



Eötvös Loránd Tudományegyetem

Informatikai Kar

Egyetemi-Vállalati Együttműködési Intézet

Mesterséges Intelligencia Tanszék

# Környezet mobil robotok csapatainak megerősítő tanulásának megvalósításához

**Szerző:**

Vörös Döme

Programtervező informatikus BSc.

**Témavezető:**

Gulyás László Csaba Dr.

Egyetemi Docens, PhD

Budapest, 2025

## **SAKDOLGOZAT TÉMABEJELENTŐ**

**Hallgató adatai:**

**Név:** Vörös Döme

**Neptun kód:** QK8IUC

**Képzési adatok:**

**Szak:** programtervező informatikus, alapképzés (BA/BSc/BProf)

**Tagozat :** Nappali

Belső témavezetővel rendelkezem

**Témavezető neve:** Gulyás László

munkahelyének neve, tanszéke: **ELTE IK, Mesterséges Intelligencia Tanszék**

munkahelyének címe: **1117, Budapest, Pázmány Péter sétány 1/C.**

beosztás és iskolai végzettsége: **Docens, PhD**

**A szakdolgozat címe:** Környezet mobil robotok csapatainak megerősítéses tanulásának megvalósításához

**A szakdolgozat témája:**

(A témavezetővel konzultálva adja meg 1/2 - 1 oldal terjedelemben szakdolgozat témájának leírását )

A szakdolgozat egy olyan szoftver kialakítása lenne, amely lehetőséget ad arra, hogy mobil robotok csapatait tudjuk megerősítéses tanulással tanítani. A megerősítéses tanulás feladatait a Stable Baselines könyvtár nyújtotta lehetőségekkel tervezem megvalósítani. A robotcsapatok teljesítményét a Gazebo nevű szimulációs platformban tesztelném, amelyen belül Turtlebot típusú robotot alkalmaznék. Az aktuális állapotot a Gazebo-ban szimulált TurtleBot-ok viselkedésének mérésével értékelném ki. Egy könnyen használható, felhasználóbarát környezet lenne a cél, amelyben lehetséges a mobil robotok csapatainak tanítása, minél valószerűbb kiértékelések mellett.

A környezet implementációjának nyelvét az alapján szeretném eldönteni, hogy a programok és könyvtárak, amelyeket alkalmaznék azok milyen nyelven íródtak és ha úgy érzem, hogy ebben megvalósítható az én elképzelésem, akkor ezen használt programok és könyvtárak nyelvében fogom megírni a környezetemet. Ellenkező esetben megvizsgálnék hasonló megoldásokat és hogy azok mennyire felelnek meg nyelvileg az elképzeléseimnek.

Budapest, 2023. 12. 01.

## Tartalom

<b>1. Bevezetés .....</b>	<b>6</b>
<b>2. Felhasználói dokumentáció .....</b>	<b>8</b>
<b>2.1. Rendszerkövetelmények .....</b>	<b>8</b>
2.1.1. Hardware követelmények .....	8
2.1.2. Program beszerzése .....	9
<b>2.2. Telepítés .....</b>	<b>9</b>
2.2.1. Program kicsomagolása .....	10
2.2.2. Docker telepítése .....	11
2.2.3. A program telepítése .....	13
<b>2.3. Indítás .....</b>	<b>14</b>
2.3.1. A program funkciójának beállítása .....	15
2.3.2. Program leállítása .....	15
2.3.3. Program törlése .....	16
<b>2.4. A program funkciói .....</b>	<b>16</b>
2.4.1. Új modell tanítása .....	16
2.4.2. Tanítás folytatása .....	19
2.4.3. Tanítási adatok kiírása .....	21
<b>2.5. Konfig és a program funkcióinak konfiguráljainak felépítése .....</b>	<b>23</b>
2.5.1. Új modell tanítása konfiguráljának felépítése .....	24
2.5.2. Tanítás folytatása konfiguráljának felépítése .....	25
2.5.3. Tanítási adatok kiírása konfiguráljának felépítése .....	26
<b>2.6. A létrehozott nyersanyagok és annak kezelése .....</b>	<b>26</b>
<b>2.7. A program logolása és annak értelmezése .....</b>	<b>28</b>
<b>3. Fejlesztői dokumentáció .....</b>	<b>30</b>
<b>3.1. Forráskód beszerzése .....</b>	<b>30</b>

<b>3.2.</b>	<b>A forrás könyvtárszerkezete.....</b>	<b>30</b>
<b>3.3.</b>	<b>Függőségek.....</b>	<b>32</b>
<b>3.3.1.</b>	<b>A függőségek beszerzése .....</b>	<b>33</b>
<b>3.3.2.</b>	<b>A függőségek frissítése .....</b>	<b>33</b>
<b>3.4.</b>	<b>A forrásmappa felépítése .....</b>	<b>33</b>
<b>3.4.1.</b>	<b>Fejlesztői konténer felépítése .....</b>	<b>33</b>
<b>3.4.2.</b>	<b>Fejlesztői Docker konténer felépítése .....</b>	<b>36</b>
<b>3.4.3.</b>	<b>Környezet felépítése.....</b>	<b>38</b>
<b>3.4.4.</b>	<b>Robot és a környezet kapcsolata, kommunikációja .....</b>	<b>39</b>
<b>3.5.</b>	<b>Követelmény-specifikáció .....</b>	<b>40</b>
<b>3.5.1.</b>	<b>Követelményelemzés.....</b>	<b>40</b>
<b>3.5.2.</b>	<b>Megvalósíthatósági terv.....</b>	<b>41</b>
<b>3.5.3.</b>	<b>Nem funkcionális követelmények.....</b>	<b>42</b>
<b>3.5.4.</b>	<b>Funkcionális követelmények.....</b>	<b>42</b>
<b>3.6.</b>	<b>Felhasználói esetek.....</b>	<b>43</b>
<b>3.6.1.</b>	<b>Új modell tanítása felhasználói esetei .....</b>	<b>43</b>
<b>3.6.2.</b>	<b>Tanítás folytatása felhasználói esetei .....</b>	<b>43</b>
<b>3.6.3.</b>	<b>Tanítási adatok kiírása felhasználói esetei .....</b>	<b>44</b>
<b>3.7.</b>	<b>Rendszerarchitektúra .....</b>	<b>45</b>
<b>3.7.1.</b>	<b>Programnyelv és fejlesztői környezetek .....</b>	<b>45</b>
<b>3.7.2.</b>	<b>Főbb komponensei a programnak .....</b>	<b>45</b>
<b>3.7.3.</b>	<b>A környezet és a program folyamatábrái .....</b>	<b>45</b>
<b>3.8.</b>	<b>A fejlesztői környezet .....</b>	<b>47</b>
<b>3.8.1.</b>	<b>A fejlesztői környezet működése .....</b>	<b>49</b>
<b>3.9.</b>	<b>Tesztkörnyezet.....</b>	<b>50</b>
<b>3.10.</b>	<b>A tesztelési terv .....</b>	<b>50</b>

3.10.1.	Manuális tesztelés .....	50
3.10.2.	Automatikus tesztelés .....	50
3.10.3.	Tesztelési eredmények .....	54
3.10.4.	Tesztek részletes leírása .....	55
3.11.	Verziókezelés .....	56
3.12.	A program logolása.....	57
3.13.	A program konfigurálása .....	58
4.	Összefoglalás.....	60
4.1.	További fejlesztési lehetőségek .....	60
5.	Irodalomjegyzék .....	61

# **1. Bevezetés**

Manapság az egyik legnépszerűbb téma a mesterséges intelligencia, viszont a legtöbb embernek nagyon kevés tudása van arról, hogy mennyi különböző helyen lehet alkalmazni ezt a témát. A legtöbb ember úgy ismeri, mint az eszköz, amely megír neked szövegeket vagy képeket generál, viszont nagyon sok tudományágban már majdnem elengedhetetlen a mesterséges intelligencia. Például az orvostudományban, ahol mesterséges intelligenciát tanítanak arra, hogy különböző betegségeket fedezzen fel képek alapján, például a rákkutatásban nagyon hasznos tud lenni [1], mivel már nagyobb valószínűséggel fogja ő felfedezni a képen a problémát, mint a szakember. A mindennapi életünkben nem is vesszük észre, de vannak olyan helyek, ahol mesterséges intelligenciát használnak évek óta, csak nem gondoltuk róla, hogyan működnek eddig, ezért nem is tudtuk, hogy ezt a technológiát használják a cégek. Ilyen például a különböző online vásárlási platformok vagy streaming szolgáltatások ajánlórendszerei [2], vagy például az email szolgáltatók ahogyan szűri a spam emaileket, ezáltal megelőzve, hogy a kártékony levelek elárhassák a felhasználó postaládáját [3].

A saját véleményem szerint a legizgalmasabb ága a mesterséges intelligenciának a robotika. A tény, hogy robotokat meg tudunk tanítani akármilyen feladatra, az szimplán lenyűgöző. Pár száz évvel ezelőtt el se tudták volna az emberek képzelni, amit manapság a robotok probléma nélkül meg tudnak oldani. A tény, hogy már vannak olyan gyárak, ahol például csak robotok végeznek el minden fizikai feladatot, az elképesztő a számomra. A robotika egyik legnépszerűbb ága, amikor egy adott robotot megkérnek arra, hogy teljesítsen egy feladatot, és ő azt magától megtanulja. Ezt nevezik megerősítéses tanításnak. A lényege a módszernek az, hogy a robot egy ideig végzi a feladatát, majd utána rájön, hogy mit is csinált, és az alapján kap jutalmat vagy büntetést, hogy mennyire volt közel a cég eléréséhez. Egy idő után pedig a sok visszajelzés után már tudni fogja, hogyan és mikor kell mozogni.

Ennek a fajta tanításnak sok módja van. Például vannak különböző szoftverek, amelyek ilyen szimulálásra vannak kitalálva, ilyen például a Gazebo is, amelyet a későbbiekben fogok használni. Viszont elég programozói tudással akármilyen környezetben tudjuk ezt a technológiát használni, például akár videójátékok esetén is. A robotot beletesszük a kedvenc videójátékunkba és megmondjuk neki, mit kell tudnia csinálni és szépen lassan megtanulja

magától, hogyan kell ezt használni. Például ezek már annyira népszerűek, hogy az egyik könyvtár a Stable Baselines 3 már támogatja azt is, hogy régi Atari játékokat tanítsunk megerősítéses tanítás segítségével [4], viszont akár az interneten nagyon sok példát találunk modern játékoknál is, ahol ezt a tanítást alkalmazzák.

Mégis azt gondolom, hogy a robotok tanítása egy nagyon komplikált feladat. A szakdolgozatom írása során nagyon sok mindent tanultam a témával kapcsolatban és nem egy egyszerű téma és teljesen megértem, hogy ha valaki feladná a tanulását, mivel neki túl komplikált. A célom a dolgozattal az lenne, hogy át tudjak adni egy környezetet, amely egy belépőpont lehet valakinek, aki nem teljesen ért még a témához, viszont szeretne többet tanulni vele kapcsolatban.

## 2. Felhasználói dokumentáció

Ebben a fejezetben szeretném bemutatni, hogy a felhasználónak milyen rendszerrel kell rendelkeznie, hogy a program megfelelően működjön. Ezen kívül bemutatom, hogy a felhasználó hogyan tudja telepíteni, elindítani és használni a programot. Azon kívül bemutatom a programnak a felépítését, funkcióit és működését.

### 2.1. Rendszerkövetelmények

#### 2.1.1. Hardware követelmények

A program futtatásához Ubuntu Linux rendszerre lesz szüksége a felhasználónak. Alapvetően erre azért van szükség, mivel a komplikált környezeti rendszert csak Ubuntu-ban lehetett biztosítani, ezzel mindig garantálva a folyamatos és rendeltetésszerű futást. Tesztelésem során arra jutottam, hogy a virtuális gépek nem elég erősek egy ilyen szoftver futtatásához. Minimum hardware követelménye a Gazebo alkalmazásnak van, ezek a következők [5]:

- Processzor (CPU) — Quad Core Intel i5 vagy vele megegyező erejű processzor
- Memória (RAM) — 4 GB vagy több
- Videókártya (GPU) — Minimum egy dedikált videó kártya, legalább 1 GB rammal

Tárhely szempontból annyira van szükségünk amennyi helyet a Docker által létrehozott image fog foglalni. Általában a Docker image ilyen 15-16 GB helyet fog használni, azért én ajánlom, hogy 25 GB szabad területet hagyjunk a programnak.



d@: ~/Github/MLSzakdoga/deploy\$ docker image ls				
REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
mlszakdoga_deploy	latest	eeb99b17a555	20 hours ago	15.6GB

1. ábra: Docker által létrehozott image mérete

A Gazebo verzió, amit ebben a szoftverben használunk az csak CPU-t használ, ezzel biztosítva, hogy minél több eszközön lehessen használni a programot, még ha a teljesítmény kicsit gyengül is miatta. Természetesen minél jobb hardware-el dolgozunk, annál jobb lesz a program teljesítménye is, de alapvető tesztelésem során arra jutottam, hogy egy belépőszintű, munkára használatos laptop elegendő lehet a kitűzött célok elérésére.

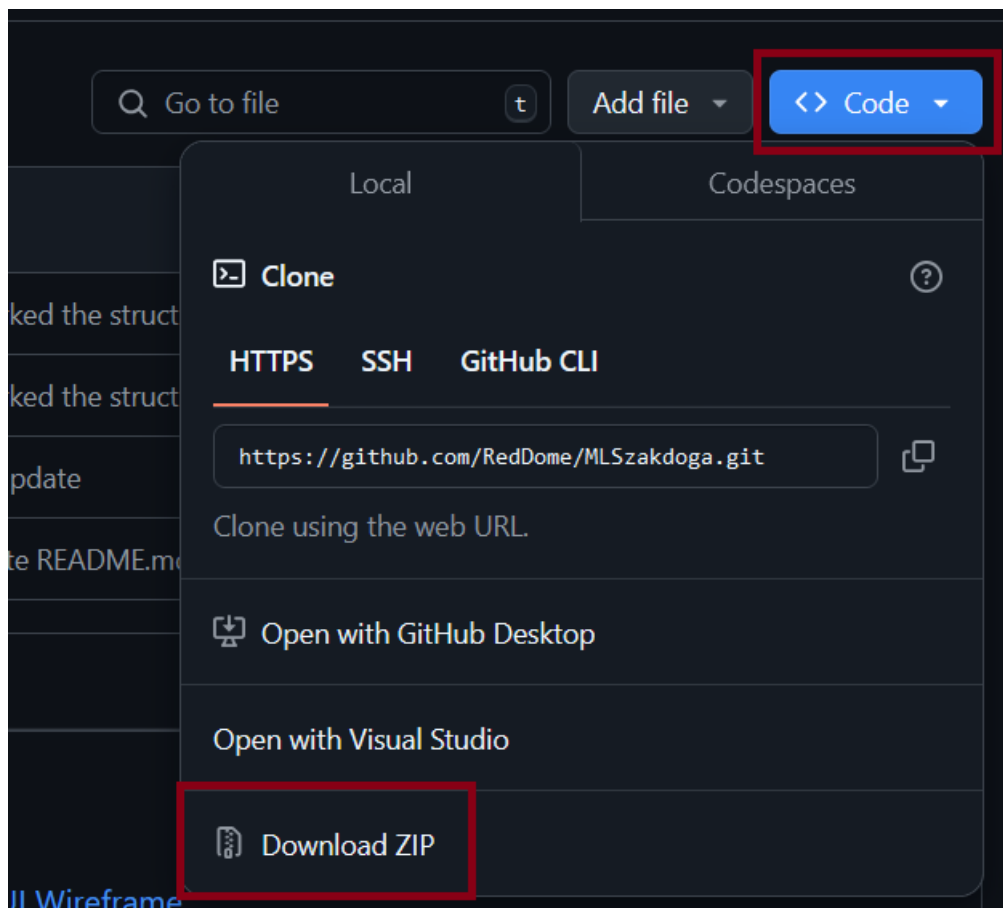


### 2.1.2. Program beszerzése

A programot az általam használt verziókezelőből lehet letölteni, a Github-ról. A következő linkről telepíthető a program:

<https://github.com/RedDome/MLSzakdogas.git>

A linkre rákattintva bekerülünk az egész program könyvtárába, ebben minden megtalálható: wiki, képek, videók, fejlesztői környezet. Nekünk ezután a jobb oldalon lévő kék színű code gombra kell rákattintanunk, ott a legalján lesz egy lehetőség, ahol az lesz kiírva, hogy „Download ZIP”, erre rákattintva letöltjük a programot.



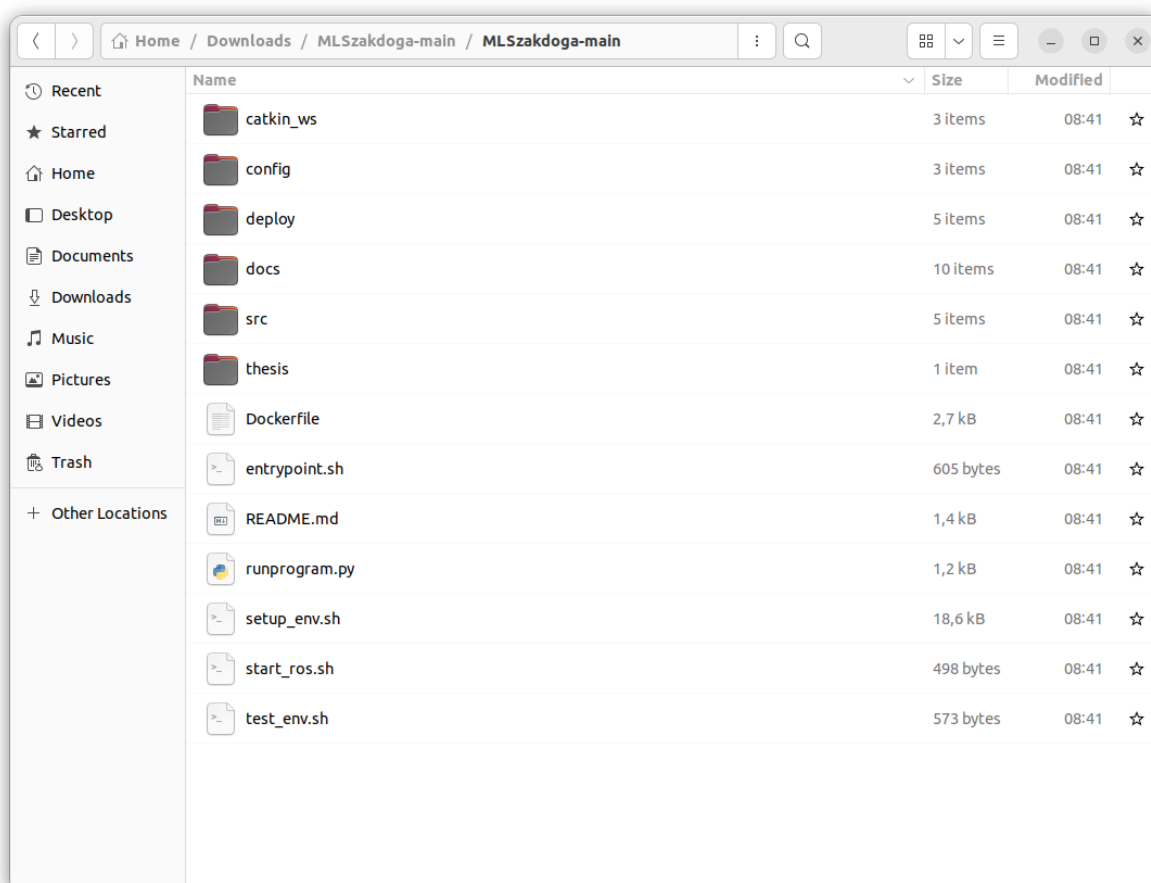
2. ábra: Github letöltési útmutató

## 2.2. Telepítés

Ebben a fejezetben bemutatom, hogy a felhasználó hogyan tudja kicsomagolni és telepíteni a letöltött zip projekt fájlt. Ezen kívül bemutatom a Docker telepítését és konfigurálását, amely elengedhetetlen része a programnak.

### 2.2.1. Program kicsomagolása

Miután beszereztük a programot, azután a tömörített fájlt helyezzük át egy általunk meghatározott könyvtárba. Miután ez megtörtént a következő lépés, amit tennünk kell, hogy kicsomagoljuk a könyvtárat. Ezt egy arra alkalmas szoftver segítségével fogjuk tudni megtenni, mint például a **WinRar** [6] vagy a **7Zip** [7]. Az általunk választott tömörítővel a kitömörítés ide opció használatával kicsomagoljuk az MLSzakdoga-main.zip fájlt. A kicsomagolt mappában fogjuk megtalálni a programot, beleértve a felhasználói és fejlesztői környezetet is. A két környezet abban fog különbözni, hogy másik Dockerfile-t fogunk használni, ami szabályozza, mit fogunk látni a programból.



3. ábra: A kicsomagolt MLSzakdoga mappa

### 2.2.2. Docker telepítése

A Dockerfile egy olyan fájl a Docker szoftverhez, amely képes arra, hogy virtuális környezetet telepítsünk, ezzel segítve a felhasználónak, hogy ne kelljen csomagokat telepítenie, hanem csak ennek telepítésével egy teljesen működő alkalmazást kap.

A **Docker Linux** rendszerben való futáshoz egy **Ubuntu** rendszerre lesz szükségünk, abból is a 24.10, 24.04, 22.04, 20.04-es rendszer ajánlott a Docker problémamentes telepítéséhez (Én alapvetően 22.04-es rendszert használtam, szóval azt ajánlom). Az első lépés a letöltéshez a Docker hivatalos GPG (Digitális kulcs, amely biztosítja az adatok bizalmasságát és hitelességét) kulcsának hozzáadása:

```
# Docker hivatalos GPG kulcsának hozzáadása:
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o
/etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

A következő lépés a **Docker** adattár hozzáadása a forrásainkhoz, hogy le tudjuk tölteni a megfelelő **Docker** csomagokat:

```
# Docker adattár hozzáadása a forrásainkhoz:
echo \
"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc]
https://download.docker.com/linux/ubuntu \
${. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}"} stable" |
\
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

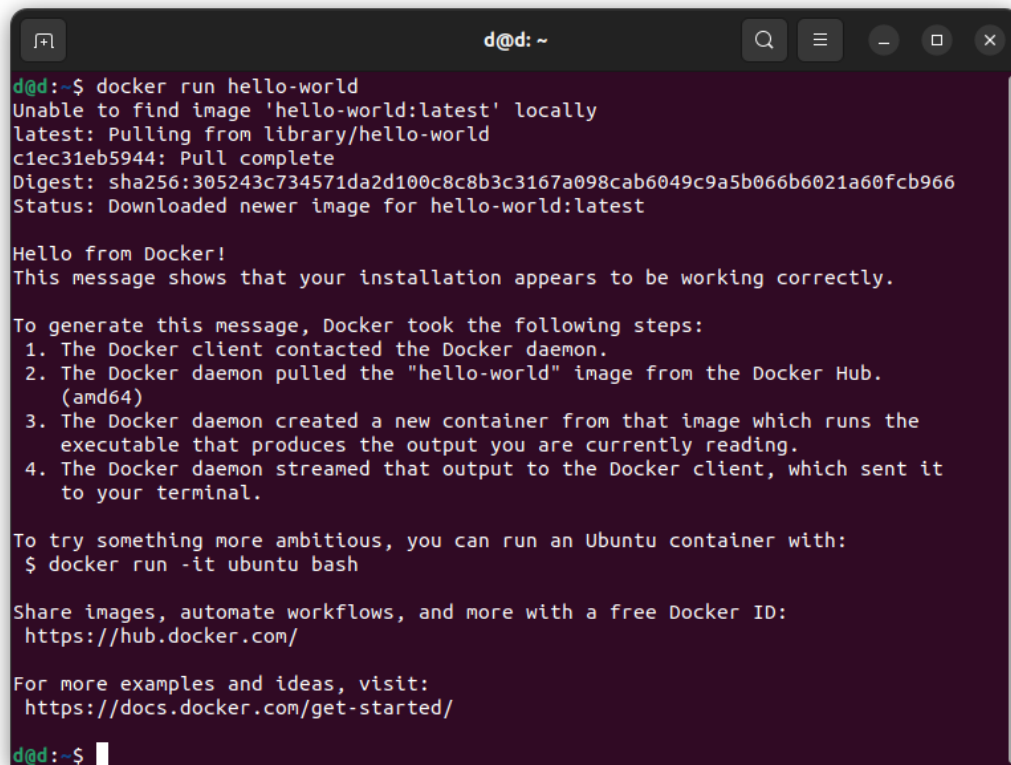
Azutáni lépés a hivatalos csomagok letöltése lesz, ez a lépés több percig is eltarthat:

```
# Docker telepítése (több percig is eltarthat):
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-
compose-plugin
```

Utolsó lépésként, ha mindent jól csináltunk a telepítéssel, teszteljük le a **Docker** környezetet a következő paranccsal:

```
# Docker tesztelése/ellenőrzése:
```

```
sudo docker run hello-world
```



```
d@d:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
c1ec31eb5944: Pull complete
Digest: sha256:305243c734571da2d100c8c8b3c3167a098cab6049c9a5b066b6021a60fcb966
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
   (amd64)
3. The Docker daemon created a new container from that image which runs the
   executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it
   to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/

d@d:~$
```

4. ábra: Docker Hello-World image futása terminálban

Miután a Docker telepítése sikeresen megtörtént, utána még egy utolsó lépést kell tennünk, hogy tudjuk használni a programmal a Docker szoftvert:

```
# User hozzáadása a docker csoporthoz
```

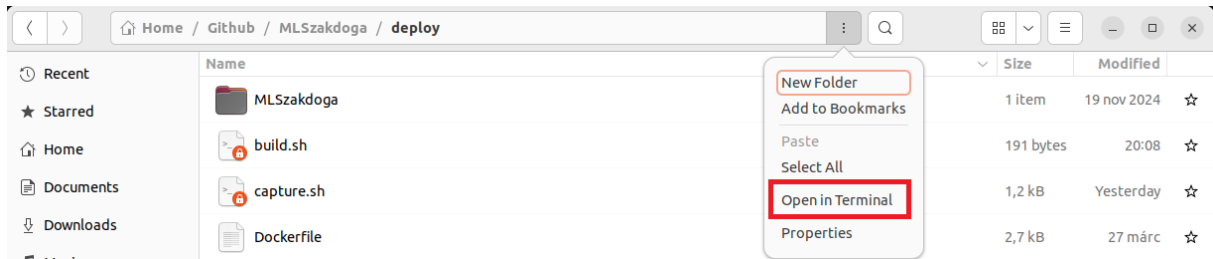
```
sudo groupadd docker
```

```
sudo usermod -aG docker $USER
```

Ez biztosítja, hogy a Docker parancsok működni fognak nem adminisztrátori módban is. Miután végeztünk ezzel a lépéssel, indítsuk újra a gépünket!

### 2.2.3. A program telepítése

Miután kicsomagoltunk és beléptünk a MLSzakdoga-main mappába, ezután menjünk bele a deploy nevezetű mappába, ugyanis ebben találhatóak a felhasználói környezethez használatos fájlok. Ezt a mappát nyissuk meg terminálban.



5. ábra: Mappa Terminálban való megnyitása Ubuntu

Amint megnyitottuk akkor pedig a `./build.sh` parancs futtatásával tudjuk telepíteni a programot. Elsőre nagyon sok minden fog kiíródni a képernyőre és elsőre ijesztő lehet olyan ember számára, aki még nem foglalkozott Dockerimage-l korábban. A programot csak hagyjuk telepíteni, nagyjából 10-20 percet vesz igénybe. Ha a képen látható kódsor megegyezik azzal, amit a saját képernyőnkön látunk, akkor sikeres volt a telepítés. Az első telepítés után már nem is ajánlott többször telepíteni (ha nem módosítunk a programon valamit, akkor újra kell amikor módosítunk a konfigurációs beállításokon), de ha mégis ezt tenné a felhasználó, akkor pillanatok alatt fel fog újra telepedni.

```

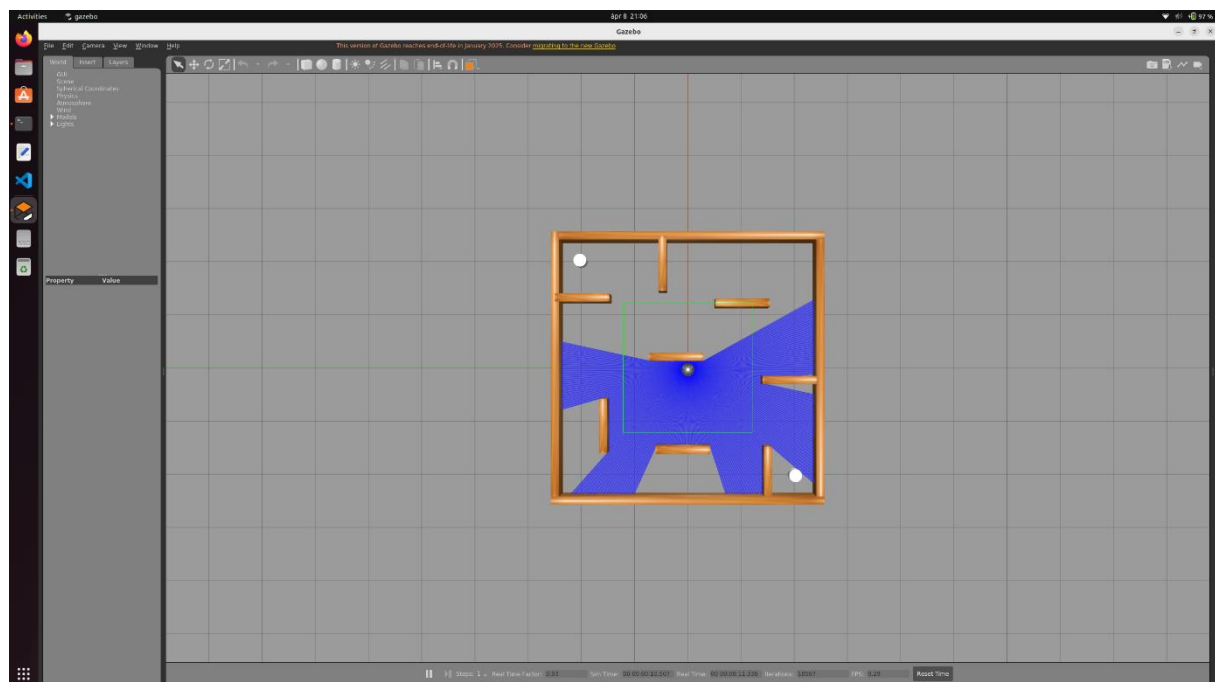
d@d:~/Github/MLSzakdoga/deploy$ ./build.sh
[+] Building 583.6s (31/31) FINISHED                                docker:default
=> [internal] load build definition from Dockerfile                  0.0s
=> => transferring dockerfile: 2.78kB                                0.0s
=> [internal] load metadata for docker.io/osrf/ros:noetic-desktop-full 1.4s
=> [internal] load .dockerignore                                    0.0s
=> => transferring context: 2B                                         0.0s
=> [internal] load build context                                    0.6s
=> => transferring context: 28.13MB                                    0.5s
=> [ 1/26] FROM docker.io/osrf/ros:noetic-desktop-full@sha256:cd6558dc6098142eda23eada4bf975e44a027b0628ba958a2005473f6d2 0.6s
=> => resolve docker.io/osrf/ros:noetic-desktop-full@sha256:cd6558dc6098142eda23eada4bf975e44a027b0628ba958a2005473f6d2e2 0.0s
=> => sha256:cd6558dc6098142eda23eada4bf975e44a027b0628ba958a2005473f6d2e2b7c 3.06kB / 3.06kB 0.0s
=> => sha256:d566acf120f33587990287101940bd94474cf24f184d627caba7858efdfb37a4 6.97kB / 6.97kB 0.0s
=> [ 2/26] RUN apt-get update && apt-get install -y python3-pip python3-tk ros-noetic-catkin python3-cat 18.3s
=> [ 3/26] RUN ln -sf /usr/bin/python3 /usr/bin/python              0.3s
=> [ 4/26] RUN git config --global http.postBuffer 104857600        0.3s
=> [ 5/26] RUN git config --global http.lowSpeedLimit 0             0.3s
=> [ 6/26] RUN git config --global http.lowSpeedTime 999           0.3s
=> [ 7/26] RUN echo 'python --version'                              0.3s
=> [ 8/26] RUN pip3 install --upgrade pip                           3.0s
=> [ 9/26] RUN pip3 install --upgrade setuptools                    2.9s
=> [10/26] RUN pip3 install --default-timeout=600 stable-baselines3[extra] 354.1s
=> [11/26] RUN pip3 install --default-timeout=600 gym                4.5s
=> [12/26] RUN pip3 install --default-timeout=600 tensorflow         106.7s
=> [13/26] RUN pip3 install --default-timeout=600 loguru             1.5s
=> [14/26] RUN if [ -d "/workspaces/MLSzakdoga/catkin_ws/src" ]; then rm -Rf /workspaces/MLSzakdoga/catkin_ws/src; fi 0.3s
=> [15/26] RUN mkdir -p /workspaces/MLSzakdoga/catkin_ws/src        0.4s
=> [16/26] WORKDIR /workspaces/MLSzakdoga/catkin_ws/src             0.3s
=> [17/26] RUN git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git && git clone -b noetic-d 11.6s
=> [18/26] RUN apt-get update && rosdep update && rosdep install --from-paths - --ignore-src -r -y 30.3s
=> [19/26] RUN /bin/bash -c "source /opt/ros/noetic/setup.bash && cd /workspaces/MLSzakdoga/catkin_ws && catkin_make" 16.7s
=> [20/26] COPY ./setup_env.sh /workspaces/MLSzakdoga/setup_env.sh 0.1s
=> [21/26] RUN chmod +x /workspaces/MLSzakdoga/setup_env.sh        0.3s
=> [22/26] RUN echo "source /opt/ros/noetic/setup.bash" >> /root/.bashrc && echo "source /workspaces/MLSzakdoga/catki 0.3s
=> [23/26] COPY entrypoint.sh /entrypoint.sh                       0.1s
=> [24/26] RUN chmod +x /entrypoint.sh                              0.3s
=> [25/26] COPY ./ /workspaces/MLSzakdoga/                          1.1s
=> [26/26] WORKDIR /workspaces/MLSzakdoga/                          0.1s
=> exporting to image                                               26.8s
=> => exporting layers                                              26.8s
=> => writing image sha256:7d74a84cb767f7319a9ee471d149bf7dd08a7be0e9f0da3ee36d443bbb555521 0.0s
=> => naming to docker.io/library/mlszakdoga_deploy                0.0s

View build details: docker-desktop://dashboard/build/default/default/x3lk4nngdidktokhpwelnv3ki
Build completed successfully. Image created as mlszakdoga_deploy.

```

6. ábra: Sikeres `./build.sh` futtatás

## 2.3. Indítás



7. ábra: Elindított program teljes képernyőn

A program elindításához lépünk be a deploy mappába a MLSzakdogában belül (Ha a telepítés után olvassuk ezt a részt, akkor már valószínűleg benne is vagyunk a mappában!). A program a következő paranccsal indítható:

```
# Program elindítása:  
./run.sh paraméterek
```

Miután elindítottuk a programot a megfelelő paraméterekkel, a felhasználónak várnia kell 15 másodpercet mire elindul a Gazebo program és utána még egy extra 10 másodpercet mire a funkció is elindul utána, és már láthatjuk is a szoftveren belül, hogy működik.

### **2.3.1. A program funkciójának beállítása**

Amikor paraméterek nélkül indítjuk el a programot, akkor nem működni a program és egy „Rossz futtatás! Megfelelő futattás: ./run.sh Learn / ./run.sh Continue / ./run.sh SaveData” hibát fog dobni a program. Mint ahogyan a hiba is írja, meg kell adni paraméterként milyen funkciót akarunk használni, itt leírom melyik funkció mit takar, de ezeket egy későbbi fejezetekben fogom jobban kifejteni:

Learn – Új modell tanítása

Continue – Tanítás folytatása

SaveData – Tanítási adatok kiírása

Ezen kívül a felhasználó még egy paramétert megadhat: A WITHLOG paramétert a funkció után, ennek az lesz az eredménye, hogy a Docker úgy fog elindulni, hogy kiír nekünk információkat, például látni fogjuk tanítás közben, hogy hol tart a program.

Például, ha a felhasználó szeretné elindítani a Learn funkciót logolással, akkor azt így teheti meg:

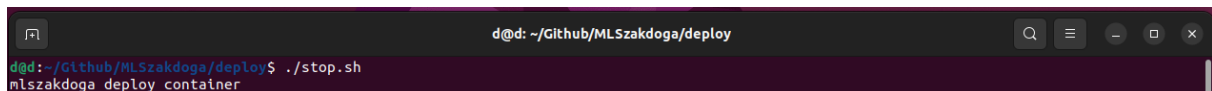
```
# Learn elindítása logolással:  
./run.sh Learn WITHLOG
```

### **2.3.2. Program leállítása**

A program mivel csak egy docker konténer, amely fut, ezért a leállítása nagyon egyszerű, a következő paranccsal lehet ezt megtenni:

```
# Program leállítása:  
./stop.sh
```

Arra kell figyelni a leállításnál, hogy pillanatokon belül ki fog lépni a szoftverből, ezért mindenképpen csak azután kapcsoljuk ki a programot, miután végeztünk a teendőkkel! Ha rendeltetésszerűen használjuk a programot, akkor magától ki fog lépni a funkció végén.



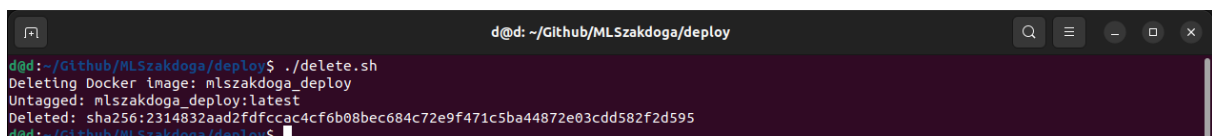
8. ábra: Sikeres leállítása a programnak

### 2.3.3. Program törlése

A program törlése olyan egyszerű, mint a leállítása:

```
# Program törlése:  
./delete.sh
```

A program ezután a telepítés funkcióval újra letölthető, viszont megint megugrik majd a várakozási idő, mivel újra kell telepítenie minden csomagot a Dockernek.



9. ábra: Sikeres törlése a programnak

## 2.4. A program funkciói

A következő fejezetben szeretném részletesen bemutatni, hogy a felhasználó milyen funkciókat fog tudni használni a programban és azon kívül.

### 2.4.1. Új modell tanítása

Ez a legfontosabb funkciója az egész programnak. A funkciónak az lesz a lényege, hogy a Gazebo-ban létrejött robotnak megmondjuk, hogy hova kell mozognia, és utána jutalmat adunk neki, az alapján ahogyan teljesített, és ezt addig csináljuk, amíg beállítottuk.



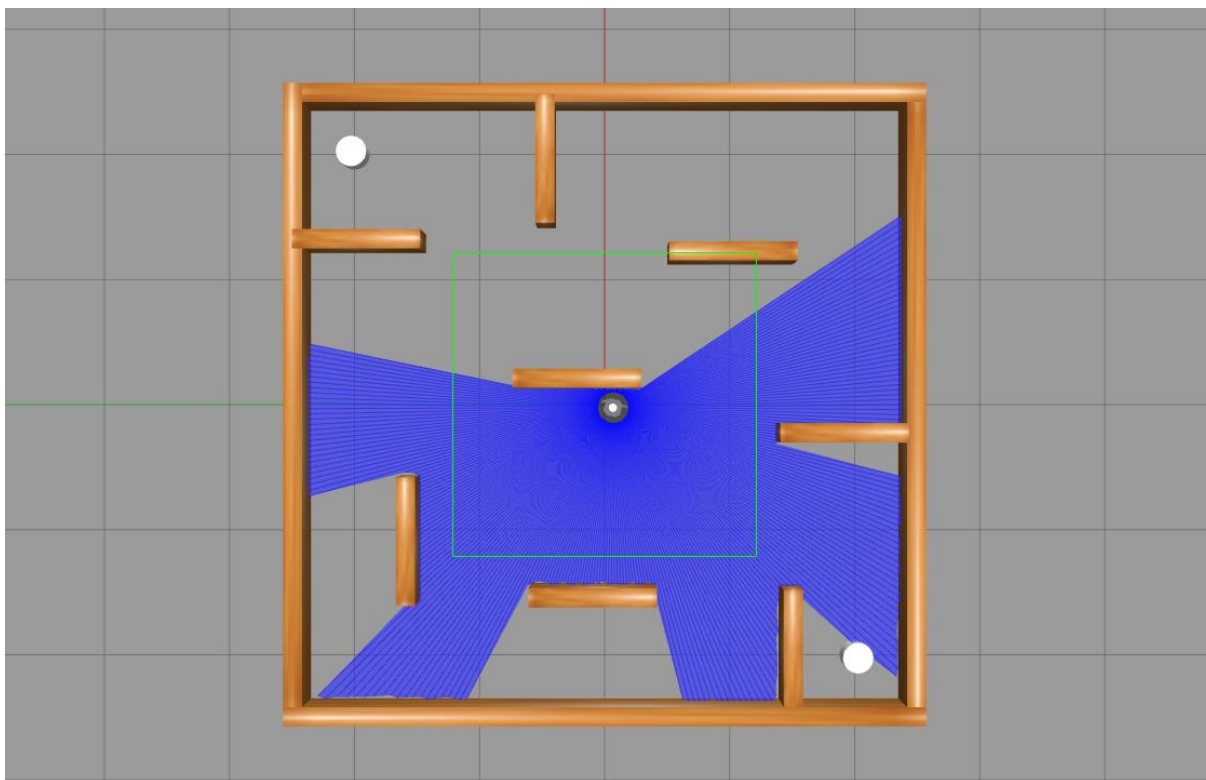
Amikor elindítjuk ezt a funkciót, akkor először a program a *resources* mappában létre fogja hozni a megfelelő mappákat, amelyekben majd a később létrejövő adatokat fogja tárolni. Ezután a létrehozza a robotnak a logikáját és a feladatát, amit át fog neki adni. Ekkor kezd el mozogni a Gazebo-ban belül a robot, ezt a felhasználó valós időben nézheti, nyomon követheti, hogyan tart a robot.

Ha az alapértelmezett beállításokat fogjuk használni, akkor 1000 lépés után menteni fog a program. Ilyenkor a *resources/models* mappában létre fog jönni egy 1000.zip nevű modell, ez minden 1000.-ik lépés után növekedni fog, amíg el nem érjük a végső lépésszámot. A *resources/log* mappában pedig a tanítással kapcsolatos nyers adatok fognak lementődni, amelyet a Tanítási adatok kiírása funkcióban fogunk majd tudni felhasználni.

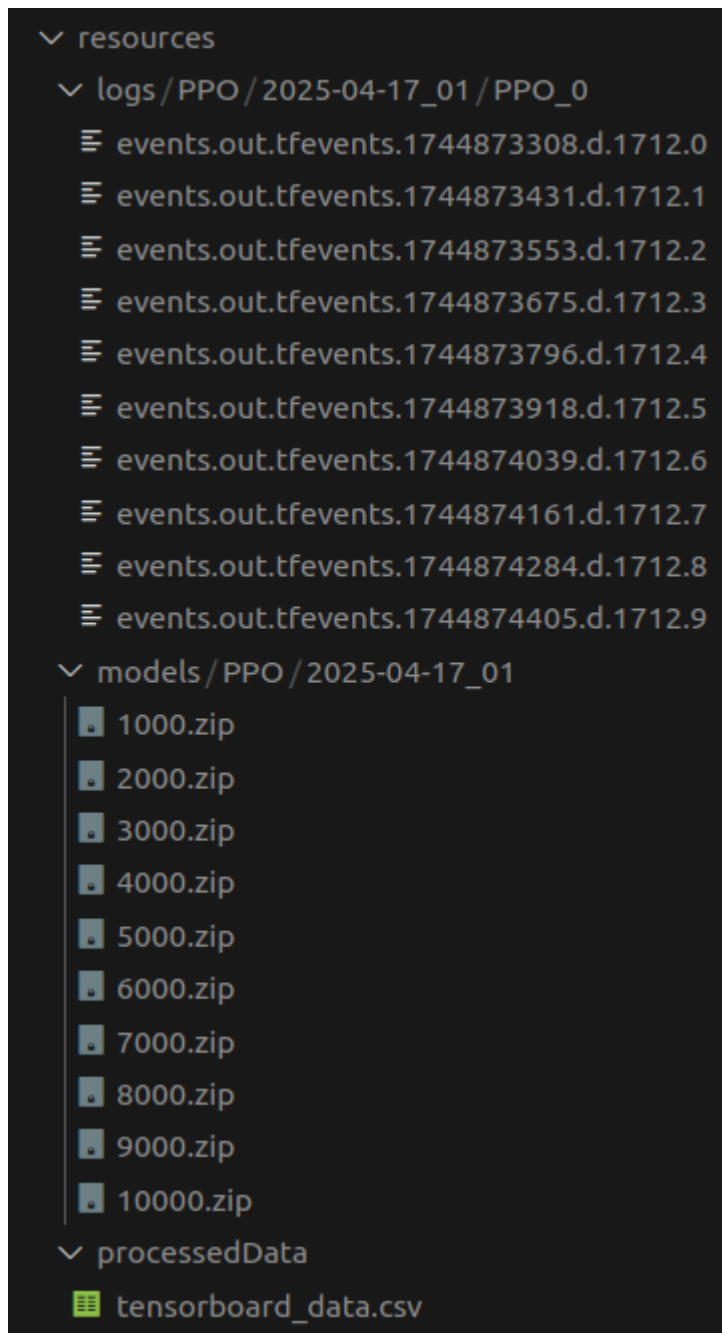
A funkcióhoz megadott konfigurációs fájlban lévő beállítások már garantálják a megfelelő működést, de egy későbbi fejezetben megmutatom melyik beállítás mit jelent, ha a felhasználó szeretné ezeket módosítani.

Amikor a program elérte a megfelelő lépésszámot, kilép a program.

A program végzése közben a felhasználónak van lehetősége felvenni a képernyőjét, ezt az Ubuntu rendszeren a vezérlőpultban találjuk meg. Utána pedig kiválasztjuk a képernyőfelvétel funkciót, és kijelöljük mekkora képernyőt lehessen felvenni, és utána a *Screencasts* mappában fogjuk tudni megtalálni a felvett videókat.



10. ábra: Robotunk tanulás közben



11. ábra: Egy 10000 lépéses tanítás közben létrejött adatok

#### 2.4.2. Tanítás folytatása

Habár ez a funkció nagyon hasonlít az Új modell tanítása funkcióra, teljesen más célt szolgál. Tudjuk, hogy egy ilyen tanítás órákig is eltarthat, és nem minden felhasználó tudja órákig ott hagyni a számítógépét, ezért a funkciónak az lenne a lényege, hogy egy lementett modellt, ott tudjunk folytatni, ahol „abbahagytuk”.

**FONTOS!** Amikor ezt a funkciót először szeretnénk futtatni, akkor a *config* mappában lévő *CONTINUE\_DEFAULT\_CONFIG.yaml*-ben a **ModelPath** értéket meg kell adni, különben nem fog működni a program! Erről és a konfigurációk felépítéséről egy későbbi fejezetben fogok írni. Ezt a funkciót csak akkor használjuk, ha már legalább egyszer használtuk az Új modell tanítása funkciót.

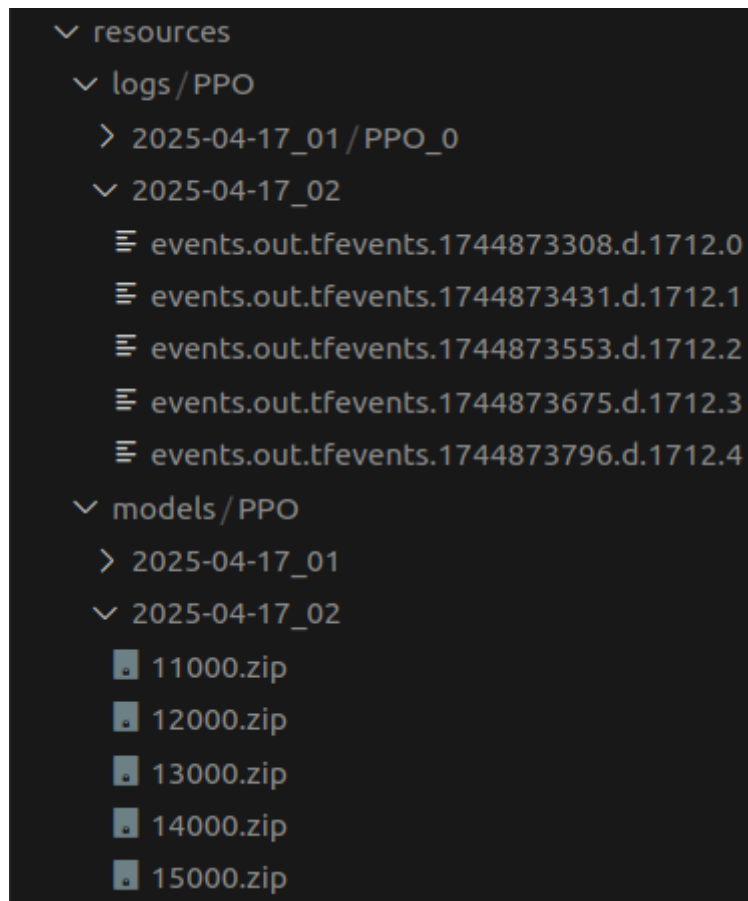
Ez a folytatás úgy történik, hogy a .zip végződésű lementett modellek, amelyek a tanítás közben jöttek létre, azokat be tudjuk olvasni, és a Stable Baselines segítségével, be tudjuk onnan újra olvasni az adatait, ahol abbahagyta korábban és csinál nekünk egy olyan szimulációt, amely ugyanúgy működött, mint ahol az előzőt abbahagytuk, viszont már okosabb lesz, ezáltal rögtön látható lesz a fejlődése.

A robot ezután megint elkezd mozogni a Gazebo alkalmazáson belül, és ha az alapértelmezett beállítást használjuk, akkor minden 1000. lépés után le fog menteni egy kész modellt a *resources* mappába, miközben a logjait, pedig a *log* mappába fogja létrehozni, ahonnan később fel tudjuk dolgozni az adatait.

A megadott lépések elvégzése után a program ki fog lépni.

A program végzése közben a felhasználónak van lehetősége felvenni a képernyőjét, ezt az Ubuntu rendszeren a vezérlőpultban találjuk meg. Utána pedig kiválasztjuk a képernyőfelvétel funkciót, és kijelöljük mekkora képernyőt lehessen felvenni, és utána a *Screencasts* mappában fogjuk tudni megtalálni a felvett videókat.

A többi funkcióhoz hasonlóan, itt is a megfelelő konfigurációs fájl módosításával lehet beállítani a funkcióit, ezekről egy későbbi fejezetben fogok részletesebben írni.



12. ábra: Egy 5000 lépéses folytatás közben létrejött adatok

### 2.4.3. Tanítási adatok kiírása

Ez a funkció nagyon különböző a többihez képest, ez lesz az egyetlen funkció, amely nem használja a Gazebo programot. Ez a funkció nem is fogja elindítani a szimulátort.

**FONTOS!** Amikor ezt a funkciót először szeretnénk futtatni, akkor a *config* mappában lévő *SAVE\_DATA\_DEFAULT\_CONFIG.yaml*-ben a **LogFolder** értéket meg kell adni, különben nem fog működni a program! Erről és a konfigurációs fájlok felépítéséről egy későbbi fejezetben fogok írni. Ezt a funkciót csak akkor használjuk, ha már legalább egyszer használtuk az Új modell tanítása funkciót.

Ez a program az alapból Tensorboard által kreált adatokkal foglalkozik. Amikor a felhasználó tanítja a programot, akkor minden egyes alkalommal amikor lement egy modellt, akkor készülni fog egy log fájl is, amelynek a szerepe az, hogy minden egyes tulajdonságváltozást feljegyez, később ebből tud egy böngésző alapú megoldást adni nekünk, amelyben látjuk, hogy a robotunk hogyan fejlődött az idő múlásával.

Amikor elindítjuk ezt a funkciót, akkor a megadott mappában lévő nyers log fájlokat összegyűjti a program, és ki fogja őket válogatni lépésszám, érték, érték neve alapján. Ami után ezzel végzett a program, utána az érték nevek alapján újra formálja a létrejött adatokat, hogy ezzel egy modern adatbázis kezelő alkalmazás is könnyedén meg tudja jeleníteni az adatainkat. Ezt az új létrehozott fájlt a *processedData* mappában fogjuk találni a *resources*-on belül.

	B	C	D	E	F	G	H
1 tag	10	20	30	40	50	60	70
2 time/fps	8	8	8	8	8	8	8
3 train/approx_kl	0.002578902291134	0.0025868476368486	0.0005070447805337	0.002153402660042	0.0012554347049444	0.0003853440284729	
4 train/clip_fraction	0	0	0	0	0	0	
5 train/clip_range	0.200000002980232	0.200000002980232	0.200000002980232	0.200000002980232	0.200000002980232	0.200000002980232	
6 train/entropy_loss	-1.09768342971802	-1.09365558624268	-1.08791089057922	-1.08175075054169	-1.06855547428131	-1.06415927410126	
7 train/explained_variance	2.92062759399414E-06	-4.76837158203125E-06	-1.08480453491211E-05	-1.78813934326172E-06	-7.98702239990234E-06	-1.64508819580078E-05	
8 train/learning_rate	0.0003000000142492	0.0003000000142492	0.0003000000142492	0.0003000000142492	0.0003000000142492	0.0003000000142492	
9 train/loss	4365.859375	4371.443359375	4376.73095703125	4380.875	4385.54248046875	4390.97509765625	
10 train/policy_gradient_loss	-0.006736968178302	-0.0169897321611642	0.0053358376026153	-0.0112779578194022	-0.0044295038096606	4.77462162962183E-05	
11 train/value_loss	8783.43359375	8791.5908203125	8799.1162109375	8804.802734375	8811.734375	8820.486328125	

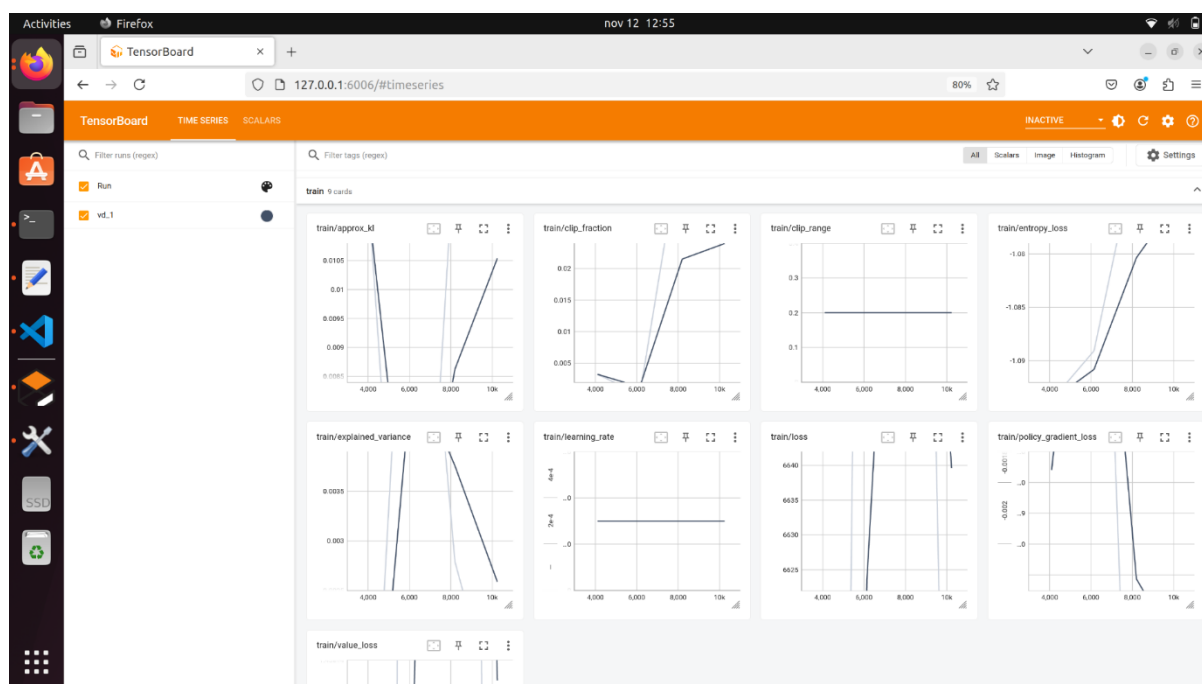
13. ábra: Kimentett adatok kinézete egy .csv fájlban beolvasva egy arra alkalmas szoftverben

Ezen felül még van egy másik megoldás is, amit tehet a felhasználó, az egy online Tensorboard felület, amely a nyers log fájlokból, különböző gráfokat létrehozni nekünk, tökéletes lehet azoknak, akik valamilyen kutatómunka miatt használják a programot.

A következő módon lehet elindítani ezt az online felületet:

```
# Tensorboard elindítása
tensorboard --logdir=resources/logs/LOG_HELYE
```

Ezután a képernyőn megjelenő porton megtekinthető a felület.



14. ábra: Online Tensorboard felület kinézete

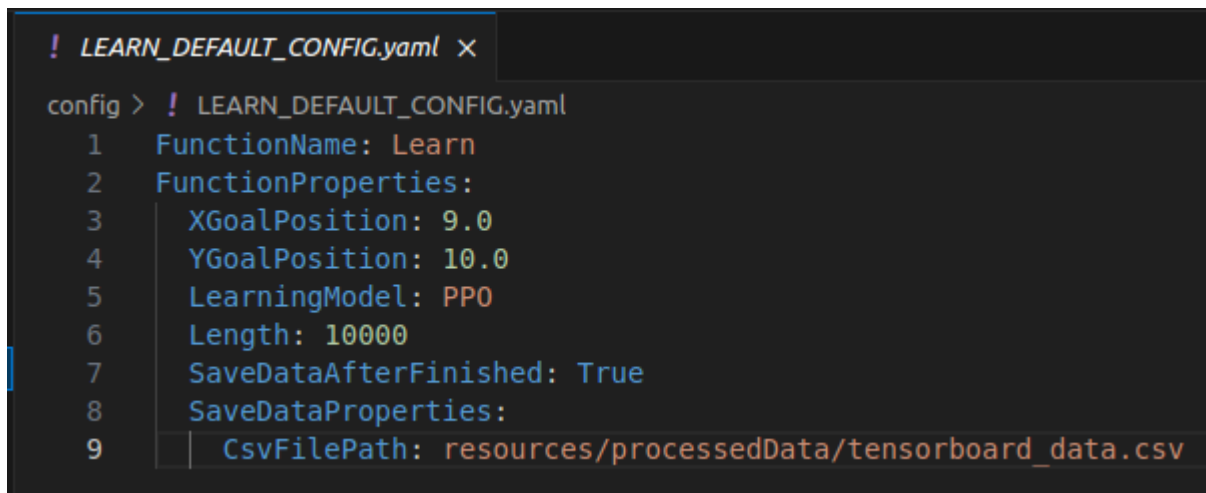
## 2.5. Konfig és a program funkcióinak konfigfájljainak felépítése

A legtöbb modern programnak szép felhasználó felülete van, amely segíti a felhasználónak a program használatát, viszont vannak olyan programok, amelyek a hordozhatóságra és a kicsi méretre pályáznak, azokban a programokban nem mindenhol biztos, hogy megtalálható lesz egy felhasználó felület. Helyette jobban preferálják a konfig alapú megoldásokat, amelyek könnyebb személyre szabhatóságot biztosítanak.

A program konfig fájljainknak .yaml végződése van, amely az egyik legnépszerűbb kiterjesztés az ilyen megvalósítás szempontjából. Úgy kell értelmezni az ilyen fájlokat, mint egy alkalmazásnál egy beállítások menüpontot, csak itt egy fájlban vannak, amelyet maga a program fog később feldolgozni. Minél egyszerűbben írtam meg őket, hogy könnyen lehessen értelmezni, mit is szeretnénk átírni. A következő alfejezetekben szeretném bemutatni az összes alfunkciónak a konfig fájljait, amelyet a programban tudunk használni, és bemutatni, hogy melyik soruk mit jelent.

**FONTOS!** Ha bármelyik konfig fájlban változtatunk, utána a programot újra kell telepíteni, futtatás előtt, különben a program nem fogja felismerni az új konfig fájlt.

### 2.5.1. Új modell tanítása konfigurációs fájljának felépítése



```
! LEARN_DEFAULT_CONFIG.yaml x
config > ! LEARN_DEFAULT_CONFIG.yaml
1  FunctionName: Learn
2  FunctionProperties:
3    XGoalPosition: 9.0
4    YGoalPosition: 10.0
5    LearningModel: PPO
6    Length: 10000
7    SaveDataAfterFinished: True
8    SaveDataProperties:
9      CsvFilePath: resources/processedData/tensorboard_data.csv
```

15. ábra: Új modell tanítása konfigurációs fájlja

**FunctionName** – Ebben adjuk meg a funkciónak a nevét, ezt semmiképpen se írjuk át! A learn az új modell tanítása funkciónak a neve

**FunctionProperties** – Az adott funkcióhoz tartó extra beállítások

**XGoalPosition** – X pozíció, amelyet a robotnak el kell érnie a tanítás során

**YGoalPosition** – Y pozíció, amelyet a robotnak el kell érnie a tanítás során

**LearningModel** – Tanítási modell, amellyel tanuljon a program. A jelenleg támogatott modellek:

- A2C
- PPO
- DQN
- SAC
- TD3

Ennek a módosítását csak akkor ajánlom, ha ért hozzá a felhasználó!

**Length** – Tanítás hossza, mennyi ezer lépésig jusson el a robot

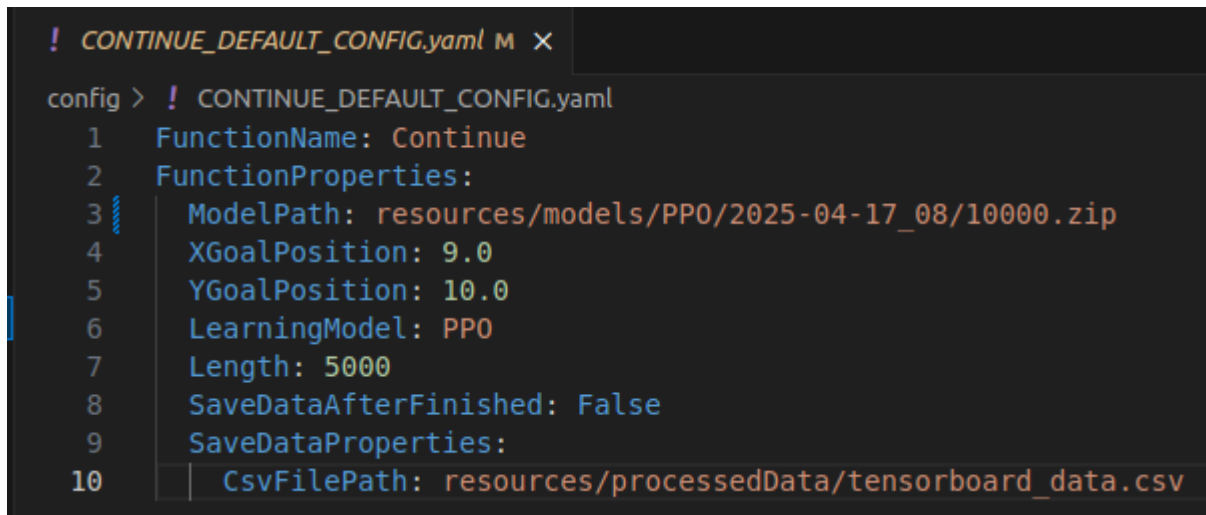
**SaveDataAfterFinished** – Amikor a tanítás végére értünk, akkor a program a tanítási statisztikákat írja-e ki egy fájlba. Ha ez az érték igaz lesz, akkor a Tanítási adatok kiírása funkció fog lefutni



**SaveDataProperties** – Ha le akarjuk menteni az adatokat, akkor itt tudjuk beállítani hozzá a dolgokat

**CsvFilePath** – A .csv végződésű fájl útvonala, ahova ki tudjuk menteni az adatokat

### 2.5.2. Tanítás folytatása konfigurációs fájljának felépítése



```
! CONTINUE_DEFAULT_CONFIG.yaml M X
config > ! CONTINUE_DEFAULT_CONFIG.yaml
1  FunctionName: Continue
2  FunctionProperties:
3    ModelPath: resources/models/PP0/2025-04-17_08/10000.zip
4    XGoalPosition: 9.0
5    YGoalPosition: 10.0
6    LearningModel: PPO
7    Length: 5000
8    SaveDataAfterFinished: False
9    SaveDataProperties:
10   CsvFilePath: resources/processedData/tensorboard_data.csv
```

16. ábra: Tanítás folytatása funkció konfigurációs fájlja

**FunctionName** – Ebben adjuk meg a funkciónak a nevét, ezt semmiképpen se írjuk át! A Continue a Tanítása folytatása funkciónak a neve.

**FunctionProperties** – Az adott funkcióhoz tartó extra beállítások

**ModelPath** – A modell útvonala, amelyet szeretnénk tovább tanítani

**XGoalPosition** – X pozíció, amelyet a robotnak el kell érnie a tanítás során

**YGoalPosition** – Y pozíció, amelyet a robotnak el kell érnie a tanítás során

**LearningModel** – Tanítási modell, amellyel tanuljon a program. A jelenleg támogatott modellek:

- A2C
- PPO
- DQN
- SAC
- TD3

Ennek a módosítását csak akkor ajánlom, ha ért hozzá a felhasználó!

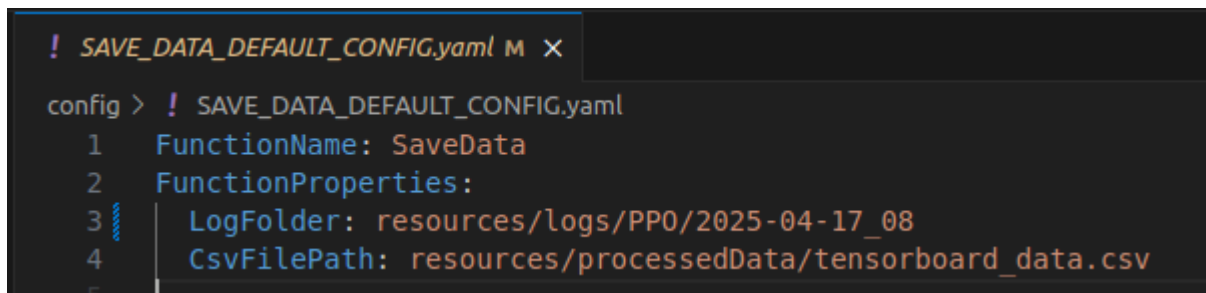
**Length** – Tanítás hossza, mennyi ezer lépésig jusson el a robot

**SaveDataAfterFinished** – Amikor a tanítás végére értünk, akkor a program a tanítási statisztikákat írja-e ki egy fájlba. Ha ez az érték igaz lesz, akkor a Tanítási adatok kiírása funkció fog lefutni

**SaveDataProperties** – Ha le akarjuk menteni az adatokat, akkor itt tudjuk beállítani hozzá a dolgokat

**CsvFilePath** – A csv végződésű fájl útvonala, ahova ki tudjuk menteni az adatokat

### **2.5.3. Tanítási adatok kiírása konfigfájljának felépítése**



```
! SAVE_DATA_DEFAULT_CONFIG.yaml M X
config > ! SAVE_DATA_DEFAULT_CONFIG.yaml
1  FunctionName: SaveData
2  FunctionProperties:
3    LogFolder: resources/logs/PP0/2025-04-17_08
4    CsvFilePath: resources/processedData/tensorboard_data.csv
```

17. ábra: Tanítási adatok kiírása konfigfájlja

Ezt a funkciót csak magában hívjuk meg, amikor a másik programoknál kimentjük a dolgokat, akkor nem fogja ezt a konfig fájlt használni!

**FunctionName** – Ebben adjuk meg a funkciónak a nevét, ezt semmiképpen se írjuk át! A SaveData a Tanítási adatok kiírása funkciónak a neve.

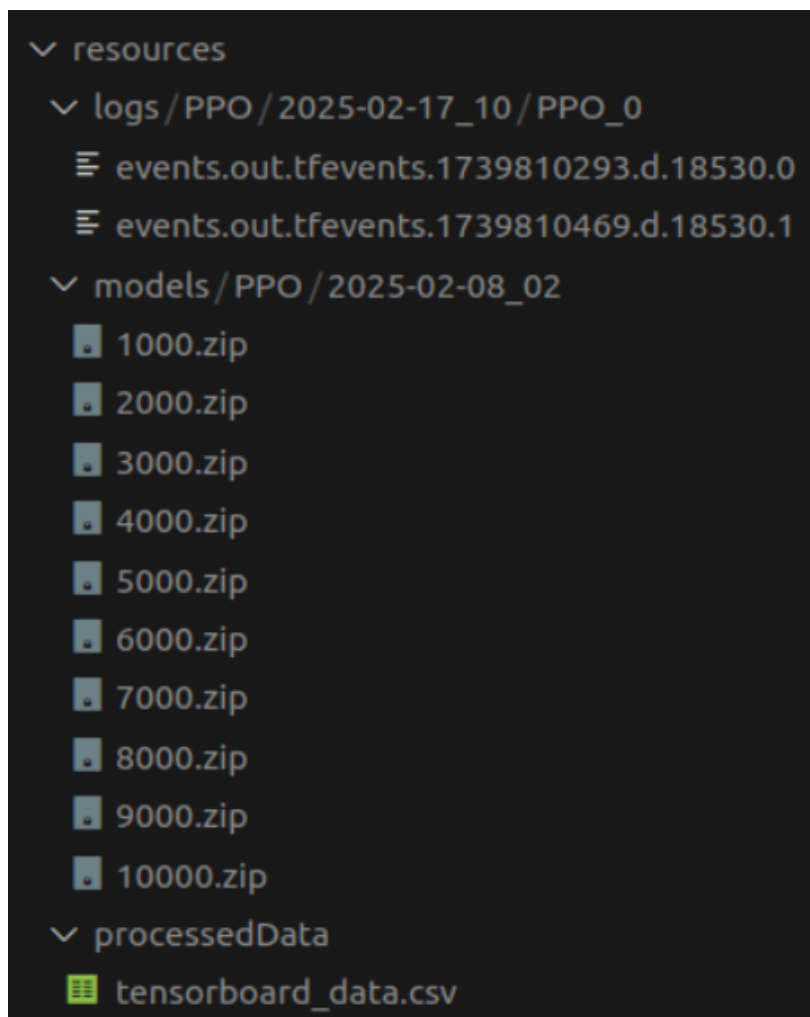
**FunctionProperties** – Az adott funkcióhoz tartó extra beállítások

**LogFolder** – A logmappának az útvonala, ahol a tanítás során a program létrehozta a log fájlokat

**CsvFilePath** – A csv végződésű fájl útvonala, ahova ki tudjuk menteni az adatokat

## **2.6. A létrehozott nyersanyagok és annak kezelése**

A program minden funkciója során valamilyen adat fog létrejönni, ebben a fejezetben szeretném ezeket bemutatni, és hogy miket tudunk velük kezelni.



18. ábra: Resources mappa és annak felépítése

A *resources* mappa, amelyben az összes adatot tároljuk, ami létrejön a program közben, azt 3 fő részre tudjuk felosztani.

Az első a *models* mappa, amelyben láthatjuk, hogy napokra vannak felosztva a mappa nevek, ez arra szolgál, hogy könnyen meg tudjuk találni, hogy melyik nap mit csináltunk. A dátum utáni szám arra szolgál, hogy megmondja, hogy azon a napon hányadik futás volt, így könnyebben megtaláljuk a fájljainkat, akkor is, ha egy nap többször használjuk a programunkat. A mappákon belül találhatóak a lementett modellek .zip formátumban. Ezeket a modelleket tudjuk majd beolvasni a Tanítás folytatása funkcióban.

A második a *logs* mappa, amelyben ugyanolyan mapparendszer található, mint a *models* mappa esetében. Ezekben a mappákban, a tanítás közben létrejött értékváltozásokat tároljuk,

amelyeket utána a Tanítási adatok kiírása funkció folyamán fogunk tudni felhasználni, azért, hogy utána az adatokat vagy meg tudjuk tekinteni a Tensorboard által létrehozott webhelyen, vagy saját gépünkre kimenteni, egy adatbázis fájlban, amivel bármit tehet a felhasználó.

A harmadik a *processedData* mappa lesz, amelyben a Tanítási adatok kiírása funkció során létrejött .csv fájlokat fogunk találni, amelyekben a kiírt adatok találhatóak. A .csv fájlokban az adatokat vesszők választják el egymástól, egy nagyon népszerű adatbázis formátum, amely a legtöbb adatbázis szoftverrel, mint mondjuk az Excellel is tökéletesen működik.

## **2.7. A program logolása és annak értelmezése**

A logolás egy olyan kifejezés, hogy a programban található kódban el vannak rejtve olyan parancsok, ahol szeretnénk, hogy a programunk kiírjon valamilyen információt, lehet ez hiba vagy esetleg csak egy üzenet is. A legtöbbször ezt a konzolunkra tudjuk kiírni, viszont a legtöbb modern programban már egy külön fájlban írják ki ezeket a dolgokat, mivel egy komplexebb program esetén több 1000 sor log is létrejöhet, akár rövid időn belül is.

A mi programunk is egy fájlban fog logolni, a projekt törzsében található lesz az *app.log* nevű fájl, ebben találjuk a program alatt létrehozott összes logot. A felhasználónak itt csak a hibákat fogom megemlíteni, mivel a többit úgy érzem, hogy nem szükséges tudnia a felhasználónak, és ezeket csak a fejlesztői részen fogom jobban kifejteni.

```
2025-04-16 18:49:46.448 | ERROR | config.processconfigfile:processConfigFile:22 - This is an example error! Do this to fix it: .....
```

19. ábra: Kilógolt hiba kinézete

A képen látható a logból kapott hibaüzenetet. A kódban úgy van megoldva, hogy minél érthetőbben leírja, hogy a mi a program hibája, és hogyan lehetne megoldani a hibát, a legtöbb probléma valószínűleg a konfigurációnál fog történni, a nem megfelelő átírása során. A log sor elején a dátumot láthatjuk, ezzel tudjuk igazolni, hogy igen mostanában történt a sikertelen elindulás, ezáltal ezt kell kijavítanom. A következő rész ERROR-t kell, hogy írjon, a logban a legtöbb sor INFO taget fog kapni, viszont ez a felhasználónak nem fontos adatok, hanem inkább a fejlesztőnek hasznos. Utána a log megadja, hogy a melyik mappában, fájlban, függvényben, sorban található a probléma, ez megint nem fontos információ a felhasználónak, de egy fejlesztőnek nagyon hasznos lehet. Az utolsó rész maga az üzenet, amit

a program át akar adni, ez a legfontosabb a felhasználónak, mert itt fogja leírni, hogy mi a probléma, és mi lehet egy esetleges megoldás.

Habár a logolás nem egy olyan dolog, amely a legtöbb felhasználónak fontos lehet, viszont úgy érzem, hogy nagyon fontos lehet mindenkinek, aki szeretné jobban megérteni, mi történik a háttérben.

## 3. Fejlesztői dokumentáció

A fejlesztői dokumentációban megismerhetjük a forráskód felépítését, a függőségeket, amelyek a program működéséhez elengedhetetlenek, illetve a programnak a belső működéseibe belemegy részletesebben a dolgozat, mint például a logolás vagy a konfigurálás. Ezen felül bemutatom a fejlesztői és tesztelési környezet beállítását és helyes használatát, a szoftver követelmény-specifikációját, és a dolgozat verziókezelését.

### 3.1. Forráskód beszerzése

A projekt forráskódját mellékeltem a dolgozathoz, *VÖRÖSDÖME\_QK8IUC\_SZAKDOLGOZAT.zip* néven, de az általam használt verziókezelőből is letölthető az alábbi paranccsal:

1. Nyissuk meg az eszközünkön a terminált
2. Telepítsük fel a GIT csomagot

```
# Git csomag telepítése  
sudo apt update  
sudo apt install git
```

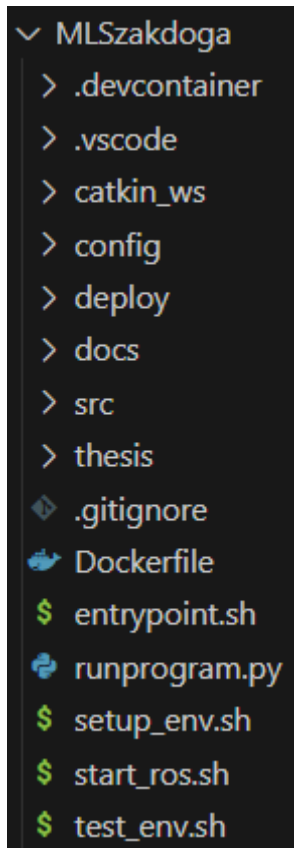
3. Ezzel a paranccsal töltjük le a csomagot:

```
# Git projekt klónolása  
git clone https://github.com/RedDome/MLSzakdoga.git
```

A kapott mappánk a MLSzakdoga mappa lesz, amelyben minden megtalálható a program megfelelő működéséhez. A forráskód beszerzése után a következő fejezetben szeretném bemutatni a forráskód könyvtárszerkezetét.

### 3.2. A forrás könyvtárszerkezete

Amint kicsomagoljuk a MLSzakdoga könyvtárat, akkor a mappában számos almappa mellett találni fogunk fájlokat is, amelyek a fejlesztői környezet megfelelő működéséhez elengedhetetlenek.



20. ábra: MLSzakdoga fájlrendszere

A MLSzakdoga gyökerében lévő fájlok leírása:

- .gitignore: GIT verziókezelőhöz használt fájl, ebben megadom azokat a fájlokat, amiket nem kell verziókövetni, mint például a létrejött adatok.
- Dockerfile: Docker kép létrejöttéért felelős fájl, ebben töltjük be a megfelelő csomagokat és függőségeket, ezáltal egy könnyű és konzisztens környezetet tudunk biztosítani.
- entrypoint.sh: A fájl, amelyet a Dockerfile a kezdőpontjának fog beállítani, vagyis, hogy ez a fájl fusson le miután sikeresen létrehozta a Docker képet, ebben a fájlban megadunk különböző ROS-al és Gazebo-val kapcsolatos beállításokat, amelyek a megfelelő futáshoz kellenek.
- runprogram.py: Innen indul el a program, mind a fejlesztői és felhasználói elindításnál.
- setup\_env.sh: A Gazebo és világa beállítását végzi el. A fájlban létrehozuk a megfelelő könyvtárakat, amelyeket a környezet felállításához használni kell, utána a megfelelő fájlokat forrásoljuk. Ezeket a lépéseket ellenőrizzük is, ezáltal biztosítva, hogy megfelelően létrejöttek. Ezután létrehozuk a Turtlebot 3 robotot, amelynek

tulajdonságokat és logikát adunk. Ezután a világnak az adatait tudjuk megadni, milyen talajon működjön, milyen világítások és akadályok legyenek rajta. Az utolsó lépésként pedig a robotnak megadjuk a megfelelő szenzorokat és lézereket, amelyek biztosítják a mozgását és látását.

- start\_ros.sh: A roscore program elindításáért felelős, azután pedig innen indul el a Gazebo program.
- test\_env.sh: Azt fogjuk ebben a fájlban tesztelni, hogy a megfelelő szenzorok elérhetőek-e, létrejöttek-e a megfelelő módon.

Ezen kívül még megtalálhatóak mappák is a MLSzakdoga-n belül, ezeket is szeretném röviden bemutatni:

- .devcontainer: A Visual Studio Code Dev Container működéséhez szükséges mappa
- .vscode: A Visual Studio Code Dev Container működéséhez szükséges mappa
- catkin\_ws: A környezet felállítása közben létrejött fájlokat tartalmazza, amiket fejlesztői célokból kivezettem a forrásba, ezáltal könnyen követhetőek.
- config: Itt találhatóak az alapértelmezett konfigurációs fájlok, amelyekkel az alkalmazás különböző verziói elindíthatóak.
- deploy: A felhasználói alkalmazás ebben a mappában található, hasonló a felépítése, mint a fejlesztői alkalmazásnak, csak könnyebben használható, így biztosítva a felhasználónak a környezetbarát megoldást.
- docs: Ebben találhatóak különböző képek, videók, dokumentumok, amelyek a programot mutatják be, segítik jobban megérteni a funkcióit.
- src: Ebben találhatóak a fájlok, amelyek a belső programot működtetik.

### **3.3. Függőségek**

A program megfelelő működéséhez különböző külső könyvtárakat használ a program. Ezek a könyvtárak a következők:

- **Stable Baselines 3**: Egy Python könyvtár, amely megerősítéses tanulási algoritmusokat tartalmaz. Célja, hogy egyszerűsítse a tanulási algoritmusok használatát és fejlesztését. Több fajta algoritmus támogat, mint például a PPO (Proximal Policy Optimization), A2C (Advantage Actor Critic) vagy a DQN (Deep Q Learning).



- **ROS (Robot Operating System)**: Nyílt forráskódú keretrendszer, amelyet olyan alkalmazások fejlesztésére használnak, ahol robotokat alkalmazunk. Habár a nevéből arra gondolnánk, hogy ez egy operációs rendszer, ez valójában eszközök és könyvtárak gyűjteménye, amelyek a robotok programozását segítik.
- **Gazebo**: Fejlett robot szimulációs szoftver, feladata a robotok valós környezetekben való szimulálása. A Gazebo szoros integrációval működik a ROS-al, gyakran használják együtt őket a robotikai fejlesztéseknél.
- **Tensorflow**: A Google Brain Team által fejlesztett könyvtár, amely számos gépi tanulás és mesterséges intelligenciával kapcsolatos feladathoz használható. Az egyik legnépszerűbb mélytanulási keretrendszer, amely elérhető manapság.
- **Loguru**: A Python alapvető logolásának újragondolása, a meglévő funkciók mellett még többet tesz bele a készítő és egyszerűbb használatával a projektben könnyebben lehet konzisztens logolást biztosítani.

### **3.3.1. A függőségek beszerzése**

Ezen függőségek alapvetően telepítve vannak a forráskód megfelelő futtatása során, és a tesztelésekkel pedig ellenőrizve van, hogy minden a megfelelő módon működik-e, ezért a felhasználónak a beszerzéssel kapcsolatban nincsen semmilyen teendője.

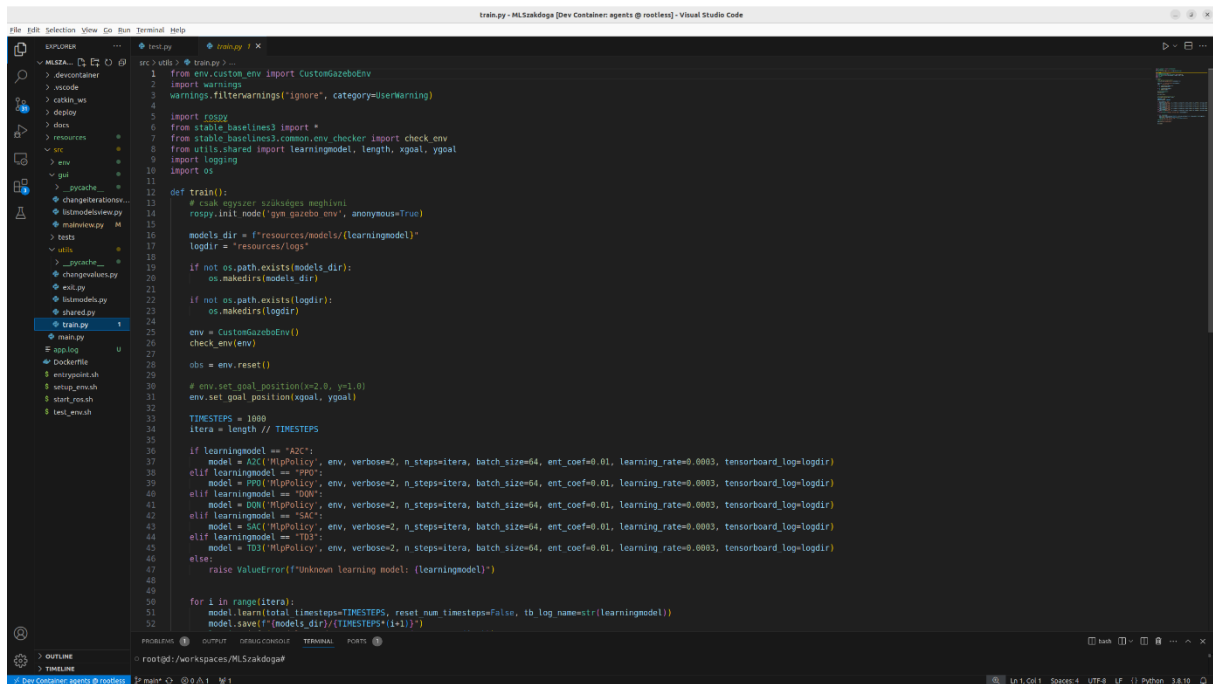
### **3.3.2. A függőségek frissítése**

A Gazebo és a ROS szoros integrációja miatt, nagyon nehéz a függőségek frissítése anélkül, hogy el ne rontsuk a másik függőségek funkcionalitását. Ezért habár van frissítési lehetőség, nem ajánlott a megfelelő működés érdekében.

## **3.4. A forrásmappa felépítése**

A következő alfejezetekben szeretném bemutatni, hogy a fejlesztői környezet milyen részekből áll, és ezekbe kicsit konkrétabban belemenni, hogy mi is történik ezekben a mappákban, és hogyan kötődnek hozzá a program működéséhez.

### **3.4.1. Fejlesztői konténer felépítése**



```
src> utils> train.py
1 from env.custom_env import CustomGazeboEnv
2 import warnings
3 warnings.filterwarnings('ignore', category=UserWarning)
4
5 import rospy
6 from stable_baselines3 import *
7 from stable_baselines3.common.env_checker import check_env
8 from utils.shared import learningmodel, length, xgoal, ygoal
9 import logging
10 import os
11
12 def train():
13     # task: copycar: x: kagades meghivini
14     rospy.init_node('gym_gazebo_env', anonymous=True)
15
16     models_dir = f'resources/models/{learningmodel}'
17     logdir = 'resources/logs'
18
19     if not os.path.exists(models_dir):
20         os.makedirs(models_dir)
21
22     if not os.path.exists(logdir):
23         os.makedirs(logdir)
24
25     env = CustomGazeboEnv()
26     check_env(env)
27
28     obs = env.reset()
29
30     # env.set_goal_position(x=2.0, y=1.0)
31     env.set_goal_position(xgoal, ygoal)
32
33     Timesteps = 10000
34     itera = length // Timesteps
35
36     if learningmodel == "A2C":
37         model = A2C(MlpPolicy, env, verbose=2, n_steps=itera, batch_size=64, ent_coef=0.01, learning_rate=0.0003, tensorboard_log=logdir)
38     elif learningmodel == "PPO":
39         model = PPO(MlpPolicy, env, verbose=2, n_steps=itera, batch_size=64, ent_coef=0.01, learning_rate=0.0003, tensorboard_log=logdir)
40     elif learningmodel == "TD3":
41         model = TD3(MlpPolicy, env, verbose=2, n_steps=itera, batch_size=64, ent_coef=0.01, learning_rate=0.0003, tensorboard_log=logdir)
42     elif learningmodel == "SAC":
43         model = SAC(MlpPolicy, env, verbose=2, n_steps=itera, batch_size=64, ent_coef=0.01, learning_rate=0.0003, tensorboard_log=logdir)
44     else:
45         raise ValueError(f'Unknown learning model: {learningmodel}')
46
47     for i in range(itera):
48         model.learn(total_timesteps=Timesteps, reset_num_timesteps=False, tb_log_name=str(learningmodel))
49         model.save(f'{models_dir}/{Timesteps*(i+1)}')
```

21. ábra: Fejlesztői konténer működés közben

A fejlesztői konténernek a célja az, hogy egy olyan fejlesztői konténer jöjjön létre, amelyben, ha bármilyen probléma történne, akkor teljesen újra lehessen indítani, nem fogja elrontani a fejlesztőnek a gépét semmilyen szinten, nem fog benne semmilyen kárt okozni, a konténer biztosít egy „sandbox” környezetet.

A fejlesztői konténer felépítését 2 részre lehet szétosztani:

1. *.devcontainer* mappa: Megtalálható benne a *devcontainer.json*, amelyben tudjuk beállítani a Docker fájlunk elérhetőségét. Azon kívül beállíthatóak a futási parancsok a Docker fájlhoz, a Python fájlok helyei és a port, amelyen a Visual Studio kommunikálni fog.

```

devcontainer.json X
.devcontainer > {} devcontainer.json > ...
1  {
2      "name": "agents",
3      "build": {
4          "context": "..",
5          "dockerfile": "../Dockerfile"
6      },
7      "workspaceFolder": "/workspaces/MLSzakdoga",
8      "runArgs": [
9          "--network",
10         "host",
11         "--env",
12         "PYTHONPATH=/workspaces/MLSzakdoga/src",
13         "--env",
14         "DISPLAY=${env:DISPLAY}",
15         "--volume",
16         "/tmp/.X11-unix:/tmp/.X11-unix:rw",
17         "--env",
18         "QT_X11_NO_MITSHM=1"
19     ],
20     "remoteEnv": {
21         "PYTHONPATH": "/workspaces/MLSzakdoga/src"
22     },
23     "forwardPorts": [3000],
24     "customizations": {
25         "vscode": {
26             "extensions": [
27                 "ms-vscode-remote.remote-containers",
28                 "ms-python.python"
29             ]
30         }
31     }
32 }

```

22. ábra: devcontainer.json felépítése

2. `.vscode` mappa: Mivel a fejlesztői konténer működéséhez elengedhetetlen a Visual Studio, ezért ebben a mappában a Docker és a Visual Studio közötti kommunikáció történik. A `launch.json` fájl azért felel, hogy amikor elindítjuk a programot, akkor milyen fájl induljon el, és annak a környezeti beállításait, amíg a `tasks.json` file azokért a folyamatokért felel, amit a program indítása előtt el kell végeznie a környezetnek.

```

1 {
2   "version": "0.2.0",
3   "configurations": [
4     {
5       "name": "Python: runprogram.py",
6       "type": "python",
7       "request": "launch",
8       "program": "/workspaces/MLSzakdogam/runprogram.py",
9       "console": "integratedTerminal",
10      "env": {
11        "PYTHONPATH": "/workspaces/MLSzakdogam/src:/opt/ros/noetic/lib/python3/dist-packages",
12        "ROS_PACKAGE_PATH": "/catkin_ws/src:/opt/ros/noetic/share",
13        "ROS_MASTER_URI": "http://localhost:11311",
14        "ROS_PYTHON_VERSION": "3",
15        "ROS_VERSION": "1",
16        "MPLBACKEND": "Agg"
17      },
18      "preLaunchTask": "Start All ROS Processes"
19    }
20  ]
21 }

```

23. ábra: launch.json felépítése

```

1 {
2   "version": "2.0.0",
3   "tasks": [
4     {
5       "label": "Start All ROS Processes",
6       "type": "shell",
7       "command": "bash /workspaces/MLSzakdogam/start_ros.sh",
8       "isBackground": true,
9       "env": {
10        "TURTLEBOT3_MODEL": "burger",
11        "ROS_PACKAGE_PATH": "/workspaces/MLSzakdogam/catkin_ws/src:/opt/ros/noetic/share",
12        "PYTHONPATH": "/workspaces/MLSzakdogam/catkin_ws/src:/opt/ros/noetic/lib/python3/dist-packages"
13      },
14       "presentation": {
15         "echo": true,
16         "reveal": "always",
17         "focus": false,
18         "panel": "new",
19         "showReuseMessage": false
20       }
21     }
22  ]
23 }

```

24. ábra: tasks.json felépítése

### 3.4.2. Fejlesztői Docker konténer felépítése

A fejlesztői Docker fájl, habár megegyezik a felhasználói Docker fájjal, azért van a projektben, hogy a fejlesztő úgy tudjon módosítani a programon, hogy egy biztosan működő programrésze maradjon. Ez az elsődleges szerepe a 2 Docker fájlok felépítésnek.

A Docker első sorában meg van adva milyen alapot használunk a konténerünkhöz, egy hivatalos ROS Noetic Docker képet használunk, amely egy Ubuntu-ra épülő kép, amelyben benne van a Gazebo 11 is, ez a verzió, amit a program is használni fog. A következő részben beállítunk különböző dolgokat, mint például, hogy a gépünkön megjelenjen a Gazebo, vagy a

Turtlebot 3-nak a használt modelljét. Utána telepíteni fogjuk a megfelelő csomagokat. A program legvégén beállítunk még extra beállításokat és utána megadjuk az entrypoint-ot, amelyet már korábbi fejezetben definiáltam, hogyan működik.

```
Dockerfile x
Dockerfile
1 # Using the official ROS Noetic image, which includes Gazebo 11
2 FROM osrf/ros:noetic-desktop-full
3
4 ENV DEBIAN_FRONTEND=noninteractive
5 ENV TURTLEBOT3_MODEL=burger
6 ENV DISPLAY=:0
7
8 RUN apt-get update && apt-get install -y \
9     python3-pip \
10    python3-tk \
11    python3-catkin-tools \
12    ros-noetic-turtlebot3-gazebo \
13    ros-noetic-turtlebot3-slam \
14    ros-noetic-gazebo-ros \
15    ros-noetic-gazebo-ros-pkgs \
16    ros-noetic-gazebo-plugins \
17    ros-noetic-gazebo-ros-control \
18    ros-noetic-ros-controllers \
19    ros-noetic-controller-manager \
20    ros-noetic-joint-state-controller \
21    ros-noetic-effort-controllers \
22    ros-noetic-joint-trajectory-controller \
23    ros-noetic-twist-mux \
24    ros-noetic-teleop-twist-keyboard \
25    ros-noetic-xacro \
26    ros-noetic-diff-drive-controller \
27    ros-noetic-robot-state-publisher \
28    git \
29    && rm -rf /var/lib/apt/lists/*
30
31 RUN git config --global http.postBuffer 104857600
32 RUN git config --global http.lowSpeedLimit 0
33 RUN git config --global http.lowSpeedTime 999
34
35 RUN pip3 install --upgrade pip
36 RUN pip3 install --default-timeout=100 stable-baselines3[extra]
37 RUN pip3 install --default-timeout=100 gym
38 RUN pip3 install --default-timeout=100 catkin_pkg empy rospkg
39 RUN pip3 install --default-timeout=600 tensorflow
40
41 RUN mkdir -p /workspaces/MLSzakdogak/catin_ws/src
42 WORKDIR /workspaces/MLSzakdogak/catin_ws/src
43 RUN git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git && \
44     git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3.git && \
45     git clone -b noetic-devel https://github.com/ROBOTIS-GIT/turtlebot3_simulations.git
46
47 RUN apt-get update && rosdep update && \
48     rosdep install --from-paths . --ignore-src -r -y
49 RUN /bin/bash -c "source /opt/ros/noetic/setup.bash && cd /workspaces/MLSzakdogak/catin_ws && catkin_make"
50
51 COPY ./setup_env.sh /workspaces/MLSzakdogak/setup_env.sh
52 RUN chmod +x /workspaces/MLSzakdogak/setup_env.sh
53
54 RUN echo "source /opt/ros/noetic/setup.bash" >> /root/.bashrc && \
55     echo "source /workspaces/MLSzakdogak/catin_ws/devel/setup.bash" >> /root/.bashrc && \
56     echo "export TURTLEBOT3_MODEL=burger" >> /root/.bashrc && \
57     echo "export ROS_MASTER_URI=http://$(hostname -I | awk '{print \$1}'):11311" >> /root/.bashrc && \
58     echo "export ROS_IP=$(hostname -I | awk '{print \$1}')" >> /root/.bashrc && \
59     echo "export DISPLAY=:0" >> /root/.bashrc
60
61 COPY entrypoint.sh /entrypoint.sh
62 RUN chmod +x /entrypoint.sh
63
64 COPY setup_env.sh /setup_env.sh
65 RUN chmod +x /setup_env.sh
66
67 WORKDIR /workspaces/MLSzakdogak/catin_ws
68
69 ENTRYPOINT ["/entrypoint.sh"]
70
71
```

25. ábra: Dockerfile felépítése

### 3.4.3. Környezet felépítése

A szakdolgozat egyik legnehezebb feladata a saját környezet létrehozása volt, és ennek felépítése. A környezet alapjai a Gazebo-ban használt alapkörnyezetnek a kiegészítése. A környezet meghívásánál megadjuk a kezdő és cél pozíciót, mielőtt feliratkozunk a különböző szenzorokra. Az init függvény végénél még beállítunk pár adatot, hogy a robot megfelelően működhessen. A szenzoroknak mind vannak funkciói, amikre fel kell iratkozni a környezetnek, és ezeket is hívja meg a robot működés közben.

```
def _odom_callback(self, data):
    self.robot_position = np.array([data.pose.pose.position.x, data.pose.pose.position.y], dtype=np.float32)
    orientation_q = data.pose.pose.orientation
    _, _, yaw = euler_from_quaternion([orientation_q.x, orientation_q.y, orientation_q.z, orientation_q.w])
    self.robot_orientation = yaw

def _laser_callback(self, data):
    self.laser_data = np.array(data.ranges, dtype=np.float32)
```

26. ábra: A szenzorok függvényeinek definiálása

Az alap Gazebo-s környezetnek 3 nagyon fontos függvénye van, amelyet mindenképpen saját igényeink szerint újra lehet alakítani:

1. Step függvény: a robot mozgását tudjuk irányítani 3 különböző irányban. A függvény végén megnézzük, hogy a robot pozíciója megegyezik-e a célpozícióval.

```
def step(self, action):
    vel_msg = Twist()
    if action == 0: # Előre
        vel_msg.linear.x = 0.2
        vel_msg.angular.z = 0.0
    elif action == 1: # Balra
        vel_msg.linear.x = 0.0
        vel_msg.angular.z = 0.3
    elif action == 2: # Jobbra
        vel_msg.linear.x = 0.0
        vel_msg.angular.z = -0.3
    self.cmd_vel_pub.publish(vel_msg)
    try:
        rospy.sleep(0.1)
    except rospy.exceptions.ROSInterruptException:
        pass

    obs = self._get_obs()
    reward = self.reward_function()

    terminated = np.linalg.norm(self.goal_position - self.robot_position) < 0.1
    done = bool(terminated)

    return obs, float(reward), done, False, {}
```

27. ábra: A lépés függvénynek definiálása

2. Reward függvény: Jutalmazás a szerepe, ahhoz képest, hogy a robot merre helyezkedik el a célponthoz képest. Ha a célponttól messze helyezkedik el, fálnak ütközne vagy túl közel lenne a falhoz, akkor a robot negatív jutalmat kap.

```
def reward_function(self):
    distance_to_goal = np.linalg.norm(self.goal_position - self.robot_position)

    if distance_to_goal < 0.1:
        reward = 100.0
    else:
        reward = -float(distance_to_goal)

    if self.laser_data is not None:
        min_distance = np.min(self.laser_data)
        if min_distance < 0.5:
            reward -= 10 * (0.5 - min_distance)

    return reward
```

28. ábra: A jutalom függvénynek definiálása

3. Reset függvény: Összesen annyi szerepe van, hogy a robot minden fontos adatát újraindítsa.

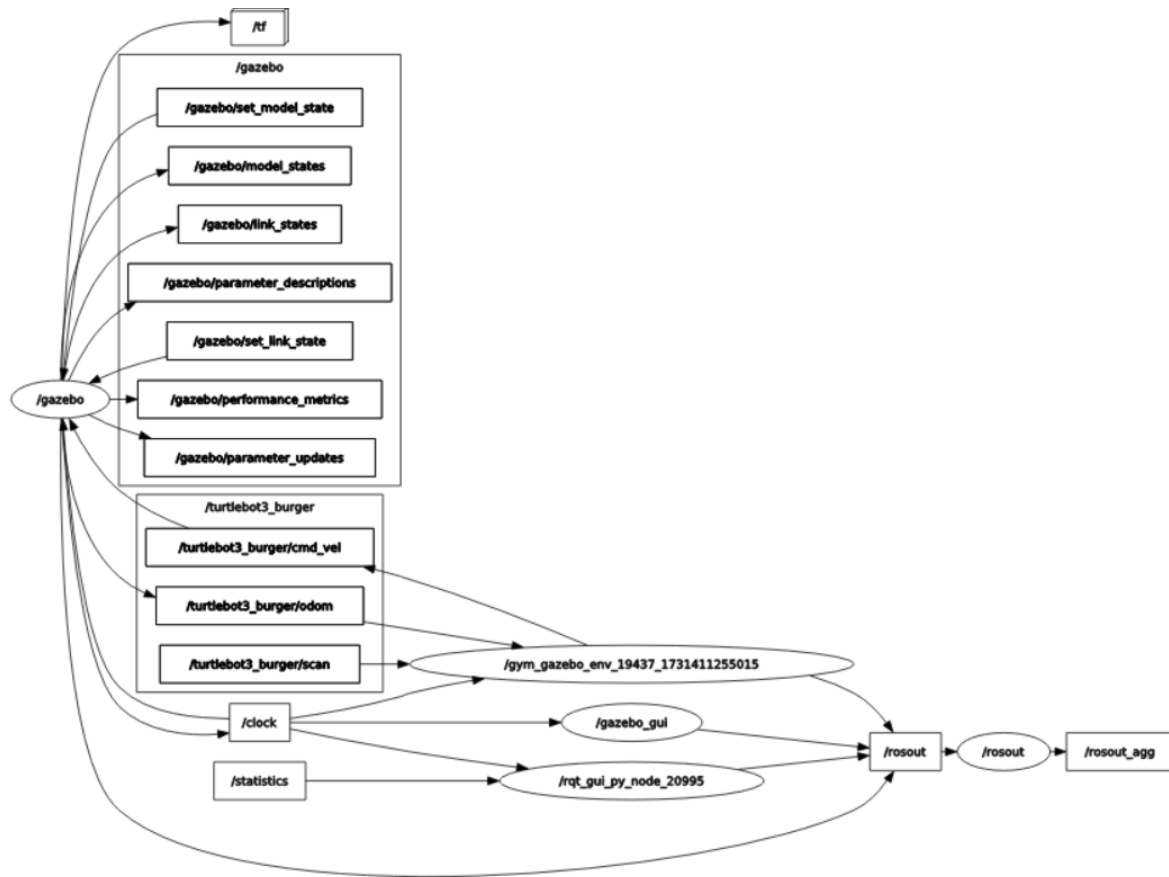
```
def reset(self, seed=None, options=None):
    super().reset(seed=seed)
    try:
        self.reset_simulation()
        rospy.sleep(1)
    except rospy.exceptions.ROSInterruptException:
        pass

    self.robot_position = np.array(self.start_position, dtype=np.float32)
    self.robot_orientation = 0
    self.laser_data = np.zeros(1, dtype=np.float32)
    obs = self._get_obs()
    return obs, {}
```

29. ábra: Az alapállapot függvénynek definiálása

#### 3.4.4. Robot és a környezet kapcsolata, kommunikációja

Az egyik legfontosabb feladat a robot és a környezet közötti megfelelő kommunikáció felállítása volt. Ezt sikerült is teljesíteni, az alábbi ábrán látható is melyik funkció hogyan kommunikál a robottal.



30. ábra: Robot és a környezet közötti működési kapcsolat

## 3.5. Követelmény-specifikáció

Ebben a fejezetben szeretném bemutatni a dolgozat elején megírt követelmény-specifikációt, amelynek tervei alapján készült a dolgozat.

### 3.5.1. Követelményelemzés

Az alapfeladat mindenképpen az lenne, hogy egy ilyen komplex mesterséges intelligencia programnak, mint a mobil robot megerősítéses tanítása, annak adjak egy olyat keretet, amely segítségével a legtöbb nem hozzáértő ember is tudja használni, nagyobb probléma nélkül. Korábbi kutatásaim alapján, úgy véltem, hogy Linuxon sokkal több segítség elérhető, mint a többi operációs rendszeren ezért mindenképpen abban a környezetben szeretném megoldani a feladatot. Mivel egy fontos része a feladatnak a környezet létrehozása, amely sok emberhez eljuthat, ezért szeretném a lehető legegyszerűbben kezelhetőre megcsinálni a programot. Mivel nagyon fontos a programnak a hordozhatóság, ezért nem szeretnék felhasználói felületet beletenni és helyette konfiguráció alapú működést szeretnék létrehozni. A legfontosabb része a tanítás lesz, ahol Gazebo pályán szeretném megvalósítani a tanítást. A felhasználónak



lehetősége lesz nézni élőben, ahogyan a program működik és tanul a robot. A következő funkció szerepe egy olyan funkció lenne, ahol a tanítást lehetne folytatni, így megadva a lehetőségét, hogy bármikor használható legyen a program, bármilyen időhosszig. A harmadik funkciónak a lényege az lenne, hogy a tanítással szerzett adatokat feldolgozza, hogy felhasználható állapotba tegye őket. A végleges funkciója a szoftvernek az lenne, hogy a felhasználónak vegyen fel a szoftver egy videót, amin látszódik a robot mozgása, ezzel a megfelelő kutatáshoz, lesz videós anyaga is a kutatónak.

A felhasználónak legyen lehetősége a konfigurációs rendszerben a beállítások könnyű módosítására, ennek könnyű és gyors feldolgozására.

A programot a Linux környezeten kívül (22.04 Ubuntu), Python programozási nyelvben szeretném megvalósítani. Kutatásaim alapján arra jutottam, hogy a Linux + Python a legnépszerűbb környezet mesterséges intelligencia alkalmazások megvalósítására. A konfigurációs alapú működést .yaml kiterjedésű fájlok fogják biztosítani. A mesterséges intelligencia részénél pedig a Turtlebot oldalán található útmutatót fogom használni kiindulópontnak, annak segítségével szeretném összekötni azt a projektet a sajátommal, esetlegesen módosítani azon, ha a program megköveteli. A korábbi egyetemes tanulmányaimat is át fogom nézni, mi szállítható át belőle ebbe az új projektbe.

Az alkalmazás célja, hogy gyors, megbízható és környezetbarát legyen. Célja, hogy minél kisebb mérete miatt könnyen szállítható legyen mindenféle rendszerre, erőforrástól függetlenül.

### **3.5.2. Megvalósíthatósági terv**

- Humán erőforrás: 1 tervező/fejlesztő (250 emberóra)
- Hardver erőforrás: 1 fejlesztői, tesztelői számítógép (Ubuntu 22.04)
- Szoftver erőforrások: fejlesztőkörnyezet (Visual Studio), verziókövető (Github)
- Üzemeltetés: nem kell biztosítani
- Karbantartás: nem kell biztosítani
- Megvalósítás időtartama: Összesen 250 emberóra

### **3.5.3. Nem funkcionális követelmények**

- Hatékonyság: A program terhelés jelent a processzorra és memóriára, mentés esetén a háttértárat is terhelheti. A program igényel hálózati kapcsolatot. A program gyorsan működik, a tanítás és megjelenítés részei a programnak időigényesek lehetnek. A program hatékony működéséhez egy jól felszerelt rendszer ajánlott.
- Megbízhatóság: Szabványos használat esetén nem fordul elő hibajelenség, és nem jelenik meg hibaüzenet. Ha valamilyen adat megsérül arról a program logolni fog a megfelelő helyre, ahol a felhasználó megtudhatja mi a hiba és hogy mit tegyen.
- Biztonság: A program csak a telepítés idejére fog internet kapcsolatot igényelni, a program offline működése közben az adatok biztonságban vannak.
- Működési: Hosszabb használati idő, akár 5-6 óra is, gyakori használat várható
- Fejlesztési: Python nyelv, Linux rendszerben megvalósítva

### **3.5.4. Funkcionális követelmények**

Alapprogram:

- Tanítás lehetősége
- Tanítás folytatása
- Adatok megformázása
- Képernyő felvétele

Adatok megformázása:

- Nyers adatok kigyűjtése, átalakítása
- Feldolgozható formátumba alakítása

Képernyő felvétele:

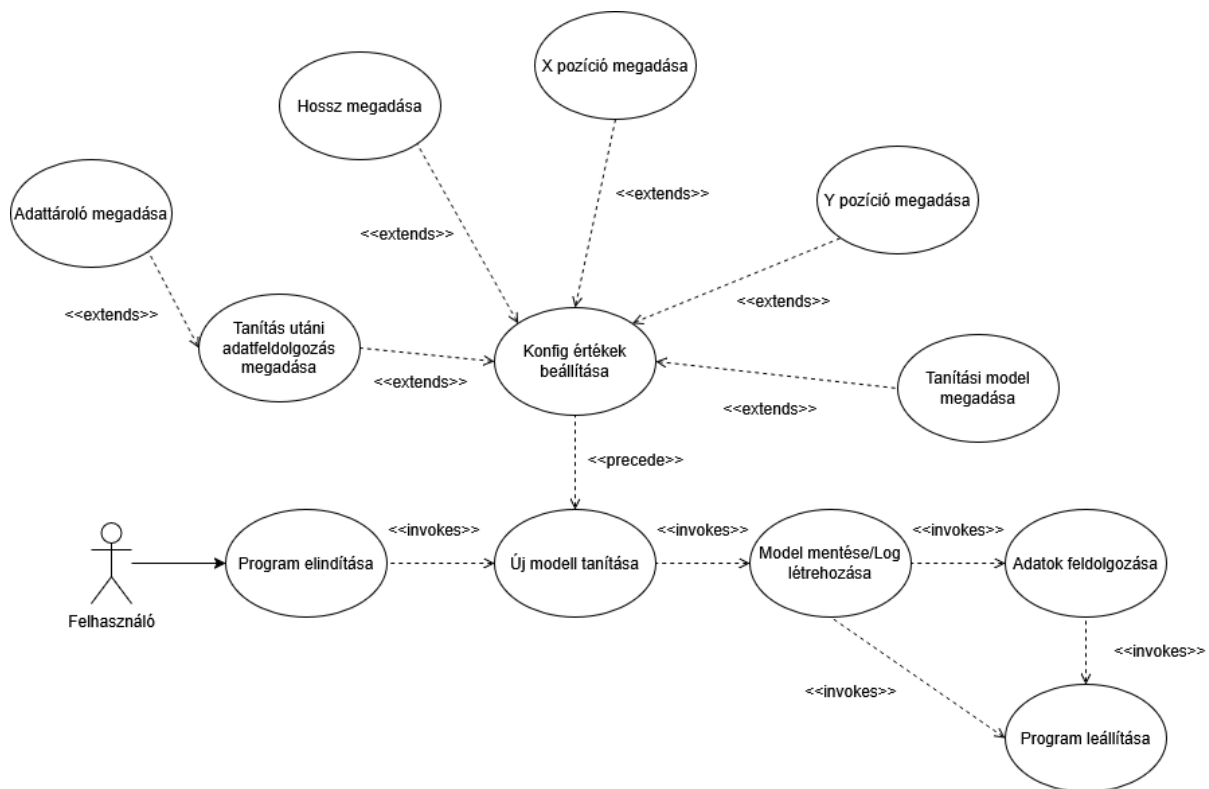
- Program működése közben létrejövő videó
- Kutatásokhoz tökéletes

### 3.6. Felhasználói esetek

A program 3 részre bontható funkcionalitás szempontjából: Új modell tanítása, Tanítás folytatása és Tanítási adatok kiírása. A következő alfejezetekben szeretném ezeknek a funkcióknak bemutatni a felhasználói eseteit és azok diagrammjaikat.

#### 3.6.1. Új modell tanítása felhasználói esetei

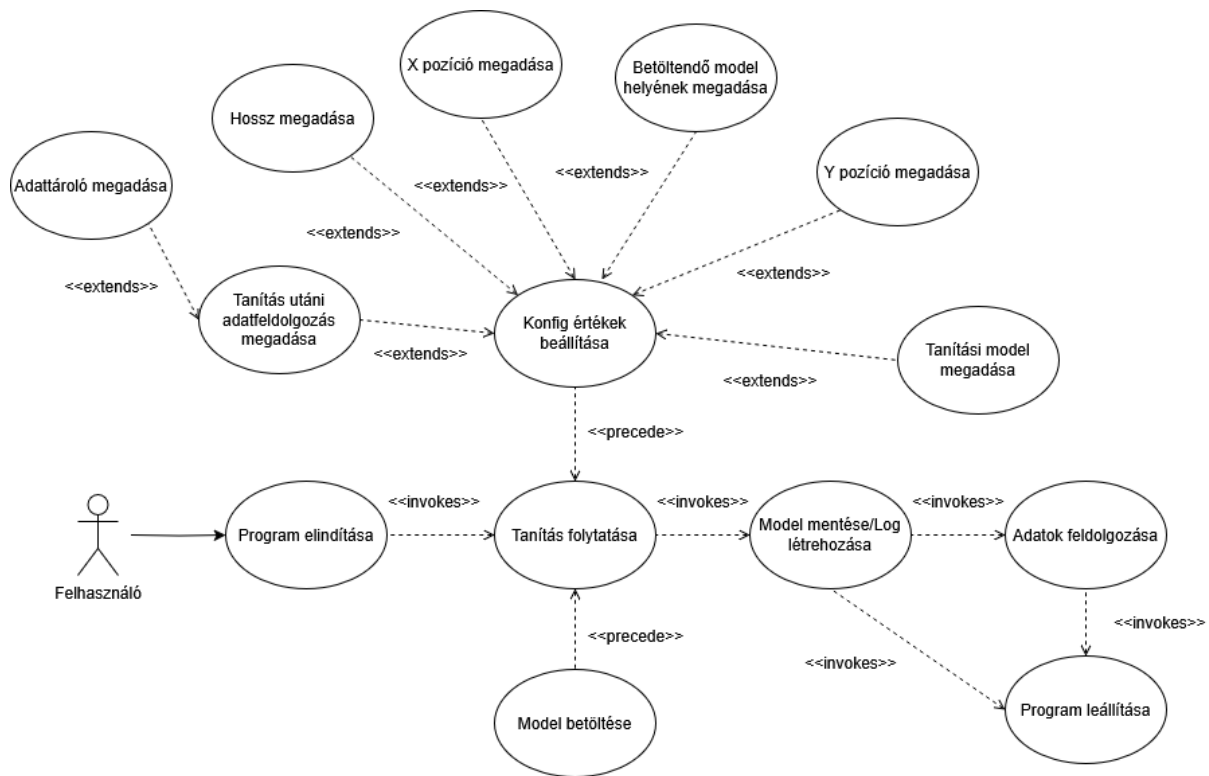
Ennek a funkciónak a futása előtt van lehetőségünk beállítani a konfigurációban található beállításokat. Ezután ezt a program betölti és elindítja a tanítást. Időközönként a program menteni fog, és ha be van kapcsolva a mentés funkció, akkor a tanítás vége után feldolgozza a keletkezett adatokat. A legvégén pedig kilép a program.



31. ábra: Új modell tanítása felhasználói esetei diagrammon ábrázolva

#### 3.6.2. Tanítás folytatása felhasználói esetei

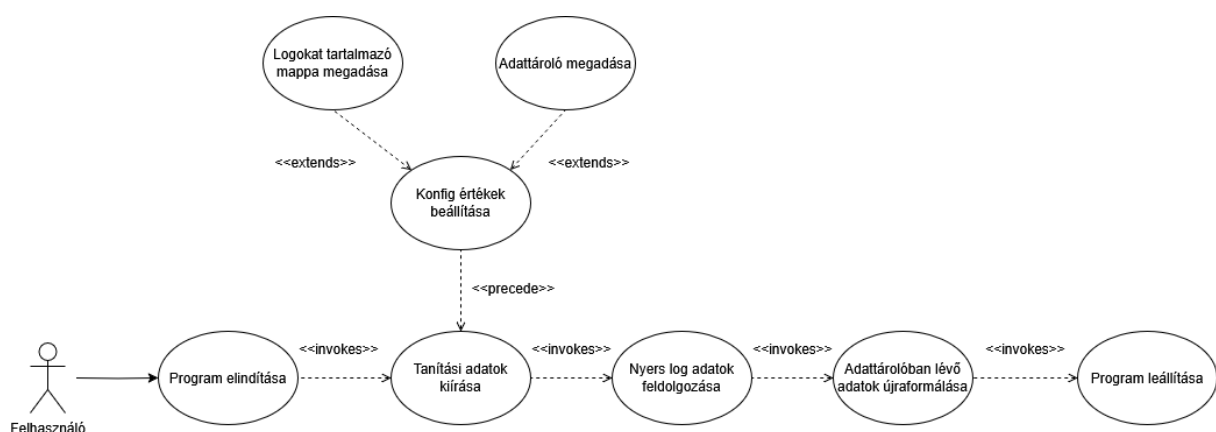
Ennek a funkciónak a futása előtt van lehetőségünk beállítani a konfigurációban található beállításokat. Ezután ezt a program betölti és mielőtt elindítja a tanítást, azelőtt betölti a folytatandó modellt. Időközönként a program menteni fog, és ha be van kapcsolva a mentés funkció, akkor a tanítás vége után feldolgozza a keletkezett adatokat. A legvégén pedig kilép a program.



32. ábra: Tanítás folytatása felhasználói esetei diagrammon ábrázolva

### 3.6.3. Tanítási adatok kiírása felhasználói esetei

Ebben a funkcióban, miután beállítottuk a megfelelő konfigurációs beállításokat, azután a program a megadott mappában lévő log fájlokat fel fogja dolgozni és ki fogja belőlük nyerni a fontos adatokat. Ezután a program a már létrejött adatokat átalakítja, hogy jobban értelmezhető legyen a felhasználó számára. Ezután leáll a program.



33. ábra: Tanítási adatok kiírása felhasználói esetei diagrammon ábrázolva

### **3.7. Rendszerarchitektúra**

A következő részben a rendszerben használt programnyelvet és főbb komponenseket szeretném bemutatni.

#### **3.7.1. Programnyelv és fejlesztői környezetek**

A programhoz Python programozási nyelvet választottam, ez a nyelv támogatta legjobban, azt a fajta dolgot, amit létre akartam hozni. A fejlesztői környezet a legfrissebb Visual Studio Code volt, amelyben Dev Container segítségével történt a fejlesztés. A Visual Studio Code-ban használtam a Remote Containers és Python csomagokat is a könnyebb használat érdekében.

A környezet felállításához Docker fájlokat és különböző Bourne shell scripteket használok.

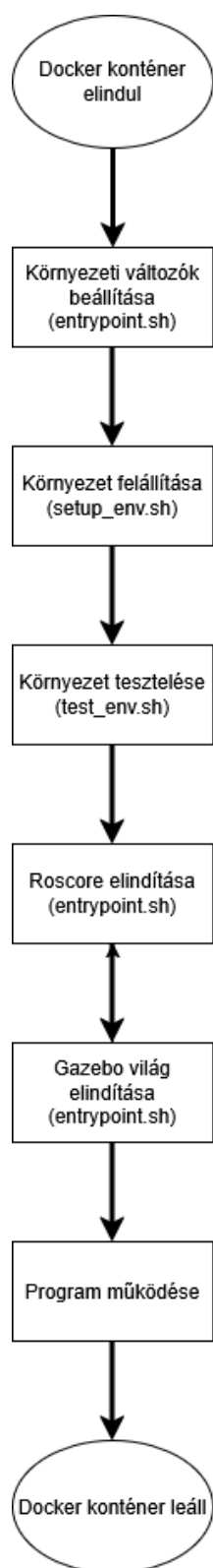
#### **3.7.2. Főbb komponensei a programnak**

A programnak több főbb komponense van, ezeket szeretném bemutatni a következő sorokban:

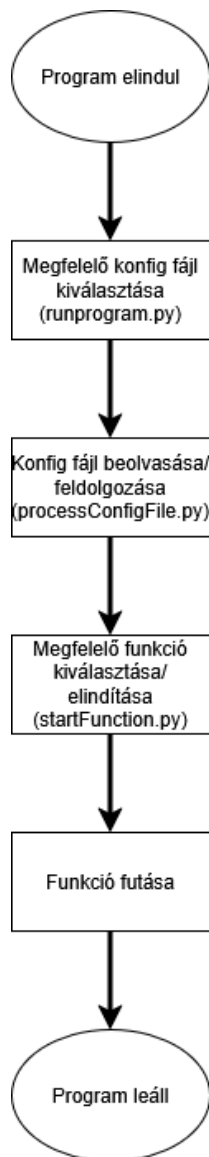
- **runprogram.py**: A Docker indítása utáni kiinduló pont, itt dől el milyen konfiguráció fog betöltődni a programban.
- **sharedValues.py**: Ebben a fájlban találhatóak olyan értékek, amelyeket a program minden része használni fog
- **processConfigFile.py**: A konfiguráció betöltéséért és feldolgozásáért felelős fájl
- **startFunction.py**: A megfelelő funkcióhoz tartozó metódus elindításáért felel
- **createDirectories.py**: A *resources* és tanulási funkciók során létrejövő mappákat hozza létre
- **trainGazebo.py**: Az új modell tanítása funkció implementációja
- **continueTrainingGazebo.py**: A tanítás folytatása funkció implementációja
- **saveDataFromTensorboardFiles.py**: A Tanítási adatok kiírása funkció implementációja

#### **3.7.3. A környezet és a program folyamatábrái**

A program működését 2 részre lehet szétszedni, az első a környezet és a második a program folyamata, mi mivel kommunikál, hogy történnek egymás után. Erről részletesebben a fejlesztői környezetről szóló részben fogok írni.



34. ábra: Környezet felépítésének folyamatábrája



35. ábra: Program működésének folyamatábrája

### 3.8. A fejlesztői környezet

Miután kicsomagoltuk a MLSzakdoga.zip-et, ezeket a lépéseket kell tennünk mielőtt el tudjuk indítani a fejlesztői környezetet:

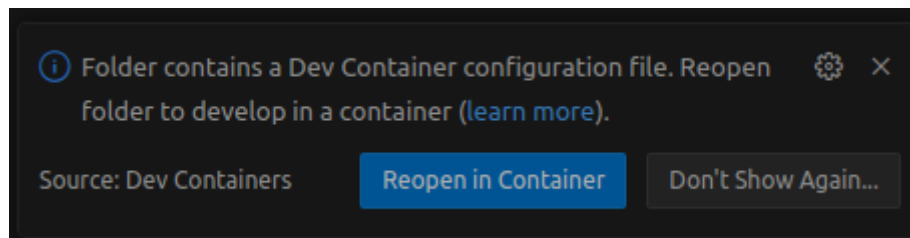
1. Visual Studio Code telepítése: [8]
2. Megjelenítő képernyő beállítása a terminálban:

```
# X szerver konfigurálása  
xhost +local:docker
```

Ezzel a beállítással azt fogjuk elérni, hogy amikor a Dockeren belül elindul az alkalmazás, akkor a mi képernyőnkön jelenjen meg a Gazebo. Amikor megnyitjuk először a kódot a Visual Studio

Codeban, akkor a jobb alsó sarokban meg fog jelenni az opció, hogy beállítsuk a fejlesztői környezetet:

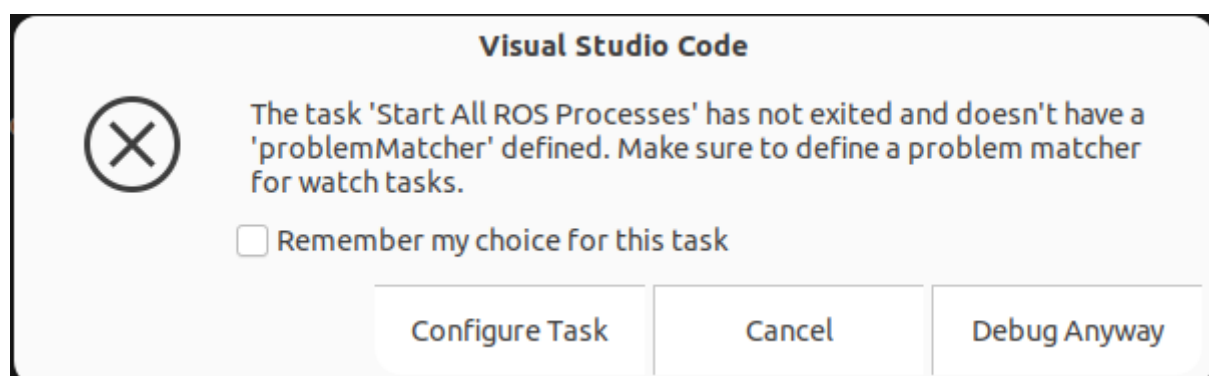
**FONTOS!** Mielőtt megnyitnánk a mappát a Visual Studio Codeban, bizonyosodjunk meg róla, hogy a mappa neve MLSzakdoga! Különben két külön részre fogja szedni a fájlokat a Visual Studio Code. Ha mégis ez történne, akkor a File – Open Folder menüpontban fogjuk megtalálni a fájlokat a nem MLSzakdoga nevű mappában.



36. ábra: Visual Studio PopUp amiben elindítható a Dev Container

Miközben a fejlesztői konténer beállítása történik, közben a Docker kép le fog tölteni, ezáltal az első elindításkor sokat kell majd várnunk mire összeáll a program. Ezután végre készen állunk a program elindításához.

Az alkalmazást az F5 gomb megnyomásával tudjuk elindítani. Ezután összeáll a Gazebo környezet és elindul a világunkkal és a robotunkkal. Ezután a Visual Studio Code ki fog írni egy figyelmeztetést, mivel a Gazebo elindítása egy folytonos feladat, ezért az alkalmazásunk nem fog tudni elindulni magától.



37. ábra: Visual Studio Code Warning üzenet



A felhasználónak a Debug Anyway gombot kell megnyomnia és akkor fog elindulni a *runprogram.py* fájl, és pár másodperccel később már látni fogjuk a terminálban és a Gazebo-ban, hogy elindult a kívánt funkció.

Ha esetleg hibába ütközünk miközben elindítjuk a programot, bizonyosodjunk meg róla, hogy rendesen beállítottuk a megjelenítő képernyőt és utána építsük újra a konténert. Ezután már normálisan el kéne indulnia a programnak.

### **3.8.1. A fejlesztői környezet működése**

A fejlesztői környezet működését 2 főbb részre lehet szétosztani, a környezet felépítésére és a programra.

A környezet már akkor felépül amikor elindítjuk a Visual Studio-t. Amikor elindul a szoftver, akkor először is csatlakozik a konténerhez, amely a már feltelepített Docker konténer. Ilyenkor megtörténik minden a Gazebo és Ros-al kapcsolatos telepítés, ezért amikor majd el akarjuk indítani a környezetet, akkor sokkal gyorsabb lesz, mint a felhasználói változat, mivel már előre be van „töltve” a Gazebo környezet, csak annyi dolga van hátra, hogy elindul, amíg a felhasználói környezetnek teljesen fel kell épülnie minden alkalommal, mivel akkor indul el a Docker konténer.

Az olyan környezettel kapcsolatos fájlok, mint a *setup\_env.sh*, *test\_env.sh* vagy a *catkin\_ws* mappa, amely a Turtlebot3 Github projekteket tartalmazza vagy a Ros környezet felállításához használt scriptek, azok már le fognak futni a környezet felépítése közben.

A fejlesztői környezet másik fontos része a program lesz. Amikor elindítjuk a programot, akkor először a *runprogram.py* fájlba fogunk belemenni, ott fogja beolvasni, hogy a *launch.json* fájlban milyen **FUNCTION\_NAME** paramétert adtunk meg. Ezután a *main.py*-ban meghívom a *processConfigFile.py* fájlt, amelyben a konfigfájlok feldolgozása zajlik, erről részletesebben később fogok beszámolni. Ezután pedig a *runprogram.py* segítségével a megfelelő funkciót el fogja indítani a program.

Ezen kívül megtalálható a *deploy* mappa, amelyben a felhasználói felület található, amíg a *docs* és *thesis* mappában pedig a dolgozat dokumentuma, és pár extra screenshot és videó található a program működéséről.

### **3.9. Tesztkörnyezet**

A tesztkörnyezet a fejlesztői környezetben lesz megtalálható. Egybe van importálva a *src* fájlban található kódokkal, amelyen belül a tests mappán belül találhatóak meg a tesztek. A tesztek nagy része, mivel a Gazebot fogja tesztelni, ezért a futásuk előtt el kell indítani a Gazebo-t a fejlesztőnek.

### **3.10. A tesztelési terv**

Bármelyik szoftverfejlesztés egyik legfontosabb feltétele a megfelelő tesztelés. Egy megfelelően tesztelt program nagyon sok későbbi fejfájástól meg tudja kímélni a fejlesztőt. Az alábbi részen szeretném bemutatni a teszteléseket, amelyeket a programon elvégeztem.

#### **3.10.1. Manuális tesztelés**

A manuális tesztelés minden elkészült funkció után készült. A programfordítás után a program futása közben különböző check-ekkel tesztelem, hogy teljesül-e a meghatározott szerepe a funkciónak. Ha ez nem teljesül, akkor a kódban hibajavítással javítom ki a hibát és ezt addig ismétlem, amíg az elvárt eredmény nem következik be. Minden funkció, ami a programban található az alapos manuális tesztelési folyamaton esett át, ezért bárki, aki szeretné a szoftvert fejlesztői szempontból megközelíteni a jövőben, annak már nem kell majd aggódnia a manuális tesztelés miatt, amíg az eredeti funkciólistát megtartja.

#### **3.10.2. Automatikus tesztelés**

A projekt automatikus tesztelését a Python unittest tesztelő rendszer segítségével hoztam létre. Ez a rendszer alapvetően a TestCase osztályra épül, ez teszi lehetővé számunkra, hogy több tesztesetet tudjunk definiálni. A TestCase osztály metódusai is elérhetőek ebben a rendszerben, mint például az `assertEqual`, ahol két különböző értéket nézünk meg, hogy megegyeznek-e, vagy például az `assertRaises`, ahol pont a helytelen működést tudjuk azzal tesztelni, hogy megnézzük, hogy a teszt hibás megoldással tér-e vissza.

A mi projektünknel az automatikus tesztelés, 3 különböző dolgot fog tesztelni, ezekről szeretnék röviden írni.

1, ROS környezet tesztelése:

Ezekben a tesztekben a saját általunk használt ROS környezetet teszteljük a Gazebo segítségével. A lényege ezeknek a teszteknek, hogy kisebb mozgásokra és értékváltozásokra reagál-e megfelelően a program.

```
customGazeboEnvironmentTest.py 2 X
src > tests > customGazeboEnvironmentTest.py > customGazeboEnvironmentTest > test_ChangeGoalPosition
1 import unittest
2 import rospy
3 import time
4 import warnings
5 from env.customGazeboEnv import customGazeboEnv
6 from stable_baselines3.common.env_checker import check_env
7 import numpy as np
8 from nav_msgs.msg import Odometry
9
10 class customGazeboEnvironmentTest(unittest.TestCase):
11     env = None
12
13     def _odom_callback(self, data):
14         self.robot_position = np.array([data.pose.pose.position.x, data.pose.pose.position.y], dtype=np.float32)
15
16     @classmethod
17     def setUp(cls):
18         warnings.simplefilter('ignore', category=ResourceWarning)
19         rospy.init_node('gym_gazebo_env', anonymous=True)
20         cls.env = customGazeboEnv()
21
22     def test_CheckEnv(self):
23         print("test_CheckEnv started!")
24         check_env(self.env)
25
26     def test_CheckEnvInitValues(self):
27         print("test_CheckEnvStartAndGoalPositions started!")
28         self.assertEqual(self.env.robot_position.tolist(), [0, 0])
29         self.assertEqual(self.env.goal_position.tolist(), [5, 5])
30         self.assertEqual(self.env.robot_orientation, 0)
31
32     def test_ChangeGoalPosition(self):
33         print("test_ChangeGoalPositionTest started!")
34         new_goal_position = (10, 10)
35         self.env.set_goal_position([*new_goal_position])
36
37         self.assertEqual(self.env.goal_position.tolist(), [10, 10])
```

38. ábra: customGazeboEnvironmentTest.py felépítése

## 2, Konfig beolvasás tesztelése:

Az ilyen fajta tesztekben a program konfigbeolvasási funkcióit tesztelem, hogy ha a megfelelő értékek hiányoznak, akkor kilép a program ahogyan annak kell, vagy ha esetleg olyan érték hiányzik, amelyik pótolható, akkor a megfelelő helyettesítést használja. A végén pedig van egy pár tesztelés, ahol tökéletesen beolvas minden értéket, minden megfelelően működik.

```
gazeboProgramTest.py X
src > tests > gazeboProgramTest.py > processConfigFileTest > continueFunctionWithSave

8 import csv
9
10 class processConfigFileTest(unittest.TestCase):
11
12     @classmethod
13     def setUpClass(cls):
14         for folder in [
15             "testSources/gazeboProgramTests/learnFunction/logFolder",
16             "testSources/gazeboProgramTests/learnFunction/modelFolder",
17             "testSources/gazeboProgramTests/learnFunction/csvFolder",
18             "testSources/gazeboProgramTests/continueFunction/logFolder",
19             "testSources/gazeboProgramTests/continueFunction/modelFolder",
20             "testSources/gazeboProgramTests/continueFunction/csvFolder",
21             "testSources/gazeboProgramTests/saveDataFunction/csvFolder"
22         ]:
23             if os.path.exists(folder):
24                 for file in os.listdir(folder):
25                     os.remove(os.path.join(folder, file))
26
27         csv_file_path = os.path.join("testSources/gazeboProgramTests/learnFunction/csvFolder", "data.csv")
28         with open(csv_file_path, 'w', newline='') as f:
29             writer = csv.writer(f)
30
31         csv_file_path = os.path.join("testSources/gazeboProgramTests/continueFunction/csvFolder", "data.csv")
32         with open(csv_file_path, 'w', newline='') as f:
33             writer = csv.writer(f)
34
35         csv_file_path = os.path.join("testSources/gazeboProgramTests/saveDataFunction/csvFolder", "data.csv")
36         with open(csv_file_path, 'w', newline='') as f:
37             writer = csv.writer(f)
38
39     def setUp(self):
40         self.setUpClass()
41
42     def tearDown(self):
43         self.setUpClass()
44
45     @patch('utils.trainGazebo.createDirectories')
46     def test_learnFunctionWithSave(self, mock_start):
47         print("test_learnFunctionWithSave started!")
48
49         logFolder = "testSources/gazeboProgramTests/learnFunction/logFolder"
50         modelFolder = "testSources/gazeboProgramTests/learnFunction/modelFolder"
51         csvFilePath = "testSources/gazeboProgramTests/learnFunction/csvFolder/data.csv"
52
53         sharedValues.setXGoal(5.0)
54         sharedValues.setYGoal(5.0)
55         sharedValues.setLearningModel("PP0")
56         sharedValues.setLength(200)
57         sharedValues.setSaveDataAfterFinished(True)
```

39. ábra: gazeboProgramTest.py felépítése

### 3, Program tesztelése:

Az utolsó tesztelési csoportban, a Gazebo-ban belül tesztelünk különböző funkciókat.

```

processConfigFileTest.py X
src > tests > processConfigFileTest.py > processConfigFileTest > test_emptyLogFolderConfig
1 import unittest
2 from unittest.mock import patch, MagicMock, mock_open
3 import yaml
4 from config.processConfigFile import processConfigFile, DEFAULTS
5 from utils.sharedValues import sharedValues
6
7 class processConfigFileTest(unittest.TestCase):
8
9     @classmethod
10    def setUpValues(cls):
11        sharedValues.setXGoal(0.0)
12        sharedValues.setYGoal(0.0)
13        sharedValues.setLearningModel("")
14        sharedValues.setLength(0)
15        sharedValues.setSaveDataAfterFinished(False)
16        sharedValues.setCSVFilePath("")
17
18    def tearDown(self):
19        self.setUpValues()
20
21    @patch('config.processConfigFile.startFunction')
22    def test_emptyYamlConfigTest(self, mock_start):
23        print("test_emptyYamlConfigTest started!")
24        path = "testSources/configProcessingTests/emptyConfigFile.yaml"
25
26        with self.assertRaises(ValueError) as context:
27            processConfigFile(path)
28
29        self.assertIn("FunctionName is not defined", str(context.exception))
30
31    @patch('config.processConfigFile.startFunction')
32    def test_emptyFunctionNameConfig(self, mock_start):
33        print("test_emptyFunctionNameConfig started!")
34        path = "testSources/configProcessingTests/missingFunctionNameConfigFile.yaml"
35
36        with self.assertRaises(ValueError) as context:
37            processConfigFile(path)
38
39        self.assertIn("FunctionName is not defined", str(context.exception))
40

```

40. ábra: processConfigFileTest.py felépítése

A unittest *setUp* metódusa azt teszi lehetővé, hogy a tesztfájl kezdete előtt egy teljesen új környezet jöjjön létre, ezzel biztosítva, hogy az előző tesztfutás nem rontotta el a környezetet a következő számára. Azért is fontos mert ha esetleg valamikor az egyik tesztfutás helytelen lenne, akkor ne történjen meg az, hogy a következő tesztfutások mint dominók, úgy bukjanak utána, hanem minden tesztfutás egy külön kis környezetben menjen, ezzel biztosítva van, hogy nem tudják befolyásolni egymást. A környezet lebontásáért a *tearDown* metódus felel, ezáltal nem marad lezáratlanul az objektum. A tesztek közben naplózás segítségével leírom, merre tart a tesztfájl, ezáltal a futása során tudjuk követni éppen melyik tesztnél tart. Ez azért kell, mert mint mondjuk a „Google Test”-nél nincsen vizuális kiírás a legvégén kívül, ezért ennyiben akartam segíteni a fejlesztőnek.

A környezetet úgy tudjuk elindítani, hogy nyitunk a Visual Studio Code környezetünkben egy új terminált, és utána belemegyünk a megfelelő könyvtárba:

# Automatikus tesztelés elindítása

`cd src/tests`

# Attól függ a felhasználó melyiket szeretné futtatni

`python3 customGazeboEnvironmentTest.py/gazeboProgramTest.py/processConfigFileTest.py`

A terminál figyelésével látszódik melyik teszt indult el, és minden esetleges hibát kiír a rendszer, ezután elmondja, hogy mennyi tesztet futtatott le a program és utána kilép.

### 3.10.3. Tesztelési eredmények

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2
2025-04-22 05:25:16.184 INFO | utils.sharedValues:printValues:20 - x, y goal value: 16.0, 20.0
2025-04-22 05:25:16.184 INFO | utils.sharedValues:printValues:21 - functionName: Learn
2025-04-22 05:25:16.184 INFO | utils.sharedValues:printValues:22 - modelPath: /workspaces/MLSzakdoga/resources/models/mycoolmodel.zip
2025-04-22 05:25:16.184 INFO | utils.sharedValues:printValues:23 - shared::printValues Ended!
.test_usingDefaultValuesConfig started!
2025-04-22 05:25:16.186 INFO | config.processConfigFile:getDataWithDefault:19 - Key: XGoalPosition is empty, default value: 2.0 will be
2025-04-22 05:25:16.186 INFO | config.processConfigFile:getDataWithDefault:19 - Key: YGoalPosition is empty, default value: 2.0 will be
2025-04-22 05:25:16.186 INFO | config.processConfigFile:getDataWithDefault:19 - Key: LearningModel is empty, default value: PPO will be
2025-04-22 05:25:16.186 INFO | config.processConfigFile:getDataWithDefault:19 - Key: Length is empty, default value: 8000 will be used
2025-04-22 05:25:16.186 INFO | config.processConfigFile:getDataWithDefault:19 - Key: SaveDataAfterFinished is empty, default value: Fal
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:17 - shared::printValues Started!
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:18 - length: 8000
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:19 - LearningModel: PPO
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:20 - x, y goal value: 2.0, 2.0
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:21 - functionName: Learn
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:22 - modelPath: /workspaces/MLSzakdoga/resources/models/mycoolmodel.zip
2025-04-22 05:25:16.187 INFO | utils.sharedValues:printValues:23 - shared::printValues Ended!
.
-----
Ran 11 tests in 0.024s
OK
```

41. ábra: Helyes tesztfutás

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS 2
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:17 - shared::printValues Started!
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:18 - length: 8000
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:19 - LearningModel: PPO
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:20 - x, y goal value: 2.0, 2.0
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:21 - functionName: Learn
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:22 - modelPath: /workspaces/MLSzakdoga/resources/models/mycoolmodel.zip
2025-04-22 05:25:55.615 INFO | utils.sharedValues:printValues:23 - shared::printValues Ended!
F
=====
FAIL: test_usingDefaultValuesConfig (__main__.processConfigFileTest)
Traceback (most recent call last):
  File "/usr/lib/python3.8/unittest/mock.py", line 1325, in patched
    return func(*newargs, **newkeywargs)
  File "processConfigFileTest.py", line 68, in test_usingDefaultValuesConfig
    self.assertEqual(sharedValues.xGoal, 5.0)
AssertionError: 2.0 != 5.0
-----
Ran 11 tests in 0.025s
FAILED (failures=1)
○ root@d:/workspaces/MLSzakdoga/src/tests#
```

42. ábra: Helytelen tesztfutás

A manuális tesztelés során megbizonyosodtam róla, hogy minden egyes funkció az elvárt módon fog szerepelni, akár a saját gépem, akkor egy teljesen új számítógépen, és ugyanazt

az eredményt érem el, ha felhasználói módban indítom el a programot vagy ha fejlesztői módban végzem el ugyanazt a feladatot.

Az automatikus tesztelés is tökéletes működik a saját rendszeremen és bármelyik rendszeren, ahol próbáltam. A tesztek gyorsan és precízen lefutnak és mindenhol az elvárt eredményt adják vissza. Azok a mappák és fájlok, amiket létrehoz működés közben a tesztelés törlésre kerül, ezzel is megspórolva tárhelyet.

Ha esetlegesen megbukik valamelyik teszt, akkor látható lesz, hogy melyik érték nem lesz megfelelő, és hogy mit kell változtatni, hogy újra megfelelő legyen a teszt, viszont, ha a fejlesztő nem módosít a teszteken vagy a program belső működésén, akkor ez nem fog megtörténni.

#### **3.10.4. Tesztek részletes leírása**

A három csoport alapján szeretném bemutatni milyen tesztek találhatók meg az automatikus tesztelésben:

##### **1, ROS környezet:**

- test\_CheckEnv: A környezet ellenőrzéséért felelős teszt
- test\_CheckEnvInitValues: A környezet alapértelmezett értékeinek vizsgálatáért felelős teszt
- test\_ChangeGoalPosition: Ha a környezetben módosítjuk a végcél, akkor a program ezt érzékeli, ezt vizsgálja a teszt
- test\_ResetEnv: Környezet újraindításának tesztelése, értékek visszaállnak-e, az alapértelmezett állapotukra
- test\_Reward: A reward funkció helyes működését ellenőrző teszt
- test\_Odom: Odom megfelelően működéséért felelős teszt

##### **2, Konfig beolvasás:**

- test\_emptyYamlConfigTest: Üres Yaml fájl esetén hibát kap-e a program
- test\_emptyFunctionNameConfig: Üres FunctionName esetén hibát kap-e a program
- test\_emptyLogFolderConfig: Üres LogFolder esetén hibát kap-e a program
- test\_emptyModelPathConfig: Üres ModelPath esetén hibát kap-e a program

- test\_usingDefaultValuesConfig: Ha üresek az értékek, amelyek helyettesíthetők, akkor megfelelően pótolja őket
- test\_usingCsvFilePathDefaultValueConfig: Ha üres a csvFilePath, akkor helyesen helyettesíti a program
- test\_goodLearnWithSaveConfig: Helyes Learn konfiguráció beolvasása, a Save funkcióval
- test\_goodLearnWithoutSaveConfig: Helyes Learn konfiguráció beolvasása, a Save funkció nélkül
- test\_goodContinueWithSaveConfig: Helyes Continue konfiguráció beolvasása, a Save funkcióval
- test\_goodContinueWithoutSaveConfig: Helyes Continue konfiguráció beolvasása, a Save funkció nélkül
- test\_goodSaveDataConfig: Helyes SaveData konfiguráció beolvasása

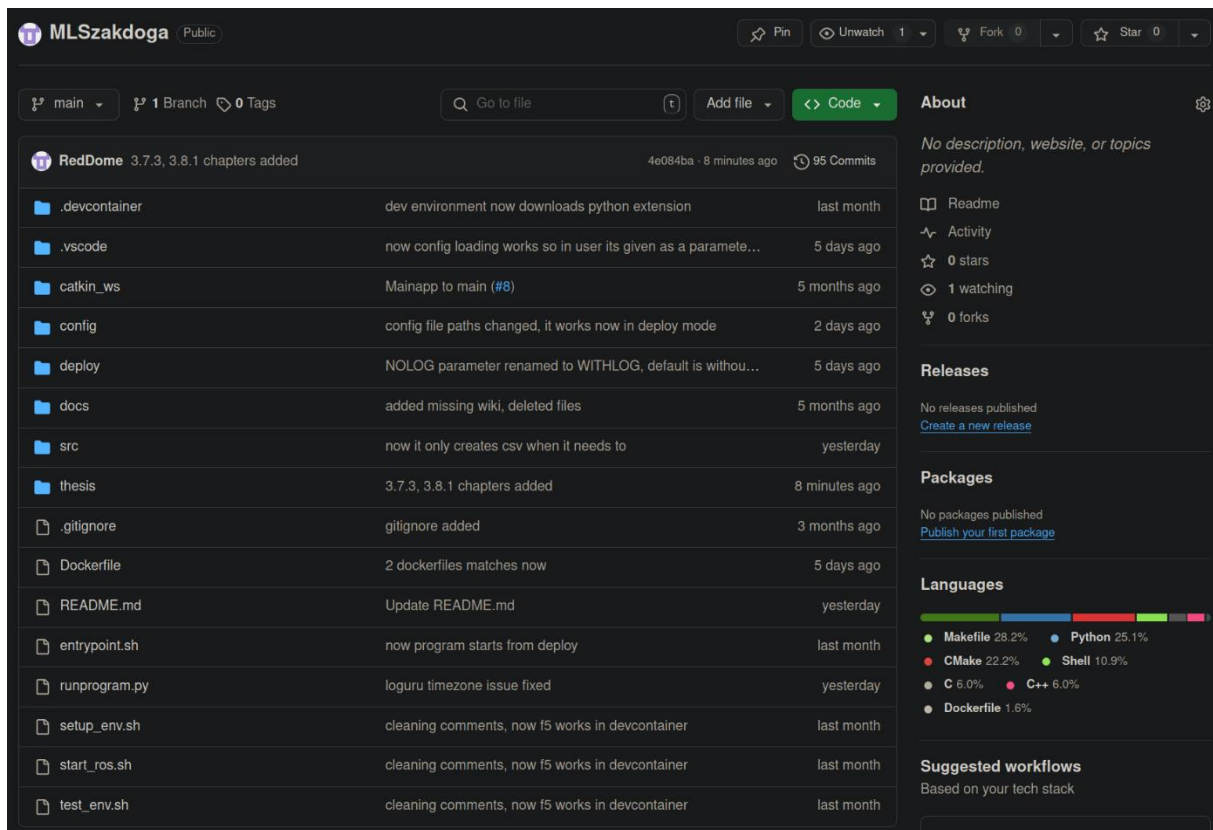
### 3, Program:

- test\_learnFunctionWithSave: Learn funkció megfelelően működik, és a végén lementi az adatokat a program
- test\_saveDataFunction: SaveData funkció megfelelően működik

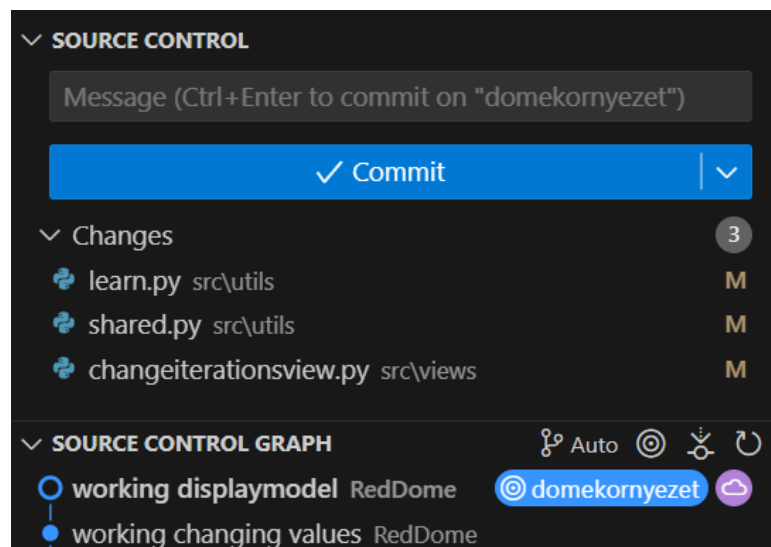
## 3.11. Verziókezelés

A projekt verziókezelése a Github webes, GIT verziókezelőjén valósult meg. Ennek az volt a nagy előnye, hogy nem csak lokálisan, hanem bárhol is el tudtam érni a projektemet. Nem beszélve a biztonsági mentesről, amit biztosít ez a weboldal. Nagy előnye volt még az oldalnak, hogy a konzulensnek sokkal könnyebben el tudtam küldeni a munkám állapotát. A Visual Studio Code alapvetően támogatja a verziókezelőket, főleg a GIT alapúakat. A szoftveren belül lehet megnézni minden kódot, amit feltöltöttél a felületre, miközben ad arra lehetőséget, hogy megnézd milyen módon írtad át a kódot ahhoz képest, ahogy a verziókezelőben van feltüntetve. Lehetséges a Visual Studio Code-on belül commitolni, pusholni és minden parancsot megcsinálni, amelyet terminálból csinálnál alapvetően, megkönnyítve ezzel a fejlesztőknek a feladatát.





43. ábra: A projekt „Github” oldala

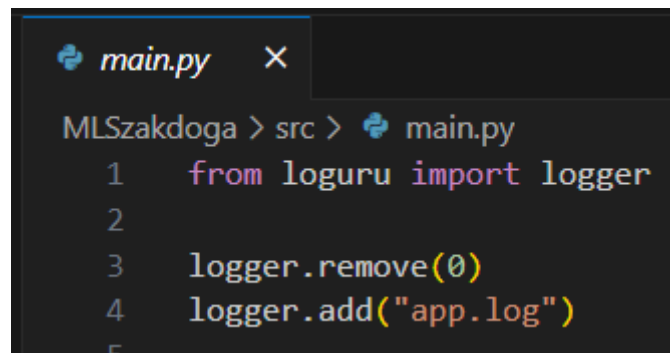


44. ábra: A Visual Studio Code verziókezelési felülete

### 3.12. A program logolása

Minden program esetén elengedhetetlen a megfelelő logolás, amely teljesül itt is. A *resources* mappában található *log* mappában lesz található a *app.log* nevű fájl, ebben lesz található az összes log sor, amelyet a program létrehozott. A Python által biztosított logolási

könyvtárban voltak olyan problémák, amelyek miatt amellett döntöttem, hogy egy másik logolási könyvtárat fogok használni. A loguru [9] nevű nagyon népszerű logolási könyvtárat használtam a program során. A loggert nagyon könnyű beállítani, csak 2 sor kóddal már be is lehet állítani. Utána pedig minden egyes fájlban, ahol használni akarjuk, ott csak meg kell importálni az osztályt, és a main metódusban definiált tulajdonságokat meg fogja jegyezni, így az egész programon keresztül azonos beállításokkal fog futni a logolás.

A screenshot of a code editor window titled 'main.py'. The editor shows the following Python code:

```
MLSzakdog > src > main.py
1  from loguru import logger
2
3  logger.remove(0)
4  logger.add("app.log")
5
```

45. ábra: Loguru beállítása ennyire egyszerű a programunkban

### 3.13. A program konfigurálása

A programban úgy éreztem, hogy a konfiguráció beolvasás biztosításával könnyebben hordozható lesz a program, és könnyebb lesz a használata, személyre szabása. A program indulásakor meg tudjuk adni milyen konfigurációs fájlt szeretnénk használni, alapvetően biztosítok a felhasználónak 3 alapbeállítást, aminek az előnye az, hogy aki nem ért hozzá, annak semmit se kell rajta módosítania, hanem csak elindítja a megfelelő funkció konfigurációval és már működik is a program. Ezek adatok feldolgozása a *processConfigFile.py* fájlban fognak megtörténni.

```

processconfigfile.py x
src > config > processconfigfile.py > processConfigFile
1 import yaml
2 import utils.commonvalues as cm
3 from utils.startfunction import startFunction
4 from loguru import logger
5
6 DEFAULTS = {
7     'FunctionName': 'None',
8     'XGoalPosition': 2.0,
9     'YGoalPosition': 2.0,
10    'LearningModel': 'Test',
11    'Length' : 10000,
12    'SaveDataAfterFinished' : False,
13    'ModelPath': '/workspaces/MLSzakdoga/resources/models/PP0/10000.zip',
14    'CsvFilePath': '/workspaces/MLSzakdoga/resources/processedData/tensorboard_data.csv',
15    'LogFolder': '/workspaces/MLSzakdoga/resources/logs/PP0_0'
16 }
17
18 def processConfigFile(path):
19     with open(path, 'r') as f:
20         data = yaml.full_load(f)
21
22     functionName = data.get('FunctionName', DEFAULTS['FunctionName'])
23
24     if functionName in ("None", ""):
25         logger.error("FunctionName is not defined, check the config file! Program will be exiting now!")
26         raise ValueError(f"FunctionName is not defined!")
27
28     cm.setFunctionName(functionName)
29
30     if functionName == "SaveData":
31         processSaveData(data)
32         startFunction()
33         return
34
35     if functionName == "Capture":
36         processCaptureData(data)
37         startFunction()
38         return
39
40     if functionName == "Continue":
41         processContinueData(data)
42
43     cm.setXGoal(data.get('FunctionProperties', {}).get('XGoalPosition', DEFAULTS['XGoalPosition']))
44     cm.setYGoal(data.get('FunctionProperties', {}).get('YGoalPosition', DEFAULTS['YGoalPosition']))
45     cm.setLearningModel(data.get('FunctionProperties', {}).get('LearningModel', DEFAULTS['LearningModel']))
46     cm.setLength(data.get('FunctionProperties', {}).get('Length', DEFAULTS['Length']))
47

```

46. ábra: processConfigFile.py kinézete

A fájl elején megadok alapértékeket, így a hibás indítás esetén is, legalább valami el fog tudni indulni, minden hibáról logolni fog a program és megtekinthető lesz a log fájlban. Ezután szétszedem, hogy funkciókhoz képest, mely értékeket állítsa be a program. Miután sikeresen beolvassa a fájlokat, azután a program a megfelelő funkciót el fogja indítani.

## **4. Összefoglalás**

A program célja egy olyan program létrehozása volt, amely a felhasználónak lehetőséget ad arra, hogy könnyen tudjon tanítani robotokat a Gazebo program segítségével. Véleményem szerint ez tökéletesen sikerült, és mindenfajta korosztálynak és tudással rendelkező embernek is létrehoztam egy könnyen használható programot. A fejlesztőknek is létrehoztam egy olyan alapot, amelyet nagyon könnyen lehet tovább fejleszteni és a megfelelő fejlesztői környezet létrehozásával, egy könnyű környezetet kap képhez a fejlesztő, ahol a megfelelő tesztelésnek és dokumentálásnak köszönhetően, könnyen megtudja érteni a program működését erről az oldalról is.

### **4.1. További fejlesztési lehetőségek**

Jelenleg a komplikált környezet miatt, a program csak Linux rendszerben működik, ezért mindenképpen lehetne egy későbbi fejlesztés célja a platformfüggetlenség, ezzel minél több embernek eljuttatva a programot.

Ezen kívül a programon belül is voltak fejlesztések, amiket egy későbbi időpontban meg lehetne valósítani, például legyen lehetőség arra, hogy a felhasználó a saját robotját/világát használja a programon belül, esetlegesen egy világkreáló, amiben a felhasználó tud magának csinálni világokat, ahhoz képest, hogy milyen tanulást szeretne elvárni a robottól.

A teljesítmény növelésének érdekében mindenképpen egy későbbi fejlesztés témája, hogy a program a GPU támogatott Gazebo verziót használja, ezzel jelentősen meggyorsítaná az összes funkcióját, így a felhasználó kevesebb idő alatt többet tudna elérni a programban.

Egy képernyőfelvevő funkció is lehetne egy későbbi fejlesztés része, amely szorosan együttműködik a programmal, és így könnyen használható lenne a felhasználónak.

## 5. Irodalomjegyzék

- [1] „Applied machine learning in cancer research: A systematic review for patient diagnosis, classification and prognosis” [Online]. Available: <https://pmc.ncbi.nlm.nih.gov/articles/PMC8523813/> [Hozzáférés dátuma: 2024. 11. 21.]
- [2] „8 ways Amazon is using generative AI to make life easier” [Online]. Available: <https://www.aboutamazon.com/news/innovation-at-amazon/how-amazon-uses-generative-ai> [Hozzáférés dátuma: 2024. 11. 21.]
- [3] „Email that keeps your private information safe.” [Online]. Available: <https://safety.google/gmail/> [Hozzáférés dátuma: 2024. 11. 21.]
- [4] „Stable Baselines3 – Train on Atari Games” [Online]. Available: [https://colab.research.google.com/github/Stable-Baselines-Team/rl-colab-notebooks/blob/sb3/atari\\_games.ipynb](https://colab.research.google.com/github/Stable-Baselines-Team/rl-colab-notebooks/blob/sb3/atari_games.ipynb) [Hozzáférés dátuma: 2024. 11. 22.]
- [5] „System Requirements for Gazebo: A Comprehensive Guide” [Online]. Available: <https://magnetica42.rssing.com/chan-80216447/article16.html> [Hozzáférés dátuma: 2025. 03. 04.]
- [6] „WinRAR download free and support: WinRAR” [Online]. Available: <https://www.winrar.com> [Hozzáférés dátuma: 2025. 03. 28.]
- [7] „7-Zip” [Online]. Available: <https://www.7-zip.org/> s [Hozzáférés dátuma: 2025. 03. 28.]
- [8] „Visual Studio Code” [Online]. Available: <https://code.visualstudio.com/docs/setup/linux> [Hozzáférés dátuma: 2025. 04. 16.]
- [9] „Loguru – Python logging made (stupidly) simple” [Online]. Available: <https://github.com/Delgan/loguru> [Hozzáférés dátuma: 2025. 03. 04.]