Szabó Attila  
WC7YLI

Tánc lépés kitaláló program.

Dokumentáció

Tartalom

[Bevezetés 1](#_Toc514101771)

[Környezet választás a feladat megvalósításához 2](#_Toc514101772)

[A lépések létrehozása 3](#_Toc514101773)

[Diagramm blokk létrehozása és a PMT file blokká alakítása 6](#_Toc514101774)

[A táncolás megvalósítása 8](#_Toc514101775)

[A kitalálás létrehozása 10](#_Toc514101776)

[Python kód 11](#_Toc514101777)

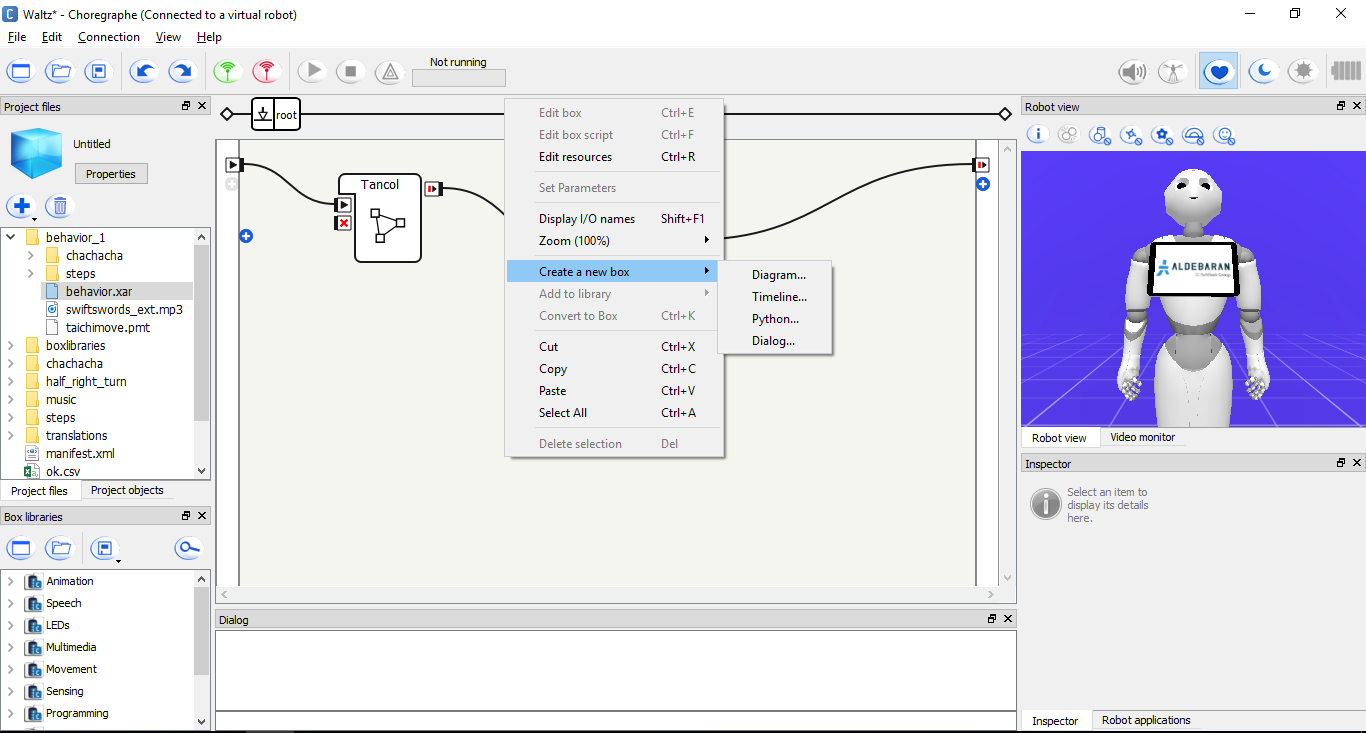
[A blokk diagramm további része. 12](#_Toc514101778)

[A végeleges kód, a fizikai roboton történő futtatáshoz. 13](#_Toc514101779)

# Bevezetés

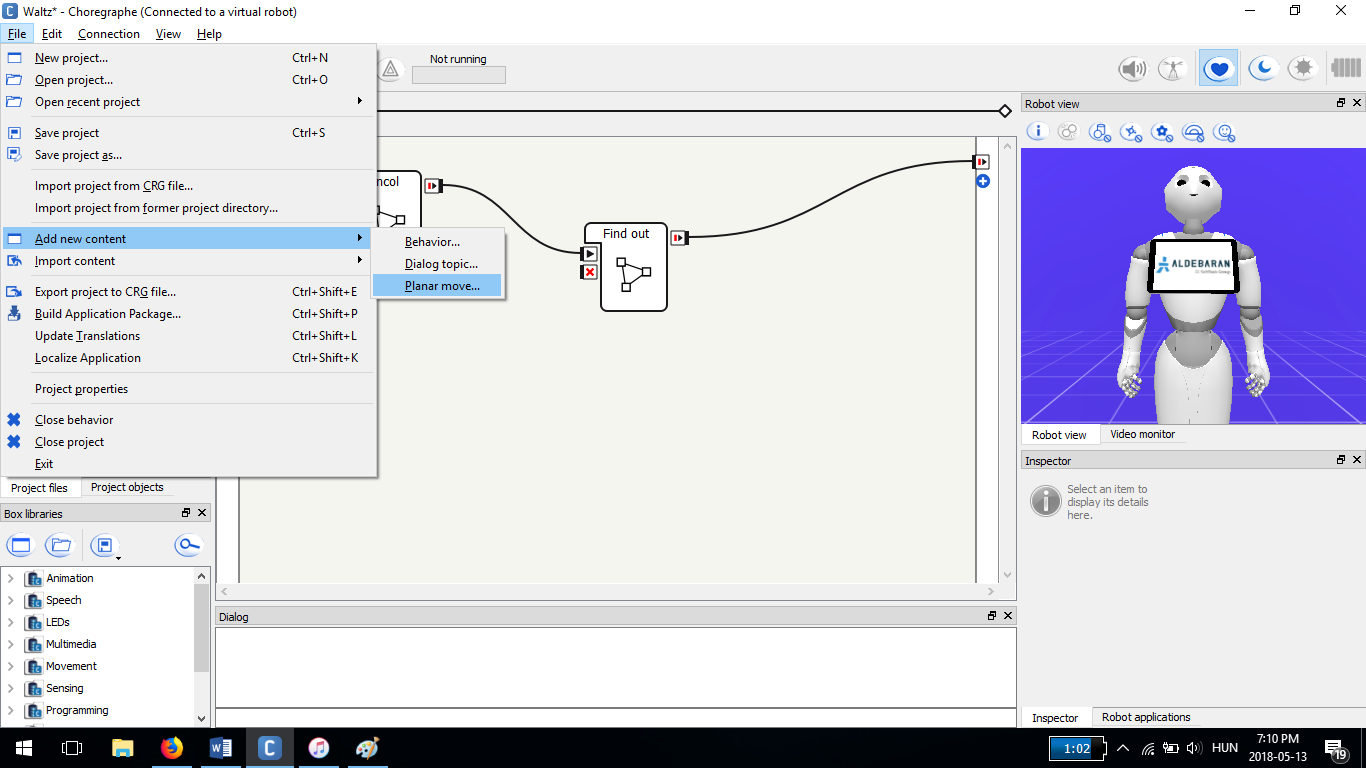
Azért választottam ezt a témát, mert elsős korom óta folyamatosan kapcsolatom van a táncművészetekkel. Időről időre új táncokat tanulok és úgy gondoltam érdekes elképzelés lenne egy láb nélküli robotra átültetni a táncok lépéseit. Azért is érdekes a kihívás, mert a motorok gyorsítása és lassítása nagymértékben eltér az emberi lábtól. A célom az volt, hogy ennek ellenére is találjak, olyan jellegzetes tér bejárásokat, amik 1-1 klasszikus vagy latin táncra jellemzőek. Választásom, így esett a bécsi keringő jellegzetes forgására, a tangó gyors hosszú menetére, közepén egy helyben forgással és a chachacha hand in hand mozdulatára.

# Környezet választás a feladat megvalósításához

A programomat oktató jellegűnek szánnom a programozás terén épp úgy ahogyan a tánc lépés felismerés terén, ezért szeretném a leg egyszerűbben reprodukálható és átlátható módon létrehozni a lépéseket a Pepper robot gyártója által biztosított Choreographe környezetben. A megvalósítást szeretném az átláthatóság és reprodukálás érdekében a lehető legtovább blokkos környezetben tartani. A lehetőségeim feltérképezéséhez, szükséges megismerkednem a létrehozható doboz típusokkal. Ezek megtekinthetők a szerkesztő felületen jobb egérgombbalé kattintva a new boksz lenyíló menü alatt (Lásd: 1. ábra).

Ábra: 1

Az itt található dobozok között találjuk a pythont, ami klasszikus python kód fogadására alkalmas, sajnos teljesen nem tudom megkerülni a programkód írást, de a fent említettek miatt minimalizálom. Lehetőség van idővonalas programozásra, ahol a robot mozdulatait időre lehet rögzíteni. Ez a forma sokkal inkább a kéz, törzs és fejtartásra koncentrál a térbeli elmozgatás helyett, ami nekem nem megfelelő, mert a bejárt út alapján akarom felismertetni a táncot. A dialóg boxban lehetőségünk van adott felismert szövegekhez, adott válaszokat rendelni, amikből további dialógusokat származtathatunk, egy speciális kód struktúrával. Ilyen probléma nem merült fel az eredeti elképzelésembe, így ezen box leírása sem kerül részletezésre. A kezelő felületnek van egy egyedi mozgás leíró nyelve, ami a Planer Move névre hallgat, pmt filet készít és grafikus szerkesztő felülettel rendelkezi. Itt felülnézetbe kapjuk a robot helyzetét és lehetőségünk van megadni a kezdő és a cél pozíciókat, valamint a robot kiindulási és érkezési irányszögét. Ennek elérése a File menü, add new content menüpontja alatt található Planar Move néven (Lásd: Ábra: 2).

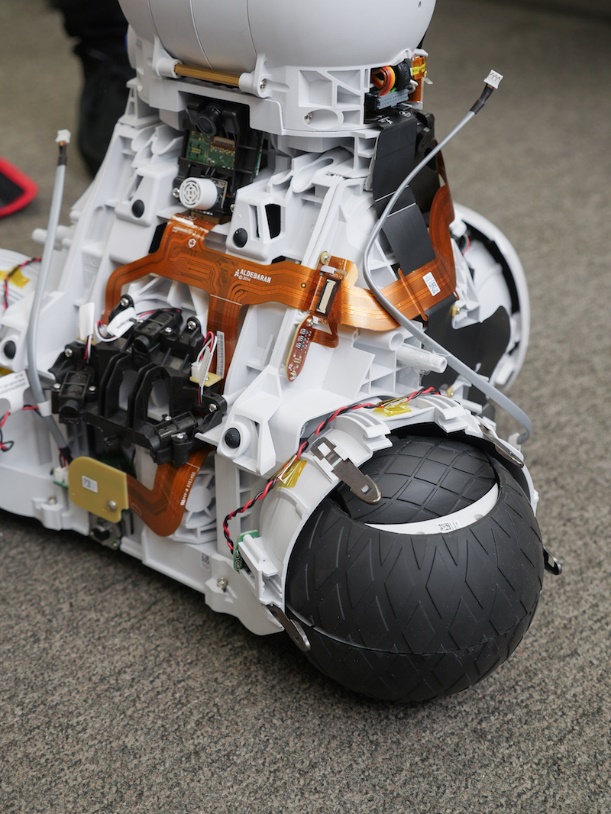


Ábra: 2

A programom megvalósításához a az új diagram boxot fogom alkalmazni, hogy alprogramokkal átláthatóbbá tegyem a teljes kódot. A lépések létrehozásához a planar move funkciót fogom használni. Sajnos 1-2 részfeladatnál megkerülhetetlenné válhat a python box így ez is a felhasznált elemek közé kerül.

# A lépések létrehozása

A fentiekben leírt alapján és az ábra 2 mintája alapján megnyitom a planar move szerkesztőjét. Induláskor meg kell adnunk a mozgásnak a nevét, amiből a gép származtatja a mozgás file nevét is. A szerkesztő felület közepén pirossal jelölten látható a robot mozgásának vég állapota. Amennyiben ezt az egérrel elmozgatjuk, lehetőségünk van új cél pozíciót beállítani, ilyenkor az alatta elhelyezkedő kiindulási állapot zölden látszódik. Egy kör jelképezi mindig a robotot, felülnézetből és egy nyíl jelzi, hogy merre áll a felsőtestével.

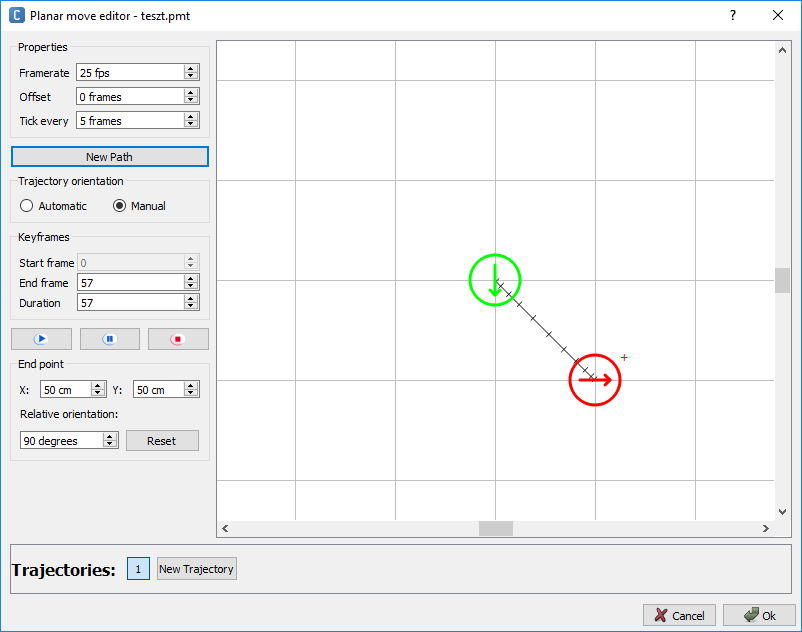
Erre azért van szükség, mert a szerkesztő nem csak elmozdítani, de elforgatni is képes a térben a robot talpazatát, köszönhetően a lábaban található 3 független keréknek, amiket, egy külső merőleges kerékbe, integráltak. Így a külső kerék lehetővé teszi a belső kerék irányba fordítását, míg a belső kerék forgatása a robot tényleges megmozdítását végzi. A kerekek mozgatását feltehetően léptető motorok végzi. Eme kerék felépítését az Ábra: 3 tartalmazza.

Ábra: 3  
(Forrrás: <http://tech.nikkeibp.co.jp/dm/english/NEWS_EN/20150703/426144/?SS=imgview_en&FD=-103440614> )

A szerkesztő felületen, a cél pozíció elmozdítása után a robot, útja is láthatóvá válik. Az utat jelző vonalon, kereszt vonalak sűrűsége jelzi a robot sebességét, ugyanis a robot egységnyi idő alatt érkezik egyik keresztvonaltól a következőhöz, így látható a gyorsulása és a lassulása. A vonal megfogható és elmozdításával, a robot más íven járja be a távolságot.

A mozgások tervezésekor, elhanyagolhatatlan, a tér méretének ismerete, ennek megfelelően a terület egy négyzethálós felbontásban látható, ahol a vonalak a valós tér méreteit jelzik. 50 cm azaz fél méter van a vonalak között. Azért nagyon fontos figyelembe vanni a tér méreteit, mert az Ábra 3 képen látható, hogy a robot talpazata rendelkezik távolság szenzorokkal és amennyiben ezek a mozgási területen belül akadályt érzékelnek, a mozgás nem indul el.

A szerkesztő felület bal oldalán lehetőség van számokkal is bevinni a cél pozíció adatait, ezzel pontosabb mozgás érhető el, mint az egérrel történő elmozgatáskor. A kezelőfelület valós időben intuitívan reagál. Oda-vissza szinkronizálva a számadatokat és a grafikus tervet. A 4-es ábrán a kezelőfelület látható X:50 cm Y:50 cm elmozdulással és 90 fokos elfordulással. A fordulás kapcsán érdemes megemlíteni, hogy a robot az óramutató mozgásával ellentétesen fogadja a pozitív, és megegyezően a negatív fokokat.



Ábra: 4

A program automatikusan kiszámolja a minimálisan szükséges mozgási időt, amit keyframenek nevez. Ez az érték változtatható a baloldalon, nagyobb keyframe esetén a robot mozgása lassabb lesz. Természetesen a keyframe számot elosztva a framrateel, megkapjuk a mozgásidőtartamát. Tehát, ha a mozgást dupla frame számra emelem, de a frameratet is leduplázom, a mozgás sebessége azonos lesz. A tick every-vel meghatározza, hogy a grafikus felületen, minden alapból 5. frame legyen ábrázolva.

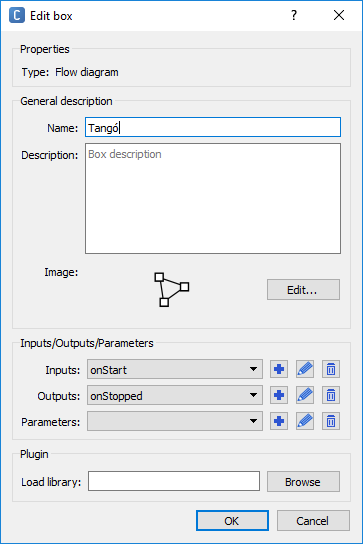
A lejátszás gomb megnyomásával, megjelenik, egy fehér pont a vonalon és leszimulálja valós időben a robot mozgását az adott pályán.

Az alsó menüsorral lehetőségünk van több trajectories összefűzésére. Ennek lényege, hogy a cél pozícióban látható robot pozíció, lesz az új trajectory kezdőpozíciója. Ezzel bonyolult mozgássorok is kialakíthatók, egyszerű egymás utánni tervezéssel. Az ok gombra kattintáskor a pmt file mentésre kerül.

Ezzel a módszerrel létrehoztam a táncok jellegzetes lépéseit, majd elkezdtem a láthatóság miatt a blokká alakításukat.

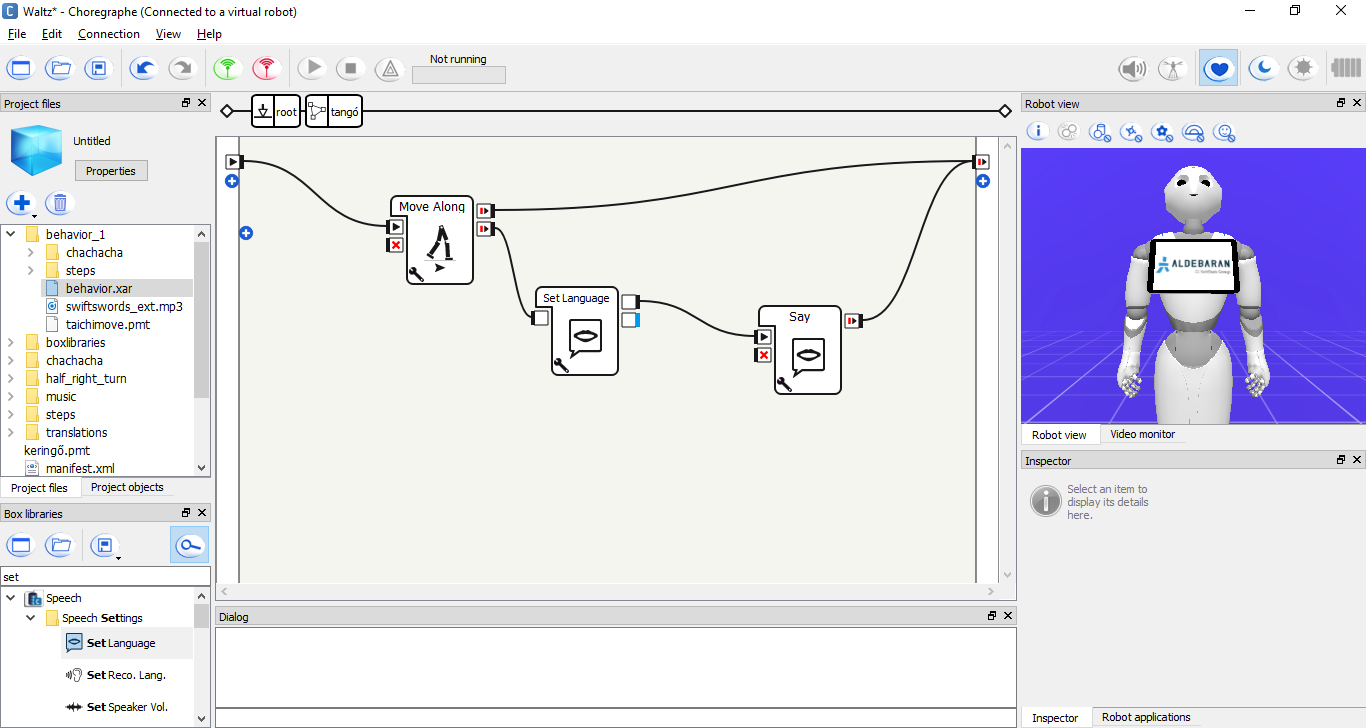
# Diagramm blokk létrehozása és a PMT file blokká alakítása

A mentés után a pmt állomány baloldalt a project file területen lesz látható, ahonnan drag’n’drop módszerrel a felületre helyezve Move Along blokká. A szerkesztő felületen jobbegérgombos menüből kiválasztom a create new box lehetőséget és azon belül a diagrammot (A menüszerkezetet Lásd ábra: 1). Ekkor megjelenik a boksz létrehozás párbeszéd ablak, ahol lehetőségünk van beállítani a bemenő és kimenő adatokat, a blokk ikonját, nevét, leírását, paramétereit. Az egyszerűbb érthetőség miatt, elnevezem tangónak a blokkot, így később könnyebb lesz a diagrammon azonosítani.(Lásd: ábra: 5)



Ábra: 5

Sajnálatos módon itt drag’n’drop módszerrel, nem lehet a move along dobozt az új tangó diagramm dobozba helyezni, ezért törlöm a szerkesztő területről. Ezt követően duplán a tangó blokkra kattintva megnyitom, és ide húzom, be a mozgás filet. Majd a bemenetét az indítóra és a kimenetének a success gombját a stop ikonra kötöm, míg a failuer gombját egy setlanguageblokkba, amit egy say blokkba kötök, amit a bal alul található box liberaries területen kerestem a nagyító ikonnal. Erre azért van szükség, hogy hely hiányában a robottól kapjunk egy visszajelzést. A blokk bal alsó sarkábabn található kulcsra kattintva, megadható a kimondandó szöveg. Érdemes angolul vagy az alábbi listából választott nyelven fogalmazni, mert a robotnak csak korlátozott számú nyelvhez van beszéd szintetizátora. A setLanguage blokkot szintén a kulcsal állítom be. Ennek megfelelően a diagrammom az ábra 6 képen található.



Ábra: 6

A használható nyelve:

* Angol
* Arab
* Brazil
* Cseh
* Dán
* Finn
* Francia
* Görög
* Japán
* Kínai
* Koreai
* Lengyel
* Német
* Norvég
* Olasz
* Orosz
* Portugál
* Spanyol
* Svéd
* Török

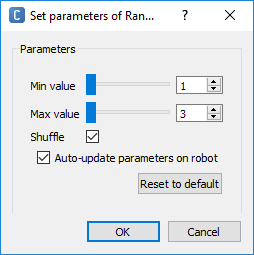
Érdemes megjegyezni, hogy a szerkesztő felület felső részén látható, hogy a fő diagrammon belül milyen blokkban helyezkedünk el.

Ezt követően egy szintel fentebb megyek és a boksz libaries-nél létrehozok egy új libarit a tánc nevével. Ezután blokkra jobb egérgombal kattintva az add to libaryn belül kiválasztom a cél libarit. Természetesen a boksz libaries panelen a mentés gombbal elmentem a módosítást. Innentől kezdve, minden táncom, egy blokként felpakolható a könyvtárból.

A fenti lépések alapján előállítom mind 3 tánc blokkját.

# A táncolás megvalósítása

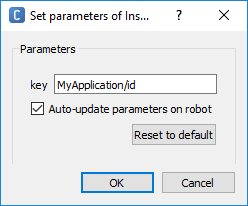
A robot feladatait két nagy részletre lehet bontani. Az első amikor kisorsolja a táncot, eltárolja, a sorsolást, annak érdekében, hogy később ellenőrizhető legyen, a felhasználó jó tippet adott meg vagy sem. Majd elindul a táncból létrehozott blokkot. Az első ábrán látható módon. A programomat azzal folytatom, hogy megkeresm a random blokkot, ezzel sorsolva táncot. A kulcsra kattintva beállítóm, hogy 1,2,3 számokat sorsolhat. Ezt a 7. ábra mutatja.



Ábra: 7

Következő lépésként, egy insert data blokkal felmentem az adatot a memóriába, a kulcsak beállítva anevét „MyApplication/id”-ra, (lásd: 8. ábra) hiszen az ellenőrzéskor szükségem lesz a tánc sorszámára. Majd ismételten egy random blokkot használok, hogy létrehozzak egy egész számot ami 0. Ehhez a random beállításainál minimum és maximum értéknek is a 0-át állítom be és ismét insert data blokkal elmentem a „MyApplication/try” névvel ellátott területre, ez fogja jelezni, hogy az adott táncra mennyi próbálkozást adott a felhasználó.

Mivel a tánc kiválasztásához szükséges a tánc száma, így getdata blokkal vissza hívom az id-t.

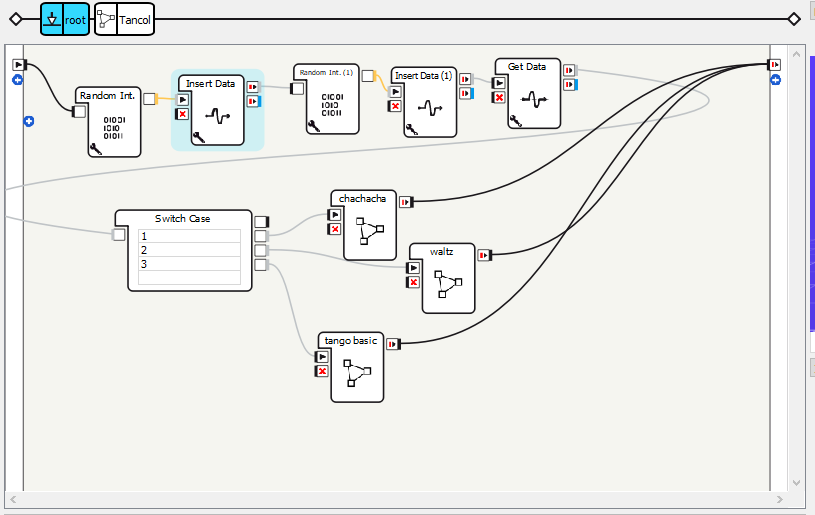


Ábra: 8

Switch Case rendszerrel szétválogatom a táncokat, úgy hogy a blokk soraiba, írom az 1, 2, 3 számokat. Ilyenkor a számhoz létrejön egy kimenet amit az adott tánc blokkjára kötök. A táncból pedig a stopra kötöm a vezérlést. Fontos, hogy mindig megegyező legyen a sorszám a tánccal, így ehhez az alábbi táblázatot készítettem (Lásd 8. ábra). A teljes programrészt lásd a 10. ábrán.

|  |  |
| --- | --- |
| ID | Tánc |
| 1 | ChaChaCha |
| 2 | Keringő |
| 3 | Tangó |

Ábra: 9



Ábra: 10

Egy szinttel feljebb lépek és létrehozom a kitaláláshoz a Find out Diagramm blokkot.

# A kitalálás létrehozása

Az egyszerű virtuális roboton történő teszteléshez, először ebben a diagrammban a startot egy text edit blokkra kötöm, amibe a válasz írható, ezzel biztosítva, hogy a virtuális roboton is tesztelhető legyen a szoftver, annak ellenére, hogy az nem rendelkezik beszéd felismerővel. A későbbiekben a fizikai robotra töltéskor ez kicserélhető egy speach recognison blokkra.

Ezt követően a blokkot egy switch casebe kötjük, amiből 3 kivezetést indítunk, a 9 ábra szerint. Ezek az ágak Insertt data blokkal a „MyApplication/uid” memória területre mentik a felhazsnáló tippét. Ezeket a blokkokat sajnos át kell írni, így a blokkokra kattintva megjelenítjük a python kódjaikat. Ebben a def onInput\_onStart(self, p): függvényt keresük, aminek első sora végén található a p paramétert amit egy statikus számra cserélünk, szintén a 9. ábra alpján. Arra az esetre, ha a felismerés sikertelen vagy nincs a listába a switch case alapértelmezett ágát, egy Say blokkba kötöm, amit a kulcsal beállítok egy olyan szövegre ami megmutatja, hogy a robot nem tudta felismerni a szöveget. Ehhez én az „I can not understand” mondatot választottam.

## Python kód

Sajnos a következő lépésben megkerülhetetlennek találtam a python kódot, mert a memóriából lehívott adatokat kelett összehasonlítani, amihezmegfelelő lenne a blokknézet, ám csak 1 paramétert egy beégetett értékkel lehet összehasonlítnai. Így célszerűbb egyetlen python kóddal lehívni a memóriából az értékeket, összehasonlítnai, megnövelni a próbálkozás számot, visszaírni és a megfelelő kimenetet aktiválni.

A blokk létrehozásakor, az output sorban a plsuz gombbal létrehoztam 3 új „bang” tipusú kiemenetet. Ez egy speciális tipus ami csak itt található, meg és csak arra szolgál, hogy azt a vezélés szálat indítja, visszatérési érték nélkül.

A kód onInput\_onStart(self): függvényében először lehívtam a változókat a memóriábol, egy-egy lokális változóba. Ehhez a memória kezelést kell használni, így még a betöltés során lefutó onLoad(self) függvényben létrehozom a self.memory változót, amely az ALProxy-t hívja meg “ALMemory” paraméterrel. Erre azért van szükség, mert minden speciális a robothoz kapcsolódó, utasítást az ALProxyn keresztül lehet elérni, ám meg kell határoznunk, melyik részét szeretnénk használni, ezzel optimalizálva a memória kezelést. Nállam ez így néz ki: self.memory=ALProxy(“AlMemory”).

Annak érdekében, hogy a programkód többi részét biztosan ne zavarja, az ALMemory meghívása a destruktorban, ami onUnload(self) függvény névre hallgat a self.memory-t none-ra állítom, így self.memory=None.

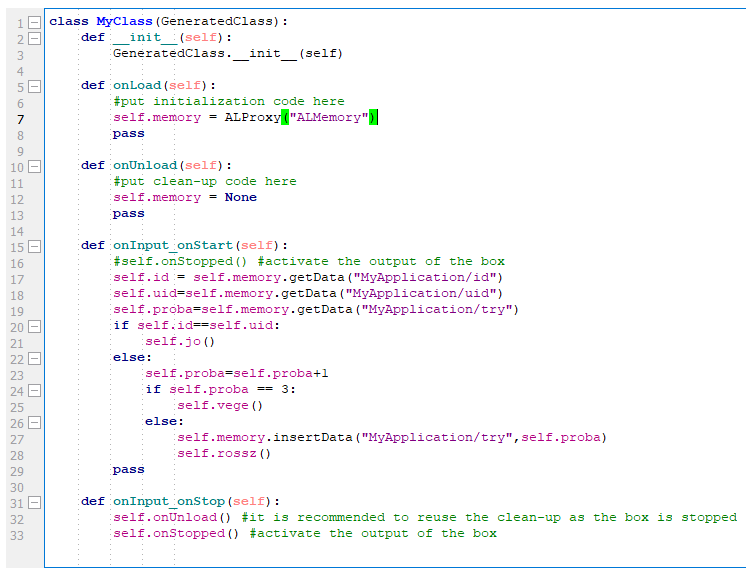
Az adatok lehívás egyszerű sémát követ. Self.adatneve = self.almemorynakadottnev.getData(változónév). Így nállam ez a következő: self.id=self.memory.getData(“MyApplication/id”). Természetesen az összes változómat, így meghívom. Változóim a 11-es ábrában láthatóak.

|  |  |  |
| --- | --- | --- |
| Változó neve | Változó funkciója | |
| MyApplication/id | | A robot által sorsolt tánc azonosítója. |
| MyApplication/uid | | A felhasználó által tippelt tánc azonosítója. |
| MyApplication/try | | A felhasználó eddigi próbálkozásainak száma. |

ábra 11

Ezt követően két egymásba ágyazott kétirányú elégazás (if else) következik. Az első if megvizsgálja, hogy a tippek azonosak-e (uid==id), mert amennyiben igen a válasz, jó és nincs más dolgunk, mint self.jo() függvénnyel aktiválni akimenetnél felvett jo kimenetet. Különben (else) ág esetén a próbálkozás számát megnöveljük slef.try+=1 és újabb kétirányú elágazást nyitunk. Ezen elágazás megvizsgálja, hogy a probálkozásaink száma azonos 3-la, amennyiben igen self.veg() kimenetet aktiválja, ha nem akkor self.memory.insertData(„MyApplication/try”, self.try) sor felviszi a próbálkozások új számát a memóriába és meghívja a self.rossz() kimenetet.

Ezen kódom a 12-es ábrán tekinthető meg teljes egészében.



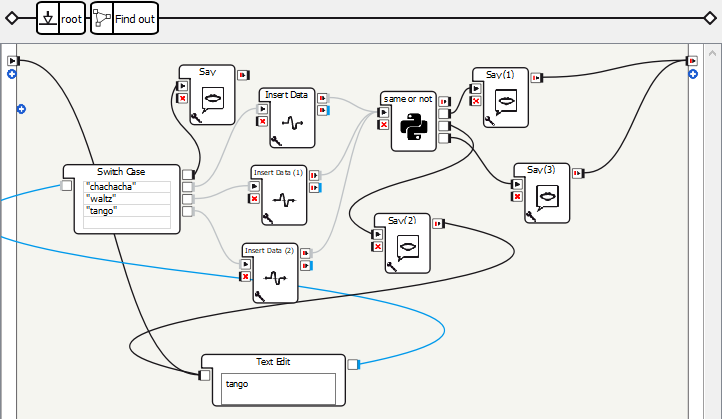
ábra 12

## A blokk diagramm további része.

Jó válasz esetén kimendatom egy say blokkal, hogy gratulálok, „Congratulation!”, míg vég kiemnet esetén egy say blokkal kimondatom, hogy túl sokszor próbálkozott „You tried too much”.

Rossz válasz esetén szintén egy say bekötése a célszerű, ám jelezzük a felhasználónak, hogy újra próbálkozhat „No! Please try again!”. Ezt vissza is kötjük, a szöveg dobozra. Ezzel ki lehet próbálni, hogy 3 alkalommal történő rossz válasz esetén, leáll-e a futás.

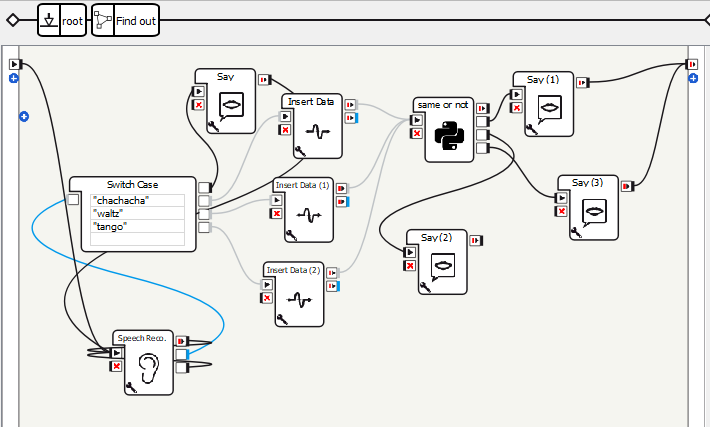
Ateszteléshez használatos blokk diagrammomat a 13-as ábra mutatja.



ábra 13

# A végeleges kód, a fizikai roboton történő futtatáshoz.

Itt már csak annyi a dolgunk, hogy a text blokkot kicseréljük, egy speach recognition (beszéd felismerő) blokkra. Ennek a blokknak az onNothin és a Stop kimenetés a saját startjára érdemes visszakötni, míg a felismerést a switchbe. Figyeljünk rá, hogy a defoult ágon lévő say visszakötésre kerüljön a szöveg felismerőre és a diagramm indítás blokk is be legyen kötve a felismerésre, ennek ábrája a 14-es.



ábra 14