**Szegedi Tudományegyetem**

**Informatikai Intézet**

Szakdolgozat

**Smit Alex**

**2023**

**Szegedi Tudományegyetem**

**Informatikai Intézet**

Önvezető autó tervezése és megvalósítása Webots környezetben

Szakdolgozat

*Készítette*: *Témavezető*:

**Smit Alex 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, autó, szenzor, út, célpont, supervisor, kontroller

# **Tartalomjegyzék**

[1. Feladatkiírás 3](#_Toc133159738)

[2. Tartalmi összefoglaló 4](#_Toc133159739)

[3. Tartalomjegyzék 5](#_Toc133159740)

[4. Bevezetés 7](#_Toc133159741)

[4.1 A Webots program 7](#_Toc133159742)

[4.2 A Python nyelv 7](#_Toc133159742)

[5. A világ felépítése 9](#_Toc133159743)

[5.1 Az autó felépítése 9](#_Toc133159744)

[5.1.1 AckermannVehicle 11](#_Toc133159747)

[5.1.2 Az autó részei 11](#_Toc133159747)

[5.1.3 Szenzorok 11](#_Toc133159747)

[5.2 A környezet 9](#_Toc133159745)

[5.2.1 Az utak 11](#_Toc133159747)

[5.2.2 Az épületek 11](#_Toc133159747)

[5.2.3 A fák 11](#_Toc133159748)

[6. Supervisor 9](#_Toc133159743)

[6.1 Út adatok lekérése 9](#_Toc133159745)

[6.2 Koordináták kiszámolása 9](#_Toc133159745)

[6.3 Adatok kiírása fájlba 9](#_Toc133159745)

[7. Kontroller 9](#_Toc133159743)

[7.1 Adatok beolvasása 9](#_Toc133159745)

[7.2 Legrövidebb út keresés 9](#_Toc133159745)

[7.3 Bearing megállapítása 9](#_Toc133159745)

[7.4 Vezérlés 9](#_Toc133159745)

[7.4.1 Út lekérése 11](#_Toc133159748)

[7.4.2 Célpont elérése 11](#_Toc133159748)

[7.4.3 Bearing felhasználása 11](#_Toc133159748)

[7.4.4 Haladás az utakon 11](#_Toc133159748)

[7.4.5 Szenzorok felhasználása 11](#_Toc133159748)

[7.4.6 Akadálykerülés 11](#_Toc133159748)

[7.4.7 Elsőbbségadás 11](#_Toc133159748)

[8. További fejlesztési lehetőségek 27](#_Toc133159778)

[9. Összefoglaló 28](#_Toc133159779)

[10. Irodalomjegyzék 29](#_Toc133159780)

[11. Nyilatkozat 30](#_Toc133159781)

[12. Köszönetnyilvánítás 31](#_Toc133159782)

# Bevezetés

A mechatronika nevű egyetemi kurzusomon találkoztam először a Webots szoftverrel és ott készítettem el benne az első robotomat is (azt még messze nem lehetett autónak nevezni).

Mivel manapság nagyon felkapott az önvezető autó kérdése, és engem is érdekel a téma, így arra gondoltam, hogy szakdolgozatnak megvalósítom egy nagyon leegyszerűsített változatát.

## 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, mint fő termékét.

## A Python nyelv

A Python egy magas szintű, általános célú programozási nyelv is. A könnyen olvashatóságra törekszik, kifejező behúzások segítségével. A Python dinamikusan gépelhető és a szemetet is ily módon gyűjti. Több programozási paradigmát támogat, beleértve a strukturált, objektumorientált és funkcionális programozást. Átfogó szabványos könyvtára miatt gyakran „batteries included” (elemekkel együtt) nyelvként emlegetik.

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.
* A komplex jobb, mint a bonyolult.
* Számít az olvashatóság.

A Python folyamatosan az egyik legnépszerűbb programozási nyelv.

# A világ felépítése

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:

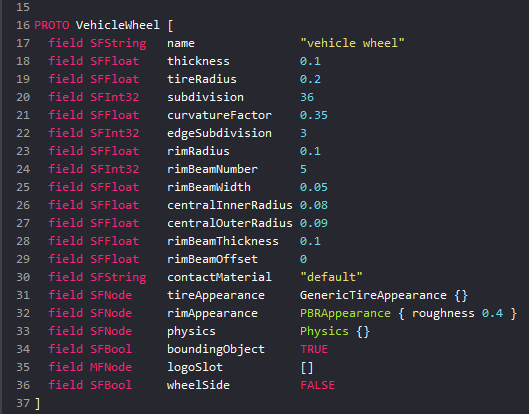
* Transform node, más csomópontok csoportosítására és átalakítások felhasználására szolgálnak.
* Shape node, amelyek meghatározzák a szimulációban szereplő objektumok geometriáját.
* Sensor node, amelyeket érzékelők, például kamerák, lidarok (fényérzékelés és távolságmérés) és közelségérzékelők modellezésére 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++.

## Az autó felépítése

### AckermannVehicle

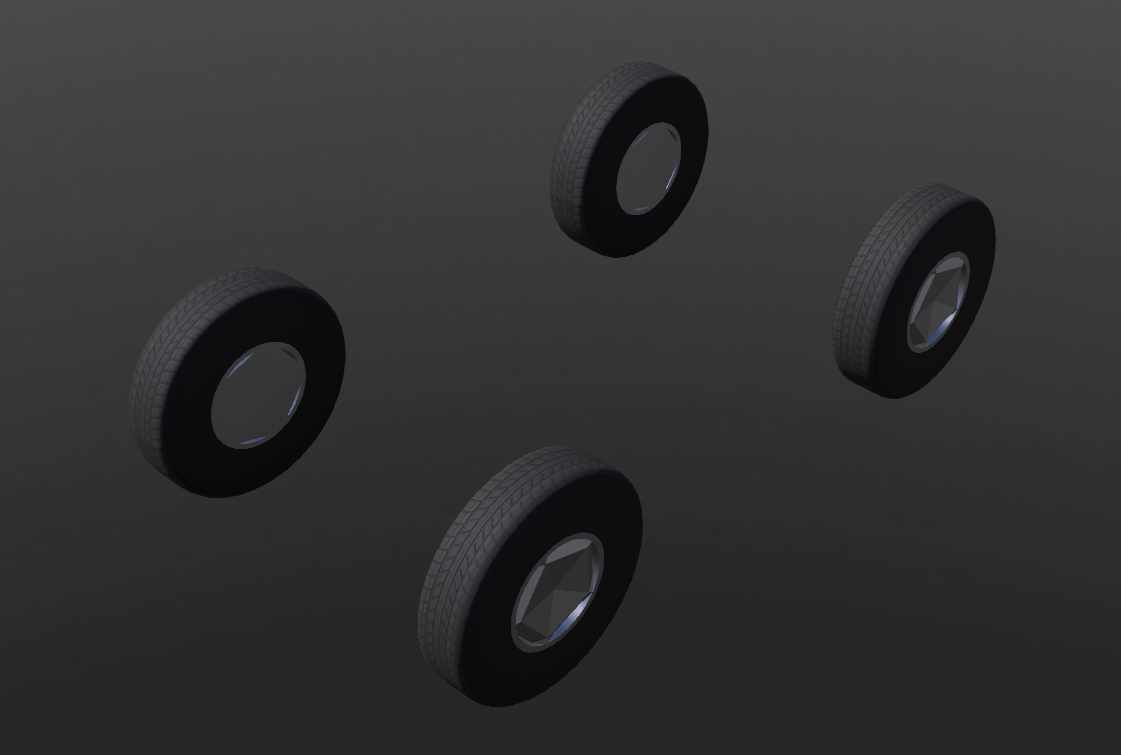
A megvalósítás során törekedtem arra, hogy minél valósághűbb legyen az autóm, ezért a beépített AckermannVehicle-ből indultam ki. Az AckermannVehicle PROTO segítségével könnyedén létre lehet hozni bármilyen járművet, amely megfelel az Ackermann modellnek. Ez a modell azt az autó, vagy más jármű kormányzásánál 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.



5.1. 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.

miért 4x4 meghajtás miért nem rendes leírni



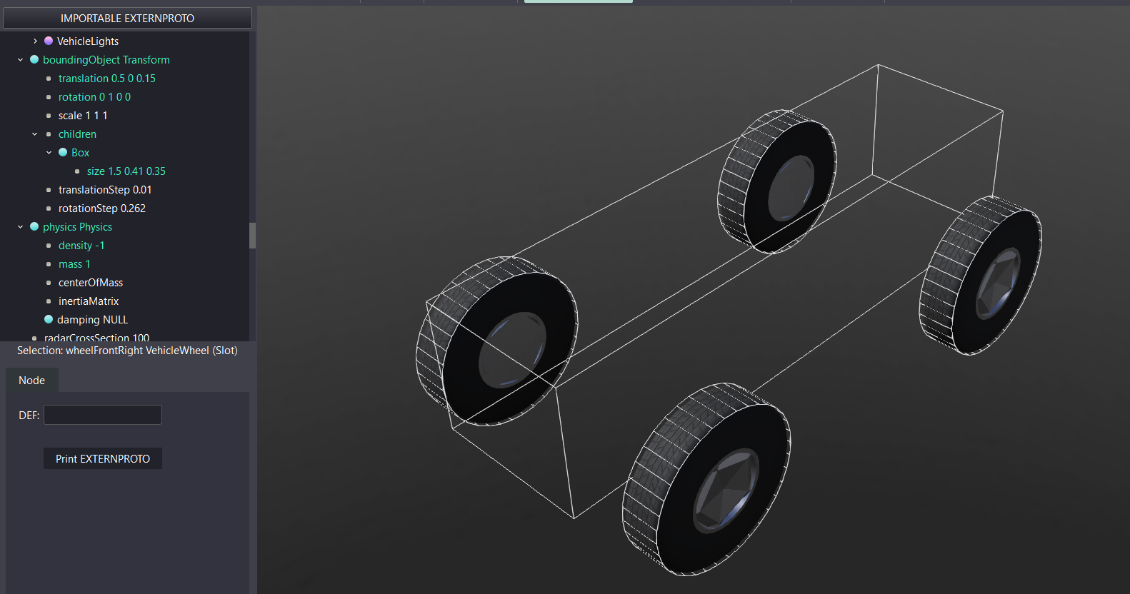
5.1. ábra: Ackermann Vehicle alap

### 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 alábbi képen látható a jármű fizikájának beállítása. A boundingboxObject mezőjéhez hozzáadtam egy Transfrom node-ot, amely téglatest alakú, méretei pedig 1.5 x 0.41 x 0.35.

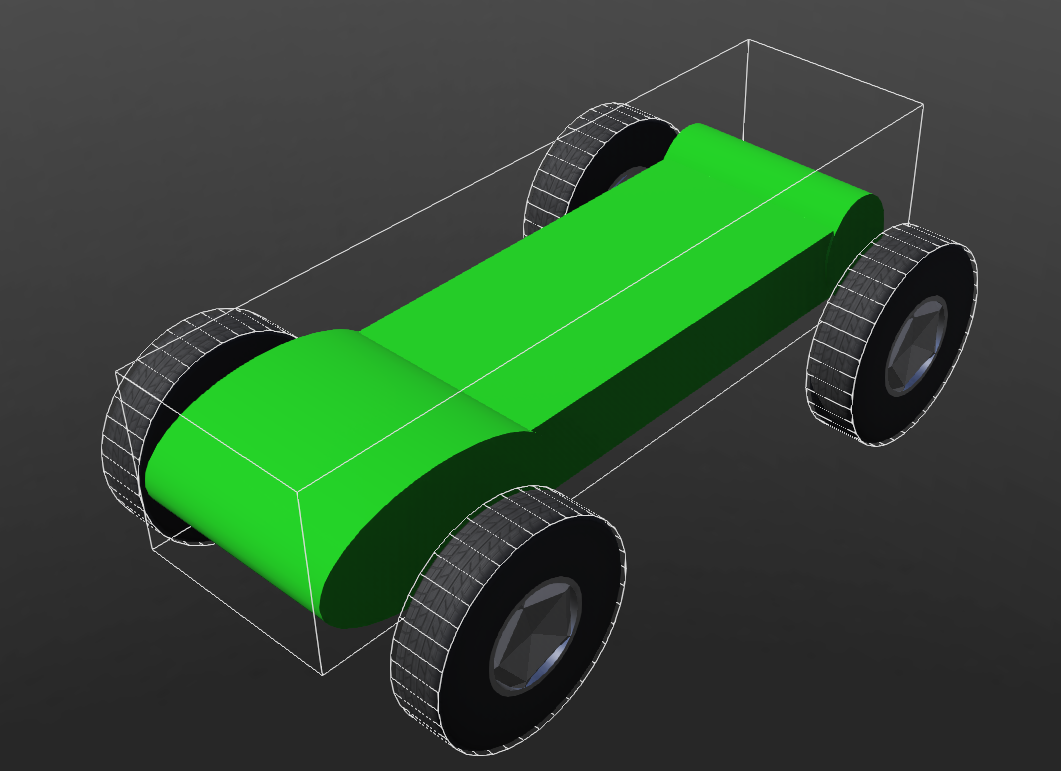
A physics mezőjéhez hozzáadtam a Webots-ba beépített alap fizikát, és átállítottam, hogy a járműnek ne legyen sűrűsége, tömege pedig 1 (ezek a szimuláció megfelelő működéséhez szükségesek).

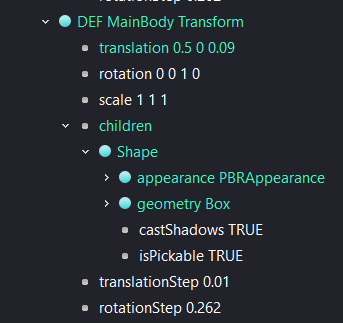
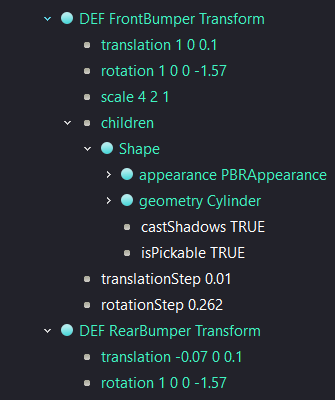


5.2. ábra: Bounding Box

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.



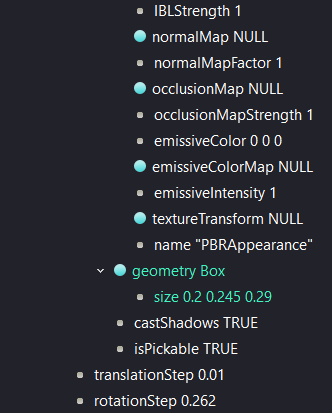
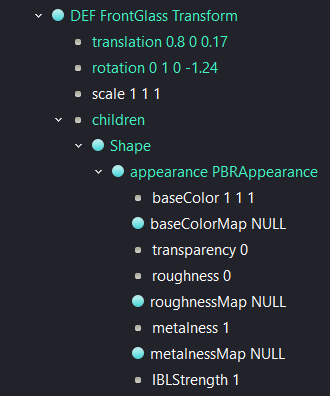
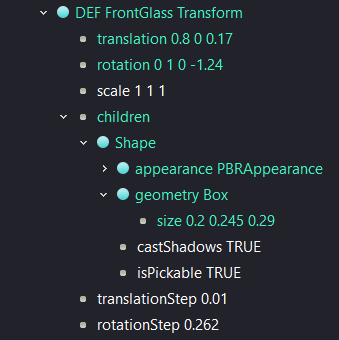
5.3. ábra: Az autó váz alapja

5.3. ábra: Az autó testének és az első lökhárítóknak értékei

Az utastér részt még egy téglatesttel oldottam meg, ami feljebb van és kisebb, mint az előző.

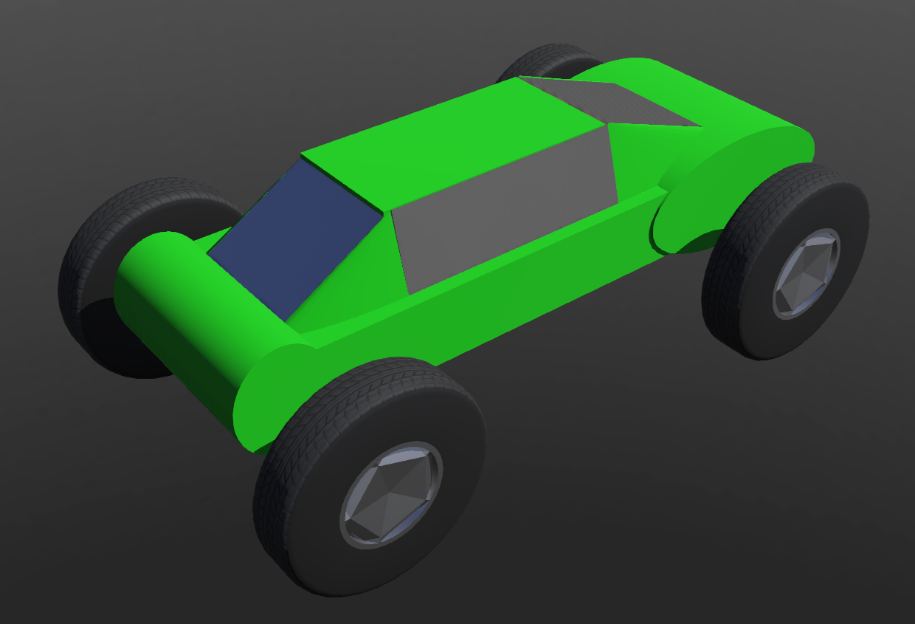
Az ablakok szintén téglatestek, amik el vannak forgatva, fekete színűek, és a metalness (fémesség) tulajdonságuk 1-re van állítva, így sötétített üveg hatását keltik, és visszatükrözik a fényt.

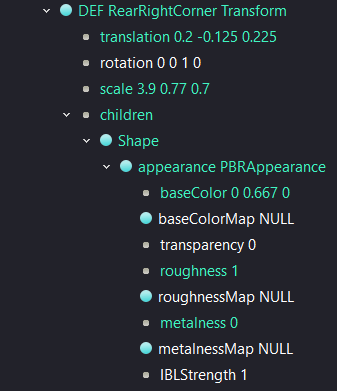
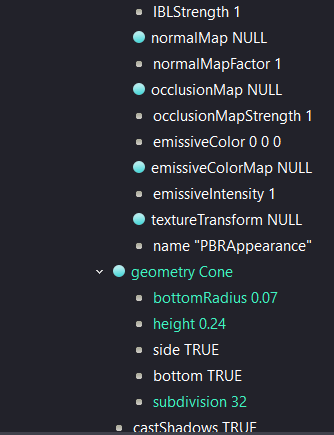


5.4. ábra: Ablakok és felső rész

5.4. ábra: 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.



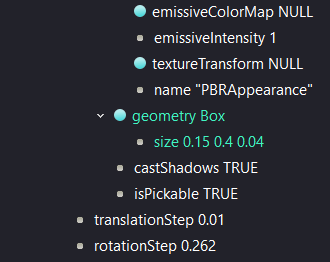
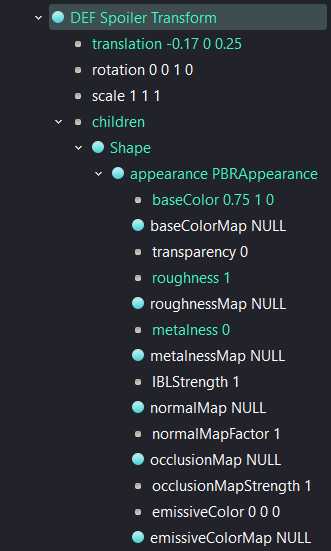
5.5. ábra: Sarkok

5.5. ábra: A jobb hátsó sarok node értékei

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



5.6. ábra: Spoiler

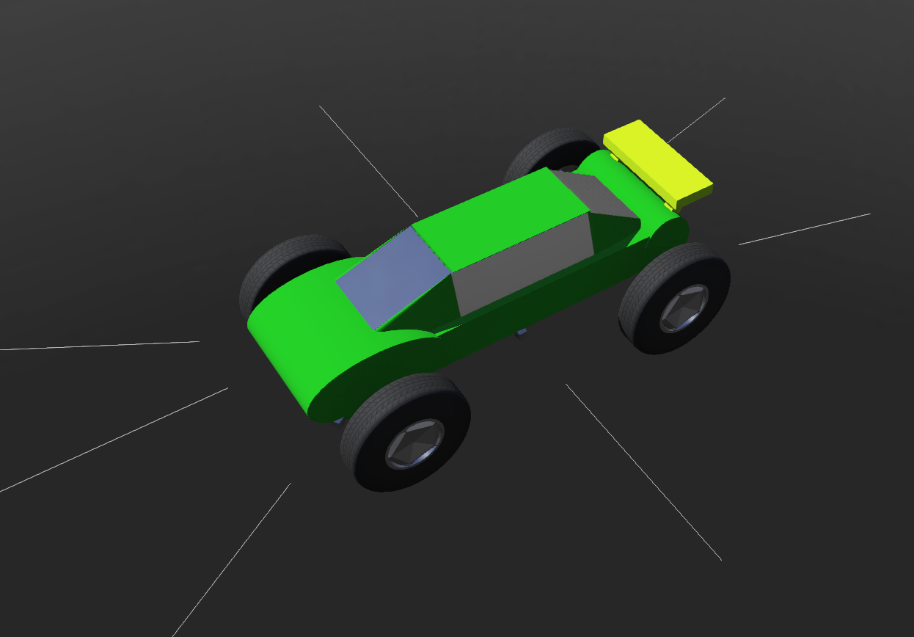


4.6. ábra: A spoiler értékei

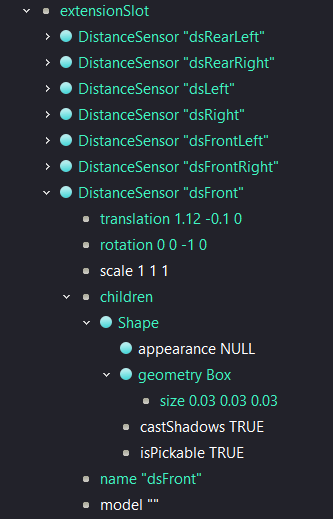
### Szenzorok

7 távolságszenzort használtam, hatot az akadályok és falak kerülésére, egyet pedig, hogy ne menjen neki az úton haladó többi autónak. Minden szenzornak beállítottam a LookUpTable-jét, amely megmondja, hogy milyen távolságról érzékeljen, és hogy ha x távolságra érzékel objektumot, arra mekkora értéket adjon vissza.

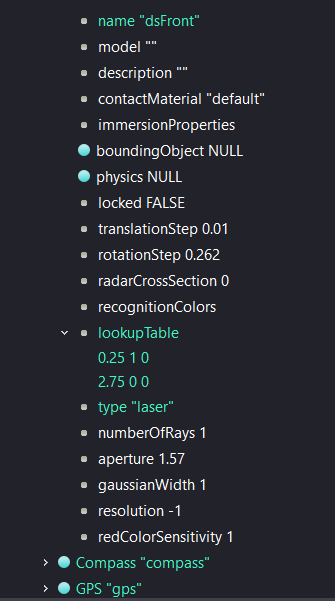
Minden szenzornak be kellett állítanom a lookupTable-jét, ami tulajdonképpen megmondja, hogy a szenzor mekkora távolságra képes mérni. Tehát az alábbi képen látható értékek azt jelentik, hogy az előre néző távolságérzékelő 2.75 egységnyi távolságról kezd el érzékelni, de ekkor még nullás értéket ad vissza, 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.



5.7. ábra: Távolságszenzorok



.7. ábra: Az autón lévő távolságszenzorok



5.7. ábra: Az elöl lévő távolságszenzor lookupTable-je

## A környezet

Az autón kívül a környezetet is nekem kellett létrehoznom. Ehhez a Webots-ba benne lévő már létező elemet használtam fel. Raktam talajt, amihez a Floor-t használtam. Megcsináltam az úthálózatot amin majd a járművek közlekedni tudnak, ezekre a kereszteződésekhez raktam gyalogátkelőhelyeket a szebb kinézet miatt, és hogy ne legyen olyan üres az egész, raktam bele fákat és épületeket.

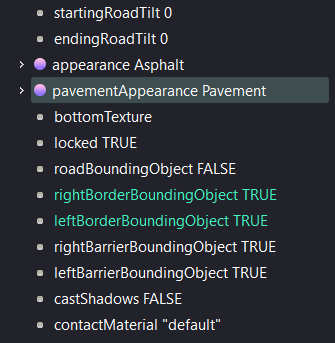
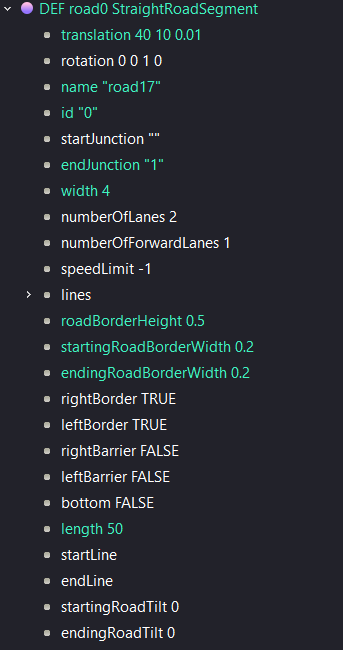


5.8. ábra: A Webots világ

### Utak

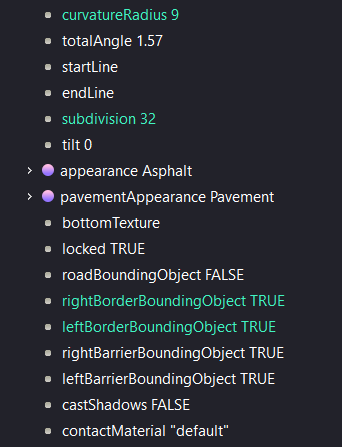
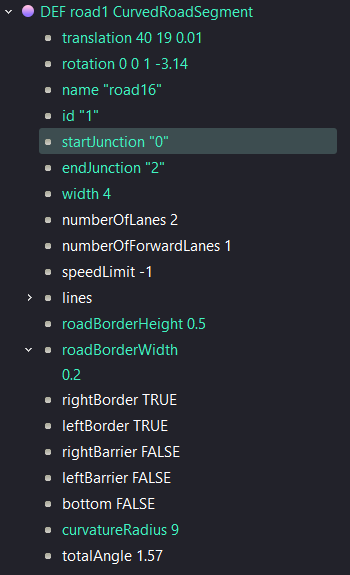
Háromféle út elemet használtam fel: egyenes, kanyar és kereszteződés. Mindegyiket másképp kellett lerakni és kezelni.

Az egyenesnek a saját koordinátái az egyik végén vannak (x.y kép), 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.



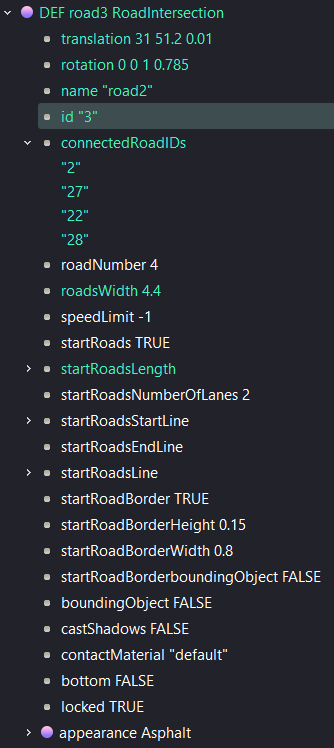
*5.8. ábra: Egy egyenes út értékei*

A kanyarnak a saját koordinátái az kör középpontjában vannak, amit az íve leír (x.y kép), 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.



*5.8. ábra: Egy kanyar értékei*

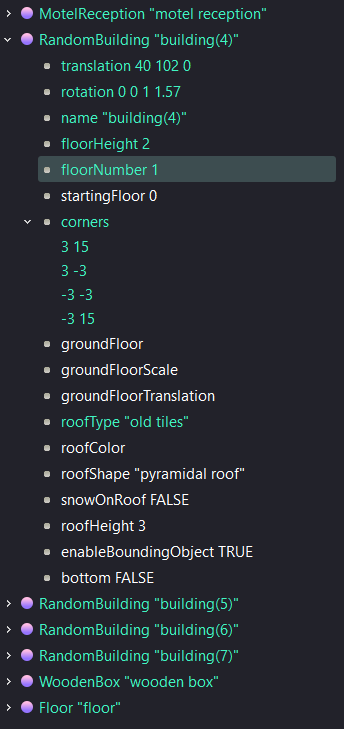
A kereszteződésnek a saját koordinátái egyben a gráf pontom koordinátái is, mivel az elem közepét jelölik (x.y kép). Minden kereszteződés el van forgatva, mert alapból keresztben álltak. Meg kellett adnom a hozzá csatlakozó utak szélességét, és azt, hogy hány van belőlük. Ezen kívül meg kellett adnom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



*5.8. ábra: Egy kereszteződés értékei*

### Épületek

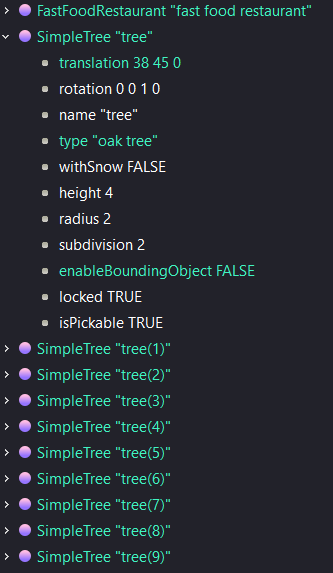
A legtöbb épület a Webots-ba beépített RandomBuilding elem, de van egy motel és egy FastFoodRestaurant, valamint egy hirdetőtábla is. Az egyszerű épületek különböznek a méretükben és mintájukban.



*5.8. ábra: Egy egyszerű épület értékei*

### Fák

A fák a Webots-ba beépített SimpleTree-k. Ezek méreteikben és fajtájukban különböznek. Van tölgy, hackberry (ez más lesz) és mogyoró.



*5.1 ábra: Egy egyszerű fa értékei*

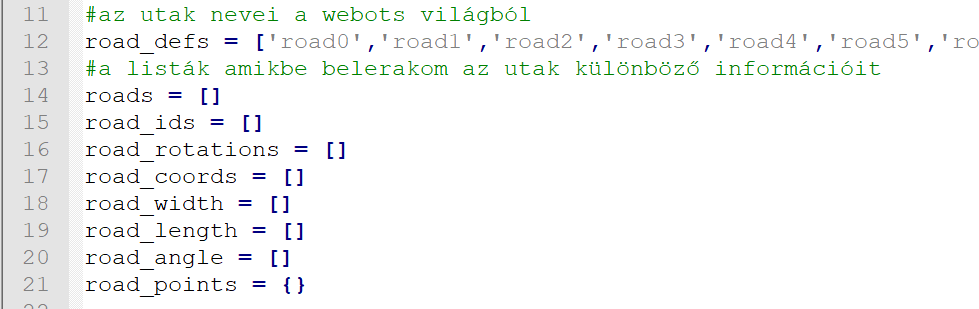
# Supervisor

Az útelemekkel való számolásokat nem a kontrolleremben végeztem el, hanem egy Supervisor-ben. 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

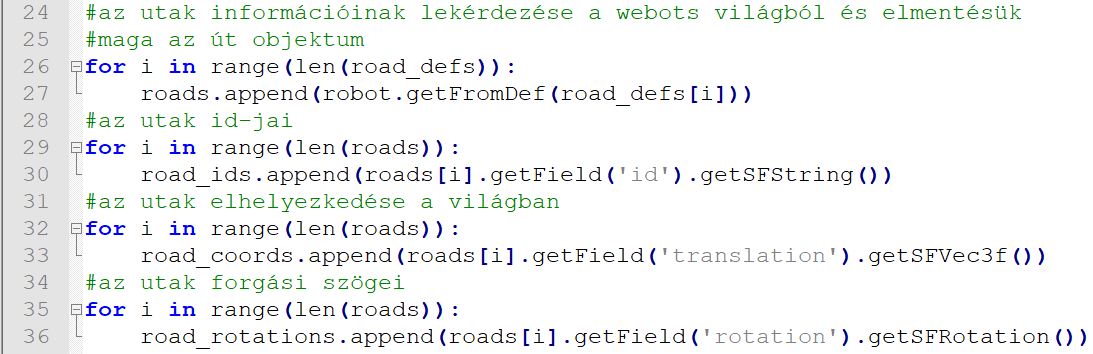
A supervisor-t is azzal kezdem, hogy meghívom a megfelelő könyvtárakat, amelyekre majd szükségem lesz, ilyen maga a Supervisor, a pickle és a math. Ezután létrehozom a robot objektumomat, ami ebben az esetben egy supervisor lesz.

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.



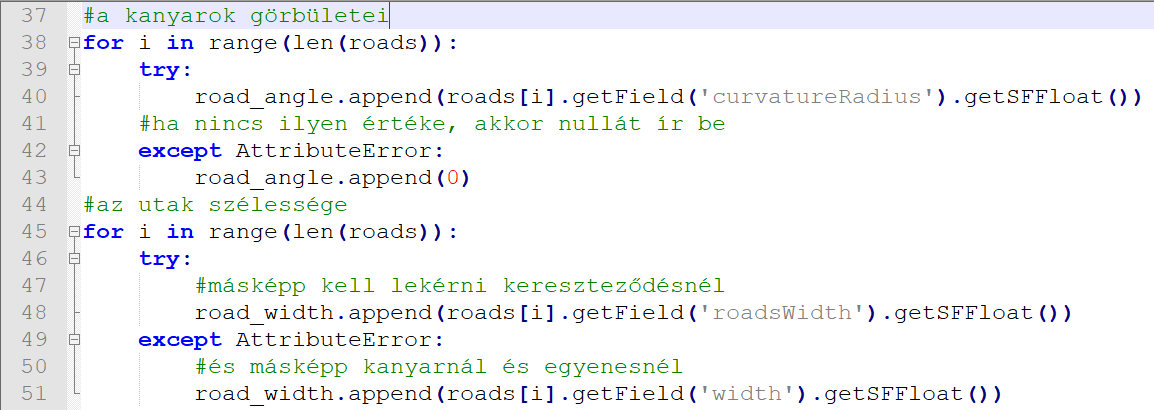
*5.1 ábra: Az utak listái*

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 (xy kép).



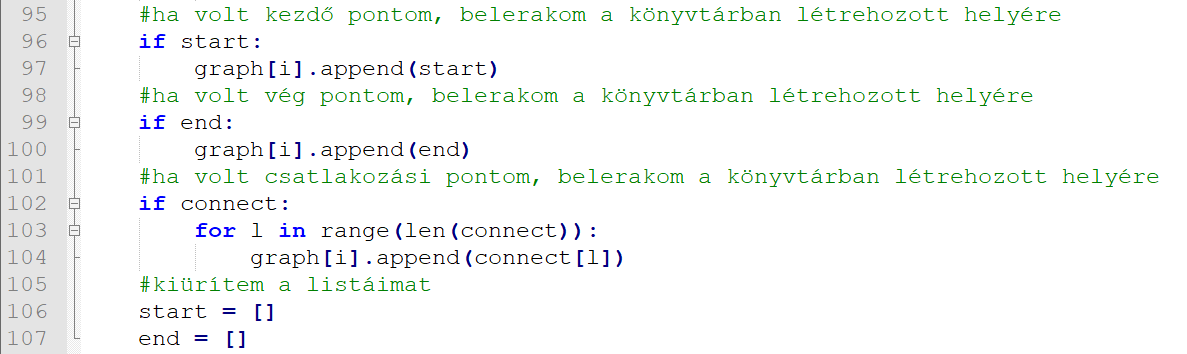
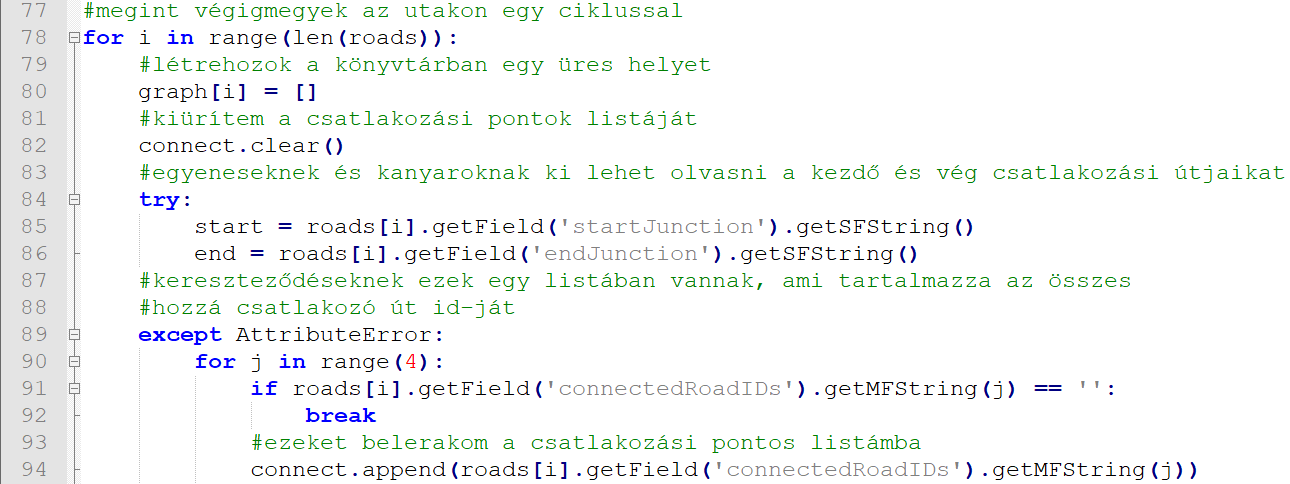
*5.1 ábra: Egyszerű út információk lekérése*

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 (xy kép).



*5.1 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.



*5.1 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.

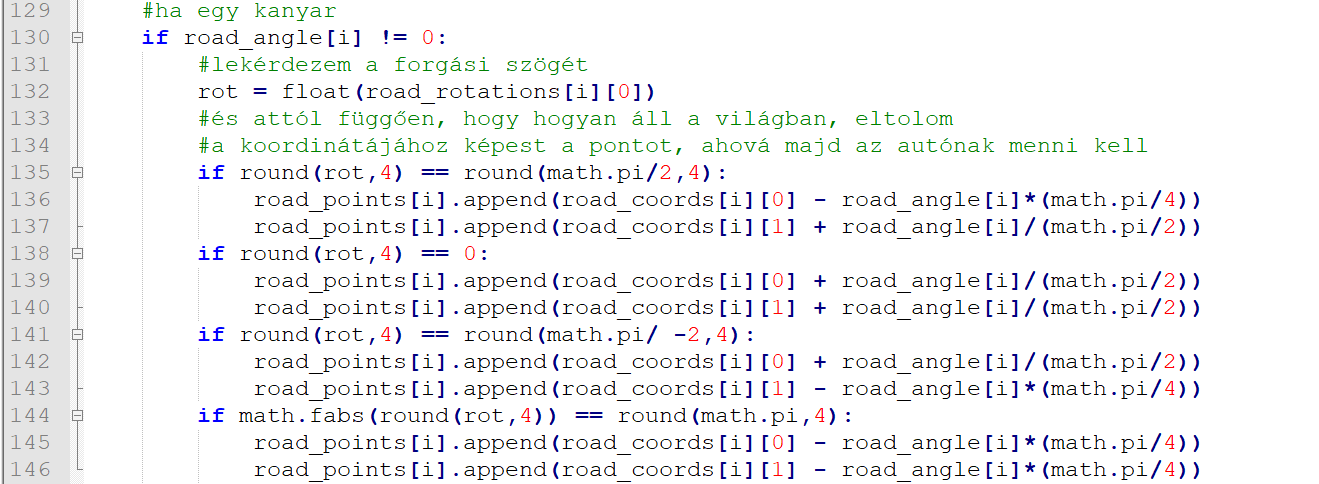
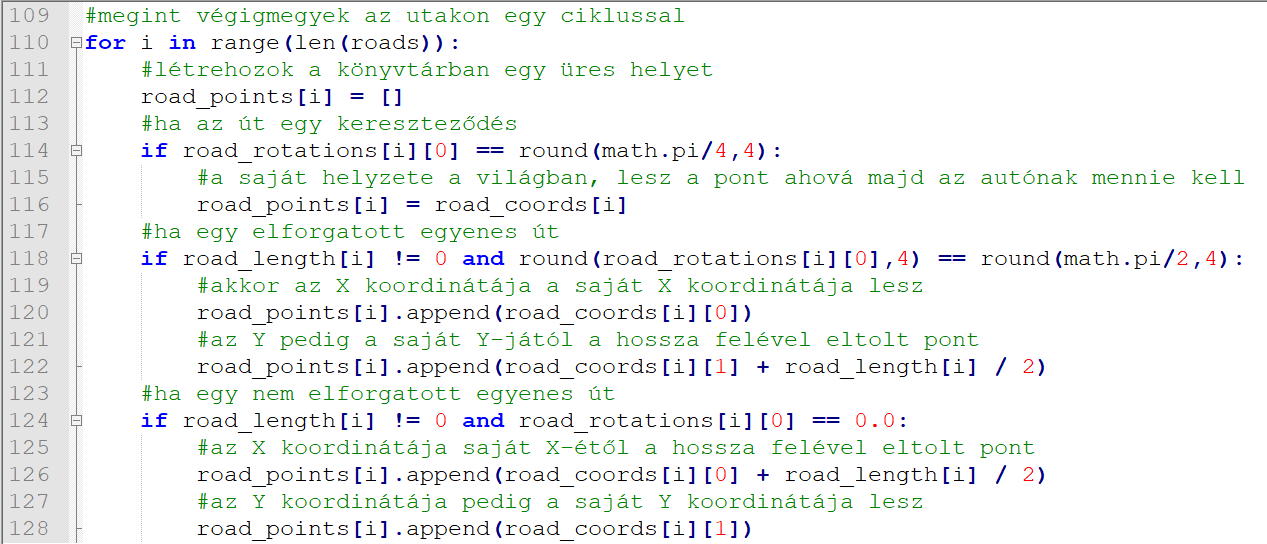
Előbbinél tehát az elfordulás π/2 lesz, a hossza pedig ugye nem 0, ekkor az x koordinátája a sajátja lesz, y-t pedig el kell tolnom az út hosszának felével. Így a pontom az út közepe lesz.

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.

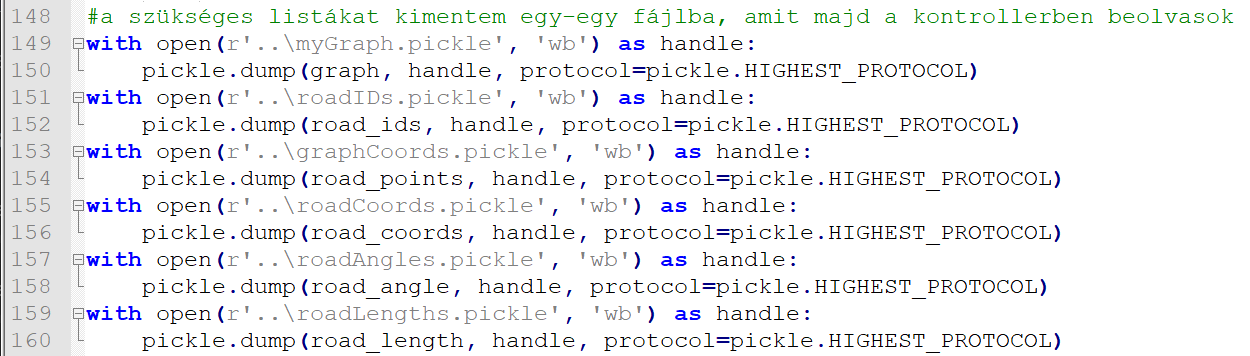
Ezzel megvannak a gráfpontjaim koordinátái, ahová az autónak majd menni kell.



*5.1 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.



*5.1 ábra: A supervisor-ból az adatok kiírása fájlba*

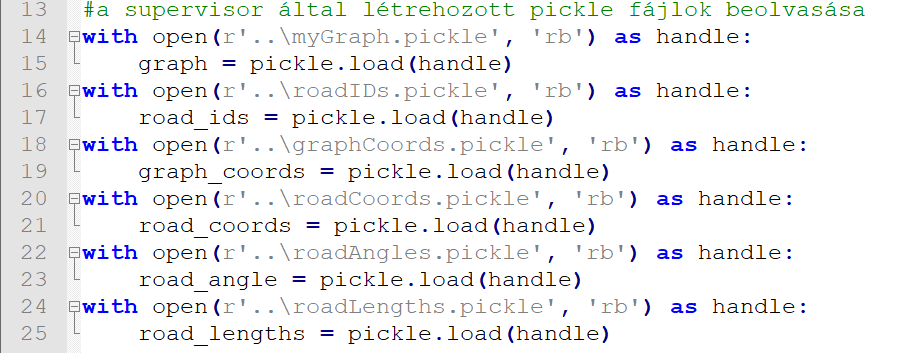
# Kontroller

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

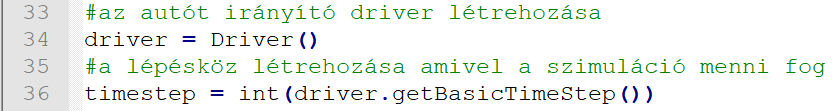
A kód elején itt is meghívom a megfelelő könyvtárakat, amiket majd használni fogok (Driver, math, Compass, GPS, pickle).

Ezután beolvasom a supervisor-rel kiírt fájlokat.



*5.1 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.

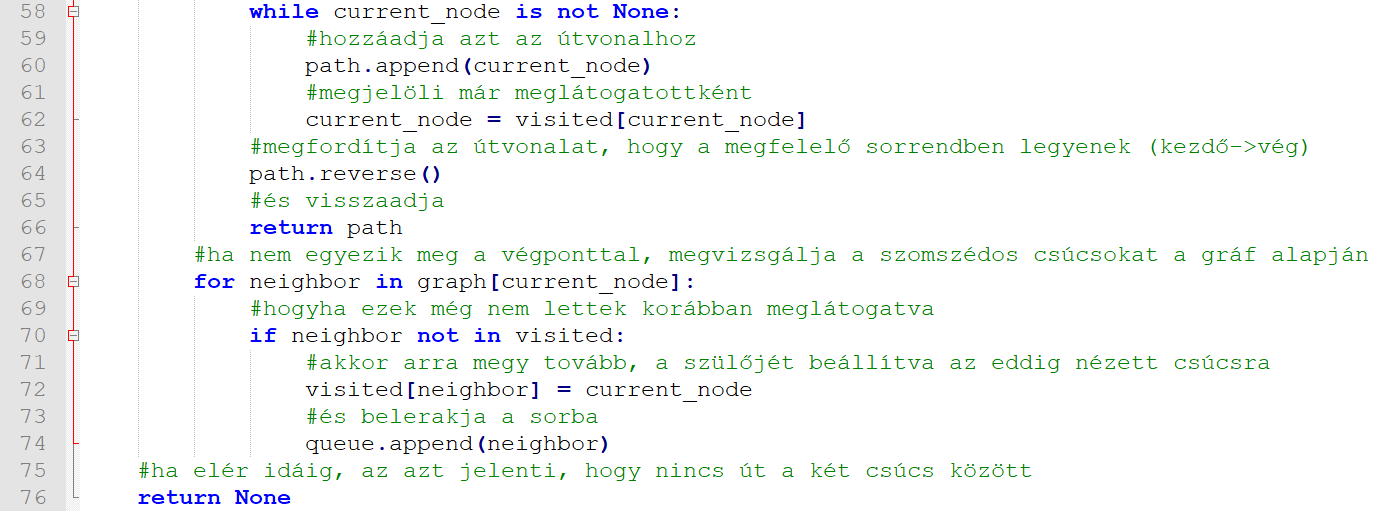
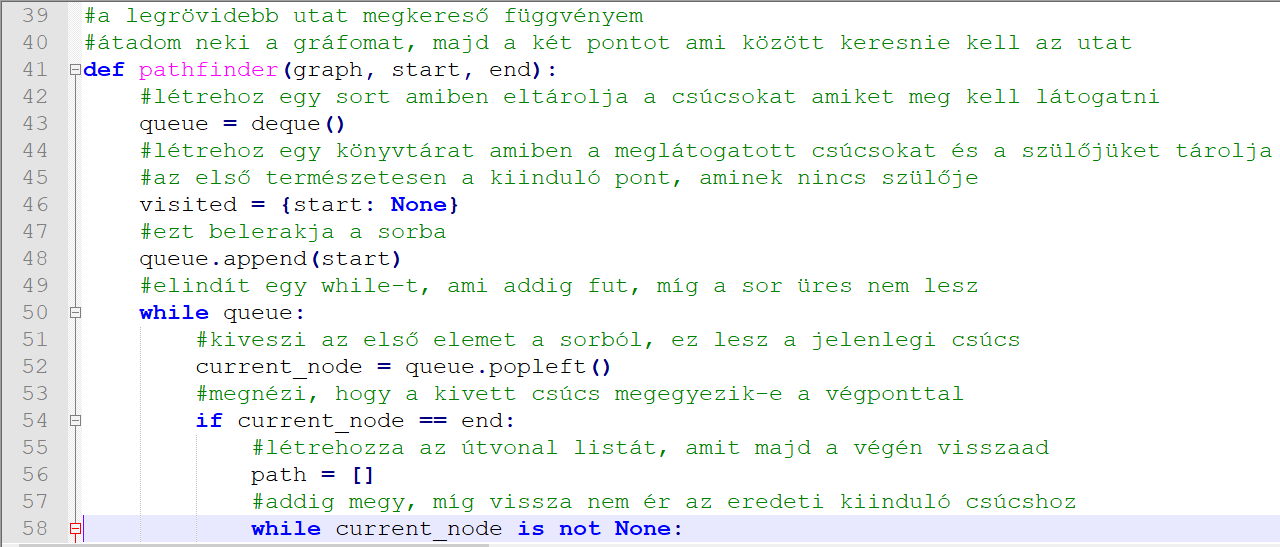


*5.1 ábra: A Driver instance és a timestep*

## 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 megadom 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.

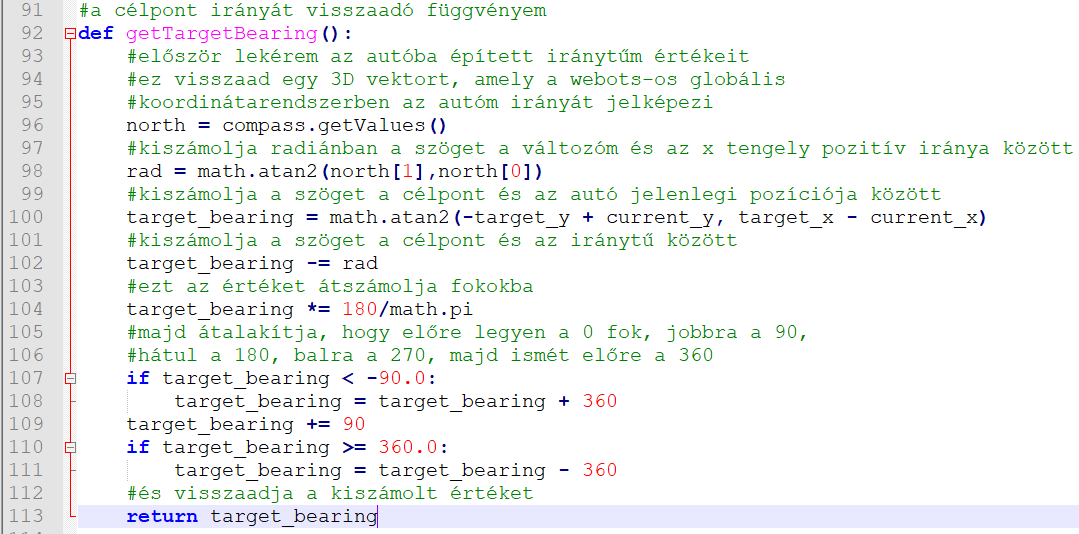
ezt részletesebben leírni



*5.1 ábra: A BFS algoritmus*

## Bearing megállapítása

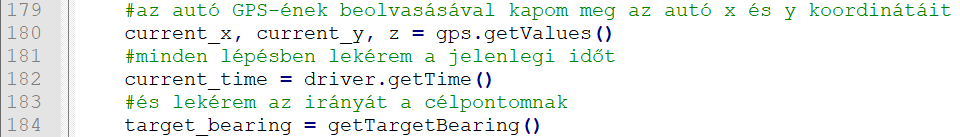
ezt le kell írni

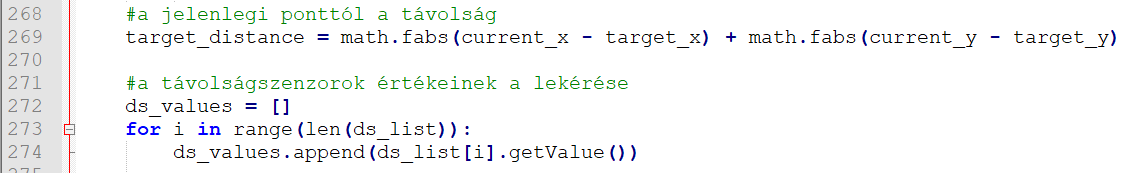


*5.1 ábra: A célpontom irányát kiszámoló kód*

## 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. 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.





*5.1 ábra: GPS, idő, irány, távolság és szenzorok beolvasása*

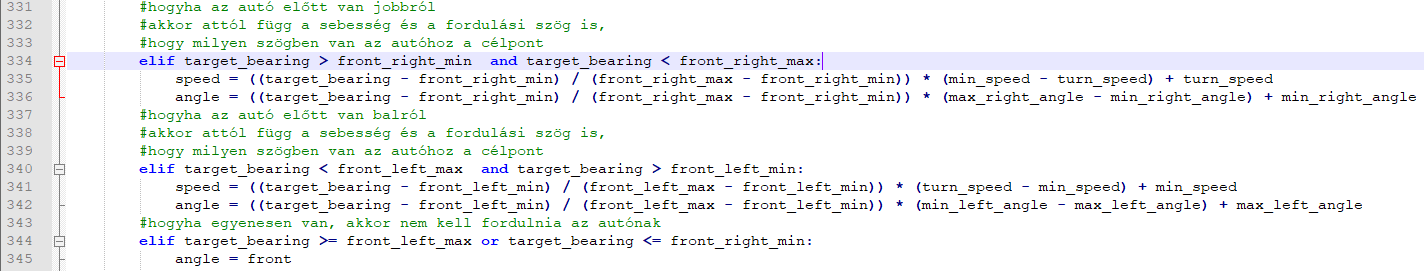
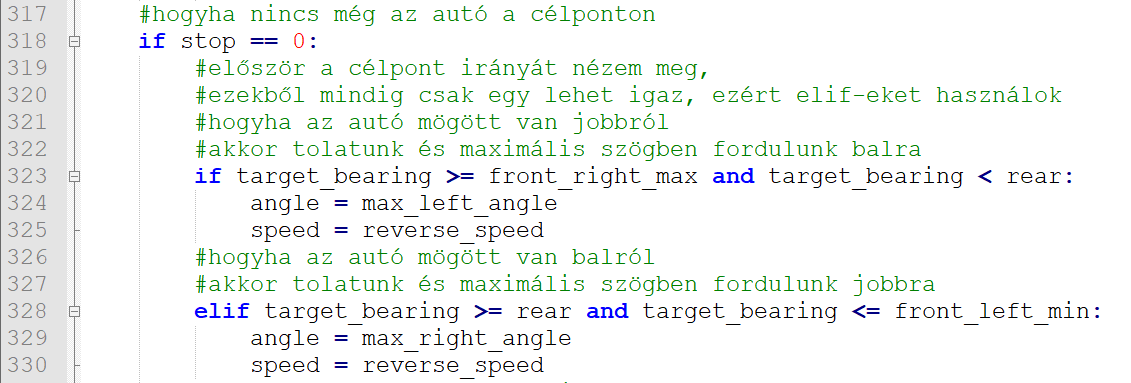
### Út lekérése

### Célpont elérése

### Bearing felhasználása

Figyelni kell arra, hogy az autó a célpont felé haladjon, ezt a bearing (irány) 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 elforulni, 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.

ezt még részletesebben, ez az arányos szabályzás



*5.1 ábra: A bearing felhasználása*

### Haladás az utakon

Ezután figyelem azt, hogy az autó milyen útszakaszon van éppen. Ha egyenesen vagy kanyaron van, akkor mindenképp tartsa a jobb sávot (ha már irányban van), ha pedig kereszteződésben van, akkor előbb forduljon el a megfelelő irányba, majd amikor már elhagyta a kereszteződést, csak akkor tartson jobbra. A sebesség, addig amíg az autó nincs irányban 1-10 (attól függ mennyire kell kanyarodni) között változik, majd ha már irányban van, akkor felmehet 50-ig, mintha településen belül vezetnénk.

ezt még részletesebben

itt leírni hogy nem jó a fék

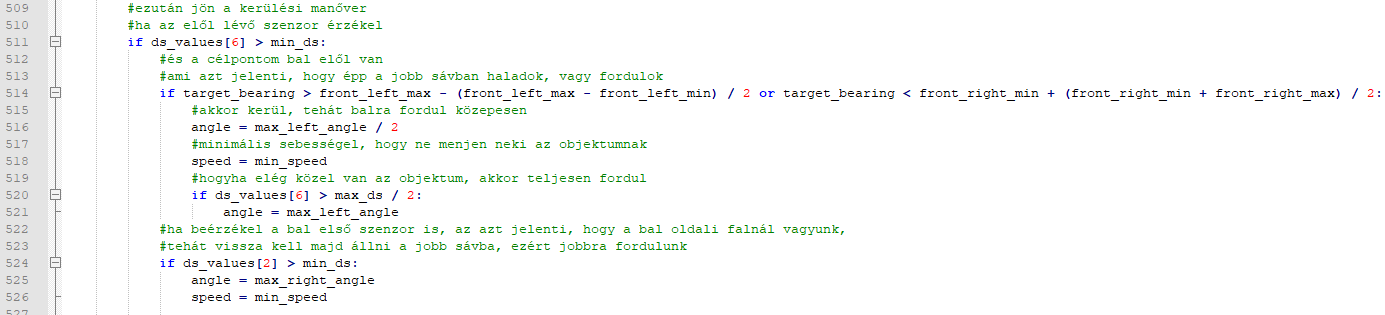
sebesség grafikon

### Szenzorok felhasználása

Legvégül nézem a távolságszenzoraim értékét. Ezeknek az értékei mindenképpen felülírnak minden más eddigi értéket (sebesség és fordulás), annak érdekében, hogy neki ne menjen valaminek.

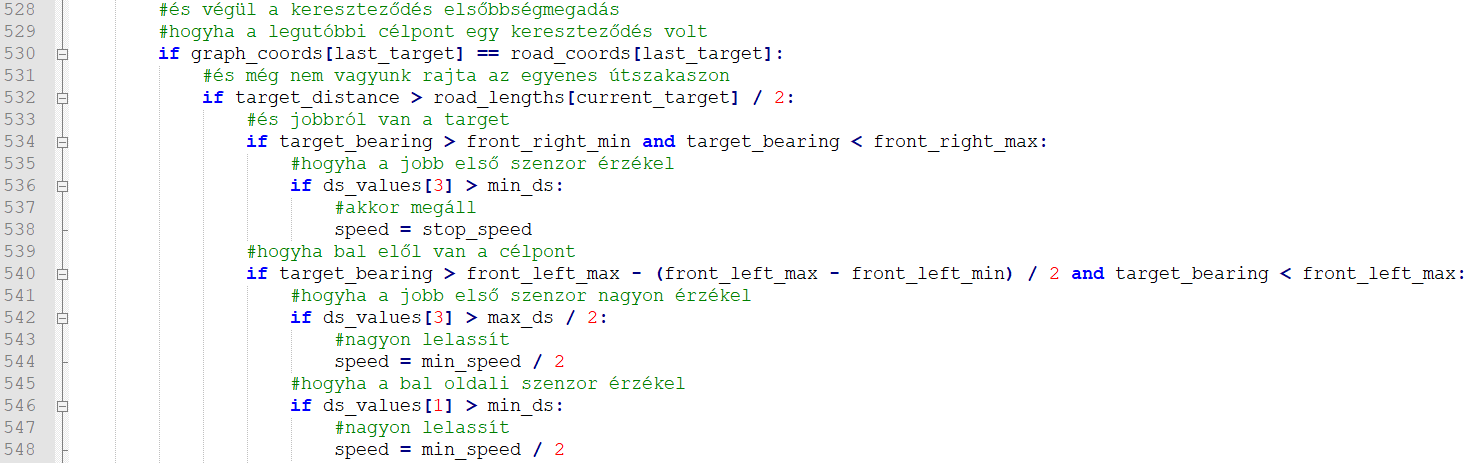
ezt még részletesebben, itt is arányos

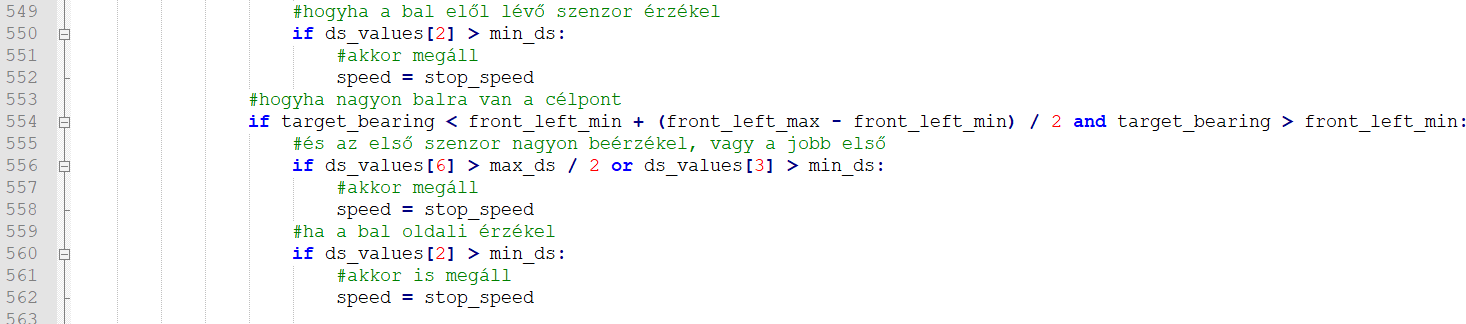
### Akadálykerülés



*5.1 ábra: Akadálykerülés*

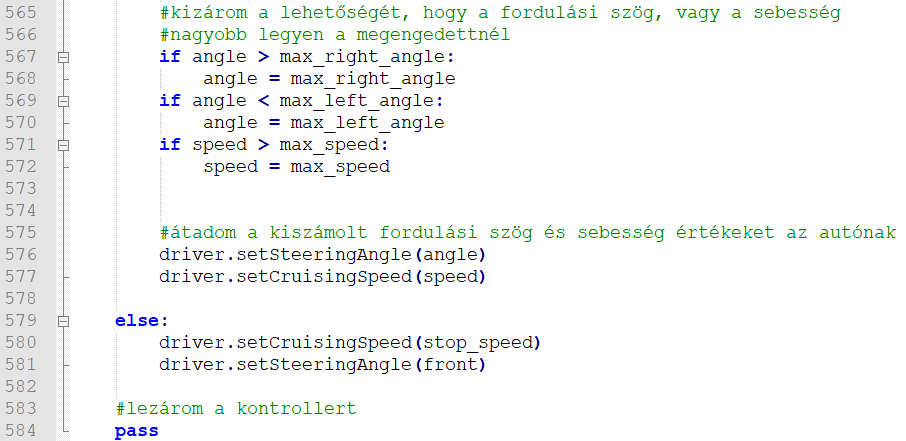
### Elsőbbségadás





*5.1 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.



*5.1 ábra: Rossz érték kizárása és értékátadás*

# További fejlesztési lehetőségek

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

# Összefoglaló

# Irodalomjegyzék

# 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, Schäffer Lászlónak, azt, hogy elvállalt engem és a munkámat, illetve tanácsokkal látott el és végig támogatott a szakdolgozatom elkészítése folyamán.