Robotika az általános iskolában és a RoboMind programozási környezet

Bernát Péter

bernatp@inf.elte.hu
ELTE IK

Absztrakt. Az informatikán belül a robotika témakörének tanítása a közoktatásban egyre népszerűbb: segítségével közérthető és gyakorlati problémák tűzhetők ki, amelyek megoldása mégis összetett lehet. Cikkemben bemutatom az általános iskolába szánt oktatási robotok általános jellemzőit és néhány ismertebb képviselőjét, majd pedig feladatok megoldásán keresztül a RoboMinddal teljesíthető oktatási célokat. A RoboMind programozási környezetet egy rövid szakaszban külön ismertetem.

Kulcsszavak: RoboMind, robotika, algoritmizálás, programozás

1. Bevezetés

Az informatikán belül a robotika témakörének tanítása a közoktatásban egyre népszerűbb: segítségével közérthető és gyakorlati problémák tűzhetők ki, amelyek megoldása mégis összetett lehet.

Az általános iskolai robotikatanítás legfontosabb célja a robotika alapjainak és egyszerűbb problémáinak megismertetése, és a robotok programozásán keresztül az algoritmikus gondolkodás fejlesztése. A programozást – további oktatási célként vagy motivációs jelleggel – az elektromechanikai ismereteket igénylő robotépítés is kiegészítheti.

Cikkemben bemutatom az általános iskolába szánt oktatási robotok általános jellemzőit és néhány ismertebb képviselőjét, majd pedig feladatok megoldásán keresztül a RoboMinddal teljesíthető oktatási célokat. A RoboMind programozási környezetet egy rövid szakaszban külön ismertetem.

2. Az általános iskolába szánt oktatási robotok

2.1. Általános jellemzők

Az általános iskolásoknak szánt oktatási célú robotok segítségével egyszerűbb robotikai problémák modellezhetők és oldhatók meg. Jellemzően egyenesen haladni és kanyarodni képes szerkezetek, amelyekhez érzékelők is tartozhatnak. Az érzékelő nélküli változatok csak a külső hatásoktól függetlenül képesek működni, az érzékelőkkel rendelkezők viszont reagálhatnak a környezet bizonyos változásaira. Az érzékelők közül a távolság-, az ütközés-, a fény- és a színérzékelők a leggyakoribbak.

Néhány fajtájuk készre szerelt és felépítésük nem módosítható, más típusok azonban a rendelkezésre álló elemekből mindig az aktuális problémának megfelelően építhetők össze. Szerelhető robot esetén a robotikai problémák szélesebb körével lehet foglalkozni, és a programozást jelentős mértékben kiegészíti az elektromechanikai ismereteket igénylő robot- és pályaépítés. Nem szerelhető robot esetén a problémák szűkebb körével lehet foglalkozni, és a figyelem legnagyobb része a programozásra irányul.

Más szempontból az oktatási robotok lehetnek valódiak vagy szimuláltak. A valódi robotok szó szerint kézzel foghatók és kapcsolatba kerülhetnek a valós világ tárgyaival, ezért működésük látványosabb. Egyszerre viszont csak kevesen tudják használni őket, és a program és a pálya módosítása körülményes lehet. Habár a szimulált robotok csak a virtuális térben léteznek, mindenki egyszerre használhatja őket, és a pálya és a program gyorsan változtatható és azonnal kipróbálható.

Programozási nyelvük jellemzően automata elvű és eljárásorientált. Az alaputasítások az állapotkomponensek megváltoztatásáért és lekérdezéséért, illetve érzékelőkkel rendelkezés esetén az érzékelők állapotának lekérdezéséért felelősek. A szekvencia mellett a programozási nyelv tartalmazhatja az elágazást és a ciklust, amelyek feltételei a paraméterektől, az állapotkomponensektől és az érzékelők állapotától függhetnek. Ezenkívül támogathatja saját eljárások készítését, és azon belül a paraméterezhető, a visszatérési értékkel rendelkező, és a rekurzívan is meghívható eljárásokét.

2.2. Ismertebb eszközök

2.2.1. Bee-Bot [1]

A Bee-Bot az óvodások és az alsó tagozatosok számára fejlesztett méhecske alakú valódi robot. Előre és hátra haladni egyszerre 15 cm-t, balra és jobbra fordulni 90 fokot képes. Más módon beavatkozni a környezetbe nem tud, érzékelői nincsenek. A mozgató utasítások szekvenciáját a robot hátán elhelyezett gombok megfelelő sorrendű megnyomásával lehet létrehozni. Más nyelvi elemek nem nincsenek.

2.2.2. Pro-Bot [2]

A Pro-Bot a Bee-Botnak elsősorban az alsó tagozatosoknak szánt továbbfejlesztése. A haladás távolsága és az elfordulás szöge pontosan megadható, képes a tartójába helyezett felemelhető és letehető filctoll segítségével a földre terített papírra rajzolni, illetve rendelkezik első és hátsó ütközésérzékelővel, hang- és fényérzékelővel. Ismeri a számlálós ciklust (de az elágazást nem), készíthetők saját paraméter nélküli eljárások, illetve egy-egy eseménykezelő eljárás kitöltésével reagálni lehet az érzékelők jelzéseire.

2.2.3. Codie [3]

A Codie egy előre és hátra haladni, illetve kanyarodni tetszőleges mértékben képes nem szerelhető valódi robot. Rendelkezik egy többféle színben világítani képes LED-gyűrűvel, egy hangszóróval, illetve számos érzékelővel: gyorsulás- és elfordulás-érzékelővel, hang-, távolság- és fényérzékelővel, illetve egy iránytűvel.

Okostelefonokon programozható egy folyamatábraszerű grafikus programozási nyelven. A nyelv tartalmazza a hagyományos vezérlési szerkezeteket, de eljárások nem hozhatók létre.

2.2.4. Lego Mindstorms [4]

A Lego Mindstorms a jelen cikkben említettek közül a legáltalánosabban használható szerelhető valódi robot. Programozható téglájának kimeneti portjaira motorokat, bementi portjaira különböző érzékelőket – például érintés-, távolság-, fény-, szín- és hangérzékelőket – kapcsolhatunk, de emellett hagyományos (passzív) legóelemeket is használhatunk.

A robotot működtető program számítógépről tölthető át. A gyártó által mellékelt programozási környezetben folyamatábraszerűen lehet programozni. Léteznek a hagyományos vezérlési szerkezetek, létrehozhatók paraméterezhető és visszatérési értékkel is rendelkező eljárások, és változók is használhatók.

3. A RoboMind programozási környezet [5]

A RoboMind programozási környezetet a Research Kitchen holland szoftvervállalat fejleszti. A 2007-ben megjelent 1.0-s verziója nyílt forráskódú és ingyenes volt, a 4.0-s változattól (2012) kezdve azonban nem nyílt forráskódú, az 5.0-s verziótól (2013) kezdődően pedig fizetős (az egy éves desktop licence jelenlegi ára 10 euró).

A RoboMind 4.0-s és újabb változatai magyarul is használhatók, a fordítást Magyari-Sáska Zsolttal közösen készítettük.

3.1. A virtuális robot és környezete

A RoboMind virtuális robotja egy négyzetrácsos terepen mozog (1. ábra). A terep bizonyos mezőin áthatolhatatlan falak, mozdítható bóják illetve fekete és fehér festékcsíkok fordulhatnak elő. A robot előre és hátra haladni egész mezőnyit képes, fordulni 90 fokonként tud. Markolója segítségével képes egyszerre egy bóját magával vinni és máshol letenni, illetve azokat korlátlan számban „megenni”. Ecsetével pedig maga is tud fehér vagy fekete festékcsíkot húzni.

Terepek letölthetők a RoboMind honlapjáról, de egy külön szerkesztőprogrammal is készíthetők.

1. ábra: A RoboMind robotja

3.2. A programozási nyelv

A RoboMind programozási nyelve automata elvű eljárásorientált nyelv. Szintaktikája és (eredeti angol nyelvű) kulcsszavai a C++ és Java programozási nyelvéhez hasonlók. A hagyományos vezérlési szerkezetek közül az elágazás, a ciklusváltozó nélküli számlálós ciklus, az elöltesztelő feltételes ciklus és a végtelen ciklus használható. A ciklusok megszakíthatók. Az eljárások rendelkezhetnek bemeneti paraméterekkel és visszatérési értékkel, illetve rekurzívan is meghívhatók. Előállítható véletlenszerű logikai érték, amely egyenlő valószínűséggel lesz igaz vagy hamis.

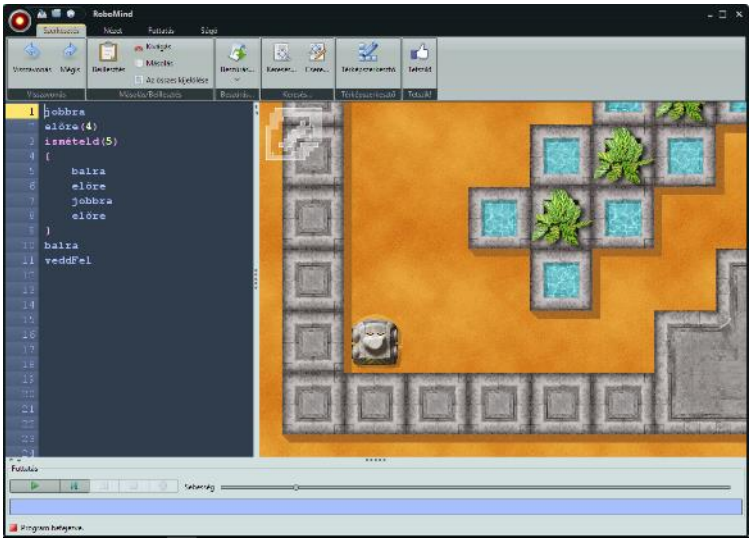
A programozási nyelv használatára számos példa található a **4. részben**.

3.3. A kezelőfelület

A robotot és a terepet mutató fő ablakrészen kívül a RoboMind képernyőjének fontos része a bal oldali programablak és az alsó futtatásvezérlő gombok (2. ábra).

A programablak a programkód sorait automatikusan sorszámozza, és eltérő színekkel emeli ki a különböző kategóriájú nyelvi elemeket. A kulcsszavak kis- és nagybetűket vegyesen tartalmaznak, de az értelmező nem érzékeny a kis- és nagybetűk különbségére. Az utasítások menü-

ből is beszúrhatók. A programkód az új sorban kezdések és a behúzások tekintetében automatikusan formázható.



2. ábra: A RoboMind kezelőfelülete

A programok és a terepek egymástól függetlenül betölthetők, így például kipróbálható ugyanaz a program különböző terepeken, illetve különböző programok ugyanazon a terepen.

A szemantikai hibák keresését megkönnyítheti, hogy a program lépésenként is futtatható, illetve hogy a programkódba töréspontok helyezhetők. Ezenkívül a lejátszási sebesség csökkentése vagy növelése is segíthet valamely hiba okának felderítésében.

Megjeleníthetünk a program hatékonyságát jellemző különböző mutatókat is: a robot által megtett út hosszát, a felderített mezők számát, a szomszédos mezők vizsgálatának számát, illetve a kétféle festék használatának mértékét.

Érdemes még tudni, hogy a mozgáshoz, az ecset- és a markolóhasználathoz kapcsolódó alaputasítások egy távirányító-ablakban is elérhetők, amelyben a kiadott utasítások programja automatikusan megjelenik.

4. Robotikai problémák megoldása a RoboMinddal

A RoboMindban a síkon közlekedni tudó és a környezettel különböző módokon interakcióba lépő robotokkal kapcsolatos egyszerűbb problémák modellezhetők. A továbbiakban feladatokon és a legtöbbjük megoldásán keresztül mutatom be a jellemző problématípusokat, kiemelve mindig a szükséges programozási kellékeket. A feladatok egy része a RoboMind honlapjáról [6] származik, másik részük saját készítésű.

A problémákat érdemes aszerint csoportosítanunk, hogy a megoldásukhoz szükség van-e a robot érzékelőire. Ha a terep kiindulási állapotát teljesen ismertnek tekintjük – például egy pakolási feladatban tisztában vagyunk az átpakolandó bóják pontos helyével és darabszámával –, akkor a robot az érzékelők használata nélkül „vakon” működhet. Ha azonban a terepre vonatkozó ismereteink nem teljesek – például egy útvonalkövetési feladatban tudjuk, hogy a kiindulási

pontból egy fehér festékcsíkkal jelölt útvonal indul, de annak pontos alakját nem ismerjük –, az érzékelőkre is szükség lehet.

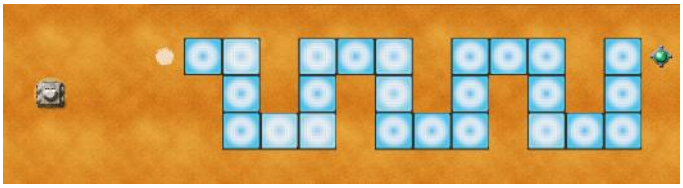
4.1. Érzékelés nélkül megoldható problémák

Az érzékelés nélkül megoldható problémák esetén tehát a terep kiindulási állapotát teljesen ismertnek tekintjük, ennek megfelelően a megoldásukhoz semmilyen feltételre – elágazásra vagy feltételes ciklusra – nem lehet szükség. Jellemzően az ismétlődő tevékenységek egyszerűsíthetők számlálós ciklusok szervezésével.

4.1.1. Útvonal- és területbejárás

A legegyszerűbb érzékelő nélkül megoldható feladatok közé tartozik az előre ismert útvonalak illetve területek bejárása, amelyek esetén legfeljebb bizonyos útvonalrészletek sormintaszerű ismétlődése használható ki.

Első példánkban a robotnak először el kell jutnia a fehér foltra, majd a villogó csempéken kell szlalomoznia egészen a bójáig (amelyet fel is kell vennie) (3. ábra).

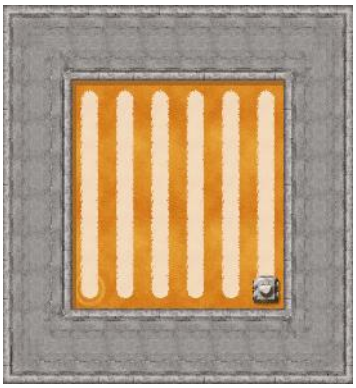


3. ábra: Szlalomozás [6]

Felfedezhető, hogy a szlalomozás első néhány lépésének szekvenciája összesen 3 alkalommal ismétlődik. A feladatot megoldó program a következő:

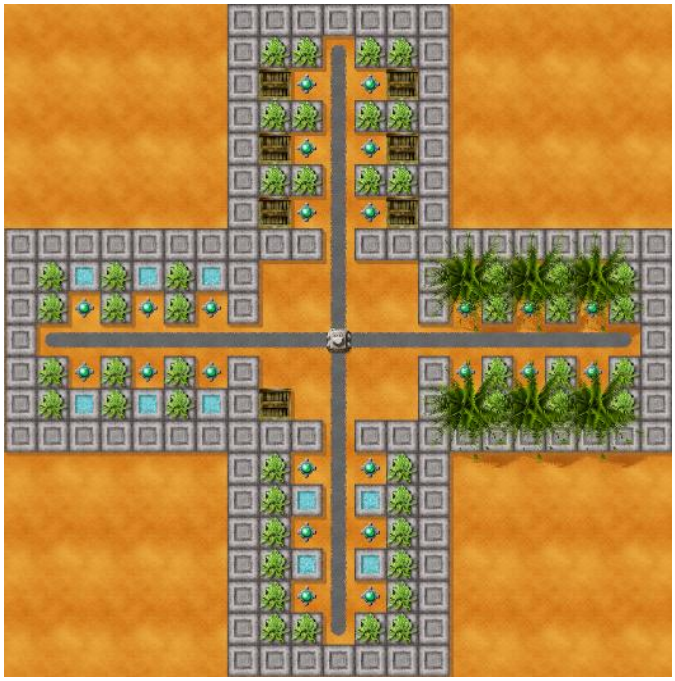
```
# séta a fehér foltig:
előre jobbra előre(3)
# szlalomozás:
ismételd(3)
{
    előre(2) jobbra előre(2) balra előre(2) balra előre(2) jobbra
}
# az elért bója felvétele:
veddFel
```

A következő feladatban egy négyzet alakú területet kell az ábrán látható módon függőlegesen vonalazni (4. ábra). Miután a bal alsó sarokból indulva meghúztunk egy vonalat, és eljuttattuk a robotot a következő kezdőpontba – a kiindulási pont jobb oldali szomszédjába –, ugyanezt a tevékenységet további öt alkalommal megismételhetjük



4. ábra: Padló lefestése vonalasan [6]

Esetenként az ismétlődő részleten belül is felfedezhetünk ismétlődést – ilyenkor egymásba ágyazott számlálós ciklusokat használhatunk. Az alábbi feladatban körben elhelyezett üvegházakban kell beszüretelni a termést (5. ábra).

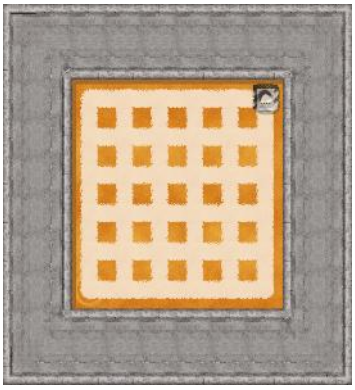


5. ábra: Üvegházak (saját feladat)

Az üvegházak maguk sormintaszerűen helyezkednek el, de egy üvegházon belül a termések begyűjtése is ismétlődő tevékenység. Ennek megfelelően először a felső üvegházban gyűjtjük be a termést a megfelelő műveletsor háromszori ismétlésével, majd visszajuttatva a robotot a kezdőpontba, és a következő üvegház felé fordítva az eddigieket még további három alkalommal végrehajtjuk:

```
ismételd(4)
{
    # egy üvegház bejárása:
    előre(2)
    ismételd(3)
    {
        előre(2) balra eddMeg jobbra(2) eddMeg balra
    }
    # vissza a kezdőpontba és elfordulás:
    hátra(8) jobbra
}
```

Szintén egy önmagában is ismétlődő tevékenységet kell ismételnünk, ha a korábbi négyzet alakú üres területet nem vonalazni, hanem négyzetrácsozni szeretnénk (6. ábra). Először elvégezhetjük a vonalazást a bal alsó sarokból indulva északra nézve, majd (például) a jobb alsó sarokból nyugatra nézve.

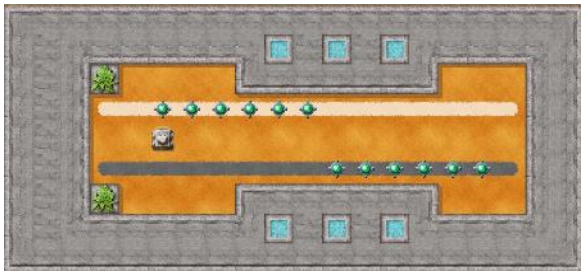


6. ábra: Padló lefestése négyzetrácsosan [6]

4.1.2. Pakolás

A szabályosan – például sorban vagy téglalapban – elhelyezett tárgyak átpakolása is jellemzően ismétlésre épülő tevékenység.

Az első lehetőség, hogy a bóják sora a robot mellett található. Ilyenkor elegendő egyetlen bója átpakolását és a következő bójához lépést összesen a bóják számával megegyező számban ismételni. A következő feladatban egymás melletti termékeket kell átpakolni először a fehérrel jelölt gyártósorról a feketére, majd fordítva (7. ábra).

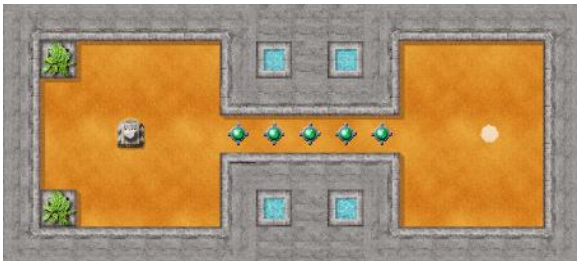


7. ábra: Pakolás futószalagokról [6]

Megállapodásunk szerint felhasználhatjuk a megoldásban a bóják darabszámát, ezért egy-egy számlálós ciklussal kivitelezhetjük az átpakolást. Figyelnünk kell arra, hogy az egyes átpakolásokkal a robot iránya is megváltozik:

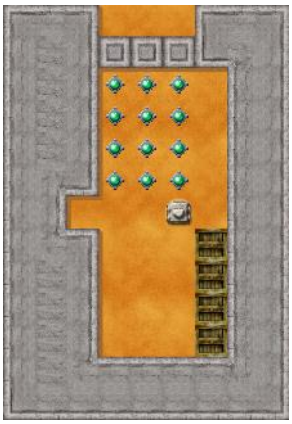
```
ismételd(6)
{
    veddFel jobbra(2) teddLe
    # következő hely:
    balra előre balra
}
jobbra(2)
ismételd(6)
{
    veddFel jobbra(2) teddLe
    # következő hely:
    jobbra előre jobbra
}
```

A másik lehetőség, hogy a bóják sora a robottal szemben található, amelyen a robotnak – mindig a soron következő bója maga mögé pakolásával – keresztül kell haladnia (8. ábra).



8. ábra: Át a bójákon [6]

Ha pedig a bóják két dimenzióban, például téglalap alakban vannak elrendezve, átpakolásuk a legegyszerűbben egymásba ágyazott számlálós ciklusokkal valósítható meg. Az alábbi terepen téglalap alakba rendezett hordókat kell átpakolni a pince egyik feléből a másikba (9. ábra).



9. ábra: Borospince (saját feladat)

Először átpakolhatjuk a téglalap jobb oldali oszlopát, majd átkerülve a szomszédos oszlop elé az egészet annyiszor ismételhetjük, ahány oszlopból a téglalap áll:

```
ismételd(3)
{
    ismételd(4)
    {
        # egy bója átpakolása az oszlopból:
        veddFel balra(2) előre(3) teddLe
        # séta az oszlopban lévő következő bójához:
        balra(2) előre(4)
    }
        # következő oszlop:
        hátra(4) balra előre jobbra
}
```

4.2. Érzékeléssel megoldható problémák

Az érzékelés nélkül megoldható problémák esetén a terepet teljesen ismertnek tekintettük, ezért a megoldásuknak semmilyen a környezetre vonatkozó feltételt nem kellett tartalmaznia. Foglalkozhatunk azonban olyan feladatokkal is, amelyekben csak a terep bizonyos adottságaival vagyunk tisztában. (A RoboMindban megtehetjük, hogy egy problémához több térképet is mellékelünk, így is világossá téve a terep állandónak és változónak tekintendő vonásait.)

Az érzékeléssel megoldható egyszerűbb feladatokban (például útvonalkövetés, labirintus megfejtése, keresés) a robotnak minden lépésben ugyanazon szabályok szerint kell cselekednie. A feladatot megoldó algoritmus jellemzően egy végtelen ciklusból és a benne elhelyezett, a különböző eseteket kezelő elágazásokból áll, amelyek közül az egyik a program leállításáért felelős.

A bonyolultabb feladatok esetén (például pakolás, területbejárás) a megoldás különböző szakaszokra osztható, amelyekhez a működést meghatározó különböző szabályok tartozhatnak. Ilyenkor a megszakításos végtelen ciklusok helyett már praktikusabb feltételes ciklusokat használni.

Mindkét esetben érdemes összevetni a lehetséges megoldások hatékonyságát, és főleg az érzékelések számát, amely az elágazások megfelelő megszervezésével minimalizálható.

4.2.1. Útvonalkövetés

Ha egy a robot által előre nem ismert útvonal egyenes szakaszait illetve bal és jobb fordulóit valamilyen következetes és a robot által érzékelhető módon megjelöljük a terepen, a robotot a megfelelő programmal képessé tehetjük az útvonal követésére. A legkézenfekvőbb jelölési mód a folyamatos festékcsík, de például a fordulókat kétféle színű festékfolttal is megjelölhetjük (az egyenes szakaszok mezői pedig maradnak üresen), illetve kitalálhatók egyéb jelölések is.

A robotnak minden egyes lépésben meg kell vizsgálnia (a jelölési módtól függően) egy vagy több szomszédos mezőt, majd a vizsgálat eredményétől függően előre, balra vagy jobbra haladnia. A vizsgálatokat végző elágazásokat ezért egy végtelen ciklusba célszerű foglalni, amelynek leállításáról egy további feltétel gondoskodik (például a robot elérte az útvonal végét jelző bóját).

Tekintsük részletesebben a tényleges vonalkövetést (10. ábra).

10. ábra: Vonalkövetés [6]

Ha a robot az útvonal egy egyenes szakaszán van, akkor a szomszédos mezők közül csak a szemben lévőn, ha pedig valamelyik fordulóban áll, akkor csak a bal vagy a jobb oldali mezőn található festéknyom. Ennek megfelelően az alábbi programváltozat készíthető el:

```
ismételd
{
    ha(szembenFehér){előre}
    ha(jobbraFehér){jobbra előre}
    ha(balraFehér){balra előre}
    ha(szembenBója){veddFel vége}
}
```

A program hatékonyabbá tehető, ha a feltételeket egymást kizáróan szervezzük meg, és a legvalószínűbbel kezdünk. A jelenlegi változattal a robot 96 vizsgálatot végez a példaként tekintett terepen, a következővel viszont csak 47-et:

```
ismételd
{
    ha(szembenFehér){előre}
    különben ha(jobbraFehér){jobbra előre}
    különben ha(balraFehér){balra előre}
    különben ha(szembenBója){veddFel vége}
}
```

Az útvonalkövetési feladat számos módon bonyolítható, például megmentendő áldozatok elhelyezésével az úton és annak szomszédos mezőin (11. ábra). Ebben az esetben egy adott helyen nemcsak a továbbhaladás irányával, de az onnan elérhető áldozatok felvételével is foglalkozni kell.

11. ábra: Áldozatok (saját feladat)

4.2.2. Labirintus megfejtése

Az útvonalkövetési feladatok után felmerül a kérdés, hogy egy elágazásokat is tartalmazó útvesztőben navigálható-e a robot úgy, hogy biztosan megtalálja a labirintusban valahol elhelyezett bóját.

Tekintsük először a körmentes labirintusokat, így például az alábbit (12. ábra).

12. ábra: Labirintus (saját feladat)

Mint ismeretes, körmentes labirintusok esetén garantáltan eljutunk a kijárathoz (vagy a keresett tárgyhoz) úgy, ha képzeletben jobb (vagy bal) kezünkkel megérintjük a falat, majd az elágazásokban mindig úgy haladunk tovább, hogy a falat egy pillanatra se kelljen elengednünk.

A „jobb kéz szabályt” alkalmazva kapjuk az alábbi programot:

```
ismételd
{
    ha(szembenBója){veddFel vége}
    különben ha(jobbraSzabad){jobbra előre}
    különben ha(szembenSzabad){előre}
    különben ha(balraSzabad){balra előre}
    különben
    {
        # zsákutca
          balra(2)
    }
}
```

A RoboMindban a kört tartalmazó labirintusok is megfejthetők. Trémaux algoritmusában [7] éppen kétféleképpen kell tudnunk megjelölni bizonyos útszakaszokat, amelyre a RoboMindban a fehér és a fekete festéket használhatjuk. Az említett algoritmus mindig elvezet a célhoz, és (jelen esetben) fehérrel jelöli a megtalált utat (13. ábra).

A programozás során tekintettel kell lennünk arra, hogy nem kérdezhető le az aktuálisan használt festék színe; helyette a robotnak vissza kell tolatnia egy mezőt, majd megvizsgálnia az előtte lévő mező színét.

13. ábra: Körmentes labirintus megfejtése Trémaux algoritmusával (saját feladat)

4.2.3. Keresés

Feladat lehet nyílt, legfeljebb kevés akadályt tartalmazó területen egy vagy több bóját megtalálni (14. ábra).

14. ábra: Keresés (saját feladat)

Megpróbálhatjuk a terület minél nagyobb részét véletlen bolyongással bejárni és közben a szemben (és esetleg az oldalt) található bójákat megenni. Kihasználhatjuk a robot „pénzfeldobásos" véletlengenerátorát, amely ugyanakkora valószínűséggel ad igaz vagy hamis értéket:

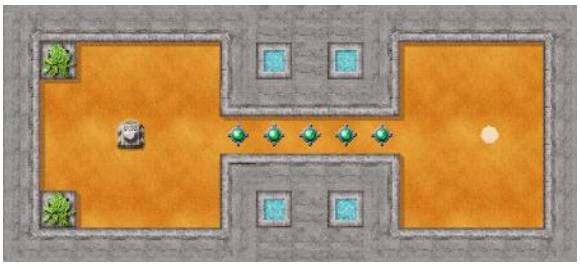
```
ismételd
{
    ha(szembenBója){eddMeg}
    ha(érmeFej)
    {
        ha(balraSzabad){balra előre}
    }
    különben
    {
        ha(jobbraSzabad){jobbra előre}
    }
}
```

Az iménti megoldás esetén azonban nem lehetünk abban biztosak, hogy a robot záros határidőn belül minden mezőt meglátogat. Az akadálymentes nyílt területek szisztematikus bejárására a **4.2.5 Területbejárás** című rész tér ki.

4.2.4. Pakolás

Ha a szabályosan – például sorban vagy téglalapban – elhelyezett bójákat a darabszámuk kihasználása nélkül kell átpakolnunk, akkor a pakolás befejezésének feltétele a bóják elfogyása lehet.

A már korábban látott feladatban ezúttal anélkül kell átjutnunk a bójákon keresztül a fehér foltig, hogy tisztában lennénk a bójákig vezető út hosszával, a bóják darabszámával vagy a legutolsó bója és a festékfolt távolságával (15. ábra).

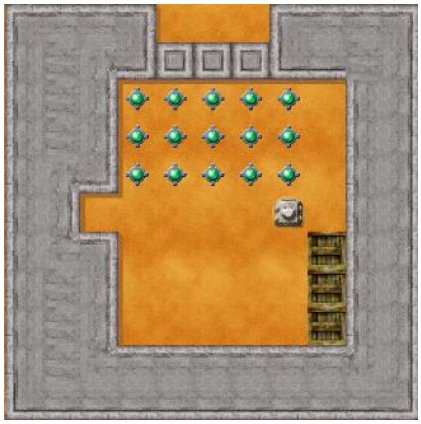


15. ábra: Át a bójákon [6]

A megoldás ezúttal három szakaszra osztható, amelyekben a robotnak eltérő szabályok szerint kell viselkednie: az elsőben addig kell a bóják felé haladnia, amíg el nem éri az első bóját, a másodikban addig kell a bójákat maga mögé pakolnia, amíg azok el nem fogynak, a befejező szakaszban pedig addig kell előre haladnia, amíg el nem éri a fehér foltot. Mindhárom szakaszon belül a robotnak van egy alaptevékenysége és egy azt megszakító feltétele, ezért a megszakításos végtelen ciklusok helyett már egyértelműen praktikusabb a feltételes ciklusok használata:

```
jobbra
# a bóják megközelítése:
ismételdAmíg(szembenSzabad){előre}
# a bóják átpakolása egyenként:
ismételdAmíg(szembenBója)
{
    veddFel jobbra(2) teddLe jobbra(2) előre
}
# a fehér folt megközelítése:
ismételdAmíg(nem szembenFehér){előre}
előre vége
```

Szintén korábbi feladat volt a téglalapba rendezett bóják átpakolása. Ezúttal megoldhatjuk úgy, hogy nem használjuk ki a téglalap konkrét méreteit (16. ábra).



16. ábra: Borospince (saját feladat)

A megoldás megint több szakaszos. Egy oszlopon belül először felvesszük az első bóját és letesszük közvetlenül a hátsó fal elé. Ezután visszamegyünk a következő bójáért. Mindezt addig ismételjük, amíg az oszlopon belül el nem fogynak a bóják. Utána átállunk a következő oszlop-

ra, és az előzővel megegyezően pakoljuk át azt is. Ha pedig nincs több oszlop, befejezzük a programot:

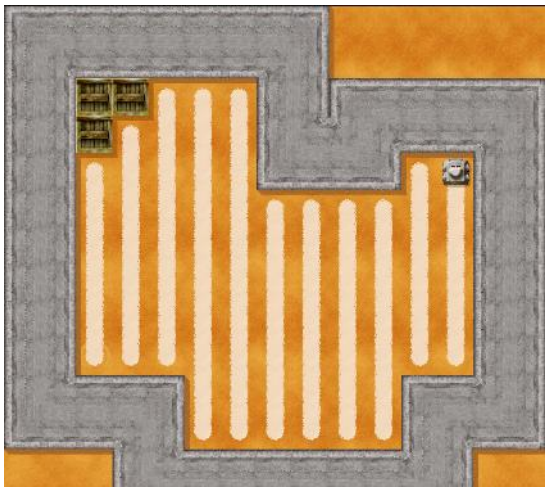
```
ismételdAmíg(szembenBója)
{
    # egy oszlop átpakolása:
    ismételdAmíg(szembenBója)
    {
        # egy bója átpakolása:
        veddFel balra(2)
        ismételdAmíg(szembenSzabad){előre}
        hátra teddLe balra(2)
        ismételdAmíg(szembenSzabad){előre}
    }
    # következő oszlopra állás:
    ismételdAmíg(balraAkadály){hátra}
    balra előre jobbra
}
```

4.2.5. Területbejárás

Amint szó volt róla, a keresési feladatra adott, véletlen bolyongáson alapuló megoldás során nem lehetünk biztosak abban, hogy a robot elfogadható időn belül minden mezőt meglátogat.

Foglalkozhatunk ezért nyílt területek következetes bejárásával – az egyszerűség kedvéért olyanokéval, amelyek függőlegesen vagy vízszintesen megszakítás nélkül bevonalkázhatók. Menetközben festhetünk is, hogy a bejárás látványosabb legyen.

A következő példában (17. ábra) a robot a bal alsó sarokból indulva függőlegesen be tudja vonalkázni a teljes területet az említett módon.



17. ábra: Területbejárás (saját feladat)

A megoldás során a robot oszloponként halad. Miután egy oszlopot bejárt, visszatér az oszlop „aljához”, majd megkeresi a következő oszlop „alját”, amely az előzőnél „magasabban” és „alacsonyabban” is lehet. Ha van következő oszlop, az eddigiek szerint jár el, különben befejeződik a program:

```
ismételd
{
    # aktuális oszlop megfestése:
    fessFehéren
    ismételdAmíg(szembenSzabad){előre}
    # vissza az oszlop aljához:
    nefess jobbra(2)
    ismételdAmíg(szembenSzabad){előre}
    balra(2)
    # a következő oszlop alja magasabban van?
    ha(jobbraAkadály)
        {
        ismételdAmíg(jobbraAkadály)
        {
            előre
            # nincs következő oszlop?
            ha(szembenAkadály){vége}
        }
        jobbra előre balra
    }
    különben
    {
        jobbra előre jobbra
        ismételdAmíg(szembenSzabad){előre}
        balra(2)
    }
}
```

Ezután már a program kiegészíthető az útba eső bóják összeszedésével – így biztosan minden bóját megtalálunk. Érdemes a szükséges mozgások számát összehasonlítani a véletlenszerű bolyongáséval.

5. A RoboMind értékelése

A RoboMind beavatkozni és érzékelni is képes robotjával jól bemutathatók a robotika elméleti alapjai, és segítségével számos tipikus robotikai probléma modellezhető. Habár a robot nem megfogható és nem is átalakítható, mindenkinek külön példány jut belőle, amelynek programja és környezete gyorsan módosítható és kipróbálható. Mindezeknek köszönhetően pedig a figyelem az algoritmizálásra irányítható.

Programozási nyelve rendelkezik az automata elvű eljárásorientált nyelvek minden kellékével, amelyek közül elsősorban a vezérlési szerkezetek használata motivált, de összetettebb feladatokban eljárások bevezetésére is szükség lehet. Szintaktikája népszerű programozási nyelvekéhez hasonló, de kulcsszavai (a kezelőfelülettel együtt) magyarul is használhatók.

A RoboMindot tehát akkor érdemes választanunk, ha tanítványainkat megismertetnénk a robotikával, de a hangsúlyt az algoritmizálásra és a programozásra kívánjuk helyezni. Bízom benne, hogy cikkemmel a programozási környezet iránti érdeklődés felkeltésén túl a felhasználására vonatkozó iránymutatással is szolgálni tudtam.

Irodalom

1. Bee-Bot Home Page
   https://www.bee-bot.us/ (utoljára megtekintve: 2015.11.01.)

2. Programming the Pro-Bot
http://doc.terrapinlogo.com/doku.php/logo:programming_probot (utoljára megtekintve: 2015.11.01.)
3. Codie - Cute Personal Robot That Makes Coding Fun
https://www.indiegogo.com/projects/codie-cute-personal-robot-that-makes-coding-fun#/
(utoljára megtekintve: 2015.11.01.)
4. Lego Mindstorms
http://www.lego.com/en-us/mindstorms (utoljára megtekintve: 2015.11.01.)
5. RoboMind.net
http://robomind.net/en/index.html (utoljára megtekintve: 2015.11.01.)
6. RoboMind Academy
https://www.robomindacademy.com (utoljára megtekintve: 2015.11.01.)
7. Budd, C. J., Sangwin, C. J.: Mathematics Galore! Oxford University Press (2001)