A blokkalapú és a szövegalapú kódolás értékelése

Bernát Péter

bernatp@inf.elte.hu
ELTE IK

Absztrakt. A kezdő programozók számára nemcsak a szemantikailag, de már a szintaktikailag helyes program létrehozása is kihívást jelentő feladat. Utóbbit esetenként a szöveges programkód bevitelében segítő szolgáltatásokkal, más esetekben a szöveges kódolást kiváltó blokkalapú programozással kívánják megkönnyíteni. Cikkemben a kétféle kódlétrehozási lehetőség összehasonlításával és értékelésével igazolom, három informatikatanárral készített interjúval pedig alátámasztom, hogy a kezdők számára könnyebb a blokkalapú programozást elsajátítani, a haladó szintű programozáshoz viszont szükséges áttérni a szöveges kódolásra. Majd rámutatok arra, hogy az áttérés változatlan programozási témakörön belül is lehetséges, és utalok az interjúalanyaimnak az áttéréssel kapcsolatos tapasztalataira.

Kulcsszavak: programozástanítás, blokkalapú kódolás, szövegalapú kódolás, interjú

1. A kétféle kódolás

A kezdőknek szánt oktatási célú programozási nyelvek megalkotói számos megoldással igyekeznek a programozás legelső lépéseit minél érthetőbbé és motiválóbbá tenni. A Carnegie Mellon egyetem oktatói által létrehozott taxonómia [1] a törekvéseket három nagy kategóriába sorolja aszerint, hogy a programozási paradigmával, a program létrehozásával, vagy annak futtatásával kapcsolatosak-e.

A kezdő programozók számára nemcsak a szemantikailag, de már a szintaktikailag helyes program létrehozása is kihívást jelentő feladat. Utóbbit esetenként a szöveges programkód bevitelében segítő szolgáltatásokkal, más esetekben a szöveges kódolást kiváltó blokkalapú programozással kívánják megkönnyíteni. A blokkalapú programozás során a programkód paraméterezhető grafikus blokkok összeillesztésével állítható elő. A blokkok funkcióját szöveges vagy képi tartalmuk fejezi ki (1. ábra).

1. ábra: Programkód egy képes (ScratchJr) és egy szöveges (Alice) blokkalapú programozási nyelven

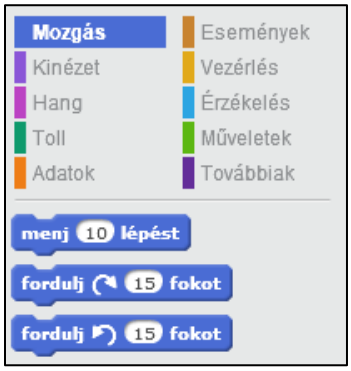
Cikkemben a kétféle kódlétrehozási lehetőség összehasonlításával és értékelésével igazolom, három informatikatanárral készített interjúval pedig alátámasztom, hogy a kezdők számára könnyebb a blokkalapú programozást elsajátítani, a haladó szintű programozáshoz viszont szükséges áttérni a szöveges kódolásra. Majd rámutatok arra, hogy az áttérés változatlan programozási témakörön belül is lehetséges, és utalok az interjúalanyaimnak az áttéréssel kapcsolatos tapasztalataira.

Az összehasonlítás során mindig zárójelben tüntetem fel, hogy valamely tulajdonsággal a példaként választott ScratchJr (scratchjr.org), Scratch (scratch.mit.edu) és Alice (alice.org) blokkalapú, illetve Imagine Logo (logo.sulinet.hu), RoboMind (robomind.net) és Small Basic (smallbasic.com) szövegalapú oktatási célú programozási környezetek közül melyek rendelkeznek.

2. A kétféle kódolás összehasonlítása

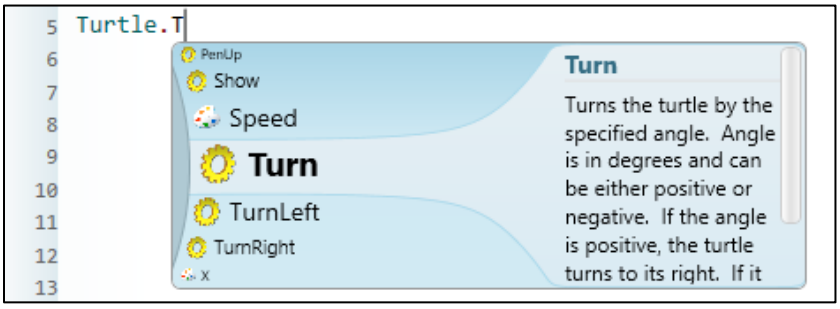
2.1. A nyelvi elemek beillesztése a programkódba

A blokkalapú programozási környezetekben a programozási területre behúzható nyelvi elemek egy blokk-készletben érhetők el funkció szerinti csoportosításban (ScratchJr, Scratch és Alice) (2. ábra). Ezáltal nem szükséges pontosan emlékezni a nevükre, és gépelési hibáktól mentesen lehet azokat beilleszteni a programba.



2. ábra: A Scratch blokkjainak csoportjai, és a Mozgás csoport néhány utasítása

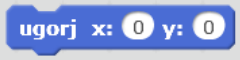
A szöveges programkód bevitelét egyes fejlesztői környezetek az automatikus kódkiegészítéssel segítik (Small Basic). Ez a funkció a kontextus és a már beírt karakterek alapján egy a kurzornál megjelenő legördülő listában kínálja fel a lehetséges kulcsszavakat és azonosítóneveket, amelyeket egy vagy legfeljebb egy-két billentyű lenyomásával be lehet szúrni (3. ábra). Ez is segít a nyelvi elemek felidézésében, és a helyes bevitelükben, ugyanakkor az elírási hibák vele együtt sem zárhatók ki, és nem is mindegyik fejlesztői környezet kínálja fel ezt a szolgáltatást.



3. ábra: Automatikus kódkiegészítés a Small Basicben

2.2. A nyelvi elemek paraméterezése

A blokkalapú programozási nyelvekben a nyelvi elemek és a paramétereik együtt alkotnak egy blokkot, ezért csak a megfelelő *számú* paraméter adható meg a számukra a megfelelő *helyeken* (prefix vagy infix alakban) (ScratchJr, Scratch és Alice), és több paraméter esetén a szöveges tartalmukból kiderül az egyes paraméterek *funkciója* (Scratch és Alice) (1. táblázat, első oszlop).

Scratch	Imagine Logo
	`lenyomat`
	`tollszín!`
	`xypoz!`
	`szó`
	`és`

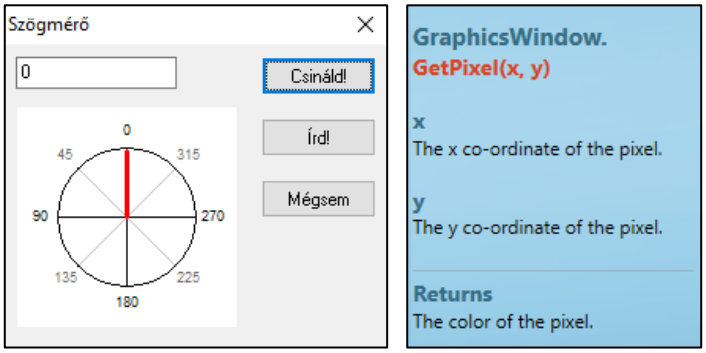
1. táblázat: Ugyanazon funkciókat betöltő nyelvi elemek a Scratch-ben és az Imagine Logóban; az előbbi nyelvben kiderül a paraméterek darabszáma, helye és funkciója, az utóbbiban nem

Az alábbi lehetőségek valamelyikével pedig azt is biztosítják, hogy csak megfelelő *típusú* értéket lehessen megadni paraméterként:

1. a paraméterhely alakja megszabja a beilleszthető elem típusát (például a táblázat első oszlopában az *és* művelet paramétereiként csak két logikai értékű kifejezés adható meg) (Scratch);
2. csak a megfelelő típusú érték gépelhető be (például a táblázat első oszlopában az ugorj utasítás paraméterhelyeibe csak számok írhatók) (ScratchJr, Scratch és Alice);
3. egy lenyíló listából választható ki a paraméter lehetséges értékei közül valamelyik (Scratch és Alice).

A szövegalapú programozási nyelvekben csupán a nyelvi elemekből nem derül ki, hogy szükségük van-e paraméterre, és ha igen, akkor hány *darabra,* milyen *típusúra,* és több paraméter esetén milyen *pozícióban* és *szereposztásban* (1. táblázat, második oszlop). A fejlesztői környezetek egy részében ezért előhívható egy olyan ablak, amelyben a kurzornál található nyelvi elem egy űrlap segítségével paraméterezhető fel, így – a blokkalapú kódoláshoz hasonlóan – csak a megfelelő számú és típusú paraméter adható meg, amelyeknek kiderül a funkciója, és amelyeket a környezet másol be a programkód megfelelő helyeire (Imagine Logo) (4. ábra, bal oldal).

A szöveges fejlesztői környezetek egy másik része a kurzornál található nyelvi elem paraméterezéséről egy helyzetérzékeny súgóban tájékoztat (Small Basic), ebben az esetben azonban megadható tévedésből túl sok vagy túl kevés, esetleg hibás típusú vagy rossz helyre írt paraméter (4. ábra, jobb oldal). És vannak olyan fejlesztői környezetek is, amelyek egyik lehetőséget sem kínálják fel (RoboMind).



4. ábra: A kurzornál található utasítás paraméterezését segítő űrlap az Imagine Logóban, illetve helyzetérzékeny súgó a kurzornál lévő utasítás paraméterezéséről a Small Basicben

2.3. A vezérlési szerkezetek létrehozása

Hasonló a helyzet a vezérlési szerkezetek létrehozásával kapcsolatban. A blokkalapú nyelvekben a vezérlési szerkezetek kulcsszavai és utasításblokkjai együtt alkotnak egy blokkot, ezért csak a megfelelő mennyiségű utasításblokk adható meg a számukra, amelyekbe csak utasítások illeszthetők (ScratchJr, Scratch és Alice) (5. ábra).

5. ábra: Elágazás a Scratch-ben

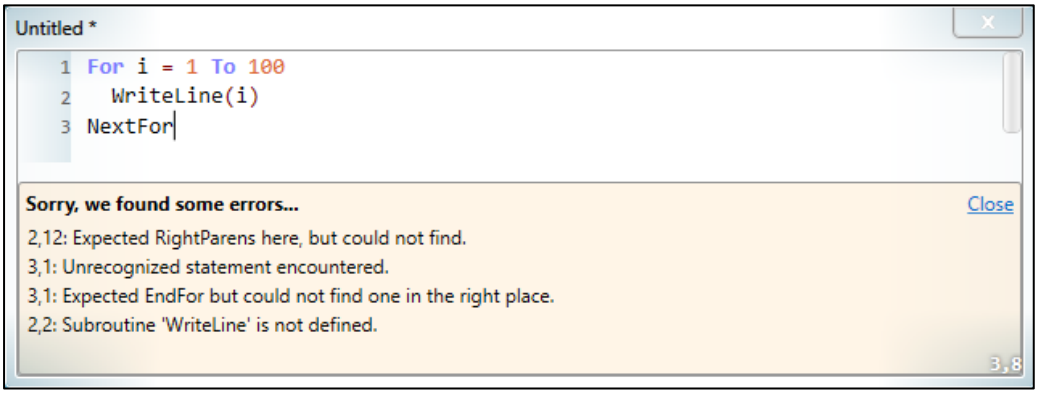
A szövegalapú kódolás során helyzetérzékeny súgó informálhatja a programozót a szükséges utasításblokkok számáról (Small Basic), de ennek ellenére megadható tévedésből túl kevés vagy túl sok utasításblokk (például a ha–akkor–különben szerkezet számára csak egy), és azokba nemcsak utasítások írhatók.

2.4. Szintaktikai helyesség és a szintaktikai hiba jelzése

A blokkalapú programozás során tehát csak létező és jól paraméterezett nyelvi elemek használhatók, a vezérlési szerkezetek is csak a megfelelő számú, és kizárólag utasításokat tartalmazó utasításblokkokból állíthatók össze, továbbá nincsen szükség azokra a határolójelekre, amelyekkel a szöveges programozási nyelvekben például az utasításokat és a paramétereket tagolják. Ezeknek köszönhetően a blokkokból csak szintaktikailag helyes programkód állítható össze. A programozó azzal, hogy valamely lépése szintaktikai hibát okozna (például számadatot szeretne felhasználni logikai feltételként), anélkül szembesülhet, hogy a hiba ténylegesen létrejönne, és később a futó program hibaüzenettel leállna (ScratchJr, Scratch és Alice). Így a tévesztés nem jár kudarccal, a visszacsatolás mégis azonnali (6. ábra).

6. ábra: Számérték nem, de logikai értékű kifejezés lehet az elágazás feltétele a Scratch-ben

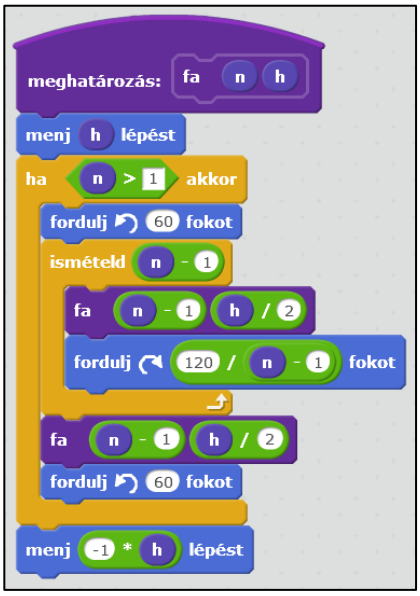
A szövegalapú programozás közben a fentebb ismertetett segítségek ellenére is (ha egyáltalán azok rendelkezésre állnak) szintaktikai hibák véthetők. A tévesztésekre csak a programfutás indításakor vagy megszakadásakor megjelenő hibaüzenetek hívják fel a figyelmet (Imagine Logo, RoboMind és Small Basic). Mivel a visszajelzések késnek, a hiba helye és oka utólag nem mindig egyértelmű, a sok hibaüzenet pedig frusztráló lehet (7. ábra).



7. ábra: Megjelenő hibaüzenetek a program indításakor a Small Basicben

2.5. Olvashatóság

A blokkalapú programozási nyelvekben még az összetett kifejezések esetén is jól látható, hogy az egyes értékek és részkifejezések mely nyelvi elemnek a paraméterei (Scratch és Alice) (8. ábra, bal oldal). A szövegalapú kódolásban általában a zárójelezés segíti az összetett kifejezések kiolvasását (RoboMind és Small Basic), a zárójelpárokat azonban meg kell találni. Ráadásul az oktatási célú nyelvek között található olyan is, amelyben a paraméterek – a nyelv más szempontú egyszerűsítése érdekében – nincsenek zárójelbe téve (Imagine Logo) (8. ábra, jobb oldal).



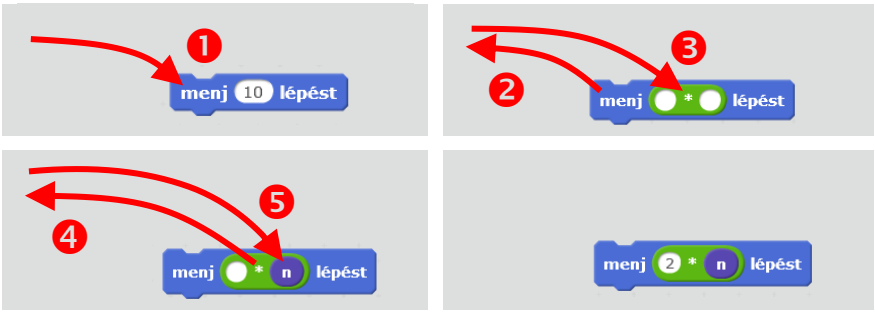

```
eljárás fa :n :h

előre :h
ha :n > 1 [
balra 60
ismétlés :n - 1 [
fa :n - 1 :h / 2
jobbra 120 / (:n - 1)]

fa :n - 1 :h / 2
balra 60]

hátra :h
vége
```

8. ábra: Ugyanazon feladatot végrehajtó program egymásnak megfeleltethető sorai a Scratch-ben és az Imagine Logóban

A blokkalapú nyelvekben az utasításblokkok egymásba ágyazásai is jól áttekinthetők, programozói beavatkozás nélkül (ScratchJr, Scratch és Alice) (8. ábra, bal oldal). Ugyanezt a szövegalapú kódolás során a különböző mértékű behúzások alkalmazásával szokták elérni. Ha azonban a programozó nem igazítja a sorokat, az utasításblokk-struktúra nehezebben áttekinthetővé válik (8. ábra, jobb oldal). Néhány fejlesztői környezet ezért felkínálja a sorok utólagos automatikus igazításának a lehetőségét (Small Basic és RoboMind).

2.6. A bevitel sebessége

A blokkalapú programozási környezetekben egy új nyelvi elem beszúrásához először ki kell választani a blokk-készlet megfelelő kategóriáját, majd azon belül a szóban forgó nyelvi elemet, amelyet végül be kell húzni a programkód megfelelő helyére (ScratchJr, Scratch és Alice). Így kell eljárni minden egyes kulcsszó, azonosító, sőt operátor esetén is (9. ábra).



9. ábra: A jobb alsó fázisban látható utasítás összeépítéséhez szükséges egérmozgások a képernyő bal szélén található (a képen nem látható) blokk-készlet és a programozási terület között a Scratch-ben

Ugyanezek a nyelvi elemek egy szövegalapú programozási nyelv fejlesztői környezetében gyorsabban begépelhetők (Imagine Logo, RoboMind és Small Basic), még akkor is, ha nem áll rendelkezésre az automatikus kódkiegészítés funkció.

2.7. Terjedelem

A blokkokból összeállított programkód lényegesen több helyet foglal el a vele egyenértékű szöveges kódnál. Ez a blokkok méretén túl annak is köszönhető, hogy a blokkokat csak egymás után lehet csatlakoztatni (ScratchJr, Scratch és Alice), miközben a legtöbb szövegalapú programozási nyelvben az utasítások egy sorba is írhatók (Imagine Logo és RoboMind).

2.8. Szerkesztés más környezetben

A blokkalapú programkódok csak a fejlesztői környezeten belül szerkeszthetők, és azon kívül csak képként használhatók fel (ScratchJr, Scratch és Alice). Ezzel szemben a szöveges programkódok tetszőleges kód- vagy szövegszerkesztőben módosíthatók, majd onnan vissza is másolhatók a fejlesztői környezetbe (Imagine Logo, RoboMind és Small Basic).

3. A kétféle kódolás értékelése

A kezdő programozók számára a blokkalapú programozás a megfelelőbb: elsősorban azért, mert nem lehet szintaktikai hibát ejteni, a blokkok formái pedig fokozzák a programkód olvashatóságát. További előnyt jelenthet, hogy a blokkokon szerepelhetnek piktogramok – ezáltal elérhetővé téve a kódolást a legkisebbek számára –, vagy pedig az élőbeszédhez közelebb álló bővebb kifejezések is, mivel azokat nem kell begépelni. Az önálló kísérletezést támogatja az is, hogy az összes nyelvi elem elérhető a blokk-készletben. Kutatások szerint is a kezdő programozók számára könnyebb a blokkalapú programozás [2], és a szövegalapú programozásban mérhetően jobb eredményeket érhetnek el azok, akik korábban blokkalapú programozást tanultak [3].

Ugyanakkor nem véletlen, hogy a professzionális programozásra a szöveges kódolás a jellemző: a billentyűzethasználatban gyakorlott programozók gyorsabban viszik be a szöveget gépelés útján. A haladó programozók számára, akiknek a szintaktikai szabályok betartása már kisebb gondot jelent, és akik már jelentős mennyiségű programot állítanak elő, a szövegalapú kódolás a hatékonyabb.

4. Az interjúalanyok véleménye a kétféle kódolásról

Az interjú olyan kvalitatív kutatási módszer, amely mélyre ható kikérdezést tesz lehetővé, és ezáltal nézőpontokat, indoklásokat és összefüggéseket tárhat fel a kutató számára [4]. Az első interjúalanyom négy éve oktat 2–8. osztályos gyermekeket egy programozóiskolában egy olyan oktatási platformon, amelyben a szövegalapú és a blokkalapú kódolás is lehetséges. A második hatosztályos elitgimnáziumban tanít a normál órákon szövegalapú, programozószakkörein azonban blokkalapú programozási nyelveket is. A harmadik egy szintén jónevű, nyolcosztályos gimnázium informatikatanára, aki azonban csak szövegalapú programozási nyelveket tanít.

Anélkül, hogy alanyaimnak konkrét szempontokat adtam volna, arra kértem őket, hogy hasonlítsák össze a kétféle beviteli módszert, saját tapasztalataik alapján. Az interjúk anonim módon hangrögzítéssel készültek, feldolgozásukhoz pedig a kategorizáció [5] módszerét használtam, amelynek megfelelően először a blokkalapú, majd a szövegalapú kódolás mellett szóló érveket csoportosítottam.

4.1. A blokkalapú kódolás mellett szóló érvek

Interjúalanyaim megfogalmazták, hogy a blokkalapú programozás során **a nyelvi elemek könnyebben elérhetők.** „Nem szükséges tudni, hogy milyen utasítások léteznek, azok a menüben megtalálhatók, tanári segítség nélkül is." (3. interjúalany)

Kiemelték, hogy a blokkokból könnyebben állítható elő **szintaktikailag helyes programkód.** „A blokkos programozás azért jó, mert szintaktikai hibát nem lehet ejteni, csak szemantikait; ebből a szempontból jobb, mint a szöveges kódolás." (1. interjúalany) „Könnyebb vele hibátlan kódot előállítani kezdő programozóként." (2. interjúalany)

A blokkokból összeállított **kód jobb olvashatóságáról** is beszéltek. „[A blokkalapú programozás] véleményem szerint áttekinthetőbb is [mint a szövegalapú]: például jobban képes vizualizálni az egymásba ágyazott ciklusok vagy elágazások viszonyát. A kód igazítása minden esetben hibátlan és egységes, míg például a szöveges kódolás igazítására többféle szokás terjedt el, amelyek közül néhány gyakran megnehezíti számomra a kód olvasását." (1. interjúalany)

Lényegesnek tartották, hogy a blokkokkal **kiváltható a gépelés.** „Egy alsó tagozatos tanuló számára nagy könnyebbséget jelent, ha nem kell begépelnie az utasításokat, legfeljebb csak a paramétereiket." (1. interjúalany) „[A blokkalapú nyelvek] képesek elérhetővé tenni a programozást a betűket és a számokat még nem ismerő kisgyermekek, valamint a rendesen még gépelni nem tudó fiatalok számára, körülbelül 4-6. osztályig." (2. interjúalany)

4.2. A szövegalapú kódolás mellett szóló érvek

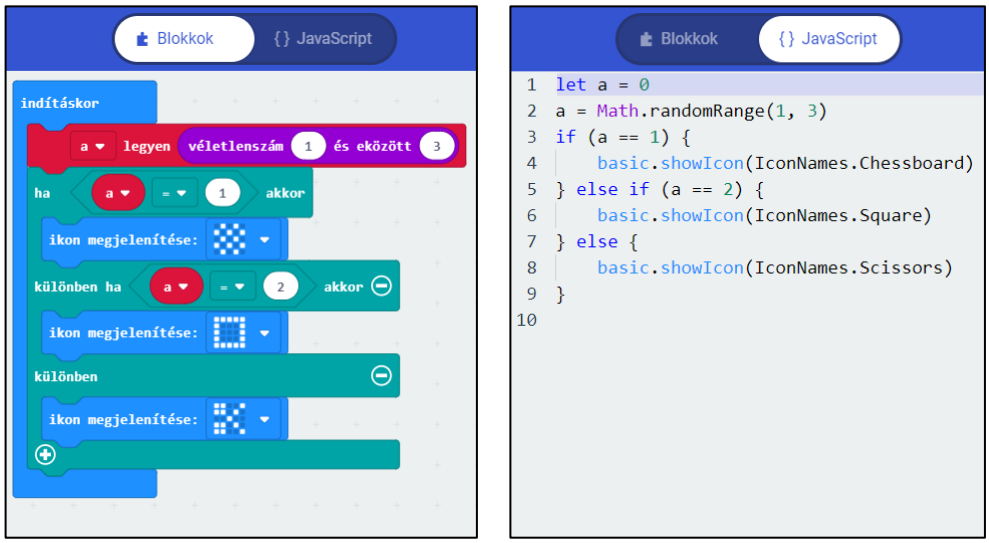
Interjúalanyaim rámutattak a szövegalapú kódolás előnyeire is, így például a **gyorsabb bevitelre.** „Egy nagyon egyszerű problémát nyilván nagyon könnyű blokkokkal megoldani. Ahogy azonban nő a feladat komplexitása, úgy csökken az, hogy mennyire hatékony ez a kódlétrehozási módszer, hiszen időigényessé válik a megoldás »összelegózása« az egérrel, a gépelés helyett." (1. interjúalany) „Egy »e« betűt leírni, és utánaírni a számot, sokkal gyorsabb, mint egy panelon kikeresni az előre-utasítást, egérrel behúzni a képernyőre, majd utána ugyanúgy begépelni a megfelelő helyre a paramétert." (3. interjúalany)

Rávilágítottak továbbá, hogy praktikusabb a szöveges programkódok **kisebb terjedelme.** „Egy nagyobb alkalmazás kódját összerakni és áttekinteni is nagyon nehéz a blokkalapú környezetekben." (2. interjúalany) „Ha már nagyon sok doboz van a képernyőn, akkor sem a távoli, sem a közeli nézetben nem lehet jól áttekinteni a programot, és gyakran nem világos, hogy mikor mi történik. Természetesen a szöveges kód is lehet hosszú, és az sem feltétlenül fér el egy képernyőn, ugyanakkor ott motiváltabb az eljárások bevezetése, amelyekre a megfelelő helyen elég hivatkozni. Azt gondolom tehát, hogy amikor már egy bizonyos szint felett van egy tanuló, a szöveges programozás átláthatóbb." (3. interjúalany)

Interjúalanyaim tapasztalatai, érvei és ellenérvei alátámasztják a korábbi értékelést.

5. Áttérés a blokkalapú kódolásról a szövegalapúra

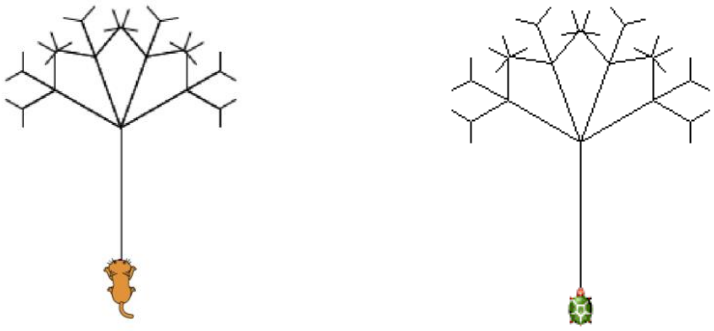
A blokkalapú programozásról a szövegalapúra áttérni egy olyan fejlesztői környezetben a legkönnyebb, amely támogatja a kétféle kódolási mód közötti folyamatos konvertálást, vagyis amelyben ugyanaz a (helyes) programkód bármikor megtekinthető és szerkeszthető az egyik vagy a másik nézetben. Ilyen például a Micro:bit (microbit.org) miniszámítógép Microsoft MakeCode (makecode.microbit.org) elnevezésű kódszerkesztője, amelyben egy blokkalapú, valamint egy a JavaScript szintaktikájára épülő szöveges programozási nyelv között lehet váltogatni (10. ábra).



10. ábra: Ugyanaz a program blokkos és szöveges nézetben a Microsoft MakeCode-ban

Természetesen a szintaktikailag hibás szöveges programkód nem futtatható és nem is konvertálható vissza a blokkalapúba. Ebben az esetben kérhetjük a legutóbbi hibátlan változat visszaalakítását blokkokká.

De a blokkalapú programozásról a szövegalapúra áttérés akkor is elképzelhető, ha a blokkalapú fejlesztői környezet nem biztosítja a kétféle kódbevitel közötti váltogatást. Annak érdekében, hogy egyszerre csak egy újdonság legyen – tudniillik a megváltozott programozási nyelv –, célszerű a programozást ugyanazon témakörön belül folytatni. Erre biztosít lehetőséget például a Scratch és az Imagine Logo programozási nyelve, az előbbi ugyanis blokk-, az utóbbi pedig szövegalapú, és mindkettővel megoldható a teknőcgrafika számos feladattípusa (11. ábra). Elérhetők bennük az automata elven irányítható és tollal rendelkező objektumok (szereplők, illetve teknőcök), mozgatóutasításaik és a tollhasználat parancsai pedig javarészt megfeleltethetők egymásnak.



11. ábra: A 8. ábrán látható, egymásnak utasításonként megfeleltethető programkódok futtatásának eredménye a Scratch-ben és az Imagine Logóban

A Logo versenyfeladatok megoldása a Scratch programozási nyelven [6] című kiadványom segítséget nyújthat a két nyelv közötti váltásban, amelyben összehasonlítottam a két programozási nyelv teknőcgrafikai lehetőségeit, és számos Logo-versenyfeladat megoldását ismertettem a Scratch-ben, feladattípus szerinti csoportosításban.

6. Az interjúalanyok tapasztalatai az áttérésről

Első interjúalanyom arról számolt be, hogy programozóiskolájuk diákjai „a szöveges kódolást és annak kulcsszavait nagyon izgalmasnak találják, és nagyon hamar meg akarják tanulni az összeset", és „nem szokták »visszasírni« a blokkalapú kódbevitelt". Ez vélhetően annak köszönhető, hogy jórészt a programozásra nagyon fogékony tanulók járnak abba az iskolába.

A gimnáziumban tanító két beszélgetőpartnerem azonban arról vallott, hogy a blokkosan programozó diákjaikat sokszor nekik kell motiválttá tenniük. „Akinek a blokkalapú programozás jól megy, és azt szereti, az önmagától nem igazán szeretne áttérni a szöveges kódolásra. […] Néhányan rájönnek például egy egyszerű prímszámkeresési feladat megoldása során arra, hogy azt a Scratch-ben nem lehet hatékonyan megvalósítani. És akkor közösen kereshetünk egy olyan programozási nyelvet, amelyen hatékonyabban lehet programozni. Ugyanakkor az a tapasztalatom, hogy magától a Scratch-et használók többsége nem jut el eddig a gondolatig." (2. interjúalany) „Fájdalmas volt a számára [az áttérés], ugyanis a blokkok használatában már nagyon rutinos volt. Gépelni pedig nagyon nem szeretett." (3. interjúalany) Második interjúalanyom viszont kiemelte, hogy a szövegalapú kódolás során a korábban blokkalapon programozó diáknak „a programozásról és a programok felépítéséről kialakult szemlélete nagyon jól jön, azt nem kell újra tanítani. Könnyen megérti a vezérlési szerkezeteket, és nagyobb nehézségek nélkül megszokja a szintaktikai szabályokat is."

7. Konklúzió

Cikkemben a kétféle kódlétrehozási lehetőséget az oktatási célú programozási nyelvek más adottságaitól függetlenül hasonlítottam össze és értékeltem, amelynek alapján megállapítottam, hogy a kezdők számára a blokkalapú, a haladók számára viszont a szövegalapú kódbevitel az előnyösebb.

Úgy gondolom, hogy a kezdőknek életkortól függetlenül is a blokkos programozás javasolható, amelyet nemcsak a köznevelésben, de a felsőoktatásban is egyre bővülő körben használnak a bevezető programozástanításban [7]. Személyes meggyőződésem, hogy habár a haladó programozás során a szöveges kódolás egyértelműen hatékonyabb, bármely programkód blokkalapú nézete és szerkesztése közelebb áll a programozási nyelv struktúrájához az egyszerű szövegnél.

Irodalom

1. Kelleher, C. & Pausch, R., 2005. *Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers.* ACM Computing Surveys, 37(2), pp. 83-137.
2. Weintrop, D. & Wilensky, U., 2015. *To block or not to block, that is the question: students' perceptions of blocks-based programming.* 14th International Conference on Interaction Design and Children, pp. 199-208.
3. Armoni, M., Meerbaum-Salant, O. & Ben-Ari, M., 2015. *From Scratch to "real" programming.* Transactions on Computing Education, 14(4).
4. Falus, I. & Ollé, J., 2008. *Az empirikus kutatások gyakorlata – Adatfeldolgozás és statisztikai elemzés.* Oktatáskutató és Fejlesztő Intézet, Budapest.
5. Schleicher, N., 2007. *Kvalitatív kutatási módszerek a társadalomtudományban.* Századvég, Budapest.
6. Bernát, P., 2017. *Logo versenyfeladatok megoldása a Scratch programozási nyelven.* ELTE Informatikai Kar, Budapest.
   http://logo.inf.elte.hu/tanaroknak/logo-scratch%20v11.pdf (utoljára megtekintve: 2019.11.10.)
7. code.org, 2014. *Why top universities teach drag and drop programming.*
   https://www.youtube.com/watch?v=_Mwc1gc77dc (utoljára megtekintve: 2019.11.10.)