Multidiszciplináris tudományok, 11. kötet. (2021) 3 sz. pp. 11-19 **https://doi.org/10.35925/j.multi.2021.3.2**

LEGO MINDSTORMS EV3 ROBOTOK VISELKEDÉS-ALAPÚ PROGRAMOZÁSA

Truzsi Patrik
BSc hallgató, Miskolci Egyetem, Általános Informatikai Intézeti Tanszék
3515 Miskolc, Miskolc-Egyetemváros, e-mail: truzsi.patrik@gmail.com

Baksáné Varga Erika
egyetemi docens, Miskolci Egyetem, Általános Informatikai Intézeti Tanszék
3515 Miskolc, Miskolc-Egyetemváros, e-mail: vargaerika@iit.uni-miskolc.hu

Absztrakt

A programozási nyelvek fejlődésének egyik irányzata a viselkedéstípusok alkalmazása annak érdekében, hogy biztosítsák a nagyméretű, kommunikáció-alapú rendszerek pontosabb működését. A viselkedéstípusok interfészeket, kommunikációs protokollokat, szerződéseket és koreográfiát foglalnak egységbe. Sikeres alkalmazásukhoz ismerni kell, hogy az adott programozási nyelvben milyen nyelvi elemekkel írhatók le és meg kell tanulni integrálni ezeket a nyelvben definiált más szintaktikai egységekbe. Jelen kutatás célja a viselkedéstípusok Java nyelvi integrációjának vizsgálata egy Lego Mindstorms EV3 robot működését leíró program segítségével.

Kulcsszavak: *viselkedés-alapú programozás, viselkedéstípus használata Javaban, robot programozás*

Abstract

A recent trend in the evolution of programming languages is to use behavioral types to ensure the accurate operation of large-scale communication-based systems. Behavioral types define a set of interfaces, communication protocols, contracts and choreography. Their successful application presumes that the programmer knows how they are represented in a given programming language and how to integrate them with other programming constructs. The present paper aims at studying the integration of behavioral types in Java by means of a Lego Mindstorms EV3 robot program.

Keywords: *behavior-based programming, behavioral types in Java, robot programming*

1. Bevezetés

A Neumann-elvű számítógépek központi egysége szekvenciális sorrendben dolgozza fel a programok utasításait. Az egymagos processzorokban az utasítások egymás utáni végrehajtása a végletekig optimalizált, aminek következtében azok teljesítményét sikerült jelentősen feljavítani. Mivel a programozók számára ez a legegyszerűbb modell, a programozás alapjainak oktatásában és a hagyományos algoritmusok hatékonyságának méréséhez továbbra is ezt használjuk, és az imperatív programozási nyelvek is ezt a modellt támogatják.

A modernebb programozási nyelvek a többszálú, párhuzamos modellű programok fejlesztéséhez is nyújtanak támogatást, vagyis olyan eljárásokat, amelyek segítségével váltani lehet az egyes programszálak között. Az egymagos processzor úgy tudja megvalósítani a párhuzamos programszálak futtatását, hogy megadott pontokon az egyik szál futását megszakítva átlép a másik szál feldolgozására, majd újra vissza az elsőre. Amikor több magon futnak a programszálak, a szálváltás mechanizmusa egysze-

rűbb és gyorsabb a kód végrehajtása. Ezért napjainkban az egyik legfontosabb elvárás a programozási nyelvekkel és a hatékony algoritmusokkal szemben a többmagos processzorok lehetőségeinek kihasználása.

Egy viselkedés-alapú program moduljai különböző forgatókönyveket implementáló programszálak, amik tulajdonképpen egy alkalmazás használati eseteinek nagyon részletes leírásai. Minden egyes viselkedésminta egy adott helyzetben (előre meghatározott feltételek teljesülése esetén, illetve valamely esemény bekövetkezésének hatására) végrehajtásra kerülő eseménysort reprezentál. Ezek a programszálak futási időben összefonódnak és így alakul ki az egész rendszer integrált viselkedése [1].

A viselkedés-alapú modellt és szoftverfejlesztési módszertant a játékprogramok programozásához fejlesztették ki, de robotvezérlő programok írásakor is alkalmazható [2]. A robot elvárt viselkedésének egyes komponenseit egymástól független modulokban implementáljuk, majd futási időben egy közös végrehajtási mechanizmus összefűzi a programszálakat és azok bemeneti követelményeit, vagy magasabb szintű prioritásokat figyelembe véve kiválasztja azt az eseményszálat, amit az adott helyzetben aktiválni fog. Egy viselkedés-alapú program modellezéséhez az LSC (Live Sequence Charts) grafikus formalizmus használható [3].

A kutatás célja a viselkedés-alapú programozási modell megismerése és összevetése a szekvenciális programozási modellel. Ehhez megépítésre került egy Lego Mindstorms EV3 futballjátékos robot, aminek a működtető programját mindkét programozási modell alkalmazásával implementáltuk.

2. Live Sequence Charts

Az LSC diagram alapja a Message Sequence Chart (MSC) [4], ami az üzenetváltással kommunikáló rendszerkomponensek együttműködését leíró szabványos ábrázolási mód és az iparban széleskörben alkalmazzák folyamatok és objektumok együttműködésének leírására. A gyakorlat igazolta, hogy az MSC a rendszerfejlesztés korai szakaszában nagyon hasznos eszköz, de a fejlesztés későbbi fázisaiban korlátokba ütközünk. Az eredeti modellben a felhasználói esetek forgatókönyveit írjuk le: azaz, ha egy adott feltétel igaz lesz, akkor bekövetkezhet egy forgatókönyv. Később azonban már azt szeretnénk leírni, hogy amikor a feltétel igaz, akkor be kell következzen a forgatókönyv szerinti működés, tehát szeretnénk előírni kötelező viselkedés módokat.

Az MSC másik hiányossága objektum orientált programtervezés során merül fel. A viselkedés-alapú programtervezés korai fázisaiban rendszerint a folyamatok és objektumok közötti kapcsolatokat definiáljuk időben lineáris vagy kvázi-lineáris módon. Később azonban azt szeretnénk, hogy előálljon egy adott folyamat vagy objektum teljes viselkedés-alapú specifikációja, vagyis az adott folyamat vagy objektum viselkedését szeretnénk látni minden lehetséges futási forgatókönyv és feltétel teljesülése esetén. Ez utóbbi szemléltetésére MSC diagram helyett állapotgép diagramot [5] használnak.

Ezen problémák kiküszöbölése révén, a viselkedés-alapú programtervezés támogatására született meg az MSC leírónyelv kiterjesztéseként az LSC (Live Sequence Chart) [3], amelyet mi is alkalmaztunk a futballjátékos robot viselkedés-alapú programjának tervezése során.

3. A Lego Mindstorms EV3 robotok programozása

A LEGO Mindstorms EV3 robotok a LEGO által 2013-ban piacra dobott robotikai építőkészletből konstruálható szerkezetek. A LEGO a programozásukhoz is lehetőséget kínál a LEGO Mindstorms EV3 szoftver segítségével. Ebben a szoftverben EV3-G grafikus programnyelven írhatjuk meg a programokat, amelynek használata könnyen elsajátítható Kiss Róbert könyve [6] segítségével.

A szoftver a programozási eszközöket és a robot hardverelemeit blokkokkal reprezentálja. Többek között találhatunk blokkokat a motorok vezérléséhez, programszerkezeti egységekhez, szenzorok működtetéséhez. Saját blokkok létrehozására is van lehetőség. Amennyiben olyan szenzort használunk, amelyet nem tartalmaz a szoftver alapértelmezettként, úgy lehetőségünk van letölteni szenzorokat kezelő programozási modulokat.

A program az egyes blokkok megfelelő sorrendben történő egymáshoz láncolásával épül fel. Az EV3-G programozási nyelv lineáris és többszálú programozásra is használható. Más moduláris programozási nyelvekhez hasonlóan a szintaktikai hibák nagy részével itt sem kell foglalkozni. A programozáshoz szükséges idő nagy részét az algoritmus megalkotása teszi ki. A grafikus programnyelv sajátja, hogy a programkód sokkal áttekinthetőbb marad.

Az EV3 robot számítógéphez csatlakoztatása után elérhetővé válik a roboton megtalálható programok listája, a szabad memória, illetve a robotra felszerelt érzékelők. Az elkészített programot fel kell tölteni a robotra, ahol ezt követően lehet futtatni.

A LEGO Mindstorms EV3 robotok programozhatók Java nyelven is. Ehhez kínál megoldást a leJOS (Lego for Java Operating System). Használata az Eclipse, a megfelelő verziójú Java JDK és az Eclipse leJOS plugin telepítése után lehetséges. Emellett az EV3 téglába egy SD-kártya behelyezése szükséges, amelyre előzetesen fel kell telepíteni a leJOS operációs rendszert, ami magába foglalja a Java futtató környezetet.

A Java objektum orientált programozási nyelv, amelyen szekvenciális programot, illetve többszálon futó, a viselkedés-alapú modellt követő programot is írhatunk. A robot funkciói osztályok segítségével érhetők el, amelyek különböző csomagokban találhatók meg. Ezeket a csomagokat és osztályokat a program elején kell importálni. A legfontosabb csomagok (*www.lejos.org/ev3/docs/*):

- lejos.hardware.ev3
- lejos.hardware.motor
- lejos.hardware.port
- lejos.hardware.sensor
- lejos.hardware.lcd
- lejos.robotics.navigation

Szekvenciális programozás során a programrészek meghatározott sorrendben futnak le. Ilyen programok írásához nincs szükség további, speciális csomagokra. Viselkedés-alapú programok írásához viszont sajátos szoftverfejlesztési módszertant kell követni [7] és a BPJ függvénykönyvtárat kell felhasználni [2].

Az egyes osztályokon belül implementálni kell a *lejos.robotics.subsumption.Behavior* interfészt. Ennek az interfésznek három metódusa van: *boolean takeControl()*, *void action()* és *void suppress()*.

A *takeControl()* logikai visszatérésű, így a metódus igaz vagy hamis értékkel térhet vissza. Ha a *takeControl()* értéke igaz, akkor az adott viselkedésnek át kell vennie a robot irányítását. A vizsgálandó érték többféle lehet. Például igaz-e, hogy az ultrahangos távolságérzékelő által mért távolság kevesebb lett egy feltételként megadott értéknél vagy benyomódott-e a robotra szerelt ütközésérzékelő.

Az *action()* azt a kódot tartalmazza, amely a viselkedés *takeControl()* metódusának igaz visszatérési értéke után hajtódik végre. Az *action()* tartalma lehet bonyolult, de lehet nagyon egyszerű is, mint például hang lejátszása. Ha az *action()* végrehajtódik, a függvény visszatér.

A *suppress()* metódusba írt kódnak a feladata a jelenleg aktív viselkedés befejezése. Ezért célszerű a gyors kilépésre törekedni, például megszakítást okozó logikai változók beállításával.

Ezenkívül a main függvényben példányosítani kell a *lejos.robotics.subsumption.Arbitrator* osztályt. Az Arbitrator objektum kezeli a viselkedések indítását és leállítását. Egy viselkedés indításakor meg-

hívja az *action()*-t, leállításkor pedig a *suppress()*-t. Az egyes viselkedések objektumait tömbben tárolja a prioritási sorrend alapján. Az Arbitrator feladatai a következők [8]:

1. Meghatározza azon viselkedések között a legmagasabb prioritásút, amelyek *takeControl()* metódusa igaz értékkel tért vissza.
2. Megszakítja az aktív viselkedést, ha annak prioritása alacsonyabb, mint a legmagasabb prioritás.
3. Amikor egy viselkedés befejeződik, meghívja a legmagasabb prioritású viselkedés *action()* metódusát.

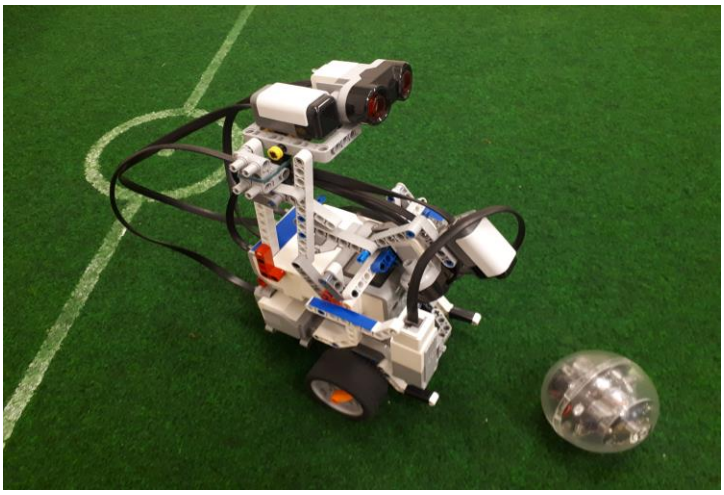
4. A futballjátékos robot

4.1. Konstrukciós leírás

A futballjátékos robot alapja a 300MHz-es, Linux alapú intelligens tégla (brick). Ehhez vannak csatlakoztatva a működéshez szükséges szenzorok és motorok. A mozgást két interaktív, kerekekkel felszerelt nagy szervomotor (EV3 Large Servo Motor) végzi. A robot elején helyet kapott egy közepes szervomotor (EV3 Medium Servo Motor), melynek forgási tengelye a motor tengelyével párhuzamos. Ennek feladata a labda begyűjtése és elütése egy felszerelt építőelem segítségével.

A robot négy érzékelővel rendelkezik. Minden érzékelő meghatározott célú. Egy infravörös kereső (HiTechnic NXT IRSeeker V2) pásztázza a környezetet, infravörös sugarakat keresve. Ez a szenzor határozza meg a labda irányát. Az első ultrahangos távolságérzékelő (EV3 Ultrasonic Sensor) feladata a labda távolságának mérése. Egy tájoló (HiTechnic NXT Compass Sensor) gondoskodik a robot pontos irányba állásáról. A második ultrahangos távolságérzékelő (EV3 Ultrasonic Sensor) pedig a futballpályán belüli távolságok méréséért felel.

A futballjátékos robot további tartozéka egy infravörös jelet kibocsátó labda (HiTechnic Infrared Electronic Ball). Az ebből kibocsátott jeleket veszi a már említett infravörös kereső.



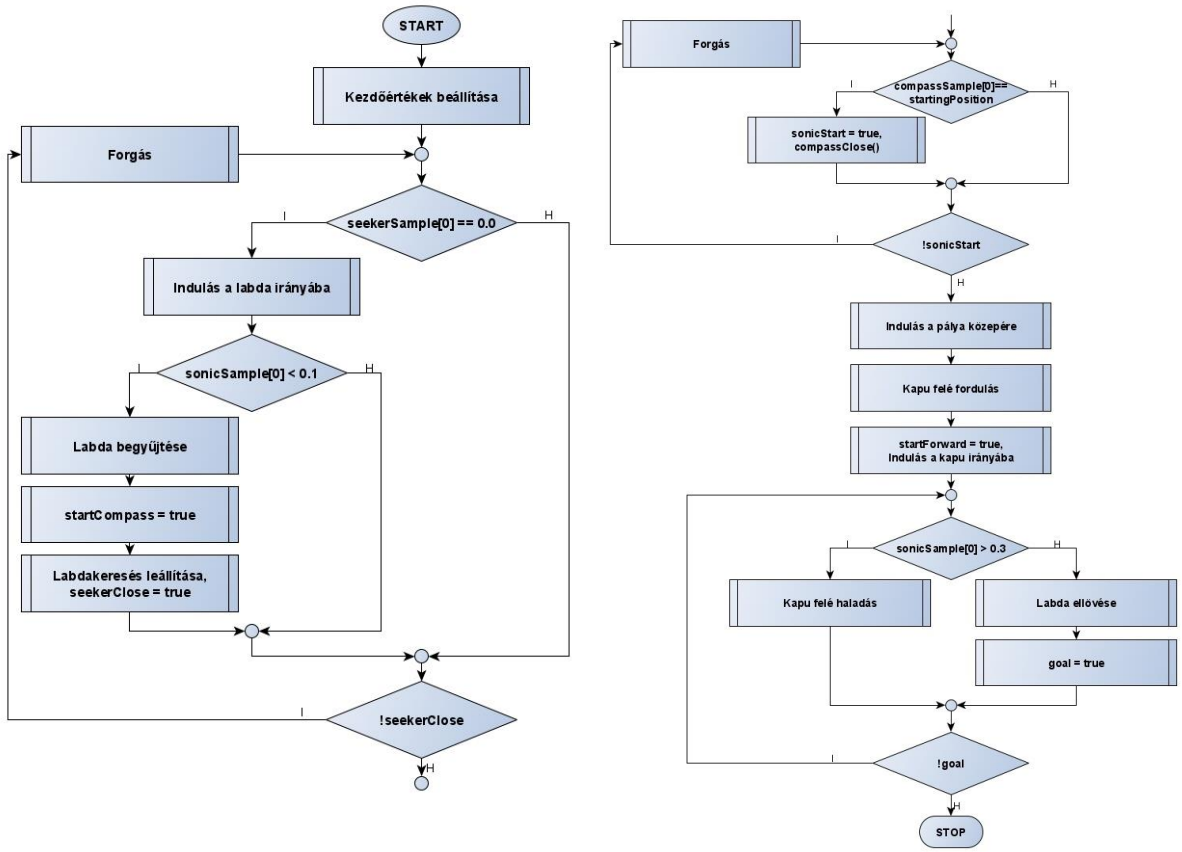
1. ábra. *A futballjátékos robot és az infravörös jeladó labda*

4.2. Szekvenciális program

A program futtatása előtt a robotot a futballpálya közepére kell helyezni középkezdéshez, szemben a kapuval. A program elején négy logikai változó (seekerClose, startCompass, startForward, sonicStart)

értéke hamisra van állítva. A futballjátékos robot szekvenciális programjának futása a felhasználó gombnyomásával indul. Ekkor a tájoló mintát vesz, majd a robot aktuális kezdőpozícióját eltárolja a float típusú startingPosition változóban. Ezután 90 fokos fordulás következik a pálya szélességének meghatározásához. A fordulás után az egyik ultrahangos távolságérzékelő mintát vesz a távolságról és ezt az értéket tárolja el a float típusú startingDistance változóban.

Az első szelekció a seekerClose változó értékét vizsgálja. Ez a változó engedélyezi az infravörös kereső működését. Amíg a változó értéke hamis, addig az infravörös kereső által vett mintát (seekerSample) összeveti a 0.0 értékkel. Ez azt jelenti, hogy a labda pontosan a robot előtt van. Ekkor a robot elindul előre, tehát a labda irányába. Amennyiben a seekerSample tömbben tárolt szám nem egyenlő a 0.0 értékkel, úgy a robot forogni kezd, és ezt addig végzi, amíg a szenzor által vett minta nem lesz 0.0.



2. ábra. *A futballjátékos robot szekvenciális programjának folyamatábrája*

A labda közelségét az egyik ultrahangos távolságérzékelő figyeli, így ha a robot pontosan a labda mellé ért, begyűjti azt a közepes szervomotor segítségével. Itt lesz igaz értékűre állítva a seekerClose és a startCompass változó. A seekerClose igazra állításával az első szelekció hamis ágán halad tovább a program, tehát leellenőrzi, hogy a startCompass igaz-e, vagyis hogy indulhat-e a következő szenzorra megírt programrész. A compassSample tartalmát összeveti a startingPosition változóban tárolttal.

Ha nincs egyezés, akkor addig forog, amíg nem találja meg a megfelelő pozíciót. Ha egyezés van, akkor a robot megáll, majd elfordul 90 fokkal, hogy megint a futballpálya oldalsó fala felé nézzen.

Ekkor következik a második ultrahangos távolságérzékelőtől származó érték vizsgálata. Ha ez az érték nagyobb, mint a kezdő távolság + 0.1, akkor a robot előre megy, hogy csökkentse a távolságot. Ha kisebb, akkor hátra megy, hogy növelje azt. Így találja meg a pálya közepét. Visszafordul 90 fokkal, hogy újra a kapu felé álljon.

A program utolsó része a gólszerzés. Amennyiben a robot 30 centiméterre vagy annál messzebb helyezkedik el a kaputól, úgy elindul előre. Amint a 30 centiméteres határ alá ér, megáll és a közepes szervomotor használatával kapura lövi a begyűjtött labdát. A program működését a 2. ábrán látható folyamatábra szemlélteti.

4.3. Viselkedés-alapú program

A futballjátékos robot viselkedés-alapú programjának, illetve szekvenciális programjának futásakor megfigyelhető, végrehajtott cselekvéssorozat azonos. A különbség az eltérő programozási módszertan alkalmazásában rejlik. Amíg a szekvenciális program esetén a programíráskor meghatározott sorrendben hajtódnak végre az egyes programrészek, addig a viselkedés-alapú program a megadott viselkedések közül futási időben választja ki a soron következőt. A viselkedés-alapú program 3. és 4. ábrán látható LSC modellje a PlayGo szoftverrel [9] készült.

Minden egyes viselkedést reprezentáló osztály tartalmaz egy logikai változót, amely a folyamatban lévő viselkedés egy másik viselkedés által történő megszakítását szemlélteti. Tehát amíg a logikai változó értéke hamis, addig lehet a vezérlés a futó viselkedésnél. Amint egy viselkedés vezérlésátadási feltétele igazzá válik, akkor a megszakított viselkedés *suppress()* metódusa hajtódik végre, amely átállítja a megszakítást szemléltető logikai változó értékét igazra.

A futballjátékos viselkedés-alapú programja szintén a felhasználó gombnyomására vár. Miután ez megtörténik, a szekvenciális programhoz hasonlóan itt is a tájoló által vett minta tárolódik el, majd egy 90 fokos fordulást követően a program elmenti az ultrahangos távolságérzékelő által mért adatokat.

Ezután elkezdődik az első viselkedésbe programozott forgás. Ez a forgás addig tart, amíg egy viselkedés meg nem szakítja.

Minden viselkedésnek van egy feltétele, amely teljesülése esetén megkapja a vezérlést. Az infravörös kereső viselkedési osztályának indulási feltétele a labda 0.0 irányban történő észlelése. Amint ez teljesül, a robot megközelíti a labdát a két nagy szervomotor segítségével.

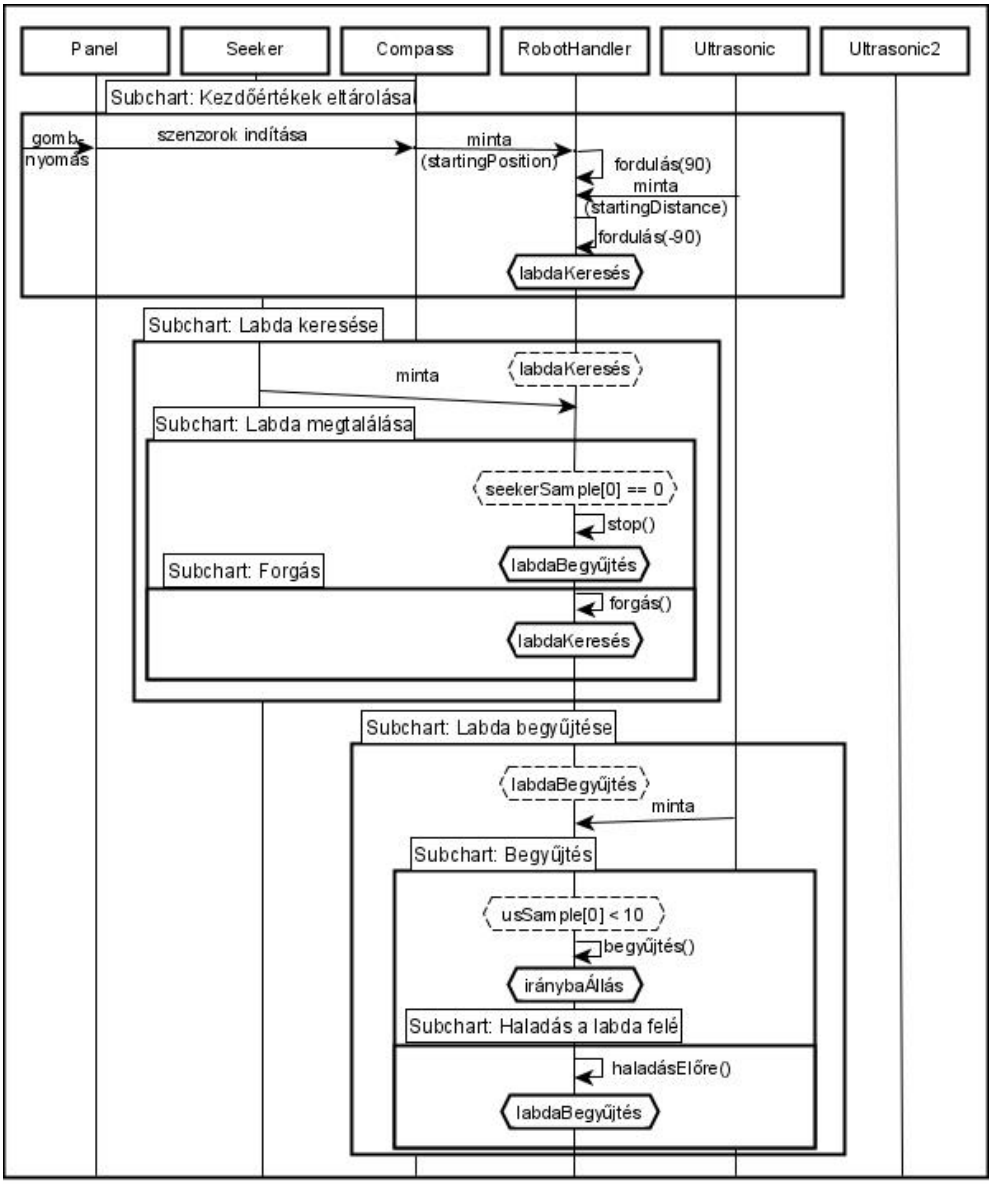
Az első ultrahangos távolságérzékelő viselkedése akkor kapja meg a vezérlést, ha a szenzor 10 centiméternél kevesebbet mér. Így az előre haladás során, amint a labda pontosan a robot előtt helyezkedik el, az ultrahangos távolságérzékelő viselkedés aktívvá válik. Ekkor történik a labda begyűjtése a robot elejére szerelt közepes szervomotor segítségével. Ezután egy korábban hamis logikai változót igazra állítunk, amely hatására a tájoló működésbe léphet.

A tájoló által elvégzendő feladat két viselkedésre van osztva. Az egyik akkor hajtódik végre, ha a tájoló által mért érték nem egyezik meg a program elején lemért és elmentett számmal. Ekkor a robot elkezd körbe forogni. Ezt a másik, tájolóhoz köthető viselkedés szakítja meg, ha a mért érték megegyezik a kezdeti értékkel. Egy 90 fokos elfordulást végző metódus fut le. A második ultrahangos távolságérzékelőt indító logikai változó igaz értéket vesz fel.

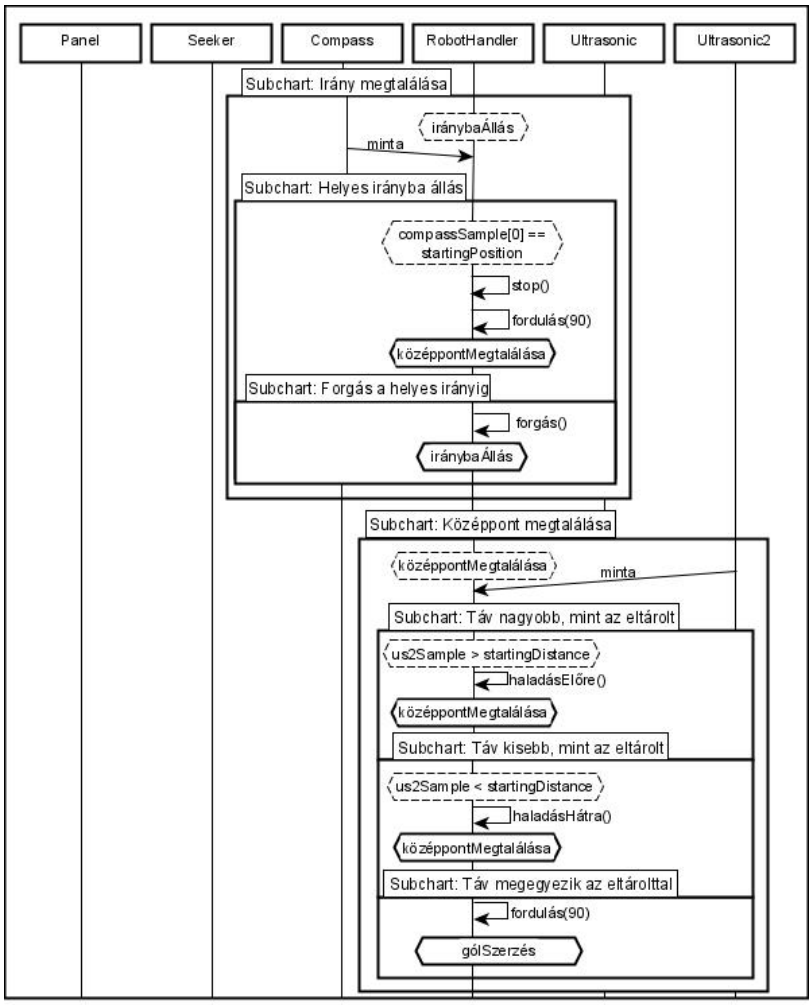
A második ultrahangos távolságérzékelő feladata a pálya közepének megtalálása. Három osztályra van bontva. Az első reprezentálja azt az esetet, amikor a robot a pálya oldala és a középpont felé néz, de a középponttól messzebb helyezkedik el. Ekkor előre megy, hogy csökkentse a távolságot. A má-

sodik eset, amikor a robot a pálya közepe és a pálya oldala között van. Ekkor tolat, hogy növelje a távolságot. A harmadik eset, amikor a robot pontosan a megfelelő helyen tartózkodik, tehát az ultrahangos távolságérzékelő által mért adat megegyezik a program elején lemért távolsággal. Ebben az esetben a robot megáll, majd 90 fokos fordulást tesz az ellenkező irányba, tehát visszaáll a kapu irányába.

Az utolsó rész, amely a gólszerzést vezérli, két viselkedési osztályra van osztva. Az egyik osztály akkor lesz aktív, ha az ultrahangos távolságérzékelő által mért távolság 30 centiméternél több, ilyenkor előre halad. Ha 30 centiméternél közelebb van a kapuhoz, akkor a másik viselkedés válik aktívvá, amely a labda ellövését végzi.



3. ábra. *A futballjátékos robot viselkedés-alapú programja (1. rész)*



4. ábra. *A futballjátékos robot viselkedés-alapú programja (2. rész)*

5. Összefoglalás

A robotok működését vezérlőszabályok és viselkedéstípusok definiálásával írhatjuk elő. A gyakorlatban egy adott feladat megvalósítása rendszerint három fő lépésre bontható: cél objektum megkeresése, az objektumra irányuló tevékenység meghatározása, majd elvégzése. Egy végrehajtandó tevékenységsorozat többféleképpen is implementálható. Szekvenciális feladatmegoldás során egy adott tevékenység végrehajtása akkor kezdődik el, amikor a megelőző sikeresen befejeződött. Egy robot működése közben azonban gyakran adódnak váratlan események, amik megszakítják a működését és amikre nem tud reagálni, ha a szekvenciális programjában nincs az adott helyzetre vonatkozó feltételvizsgálat. Ezért célszerű a robot működését kiváltó eseménytől függő tevékenységsorozatokra bontani, majd egy vezérlő programban előírni, hogy milyen feltétel milyen viselkedési formát aktivál. Ily módon a robot működése során egy adott tevékenységsorozat addig fut, amíg valamilyen feltétel teljesülése ki nem vált egy másik viselkedést. Természetesen ennél a megoldásnál is a programozó felelőssége felkészíteni a robot programját az esetlegesen előálló különböző események és feltételek vizsgálatára, de

olyan eset nem fordulhat elő, hogy egy adott helyzetben a robot „nem tudja, mit csináljon”. Amíg nem lesz új viselkedés aktív, addig az előzőt folytatja. A két megközelítési mód implementációja és futása közötti különbség vizsgálata volt a cél ebben a tanulmányban. Ehhez megépítettünk egy futballjátékos Lego EV3 robotot és implementáltuk a szekvenciális és viselkedés-alapú programját Java-ban. A forráskódok az alábbi linken érhetők el: https://github.com/TruthP/Lego_robot. A szekvenciális programban a robot által elvégzendő tevékenységsorozatok egy futtatható osztály különböző metódusaiba kerültek, melyek előírt hívási sorrendjét a main metódus definiálja. A viselkedés-alapú programban minden tevékenységsorozatot külön osztályban definiáltunk, amelyek a main metódusban lettek példányosítva. Ezek között a prioritási sorrendet az Arbitrator objektumban rögzítjük, de ezt a sorrendet felülírja egy viselkedés aktívvá válása.

Általános esetben, és amikor váratlan helyzet nem befolyásolja a robot működését, mindkét program ugyanúgy fut le. Ha azonban a labdát a pálya közepére helyezzük induláskor, a viselkedés-alapú program eltárolja a középpontot, aztán nem kezdi el keresni a labdát, hanem egyből a labda begyűjtése tevékenység aktiválódik. Ez a program tehát képes alkalmazkodni az aktuális körülményekhez, szemben a szekvenciális programmal, amely minden esetben ugyanazt a tevékenységsorozatot hajtja végre.

6. Köszönetnyilvánítás

A cikkben ismertetett kutató munka az EFOP-3.6.1-16-2016-00011 jelű „Fiatalodó és Megújuló Egyetem – Innovatív Tudásváros – a Miskolci Egyetem intelligens szakosodást szolgáló intézményi fejlesztése” projekt részeként – a Széchenyi 2020 keretében – az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Irodalom

[1] Harel, D., Marron, A., Weiss, G.: *Behavioral programming*, Communications of the ACM, Vol. 55. No. 7. (2012) **https://doi.org/10.1145/2209249.2209270**

[2] Harel, D., Marron, A., Weiss, G.: *Programming coordinated behavior in Java*, (2010) **https://doi.org/10.1007/978-3-642-14107-2_12**

[3] Damm, W., Harel, D. LSCs: *Breathing life into message sequence charts*, J. on Formal Methods in System Design, 19 (1): (2001) pp. 45-80. **https://doi.org/10.1023/A:1011227529550**

[4] Harel, D., Thiagarajan, P. S.: *Message sequence charts*, In: Lavagno L., Martin G., Selic B. (eds) UML for Real. Springer, Boston, MA., 2003. **https://doi.org/10.1007/0-306-48738-1_4**

[5] Friedenthal, S., Moore, A., Steiner, R.: *Modeling event-based behavior with state machines*, Ch.11 in A practical guide to SysML – The System Modeling Language, The MK/OMG Press, (2015) pp. 273-294. **https://doi.org/10.1016/B978-0-12-800202-5.00011-4**

[6] Kiss, R.: *A MINDSTORMS EV3 robotok programozásának alapjai*, National Instruments Hungary Kft., H-Didakt Kft., 2014.

[7] Ancona, D. et al.: *Behavioral types in programming languages*, Foundations and Trends in Programming Languages Vol. 3. No. 2-3. (2016) pp. 95-230. **https://doi.org/10.1561/2500000031**

[8] leJOS EV3 API documentation, http://www.lejos.org/ev3/docs/ (utolsó elérés: 2020. 11. 18.)

[9] Harel, D. et al.: *PlayGo: Towards a comprehensive tool for scenario based programming*, 2010 ASE’10, September 20-24, Antwerp, Belgium, ACM 978-1-4503-0116-9/10/09