Posredovani prijenos u poučavanju programiranja s vizualnim programskim jezicima

Divna Krpan[1], Damir Brčić[2]

[1] *Prirodoslovno-matematički fakultet, Sveučilište u Splitu,*

[2] *Splitsko-dalmatinska županija, Split*

Sažetak

Percepcija da je programiranje teško negativno utječe na motivaciju početnika koji odustaju već kod prvih neuspjelih pokušaja. Pojednostavljivanjem sintakse moguće je spustiti početni prag kojeg početnici teško prelaze, a odabir jezika i okruženja primjerenog dobi obećava pomicanje dobnih granica za početno programiranje čime bi se potencijalno moglo zainteresirati više učenika za buduće programerske poslove. U radu se opisuju primjeri didaktičkih alata za poučavanje programiranja te iskustvo u primjeni.

Uvod

Djeca danas odrastaju okružena tehnologijom kod kuće i u školama u obliku mobitela, tableta, računala i ostalih „pametnih“ uređaja. Često ih nazivaju „net generacijom“ ili „digitalnim urođenicima“ (eng. Digital natives) [1] koji uče u „digitalnom kontekstu“ [2]. Takva djeca, odnosno učenici zahtijevaju drugačije metode učenja i poučavanja kao što je primjerice aktivno učenje, interakcija, rješavanje problema i sl. Međutim, događa se da nastavnici poučavaju programiranje na isti način kako su i sami učili iako su i tehnologija i učenici drugačiji. Učenje programiranja je teško bez obzira na dob učenika jer učenici moraju naučiti rješavati probleme, razumjeti izvršavanje programa te usvojiti strogu sintaksu programskog jezika [3]. Uobičajen pristup poučavanju programiranja je poučavanje osnovnih koncepata te nakon toga usmjeravanje prema složenijim konceptima i strategijama programiranja [4]. Za izradu složenih algoritama dovoljne su tri osnovne algoritamske strukture: slijed, grananje i ponavljanje [5].

Aktivno sudjelovanje učenika zahtijeva motivaciju, a percepcija da je programiranje teško može imati negativan učinak pa je stoga potrebno pronaći način kako olakšati početno programiranje i motivirati buduće programere. Splitsko-dalmatinska županija (SDŽ) pokrenula je projekt *ICT Županija* (ICT: Information and Communication Technology) koji nizom aktivnosti potiče razvoj ICT sektora županije (https://www.ictzupanija.hr/). Osnovne aktivnosti podijeljene su na edukativni i poslovni dio. U okviru edukativnog dijela osnovani su *Edukacijski IT centri* (EDIT) za suvremeno informatičko educiranje nastavnika informatike. Nastavnici informatike iz osam osnovnih i devet srednjih škola SDŽ koji su

uključeni u prvu fazu edukacije, u drugoj fazi su i sami postali edukatori suvremenih Škola programiranja namijenjenih širokom krugu zainteresirane djece u osnovnim i srednjim školama. Cilj edukacije je popularizirati i omogućiti dostupnim svijet programiranja što većem broju djece u županiji. U prvom dijelu rada opisane su osnovne vrste jezika koje se koriste za početno programiranje primjereno razvoju i dobi učenika, a u drugom dijelu rada opisan je pristup poučavanju programiranja metodom didaktičkog skrivanja i posredovanog prijenosa korištenjem didaktičkih pomagala za poučavanje programiranja nastalih na ideji vizualnih programskih jezika u suradnji s istraživačkom grupom AI-COMPILE. Razvoj jednog od pomagala opisanog u tom dijelu je upravo potaknut aktivnostima EDIT-a.

Kognitivni razvoj i način poučavanja programiranja

Učenje programiranja sastoji se od učenja koncepata programiranja, sintakse i učenja snalaženja u okruženju za programiranje [6]. Obzirom da je sintaksa profesionalnih programskih jezika često složena, teško se usredotočiti na rješavanje problema dok se treba paziti na svaki znak koji se upisuje. Neki od tih jezika imaju poveznica s prirodnim engleskim jezikom (npr. slične riječi), ali su često te sličnosti više problem nego pomoć. S ciljem pojednostavljenja razvijeni su „mali“ programski jezici namijenjeni poučavanju koji sadrže manji skup naredbi te se takvi jezici nazivaju „mini jezici“, a pojam mini jezika prvi put uvodi Brusilovsky [7], a sama ideja je počela još u sedamdesetim godinama od Paperta i Logo programskog jezika [8].

Vizualni programski jezici

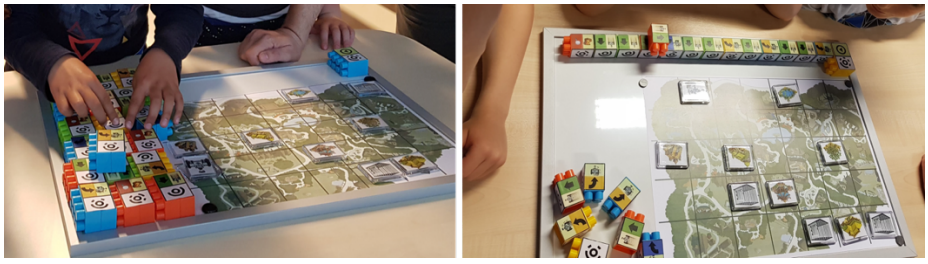
Razvojem računalne podrške stvorile su se dodatne grafičke mogućnosti pa su se pojavili i jezici kojima su naredbe predstavljene grafičkim blokovima: *vizualni programski jezici* kao nasljednici Logo programskog jezika. Tekstualni dijelovi programskog koda su zamijenjeni vizualnim blokovima koji su najčešće u obliku slagalice (eng. Puzzle) kao što je to slučaj kod programskog jezika *Scratch* (https://scratch.mit.edu/).

Vizualizacije i okruženja za programiranje koriste vizualne elemente ili animacije kako bi se početnicima olakšali prvi koraci u programiranju jer na taj način rade s konkretnijim stvarima [9]. Smatra se da je potreba za vizualizacijom u informatici toliko prisutna zbog same apstraktne prirode sadržaja. Vizualni programski jezici se globalno mogu podijeliti na: potpuno vizualne (svi elementi su vizualni) i hibridne (postoji podrška za tekst). Postoji mnogo vizualnih programskih jezika, kao što su na primjer: Alice, Greenfoot, Karel J. Robot, MicroWorlds, Google Blockly, App Inventor, Squeak, StarLogo i sl.

Prednost programskog jezika Scratch-a je mogućnost odabira prirodnog jezika koji će pisati na blokovima naredbi kako bi učenicima bilo što razumljivije. Trenutno se Scratch koristi u 150 zemalja i preveden je na 40 jezika. U travnju 2018 zabilježeno je preko 27 milijuna registriranih korisnika. Oblikovan je za dob od 8-16 godina no nije ograničen na tu dob. Scratch ima nizak prag (eng. Low floor) i visok strop (eng. High ceiling) [10]. To znači da je lako s njim početi, a može se mnogo postići. Smatramo da je primjeren za učenje i poučavanje programiranja kad djeca ovladaju čitanjem i motoričkim sposobnostima za osnovnu upotrebu računala vjerojatno od drugog razreda osnovne škole do šestog razreda gdje bi se postupno trebalo prijeći na tekstualne programske jezike. Na Sveučilištu Berkeley razvijen je njegov dijalekt: Byob (Build Your Own Blocks) koji je bio odgovor na sve prigovore, odnosno nedostatke vezane za tada aktualnu verziju Scratch-a (1.4), te nakon toga i nasljednik Byob-a: Snap! (http://snap.berkeley.edu/). Neki autori ističu kako vizualni jezici ipak ograničavaju dob učenika jer zahtijevaju razumijevanje teksta koji piše na blokovima te spretnost u radu s mišem i tipkovnicom te da su za najmlađu djecu primjereniji *opipljivi* (eng. tangible) programski jezici [11].

Opipljivo programiranje

Prema teoriji razvojnih faza djece J. Piaget-a, djeca u dobi od 7-11 godina ulaze u konkretnu fazu u kojoj razvijaju logičko razmišljanje i sposobnosti rješavanja problema [12] te bi bilo logično da djecu treba poučavati u skladu s njihovim razvojnim fazama. Prema nekim autorima smatra se da bi djeca mogla ući u neke faze i ranije, ali pod uvjetom da im se problemi ili sadržaji stave u primjereni, odnosno razumljivi kontekst [13]. Djeca u dobi od 5-8 godina razvijaju veze između konkretnih i simboličkih prikaza upotrebom stvarnih objekata [14]. Takvi objekti mogu biti kocke, kartice, roboti i sl. Ako se ti objekti upotrijebe u kontekstu programiranja kao elementi programa onda se dobije jezik koji je *opipljiv*, te imamo *opipljive programske jezike* [15]. Kod opipljivog programiranja naredbe su predstavljene fizičkim predmetima, u obliku kocaka, pločica ili kartica i sl. (Slika 1)



Slika 1. Opipljivo programiranje

Zanimljivo je da je zapravo najraniji primjer opipljivog programiranja iz 1970-tih, a radilo se o elektromehaničkom uređaju pod nazivom TORTIS (Toddler's Own Recursive Turtle Interpreter System) kojeg je oblikovala Radia Perlman [16]. Autorica je krenula s idejom da čak i djeca od 3-5 godina mogu naučiti programirati, a koristila je plastične kartice kao opipljive ulazne naredbe, te Papert-ovu Logo kornjaču kao opipljivi izlaz tj. vezu programa sa stvarnim svijetom. TORTIS uređaj je bio samostalan, odnosno nije imao vezu s računalom, a jedan od prvih opipljivih programskih jezika povezan s računalom je AlgoBlocks iz 1990-tih [17]. Istraživanje koje su proveli Sapounidis i Demetriadis [18] na tri dobne skupine djece (5-6, 7-8 i 11-12 godina) pokazalo je da je opipljivo sučelje djeci privlačno svima, pogotovo kao novo iskustvo, ali su starija djeca ipak radije birala grafičko sučelje na računalu radi brzine rada.

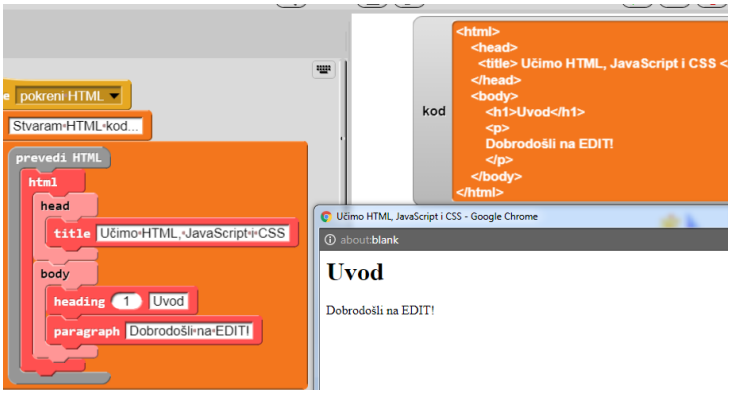
Skrivanje detalja i posredovani prijenos

Primjenom metode didaktičkog skrivanja [19] odnosno skrivanja složenosti detalja može se pomoći pri smanjivanju kognitivnog opterećenja učenika [20]. Na primjer, ako lik u igri treba prijeći put od *n* točkica pod zadanim kutom, to u dvodimenzionalnom prostoru zapravo znači da bi trebalo izračunati koliko je to točkica po x, odnosno po y osi što se može izračunati uz pomoć trigonometrijskih funkcija. Navedeni primjer može poslužiti kao povezivanje koncepata iz matematike s programiranjem, ali nije dobar za djecu nižih uzrasta koji još nisu učili trigonometriju te oni ne moraju znati što je „u pozadini" naredbe za pomak.

Prijenos učenja (eng. Transfer of learning) je sposobnost primjene znanja i vještina usvojenih u jednom kontekstu na nove kontekste. Događa se kad učenik prepozna neke slične karakteristike, poveže potrebne informacije u memoriji te shvati da može primijeniti koncepte iz poznatog konteksta [21]. Ponekad nastavnik treba pomoći prilikom prijenosa pa se radi u *posredovanom prijenosu* (eng. Mediated transfer).

Vizualni programski jezici mogu pomoći i odraslim učenicima, odnosno studentima, ali oni takve jezike brzo "prerastu" [22] te im treba omogućiti prijelaz u tekstualno okruženje. Kontekst izrade jednostavnih igara u Scratch-u je zanimljiv i motivirajući, ali izrada ekvivalentnih igara u profesionalnom programskom jeziku kao što je npr. C# zahtijeva poznavanje dodatnog skupa koncepata. U skladu s tim, odlučili smo posredovati, odnosno pomoći studentima u prijenosu koncepata na način da smo im pripremili posebno okruženje u kojem mogu izraditi jednostavne igre [23]. Testirali smo rad prototipa za posredovani prijenos iz vizualnog jezika na maloj skupini studenata koji nikad ranije nisu vidjeli C#, ali su ipak uspjeli izraditi igru te prenijeti i izvršiti C# kod u okruženju Microsoft Visual Studia.

Na temelju ideja vizualnog programskog jezika tipa Scratch i posredovanog prijenosa izradili smo okruženje HELIoS (**H**ypert**E**xt Markup **L**anguage Javascr**I**pt and Cascading **S**tyle Sheets) za poučavanje programiranja upotrebom HTML-a, JavaScripta i CSS-a.



Slika 2. Okruženje za poučavanje HELIOS

Okruženje je korišteno za educiranje nastavnika informatike u okviru Edukacijskog IT centra (EDIT) SDŽ. Cilj je stvoriti ugodno okruženje u kojem će učenici pomoću vizualnih blokova moći naučiti osnove spomenutih jezika te na kraju izraditi aplikacije za mobilne uređaje. Primjenom blokova i metode didaktičkog skrivanja učenici koriste prednosti „niskog praga“ vizualnog programskog jezika i smanjivanja kognitivnog opterećenja. Učenici i nastavnici u bilo kojoj fazi rada mogu prijeći iz vizualnog HELIoS okruženja u tekstualno. Prednost HELIoSa je što omogućuje rad s blokovima, generira programski kod u obliku teksta te prikazuje rezultat izvršavanja tog koda. Na Slici 2 su vidljive sve tri funkcionalnosti. Dodatno, učenici mogu spremiti datoteke s kodom te ih pokrenuti u web pregledniku neovisno o samom okruženju.

Upotreba Scratch-a i sličnih programskih jezika u nastavi također znači da treba vrednovati i ocijeniti zadatke koji se rješavaju u takvim okruženjima.

Vrednovanje zadaća u vizualnim programskim jezicima

Vrednovanje je jedna od najčešćih zadataka koje nastavnici moraju obaviti, a cilj je da nastavnici shvate koliko učenici znaju te da učenici dobiju povratnu informaciju o svom znanju [9]. Potrebno je vrednovati različite aspekte znanja i kognitivnih sposobnosti učenika. Međutim, pregledavanje rješenja u vizualnom jeziku je vremenski zahtjevno. Svako rješenje se mora posebno otvoriti i pokrenuti, a ako rješenja sadrže multimedijske sadržaje onda treba gledati i slušati. Ako je u igri planirano više razina onda nastavnik često treba „odigrati“ te razine radi provjere funkcionalnosti.

Studenti su tijekom rada u Scratch-u [22] imali više zadaća koje je trebalo pregledati i ocijeniti: zadaci na vježbama, zadaci za samostalan rad kod kuće, završni projekt i kolokvij.

Ukupan broj zadataka po studentu je na kraju bio oko 15-20 ovisno o akademskoj godini i prisutnosti, no za 35 studenata upisanih na predmet radi se o približno 500-700 rješenja koje nastavnik tijekom semestra treba pregledati. Obzirom da se Scratch sastoji od grafičkih naredbi, na prvi pogled se čini da ne postoji mogućnost automatskog ocjenjivanja, međutim Boe i dr. su za tu namjenu izradili biblioteku koju su nazvali *Hairball* [24]. Na temelju biblioteke razvijen je online alat *Dr. Scratch* kojeg mogu koristiti učenici i nastavnici za provjeru projekata [25]. Prednost automatskog ocjenjivanja je za domaće zadaće [26] ili ispitivanje trenda upotrebe nekog programskog koncepta [27]. Ovdje je važno napomenuti da se na ovaj način može vidjeti da učenici koriste neke koncepte ili blokove, ali postavlja se pitanje znači li to da je učenik taj koncept usvojio i je li koncept upotrijebljen smisleno?

Pomoću Hairball biblioteke koja nam je omogućila analizu u obliku ciklomatske složenosti (eng. cyclomatic complexity, CC) i Halstead-ove metrike (eng. Halstead's metrics, HM) [28] te način ocjenjivanja opisan kod Dr. Scratch-a, analizirali smo složenost ukupno 55 rješenja zadaća za studente iz ak. godine 2015/16 i 2016/17.

2015/16

Attributes	DrScratch	Cyclo	timeSec	ManualGrade
DrScratch	1	0.463	0.458	0.464
Cyclo	0.463	1	0.556	0.529
timeSec	0.458	0.556	1	0.597
ManualGrade	0.464	0.529	0.597	1

2016/17

Attributes	DrScratch	Cyclo	timeSec	ManualGrade
DrScratch	1	0.545	0.413	0.312
Cyclo	0.545	1	0.700	0.353
timeSec	0.413	0.700	1	0.522
ManualGrade	0.312	0.353	0.522	1

Slika 3. Korelacija automatskog ocjenjivanja i ocjene nastavnika

Za ak. god. 2015/16 je postojala korelacija između CC složenosti i ocjene nastavnika (stupac označen s *ManualGrade*) od 0.59, a za ak. god. 2016/17 od 0.52. Dobivene korelacije nisu značajne. Zadaci s kolokvija su imali točno definirane zahtjeve, npr. za ponašanje likova, interakcije s drugim likovima i korisnikom, uvjete za završetak igre i sl. Događalo se da studenti naprave igru koja se ne može igrati jer se npr. likovi kreću prebrzo, nisu vidljivi, prebrzo mijenjaju vrijednosti varijable zbog pogrešnog koda za dodir i sl. Takve stvari se nisu mogle ocijeniti automatski.

Zaključak

Upotrebom primjerenog jezika i okruženja početni koraci u programiranju mogu se olakšati i približiti većem broju učenika. Zaključujemo da vizualni programski jezici mogu poslužiti kao početna točka ili *odskočna daska* za početak učenja programiranja bez obzira na dob. Smatramo da bi kao prvi susret s programiranjem za djecu od prvog do četvrtog razreda osnovne škole i ranije, primjerenije bilo koristiti opipljivo programiranje. Pri tome se ne misli

na više godina kontinuiranog opipljivog programiranja već dok ne usvoje osnovne koncepte i osnove rada s računalom kako bi se dalje moglo prijeći na vizualne programske jezike. Odabir vizualnog programskog jezika može utjecati na daljnji rad u tekstualnim ili profesionalnim programskim jezicima ukoliko postoji povezanost ili prijenos između tih jezika. Obzirom na multimedijsku prirodu Scratch-a i interaktivnost, nije se moguće u potpunosti osloniti na automatsko ocjenjivanje, ali se može koristiti kao pomoć pri vrednovanju zadaća posebno ako učenici rade slobodne projekte jer se u tom slučaju mogu usporediti prema složenosti.

Popis literature

1. M. Prensky, "Digital natives, digital immigrants part 1," *On the horizon*, vol. 9, no. 5, pp. 1–6, 2001.
2. C. C. Hernandez *et al.*, "Teaching Programming Principles through a Game Engine," *CLEI ELECTRONIC JOURNAL*, vol. 13, no. 3, 2010.
3. C. Kelleher and R. Pausch, "Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers:," Defense Technical Information Center, Fort Belvoir, VA, May 2003.
4. K. Ala-Mutka, "Problems in learning and teaching programming–a literature study for developing visualizations in the Codewitz-Minerva project," *Codewitz Needs Analysis*, pp. 1–13, 2004.
5. C. Böhm and G. Jacopini, "Flow diagrams, turing machines and languages with only two formation rules," *Communications of the ACM*, vol. 9, no. 5, pp. 366–371, 1966.
6. A. Robins, J. Rountree, and N. Rountree, "Learning and Teaching Programming: A Review and Discussion," *Computer Science Education*, vol. 13, no. 2, pp. 137–172, 2003.
7. P. Brusilovsky, E. Calabrese, J. Hvorecky, A. Kouchnirenko, and P. Miller, "Mini languages: A Way to Learn Programming Principles," *Education and Information Technologies*, vol. 2, no. 1, pp. 65–83, 1997.
8. S. Papert, *Mindstorms: children, computers, and powerful ideas*. New York: Basic Books, 1980.
9. O. Hazzan, T. Lapidot, and N. Ragonis, *Guide to teaching computer science: An activity-based approach*. Springer, 2015.
10. M. Resnick *et al.*, "Scratch: Programming for All.," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, Nov. 2009.
11. P. Wyeth, "How young children learn to program with sensor, action, and logic blocks," *Journal of the Learning Sciences*, vol. 17, no. April, pp. 517–550, 2008.
12. J. Piaget and B. Inhelder, *The growth of logical thinking from childhood to adolescence: An essay on the construction of formal operational structures*, vol. 84. Routledge, 2013.

13. K. Richardson, *Models of Cognitive Development*. Psychology Press, 1998.
14. L. P. Flannery, B. Silverman, E. R. Kazakoff, M. U. Bers, P. Bontá, and M. Resnick, "Designing ScratchJr: Support for early childhood learning through computer programming," in *Proceedings of the 12th International Conference on Interaction Design and Children*, 2013, pp. 1–10.
15. M. U. Bers and M. M. S. Horn, "Tangible programming in early childhood: Revisiting developmental assumptions through new technologies," *High-tech tots: Childhood in a digital world*, pp. 1–32, 2010.
16. R. Perlman, "TORTIS: Toddler's Own Recursive Turgle Interpreter System," Cambridge, 1974.
17. H. Suzuki and H. Kato, "AlgoBlock: a tangible programming language, a tool for collaborative learning," *Proceedings of 4th European Logo Conference*, no. June 2016, pp. 297–303, 1993.
18. T. Sapounidis and S. Demetriadis, "Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences," *Personal and Ubiquitous Computing*, vol. 17, no. 8, pp. 1775–1786, Dec. 2013.
19. G. Futschek, "Extreme didactic reduction in computational thinking education," in *X World Conference on Computers in Education*, 2013, pp. 1–6.
20. J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*, 1st ed., vol. 10. Springer-Verlag New York, 2011.
21. D. N. Perkins and G. Salomon, "Transfer of learning," *International Encyclopedia of Education, Second Edition*, vol. 2, pp. 1–11, 1992.
22. D. Krpan, S. Mladenović, and G. Zaharija, "Vizualni programski jezici u visokom obrazovanju," in *16. CARNetova korisnička konferencija - CUC 2014 - Zbornik radova*, Zagreb, 2014.
23. D. Krpan, S. Mladenović, and G. Zaharija, "Mediated Transfer from Visual to High-level Programming Language," in *MIPRO 2017*, 2017.
24. B. Boe, C. Hill, M. Len, G. Dreschler, P. Conrad, and D. Franklin, "Hairball: Lint-inspired Static Analysis of Scratch Projects," *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, pp. 215–220, 2013.
25. J. Moreno-León and G. Robles, "Dr. Scratch: a web tool to automatically evaluate Scratch projects," *Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE '15*, no. October 2015, 2015.
26. D. E. Johnson, "ITCH : Individual Testing of Computer Homework for Scratch Assignments," *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*, pp. 223–227, 2016.
27. E. Aivaloglou and F. Hermans, "How Kids Code and How We Know," in *Proceedings of the 2016 ACM Conference on International Computing Education Research - ICER '16*, 2016, pp. 53–61.
28. J. Moreno-Leon, G. Robles, and M. Roman-Gonzalez, "Comparing computational thinking development assessment scores with software complexity metrics," *IEEE Global Engineering Education Conference, EDUCON*, vol. 10-13-Apri, no. April, pp. 1040–1045, 2016.