Rekurzió tanítása LOGO-ban programozási fogalmak előkészítésével

Erdősné Németh Ágnes

erdosne@blg.hu
Batthyány Lajos Gimnázium, Nagykanizsa
ELTE IK Doktori Iskola

Absztrakt. A rekurzió erőteljes eszköz, de a legtöbb diák és tanár egyetért abban, hogy nehéz megtanulni, megérteni és tanítani. A LOGO ábrákon keresztül, a rekurzió vizualizálásával könnyebben felfedeztethetővé és megérthetővé válik ez a nehéz koncepció. A cikk a felső tagozatosok tanításakor alkalmazható felépítést mutatja be. A számítógépes gondolkodás elvei alapján felépítve a későbbi, programozással kapcsolatos fogalmak megalapozását vetíti előre.

Kulcsszavak: LOGO tanítása, rekurzió, számítógépes gondolkodás, felsőtagozat, LOGO verseny

1. Áttekintés

A rekurzió a számítástudomány egyik alaptétele, erőteljes eszköz bizonyos problématípusok megoldásakor [2]. A diákok a kezdő programozás kurzusokon és a matematikai tanulmányaik során is találkoznak vele. Nehéz, mély gondolat és ugyanakkor absztrakt, nehezen magyarázható és nehezen határozható meg [3]. Sokan egyetértenek azzal, hogy bár erőteljes és jelentős elv, nehéz megérteni és megtanulni [4]. Úgy tűnik, hogy a taníthatóság nehézségeit minden iskolai szinten tapasztalták már [8].

A programozás tankönyvekben három fő példát találunk a rekurzióra: a Hanoi tornyait, a faktoriális számítását és a Fibonacci sorozatot [10]. Ezek a példák erőteljesek, de nem elég kifejezők az általános iskolások számára, akiknek manipulációra és/vagy vizualizációra lenne szükségük a megértéshez. Ezeken kívül a legtöbb programozás tankönyv ciklusos példákat mutat be, a valódi rekurzív példák helyett.

Hromkowitz tankönyvében [1] a rekurzió LOGO-val történő tanítása egy egyszerű definícióval kezdődik: „A rekurzió az ismétlődő elemek önmagukhoz hasonló módon történő feldolgozása." Az első példa egy végtelen, paraméterek nélküli rekurzió, mely nem csinál semmit, mivel az első utasítás már maga a rekurzív hívás. Ezután különböző szöggel és hosszúsággal rajzolt spirálok variációi következnek, egyre növekvő paraméterekkel. Csak ezután mutat példát a feltételes megállításra, majd elemzi a hívások mélységét. Ezeket az elméleti példákat követik a hagyományos feladatok: a zárójelezés helyessége, a Koch-görbe, a Sierpinski-háromszög, fák és hópihék. Ez az elméleti felépítés felnőttek számára megfelelhet, de gyerekek számára a LOGO vizualitásának kihasználása nélkül elég használhatatlan.

2. Rekurzió tanítása LOGO-val

Gimnáziumunkban ötödiktől kezdve a tanórákon a LOGO programozási nyelv segítségével gondolkodásfejlesztés és problémamegoldás zajlik: analizálás, elemekre bontás és újraépítés, az algoritmikus gondolkodás megismerése. A LOGO-ban természetes módon elérhető vizualizáció, a hétköznapi élettel való kapcsolat, a sokféle példa segít a megértésben, növeli a motivációt, segíti a koncepciók kialakulását és erős alapot nyújt a további tanulmányokhoz. Órákon a legfontosabb a számí-

tógépes gondolkodás erősítése, míg a LOGO szakkörökön a későbbi programozási tanulmányok előkészítése is zajlik.

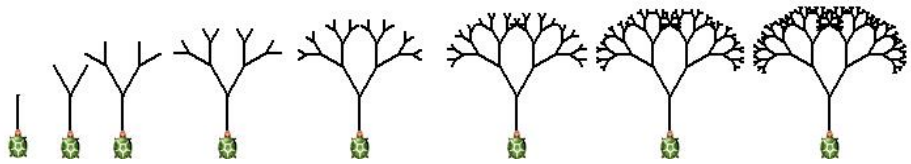
A rekurziók tanítására – a számítógépes gondolkodást szem előtt tartva – 11-12 éves kortól kerülhet sor. A rekurzió tanulása előtt a gyerekeknek nagy biztonsággal kell tudniuk kezelni a fejlesztőfelületet, ismerniük kell az előre, hátra, jobbra, balra alaputasításokat, az elágazás fogalmát és használatát, az ismétléses ciklus fogalmát és kezelését. Kell tudniuk eljárásokat írni és azokat paraméterezniük. Ezek biztos használata után kerülhet sor erre a nem könnyű gondolatra, a rajzokon keresztül viszont – tapasztalataim alapján - könnyedén megérthető és alkalmazható alapismeretre. A LOGO programozási nyelv egy természetes eszköz a rekurzió tanítására, jól működik a képekben a kép részeinek újrafelfedezése.

A cikkben a rekurzió tanításának egy, a számítógépes gondolkodást és a későbbi tanulmányok előkészítését célzó, tudatos felépítését írom le. Természetesen az egyes lépések megértése után mélyíteni és rögzíteni kell az épp megtanult ismereteket, sok-sok feladaton keresztül.

Az ábrák alapjai LOGO versenyfeladatokból származnak, saját megvalósításban jelentek meg az ISSEP 2015 konferencia kiadványában, a doktori dolgozatomban és itt is.

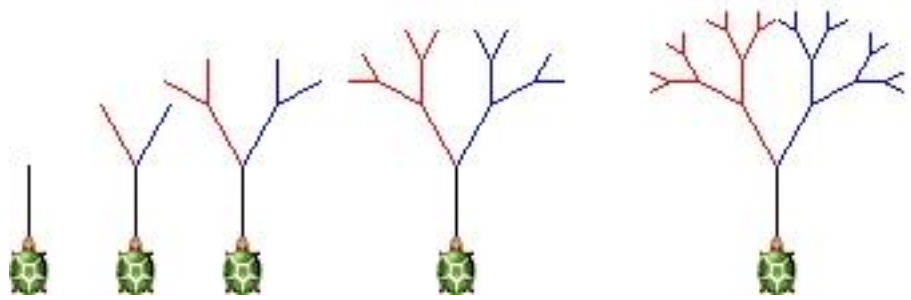
3. A rekurzív lépés megértése

Az ismerkedést a bináris fákkal kezdhetjük. A példa a valós életből való, a gyerekek emlékeznek arra, hogyan nő évről-évre egy fa – ezt először egy, a növekedést bemutató programmal szemléltethetjük. (1. ábra)



1. ábra Bináris fa növekedése

A következő kép (2. ábra) a módszer bemutatására szolgál, érdemes előre elkészíteni és megbeszélni a rajzot, megfogalmaztatni a következőket: „A fa piros része pont ugyanolyan, mint a teljes fa egy évvel korábbi állapota. A kék rész is az egy évvel fiatalabb teljes fa, csak más irányban nőtt, de ugyanazon a helyen, mint a piros. Rajzolj egy egyenest, fordulj balra, rajzolj egy évvel fiatalabb fát, fordulj jobbra, rajzolj újra egy évvel fiatalabbat, fordulj vissza balra és tolass vissza a teknőccel a kiindulási helyzetbe!”



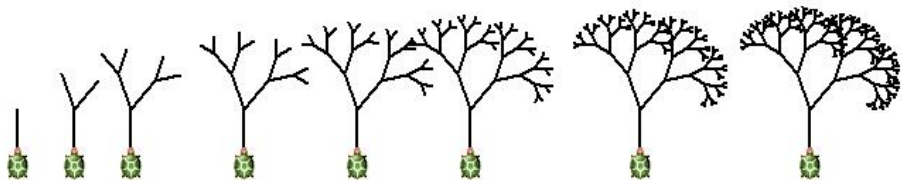
2. ábra Bináris fa színesen, magyarázathoz

A gyerekeknek saját szavaikkal, egyszerűen, maguknak kell megfogalmazniuk a szigorú csökkenést és az állapotátlátszóság követelményét. A rekurzív mintát nekik kell részekre bontani, amiben

újra felfedezik az eredeti mintát, kisebb verzióban. Az ismerkedés során fel kell velük fedeztetni a rekurzió alapalgoritmusát:

```
mintarajzolás
    alapelem megrajzolása
    mozgás/fordulás
    kisebb állapotú minta rajzolása
```

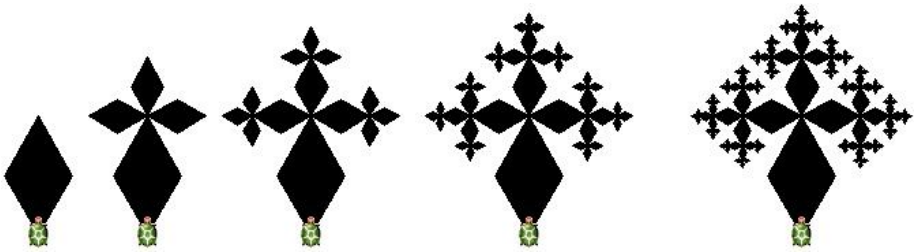
A vizualizálás segít megfogalmazni a rekurzív minták elvét. Az eljárás megfogalmazása után LOGO-ban implementálni tudják a rajzolás folyamatát. A megértést különböző számú ágakkal, szélfútta fára hasonlító más szögű forgásos fákkal, illetve ezek kombinációjával tudjuk ellenőrizni. Például a 3. ábrán látható fával.



3. ábra Szélfútta bináris fa

4. Kiinduló állapot

Az első mintában a kiinduló állapot egy nagyon egyszerű alakzat volt, egy szakasz. Így a kiinduló lépésben csak előre és hátra utasításokat kellett használni. Következő gondolati elem a kiinduló állapot bonyolítása, például rombuszrajzolással. (4. ábra)



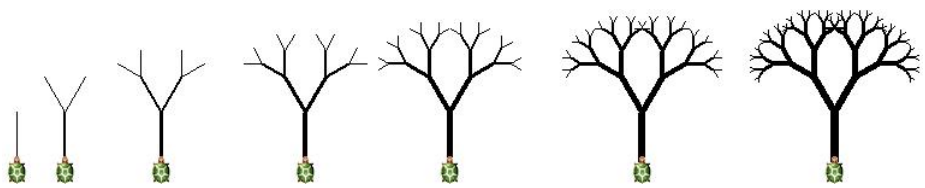
4. ábra Növekvő gyémánt

Ebben a mintában már kilépési feltételt is meg kell fogalmazni és alapállapotként egy trapézt kell rajzolnia a teknőcnek. Azaz fel kell ismerni és meg kell fogalmazni a rekurzív eljárások módosított sablonját:

```
alap_eljárás
    alapállapot esete
    rekurzív eset
```

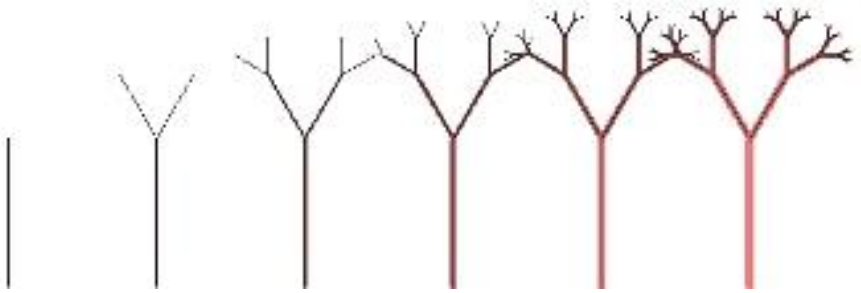
5. Szintek

A rekurzió végrehajtásának folyamán minden állapotban tudnunk kell, hogy épp hányadik rekurziós szinten vagyunk. A következő két példában a szintre vonatkozó információt kell felhasználni. Sokkal jobban hasonlít a fa az igazi növekedésre, ha az évek számával arányos az ág vastagsága. (5. ábra)



5. ábra Vékonyodó bináris fa

Minden lépésben megvizsgáltathatjuk a verem állapotát és akár ki is írathatjuk a rekurzió szintjét. Vizuálisan színezésre és a fa vastagságának megadására használhatjuk ezeket az információkat. (6. ábra)



6. ábra Színes bináris fa

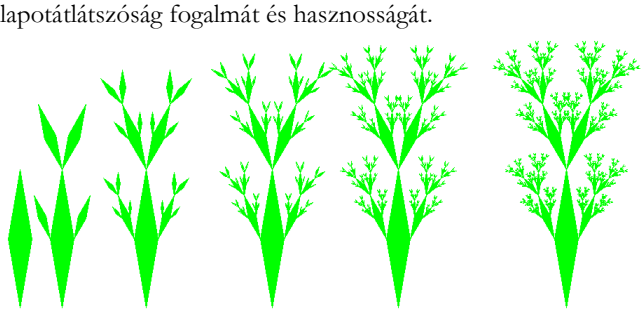
Eközben egy nagyon fontos új fogalommal is megismerkedhetnek a diákok, a verem fogalmával – itt még csak gyakorlati megvalósításban.

Ezután – akár megszakítva a rekurziókkal kapcsolatos ismereteket – kerülhet sor a funkcionális programozás alapjainak, ezen belül a listakezelés megértésére.

6. Állapotátlátszóság

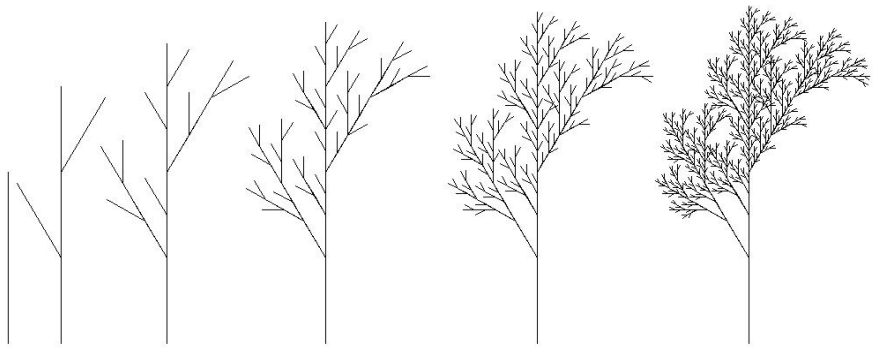
A rekurziók felfedezésében a következő lépés az állapotátlátszóság elvének pontosítása. A rekurzió nagyon rossz ábrákat adhat, ha nem tisztázzuk előre, hogy a teknőc hol van és milyen irányban áll a rekurzív lépés meghívása előtt és után.

A következő ábrák rajzolásakor a rekurziós lépések meghívása előtt a teknőcnek bonyolult mozgást kell tennie. (7. ábra, 8. ábra) Ezek és hasonló ábrák rajzolása közben, a gyakorlati megvalósítás során értik meg az állapotátlátszóság fogalmát és hasznosságát.



7. ábra Növekvő kaktusz

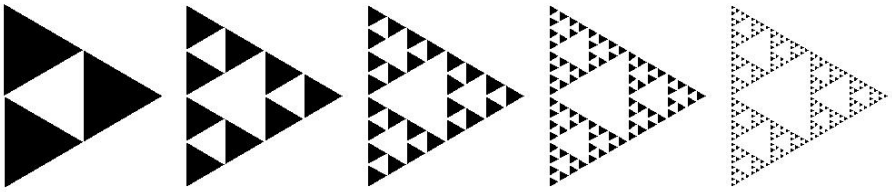
A megvalósítás során a kulcs és az első lépés a helyzetek és irányok tisztázása, az alapábra megtalálása. Pontosítani kell a rekurzió mélységét is az egyes esetekben. Csak ezek után lehet nekilátni a teljes eljárás kódolásának.



8. ábra Szélfútta fa variáció

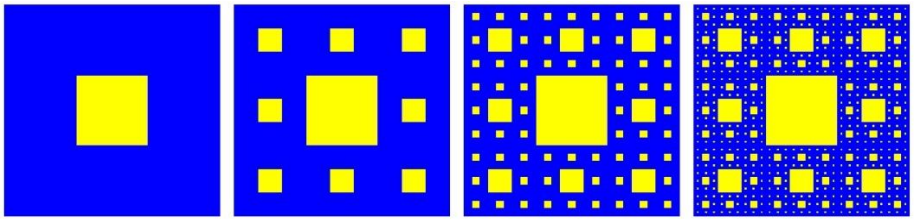
7. Az adatszerkezetek határai

A következő két klasszikus rekurzív mintán a rekurziós hívások számának korlátait tudják a diákok megtapasztalni. A Sierpinski-háromszöget (9. ábra) és a Sierpinski-szőnyeget (10. ábra) többféleképpen is lehet kódolni, attól függően, hogy a színes háromszögeket rajzoljuk meg vagy egy alapállapot színes ábráján törléssel hozzuk létre a mintát.



9. ábra Sierpinski-háromszög

Ezeken az ábrákon lehet beszélgetni a gyerekekkel az adatszerkezetek véges voltáról is. Ha a rekurziós hívások száma nagyobb, mint 5-6, akkor az ábrán már semmilyen változás nem észlelhető, hiszen a következő lépésben rajzolandó rész már túlságosan kicsi. A rajzolandó négyzet/háromszög mérete hamar kisebb lesz, mint egy pixel a képernyőn, így megjeleníthetetlen. Tipikus ötlet ennek áthidalására a gyerekek részéről, hogy akkor legyen az első ábra sokkal nagyobb – kipróbálással megtapasztalhatják, hogy ez maximum egy szinttel bővíti a lehetőséget.



10. ábra Sierpinski-szőnyeg

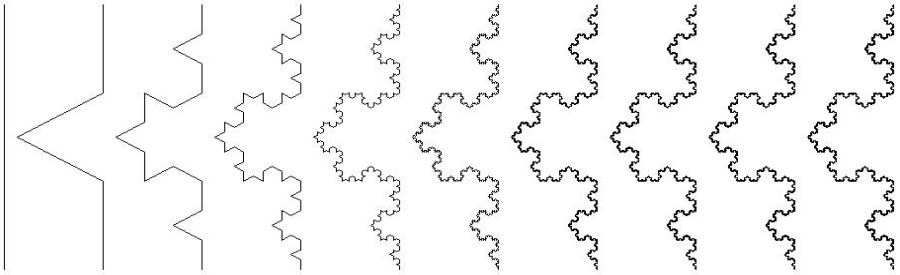
Elérve a megjelenítési határt, a számolás megy tovább, de a teknőc gyors forgásán kívül semmi változást nem látunk a képernyőn. Könnyen kiszámolható az egyes esetekben az a határ, amikor a rajz eltűnik ($h/3^n<1$).

A számítógépes gondolkodás tanításában ez egy nagyon fontos lépés – lenyűgöző lehetőség az adatstruktúrák végességének vizualizálására – ne hagyjuk ki! A későbbi tanulmányokban, amikor a

különböző adatszerkezetek (egész, hosszú egész, valós, szöveg) határairól tanulnak, könnyen tudunk ide visszahivatkozni.

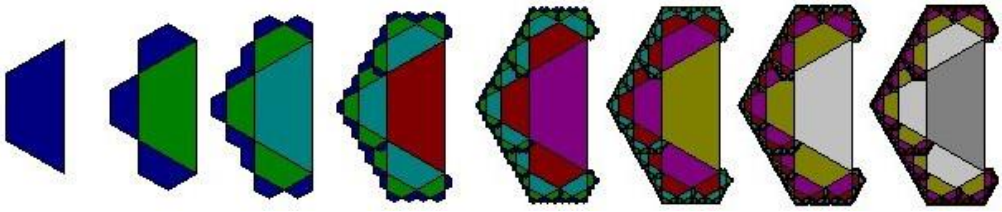
8. Futási idő vizsgálata

A Koch-görbe (11. ábra) megrajzoltatása az exponenciális futási idő szemléltetésére alkalmas. A diákok azt gondolják, a számítógépek hihetetlenül gyorsak, bármit képesek kiszámolni egy szempillantás alatt. Az első néhány szint még elég gyorsan megy, de a 6. szinttől a rajzolás egyre lassabb lesz. Láthatjuk, hogy a teknőc hihetetlen gyorsan forog és számol. A 10. szint megrajzolása már perceket vesz igénybe. Érdemes mérni a futási időt, feljegyezni és megtapasztalni a futási idő növekedését.



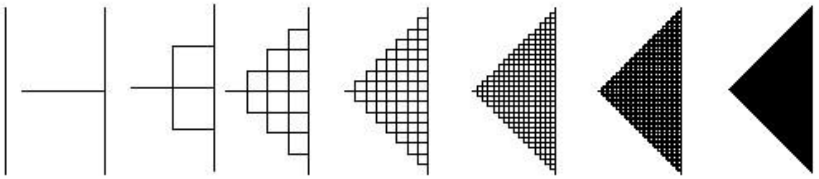
11. ábra Koch görbe

A trapézos feladaton (12. ábra) jól lehet gyakorolni az összes eddigi ismeretet, s újra megtapasztalni a futási idő és az adatszerkezet határait.



12. ábra Trapézok

Egyenesekből álló, egyszerű alakzattal (13. ábra) is lehet a területet teljesen lefedő alakzatot generálni, néhány lépéssel.

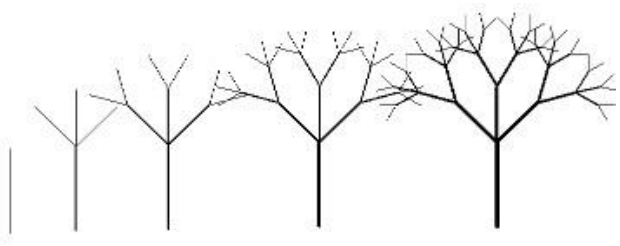


13. ábra Területlefedő alakzat

9. Egymást hívó rekurziók

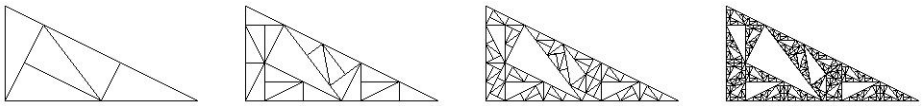
A rekurziók következő kognitív szintje, amikor egyik eljárás hívja a másikat, majd a másik visszahívja az elsőt. A LOGO vizualitásával ez az elv könnyen látvánnyá tehető és így könnyebben megérthető.

Első példaként újra visszatérhetünk a legegyszerűbb ábrához, a jól ismert, sokféle variációban már programozott fához. Ebben még akár egy elágazással meg tudjuk oldani a kétféle ábrát (14. ábra), az elvi lépés ezután következhet.

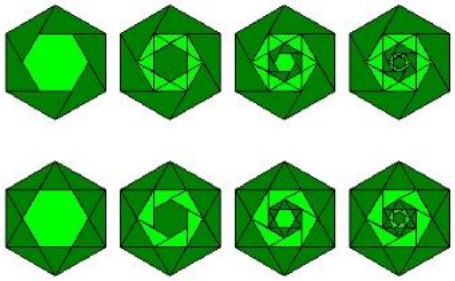


14. ábra Felváltva kettő-három fa

A következő bonyolultabb ábra (15. ábra) gondos tanulmányozásával a gyerekek felfedezhetik, hogy a háromszögek némely esetekben balsodrásúak, míg máskor jobbsodrásúak. A jobbsodrású háromszögek belsejében három jobbsodrású és egy balsodrású, míg a balsodrásúak belsejében három balsodrású és egy jobbsodrású ábra van – ennek megvalósításához már két eljárást kell írni, amik egymást hívják.



15. ábra Egymást hívó háromszög eljárások



16. ábra Hatszögek – szimultán rekurzió

A hatszöges példában (16. ábra) egy újabb paramétert is be kell vezetni annak tárolására, hogy melyik fajta eljárással kezdődjön a szimultán rekurzió.

Az egymást hívó eljárásokat – tapasztalatom szerint – már csak a kiemelkedő képességű diákok tudják önállóan lekódolni. A többiekkel elegendő addig eljutni, hogy megértsék és saját szavaikkal meg tudják fogalmazni, hogyan lehet ezeket az ábrákat megrajzolni.

10. Összefoglalás

A cikkben tanári szempontból mutattam be a rekurzió alapfogalmainak bevezetését és a további programozási ismeretek előkészítését. A legfontosabb, hogy tudatosan és egyszerűen, a gyerekek által megfogalmazva, a saját szintjükön legyenek kimondva a rekurzióval kapcsolatos fogalmak. Ha ezek megvannak, akkor később, amikor vizualizáció nélkül kerül elő a felülről-lefele történő építkezés elve, majd ebből a tudásból kiindulva lehet megfogalmazni a memorizálást s aztán áttérni a dinamikus programozás lentről-felfele építkezési elvére.

A megértést ellenőrizni és az elv használatának gyakorlását sok-sok további feladattal lehet. Ehhez elég sok példát találhatunk a LOGO versenyfeladatok gyűjteményeiben. A tapasztalatom az, hogy a gyerekek nagyon kreatívak abban is, hogy új feladatokat találjanak ki, illetve egy-egy elrontott feladatmegoldásból is lehet újabb ötleteket meríteni.

Irodalom

1. Hromkovic, J.: *Einführung in die Programmierung mit Logo: Lehrbuch Informatik*, Vieweg+Taubner GWv Fachverlage GmbH, Wiesbaden, 2010
2. Kalas, I., Blaho, A.: *I Beg Your Pardon Turtles: Don't Forget About Data Structures*, Eurologo'97 Proceedings, Budapest
3. Levy, D.: *Classification, Discussion, Recursion: Helping the Development of Computer-Science Concepts*, Eurologo'97 Proceedings, Budapest
4. Leron, U.: *What makes recursion hard*, Proceedings of the Sixth International Congress on Mathematics Education (ICME6), 1988, Budapest, Hungary
5. Murnane, J.: *To iterate or to recurse?* Computers & Education, Volume 19/4, 1992, pp. 387–394.
6. Er, M. C.: *On the complexity of recursion in problem-solving*, International Journal of Man-Machine Studies, Volume 20/6, 1984, pp. 537–544.
7. Wilcocks, D., Sanders, I.: *Animating recursion as an aid to instruction*, Computers & Education, Volume 23/3, 1994, pp. 221–226.
8. Levy, D.: *Collaborative Conceptual Change: The Case of Recursion*, Journal of Intelligent Systems 01/2002; 12(2)
9. Close, J., Dicheva, D.: *Misconceptions in Recursion: Diagnostic Teaching*, Eurologo'97 Proceedings, Budapest
10. Wirth, M. A.: *The far side of recursion*, Teaching mathematics and computer science, 2015/1, pp. 57-71.
11. Heizlerné Bakonyi, V., Zsakó, L.: *Strategy of guessing exercises – Variations of drawing trees*, Proceedings of the 9th International Conference on Applied Informatics Eger, 2014. Vol. 1., pp. 285–294.
12. Wing J. M.: *Computational thinking*, Communications of the ACM, 49(3), p. 33-35, 2006. https://www.cs.cmu.edu/~15110-s13/Wing06-ct.pdf
13. Rónai O. R., Vörös V.: *Lógós ecsetvonások*, NJSZT 2008, http://tetlabor.inf.elte.hu/logosecsetvonasok/lecke6.html & lecke7.html
14. *LOGO versenyfeladatok tára 1998-2002*, Mészáros Tamásné, Zsakó László, NJSZT, 2002, http://logo.inf.elte.hu/peldatar/LogoPeldatar1998_2002.pdf
15. *LOGO versenyfeladatok tára 2003-2007*, Heizlerné Bakonyi Viktória, Zsakó László, NJSZT, 2008, http://logo.inf.elte.hu/peldatar/Logo2003_2008.pdf
16. *LOGO versenyfeladatok tára 2008-2012*, Heizlerné Bakonyi Viktória, Zsakó László, NJSZT, 2013, http://logo.inf.elte.hu/peldatar/LogoPeldatar2008_2012.pdf
17. *NJSZT Logo Országos Számítástechnikai Tanulmányi Verseny archívum*, http://logo.inf.elte.hu/logo_archivum.html
18. Erdősné Németh Á.: *Introducing recursion with LOGO in upper primary school*, ISSEP Proceedings 2015, Ljubljana, pp. 121-129.