékonyan üzemelő és jól kihasznált parkolóház minden szempontból hasznos a forgalmasabb városrészek számára.
A dolgozat célja olyan matematikai és számítógépes szimulációs eljárások ismertetése, elemzése és implementációja, amelyek képesek egy virtuális okos-parkolóház működéséhez realisztikus bementi adatokat biztosítani. Az irodalomban elérhető megközelítések megismerése után tervezzen meg egy olyan szimulációs szoftvert, amely a parkolóház forgalmát képes a valósághoz minél hűbben utánozni, figyelembe véve például a sofőrök érkezésének eloszlását vagy vezetési stílusaik bizonyos jellemzőit. A szoftver legyen alkalmas egy virtuális környezetben megvalósított parkolóház számára alkalmas bemeneti adatformátum előállítására.

A dolgozatnak tartalmaznia kell:

- a probléma céljának és hátterének rövid összefoglalóját,
- a hasonló területen alkalmazott modellek és szimulációs technikák bemutatását és elemzését,
- a megvalósítandó szimulációs rendszer specifikációját, a szoftver rendszertervét, illetve az alkalmazott eljárások működésének ismertetését,
- az implementáció menetének bemutatását,
- a tesztelési tervet és a teszteredmények ismertetését,
- az esetleges továbbfejlesztési lehetőségek megvitatását.

Ph.

.....
Dr. Vámossy Zoltán
intézetigazgató

A szakdolgozat elévülésének határideje: **2024. május 15.**
(OE TVSz 55.§ szerint)

A dolgozatot beadásra alkalmasnak tartom:

.....
külső konzulens

Kiss Dániel
.....
intézményi konzulens

HALLGATÓI NYILATKOZAT

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

Budapest, 2022. május 13.

.....
hallgató aláírása

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
Fülep Fanni	[redacted]	**nappali**

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Intelligens parkolóház szimulációja virtuális környezetben

Szakdolgozat címe angolul:

Simulation of an intelligent parking garage in a virtual environment

Intézményi konzulens: Külső konzulens:

Kiss Dániel

Alk.	Dátum	Tartalom	Aláírás
1.	2021. 09. 29.	Tématerv egyeztetése	Kiss Dániel
2.	2021. 10. 18.	Modellek, szimulációs eljárások	Kiss Dániel
3.	2021. 11. 24.	Rendszerterv egyeztetése	Kiss Dániel
4.	2021. 12. 09.	Dolgozat szövegének korrekciója	Kiss Dániel

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

Kiss Dániel
.....

Intézményi konzulens

Budapest, 2021. december 10.

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Neumann János Informatikai Kar

KONZULTÁCIÓS NAPLÓ

Hallgató neve:	Neptun Kód:	Tagozat:
Fülep Fanni	[redacted]	**nappali**

Telefon: Levelezési cím (pl.: lakcím):

[redacted]

Szakdolgozat címe magyarul:

Intelligens parkolóház szimulációja virtuális környezetben

Szakdolgozat címe angolul:

Simulation of an intelligent parking garage in a virtual environment

Intézményi konzulens: Külső konzulens:

Kiss Dániel

Alk.	Dátum	Tartalom	Aláírás
1.	2022. 03. 17.	Úthálózat felépítése	Kiss Dániel
2.	2022. 04. 12.	Járműkövetés megvalósítása	Kiss Dániel
3.	2022. 05. 03.	Tesztelési terv egyeztetése	Kiss Dániel
4.	2022. 05. 10.	Dolgozat szövegének véglegesítése	Kiss Dániel

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

Kiss Dániel
.....

Intézményi konzulens

Budapest, 2022. május 11.

TARTALOMJEGYZÉK

RÖVID TARTALMI ÖSSZEFOGLALÓ A TÉMA TERÜLETÉRŐL, A FELADATRÓL

Az okos parkolási rendszerek, azon belül is a parkolási tanácsadó és információs rendszerek célja, hogy segítsék a járművezetőket a szabad parkolóhelyek megtalálásában. A dolgozatom célja egy ilyen feladatot ellátó, virtuális okos parkoló rendszer számára a realisztikus adatok szolgáltatása.

Az irodalomkutatás során különböző lehetőségek összehasonlítását végzem el, amelyben kitérek arra is, hogy milyen viselkedési modellel lehet leírni a járművezetés feladatát, hogyan történik a lassítás illetve gyorsítás. Ezen kívül figyelmet fordítok arra is, hogy milyen gyakorisággal érkeznek járművek a parkolóházba. Az irodalomkutatás befejezése után egy részleges rendszertervet valósítok meg.

A megvalósítás során megvizsgálom a lehetséges implementálási lehetőségeket, majd azokat megvalósítom Unreal Enigne-ben. Különös figyelmet szentelek a járművek mozgására, a lehetséges akadályok kikerülésére, illetve az egy nyomon haladó járművek követésére.

1. ALAPFOGALMAK

1.1. Közúti Forgalom

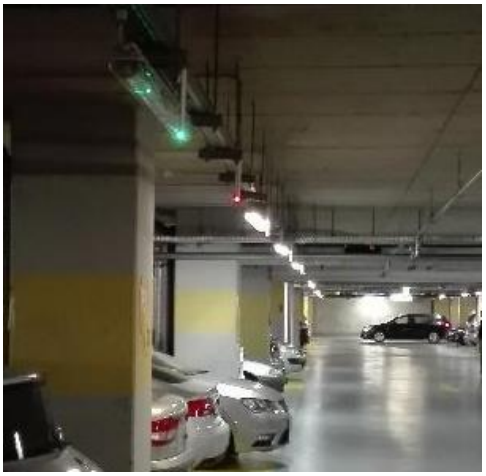
A közúti forgalom egy gyűjtő név, amely utal minden olyan járműre, amely egy megadott nyomtávon halad (út vagy autóút/pálya), és a KRESZ szabályokat betartva közlekednek, mint például a sávtartási kötelezettség, a lámpás forgalomirányításnak való elégtétel, sebességkorlátozás stb. A járművek több kategóriából állnak össze, például autóbuszok, személy gépjárművek, kamionok, kerékpárok.

1.2. Parkolás

Parkolás alatt egy jármű hosszabb ideig történő egyhelyben tartózkodását értjük, járművezető felügyelete nélkül. A jogszabály „várakozás" néven hivatkozik rá, de a köznyelv szinte kizárólagosan a „parkolás" kifejezést használja, ezért az egyszerű érthetőség jegyében így fogunk utalni rá a továbbiakban.
Megvalósítás szempontjából lényeges különbség van a közterületi és a beléptető rendszerrel rendelkező parkolóhelyek (mélygarázsok, parkolóházak stb.) között. Utóbbi esetén például a rendelkezésre álló helyek száma könnyen megállapítható a kapunál kiadott jegyek száma alapján, míg közterületen ez egy ennél sokkal összetettebb feladat, amely magában foglalja valamilyen szenzoros hálózat kiépítését. Ebből kifolyólag az irodalomban két fő kutatási irány követhető nyomon: az egyik a közúti (publikus, *on-street*) parkolásra helyezi a hangsúlyt; a másik pedig, a beléptető rendszerrel rendelkező (privát, *off-street*) parkoló létesítményekre fókuszál.
Ezen felül beszélhetünk még kültéri és beltéri parkolásról is, amelyek szintén eltérő megoldásokat igényelnek. Beltér esetén a mennyezet is alkalmas lehet szenzorok, valamint különböző jelzések elhelyezésére, például gyakran szerelnek fel foglaltságot indikáló piros/zöld színű LED-eket a parkolóhelyek fölé, ilyen található például az Újbudai Allee bevásárlóközpontban is, a 1.1. ábrán látható módon. A 1.1. táblázat összefoglalja a különböző parkolási típusokat és példákat mutat azokra.

1.1. táblázat: A parkolás típusainak összefoglalója

	On-Street	**Off-Street**
Beltéri		Parkolóházak, mélygarázsok
Kültéri	Utcai parkolás	Parkoló udvarok



1.1. ábra: Beltéri parkolás lehetőségei: foglaltságot jelző piros és zöld LED-ek (Forrás: **[1]**)

1.3. Okos városok

A Smart City, vagyis az „okos” vagy más néven „intelligens” város pontos definícióját nehéz egyértelműen meghatározni, az ezen elnevezés alatt megvalósított technológiák széles skálája miatt. Magyarországon a hivatalos definícióját egy kormányrendeletben határozták meg 2017-ben. [2] Eszerint az okos város egy olyan település, amely korszerű és innovatív információtechnológiák alkalmazásával fejleszti környezetét, infrastruktúráját és szolgáltatásait. Az okos város, mint minden rendszer, különböző alrendszerekre bontható. Ezek az alrendszerek megközelítéstől függően különbözhetnek egymástól, de több szakirodalom is [3] [4] kiemeli a közlekedést, mint egyik fő alrendszert.

1.3.1. Intelligens közlekedési rendszerek

Az Intelligens közlekedési rendszerek (Intelligent Transportation Systems - ITS) fontos elemei az okos városok tervezésének és megvalósításának. Hivatalosan a kifejezést 1994 óta használják, napjainkra a fogalmat kiterjesztették, így már számos egyéb szolgáltatást is lefed. Az Európai Parlament és a Tanács irányelve szerint [5] ezek olyan alkalmazások, amelyek szolgáltatásokat nyújtanak különféle közlekedési módokhoz, valamint lehetővé teszik a jobb tájékoztatást, és biztonságosabb, összehangoltabb közeledést.
Feladatai közé tartozik a forgalomirányítás és forgalomszabályozás, forgalmi tájékoztatás, forgalmi ellenőrzés. Mivel a forgalomirányítás a helyváltoztatási lánc teljes egészére kihat, így a parkolás-szabályozásra is. [6]

1.3.2. Okos parkolási rendszerek, PGI rendszerek

A Smart Parking Systems (SPS) vagyis az un. okos parkolási rendszerek az általuk nyújtott szolgáltatások alapján osztályozhatóak, ezek alapján öt fő szolgáltatás kategóriát különböztetünk meg [7]: a parkolási tanácsadó és információs rendszerek (*Parking Guidance and Information* – PGI Systems), a tranzit alapú információs rendszerek (*Transit Based Information* – TBI Systems), az automatikus parkolási rendszerek (*Automated Parking System* – APS), az E-parkolás és az okos fizetési rendszerek.

A parkolási tanácsadó és információs rendszerek célja, hogy támogassa a járművezetőket a szabad parkolóhely keresésében. Az első PGI rendszert az 1970-es évek elején helyezték üzembe Németországban, Aachen város területén. Többségében az első rendszerek a városközpontok parkolási lehetőségeiről nyújtottak tájékoztatást, de a későbbiekben egyre szélesebb körben alkalmazták parkolási létesítményekben is (például repülőtereken és bevásárlóközpontokban). [5]

A fentebb említett TBI rendszerek egyébként nagyon hasonlóak a PGI rendszerekhez, azzal a különbséggel, hogy a közösségi közlekedés megállóinak közelében elhelyezkedő un. P+R parkolók számára nyújt tájékoztatást, így figyelembe veszi például a tömegközlekedési menetrendeket is.

1.4. Modellezés és szimuláció

1.4.1. Szimuláció

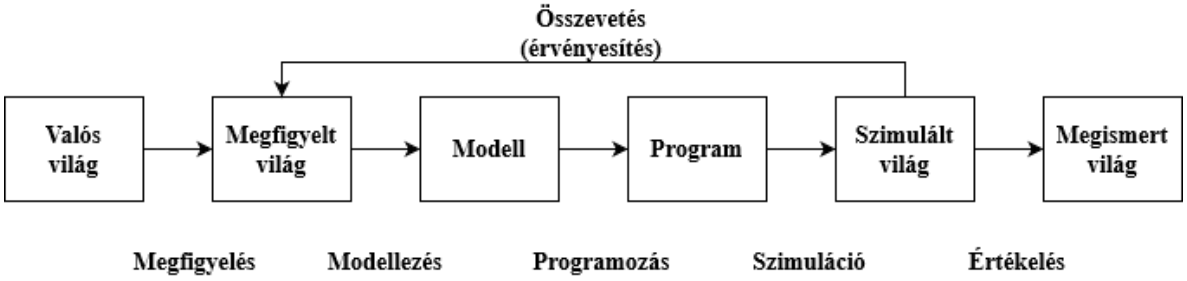
A szimulációt manapság a tudomány számos terén használják. A szimuláció legelterjedtebb jelentése nem más, mint amikor egy rendszernek vagy folyamatnak a várható vagy valós viselkedési módját számítógépes modellek segítségével vizsgálják meg. A számítógépes szimulációknak jellegzetesen sok előnye van a tradicionális „fizikai” kísérletekhez képest, olyan esetekben használják, ahol:

- Túl veszélyes lenne a kísérletet végrehajtani (mérgező/robbanó anyagok tesztelése)
- Az adott kísérletet túl drága lenne megvalósítani
- Túl sok időt venne igénybe (geológiai/botanikus kísérletek)
- Az adott kísérlet túl gyorsan menne folyamatba (robbanások/láncreakciók)
- Kisméretű elemekkel dolgozunk (atomi szint)
- Nagyméretű elemekkel dolgozunk (bolygók, vagy geológiai eróziók vizsgálata)
- Az ideális környezet nem valósítható meg (kémiai anyagok tisztasága/bármilyen elektromos rendszer)
- Az eszköz nem áll rendelkezésre
- Túl sokszor kell elvégezni
- A kísérlethez használt anyag(okból)/tárgy(akból) csak kevés vagy egy létezik
- Túl sok adattal dolgozunk, és vizsgálatuk/megértésük több évet venne igénybe

Ezekben az esetekben roppant fontos a számítógépes szimuláció, és szinte univerzálisan lehet használni a tudomány bármely területén, hogy gyors és pontos eredményeket érjünk el.

1.4.2. Modell

A modell a szimuláció szerves részét képezi, amely pontosan (matematika nyelvén) megfogalmazott állítások (hipotézis) és állítás-rendszereket foglal magába. A modell megalkotása során csak azokat az elemeket emeljük ki, melyek ismertek, feltételezettek és a szimuláció működéséhez feltétlenül fontosok, és ezen elemek hipotéziseit megfelelő módon kapcsolatba hozzuk egymással. Amikor a modell tervezésével és felépítésével végeznek, a tesztelés veszi kezdetét, ahol ellenőrzik, hogy a modellezni kívánt rendszer vagy elem módjára viselkedik. Ezt a folyamatot nevezzük validálásnak vagy verifikálásnak. Amennyiben a modell nem a valóságot tükrözi, úgy ebben az esetben finomítani kell, vagy az alapjaitól újra felépíteni, amely az addigi munka elvetésével jár. A modellezés folyamatának lépéseit a 1.2. ábrán láthatjuk.



1.2. ábra: A modellezés folyamata (Forrás: Saját készítés, [8] alapján)

Modell típusok [9]:

- Diszkrét Modell: Abban az esetben beszélünk diszkrét modellről, amikor az állapotváltozás nem folyamatosan, hanem (diszkrét) időintervallumokban következik be.
- Folytonos Modell: Az a modell, amely állapotváltozása minden pillanatban bekövetkezik, folyamatos modellnek hívjuk
- Determinisztikus modell: Az a modell, amelynél az eredmény egyértelműen meghatározható a program futásához megadott bemenő adatokból, determinisztikus modellnek hívjuk. Ehhez hozzá tartozik az is, hogy mindig ugyan azt az eredményt adja azonos bemeneti paraméterek és feltételek alapján, függetlenül, hogy mikor és ki futtatja a szimulációt
- Sztochasztikus modell: Ezt a szimulációt összefoglaló néven Monte Carlo-módszernek is szokták hívni. A determinisztikus modellel ellentétben itt lehetséges a véletlen számok is, melynek következményében a kimenet eltérhet, hiába az azonos bemeneti paraméterezés.

2. A TÉMA AKTUALITÁSA, JELENTŐSÉGE

A szabad parkolóhely keresése nem csak frusztrációt okoz a sofőröknek, de az üzemanyag felesleges pazarlásával növeli környezetszennyezést, valamint a bizonytalan mozgásuk miatt balesetveszélyes, illetve feltartóztatják a többi járművet [2]. Egyes tanulmányok szerint évente 700 millió eurós veszteséget okoz pusztán Franciaországban a parkolóhelyek keresésével töltött mintegy 70 millió óra. [13] A helyzetet tovább súlyosbítja a gépjárművek számának folyamatos növekedése. A KSH adatai szerint közel 4 millió személygépkocsi fut a magyar utakon, az elmúlt 20 évben több, mint másfélmillióval emelkedett. [10]

Átlagosan a nagyvárosi gépjármű forgalom 30 százalékát az épp parkolóhelyet kereső járművek teszik ki. [10] Ez a probléma parkolási létesítmények esetén is fennáll: a Repülőterek Nemzetközi Tanácsa (ACI-NA) által végzett kutatás szerint az utasok jelentős késéseket szenvednek el a parkolóhelyek keresésének következtében, mivel jellemzően gépjárművel érkeznek a repülőtérre. [11]

2.1. Parkolóingatlanok piaca

A parkolóház-beruházások során kiemelkedően magas a csődbe jutott, leállított és elhalasztott projektek aránya. Budapesten a 2000 és 2015 közötti időszak folyamán összesen 30 parkoló-létesítményt létrehozó fejlesztés valósult meg. Fontos megemlíteni, hogy ennek jelentős része olyan projekt volt, ahol az építési szabályozás írta elő az ingatlan típusához kötelezően kialakítandó parkolóhelyek darabszámát (ilyenek jellemzően a bevásárlóközpontok). Ezeket a beruházásokat figyelmen kívül hagyva 15 év alatt mindössze 7 projekt valósult meg önálló fejlesztés részeként. Ehhez a számhoz viszonyítva jelentős mennyiségű, megközelítőleg 130 parkolóház és/vagy mélygarázs beruházás állt le csak Budapesten ugyanezen időszak alatt. [12]

A nehézkes fejlesztéseknek műszaki, valamint gazdasági vonatkozású okai is vannak: ilyenek például a mélygarázsok talajvíz elleni szigetelésének megoldása, valamint a hosszú megtérülési idő által okozott, külső finanszírozással kapcsolatos problémák. Utóbbihoz hozzájárul az is, hogy a szabadtéri parkolók alacsony parkolási díjai miatt a parkolóházat üzemeltető cég is kénytelen alacsonyabb árszintet alkalmazni, ami akár 35–40 éves megtérülési időt is eredményezhet.

Ugyanakkor az utóbbi évtizedek során a gépjárművek folyamatosan növekvő számából kifolyólag a parkolás kérdése a főváros közlekedésének egyik legnagyobb problémájává nőtte ki magát. A kialakult parkolási krízisből számos környezeti konfliktus is ered, többek között a városi életminőség csökkenése, a zöld területek hiánya és a városkép romlása a járdán parkoló gépkocsik miatt.

A parkolási problémák legkézenfekvőbb megoldása a saját tulajdonú gépkocsi iránti igény csökkentése a lakosság körében, például a helyi tömegközlekedés fejlesztésével, járatok

sűrítésével vagy autómegosztó szolgáltatás (car sharing) segítségével. Azonban az ilyen, szemléletformáláson alapuló megoldások hatása előreláthatólag több évtizedes távlatban lesz csak érzékelhető, a gépkocsihasználati szokások viszonylag lassú változása miatt. Ugyanakkor az is belátható, hogy a fent említett nagymértékű társadalmi feszültség csökkentése sürgető feladat, amely rövidtávú gyors fejlesztéseket is igényel.

Számos parkolásmenedzsment stratégia épít a már rendelkezésre álló parkoló létesítmények minél jobb kihasználása, hiszen aránylag gyorsan megvalósítható, továbbá gazdaságilag is igen kedvezőnek mondható megoldás, mivel nem jár új építési beruházással. Ennek több lehetséges módja van, egyrészt a kapacitás növelése által, például a létesítményben meglévő üres vagy kihasználatlan területek jobb felhasználása révén. Sajnos a tapasztalat viszont azt mutatja, hogy a járművezetők általában az utcai parkolást választják, így a létesítmények eleve nincsenek teljesen kihasználva. Következésképpen ösztönözni kell őket arra, hogy többször válasszák ezeket a létesítményeket; ennek lehetséges eszköze a PGI rendszerek alkalmazása, amivel vonzóbbá tehetőek ezek a létesítmények a gépjárművezetők számára.

3. A JÁRMŰVEZETŐK VISELKEDÉSE

3.1. A járművezető feladatai

Lunenfeld és Alexander [13] a vezetés feladatát három hierarchikus szintre osztotta: kontroll, irányítás és navigáció. Ez látható a 3.1. ábrán.

3.1. ábra: A járművezető feladatainak hierarchikus csoportosítása (Forrás: Saját készítés)

A legalacsonyabb szint, a kontroll foglalja magában az olyan tevékenységeket, amelyek során az információcsere a jármű és annak vezetője között jellemzően tizedmásodpercek alatt végbemegy. Ilyen például a kormányzás, gyorsítás vagy fékezés. A következő szint, amely az irányítási szint, magában foglalja a biztonságos sebesség megválasztását, valamint a jármű az úttesten való megfelelő elhelyezkedését a különböző akadályok, veszélyek és természetesen a közlekedés többi résztvevőjéhez képest. A harmadik, legmagasabb szint, a navigáció, az útvonaltervezés feladata, amely történhet például térkép vagy útirányjelző táblák alapján. A szinteken lentről felfelé haladva az információ kezelésének bonyolultsága egyre növekszik, míg a prioritás biztonság szempontjából nő csökken.
Kiemelten fontos szintek egy létesítmény modellezésekor a kontroll és az irányítás szintje, hiszen feltételezzük, hogy a járművezetők követni fogják a parkolási tanácsadó és információs rendszer javaslatait.

3.2. Gyorsítás

Minden egyes időpillanatban a járművek elmozdulhatnak, amelynek nagyságát annak sebessége határozza meg. Egy parkolási létesítményben a járművezetők alacsony sebességgel közlekednek. Egy felmérés szerint a sofőrök parkolókeresés közben 20-25 km/h sebességgel haladnak; majd ezt további 10-12 km/h-ra csökkentik, miközben megközelítik az általuk kiválasztott parkolóhelyet. Mindazonáltal azt is figyelembe kell venni, hogy a sofőrök az előttük haladó járműhöz is igazítják mozgásukat

3.3. Járműkövetési modellek

A járműkövetési modellek feladata, hogy leírja a járművezető viselkedését az azonos sávban előtte haladó járművel szemben. Már az '50-es évek óta tanulmányozott terület [14] [15], azonban számos összefüggését a mai napig sem sikerült teljesen tisztázni vagy bizonyítani. A modelleket az általuk használt logika alapján különböző típusokra lehet osztani. Ezeket mutatja be a 3.1. táblázat:

3.1. táblázat: A járműkövetési modellek típusai

Modell típus	Alkalmazott logika
Biztonságos távolság modell (Safety-distance model, SDM)	A követő jármű minden esetben egy adott biztonságos távolságot tart az előtte haladó járműtől.
Inger-Válasz Modell (Stimulus–Response Model)	A követést végző jármű gyorsulása arányos az általa követett jármű gyorsulásával, valamint a két jármű közötti sebességkülönbséggel és a rendelkezésre álló követési távolsággal.
Action-Point vagy Pszichofizikai modellek	A vezető bizonyos időpillanatokban (az *akciópontokban*) megváltoztatja a viselkedését, ilyenkor a gyorsulás hirtelen megváltozik, de két pont között állandó marad.

3.3.1. Nagel-Schreckenberg modell

Nagel-Schreckenberg (NaSch, NS) [16] modell a közlekedési sávot egyenlő hosszúságú „cellákra” osztja. Egy ilyen cella vagy üres, vagy pontosan egy darab jármű helyezkedik el benne. Térben diszkrét modellről beszélünk; az ilyen, szabályos rácsozatban elrendezett cellákból álló modellt celluláris automatának hívjuk (Cellular Automaton − CA).
Minden jármű rendelkezik egy v sebességgel, amely csak egész számokat vehet fel, illetve egy v_{max} maximális sebességgel, amely legegyszerűbb esetben az útszakasz sebességkorlátozása.
A modell működését a 7.1 algoritmusban láthatjuk. Ha a jármű még nem érte el a maximális sebességét, akkor egységnyivel gyorsul (2. sor). Ezt követően, ha a jármű sebessége nagyobb, mint az előtte lévő üres cellák száma, akkor a sebességét lecsökkenti ennek nagyságára (5. sor). Ez a lépés valósítja meg magát a járműkövetést. Ezután a jármű P valószínűséggel csökkentheti egységnyivel a sebességét (8. sor). Ez azért szükséges, mert enélkül a modell determinisztikus lenne. Végül megtörténik maga a jármű elmozgatása (10. sor).

3.1. Algoritmus NaSch modell

```
1:  ha v ≤ v_max akkor
2:      v ← v + 1
3:  elágazás vége
4:  ha v > d akkor
5:      v ← d
6:  elágazás vége
7:  ha random ≤ P akkor
8:      v ← v − 1
9:  elágazás vége
10: pos ← pos + v
```

Felhasznált változók és függvények:

- v: A jármű sebessége.
- v_{max}: A jármű maximális sebessége.
- d: A jármű előtti üres cellák száma.
- *random:* Egy véletlen generált szám.
- P: Egy adott P valószínűség.
- *pos:* A jármű aktuális pozíciója.

Ellenben fontos megemlíteni, hogy célravezetőbb ezt a modellt autópálya modellezésénél használni, városi környezetben általában kevésbé hatékony. Problémát jelenthet például az, ha egy utca hossza nem pontosan az adott cellahossz többszöröse, így akár több méteres különbség is jelentkezhet a modell és a valóság között. Ezenfelül nincsenek figyelembe véve az olyan alapvető manőverek, mint a sávváltás, előzés és kikerülés. További hátrányt jelenthet az is, hogy a megoldás nincs tekintettel a különböző típusú járművekre (mint

például kerékpárok, tehergépkocsik és buszok), amelyek eltérő gyorsulással és követési távolsággal rendelkeznek.
Mindezek ellenére a modellnek számos továbbfejlesztése és alkalmazása született, ugyanis jól utánozza a tipikus forgalmi hullámokat, valamint egyszerűen megvalósítható. Sikeresen alkalmazták már parkoló keresés szimulációja során is [15].

3.3.2. Gazis-Herman-Rothery (GHR) modell

A Gazis-Herman-Rothery (GHR) modell alighanem a leginkább kutatott a járműkövetési modellek egyike. Inger-válasz modelleknek is szokás nevezni (*Stimulus–Response Model*) [17], ahol az ingert az elöl haladó és az azt követő jármű közötti sebességkülönbség adja, a válasz pedig a követést végző jármű reakcióidővel késleltetett fékezése vagy gyorsítása.
A modell ötödik és egyben legújabb változata, amelyet időnként GM-5-nek is neveznek, az előző négy típus általánosítása; az *n*-edik jármű *t* időpillanatban mért gyorsulásának meghatározása a következőképp történik (2):

$$a_n(t) = cv_n^m(t)\frac{\Delta v(t-T)}{\Delta x^l(t-T)} \tag{1}$$

ahol: $a_n(t)$ az *n*-edik jármű gyorsulása $[m/s^2]$,
$v_n(t)$ az *n*-edik jármű sebessége $[m/s]$,
Δv az (n – 1)-edik és *n*-edik és jármű közötti sebességkülönbség $[m/s]$,
Δx az (n – 1)-edik és *n*-edik és jármű közötti távolság $[m]$
T a reakcióidő $[s]$,
c, *l* és *m* modellparaméterek (konstansok)

A GHR-modell legfőbb előnye annak egyszerűsége. Ugyanakkor nagy mértékben csak feltételezéseken alapszik, ami jelentős hiányosságokhoz vezet, ahogy erről egyes kutatásokban is beszámoltak. [18] Többek között az egyik probléma abban merül fel, hogy a szabadon áramló forgalomban, amikor a járművek közötti távolság nagy, a modell továbbra is azt feltételezi, hogy a járművezetők a kocsik közötti sebességkülönbségekre reagálnak.
A modell azt feltételezi, hogy a járművezetők észlelni tudják a környezetükben bekövetkező legkisebb változásokat, például a távolságban és a relatív sebességben. Ráadásul a sofőrök egyforma reakcióideje sem tükrözi jól a valóságot. E hiányosságok kiküszöbölésére a GHR-modellnek számos továbbfejlesztett változata született az évek során.

3.3.3. Wiedemann modellek

A Wiedemann egy tipikus pszichofizikai modell, ez alkotja az igen elterjedt közlekedési szimulációs eszköz, a VISSIM alapjait. A Wiedemann modellnek kétféle verziója létezik: a Wiedemann 74 leginkább a városi forgalom modellezésére ideális, a Wiedemann 99 pedig autópálya forgalomra [19]. Ennek tükrében a Wiedemann 74 modellre fogunk a továbbiakban koncentrálni.
A Wiedemann modell alapján a járművezető viselkedése attól függően változik, hogy éppen milyen forgalmi szituációban van: például szabadon halad, vagy alacsonyabb sebességű járművet közelít meg, esetleg épp egy ilyen járművet követ, netán fékezést igénylő kritikus helyzetben van. Az egyes állapotok határai az un. akciópontok (Action Point, AP).
A gépjárművezetők különböző szokásai és viselkedésmódja Wiedemann szerint normális eloszlást követ, ezért az akciópontok számításában több véletlen paraméter is található. [20] Ez azt jelenti, hogy az egyes járművezetők eltérő módon képesek észlelni, felmérni és reagálni a különböző forgalmi szituációkra, mások a biztonsági szükségleteik, valamint az ideális sebességük. Ugyanez vonatkozik az magának a járművek a maximális sebességére és gyorsulási képességére is.
Ez a megközelítés a forgalom mikroszkopikus jellemzőt tekintve igen közel áll a valósághoz, azonban jósága a makroszkopikus mérőszámokra nézve (mint a forgalomsűrűség, átlagsebesség) már nem teljesen tisztázott.

3.3.4. Összegzés

Ezek alapján a megvalósítás során a Wiedemann modell mellett döntöttem, leginkább azért, mert egy létesítménynél különösen fontosak a forgalom mikroszkopikus jellemzői, valamint amiért képes különböző vezetési stílusokat is figyelembe venni.

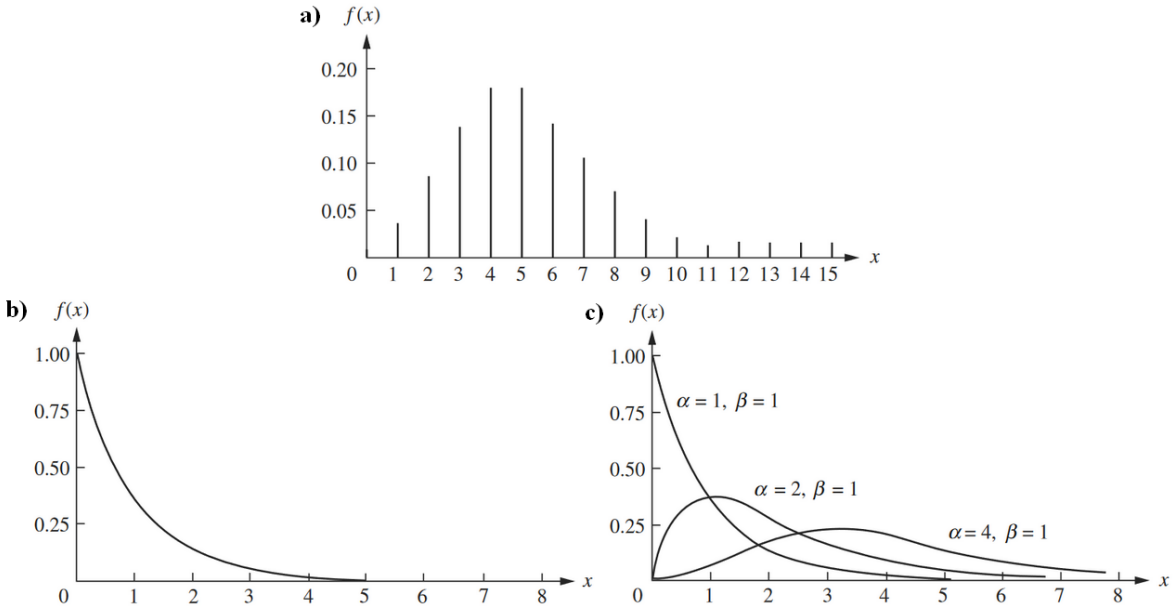
4. PARKOLÓHELY-FOGLALTSÁGI PREDIKCIÓ

4.1. Irodalmi áttekintés

A parkolóhelyek rendelkezésre állásának előrejelzésével kapcsolatos tanulmányokat két fő kategóriába sorolhatjuk, a parkolási folyamatok elemzésén alapuló módszerek, valamint a nagymennyiségű adat elemzésén (*Big Data*) és gépi a tanuláson alapuló módszerekre. Ezeket tekinthetjük elméletvezérelt (deduktív) és adatvezérelt (induktív) megközelítésnek.

4.1.1. Parkolási folyamatok elemzésén alapuló módszerek

Ezen megközelítés a parkolás modellezésén alapszik; az előrejelzés a modell paramétereinek becslésén alapul. A járművek a létesítménybe való beérkezését általában Poisson-folyamatként írják le. Ez tipikusan egy olyan folyamat, amely bizonyos események bekövetkezését számolja (un. *számláló folyamat*). Ilyen például az áruházba beérkező vevők száma, telefonközpontba beérkező hívások mennyisége, radioaktív bomlás során a részecskék darabszáma stb. A létesítményből való távozás esetén, gyakran feltételezik, hogy a parkolás időtartama negatív exponenciális eloszlást [21] vagy gamma eloszlást [22] követ. Ezek szemléltetése a 4.1. ábrán látható.



4.1. ábra: Példák; a Poisson-folyamat alapjául szolgáló Poisson-eloszlás, valamint b) a negatív exponenciális eloszlás és c) a gamma eloszlás (különböző α, β értékek esetén) súlyfüggvényei (Forrás: **[23]**)

Caliskan és mstai. sorbanállás-elméletet alkalmaztak a járművek érkezési és távozási folyamatának modellezésére, amit a folytonos idejű Markov-lánccal írnak le [24].
A sorbanállás-elméletet gyakran alkalmazzák különböző közlekedési áramlatok elemzése során. Jelen esetben a kiszolgálás, amiért a sorbanállás történik, azok maguk az egyes parkolóhelyek, vagyis m db kiszolgáló csatorna van (a parkolóhelyek száma).
Yan és mtsai. a parkolást a Markov-folyamatok egy speciális esetétnek, a születési-halálozási folyamatnak (birth–death process) tekintik, ahol a születés a parkolóhely elfoglalása, megüresedése pedig halálozásnak tekinthető [25].
Ezen megközelítések hátránya, hogy sokszor csak felételezéseken alapul. Előnye viszont, hogy nem feltétlenül szükséges hozzá nagy mennyiségű adat felhalmozása, ez később nehezítheti viszont a modell tesztelését.

4.1.2. Adatelemzésen és gépi a tanuláson alapuló módszerek

A másik megközelítés közvetlenül a mért adatok alapján próbálja jósolni az foglaltság alakulását. Ezen módszerek közé tartozik a regresszió [26], támasztóvektor-regresszió (support vector regression – SVR) [27], integrált autoregresszív és mozgóátlag modell (rövidítve ARIMA) [28], neurális hálózatok [29] [30] [31] és klaszteranalízis [32] [33].
A Zhao és mstai. [19] egy 2020-as tanulmányban összehasonlítást végeztek a különböző típusú és nagyságú parkolók kihasználtságának előrejelzésére. Az általuk vizsgált négy modell (lineáris regresszió, SVR, BPNN és ARIMA) közül az SVR felülmúlta a többit a parkolók kihasználtságának előrejelzésében.
Ezen módszerek hátránya, hogy az adatok előkészítése, tisztítása időigényes lehet. Neurális hálózatok esetén a modell tanítása szintén hosszadalmas művelet. Viszont végeredménye pontosabb, mint az elméletvezérelt megközelítéseknek, mert jobban tud illeszkedni egy adott létesítményhez, ezért gyakorlatban többet alkalmazzák.

4.2. Összegzés

Láthattuk, hogy rengeteg módszer létezik a parkolóhelyek foglaltságának előrejelzésére. Végső soron úgy döntöttem, hogy mindenképp szeretném az induktív megközelítést alkalmazni, mert így valós adatokra támaszkodhatunk, és pontosabb eredményt kaphatunk. A felsorolt módszerek közül a támasztóvektor-regressziót (SVR) választottam.

4.3. Az adatok minőségének javítása, adatdúsítás kérdése

Goodfellow és mtsai. [34] 2014-ben alkották meg az úgynevezet Generative Adversarial Network (GAN) modelleket. Magyar nyelven egyelőre még nem alakult ki egységes elnevezése; generatív „ellenséges”, „versenygő”, esetleg „kontradiktórius” (neurális) hálózatként is hivatkoznak rá. Az egyszerűség kedvéért GAN néven fog szerepelni a továbbiakban.

A GAN két neurális hálózatból áll: egy generátorból (G) és egy diszkriminátorból (D). A generátor adatokat állít elő azzal a céllal, hogy diszkriminátorral elhitesse, hogy a generált adatok valódiak. A diszkriminátor feladata az, hogy megkülönböztesse a valódi és hamis adatokat egymástól. A generátor célja tehát maximalizálni a diszkriminátor hibaarányát, míg a diszkriminátor annak minimalizálására törekszik: a két hálózat egy kétszemélyes, zéró összegű játékban vesz részt.

A kidolgozása óta a GAN jelentős figyelmet kapott a tudományos és kutatói körökben, főleg képgenerálás területén alkalmazzák előszeretettel, számos esetben használták eredményesen jó minőségű szintetikus képek generálására [35] [36] [37], ezzel szemben viszonylag csak kevés munka született a felhasználásával szintetikus idősorok előállítására. Ennek ellenére készültek már olyan tanulmányok, amelyek GAN-okat használtak parkolási adatok generálására. Zhang és mtsai. [38] egy hónapnyi parkolási adatot generáltak egy parkolóház számára feltételes GAN (Conditional GAN – CGAN) [39] segítségével. Ehhez a parkolóház adatain kívül figyelembe vették az időjárást, valamint a parkolóház körül elhelyezkedő POI-k (érdekes helyek, például: boltok, éttermek, parkok) elhelyezkedését.

A GAN alkalmazásának hátránya, hogy még viszonylag kiforratlan, főleg az olyan területeken, ami nem a képgeneráláshoz kapcsolódik. Egyelőre nincs olyan általánosan elfogadott mérőszám, amely számszerűsíteni tudná a modell teljesítményét. Bár szubjektív, de képgenerálás során a modell hatékonysága emberi szemmel könnyedén becsülhető, másfajta adatok jóságát viszont nehéz ilyen közvetlenül megítélni.

5. RENDSZERTERV

5.1. Viselkedés terve

A járműkövetés viselkedési modell megvalósítása a szimulációhoz használt programban fog megtörténni, ami nem más, mint az Unreal Engine 5.
A viselkedési modell a szimulációban található Modell modulnak a kiegészítése lesz (lsd. Kovács Krisztán dolgozata). Tartalmazni fogja azokat az osztályokat és funkciókat, amelyek egy jármű viselkedési állapotát fogják paraméterezni.
Egy jármű viselkedését a parkolóházban 5 fő szakaszra oszthatjuk, melyek a következők:

1. Parkolóhelyhez való eljutás
2. Parkolási manőver a parkolóhely elfoglalásához
3. Parkolás
4. Parkolási manőver a parkolóhely elhagyásához
5. A parkolóház elhagyása

Ezek az állapotok szigorúan csak egymás után, a fenti sorrendben következnek be. Ebből az is következik, hogy a modell azt feltételezi, hogy a járművezető sosem fogja elhagyni a létesítményt úgy, hogy előtte soha nem parkolt le, valamint leparkolás után nem változtatja meg a parkolóhelyét.
A parkolóhelyre való beparkolás man
ővere akkor kezdődik meg, amikor a jármű elérte a számára kijelölt parkolóhelyet. A parkolás időtartalma járművenként eltérő, viszont fix érték, és a parkolóházba való érkezés pillanatában inicializálva van. A letelte után azonnal megkezdi a jármű a parkolóhely elhagyásának folyamatát.
Az 1-es és 5-os szakasz lesz maga a „vezetés” állapota. Itt feltételezzük, hogy a járművezető a KRESZ szabályokat betartja és a parkolóházban elhelyezett KRESZ és iránymutató táblákat mindig figyelembe veszi. Mivel a parkoló nem fog többsávos úttal rendelkezni, így azt is feltételezzük, hogy a járművezető (a kanyarodást kivéve) végig a sávjában próbál maradni, nem kezdeményez kikerülési, előzési manővert.
A modell megalkotásához minden egyes járműnek rendelkeznie kell a következőkkel: aktuális pozíció, sebesség és gyorsulás. Ezeken kívül (a Wiedemann modell alapján) következő értékek tartoznak az egyes járművekhez: álló járműtől számított (a járművezető által biztonságosnak ítélt) optimális távolság, mozgó járműtől való minimális és maximális követési távolság, valamint két érzékelési pont, az egyik, ahol a járművezető észreveszi az előtte lévő járművet, míg a másik, ahol észreveszi, hogy hozzá képest a sebessége lassabb lett. Ezeket felhasználva állapítjuk meg az adott vezetőhöz tartozó, vezetés közbeni viselkedési állapotokat.

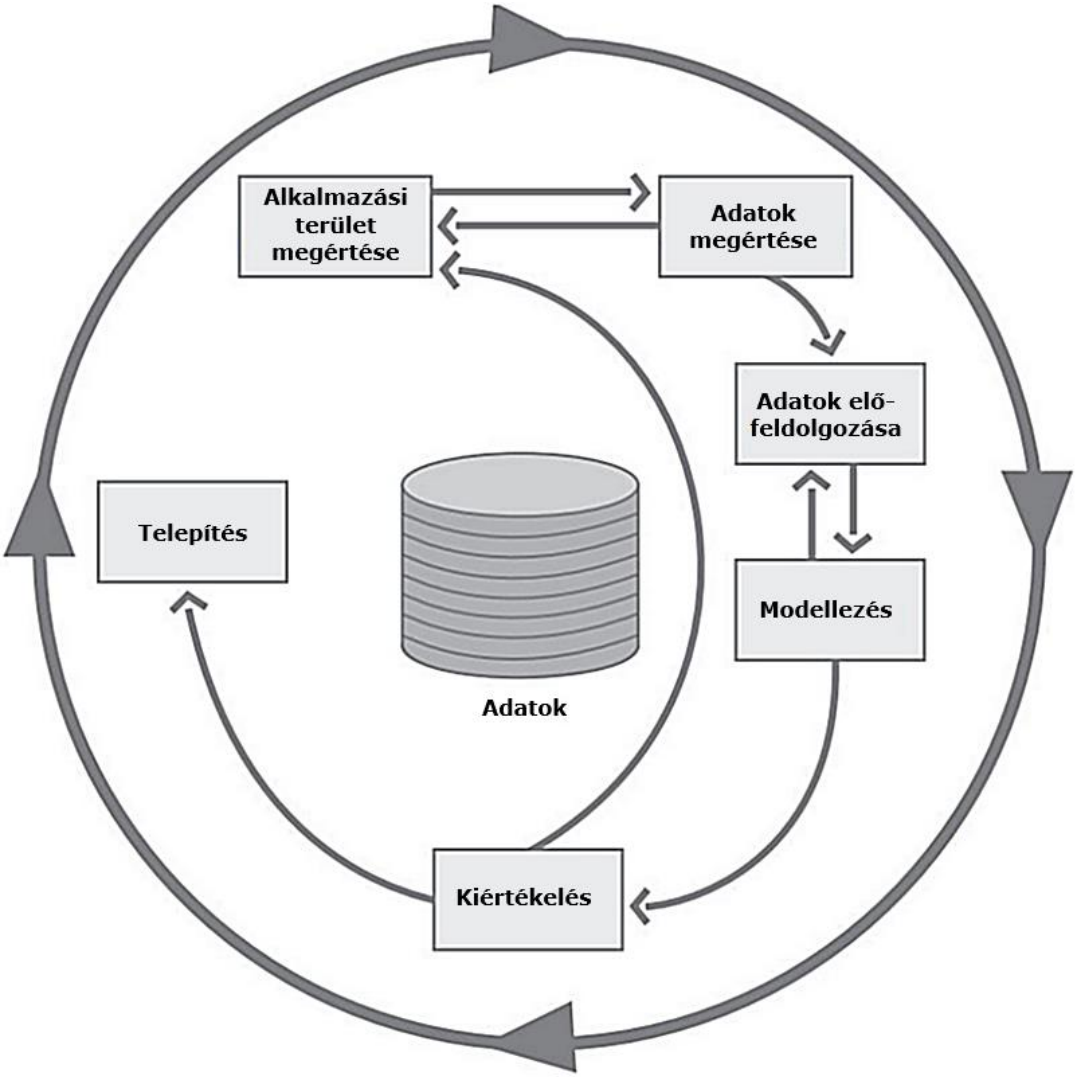
5.2. Előrejelzés terve

5.2.1. Fejlesztési folyamatmodell

A parkolási adatok generálása során a CRISP-DM (Cross Industry Standard Process for Data-Mining) a Főleg az adatbányászatban ismert, ipari szabvány, amely meghatározza a projekt fázisait,
Több, mint 20 éves módszertan, független az alkalmazott konkrét platformtól és eszközöktől. Mind a kereskedelmi, mind a tudományos célú alkalmazások esetében releváns módszertan. CRISP-DM a projekt életciklusát hat fő fázisra osztja, ahogy ezeket a 5.1. ábra is szemlélteti. A fázisok közötti nyilak jelölik a közöttük fennálló leglényegesebb függőségeket, de a sorrendjük nem kötött, bármikor lehetséges köztük az oda-vissza lépés. A legkülső kör-körös nyilak a projekt ciklikus jellegét hivatottak jelképezni, hiszen egy teljes ciklus során szerzett tapasztalatok gyakran további új ötleteket szülnek. [40]
A 5.1. táblázatban az egyes fázisok rövid ismertetése található.

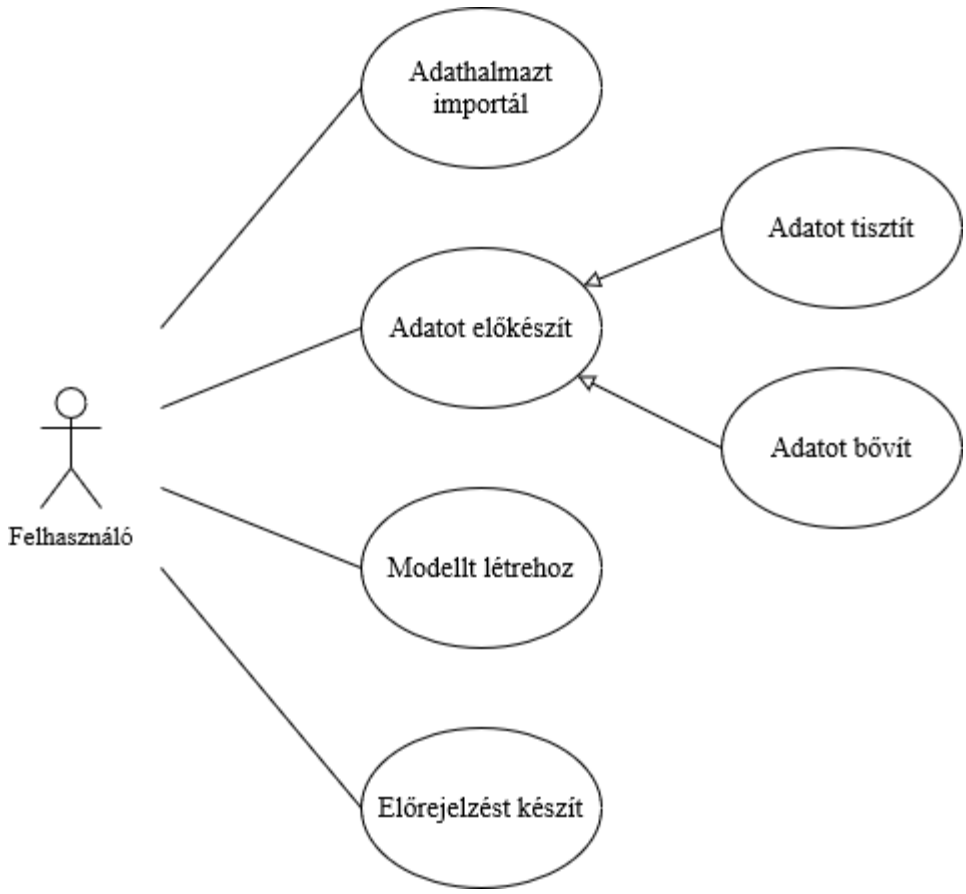
5.1. táblázat: Az egyes fázisok a CRISP-DM modell alapján

Fázis	Rövid Leírása
Alkalmazási terület megértése (*Business understanding*)	A projekt célkitűzéseinek és a követelményeinek megismerésére összpontosít.
Adatok megértése (*Data understanding*)	Kezdeti adatgyűjtés, adatok megismerése, adatok minőségével kapcsolatos problémák feltárása.
Adatok előfelfolgozása (*Data preparation*)	Milyen olyan tevékenység, amely a végleges adathalmaz létrehozásához szükséges.
Modellezés (*Modeling*)	Megfelelő algotitmus kiválasztása és a modell paramétereinek meghatározása
Kiértékelés (*Evaluation*)	Célja megbizonyosodni arról, hogy a modell megfelelően megvalósítja a kitűzött célokat.
Telepítés (*Deployment*)	Integrálás az üzleti folyamatokba, karbantartás.



5.1. ábra: A CRISP-DM módszertan fázisai (Forrás: **[40]**, saját fordítás)

Ehhez szorosan kapcsolódik a használati eset diagram, amelyet az 5.2 ábra mutat. A felhasználó itt nem a végfelhasználót, hanem a fejlesztőt jelöli.



5.2. ábra: A predikciót végző alrendszer használati eset diagramja (Forrás: Saját készítés)

5.3. Technológiák és fejlesztőeszközök

5.3.1. Verziókezelés

Mivel a fejlesztés Kovács Krisztián szaktársammal együtt párhuzamosan zajlik, a megvalósítás megkezdése előtt elengedhetetlen lépés, hogy kiválasszunk egy verziókezelő rendszert a hatékony együttműködés érdekében. Ezek a rendszerek lehetővé teszik a forráskódok ill. más adatok változásainak nyomon követését, így kiküszöbölve például az esetleges kódütközéseket a közös munka során. Mi a projekt megvalósításához a Git verziókezelő rendszert választottuk, mivel ez jelenleg az egyik leggyakrabban használt rendszer, valamint ez számunkra is jól ismert. A verziókezelés megkönnyebbítése és a munkamenet felgyorsítása érdekében, egy felhasználói felülettel ellátott Git alkalmazást

használtunk. Miután mérlegeltük a lehetőségeinket, úgy döntöttünk, hogy a GitHub Desktop alkalmazást fogjuk használni a projekt során, az általa kínált Git és GitHub szolgáltatások miatt. A szoftver mellet szükségünk volt egy adattárhosztoló szolgáltatásra is. Számos megoldást találunk ezen a területen, mint a fent említett GitHub, illetve BitBucket. Végül az úgynevezett „self-hosting" megoldást választottuk, amely nem más, mint az adattároló szerver, saját számítógépünkön való futtatása.
Ahhoz, hogy ezt meg tudjuk valósítani, szükségünk volt egy Apach illetve MySQL szerverre amelyet úgy döntöttünk hogy a XAMPP alkalmazással fogjuk kezelni. Erre a szerverre került fel a GOGS nevezetű verzió kezelő szervert használtuk, amely egy GitHub-ból kialakított verzió. Ezen alkalmazások választásának indoka, nem más, mint hogy korábbi projektek miatt már ismertük és használtuk őket.

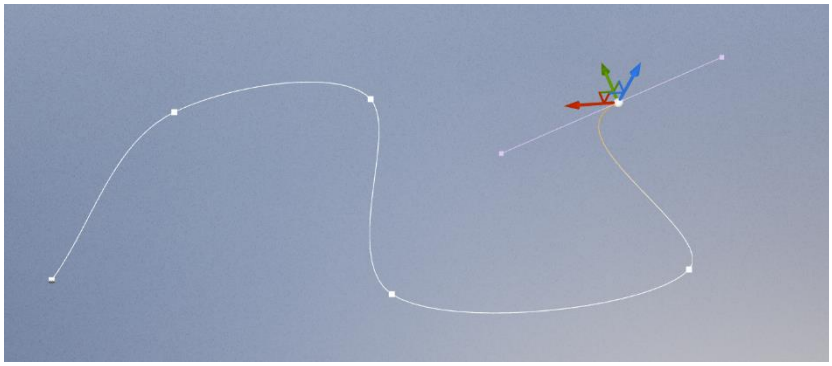
5.4. Felhasznált külső források

Mivel a feladat megvalósítása vizuális környezetben történik, szükségünk van számos 3D modellre, ideértve a járműveket, valamint a parkoló felépítésére használható környezeti tárgyak (például falak, lámpák, oszlopok, stb…). Mivel ezeknek elkészítése számottevő idővel járt volna, úgy gondoltuk, hogy előre elkészített modell csomagokat fogunk használni. A szimuláció megvalósításhoz a Switchboard Studios által készített „Vehicle Variety Pack [41]" -ot alkalmaztuk, amely különböző típusú jármű modelleket tartalmaz. Ezen felül felhasználásra került még a Roman Balakir által készített „Modular Parking [42]" ami a parkolóházhoz rengeteg környezeti elemeket tartalmaz, mint például oszlopok, falak, csövek, parkolóhelyek jelölései.

6. A MEGVALÓSÍTÁS LEÍRÁSA

6.1. Úthálózat felépítése

Az egyes sávok reprezentálására spline görbéket alkalmaztunk, amelyek a világban található 3d koordinátarendszerben pontok halmazát reprezentálja. Ezek a pontok egy egyenessel vannak összekötve, amelyeknek ívét be tudjuk állítani kézzel, mint ahogy a látható a 6.1. ábrán is látható, vagy rábízhatjuk a rendszerre is, hogy automatikusan kezelje ezeket számunkra. Az Unreal Engine szerkesztője lehetővé teszi kifejezetten spline komponensek létrehozását, és használatát, amiket később rengetek formában felhasználhatunk.
Mivel egy komponens egymagában nem használható, illetve nem helyezhető el a pályán, azért egy új, WayPoint nevű aktor részeként hoztuk létre, és azon keresztül használjuk és módosítjuk.



6.1. ábra: Spline létrehozása az Unreal Engine szerkesztőjében (Forrás: Saját készítés)

Ezt a megoldás nemcsak az egyszerű szerkeszthetősége miatt választottuk, de a jövőbeli fejlesztéseket is nagyban elősegíti. A Sumo2Unreal [43] könyvtár használatával a későbbiekben lehetőség van arra, hogy procedurálisan generáljunk spline-okat a projektünkbe *.net.xml* formátumú útadatokból. Így kifejezetten úthálózatok létrehozását és szerkesztését szolgáló programok használatával nagyobb és összetettebb parkolóházak is könnyen létrehozhatók. A netconvert segédprogram segítségével akár a környező utcák úthálózata is gyorsan importálható OpenStreetMap térképadatok alapján [44].

6.2. Jármű modell

6.2.1. Unreal Engine Járműfizikai szimuláció

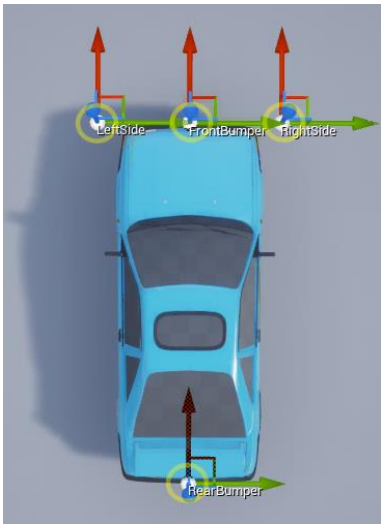
Az Unreal Engine 5, a jelenlegi verziójában (5.0.1) kettő megvalósítást használhatunk arra, hogy egy jármű fizikai tulajdonságait szimuláljuk. Az egyik az nVidia által készített PhysX

fizikai, illetve jármű szimulációs rendszer, valamint az Epic Games által megvalósított Chaos Physics plug-in. A szimuláció fejlesztése során, a Chaos Physics, azon belül is a Chaos Vehicle modult fogjuk alkalmazni, mivel a PhysX könyvtár „legacy" módba került, így már nem kap támogatást, illetve számos funkció hiányszik belőle, az őt leváltó csomaghoz képest. Ennek a plug-innek köszönhetően, számos paramétert és tulajdonságot tudunk finom hangolni a járműveken. A pár legfontosabb tulajdonság közé tartozik:

- Kerekek száma és paraméterei: A jármű egyik legfontosabb komponense a kerék. A programban beállíthatunk számos tulajdonságot, mint például, hogy az adott kereket a motor meghajtja-e, hatással van-e rá a kézifék, illetve mekkora a maximális fordulási szög. Ezen kívül tapadási változókat is képesek vagyunk megadni, illetve a kerék fizikai méretét (szélesség, átmérő) is személyre tudjuk szabni.
- Mechanikai paraméterek: Ebben a szegmensben különböző alkategóriákat találunk, ide tartozik a motor nyomaték, illetve fordulat per perc görbéje, a differenciál zár típusa, illetve csúszási állandója, valamint a váltó tulajdonságai. Itt beállíthatjuk a maximális fokozatok számát, milyen arányok tartoznak a váltóhoz, illetve a váltó típusa: automata vagy kézi.
- Kormányzási beállítások: Ebben a szakaszban tudunk egy egyedileg megadott kormányzási görbét megadni, amely különböző sebességnél különböző erőhatásokat fejt ki a kormányra, ez a szimuláció a sebesség függő kormányzás valós megvalósítása.

6.2.2. Chaos Vehicle Modul – Jármű osztály

A gépjármű fizikai megjelenítéséhez szükségünk van egy úgynevezett `Pawn` osztályra. Ez az osztály a `WheeledVehicleBase`-re épül, amely a jármű megjelenéséért felelős, illetve tartalmazza az egyedileg megírt mozgásvezérlőt is, a `VehicleMovementComponent` formájában. Ezen kívül tartalmazza a modellhez használandó socket nevek paramétereit, az AI vezérlő osztály típusát, illetve egy interfész megvalósítást, amellyel a Game debugger tud kommunikálni. A socket-ek fontos szerepet játszanak a jármű navigálásában. Ez a rendszer képes a modell bármelyik pontján elhelyezni egy "csatlakozási felületet", amely segítségével különböző effekteket, elemeket vagy komponenseket tudunk a járműre rögzíteni. Az általunk a járműn elhelyezett socket-ek helyzete látható a 6.2. ábrán. A mi esetünkben ezek a csatlakozó felületek alapvetően a jármű animálására használt csontváz rendszer egyik tagján alapul, így követve azoknak a mozgását és forgását. A feladat során a legfőbb felhasználási mód a Line Trace elemek rögzítése volt, amelyet az akadályérzékelés miatt használtunk.



6.2. ábra: A sárga karikával jelölt socket-ek elhelyezkedése a modellen (Forrás: Saját készítés)

6.3. A mozgáskomponens

Az Unreal Engine-ben mozgáskomponens (`MovementComponent`) egy absztrakt komponensosztály, feladata biztosítani a mozgást azoknak az aktoroknak, akik rendelkeznek vele. Például a belőle származtatott `RotatingMovementComponent` egy adott forgási sebességgel forgatást végez, míg egy humanoid karakter `CharacterMovementComponent`-je a járást, futást és ugrást valósítja meg.
A Chaos járművek mozgatását is egy speciális mozgáskomponens végzi, ez az úgynevezett `ChaosWheeledVehicleMovementComponent` [45]. Ez a komponens felelős a mozgás során fellépő összes fizikai jelenségért, mint például a kormánykerék, a pedálok pozíciója, a kerekek súrlódása. Négy metódusa kiemelkedő fontosságú ebben a projektben, mivel ezek lehetővé teszik a jármű irányítását, ezeket foglalja össze a 6.1. táblázat.

6.1. táblázat: A Chaos járművek mozgáskomponensének főbb metódusai

Megnevezés	**Leírás**	**Paramérer**
`SetSteeringInput`	A kormánykerék pozíciója.	float érték, -1 és 1 között
`SetThrottleInput`	A gázpedál nyomása.	float érték, 0 és 1 között
`SetBrakeInput`	A fékpedál nyomása.	float érték, 0 és 1 között
`SetTargetGear`	A sebességváltó váltása.	egész érték (-1 hátrafelé, 0 üresjárat, 1+ előre)

A projektünkben a Kovács Krisztián által megvalósított `AIController`-ben a `MoveTo` metódust kell meghívni, hogy a járművet mozgásra utasítsuk, például egy adott parkoló vagy kijárat felé. A kezdő- és célpont közötti útvonalkeresés eredménye általában útszakaszokból áll, esetünkben jellemzően egy ilyen útszakasz a következő olyan kereszteződésig tart, ahol a járműnek irányt kell változtatnia.
Az útvonalon való mozgatáshoz a mozgáskomponens `RequestDirectMove` metódusa hívódik meg, és folyamatosan meghívásra kerül minden egyes képkocka frissítéskor (*tickben*), amíg a jármű el nem éri a megadott úticélt. Benne a paraméterként átadott `MoveVelocity` értéke megadja az `AIController` által meghatározott következő útvonalszakasz irányát és a távolságát a járműtől. Ennek értéke összeszorozva a két képkocka frissítés között eltelt idővel `(DeltaTimeSeconds)`, megadja `az AIController` által igényelt pont aktuális helyzetét a járműhöz képest. Más szóval, a mozgáskomponens csak az útvonal következő szegmensének kezdőpontját ismeri, és nem ismert számára a teljes útvonal vagy akár annak vésző úticélja.
A `RequestDirectMove` metódus felüldefiniálására a `ChaosWheeledVehicleMovementComponent`-ből származó, saját mozgáskomponens került létrehozásra.

6.3.1. A célpont meghatározása

Egy adott útszakaszon való haladáshoz az első lépés annak a pontnak a meghatározása, amelyet a járműnek meg kell közelítenie (`Destination`). Ez alapvetően lehetne maga az `AIController` által igényelt pont, viszont közvetlenül ebbe az irányba való elindulás nem biztosítja például azt, hogy a jármű az úttestnek a megfelelő részén fog haladni.
Erre a problémára nyújt megoldást 6.1. fejezetben bemutatott WayPoint aktor, ugyanis a spline görbék használatával meghatározható, hogy egy adott úttesten hol helyes közlekedni. Esetünkben – mivel csak egysávos, egyirányú útszakaszaink vannak – ez mindig az úttest közepét jelöli. Ezzel szemben például, ha szembejövő forgalom is lehetne, akkor a spline az úttest jobb szélén futna.
A járműnek tehát szüksége van az adott útszakaszon futó WayPointhoz tartozó spline görbéjének a hozzá éppen leközelebb eső pontjához. Ehhez a feladathoz használható a `GetClosestLocationOnSpline` metódus, amely visszaadja egy spline-nak egy megadott ponthoz legközelebb eső pontját. Ehhez azonban meg kell határozni, hogy melyik WapPoint aktor tartozik az aktuális útszakaszhoz.

6.3.2. Aktuális WayPoint aktor meghatározása

A `MoveVelocity` paraméternek a változása azt jelzi, hogy a jármű elérte egy bizonyos útszakasz végét (pl. elérte a kereszteződést). Ehhez a `MoveVelocity` aktuális értékét a komponensben el kell tárolni, és minden `RequestDirectMove` híváskor ellenőrizni kell, hogy megváltozott-e az értéke.
Amikor észleljük, hogy új szakasz kezdődik, a `SetNewRequest` metódus végzi el az ilyenkor végrehajtandó műveleteket. A legfontosabb, amit ellenőrizni kell, hogy a következő útszakaszon melyik WayPoint aktor vezet végig. Ezt a `GetClosestWayPointToLocation` nevű metódus teszi lehetővé, amely visszaadja egy paraméterként átadott adott pozícióhoz legközelebb eső WayPoint mutatóját. Ehhez ismét felhasználjuk az előzőleg említett `GetClosestLocationOnSpline` metódust: minden egyes spline a legközelebbi pontja közül keressük a legkisebb távolságra lévőt.
Így tehát ennek a metódusnak az `AIController`-től kapott pozíció értékét átadva megkapjuk az útszakaszon áthaladó WayPoint aktort.

6.3.3. Kívánt sebesség meghatározása

A jármű kívánt sebességét három tényező befolyásolja: a távolsága a következő útszakasztól (`DistancePercent`), a 6.4. fejezetben ismertetett `VehicleFollowingComponent` által számolt követési sebessége (`VehicleFollowingSpeed`), valamint a 6.5. fejezetben ismertetett `VehicleAvoidanceComponent` által kalkulált fékezés mértéke (`AvoidanceSpeedValue`).

```
(VehicleFollowingSpeed * DistancePercent) - AvoidanceSpeedValue
```

A következő útszakasz távolságának figyelembevétele azért szükséges, mert máskülönben a jármű nem lassítana a kereszteződések előtt, és a kanyarban kisiklana a nagy sebesség miatt. Fontos megjegyezni, hogy ha a jármű nem követ senkit sem (szabadon halad), akkor a `VehicleFollowingSpeed` értéke a jármű maximális sebessége lesz.

6.3.4. A kormánykerék pozíciójának meghatározása

A kormánykerék helyzetét két fő összetevő határozza meg: a járműnek a spline legközelebbi pontjához viszonyított relatív elfordulásának a szöge (-1 és 1 közé normalizálva) (`RawSteeringPosition`) és az akadályok kikerüléséhez szükséges, 6.5. fejezetben ismertetett, a `VehicleAvoidanceComponent` által meghatározott kormányzás mértéke (`AvoidanceCorrectionAngle`).

```
RawSteeringPosition – AvoidanceCorrectionAngle
```

6.3.5. Pedálok és a kormányzás vezérlése

A PID vezérlő egy olyan három változós visszacsatolásos vezérlő, amelyet lineáris rendszereknél előszeretettel használnak. A vezérlő a három végrehajtójel arányos tagjaiból, arányosan adódik össze. Ez a három tag a:

- Hibajel (Angolul: Proportional, jele: P): A jelenlegi hiba mértékét tárolja el, a kívánt cél érték és a pillanatnyi érték közt.
- Hibajel integrál (Angolul: Integral, jele: I): Amúltbeli hibák mértékét tárolja.
- Hibajel változási sebesség (Angolul: Derivative, jele: D): A jövőbeli hibák mértékét tárolja, amely korrelációban áll a jelen, illetve múltbéli hibákkal.

Ennek a kontrollernek a jármű vezérlésében van fontos szerepe. Mivel az Unreal Engine alapértelmezetten nem támogatja a járművek AI által vezérelt mozgását, ezért a sebesség, illetve a kormányzást egyedileg kellet megvalósítani. A projekt tartalmaz egy `FPIDData` nevű struktúrát, amelyben megtalálható a három függvénytag, egy minimális és maximális érték, amely a kimenet eredményét korrigálja a két változó közé, illetve egy `CalcNewInput()` nevű metódus amely az új értékek kiszámolásáért felelős. Ennek köszönhetően a járművet képesek vagyunk a cél helyzete felé kormányozni, illetve a sebesség növelését és csökkentését a gázpedál „nyomásának” hangolásával. A sebesség vezérlésénél a kívánt sebességet, illetve a jelenlegi sebesség közti különbséget adjuk át a kontrollernek, mint hiba, az ebből származó kimeneti eredménye a pedálok lenyomásának erősségét adja meg számunkra, amely negatív esetben fékezést (a fékpedál nyomását) pozitív esetben pedig gyorsítást (gázpedál nyomást) jelent.

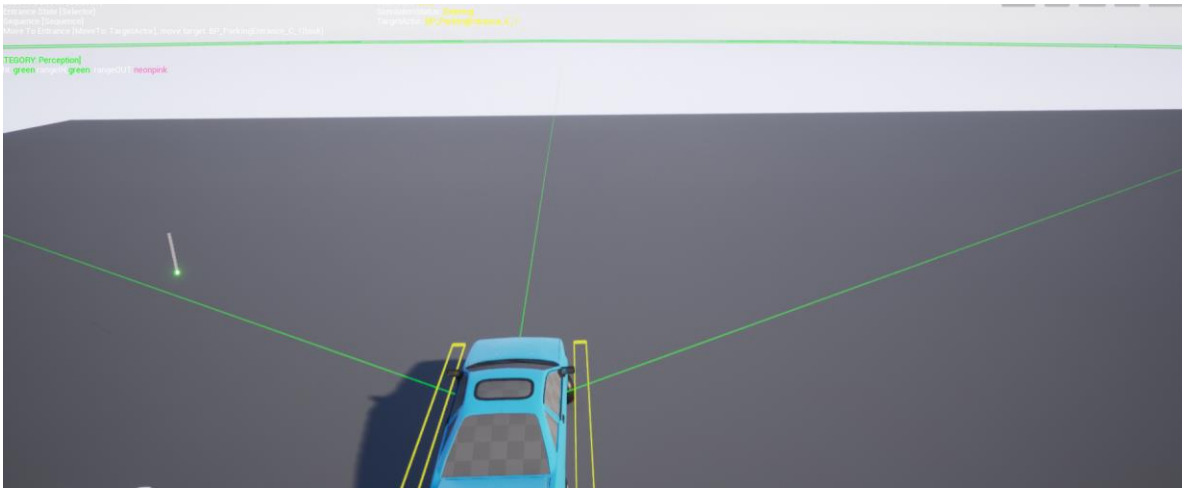
6.4. Járműkövetés megvalósítása

A következő alfejezetben a járműkövetés megvalósítását fogom ismertetni, amelyet a `VehicleFollowingComponent` elnevezésű alkomponens valósít meg.

6.4.1. A követendő jármű kiválasztása

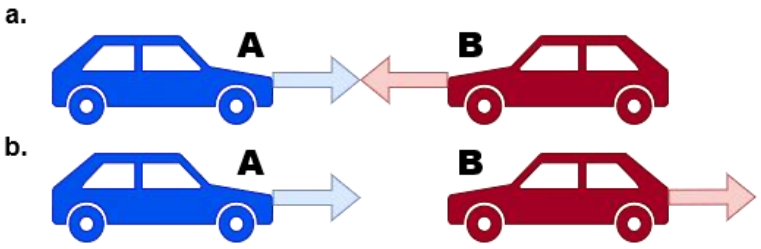
A járműkövetés megvalósításához elsőként meg kellett találni azt a "vezér" járművet, amelyhez képest a járművezető a saját sebességét fogja igazítani.
Ehhez mindenekelőtt meg kell határozni, hogy a járművezető hogyan képes érzékelni a környezetét. Erre a célra az Unreal Engine saját, kifejezetten erre a célra szánt komponense, az AI Perception került alkalmazásra, amely a különböző ingerek észlelésére és azok forrásának rögzítésére szolgál. Ilyen érzékek lehetnek például a látás, hallás, tapintás, amelyeket az `UAISense` típusú osztály definiál. Ezek közül a vezetés során egyértelműen a látás a legfontosabb érzékelési forma, ezért a komponensben ez szerepel domináns érzékelési módként. A látás paraméreit a `UAISenseConfig` objektumon keresztül lehet beállítani. A

vezető perifériás látásszöge 60°, valamint a maximális távolság, amíg érzékelni képes 200 méter. Az Engine képes ezt a látószöget megjeleníteni a beépített vizuális tesztelési eszközeivel, ez látható a 6.3. ábrán.



6.3. ábra: A járművezető látószöge, zöld vonalakkal jelölve (Forrás: Saját készítés)

A `GetCurrentlyPerceivedActors` metódus segítségével elérhetőek a komponens által észlelt különböző aktorok a pályán. Ezek természetesen nem csak járművek lehetnek, hanem szinte bármi, ami a pályán látható. Esetünkben értelemszerűen elegendő csak a `VehicleBasePawn` típusú aktorokat figyelembe venni, azok közül is csak az épp nem parkoló járműveket. Az, hogy egy jármű parkol-e vagy sem, az AI vezérlőjében található a `GetBlackBoardSimulationStatus()` metódussal ellenőrizhető.
A követés további feltétele, hogy a követő és a követett járműnek azonos menetirányba kell nézniük. Ha feltételezzük, hogy adott A jármű előre halad (vagyis nem végez tolatás közben követést), akkor a hozzá képest ellentétesen álló B jármű vagy az ellenkező irányba halad (például a szemközti sávban előre) (6.4.a. ábra), vagy a haladási iránnyal ellenkező irányba tolat (6.4.b. ábra). Egyik helyzetben sem megfelelő a B gépkocsi az A által követendő járműként.

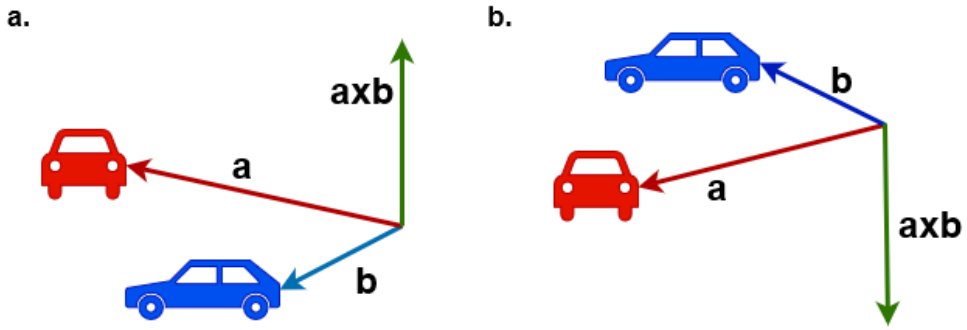


6.4. ábra: A követés és a másik jármű irányának kapcsolata (Forrás: Saját készítés)

A jármű iránya könnyen ellenőrizhető az `ActorForwardVector` segítségével, amely az aktorokhoz tartozó olyan egységvektor, amely ugyanabba az irányba mutat, mint amerre az aktor is éppen áll. A két jármű ezen vektorainak skaláris szorzata megadja a közbezárt szögük koszinuszát, vagyis amennyiben az értéke 0, akkor a két jármű egymásra merőleges, 1 esetén megegyező, míg -1 esetén pedig ellenkező irányba állnak.
Mivel a járművek nem feltétlenül haladnak egymás mögött tökéletes párhuzamban (például kanyarodás alatt), ezért bizonyos fokú eltérés megengedett, amelynek maximális mértékét 60°-ban határoztam meg.

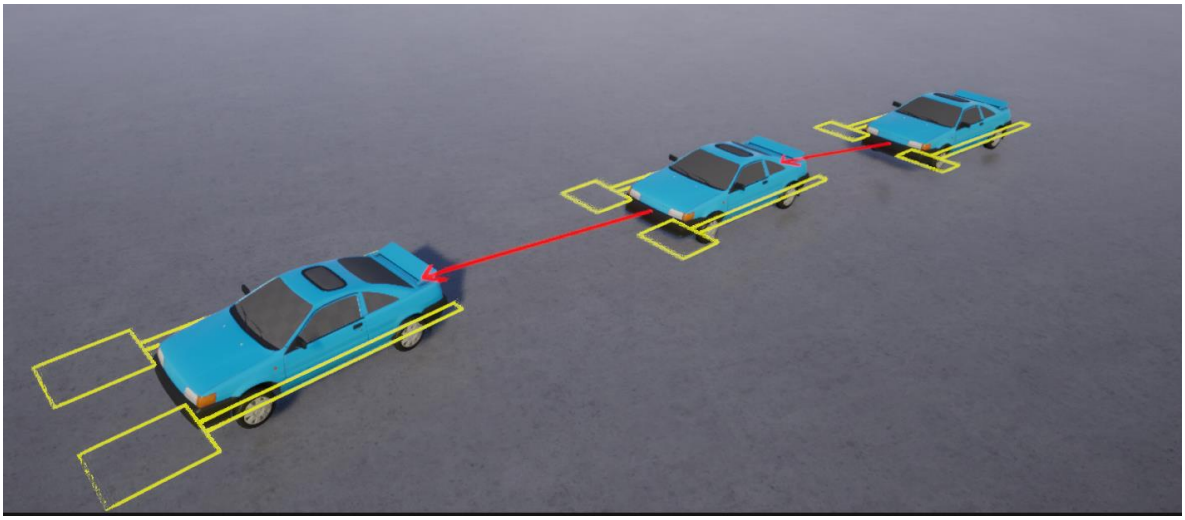
6.4.2. Jobbkéz-szabály

A fejlesztés során megfigyeltük, hogy egy másik jármű „vezér” járműnek való kiválasztása bizonyos mértékben megfelel az elsőbbségadás egy nagyon kezdetleges formájának; azaz a jármű mindaddig lassít, sőt meg is áll, amíg a másik jármű biztonságos távolságba nem kerül tőle. Ha az előbb említett vizsgálat során kivételt teszünk az olyan járművekre, amelyek bár párhuzamosak, de a jobb irányból érkeznek, akkor tulajdonképpen a jobbkéz-szabályt valósítottunk meg.
Azt, hogy a másik jármű a saját pozíciónkhoz képest jobbra vagy balra helyezkedik-e el, egyszerűen ellenőrizhető a helyzetük vektoriális szorzatának z koordinátája alapján. Ha a z koordináta pozitív, akkor a **b** helyzetű járműhöz képest az **a** helyzetű jármű a jobb oldalon található (6.5.a. ábra), ha negatív, akkor pedig a bal oldalon. (6.5.b. ábra),



6.5. ábra: A vektoriális szorzat és a két járműk elhelyezkedésének kapcsolata (Forrás: Saját készítés)

Mivel ebben a szakdolgozatban csak egyenrangú utakkal foglalkozunk, ezért ezt a módszert használtam az elsőbbségadás megvalósítására. Ez a módszer önmagában azonban természetesen nem lenne elegendő, ha további KRESZ szabályokat is figyelembe vennénk.

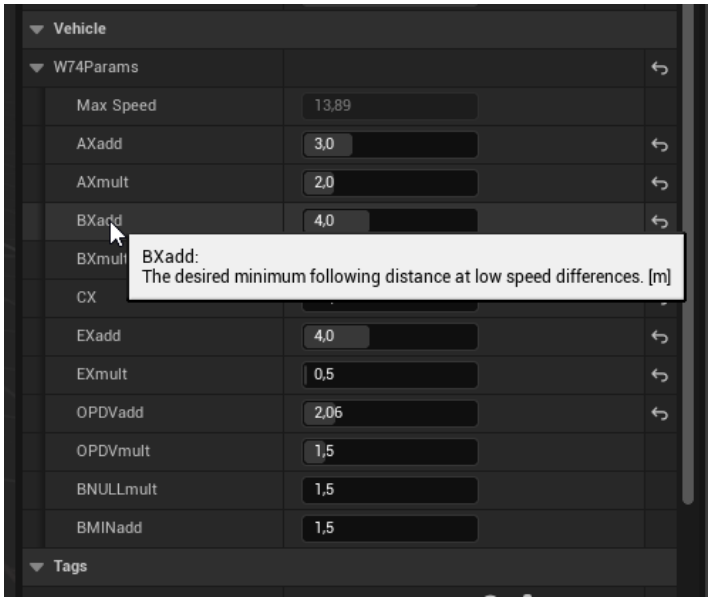


6.6. ábra: A kiválasztott követendő jármű, piros nyilakkal jelölve (Forrás: Saját készítés)

A fenti feltételeknek megfelelő gépjárművek közül a követésre az lesz kiválasztva, amelyik a járműhöz a legközelebb található. Ekkor meghívódik az OnLeaderChanged esemény, amelynek paraméterében átadódik az új, követésre kiválasztott jármű.
A `DrawDebugArrow` függvény használható az eredmény vizualizálására úgy, hogy a követő járműtől a követett járműig egy nyilat rajzolunk, ez látható 6.6. ábrán.

6.4.3. A Wiedemann modell paraméterei

A modell megvalósításához szükséges paramétereket a könnyebb áttekinthetőség érdekében a `Wiedemann74Parameters` nevű struktúra tartalmazza. Ezeket a paramétereket a `UPROPERTY` makróval elérhetővé lehet tenni az Unreal Engine szerkesztője számára, ahogy a 6.7. ábrán is látható. Ez megkönnyíti a paraméterek későbbi kalibrálását anélkül, hogy szükség lenne a forráskód újra fordítására.



6.7. ábra: A paraméterek kalibrálása a szerkesztőben (Forrás: Saját készítés)

6.4.4. A távolság- és sebességkülönbség megállapítása

A követő és a követett jármű közötti távolság- és sebességkülönbség fontos paraméterei a járműkövetési modellnek. A `CalculateDistance` metódus végzi a két jármű közti távolság (dX) megállapítását. A számítása a jármű elülső lökhárítójától kezdődik, és alapesetben a követett jármű hátsó lökhárítójáig tart. Az előző alfejezetben ismertetett kivétel miatt azonban az is lehetséges, hogy a követett jármű eleje van közelebb, ha az épp jobbról érkezik. Ezért a két lehetséges eshetőség közül a közelebb eső lökhárítót kell figyelembe venni a számítás során. A sebességkülönbség számítása a (dV) a `CalculateSpeedDifference` metódusban történik, és a két jármű sebességvektorainak nagysága közötti különbségből adódik.
Egy adott időpillanatban az első-hátsó lökhárító pontos helyzete meghatározható a 6.2.2. fejezetben ismertetett, modellen található socketek segítségével.

6.4.5. A járművezető paraméterei

Ahogy az irodalmi áttekintés is részletezte, a modell szerint minden egyes járművezető más-más viselkedésmóddal rendelkezik. E magatartásformák normális eloszlást követnek, és a `RND1`, `RND2`, `RND3` és `RND4` véletlen változók határozzák meg őket. A generáláshoz a Mersenne Twister 19937 generátort [46] használtam, amely egy sokkal jobb minőségű és gyorsabb véletlengenerátor, mint a C++ saját rand metódusa.

6.4.6. A követési állapot és sebesség meghatározása

Wiedemann négy különböző állapotot definiált, amelyben egy járművezető lehet egy adott pillanatban. Ezek a következők lehetnek: fékezés, megközelítés, követés és a szabad áramlás. A jármű állapotának meghatározása a `CheckVehicleStatus` függvényben történik, és az aktuális állapotot egy `VehicleFollowStatus` felsorolás típusú változó tárolja. Az implementáció nagy részben Wiedemann eredeti algoritmusát követi, azzal a különbséggel, hogy az eredeti algoritmusban a követtetendő jármű távolsága mindig ismert, mivel abban a megvalósításban a követés egyetlen feltétele, hogy a két jármű ugyanazon a sávban haladjon egymás után (még akkor is, ha akár több száz kilométerre vannak egymástól).
A `SetDesiredSpeed` függvény végzi az állapot alapján a kívánt sebesség kiszámítását. Wiedemann eredeti algoritmusa gyorsulásokat számolt, viszont a `DeltaSeconds` időváltozót felhasználva könnyen megkaphatjuk a sebességértékeket is.

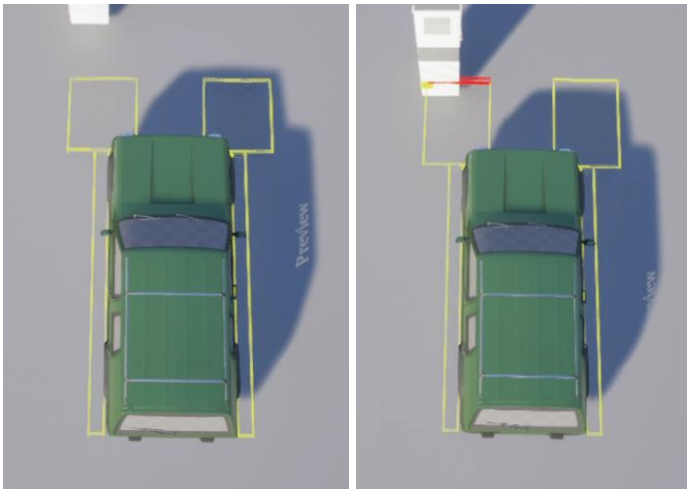
6.5. Kikerülés megvalósítása

A kikerülés azért szükséges, mert egyetlen jármű követése nem elég ahhoz, hogy az úton lévő összes többi járművet és az esetleges akadályokat is figyelembe vegyük.
Az Unreal Engine két beépített kikerülést megvalósító rendszerrel is rendelkezik [47], ám a mi megvalósításunkhoz egyik sem bizonyult megfelelőnek. Egyikük, az úgynevezett *Reciprocal Velocity Obstacles* (RVO) csak a Character (azaz humanoid) aktorokra korlátozódik, ezért nem alkalmazható a járműveken. A másik, a *Detour Crowd Manager*, bár ezen a területen nem limitált, de a nagyobb számítási igénye miatt egyszerre csak korlátozott számú aktort képes kezelni (általában maximum 50 darabot). Ez szintén nem megfelelő számunkra, mert azt szerettük volna, hogy a rendszer ennél egyszerre több járművet tudjon szimulálni. Ezen okok miatt egy saját komponens kezeli az akadályok elkerülését, amely külön, a `VehicleAvoidanceComponent` nevű komponensben valósult meg.

6.5.1. Line-tracing

A kikerülés egyik legfontosabb feladata annak meghatározása, hogy mely irányba kell elkormányozni egy akadály elkerülése érdekében. Bár használhatnánk itt is az AI Perception által adott „látás” képességét, de ez a feladat sokkal egyszerűbben megvalósítható line-tracing segítségével. Ez lényegében ugyanaz, mint a számítógépes grafikából ismert ray-tracing (sugárkövetés), amely során virtuális fénysugarakat "követünk", hogy megállapítsuk, mi található az adott sugár mentén a környezetben. Az alapötlet az, hogyha a gépkocsi első lökhárítójának két oldalán egy-egy ilyen egyenes mentén vizsgálódunk, akkor abban az esetben, ha az egyiknek az útját keresztezi valamilyen akadály, a járműnek meg kell próbálnia elkormányozni az adott irányból. Ráadásul, ha a jármű oldalainak mentén is

hasonló módon fut egy-egy egyenes, akkor az megakadályozhatja, hogy a jármű fennakadjon az éles kanyarokban, mivel így a gépjármű el tud kormányozni tőlük.

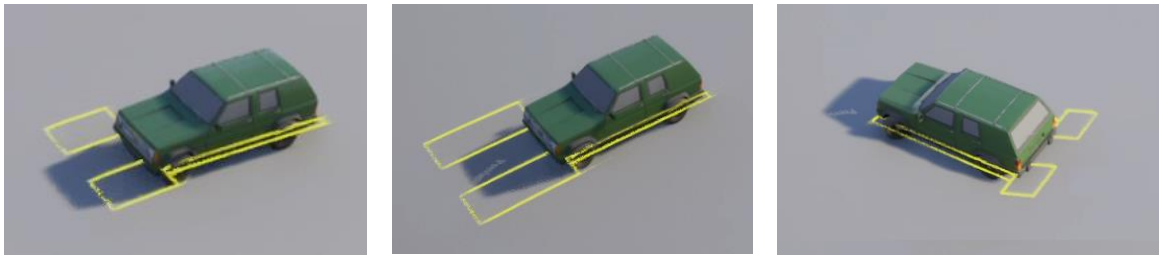


6.8. ábra: A box-trace működése (Forrás: Saját készítés)

Az Unreal Engine-ben számos módja van a line-tracing megvalósításának, mivel ez egy gyakran használt módszer a játékfejlesztésben. Ezek közül a `BoxTraceSingleByProfile`-t alkalmaztam, amely, ahogy a neve is mutatja, egyenes vonal helyett egy "doboz" mentén a térben végzi a vizsgálatot.
A 6.8. ábra a box trace hibakereséshez használt vizuális megjelenítését mutatja. A sárga vonal jelzi a box trace határait, a piros vonal pedig az ütközés helyét.

6.5.2. Box-tracing megvalósítása

A megvalósítás első kérdése a box trace kezdeti és végpontjának meghatározása. Itt valójában két fő esetet kell figyelembe venni: az előremenetet és a hátramenetet. Az előre haladáskor az első lökhárítótól előrefelé, tolatáskor pedig a hátsó lökhárítótól hátrafelé szükséges pásztázni. Hasonlóan a járműkövetéshez, a lökhárítótó széleinek térbeli helyzetét a járművön található socket-ek segítségével lehet meghatározni. Az első két pont helyzete a socketek által adott, míg a hátsó pontokat a jármű hosszával való eltolással lehet meghatározni.
A végpont meghatározásához célszerű a jármű sebességvektorát használni, nemcsak azért, mert az mindig a megfelelő haladási irányba mutat (előremenetben és hátramenetben egyaránt), hanem azért is, mert lehetővé teszi a sebességgel arányos pásztázást. Minél gyorsabban halad a jármű, annál kevesebb idő marad a kitérésre, ezért magas sebességnél érdemes nagyobb távolságon belül keresni az esetleges akadályokat. A fenti helyzeteket a 6.9. ábra szemlélteti.



6.9. ábra: A box-trace elhelyezkedésének és hosszának változása két különböző sebesség, valamint tolatás esetén (Forrás: Saját készítés)

A gépjármű oldalaival párhuzamosan futó box hossza állandó, mindkét oldalon az első lökhárítótól a jármű faráig tart. A korábban használt pontok újra felhasználhatók a kezdő- és végpontok helyzetének meghatározásához.

6.6. Járművek érkezése

Mint ahogy az irodalomkutatásban is írtam, a járművek érkezési idejének megállapításához szerettem volna valós adatokon alapuló módszert alkalmazni. Bár természetesen C++-ban is megvalósítható lett volna, de általában a Python az előnyben részesített programozási nyelv ilyesféle feladatokhoz.

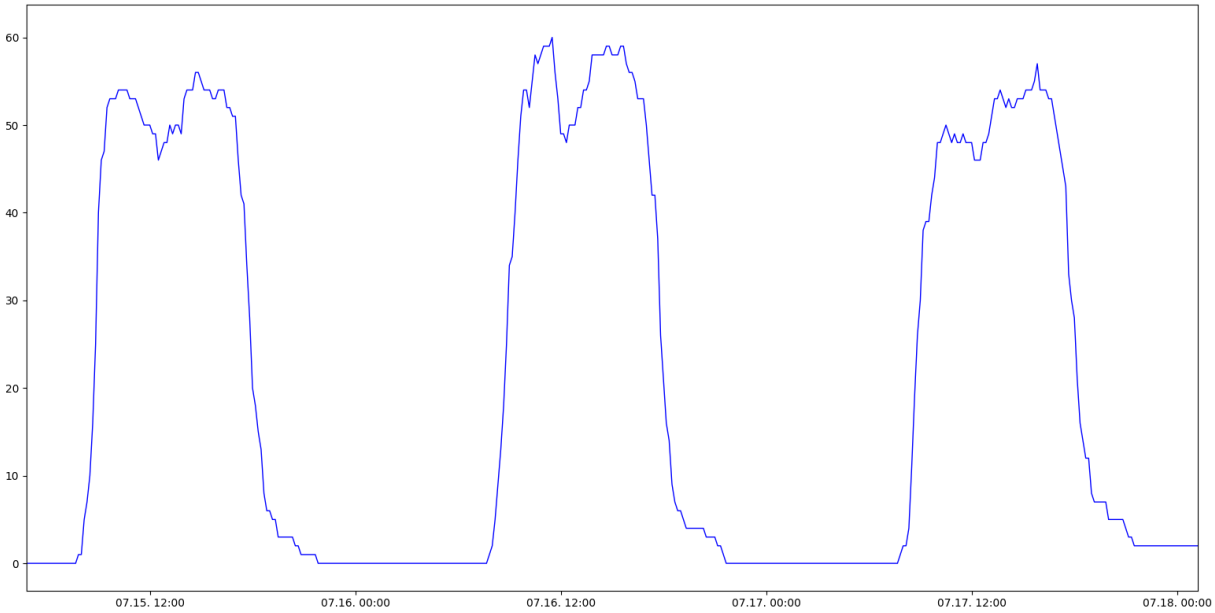
6.6.1. Felhasznált adatok

A megvalósításhoz a Zhang és mtsai. által gyűjtött [48] [49] parkolási adatokat használtam fel, amely projekt részeként 10 különböző parkolási létesítménybe beérkező járműveket vizsgálták meg. A létesítmények között bevásárlóközpontok, szállodák és egy ipari park is szerepelt, mindegyikük önálló fájlokba csoportosítva.
Az adathalmaz 23 hónapot ölelt fel 2017. október 16. és 2019. augusztus 30. között, és összesen 452.480 parkolási rekordot tartalmaz. Minden egyes rekord a jármű az érkezésének és távozásának időpontját, és az adott a parkolásért fizetett összeget tartalmazza, valamint azt, hogy a parkolás 5 percnél tovább tartott-e (az ennél rövidebb ideig tartó parkolást zajnak tekintik).

6.6.2. Adatok előfeldolgozása

A Pythonban egy Pandas nevű csomag valósítja meg az adat keret (data frame) adatszerkezetet, amelyben nagy mennyiségű adat hatékonyan elemezhető vagy módosítható. Az adat keret felépítése hasonlít egy adatbázis-táblához, amely rekordokból (sorokból) és mezőkből (oszlopokból) áll, amelyeknek neve és típusa van. Első lépésként az előbb említett, 5 percnél rövidebb időtartalmú rekordokat távolítjuk el (*dropoljuk*).

Az adatok struktúráját úgy kellett átalakítani, hogy a parkolóban lévő járművek összmennyiségének megváltozásának az idejét rögzítse az érkezés és az indulás helyett. Ez egyszerűen úgy valósítható meg, hogy az összes érkezési és indulási időponton végighaladva (időrendi sorrendben) számontartjuk a járművek aktuális mennyiségét, amelyet érkezéskor növelünk, távozáskor pedig csökkentünk. Az így kapott adatokra láthatunk szemléltető példát a 6.10. ábrán.

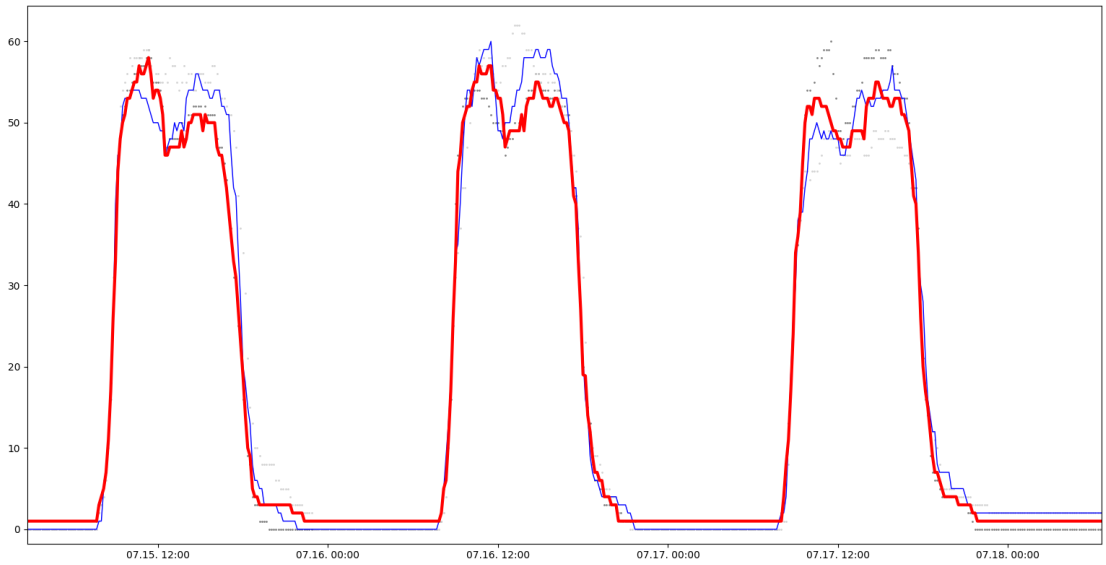


6.10. ábra: Példa három nap összesített parkolási adataira
(Forrás: Saját készítés)

Ezen kívül, csak a hétköznapokat (hétfőtől-péntekig) vettem figyelembe a vizsgálat során. Ahogy Zhao és mstai. [19], én is arra a következtetésre jutottam, hogy a hétvégék és a hétköznapok olyannyira eltérő görbékkel rendelkeznek, hogy sokkal célszerűbb külön vizsgálni őket. A Pandas `dayofweek` metódusával könnyen vizsgálható, hogy adott dátum milyen napra esik.

6.6.3. Támasztóvektor-regresszió megvalósítása

A Támasztóvektor-regresszió megvalósításához a Scikit-learn könyvtár beépített SVR függvényét használtam fel, ennek eredménye látható a 6.11. ábrán. A számítás során az előző két nap, és az egy héttel korábbi (megegyező) nap alapján végeztem jóslást a következő napra. Ezeket az adatokat legegyszerűbben a Pandas shift függvény segítségével kaphatjuk meg, amely egy bizonyos időintervallumban eltolja a rekordokat.



6.11. ábra: Három nap jósolt értékei (piros görbe) és a valódi értékek (kék görbe)
(Forrás: Saját készítés)

6.6.4. Python Szerver

Az Unreal Engine képes Python scripteket futtatni, így például különböző modulokon keresztül képesek vagyunk Tensorflow-t használni. Ebben az esetben a Tensorflow Python API-ja egy beépített Python környezetben fog futni Ennek a környezetnek azonban több hátránya is van. Mivel a motor elsődleges programozási nyelve a C++, az alapokon kívül kevés lehetőség van komplex függvények használatára, ami korlátozza a felhasználási lehetőségeket. Ráadásul a környezet frissítése erősen függ a motor frissítésétől, így a biztonsági javítások és egyéb új funkciók később kerülnek hozzá, de erre nincs garancia. Erre a módszerre épült az úgynevezett Tensorflow Unreal [50] projekt, amely az Unreal Engine 4.27 verziójáig volt támogatott. Az Unreal 5-höz egy új plugin került be az implementálás folyamatába. Az új megoldás egy helyi vagy távoli gépen futatott Python szerverrel köti össze a motort[51], így kibővítve a lehetőségek tárházát, és több funkciót felnyitva a fejlesztők felé. Ennek a kiegészítőnek a működési módja meglehetősen egyszerű. Amikor telepítjük a csomagot, kapunk egy új komponenst, amely a szerver és a motor közötti kommunikációért felelős. Ebből a komponensből kell egyetlen darabot hozzá adni bármilyen tetszőleges aktorhoz, majd amint ez megtörtént, egy vadonatúj beállítási panellel tudjuk

konfigurálni a kapcsolatot. A legfontosabb elemek, amelyeket be kell állítani a megfelelő működés érdekében a:

- Szerver címe és port: Ez a konfiguráció tartalmazza a szerver IP címét és a kommunikációs port-ot, amely a szerverhez való csatlakozásért, és azzal való kommunikációért felel.
- Alapértelmezett Szkript: Az alapértelmezett Python szkript fájl neve, amelyet futtatni szeretnénk a kapcsolat létrejöttekor.
- Szerver típusa: Itt tudjuk beállítani, hogy milyen típusú szerverrel kell a rendszernek kommunikálnia, például Python szerver vagy PyTorch szerver.

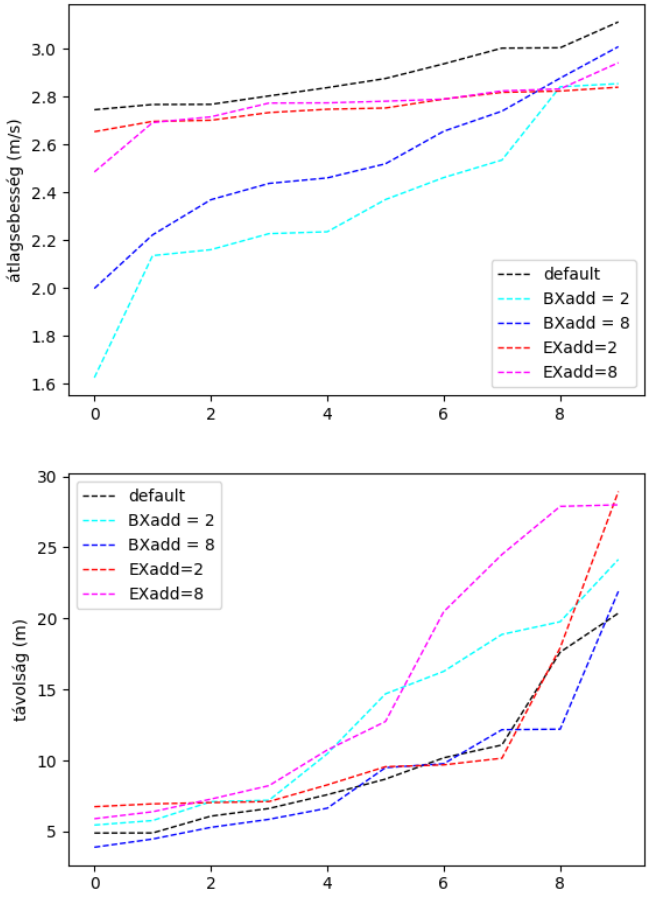
Amikor az alapvető értékeket beállítottuk akkor a `MachineLearningRemoteComponent-et` megvalósító aktoron keresztül kiküldésre kerül a feldolgozandó adat vagy lehetséges kérések, JSON formában.

7. TESZTELÉS

7.1. Viselkedés tesztelése

A Wiedemann-modell paramétereinek megváltoztatásával lehetővé válik a különböző viselkedésű járművezetők összehasonlítása. Mivel engem kifejezetten a követéshez kapcsolódó viselkedés érdekelt, a tesztpályán felállítottam egy kör alakú utat, ahol a járművek folyamatosan reagálhatnak egymásra azáltal, hogy körben haladnak rajta. A teszteléshez a mozgáskomponensben ki kell kapcsolni `WayPointFollowingEnabled` értéket, így lehet a járműveket a parkolóktól és kijáratoktól független mozgásra bírni.
Két tulajdonságát vizsgáltam ezalatt a járműveknek: az átlagsebességük és az átlagos távolságuk a követett járműtől A tesztelés során az alapértelmezett paraméterek mellett 4 további esetet vizsgáltam meg:

- alacsony és magas minimum követési távolság (BXadd)
- alacsony és magas maximális követési távolság (EXadd)

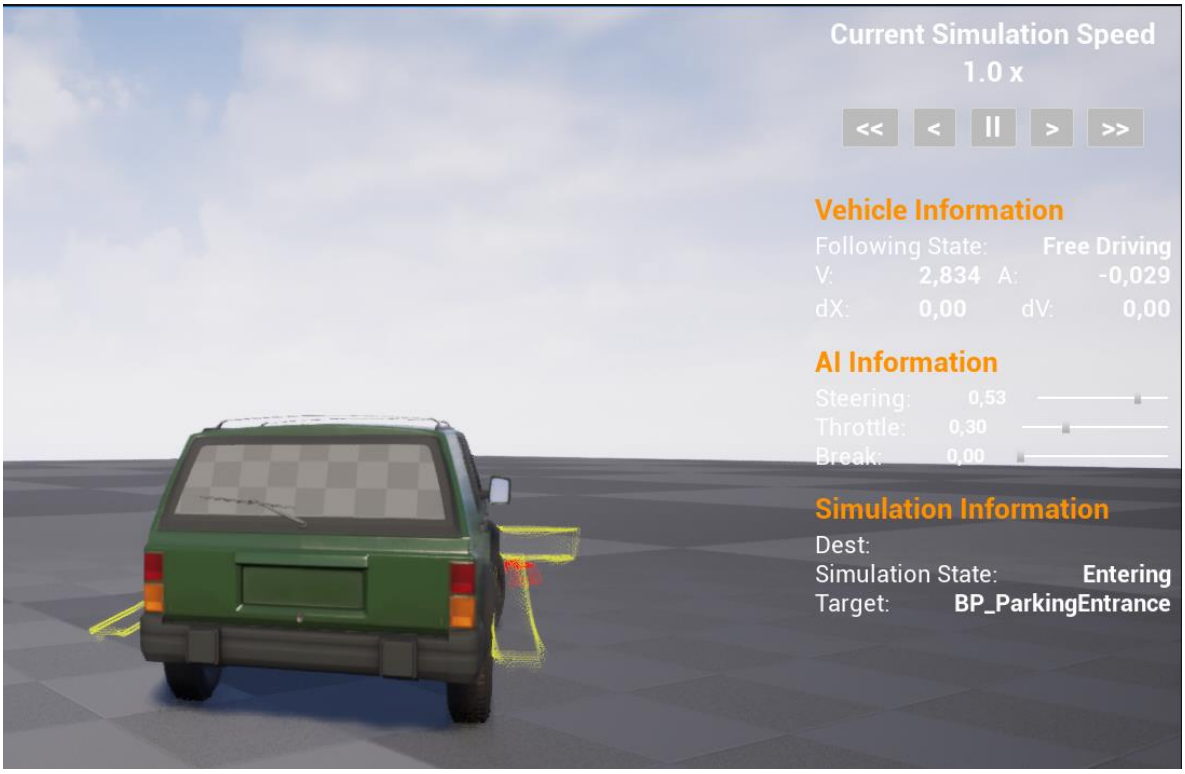


7.1. ábra: A tesztelés eredményei (Forrás: Saját készítés)

7.1. ábrát vizsgálva észrevehető, hogy a minimum követési távolság csökkentésével (`BXadd=2`) együtt drasztikusan lecsökken egyes járművek átlagsebessége. Ennek oka az, hogy ha a követett jármű váratlanul lefékez, a kis követési távolság miatt nagyobb fékezőerőre van szükség, és ezután a járműnek sokkal több időbe telik újra felgyorsulnia. Ugyanez a jelenség figyelhető meg a második grafikonon is, ahol ezek miatt a „lemaradások” miatt nő a követő és a követett jármű közötti átlagos távolság.
Ezzel szemben a minimum követési távolság megnövekedésével (`BXadd=8`) ismét meredekebb görbét kapunk, de az átlagsebesség már nem csökken olyan drasztikusan. A járművek ekkor hamarabb reagálnak az általuk követett járműre, így sokkal dinamikusabban haladnak, gyakran gyorsítanak és lassítanak, de az előző esettől eltérően már nem vészfékeznek, így a sebesség nem csökken le annyira, hogy túlságosan lemaradjanak egymástól.
Nagy maximális követési távolság esetén (`EXadd=8`) megnő a járművek közötti távolság, a járművek sebessége egyenletes, mivel elegendő távolság áll rendelkezésre a sebességük korrigálásához. Ezzel szemben, alacsony maximális követési távolság esetén (`EXadd=2`) a jármű a minimális követési távolsághoz közeli kis területen próbálja megtartani a pozícióját, ami szinte "türelmetlennek" és "erőszakosnak" tűnő vezetési stílust eredményez. Ilyen esetekben a követési távolság viszonylag alacsony marad, de egyes esetekben egy hirtelen fékezés miatt meredeken megnő, az előbb említett vészfékezés utáni lemaradás problémája miatt.
Ezek alapján jól látszik, hogy csupán ennek a két paraméternek a megváltozása mennyire különböző vezetési stílusokat eredményez.

7.2. Game debugger

A Game debugger egyik fontos szereplője a jármű adatait megjelenítő “Widget” grafikai felület. Ha egy járművet kiválasztunk, akkor az 7.2. ábrán látható módon különböző adatok kerülnek megjelenítésre, mint például a jármű sebessége (m/s), gyorsítási változó, fékezési változó, szimulációs státus, üzlet úti cél, illetve a jelenlegi cél, amelyet az útvonal keresés határoz meg.



7.2. ábra: A game debugger widget, amely tartalmazza a jármű pillanatnyi adatait
(Forrás: Saját készítés)

Ezen információk lekérése egy interfészen valósul meg, amely összeköti a felhasználó felé kirajzolt felhasználói felületet a járművel, így amikor a felhasználó rá kattint egy autóra, az interfészen keresztül meghívódik egy esemény, amely megjeleníti az adott pillanatban lévő legfrissebb értékeket, illetve ezeket frissíti. Ez nagyon hasznos volt a megvalósítása során, mivel így ellenőrizhető volt a számítások helyessége.

8. TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

8.1. A járművek mozgásának továbbfejlesztési lehetőségei

A járművek mozgásának továbbfejlesztése sokféleképpen lehetséges. Az egyik legfontosabb kérdés a parkolás alatti viselkedés pontosítása. A mostani dolgozatban a járművek ugyan képesek merőleges parkolásra (egy speciális, parkolóhelyhez tartozó spline segítségével), de nem képesek értékelni a parkolás pontosságát, azaz, hogy teljesen merőlegesek-e a parkolóhelyhez képest, szükséges-e bármilyen utólagos korrigáció.
További lehetőség a WayPoint-ok továbbfejlesztésére úgy, hogy tárolják a haladási irányt is valamilyen módon, így lehetnének többirányú és többsávos utak. A többsávos utakon ilyenkor kérdés a sávváltás megvalósítása, ami korántsem olyan részletesen kutatott terület, mint az egyetlen sávban való viselkedés. [52]
Ahogy a 6.1. fejezetben is említettem, a Sumo2Unreal könyvtár alkalmazásával számos további lehetőség is megnyílna a projektben, például a környező utcák úthálózata is importálható lehetne OpenStreetMap térképadatok alapján. Számos kutatás foglalkozik a közterületi parkolók és a létesítmények együttes összefüggéseivel [53].

8.2. A járművek érkezésének továbbfejlesztési lehetőségei

Mindenféleképpen hasznos lenne további előrejelzési módszereket alkalmazni és összehasonlítani. Szerencsére az Unreal Engine-től független Python szerver nem korlátozza, hogy milyen külső könyvtárakat használunk, így akár a Tensorflow által biztosított gépi tanulási algoritmusokat is fel lehetne használni.

9. TARTALMI ÖSSZEFOGLALÓ

Dolgozatom célja egy virtuális okos-parkolóház számára realisztikus bementi adatokat szolgáltatni.

Az irodalomkutatás során olyan alapfogalmakat mutattam be, mint a parkolás, a modell és a szimuláció. Megvizsgáltam az intelligens közlekedési rendszerek alapfogalmait és feladatait, beleértve az intelligens parkolási rendszereket és a PGI rendszereket. Bemutattam a jelenlegi parkolási helyzetet és azon belül is a parkolóházakkal kapcsolatos problémákat, azok valószínű okait, valamint lehetséges megoldásait.

A járművezetők viselkedésének megértése érdekében összehasonlítottam különböző példákon keresztül járműkövetési modellek típusait, azoknak az előnyeit és hátrányait. Ezekből kiválasztottam a saját megvalósításomhoz a legmegfelelőbb modellt.

Bemutattam a parkolóházba érkező járművek számának becsléséhez alkalmazható különféle módszereket. Végül összeállítottam a rendszer tervét.

A megvalósítás során megoldást találtam az Unreal Engine-ben elérhető járműfizikai modell irányítására, a kormánykerék és a pedálok vezérlésére.

Megvalósítottam a Wiedemann modell alapján a járműkövetést, amely magában foglalta a járművezető érzékelését, a követendő jármű kiválasztását, a követési állapot és a sebesség meghatározását ez alapján. Továbbá létrehoztam egy komponenst a járművekhez, amely box-tracing technikát használ a további akadályok elkerülésére.

A járművek érkezésének meghatározásához létrehoztam egy, az Unreal Engine-től elkülönített Python szervert, amely valós adatok alapján képes egy héttel korábbi adatok alapján becslést végezni a beérkező járművek számáról.

Eltérő viselkedésű járműveket teszteltem, megvizsgálva különböző jellemzőiket és az emögött rejlő magyarázatokat. Végezetül további fejlesztési lehetőségeket mutattam be, és elképzeléseket vázoltam fel a megvalósításukra vonatkozóan.

Fontos megemlíteni, hogy jelen dolgozat párhuzamosan íródott Kovács Krisztián szaktársam azonos című szakdolgozatával. Az Ő feladata az itt kapott adatokat felhasználva a beérkező járműveket ideális útvonalon elvezeti egy alkalmas parkolóhelyre. Ebből kifolyólag a két dokumentáció tartalmazhat megegyező témaköröket, és csak együtt adja vissza a teljes parkolóház megvalósítását.

10. ABSTRACT

The aim of my thesis is to provide realistic parking data for a virtual smart parking garage.
During the literature search I introduced basic concepts such as parking, model and simulation. I studied the basic concepts and tasks of intelligent transportation systems, including intelligent parking systems and PGI systems. I presented the current parking situation, including the problems associated with parking garages, their likely causes and possible solutions.
In order to understand driver behaviour, I compared different types of vehicle following models, their advantages and disadvantages through different examples. From these, I selected the most suitable model for my own implementation.
I have presented different methods for estimating the number of vehicles entering the car park. Finally, I have drawn up the design of the system.
During the implementation, I found a solution to control the vehicle physics model available in Unreal Engine, including the steering wheel and pedals.
I implemented the vehicle following based on the Wiedemann model, which included driver sensing, selection of the leading vehicle, tracking state and speed determination based on this. I also created a component for vehicles that uses box-tracing techniques to avoid additional obstacles.
To determine the arrival of the vehicles, I created a Python server separate from the Unreal Engine that can estimate the number of incoming vehicles based on real data from a week earlier.
I tested vehicles with different behaviours, examining their different characteristics and the explanations behind them. Finally, I presented further possibilities for improvement and outlined ideas for their implementation.
It is important to mention that this thesis was written in parallel with the thesis of the same title by my colleague Krisztián Kovács. His task is to use the data obtained here to guide the incoming vehicles along an ideal route to a suitable parking place. Consequently, the two documents may contain the same topics and only together will give the complete parking garage.

11. IRODALOMJEGYZÉK

[1] A. Tóth, „A Budapesti P+R Parkolás Informatikai Támogatása Applikáció Fejlesztésével”, Budapesti Műszaki és Gazdaságtudományi Egyetem, Budapest, 2016.

[2] „56/2017. (III. 20.) Korm. rendelet egyes kormányrendeleteknek az »okos város«, »okos város módszertan« fogalom meghatározásával összefüggő módosításáról”.

[3] R. Giffinger, C. Fertner, H. Kramar, R. Kalasek, N. Milanović, és E. Meijers, *Smart cities - Ranking of European medium-sized cities*. 2007.

[4] S. Dirks és M. Keeling, „A vision of smarter cities”, IBM Global Business Services, GBE03227-USEN-04, 2009.

[5] *Az Európai Parlament és a Tanács 2010/40/EU irányelve az intelligens közlekedési rendszereknek a közúti közlekedés területén történő kiépítésére, valamint a más közlekedési módokhoz való kapcsolódására vonatkozó keretről*. Strassburg, 2010.

[6] C. Csiszár és Z. P. Sándor, *Közlekedési informatika*. BME Tanárképző Központ, 2014.

[7] S. Shaheen, „Smart Parking Management Field Test: A Bay Area Rapid Transit (BART) District Parking Demonstration”, 2005, Elérés: 2021. október 8. [Online]. Elérhető: https://escholarship.org/uc/item/6d58554x

[8] T. Daiki, G. Siegler, P. Szlávi, és L. Zsakó, *Modellezés és szimuláció*. Budapest: Eötvös Loránd Tudományegyetem Informatikai Kar, 2012. Elérés: 2021. december 8. [Online]. Elérhető: https://dtk.tankonyvtar.hu/handle/123456789/3937?show=full

[9] G. Geda, *Modellezés és szimuláció az oktatásban*. Kempelen Farkas Hallgatói Információs Központ, 2011.

[10] D. Shoup, „Cruising for parking”, *Transp. Policy*, köt. 13, o. 479–486, febr. 2006, doi: 10.1016/j.tranpol.2006.05.005.

[11] D. A. Burdette, „An Investigation of Advanced Parking Information Systems at Airports”, *Compend. Grad. Stud. Pap. Adv. Surf. Transp. Syst. 1999*, o. 79–112, 1999.

[12] Erő Z., Bardóczi S., Germán T., Ekés A., Sipos Z., és Fehérvári Z., „Újlipótváros parkolásfejlesztési tanulmány”. 2015. Elérés: 2021. november 24. [Online]. Elérhető: https://www.kozszolgaltato.bp13.hu/letoltheto-dokumentumok/

[13] G. J. Alexander és H. Lunenfeld, *A users’ guide to positive guidance*, 3rd ed. Washington, D.C. : Springfield, VA: U.S. Department of Transportation, Federal Highway Administration, Office of Traffic Operations ; National Technical Information Service distributor, 1990.

[14] L. A. Pipes, „An Operational Analysis of Traffic Dynamics”, *J. Appl. Phys.*, köt. 24, sz. 3, o. 274–281, márc. 1953, doi: 10.1063/1.1721265.

[15] R. E. Chandler, R. Herman, és E. W. Montroll, „Traffic Dynamics: Studies in Car Following”, *Oper. Res.*, köt. 6, sz. 2, o. 165–184, ápr. 1958, doi: 10.1287/opre.6.2.165.

[16] K. Nagel és M. Schreckenberg, „A cellular automaton model for freeway traffic”, *J. Phys. I*, köt. 2, o. 2221, dec. 1992, doi: 10.1051/jp1:1992277.

[17] A. Ferrara, S. Sacone, és S. Siri, „Microscopic and Mesoscopic Traffic Models”, in *Freeway Traffic Modelling and Control*, A. Ferrara, S. Sacone, és S. Siri, Szerk. Cham: Springer International Publishing, 2018, o. 113–143. doi: 10.1007/978-3-319-75961-6_5.

[18] S. Siuhi és M. S. Kaseko, „Parametric Study of Stimulus-Response Behavior for Car-Following Models”, előadás Transportation Research Board 89th Annual Meeting, Washington DC, United States, 2010.
[19] Z. Zhao, Y. Zhang, és Y. Zhang, „A Comparative Study of Parking Occupancy Prediction Methods considering Parking Type and Parking Scale”, *J. Adv. Transp.*, köt. 2020, o. e5624586, febr. 2020, doi: 10.1155/2020/5624586.
[20] K. Fadhloun, A. Loulizi, és J. Wang, „A Validation Study of the Fadhloun-Rakha Car-following Model”, jan. 2020, o. 180–192. doi: 10.5220/0009435501800192.
[21] A. Millard-Ball, R. Weinberger, és R. C. Hampshire, „Is the Curb 80% Full or 20% Empty? Assessing the Impacts of San Francisco’s Parking Pricing Experiment”, Social Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 2338230, jan. 2014. doi: 10.2139/ssrn.2338230.
[22] F. Caicedo, C. Blazquez, és P. Miranda-Gonzalez, „Prediction of parking space availability in real time”, *Expert Syst. Appl.*, köt. 39, o. 7281–7290, jún. 2012, doi: 10.1016/j.eswa.2012.01.091.
[23] G. Roussas, „Chapter 6 - Some Special Distributions”, in *Introduction to Probability (Second Edition)*, G. Roussas, Szerk. Boston: Academic Press, 2014, o. 99–136. doi: 10.1016/B978-0-12-800041-0.00006-7.
[24] M. Caliskan, A. Barthels, B. Scheuermann, és M. Mauve, „Predicting Parking Lot Occupancy in Vehicular Ad Hoc Networks”, in *2007 IEEE 65th Vehicular Technology Conference - VTC2007-Spring*, ápr. 2007, o. 277–281. doi: 10.1109/VETECS.2007.69.
[25] G. Yan, W. Yang, D. B. Rawat, és S. Olariu, „SmartParking: A Secure and Intelligent Parking System”, *IEEE Intell. Transp. Syst. Mag.*, köt. 3, sz. 1, o. 18–30, 2011, doi: 10.1109/MITS.2011.940473.
[26] E. McGuiness és S. McNeil, „Statistical Models to Predict Commercial- and Parking-Space Occupancy”, *J. Urban Plan. Dev.*, köt. 117, sz. 4, o. 129–139, dec. 1991, doi: 10.1061/(ASCE)0733-9488(1991)117:4(129).
[27] Y. Zheng, S. Rajasegarar, és C. Leckie, „Parking availability prediction for sensor-enabled car parks in smart cities”, in *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, ápr. 2015, o. 1–6. doi: 10.1109/ISSNIP.2015.7106902.
[28] X. Zhu, J. Guo, W. Huang, F. Yu, és B. B. Park, „Real Time Short-term Forecasting Method of Remaining Parking Space in Urban Parking Guidance Systems”, *PROMET - TrafficTransportation*, köt. 30, sz. 2, o. 173–185, ápr. 2018, doi: 10.7307/ptt.v30i2.2388.
[29] Y. Ji, D. Tang, P. Blythe, W. Guo, és W. Wang, „Short-term forecasting of available parking space using wavelet neural network model”, *IET Intell. Transp. Syst.*, köt. 9, sz. 2, o. 202–209, 2015, doi: 10.1049/iet-its.2013.0184.
[30] Y. Rong, Z. Xu, R. Yan, és X. Ma, „Du-Parking: Spatio-Temporal Big Data Tells You Realtime Parking Availability”, in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, New York, NY, USA, júl. 2018, o. 646–654. doi: 10.1145/3219819.3219876.
[31] S. Yang, W. Ma, X. Pi, és S. Qian, „A deep learning approach to real-time parking occupancy prediction in spatio-termporal networks incorporating multiple spatio-temporal data sources”, *ArXiv*, 2019.

[32] F. Richter, S. D. Martino, és D. C. Mattfeld, „Temporal and Spatial Clustering for a Parking Prediction Service”, in *Proceedings of the 2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, USA, nov. 2014, o. 278–282. doi: 10.1109/ICTAI.2014.49.
[33] A. Tamrazian, Z. (Sean) Qian, és R. Rajagopal, „Where is My Parking Spot?: Online and Offline Prediction of Time-Varying Parking Occupancy”, *Transp. Res. Rec.*, köt. 2489, sz. 1, o. 77–85, jan. 2015, doi: 10.3141/2489-09.
[34] I. J. Goodfellow *és mtsai.*, „Generative Adversarial Networks”, *ArXiv14062661 Cs Stat*, jún. 2014.
[35] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, és H. Lee, „Generative adversarial text to image synthesis”, in *International Conference on Machine Learning*, 2016, o. 1060–1069.
[36] M.-Y. Liu és O. Tuzel, „Coupled generative adversarial networks”, in *Advances in neural information processing systems*, 2016, köt. 29.
[37] Z. Zhang *és mtsai.*, „Customizable GAN: A Method for Image Synthesis of Human Controllable”, *IEEE Access*, köt. 8, o. 108004–108017, 2020, doi: 10.1109/ACCESS.2020.3001070.
[38] J. Zhang, M. Zhu, és L. Peng, „Customized Parking Data Generation based on Multi-conditional GAN”, in *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, Rhodes, Greece, szept. 2020, o. 1–6. doi: 10.1109/ITSC45102.2020.9294436.
[39] M. Mirza és S. Osindero, „Conditional generative adversarial nets”, *ArXiv Prepr. ArXiv14111784*, 2014.
[40] P. Chapman *és mtsai.*, „CRISP-DM 1.0: Step-by-step data mining guide”, *undefined*, 2000, Elérés: 2021. december 7. [Online]. Elérhető: https://www.the-modeling-agency.com/crisp-dm.pdf
[41] „Vehicle Variety Pack in Blueprints - UE Marketplace”. https://www.unrealengine.com/marketplace/en-US/product/bbcb90a03f844edbb20c8b89ee16ea32 (elérés 2022. május 6.).
[42] „Modular Parking in Environments - UE Marketplace”, *Unreal Engine*. https://www.unrealengine.com/marketplace/en-US/product/modular-parking (elérés 2022. május 6.).
[43] *Sumo2Unreal*. Augmented Design Lab, 2022. Elérés: 2022. május 5. [Online]. Elérhető: https://github.com/AugmentedDesignLab/Sumo2Unreal
[44] „Import from OpenStreetMap - SUMO Documentation”. https://sumo.dlr.de/docs/Tutorials/Import_from_OpenStreetMap.html (elérés 2022. május 5.).
[45] „UChaosVehicleMovementComponent”. https://docs.unrealengine.com/4.26/en-US/API/Plugins/ChaosVehicles/UChaosVehicleMovementComponent/ (elérés 2022. május 11.).
[46] „mt19937 - C++ Reference”. https://www.cplusplus.com/reference/random/mt19937/ (elérés 2022. május 8.).
[47] „Using Avoidance With the Navigation System”. https://docs.unrealengine.com/4.27/en-

US/InteractiveExperiences/ArtificialIntelligence/NavigationSystem/Avoidance/ (elérés 2022. május 9.).
[48] NingxuanFeng, *PewLSTM*. 2021. Elérés: 2022. május 14. [Online]. Elérhető: https://github.com/NingxuanFeng/PewLSTM
[49] F. Zhang *és mtsai.*, „PewLSTM: Periodic LSTM with Weather-Aware Gating Mechanism for Parking Behavior Prediction”, júl. 2020, köt. 5, o. 4424–4430. doi: 10.24963/ijcai.2020/610.
[50] J. Kaniewski, *TensorFlow Unreal Plugin*. 2022. Elérés: 2022. május 15. [Online]. Elérhető: https://github.com/getnamo/TensorFlow-Unreal
[51] J. Kaniewski, *MachineLearningRemote Unreal Plugin*. 2022. Elérés: 2022. május 15. [Online]. Elérhető: https://github.com/getnamo/MachineLearningRemote-Unreal
[52] A. Kesting, M. Treiber, és D. Helbing, „General Lane-Changing Model MOBIL for Car-Following Models”, *Transp. Res. Rec.*, köt. 1999, o. 86–94, jan. 2007, doi: 10.3141/1999-10.
[53] T. Rajabioun és P. Ioannou, „On-Street and Off-Street Parking Availability Prediction Using Multivariate Spatiotemporal Models”, *IEEE Trans. Intell. Transp. Syst.*, köt. 16, o. 1–13, okt. 2015, doi: 10.1109/TITS.2015.2428705.

12. ÁBRAJEGYZÉK

13. TÁBLÁZATJEGYZÉK