**Szegedi Tudományegyetem**

**Informatikai Intézet**

Szakdolgozat

**Smit Alex**

**2023**

**Szegedi Tudományegyetem**

**Informatikai Intézet**

Önvezető autó tervezése és megvalósítása Webots környezetben

Szakdolgozat

*Készítette*: *Témavezető*:

**Smit Alex Dr. Schäffer László**

mérnökinformatikus egyetemi adjunktus

BSc szakos hallgató

Szeged

2023

# Feladatkiírás

A hallgató feladata egy saját tervezésű autó elkészítése és a Webots környezetben való felhasználása, úgy hogy az önvezető működésre legyen képes. Nagy sebesség mellett egy a hallgató által létrehozott szimulált környezetben különböző objektumok elkerülése a cél a hallgató által választott és az autóra elhelyezett szenzorok segítségével, miközben a jármű egy előre megtervezett útvonalon halad.

# Tartalmi összefoglaló

A téma megnevezése:

Önvezető autó tervezése és megvalósítása Webots környezetben.

A megadott feladat megfogalmazása:

Nulláról felépíteni egy Webots világot amelynek az útjain megy az általam létrehozott autó és megírni a kódot ami irányítja.

A megoldási mód:

Webots-ban elkészített világ és hozzá tartozó autó, valamint python-ban megírt kontroller és supervisor.

Alkalmazott eszközök, módszerek:

Webots R2023a, Python 3.10.6

Elért eredmények:

Az autó végigmegy a megadott útvonalon, miközben nem megy neki semminek és kikerüli az akadályokat.

Kulcsszavak:

Webots, mobil robot ágens, szimuláció, önvezetés

# **Tartalomjegyzék**

[Feladatkiírás 3](#_Feladatkiírás)

[Tartalmi összefoglaló 4](#_Tartalmi_összefoglaló)

[Tartalomjegyzék 5](#_Tartalomjegyzék)

[1. Bevezetés 6](#_Bevezetés)

[2. Felhasznált eszközök 7](#_Felhasznált_eszközök)

[2.1 A Webots program](#_A_webots_program) 7

[2.2 A Python nyelv 8](#_A_Python_nyelv)

[3. Az autó felépítése 9](#_Az_autó_felépítése)

[3.1 AckermannVehicle 9](#_AckermannVehicle)

[3.2 Az autó részei 10](#_Az_autó_részei)

[3.3 Objektumok érzékelése 13](#_Objektumok_érzékelése)

**[3.4 A környezet 14](#_A_környezet)**

[3.4.1 Utak 15](#_Utak)

[3.4.2 Épületek 17](#_Épületek)

[3.4.3 Fák 18](#_Fák)

[4. Úthálózat megvalósítása 19](#_Úthálózat_megvalósítása)

[4.1 Út adatok lekérése 19](#_Út_adatok_lekérése)

[4.2 Koordináták kiszámolása 21](#_Koordináták_kiszámolása)

[4.3 Adatok kiírása fájlba 22](#_Adatok_kiírása_fájlba)

[5. Önvezető funkció megvalósítása 24](#_Önvezető_funkció_megvalósítása)

[5.1 Adatok beolvasása 28](#_Toc133937771)

[5.2 Legrövidebb út keresés 25](#_Legrövidebb_út_keresés)

[5.3 Bearing megállapítása 26](#_Bearing_megállapítása)

[5.4 Vezérlés 27](#_Vezérlés)

[5.4.1 Út lekérése 27](#_Út_lekérése)

[5.4.2 Célpont elérése 28](#_Célpont_elérése)

[5.4.3 Bearing felhasználása 28](#_Bearing_felhasználása)

[5.4.4 Haladás az utakon 29](#_Haladás_az_utakon)

[5.4.5 Szenzorok felhasználása 30](#_Szenzorok_felhasználása)

[5.4.6 Akadálykerülés 31](#_Akadálykerülés)

[5.4.7 Elsőbbségadás 31](#_Elsőbbségadás)

[6. További fejlesztési lehetőségek 33](#_További_fejlesztési_lehetőségek)

[7. Összefoglaló 34](#_Összefoglaló)

[8. Irodalomjegyzék 35](#_Irodalomjegyzék)

[9. Nyilatkozat 36](#_Nyilatkozat)

[10. Köszönetnyilvánítás 37](#_Köszönetnyilvánítás)

# Bevezetés

A mechatronika nevű egyetemi kurzusomon találkoztam először a Webots szoftverrel és ott készítettem el benne az első robotomat is. Az még egy egyszerű téglatest volt négy kerékkel, ami egy négyzet alakú arénában dobozokat kerülgetett. Mivel manapság nagyon felkapott az önvezető autó kérdése, és engem is érdekel a téma, így arra gondoltam, hogy szakdolgozatnak megvalósítom egy leegyszerűsített változatát. Igyekeztem, ahol csak lehetett, törekedni arra, hogy valósághű legyen.

ezt egy oldalasra?

# Felhasznált eszközök

## A webots program

A Webots egy nyílt forráskódú és több platformos asztali alkalmazás, amelyet robotok szimulálására használnak. Teljes körű fejlesztési környezetet biztosít a robotok modellezéséhez, programozásához és szimulálásához. Szakmai használatra tervezték, és széles körben használják az iparban, az oktatásban és a kutatásban. A Cyberbotics Ltd. 1998 óta folyamatosan fejleszti a Webots-ot [[1]](#Unity), mint fő termékét.

ide random webots vilag kep

A Webots-ban egy világot a grafikus felület segítségével lehet felépíteni, ahol objektumok és komponensek adhatók hozzá és 3D-s környezetben rendezhetők el. A felhasználó a semmiből tud egy új világot létrehozni, vagy módosíthat egy meglévőt. A Webots világ objektumai különféle beépített komponensek vagy külső szoftverből importált egyedi 3D modellek felhasználásával jönnek létre. Ezek az objektumok lehetnek robotok, szenzorok, aktuátorok, lámpák, kamerák vagy bármilyen más entitás, amelyet szimulálni lehet a virtuális környezetben. Egy objektum hozzáadásához a felhasználó egyszerűen áthúzhatja azt a Webots könyvtárból a 3D nézetbe. Ha az objektum a világban van, akkor a tulajdonságai módosíthatók, mint például a mérete, helyzete, tájolása, vagy bármilyen más, a viselkedésével vagy megjelenésével kapcsolatos paraméter. A felhasználó összetett környezeteket is létrehozhat terepek, utak, épületek vagy bármilyen más olyan szerkezet hozzáadásával, amely segít meghatározni a virtuális világot. Ezeket az komponenseket az objektumokhoz hasonló módon lehet elhelyezni és módosítani, egér- és billentyűzetműveletek kombinációjával. Összességében a Webots világ felépítésének folyamata magában foglalja a különböző, már létező vagy egyedi komponensek 3D-s környezetben való kombinálását, és tulajdonságaik módosítását a kívánt szimuláció elérése érdekében.

A Webots-ban a node (csomópont) a scene tree (jelenetgráf) alapvető építőköve, amely a szimulált világ hierarchikus ábrázolása. A scene tree minden node-ja számos tulajdonsággal rendelkezhet, például névvel, pozícióval, tájolással, mérettel.

A Webots-ban különböző típusú node-ok léteznek, például:

* Transform node, más csomópontok csoportosítására és átalakítások felhasználására szolgálnak.
* Shape node, amelyek meghatározzák a szimulációban szereplő objektumok geometriáját.
* Sensor node, amelyeket érzékelők, például kamerák, lidarok (fényérzékelés és távolságmérés) és közelségérzékelők modellezésére használnak.
* Actuator node, amelyek aktuátorokat modelleznek, például motorokat.
* Controller node, amelyek a szimulációt vezérlő kódot reprezentálják.

A csomópontok a Webots API-n keresztül hozhatók létre, módosíthatók és elérhetők olyan programozási nyelvek használatával, mint a Python vagy a C++.

## A Python nyelv

A Python egy magas szintű, általános célú programozási nyelv is. A könnyen olvashatóságra törekszik, kifejező behúzások segítségével. Több programozási paradigmát támogat, beleértve a strukturált, objektumorientált és funkcionális programozást. Guido van Rossum az 1980-as évek végén kezdett el dolgozni a Python-on, az ABC programozási nyelv utódjaként, és először 1991-ben adta ki Python 0.9.0 néven. A Python 2.0 2000-ben jelent meg. A 2008-ban kiadott Python 3.0 jelentős újragondolás volt, amely nem teljesen kompatibilis a korábbi verziókkal. A 2020-ban kiadott Python 2.7.18 volt a Python 2 utolsó kiadása.

A fő filozófiája röviden összefoglalva:

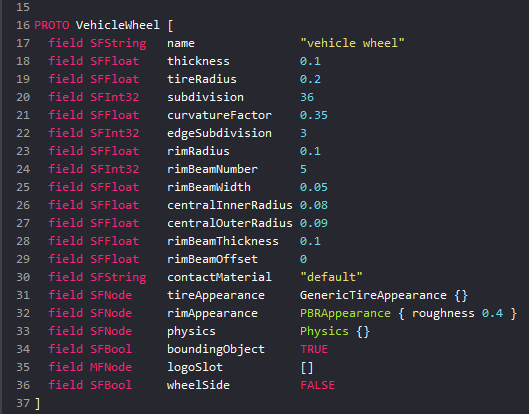
* A szép jobb, mint a csúnya.
* Az explicit jobb, mint az implicit.
* Az egyszerű jobb, mint a komplex.
* Számít az olvashatóság.

A Python folyamatosan az egyik legnépszerűbb programozási nyelv. [[2]](#Unity)

# Az autó felépítése

## AckermannVehicle

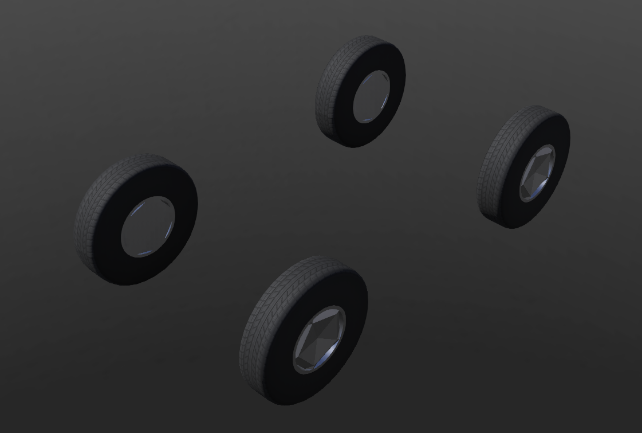
A megvalósítás során törekedtem arra, hogy minél valósághűbb legyen az autóm, ezért a beépített AckermannVehicle-ből (2. ábra) indultam ki. Az AckermannVehicle PROTO segítségével könnyedén létre lehet hozni bármilyen járművet, amely megfelel az Ackermann modellnek. Ez a modell azt az autó, vagy más jármű kormányzásánál fellépő geometriai problémát oldja meg, amikor egy kanyarban a belső és külső oldalon lévő kerekeknek különböző sugarú köröket kell leírniuk. A PROTO egy olyan webots mechanizmus, amellyel a felhasználók hozzáadhatnak saját nodeokat a már meglévőkhöz, így nagyon komplex objektumokat hozhatnak létre, és ezeket könnyen újból fel is tudják használni. Ebben a PROTO-ban kerekek pozíciója és tájolása automatikusan kiszámításra kerül a PROTO mezői alapján.



1. ábra: Kerekek értékei, PROTO fájl

Ezek a kerekek automatikusan kapcsolódnak a megfelelő csuklókhoz, hogy a megfelelő tengely mentén forogjanak. Az aktuátorok az első kerekekhez vannak csatlakoztatva, hogy lehessen kormányozni a járművet. Az AckermannVehicle PROTO egy Robot típusú node, ami azt jelenti, hogy tudom kontrollerrel irányítani. Az extensionSlot nevű mezőjében pedig hozzá tudok adni minden szükséges plusz dolgot (az autó alakja, érzékelők, egyéb aktuátorok). Felhasználtam még a beépített Car PROTO-t is, amely az AckermannVehicle-ből származódik és továbbfejleszti azt. A Driver könyvtárral együtt használva hozzáad az autóhoz egy motor modellt, sebességváltót, lámpákat és fék modellt.

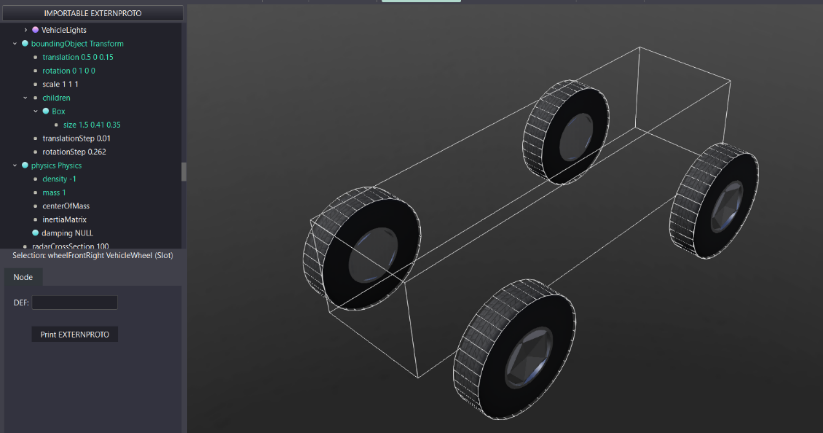
Először elsőkerék-meghajtást állítottam be az autónak, mivel elöl van a kormány, ez tűnt a valósághoz legközelebb állónak. Amikor így elindítottam a szimulációt, az első kerekek felpörögtek, a hátsókat csak húzták maguk után, összevissza csúszkált és kanyargott az autó. Kipróbáltam többféle talajt, hátha azzal van a baj, de egyiken sem viselkedett úgy, ahogy kellett volna. Amikor átváltottam hátsókerék-meghajtásra ugyanezzt tapasztaltam, csak a hátsók pörögtek fel, és az elsőket húzták magukkal. Ezek miatt használtam végül 4x4 meghajtást, amivel már jól megy az autó, nem csúszkál.



2. ábra: Ackermann Vehicle alap

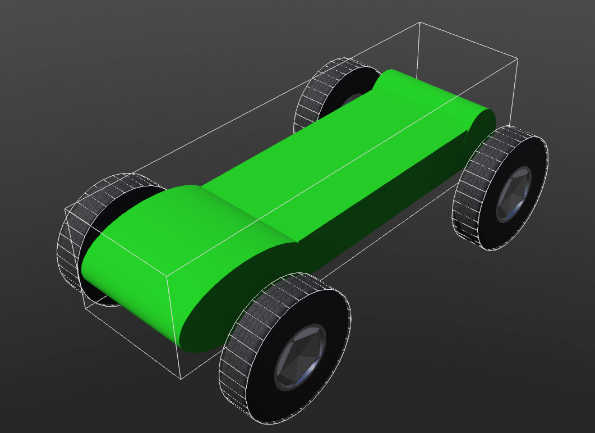
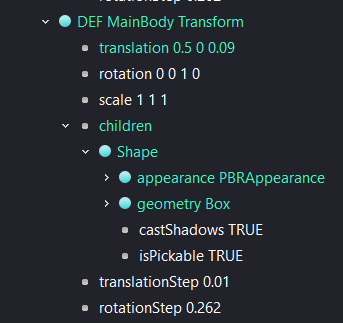
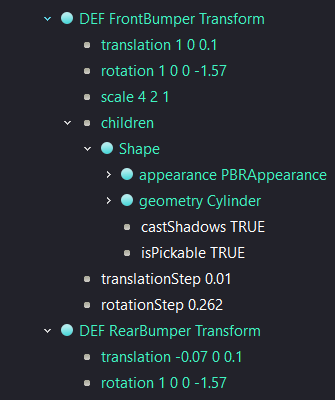
## Az autó részei

A jármű fizikáját egy box shape képviseli, a váz és a többi alkatrész pedig mind egy-egy Transform node. Ennek a nodenak nincs fizikája, így nem ütik egymást az alkatrészek a lépések számításakor. Az autó 9 téglatestből, 2 hengerből és 4 kúpból áll. Az xy képen látható a jármű fizikájának beállítása. A boundingboxObject mezőjéhez hozzáadtam egy Transfrom node-ot (3. ábra), amely téglatest alakú, méretei pedig 1.5 x 0.41 x 0.35. A Webots-ban a boundingboxObject mező egy node azon attribútuma, amely egy 3D-s határolókeretet (bounding box) határoz meg a node geometriája körül. Ez a bounding box az ütközésészlelés optimalizálására szolgál. A physics mezőjéhez hozzáadtam a Webots-ba beépített alap fizikát, és átállítottam, hogy a járműnek ne legyen sűrűsége, tömege pedig 1 (ezek a szimuláció megfelelő működéséhez szükségesek). A Webots node physics mezője a szimulációban szereplő objektum fizikai tulajdonságainak meghatározására szolgál. Olyan paramétereket tartalmaz, mint a tömeg, és a súrlódás, amelyeket a fizikai motor az objektum viselkedésének kiszámításához használ, amikor az a szimulációban más objektumokkal kölcsönhatásba lép.

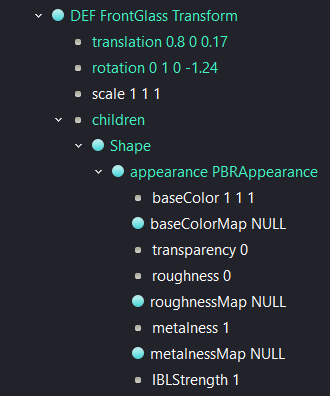


3. ábra: Bounding box

Az autó alapját egy téglatest Transform node képviseli. Az első és hátsó lökhárító egy-egy henger, aminek megváltoztattam a méreteit, elforgattam és átskáláztam őket. Ezek egyedül azt a célt szolgálják, hogy jobban hasonlítson egy tényleges autóra a robotom. Minden elem kinézetének beállításához a PBRAppearance-t használtam. A PBRAppearance a Webots funkciója, amely lehetővé teszi az objektumok fizikai alapú renderelését a szimulációban. A valós anyagok optikai tulajdonságainak szimulálására használják, például a fény visszaverődését, elnyelését és áteresztését a különböző anyagokon. A PBRAppearance lehetővé teszi a felhasználók számára, hogy a szimuláció során különböző textúrákat, színeket és anyagtulajdonságokat rendeljenek a tárgyakhoz, és valósághű fényhatások létrehozására is használható. Én ezzel állítottam be az autó részeinek színét és fényvisszaverő képességét.

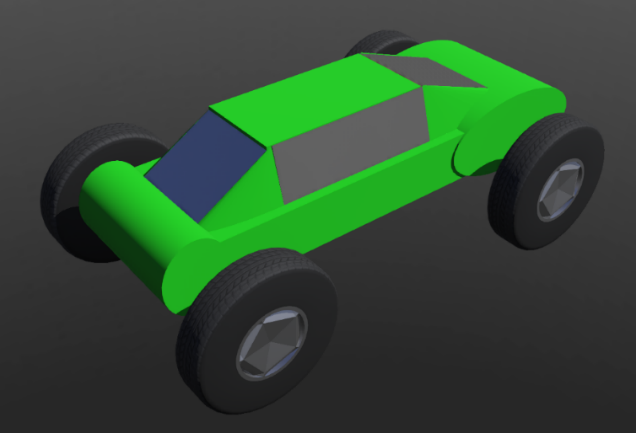


4. ábra: Az autó testének alapja és az első lökhárító értékei

Az utastér részt még egy téglatesttel oldottam meg, ami rajta van az autó testének alapján. Az ablakok szintén téglatestek, amik el vannak forgatva, fekete színűek, és a metalness (fémesség) tulajdonságuk 1-re van állítva, így sötétített üveg hatását keltik, és visszatükrözik a fényt.

5. ábra: Ablakok és felső rész, valamint az elülső ablak értékei

Az ablakok közé odaraktam egy-egy kúpot átméretezve és átskálázva, ez képviseli a köztük lévő fémet, ami magukat az ablakokat hivatott tartani.



6. ábra: Sarkok

Végül pedig raktam rá spoilert (áramlásrontó szárnyféklap). Ez 3 téglatestből áll. 2 ami tartja, és a harmadik maga a spoiler.

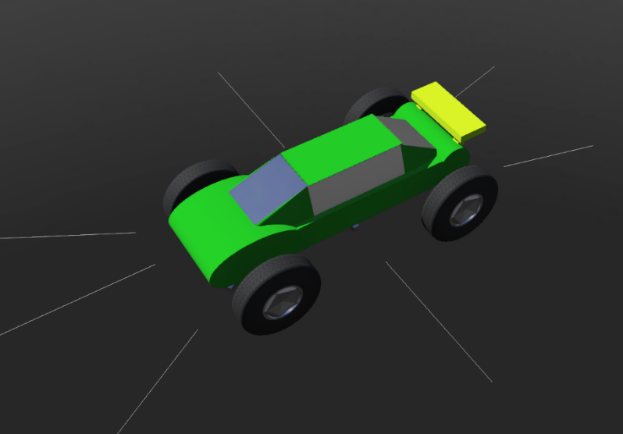


7. ábra: Spoiler

## Objektumok érzékelése

Hét darab lézeres távolságszenzort használtam, hatot az akadályok és falak kerülésére, egyet pedig, hogy ne menjen neki az úton haladó többi autónak (8. ábra). A lézeres távolságérzékelő a valóságban úgy működik, hogy lézersugarat bocsát ki, és méri azt az időt, amely alatt a sugár visszaverődik egy tárgyról, és visszatér az szenzorhoz. Az idő mérésével és a fénysebesség ismeretében kiszámítható a tárgy távolsága. A Webots-ban lézeres távolságérzékelőt úgy lehet megvalósítani, hogy hozzáadom azt egy robothoz vagy a környezethez. A Webots szoftverben a lézeres távolságérzékelő egy Sensor node, aminek be lehet állítani a tulajdonságait, például a tartományt, a felbontást és a mintavételi időközt. Ezután a vezérlő kódjában a megfelelő funkció segítségével leolvashatók a lézer távolságérzékelő értékei. A visszaadott érték megadja az érzékelő által észlelt legközelebbi tárgy távolságát.

Minden szenzornak beállítottam a LookUpTable-jét, amely megmondja, hogy milyen távolságról érzékeljen, és hogy ha x távolságra érzékel objektumot, arra mekkora értéket adjon vissza. Minden szenzornak be kellett állítanom a lookupTable-jét, ami tulajdonképpen megmondja, hogy a szenzor mekkora távolságra képes mérni. Tehát az alábbi táblázatban (1. táblázat) látható értékek azt jelentik, hogy az előre néző távolságérzékelő 2.75 egységnyi távolságról kezd el érzékelni, de ekkor még nullás értéket ad vissza, ez a minimum érték, és ha 0.25 egységnyi távolságról érzékel valamit, akkor 1-es értéket ad vissza, ami a maximum, a kettő között pedig a távolsággal arányos értéket, a 0 - 1 közötti intervallumban.



8. ábra: Távolságszenzorok

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Távolságszenzorok | Bal hátsó szenzor () | Bal oldali szenzor () | Bal első szenzor () | Első szenzor () | Jobb első szenzor () | Jobb oldali szenzor () | Jobb hátsó szenzor () |
| Minimum távolság | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 | 0.25 |
| Maximum távolság |  |  |  | 2.75 |  |  |  |

1. táblázat: Távolságszenzorok értékei

## A környezet

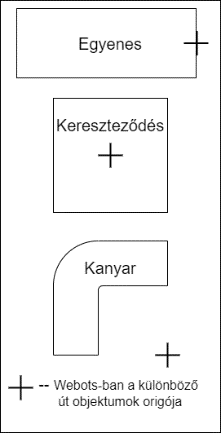
Az autón kívül a környezetet is nekem kellett létrehoznom. Ehhez a Webots-ba benne lévő már létező elemeket használtam fel. Raktam talajt, amihez a Floor-t használtam. Megterveztem az úthálózatot amin majd a járművek közlekedni tudnak, úgy, hogy több önmagába visszatérő út, zsákutcák és kanyarok is legyenek benne. A kereszteződésekhez raktam gyalogátkelőhelyeket a szebb kinézet miatt, és hogy ne legyen olyan üres az egész, raktam bele fákat és épületeket.



9. ábra: A Webots világom

### Utak

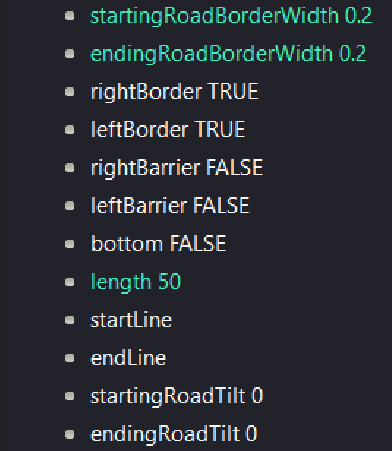
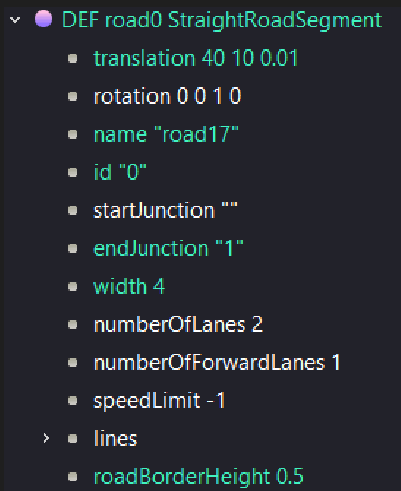
Háromféle út elemet használtam fel: egyenes, kanyar és kereszteződés. Mindegyiket másképp kellett lerakni és kezelni, amit alább tárgyalok.



10. ábra: Webots-os utak origói

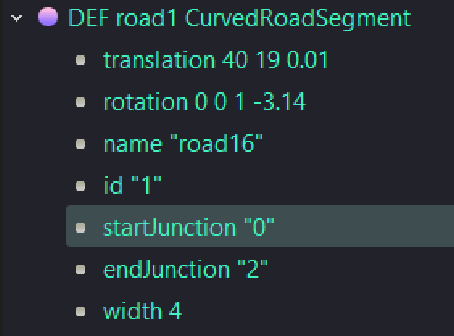
Az egyenesnek a saját koordinátái az egyik végén vannak (10. ábra), és valamelyiket el is kellett forgatni. Meg kellett adnom az út szélességét, hosszát, a hozzá tartozó fal szélességét és magasságát, és hogy hány sávos legyen. Ezen kívül meg kellett adnom az út saját ID-ját, valamint a kezdő és végpontjához (startJunction és endJunction) csatlakozó másik két út ID-ját. Ezek majd később az útvonalkeresésnél lesznek fontosak.

Az alábbi ábrán a 0 ID-s út értékei láthatók, ami egy egyenes. Láthatók a translation (eltolás) értékei, ez világban az objektum helyzete x,y,z koordinátákkal kifejezve. A rotation (elfordulás) megadja, hogy az elephelyzetéhez képest el lett-e fordítva. Ez is x,y,z koordinétékkal van megadva, a negyedik pedig az elfordulás mértéke, ez az elem nincs elforgatva. Alatta az elem neve, majd az ID-ja és a hozzá csatlakozó utak ID-jai. A width (szélesség), amely minden útnak ugyanakkora kell, hogy legyen, hogy végig tudjon rajtuk menni az autó. A roadBorder mezők az utak menti falak magassága és szélessége az út elejénél és végénél. Ezek azért szükségesek, hogy a lézerszenzorai segítségével tudjon végigmenni az autó. Length (hosszúság) mezője csak egyenes utaknak van, ennek éppen 50 az értéke, de vannak amelyek eltérő hosszúságúak.



11. ábra: Egy egyenes út értékei

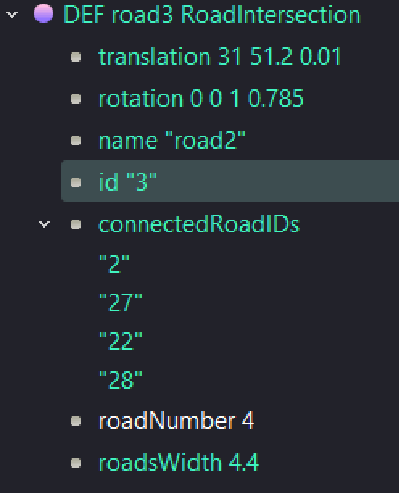
A kanyarnak a saját koordinátái az kör középpontjában vannak, amit az íve leír (10. ábra), és valamelyiket el is kellett forgatni. Meg kellett adnom az út szélességét, görbületi sugarát, teljes szögét, a hozzá tartozó fal szélességét és magasságát, és hogy hány sávos legyen. Ezen kívül meg kellett adnom az út saját ID-ját, valamint a kezdő és végpontjához csatlakozó másik két út ID-ját.

Az alábbi ábrán az 1 ID-s út értékei láthatók, ami egy kanyar. Az első néhány mező azonos az egyenes elemnél láthatókkal (eltolás, elfordulás, név, ID-k, width). Kanyarnál be kell még állítani a curvatureRadius-t (görbületi sugár), ami tulajdonképpen a kanyar hossza, és a totalAngle-t (teljes szög), ami mindegyik kanyarnak ugyanannyi. A subdivision (felosztás) az az érték, ahány egyenessel helyettesítve van a kanyarom a világban a szimuláció során, mivel valójában az sem egy görbe, hanem sok egyenes van helyette.

12. ábra: Egy kanyar értékei

A kereszteződésnek a saját koordinátái egyben a gráf pontom koordinátái is, mivel az elem közepét jelölik (10. ábra). Minden kereszteződés el van forgatva, mert alapból keresztben álltak. Meg kellett adnom a hozzá csatlakozó utak szélességét, és azt, hogy hány van belőlük. Ezen kívül meg kellett adnom az út saját ID-ját, valamint a hozzá csatlakozó összes út ID-ját.

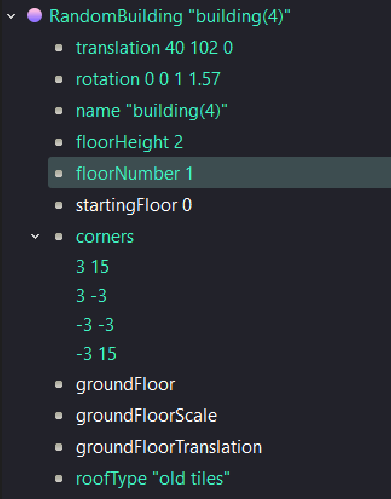
Az alábbi ábrán a 3 ID-s út értékei láthatók, ami egy kereszteződés. Az ilyen típusú elemeknek nincs fala, nem kell megadni hosszúságot sem saját szélességet, csak a hozzájuk csatlakozó utak szélességét. Eltolásuk és elfordulásuk ugyanúgy van, mint a többi útnak. Az elfordulása minden kereszteződésnek ugyanannyi, mert mindegyik ugyanúgy áll, és 4 út csatlakozik hozzájuk. Ennek a 4 útnak az ID-jait kell megadni egy listában.



13. ábra: Egy kereszteződés értékei

### Épületek

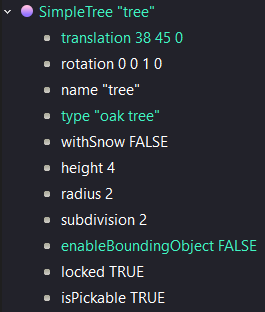
A legtöbb épület a Webots-ba beépített RandomBuilding elem, de van egy motel és egy FastFoodRestaurant, valamint egy hirdetőtábla is. Az egyszerű épületeknek, mint ahogy az az alábbi ábrán is látható, be lehet állítani a pozícióját, hogy hány szintes legyen és azok milyen magasak legyenek, valamint be lehet állítani a négy sarkának eltolását a középpontjától, ezzel áálítva az alapjának méretét. Ezen kívül be lehet még állítani a tető kinézetét, egy listából választva.



14. ábra: Egy egyszerű épület értékei

### Fák

A fák a Webots-ba beépített SimpleTree-k. Ezeknek ki lehet választani a fajtáját egy listából, be lehet állítani a pozíciójukat és a méretüket. Az én világomban van tölgy, hackberry (ez más lesz) és mogyoró.



15. ábra: Egy egyszerű fa értékei

# Úthálózat megvalósítása

Az útelemekkel való számolásokat nem a kontrolleremben végeztem el, hanem egy Supervisor-ben (felügyelő). Ha a robotomnak a supervisor mezője igazra van állítva, továbbra is rendesen robotként működik, de emellett használhatók vele a supervisor függvények, melyekkel lehet vezérelni a szimulációt, vagy módosítani a világ elemeit. Ez helyettesíti általában az emberi tevékenységeket, mint a megtett távolság mérése vagy a robot visszahelyezése kiindulási helyzetbe és még nagyon sok minden mást. A lényege az, hogy általában olyan funkciókat lát el, amelyek nem találhatók meg valódi robotokon, inkább egy emberi beavatkozásnak felelnek meg és a valóságban nehéz is átültetni őket a robotra.

## Út adatok lekérése

A supervisor-t azzal kezdem, hogy meghívom a megfelelő könyvtárakat, amelyekre majd szükségem lesz, ilyen maga a Supervisor, a pickle és a math. Ezután létrehozom a robot objektumomat, ami ebben az esetben egy supervisor lesz. Most jöhetnek az utak. Először létrehozom őket, a Webots világban lévő nevük alapján, majd létrehozok néhány listát, amikben majd az információikat tárolom, amikre később szükségem lesz.

|  |
| --- |
| #az utak nevei a webots világból  road\_defs **=** **[**'road0'**,**'road1'**,**'road2'**,**'road3'**,**'road4'**,**'road5'**,**'road6'**,**'road7'**,**'road8'**,**'road9'**,**'road10'**,**'road11'**,**'road12'**,**'road13'**,**'road14'**,**'road15'**,**'road16'**,**'road17'**,**'road18'**,**'road19'**,**'road20'**,**'road21'**,**'road22'**,**'road23'**,**'road24'**,**'road25'**,**'road26'**,**'road27'**,**'road28'**,**'road29'**,**'road30'**,**'road31'**,**'road32'**]**  #a listák amikbe belerakom az utak különböző információit  roads **=** **[]**  road\_ids **=** **[]**  road\_rotations **=** **[]**  road\_coords **=** **[]**  road\_width **=** **[]**  road\_length **=** **[]**  road\_angle **=** **[]**  road\_points **=** **{}** |

16. ábra: Az utak listái

Kezdetnek lekérem magukat az út objektumokat. Ezt úgy teszem meg, hogy egy ciklussal végigiterálok az utak neveinek listáján, és lekérem a hozzájuk tartozó node-okat. A továbbiakban ezeket használva lekérem az ID-jaikat, a helyzetüket, az elfordulásukat, a kanyarok görbületeit, az utak szélességét és hosszát. Az összes ilyen információt le tudom kérni egy-egy ciklusban, az út objektum lista i-edik elemének megfelelő mezőjének nevével és típusával (17. ábra).

|  |
| --- |
| #az utak információinak lekérdezése a webots világból és elmentésük  #maga az út objektum  **for** i **in** **range(len(**road\_defs**)):**  roads**.**append**(**robot**.**getFromDef**(**road\_defs**[**i**]))**  #az utak id-jai  **for** i **in** **range(len(**roads**)):**  road\_ids**.**append**(**roads**[**i**].**getField**(**'id'**).**getSFString**())**  #az utak elhelyezkedése a világban  **for** i **in** **range(len(**roads**)):**  road\_coords**.**append**(**roads**[**i**].**getField**(**'translation'**).**getSFVec3f**())**  #az utak forgási szögei  **for** i **in** **range(len(**roads**)):**  road\_rotations**.**append**(**roads**[**i**].**getField**(**'rotation'**).**getSFRotation**())** |

17. ábra: Egyszerű út információk lekérése

A probléma akkor van, ha egyik útnak nincs ilyen tulajdonsága, pl. görbülete csak a kanyaroknak van. Ezeket egy try-except blokkal oldottam meg. Ha van ilyen értéke az útnak, beírja a listába, ha nincs, akkor nullát ír bele (18. ábra).

|  |
| --- |
| #a kanyarok görbületei  **for** i **in** **range(len(**roads**)):**  **try:**  road\_angle**.**append**(**roads**[**i**].**getField**(**'curvatureRadius'**).**getSFFloat**())**  #ha nincs ilyen értéke, akkor nullát ír be  **except** **AttributeError:**  road\_angle**.**append**(**0**)**  #az utak szélessége  **for** i **in** **range(len(**roads**)):**  **try:**  #másképp kell lekérni kereszteződésnél  road\_width**.**append**(**roads**[**i**].**getField**(**'roadsWidth'**).**getSFFloat**())**  **except** **AttributeError:**  #és másképp kanyarnál és egyenesnél  road\_width**.**append**(**roads**[**i**].**getField**(**'width'**).**getSFFloat**())** |

18. ábra: Komplexebb út információk lekérése

Ezek után lekérem az utak csatlakozási információit. Ehhez kellett korábban a világban megadni a kezdő és végpont utakat. Létrehozok egy könyvtárat, amelybe majd ezek az információk belekerülnek. Ezek lekérése egyenes és kanyar esetén egyszerű, lekérem az utak a startJunction és endJunction tulajdonságait, majd ezeket a könyvtár i-edik helyére belerakom. Kereszteződésnél azonban nincsenek ilyen mezők. Ott egy darab connectedRoadIDs van, ami egy lista. Itt is try-except-et használtam, ha nincs kezdő és vég ID, akkor az egész csatlakozási ID listát kérem le, és ezt rakom bele a könyvtár i-edik helyére. Ezzel megvan az összes útnak, hogy melyik másik utakhoz csatlakoznak, ez maga a gráfom, amit a későbbiekben használni fogok.

|  |
| --- |
| #megint végigmegyek az utakon egy ciklussal  **for** i **in** **range(len(**roads**)):**  #létrehozok a könyvtárban egy üres helyet  graph**[**i**]** **=** **[]**  #kiürítem a csatlakozási pontok listáját  connect**.**clear**()**  #egyeneseknek és kanyaroknak ki lehet olvasni a kezdő és vég csatlakozási útjaikat  **try:**  start **=** roads**[**i**].**getField**(**'startJunction'**).**getSFString**()**  end **=** roads**[**i**].**getField**(**'endJunction'**).**getSFString**()**  #kereszteződéseknek ezek egy listában vannak, ami tartalmazza az összes  #hozzá csatlakozó út id-ját  **except** **AttributeError:**  **for** j **in** **range(**4**):**  **if** roads**[**i**].**getField**(**'connectedRoadIDs'**).**getMFString**(**j**)** **==** ''**:**  **break**  #ezeket belerakom a csatlakozási pontos listámba  connect**.**append**(**roads**[**i**].**getField**(**'connectedRoadIDs'**).**getMFString**(**j**))**  #ha volt kezdő pontom, belerakom a könyvtárban létrehozott helyére  **if** start**:**  graph**[**i**].**append**(**start**)**  #ha volt vég pontom, belerakom a könyvtárban létrehozott helyére  **if** end**:**  graph**[**i**].**append**(**end**)**  #ha volt csatlakozási pontom, belerakom a könyvtárban létrehozott helyére  **if** connect**:**  **for** l **in** **range(len(**connect**)):**  graph**[**i**].**append**(**connect**[**l**])**  #kiürítem a listáimat  start **=** **[]**  end **=** **[]** |

19. ábra: Az utak csatlakozásainak lekérése

## Koordináták kiszámolása

Mostmár kiszámolhatom a célpont koordinátákat. Ez egy újabb ciklus amellyel az út objektumokon végigmegyek. Létrehozok egy listát a pontoknak.

A kereszteződésekkel kezdtem, mert azok a legegyszerűbbek. Ha az út elfordulása π/4, akkor tudom, hogy ez egy kereszteződés, ekkor a pontom koordinátái, ahová az autónak majd menni kell, megegyeznek az utam koordinátáival.

Egyeneseknél két lehetőség áll fent, ha el lett forgatva, és ha nem. Előbbinél tehát az elfordulás π/2 lesz, a hossza pedig ugye nem 0, ekkor az x koordinátája a sajátja lesz, y-t pedig el kell tolnom az út hosszának felével. Így a pontom az út közepe lesz. Utóbbinál elfordulása 0, hossza pedig nem, a koordináták pedig fordítva, x-et kell eltolnom az út hosszának felével, y pedig a saját koordinátája lesz.

Kanyaroknál volt a legkomplikáltabb, ott ugye négyflkeképpen állhatnak (egy kör négy negyede). Először is, hogy ez egy kanyar, azt abból tudom, hogy van forgási szöge.

Miután ez megvan, akkor nézem meg az elfordulását, és attól függően, hogy merre van fordítva a világban, eltolom a pont koordinátáit a megfelelő irányba, a kanyar sugarának hosszával. Ezzel megvannak a gráfpontjaim koordinátái, ahová az autónak majd menni kell.

|  |
| --- |
| #megint végigmegyek az utakon egy ciklussal  **for** i **in** **range(len(**roads**)):**  #létrehozok a könyvtárban egy üres helyet  road\_points**[**i**]** **=** **[]**  #ha az út egy kereszteződés  **if** road\_rotations**[**i**][**0**]** **==** **round(**math**.**pi**/**4**,**4**):**  #a saját helyzete a világban, lesz a pont ahová majd az autónak mennie kell  road\_points**[**i**]** **=** road\_coords**[**i**]**  #ha egy elforgatott egyenes út  **if** road\_length**[**i**]** **!=** 0 **and** **round(**road\_rotations**[**i**][**0**],**4**)** **==** **round(**math**.**pi**/**2**,**4**):**  #akkor az X koordinátája a saját X koordinátája lesz  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**])**  #az Y pedig a saját Y-jától a hossza felével eltolt pont  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**]** **+** road\_length**[**i**]** **/** 2**)**  #ha egy nem elforgatott egyenes út  **if** road\_length**[**i**]** **!=** 0 **and** road\_rotations**[**i**][**0**]** **==** 0.0**:**  #az X koordinátája saját X-étől a hossza felével eltolt pont  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**]** **+** road\_length**[**i**]** **/** 2**)**  #az Y koordinátája pedig a saját Y koordinátája lesz  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**])**  #ha egy kanyar  **if** road\_angle**[**i**]** **!=** 0**:**  #lekérdezem a forgási szögét  rot **=** **float(**road\_rotations**[**i**][**0**])**  #és attól függően, hogy hogyan áll a világban, eltolom  #a koordinátájához képest a pontot, ahová majd az autónak menni kell  **if** **round(**rot**,**4**)** **==** **round(**math**.**pi**/**2**,**4**):**  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**]** **-** road\_angle**[**i**]\*(**math**.**pi**/**4**))**  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**]** **+** road\_angle**[**i**]/(**math**.**pi**/**2**))**  **if** **round(**rot**,**4**)** **==** 0**:**  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**]** **+** road\_angle**[**i**]/(**math**.**pi**/**2**))**  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**]** **+** road\_angle**[**i**]/(**math**.**pi**/**2**))**  **if** **round(**rot**,**4**)** **==** **round(**math**.**pi**/** **-**2**,**4**):**  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**]** **+** road\_angle**[**i**]/(**math**.**pi**/**2**))**  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**]** **-** road\_angle**[**i**]\*(**math**.**pi**/**4**))**  **if** math**.**fabs**(round(**rot**,**4**))** **==** **round(**math**.**pi**,**4**):**  road\_points**[**i**].**append**(**road\_coords**[**i**][**0**]** **-** road\_angle**[**i**]\*(**math**.**pi**/**4**))**  road\_points**[**i**].**append**(**road\_coords**[**i**][**1**]** **-** road\_angle**[**i**]\*(**math**.**pi**/**4**))** |

20. ábra: Az utak koordinátáinak kiszámolása

## Adatok kiírása fájlba

Amikor kiszámolom az utak koordinátáit, a python pickle metódusa segítségével kiírom őket egy fájlba, majd ezt a fájlt olvasom be a kontrolleremmel, így már egy kész dictionary (python tároló, amely kulcs:érték párokból tevődik össze, az én esetemben ezek „út id : x és y koordináta”) áll rendelkezésemre. Kiírtam még az utak hosszát tartalmazó listát, a fordulási szögeket és a saját koordinátáikat, mert ezeket fel fogom használni a kontrollerben.

|  |
| --- |
| #a szükséges listákat kimentem egy-egy fájlba, amit majd a kontrollerben beolvasok  **with** **open(**r'..\myGraph.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**graph**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)**  **with** **open(**r'..\roadIDs.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**road\_ids**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)**  **with** **open(**r'..\graphCoords.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**road\_points**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)**  **with** **open(**r'..\roadCoords.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**road\_coords**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)**  **with** **open(**r'..\roadAngles.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**road\_angle**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)**  **with** **open(**r'..\roadLengths.pickle'**,** 'wb'**)** **as** handle**:**  pickle**.**dump**(**road\_length**,** handle**,** protocol**=**pickle**.**HIGHEST\_PROTOCOL**)** |

21. ábra: A supervisor-ból az adatok kiírása fájlba

# Önvezető funkció megvalósítása

A Webots-ban a kontroller (vezérlő) egy olyan program, amely a szimulációval párhuzamosan fut, és interakcióba lép a szimulált robottal vagy objektumokkal a környezetben. A kontroller feladata, hogy parancsokat küldjön a robotnak vagy objektumoknak, és fogadja majd feldolgozza az érzékelők adatait. Hozzáférhet a szimulált robot vagy objektumok szenzoraihoz és aktuátoraihoz, és felhasználhatja azokat a kívánt feladatok elvégzésére. Alapvető összetevője a robotok vagy objektumok viselkedésének szimulációban történő létrehozásának és tesztelésének, mielőtt azokat fizikai rendszerre telepítenék.

## Adatok beolvasása

A kód elején itt is meghívom a megfelelő könyvtárakat, amiket majd használni fogok (Driver, math, Compass, GPS, pickle). Ezután beolvasom a supervisor-rel kiírt fájlokat.

|  |
| --- |
| #a supervisor által létrehozott pickle fájlok beolvasása  **with** **open(**r'..\myGraph.pickle'**,** 'rb'**)** **as** handle**:**  graph **=** pickle**.**load**(**handle**)**  **with** **open(**r'..\roadIDs.pickle'**,** 'rb'**)** **as** handle**:**  road\_ids **=** pickle**.**load**(**handle**)**  **with** **open(**r'..\graphCoords.pickle'**,** 'rb'**)** **as** handle**:**  graph\_coords **=** pickle**.**load**(**handle**)**  **with** **open(**r'..\roadCoords.pickle'**,** 'rb'**)** **as** handle**:**  road\_coords **=** pickle**.**load**(**handle**)**  **with** **open(**r'..\roadAngles.pickle'**,** 'rb'**)** **as** handle**:**  road\_angle **=** pickle**.**load**(**handle**)**  **with** **open(**r'..\roadLengths.pickle'**,** 'rb'**)** **as** handle**:**  road\_lengths **=** pickle**.**load**(**handle**)** |

22. ábra: A supervisor-rel kiírt fájlok beolvasása

Létrehozom a Driver instance-emet, amely maga a robot amit irányítani fogok, és megadom neki a timestep-et (lépésközt), ez az az idő millisecundumban, ami a két lefutás között el fog telni.

|  |
| --- |
| #az autót irányító driver létrehozása  driver **=** Driver**()**  #a lépésköz létrehozása amivel a szimuláció menni fog  timestep **=** **int(**driver**.**getBasicTimeStep**())** |

23. ábra: A Driver instance és a timestep

## Legrövidebb út keresés

A supervisor-ben létrehozott dictionary-t felhasználva határozom meg mely útvonalon kell az autómnak mennie, egy BFS (szélességi keresés) algoritmus segítségével, amelynek átadom a gráfomat (a dictionary-m), a kiindulási valamint a végpontomat, az pedig visszaadja a legrövidebb utak közül azt, amelyiket először megtalálta, amin végigmenve el tud jutni az autóm a kezdőpontból a végpontba. Először létrehoz egy üres sort (tároló), majd egy dictionary-t a meglátogatott csúcsoknak, amelybe belerakja a kezdőpontot egy None kulccsal, ez jelenti azt, hogy annak a pontnak nincs szülője. A kiindulási pontot belerakja a sorba, majd ki is veszi onnan, mert jelenleg azt fogja vizsgálni. Ha a vizsgált pont megegyezik a végponttal, létrehoz egy üres listát, amelybe addig pakolja bele visszafelé a csúcsokat a szülőjükön keresztül, amíg el nem ér a None-hoz, azaz ahhoz az elemhezz, aminek már nincs szülője. A jelnlegi csúcsot belerakja a meglátogatottakhoz. A kész út listát megfordítja, hogy megfelelően (kezdő -> vég) álljon és visszaadja a listát. Ha még nem találta meg a végpontot, akkor az eddig vizsgált pont szomszédja felé megy tovább, amennyiben az még nem lett meglátogatva, és beállítja az eddig vizsgáltat, mint szülőt, valamint belerakja azt a meglátogatottak közé. Ha nem talál megfelelő útvonalat, None-t ad vissza. A Python-ban használt None kulcsszó a null érték vagy a nincs értéke megfelelője. None nem ugyanaz, mint a 0, a False vagy egy üres karakterlánc. A None egy egyedi adattípus (NoneType), és csak a None lehet None [[3]](#Unity).

|  |
| --- |
| #a legrövidebb utat megkereső függvényem  #átadom neki a gráfomat, majd a két pontot ami között keresnie kell az utat  **def** pathfinder**(**graph**,** start**,** end**):**  #létrehoz egy sort amiben eltárolja a csúcsokat amiket meg kell látogatni  queue **=** deque**()**  #létrehoz egy könyvtárat amiben a meglátogatott csúcsokat és a szülőjüket tárolja  #az első természetesen a kiinduló pont, aminek nincs szülője  visited **=** **{**start**:** **None}**  #ezt belerakja a sorba  queue**.**append**(**start**)**  #elindít egy while-t, ami addig fut, míg a sor üres nem lesz  **while** queue**:**  #kiveszi az első elemet a sorból, ez lesz a jelenlegi csúcs  current\_node **=** queue**.**popleft**()**  #megnézi, hogy a kivett csúcs megegyezik-e a végponttal  **if** current\_node **==** end**:**  #létrehozza az útvonal listát, amit majd a végén visszaad  path **=** **[]**  #addig megy, míg vissza nem ér az eredeti kiinduló csúcshoz  **while** current\_node **is** **not** **None:**  #hozzáadja azt az útvonalhoz  path**.**append**(**current\_node**)**  #megjelöli már meglátogatottként  current\_node **=** visited**[**current\_node**]**  #megfordítja az útvonalat, hogy a megfelelő sorrendben legyenek (kezdő->vég)  path**.**reverse**()**  #és visszaadja  **return** path  #ha nem egyezik meg a végponttal, megvizsgálja a szomszédos csúcsokat a gráf alapján  **for** neighbor **in** graph**[**current\_node**]:**  #hogyha ezek még nem lettek korábban meglátogatva  **if** neighbor **not** **in** visited**:**  #akkor arra megy tovább, a szülőjét beállítva az eddig nézett csúcsra  visited**[**neighbor**]** **=** current\_node  #és belerakja a sorba  queue**.**append**(**neighbor**)**  #ha elér idáig, az azt jelenti, hogy nincs út a két csúcs között  **return** **None** |

24. ábra: A BFS algoritmus

## Bearing megállapítása

A bearing (irány) megállapításához az autóra rakot iránytűt használom. Először lekérem egy változóba ennek az értékét, ami visszaad egy 3D vektort, amely a Webots globális koordinátarendszerében az autóm irányát jelenti. Ezután kiszámolom az előző változómnak az x tengely pozitív irányával bezárt szögét radiánban, majd kiszámolom a szöget az autóm és a jelenlegi célpontom pozíciója között is. Ezt a kettőt kivonva egymásból megkapom az célpontom irányát radiánban, ezt átszámolom fokokba. Már csak el kell tolni, hogy egyenesen legyen a 0°, jobbra a 90° és így tovább (26. ábra), majd a megkapott értéket visszaadja.

|  |
| --- |
| #a célpont irányát visszaadó függvényem  **def** getTargetBearing**():**  #először lekérem az autóba épített iránytűm értékeit  #ez visszaad egy 3D vektort, amely a webots-os globális  #koordinátarendszerben az autóm irányát jelképezi  north **=** compass**.**getValues**()**  #kiszámolja radiánban a szöget a változóm és az x tengely pozitív iránya között  rad **=** math**.**atan2**(**north**[**1**],**north**[**0**])**  #kiszámolja a szöget a célpont és az autó jelenlegi pozíciója között  target\_bearing **=** math**.**atan2**(-**target\_y **+** current\_y**,** target\_x **-** current\_x**)**  #kiszámolja a szöget a célpont és az iránytű között  target\_bearing **-=** rad  #ezt az értéket átszámolja fokokba  target\_bearing **\*=** 180**/**math**.**pi  #majd átalakítja, hogy előre legyen a 0 fok, jobbra a 90,  #hátul a 180, balra a 270, majd ismét előre a 360  **if** target\_bearing **<** **-**90.0**:**  target\_bearing **=** target\_bearing **+** 360  target\_bearing **+=** 90  **if** target\_bearing **>=** 360.0**:**  target\_bearing **=** target\_bearing **-** 360  #és visszaadja a kiszámolt értéket  **return** target\_bearing |

25. ábra: A célpontom irányát kiszámoló kód

## Vezérlés

A kontrolleremben minden lépés elején lekérem a távolságszenzoraim, a GPS és az iránytű értékeit, a jelenlegi időt, a távolságot és az irányt a célponthoz (27. ábra). Ezután, hacsak nem vagyunk már végpontban, elindul az autó a második koordináta felé (az első a kiinduló pont). Elsősorban azt figyelem, hogy elértük-e a következő koordinátát, ha igen, és ez a végpont, megáll az autó, ha nem az, akkor megy tovább a kijelölt útvonalon.

|  |
| --- |
| #az autó GPS-ének beolvasásával kapom meg az autó x és y koordinátáit  current\_x**,** current\_y**,** z **=** gps**.**getValues**()**  #minden lépésben lekérem a jelenlegi időt  current\_time **=** driver**.**getTime**()**  #és lekérem az irányát a célpontomnak  target\_bearing **=** getTargetBearing**()**  #a jelenlegi ponttól a távolság  target\_distance **=** math**.**fabs**(**current\_x **-** target\_x**)** **+** math**.**fabs**(**current\_y **-** target\_y**)**  #a távolságszenzorok értékeinek a lekérése  ds\_values **=** **[]**  **for** i **in** **range(len(**ds\_list**)):**  ds\_values**.**append**(**ds\_list**[**i**].**getValue**())** |

26. ábra: GPS, idő, irány, távolság és szenzorok beolvasása

### Út lekérése

Mielőtt elindul az autó, ha még nincs, kell neki egy útvonal. A járműveimet a controllerArg értékeik alapján különböztetem meg, a 0 értékű a fő autóm. Ha még nincs útvonala és a segédváltozója is 1, amely azt jelöli, hogy kell-e kérni, akkor lefut a pathfinder függvényem (24. ábra). Az így megkapott útvonalat megfordítom, hogy egyesével ki tudjam venni belőle a pontokat. Így az első pont lesz a végpontom, ezt elmentem, mert később kelleni fog, majd kiveszem a végén lévőt, ez lesz a jelenlegi célpontom. Ennek is elmentem a koordinátáit, majd átállítom a segédváltozómat, mert mostmár van útvonalam.

|  |
| --- |
| #a fő autómnak út lekérése  **if** controllerArg **==** 0**:**  **if** **not** path **and** path\_helper **==** 1**:**  **while** end **==** start**:**  end **=** random**.**randint**(**0**,len(**graph**)-**1**)**  #új út kérése kezdő és végpont alapján  path **=** pathfinder**(**graph**,** start**,** end**)**  #út megfordítása, hogy egyesével ki tudjam venni belőle a pontokat  path**.**reverse**()**  #az utolsó pont kivétele  end\_target **=** path**[**0**]**  #az utolsó pont koordinátái  end\_coords **=** graph\_coords**[**end\_target**]**  #a jelenlegi pontom ami most a path-ban az utolsó  current\_target **=** path**.**pop**()**  #ennek a pontnak a koordinátái  target\_coords **=** graph\_coords**[**current\_target**]**  target\_x **=** target\_coords**[**0**]**  target\_y **=** target\_coords**[**1**]**  path\_helper **=** 0 |

27. ábra: Az út lekérése

### Célpont elérése

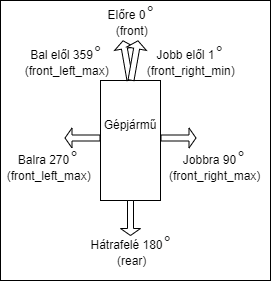
Az autóm akkor éri el a célpontot, ha beér egy az körül lévő mezőbe, azaz a ponttól való távolsága kisebb, mint a mező nagysága. Erre azért van szükség, mert ha nagyon kicsit a célpontom, akkor soha nem fogja tudni pontosan elérni. Itt használom fel az előbb (28. ábra) elmentett végpont koordináta értékeket, mert az autó csak akkor fog megállni, ha ahhoz odaért, ha köztes ponthoz ért, akkor megkapja a következőt, és megy tovább.

|  |
| --- |
| #legelsőnek azt nézem meg, hogy az autó rajta van-e már a célponton  #amikor beér az autó a cépont hatósugarába  **if** target\_distance **<** target\_radius**:**  #megnézzük ez a végpont-e  **if** **(**math**.**fabs**(**current\_x **-** end\_coords**[**0**])** **+** math**.**fabs**(**current\_y **-** end\_coords**[**1**]))** **<** target\_radius**:**  #nullára állítom a sebességet  speed **=** stop\_speed  angle **=** front  stop **=** 1 |

28. ábra: Célpont és végpont elérése

### Bearing felhasználása

Figyelni kell arra, hogy az autó a célpont felé haladjon, ezt a bearing mutatja meg, ettől függően valamekkora angle-ben (szög) kanyarodni kell. Ezt az értéket adom oda a kerekeknek, és ebből számolja ki az AckermannVehicle, hogy a kerekeknek mekkora szögben kell elfordulni, hogy az autóm akkora szögben forduljon, amekkorát én átadok neki értéknek. Ez -0,75 –től (nagyon balra), 0 –án át (egyenes) 0,75 –ig (nagyon jobbra) terjedhet. Felosztottam öt részre az irányomat, ahogy az alábbi ábrán látható.



29. ábra: Bearing felosztás

Először tehát leellenőrzöm a célpontom irányát. Ha az autó mögött van, azaz 90° és 270° között, akkor tolatni kell, és a meglfelelő irányba kanyarodni. Ha balról vagy jobbról, akkor a sebességet is és a kanyarodási szöget is a célpont irányához képest arányosan állítom be, pl. ha az irány 80°, azaz eléggé jobbra van, akkor kisebb sebességgel megy, de jobban kanyarodik, ha csak 10°, akkor alig kell kanyarodni, és mehet nagyobb sebességgel. Ha a célpont előtte van, akkor nem is kell kanyarodni, a sebességet pedig már a távolság alapján számolom, amit alább tárgyalok.

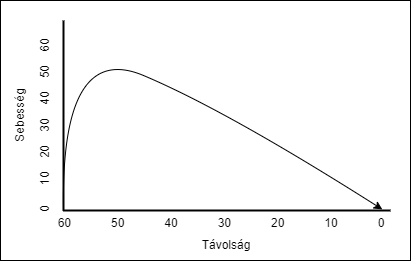
|  |
| --- |
| #hogyha nincs még az autó a célponton  **if** stop **==** 0**:**  #először a célpont irányát nézem meg,  #ezekből mindig csak egy lehet igaz, ezért elif-eket használok  #hogyha az autó mögött van jobbról  #akkor tolatunk és maximális szögben fordulunk balra  **if** target\_bearing **>=** front\_right\_max **and** target\_bearing **<** rear**:**  angle **=** max\_left\_angle  speed **=** reverse\_speed  #hogyha az autó mögött van balról  #akkor tolatunk és maximális szögben fordulunk jobbra  **elif** target\_bearing **>=** rear **and** target\_bearing **<=** front\_left\_min**:**  angle **=** max\_right\_angle  speed **=** reverse\_speed  #hogyha az autó előtt van jobbról  #akkor attól függ a sebesség és a fordulási szög is,  #hogy milyen szögben van az autóhoz a célpont  **elif** target\_bearing **>** front\_right\_min **and** target\_bearing **<** front\_right\_max**:**  speed **=** **((**target\_bearing **-** front\_right\_min**)** **/** **(**front\_right\_max **-** front\_right\_min**))** **\*** **(**min\_speed **-** turn\_speed**)** **+** turn\_speed  angle **=** **((**target\_bearing **-** front\_right\_min**)** **/** **(**front\_right\_max **-** front\_right\_min**))** **\*** **(**max\_right\_angle **-** min\_right\_angle**)** **+** min\_right\_angle  #hogyha az autó előtt van balról  #akkor attól függ a sebesség és a fordulási szög is,  #hogy milyen szögben van az autóhoz a célpont  **elif** target\_bearing **<** front\_left\_max **and** target\_bearing **>** front\_left\_min**:**  speed **=** **((**target\_bearing **-** front\_left\_min**)** **/** **(**front\_left\_max **-** front\_left\_min**))** **\*** **(**turn\_speed **-** min\_speed**)** **+** min\_speed  angle **=** **((**target\_bearing **-** front\_left\_min**)** **/** **(**front\_left\_max **-** front\_left\_min**))** **\*** **(**min\_left\_angle **-** max\_left\_angle**)** **+** max\_left\_angle  #hogyha egyenesen van, akkor nem kell fordulnia az autónak  **elif** target\_bearing **>=** front\_left\_max **or** target\_bearing **<=** front\_right\_min**:**  angle **=** front |

30. ábra: A bearing felhasználása

### Haladás az utakon

Ezután figyelem azt, hogy az autó milyen útszakaszon van éppen. Ha egyenes útvonalon halad vagy kanyarban van, akkor elsősorban tartja a jobb sávot, amíg nincs előtte akadály, ha van, azt kikerüli majd visszaáll a jobb sávba, tartva az objektumoktól a távolságot. Kereszteződés esetén előbb azt figyeli, hogy van-e már valaki a kereszteződésben, ha igen, azt elengedi és utána elkanyarodik a megfelelő irányba, majd amikor már elhagyta a kereszteződést, csak akkor áll újra a jobb sávba, tartva az objektumoktól a távolságot.

Az autó sebessége kanyarodáskor, kereszteződésben és akadálykerüléskor 1 és 10 között mozoghat, azért, hogy ne menjem neki semminek, és el is tudjon kanyarodni rendesen. Egyenes útszakaszon tud felgyorsítani maximum 50-re a távolságtól függően, mintha településen menne. A féket nem tudtam működésre bírni a legtöbb, a Webots-ba beépített, előre elkészített autó modellnél sem, és a sajátomnál sem, ezért a lassítást nem fékkezéssel oldottam meg, hanem a távolságtól függően sokkal hamarabb abbahagyja a gyorsulást, és valójában motorfékkel fékez és állítja be a megfelelő sebességet.



31. ábra: Sebességgrafikon

### Szenzorok felhasználása

Utána figyelem a távolságszenzorok értékét. Az ezekből számolt értékek mindenképpen felülírnak minden más eddigi értéket (sebesség és fordulás), annak érdekében, hogy neki ne menjen valaminek. Megnézem minden szenzoromnak az értékét, amit lekértem a kontroller elején. Az elöl lévő szenzort leszámítva, ha bármelyik beérzékel, akkor a kapott értékkel arányosan az ellentétes irányba kezd el kanyarodni az autó, pl. ha a jobb első szenzorom 0.1-es értéket ad, akkor csak egy kicsit kell balra kanyarodni, de ha a jobb oldali 0.8-as értéket ad, akkor már nagyon balra kell kanyarodni. Ha az elöl lévő szenzor beérzékel, az azt jelenti, hogy akadály van előtte és megkezdi az akadálykerülő manővert, amit alább tárgyalok.

### Akadálykerülés

Ha az elöl lévő szenzor beérzékel, az azt jelenti, hogy akadály van előtte. Először megnézem, hogy az autó a jobb sávban van-e, amennyiben igen, akkor balra kell kerülni, alacsony sebességgel. Ha nagyon közel van az objektum, akkor maximális szögben kanyarodik. Ha az elül lévővel együtt a bal első szenzor is beérzékel, az azt jelenti, hogy a bal oldali falnál van, tehát vissza kell kanyarodni, hogy vissza tudjon állni majd a jobb sávba.

|  |
| --- |
| #ezután jön a kerülési manőver  #ha az elől lévő szenzor érzékel  **if** ds\_values**[**6**]** **>** min\_ds**:**  #és a célpontom bal elől van  #ami azt jelenti, hogy épp a jobb sávban haladok, vagy fordulok  **if** target\_bearing **>** front\_left\_max **-** **(**front\_left\_max **-** front\_left\_min**)** **/** 2 **or** target\_bearing **<** front\_right\_min **+** **(**front\_right\_min **+** front\_right\_max**)** **/** 2**:**  #akkor kerül, tehát balra fordul közepesen  angle **=** max\_left\_angle **/** 2  #minimális sebességel, hogy ne menjen neki az objektumnak  speed **=** min\_speed  #hogyha elég közel van az objektum, akkor teljesen fordul  **if** ds\_values**[**6**]** **>** max\_ds **/** 2**:**  angle **=** max\_left\_angle  #ha beérzékel a bal első szenzor is, az azt jelenti, hogy a bal oldali falnál vagyunk,  #tehát vissza kell majd állni a jobb sávba, ezért jobbra fordulunk  **if** ds\_values**[**2**]** **>** min\_ds**:**  angle **=** max\_right\_angle  speed **=** min\_speed |

32. ábra: Akadálykerülés

### Elsőbbségadás

Amikor beér egy kereszteződésbe, figyeli a szenzorok értékeit, és elengedi a neki jobbra lévő autókat, és azokat akik nagyon közel vannak hozzá. Tehát ha jobbra fordulásnál beérzékel a jobb első szenzor, akkor teljesen megáll az autó. Ha balra fordulásnál a bal vagy a jobb első beérzékel, vagy az elül lévőhöz van nagyon közel egy objektum, akkor is megáll. Ha egyenesen kell mennie akkor a bal első szenzor érzékelése esetén megáll, az elöl lévő és a jobb első esetén csak nagyon lelassít.

|  |
| --- |
| #és végül a kereszteződés elsőbbségmegadás  #hogyha a legutóbbi célpont egy kereszteződés volt  **if** graph\_coords**[**last\_target**]** **==** road\_coords**[**last\_target**]:**  #és még nem vagyunk rajta az egyenes útszakaszon  **if** target\_distance **>** road\_lengths**[**current\_target**]** **/** 2**:**  #és jobbról van a target  **if** target\_bearing **>** front\_right\_min **and** target\_bearing **<** front\_right\_max**:**  #hogyha a jobb első szenzor érzékel  **if** ds\_values**[**3**]** **>** min\_ds**:**  #akkor megáll  speed **=** stop\_speed  #hogyha bal elől van a célpont  **if** target\_bearing **>** front\_left\_max **-** **(**front\_left\_max **-** front\_left\_min**)** **/** 2 **and** target\_bearing **<** front\_left\_max**:**  #hogyha a jobb első szenzor nagyon érzékel  **if** ds\_values**[**3**]** **>** max\_ds **/** 2**:**  #nagyon lelassít  speed **=** min\_speed **/** 2  #hogyha a bal oldali szenzor érzékel  **if** ds\_values**[**1**]** **>** min\_ds**:**  #nagyon lelassít  speed **=** min\_speed **/** 2  #hogyha a bal elől lévő szenzor érzékel  **if** ds\_values**[**2**]** **>** min\_ds**:**  #akkor megáll  speed **=** stop\_speed  #hogyha nagyon balra van a célpont  **if** target\_bearing **<** front\_left\_min **+** **(**front\_left\_max **-** front\_left\_min**)** **/** 2 **and** target\_bearing **>** front\_left\_min**:**  #és az első szenzor nagyon beérzékel, vagy a jobb első  **if** ds\_values**[**6**]** **>** max\_ds **/** 2 **or** ds\_values**[**3**]** **>** min\_ds**:**  #akkor megáll  speed **=** stop\_speed  #ha a bal oldali érzékel  **if** ds\_values**[**2**]** **>** min\_ds**:**  #akkor is megáll  speed **=** stop\_speed |

33. ábra: Elsőbbségadás

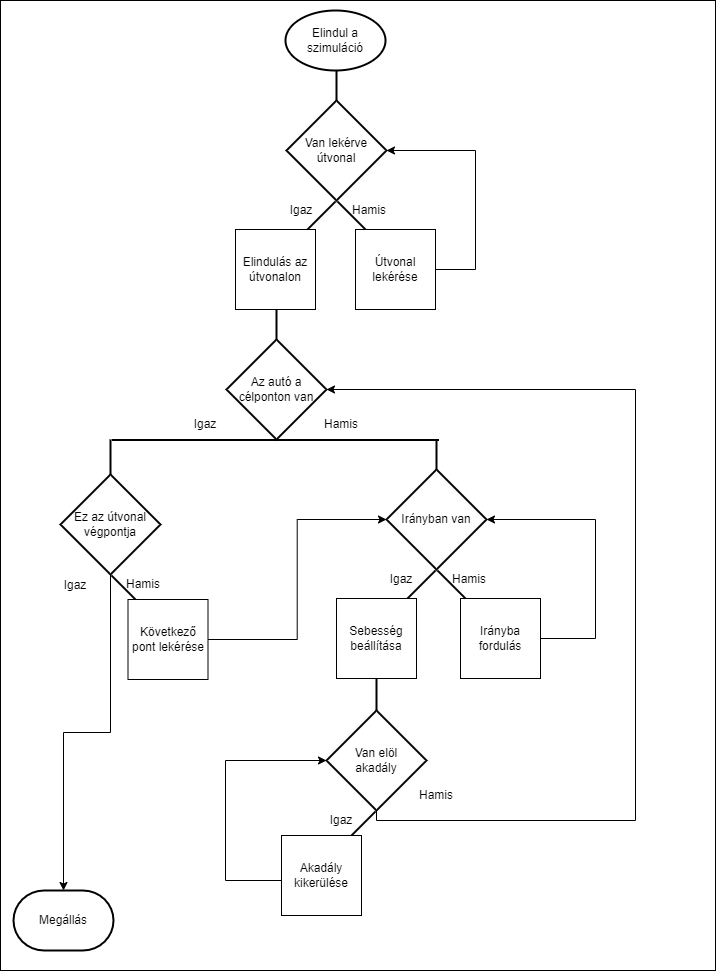
A kontroller végén kizárom a lehetőségét annak, hogy a sebesség nagyobb legyen, mint 50, és a fordulási szög nagyobb legyen a megengedettnél, utána pedig átadom a kiszámolt sebesség és fordulás értéket az autónak. Ha elérte a végpontot akkor pedig megáll.

|  |
| --- |
| #kizárom a lehetőségét, hogy a fordulási szög, vagy a sebesség  #nagyobb legyen a megengedettnél  **if** angle **>** max\_right\_angle**:**  angle **=** max\_right\_angle  **if** angle **<** max\_left\_angle**:**  angle **=** max\_left\_angle  **if** speed **>** max\_speed**:**  speed **=** max\_speed  #átadom a kiszámolt fordulási szög és sebesség értékeket az autónak  driver**.**setSteeringAngle**(**angle**)**  driver**.**setCruisingSpeed**(**speed**)**  **else:**  driver**.**setCruisingSpeed**(**stop\_speed**)**  driver**.**setSteeringAngle**(**front**)**  #lezárom a kontrollert  **pass** |

34. ábra: Rossz érték kizárása és értékátadás

# További fejlesztési lehetőségek

talán ide a döntési fa amit jelenleg használok és hogy azt hogyan lehetne még fejleszteni



35. ábra: Döntési fa

# Összefoglaló

# Irodalomjegyzék

1 – Webots, Cyberbotics, [www.webots.com](http://www.webots.com), 2023.04.30.

2 – Wikipedia Python, <https://en.wikipedia.org/wiki/Python_(programming_language)>

3 – w3schools python - <https://www.w3schools.com/python/>

Webots automobiles - <https://cyberbotics.com/doc/automobile/index>

Webots user guide - <https://cyberbotics.com/doc/guide/index>

Webots reference manual - <https://cyberbotics.com/doc/reference/index>

Legrövidebb út keresők - <https://www.geeksforgeeks.org/shortest-path-unweighted-graph/>

<https://www.geeksforgeeks.org/shortest-path-weighted-graph-weight-edge-1-2/>

Webots tutorials - <https://www.youtube.com/watch?v=luyg3plGujg&list=PLbEU0vp_OQkUwANRMUOM00SXybYQ4TXNF>

python org - <https://www.python.org>

wikipedia - https://en.wikipedia.org/wiki/Main\_Page

# Nyilatkozat

Alulírott ………………..………… szakos hallgató, kijelentem, hogy a dolgozatomat a Szegedi Tudományegyetem, Informatikai Intézet ……melyik tanszék?………………….. Tanszékén készítettem, …………………….…….…… diploma megszerzése érdekében.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök, stb.) használtam fel.

Tudomásul veszem, hogy szakdolgozatomat / diplomamunkámat a Szegedi Tudományegyetem Informatikai Intézet könyvtárában, a helyben olvasható könyvek között helyezik el.

Dátum

Aláírás

# Köszönetnyilvánítás

Ezúton szeretném megköszönni a mentoromnak, Dr. Schäffer Lászlónak, azt, hogy elvállalt engem és a munkámat, illetve tanácsokkal látott el és végig támogatott a szakdolgozatom elkészítése folyamán.