Sveučilište u Rijeci
University of Rijeka
http://www.uniri.hr

Polytechnica: Journal of Technology Education, Volume 2, Number 1 (2018)
Politehnika: Časopis za tehnički odgoj i obrazovanje, Volumen 2, Broj 1 (2018)

Politehnika
Polytechnica
http://www.politehnika.uniri.hr
cte@uniri.hr

Stručni članak
Professional article
UDK 004.43:37.091.3

Posredovani prijenos u poučavanju programiranja s vizualnim programskim jezicima*

Divna Krpan
Prirodoslovno-matematički fakultet
Sveučilište u Splitu
Ruđera Boškovića 33, 21000 Split
dkrpan@pmfst.hr

Damir Brčić
Splitsko-dalmatinska županija
Upravni odjel zajedničkih poslova
Domovinskog rata 2, 21000 Split
damir.brcic@dalmacija.hr

Sažetak

Programeri početnici često smatraju programiranje teškim i zahtjevnim. Takva percepcija negativno utječe na motivaciju, a problemi na koje nailaze u prvim koracima učenja potvrđuju njihova očekivanja tako da vrlo brzo odustaju. Jednostavnija sintaksa i okruženje mogu početnicima olakšati prvi susret s programiranjem, odnosno spustiti prag kojeg moraju prijeći kako bi nastavili dalje prema složenijim programskim jezicima. Prema tome, važno je odabrati jezik i okruženje primjereno dobi učenika. U radu se opisuju primjeri didaktičkih alata za poučavanje programiranja za različitu dob i opis vrednovanja projekata u vizualnom programskom jeziku.

Ključne riječi: *vizualno programiranje; didaktičko skrivanje; opipljivo programiranje; posredovani prijenos.*

1. Uvod

Djeca danas odrastaju okružena tehnologijom kod kuće i u školama u obliku mobitela, tableta, računala, pametnih satova i sl. Često ih nazivaju "net generacijom" ili "digitalnim urođenicima" (eng. Digital natives) (Prensky, 2001) koji uče u digitalnom kontekstu (Hernandez, et al., 2010). Takva djeca, zahtijevaju drugačije metode učenja i poučavanja kao što su: aktivno učenje, interakcija, rješavanje problema i sl. Međutim, događa se da nastavnici poučavaju programiranje na isti način kao što su i sami učili programirati premda su 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 (Kelleher, Pausch, 2003). Uobičajen pristup poučavanju programiranja je poučavanje osnovnih koncepata te nakon toga usmjeravanje prema složenijim

** Paper is accepted for the 20th CARNET Users Conference CUC, 2018, Šibenik, Croatia*

konceptima i strategijama programiranja (Ala-Mutka, 2004). Za izradu složenijih algoritama dovoljne su tri osnovne algoritamske strukture: slijed, grananje i ponavljanje (Böhm, Jacopini, 1966).

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 bili 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 na Prirodoslovno-matematičkom fakultetu u Splitu (http://aicompile.com.hr/). Razvoj jednog od pomagala opisanog u tom dijelu je upravo potaknut aktivnostima EDIT-a.

2. Kognitivni razvoj i način poučavanja programiranja

Učenje programiranja sastoji se od učenja koncepata programiranja, sintakse i učenja snalaženja u okruženju za programiranje (Robins, Rountree, Rountree, 2003). 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 poveznice ili sličnosti s prirodnim engleskim jezikom, ali su često te sličnosti više problem nego pomoć. S ciljem pojednostavljenja razvijeni su "mali" programski jezici namijenjeni poučavanju koji sadrže ograničeni skup naredbi te se takvi jezici nazivaju "mini jezici". Premda pojam mini jezika prvi put uvodi Brusilovsky (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko, Miller, 1997), ideja potječe iz sedamdesetih godina s nastankom Papertovog programskog jezika Logo (Papert, 1980).

2.1. Vizualni programski jezici

Razvojem računalne podrške stvorile su se dodatne mogućnosti prikazivanja grafičkih elemenata na zaslonu računala. Naime, kornjačina grafika programskog jezika Logo je bila jednostavna (bez slika, male rezolucije, ograničenog skupa boja i sl.) jer su računala u 1970-tima bila bitno manjih mogućnosti. S boljim računalima, tekstualne naredbe zamijenjene su grafičkim ili vizualnim elementima koje nazivamo blokovima, a takve jezike *vizualnim programskim jezicima.* Blokovi su obično u obliku slagalice (eng. Puzzle), a tipičan primjer takvog jezika je *Scratch* (https://scratch.mit.edu/).

Vizualizacije i vizualna okruženja za programiranje koriste vizualne elemente (slike, ikonice i sl.) i animacije kako bi početnici mogli raditi s konkretnijim stvarima (Hazzan, Lapidot, Ragonis, 2015), a to je bila ideja LOGO jezika s kornjačom koju su učenici mogli vidjeti na zaslonu računala i pratiti kako se ponaša. 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 (https://www.alice.org/), Greenfoot (https://www.greenfoot.org/), Google

Blockly (https://blockly-games.appspot.com/), App Inventor (http://appinventor.mit.edu/), StarLogo (http://www.slnova.org/) i sl.

Prednost programskog jezika Scratch je mogućnost odabira prirodnog jezika koji će pisati na blokovima naredbi kako bi učenicima naredbe bile još 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 ograničeno. Scratch ima nizak prag (eng. Low floor) i visok strop (eng. High ceiling) (Resnick et al., 2009). To znači da je lako s njim početi, a može se mnogo postići. Smatramo da je Scratch primjeren za početno učenje i poučavanje programiranja kod djece koja su već ovladala čitanjem i motoričkim sposobnostima potrebnim za osnovnu upotrebu računala. Prema tome, Scratch bi mogli koristiti već od drugog razreda osnovne škole pa sve do šestog razreda kada 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). Nakon toga nastaje nasljednik Byob-a: Snap! (http://snap.berkeley.edu).

Neki autori ističu kako karakteristike vizualnih jezika utječu na dobnu granicu kad učenici mogu početi učiti programiranje jer se od učenika očekuje razumijevanje teksta koji piše na blokovima te spretnost u radu s mišem i tipkovnicom, ali da to nije nužno za neke osnovne koncepte. Prema tome, ako želimo još više spustiti početnu dobnu granicu za učenje programiranja, umjesto vizualnih primjereniji su *opipljivi* (eng. tangible) programski jezici (Wyeth, 2008).

2.2. 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 (Piaget, Inhelder, 2013) tako da je očekivani zaključak kako djecu treba poučavati u skladu s njihovim razvojnim fazama odnosno dobi koja toj fazi odgovara. Međutim, prema nekim autorima smatra se da bi djeca mogla ući u neke faze i ranije pod uvjetom da im se problemi ili sadržaji koje trebaju usvojiti stave u primjereni, odnosno njima razumljivi kontekst (Richardson, 1998). Djeca u dobi od 5-8 godina razvijaju veze između konkretnih i simboličkih prikaza upotrebom stvarnih objekata (Flannery, Silverman, Kazakoff, Bers, Bontá, Resnick, 2013). Takvi objekti mogu biti kocke, kartice, roboti i sl. Ako se konkretni objekti upotrijebe u kontekstu programiranja kao elementi programa onda se dobije jezik koji je *opipljiv*, te imamo *opipljive programske jezike* (Bers, Horn, 2010). Kod opipljivog programiranja naredbe su predstavljene fizičkim predmetima, u obliku kocaka, pločica ili kartica i sl. koje djeca mogu uzeti u ruku i slagati (Slika 1).

Slika 1. Opipljivo programiranje.

Zanimljivo je da zapravo najraniji primjer opipljivog programiranja kao i LOGO potječe iz 1970-tih, a radilo se o elektromehaničkom uređaju pod nazivom TORTIS (Toddler's Own Recursive Turtle Interpreter System) kojeg je oblikovala Radia Perlman (Perlman, 1974).

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, što znači da nije imao vezu s računalom, a jedan od prvih opipljivih programskih jezika povezan s računalom je AlgoBlocks iz 1990-tih (Suzuki, Kato, 1993). Istraživanje koje su proveli Sapounidis i Demetriadis (Sapounidis, Demetriadis, 2013) na tri dobne skupine djece (5-6, 7-8 i 11-12 godina) pokazalo je da je opipljivo sučelje privlačno svim dobnim skupinama djece koje su istraživali, pogotovo kao novo iskustvo, ali su starija djeca ipak radije birala grafičko sučelje na računalu radi brzine rada.

3. Skrivanje detalja i posredovani prijenos

Primjena metode didaktičkog skrivanja (Futschek, 2013) odnosno skrivanja složenosti detalja pomaže pri smanjivanju kognitivnog opterećenja učenika (Sweller, Ayres, Kalyuga, 2011). Na primjer, zaslon ili prozor na kojem se prikazuje lik podijeljen je na točkice, a zadatak je pomaknuti lika koji se nalazi na koordinatama (x, y) po osi x, tj. desno za 5 točkica. Rješenje tog problema je jednostavno: (x+10, y). Kako ćemo dobiti nove koordinate lika koji mora prijeći 5 točkica po hipotenuzi Pitagorinog trokuta? U ovom slučaju, obzirom da nije zadan kut možemo pokušati pogoditi jedno rješenje zbog karakterističnog trokuta: (x+4, y+3), ali to se inače za poznati kut i duljinu hipotenuze može izračunati uz pomoć trigonometrijskih funkcija. Navedeni primjer može poslužiti kao povezivanje koncepata iz matematike s programiranjem, ali je loš za djecu nižih uzrasta koji jednostavno žele pomaknuti lika za 5 točkica u koso bez razmišljanja kako je tamo došao.

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, zatim poveže potrebne informacije u memoriji te shvati da može primijeniti koncepte iz poznatog konteksta (Perkins, Salomon, 1992). Ponekad nastavnik treba pomoći prilikom prijenosa pa se radi o *posredovanom prijenosu* (eng. Mediated transfer).

Vizualni programski jezici mogu pomoći i odraslim učenicima, odnosno studentima, ali oni takve jezike brzo "prerastu" (Krpan, Mladenović, Zaharija, 2014) 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, bilo je potrebno posredovati, odnosno pomoći studentima u prijenosu koncepata uz pomoć posebno pripremljenog okruženja u kojem mogu izraditi jednostavne igre, što se detaljnije može vidjeti u (Krpan, Mladenović, Zaharija, 2017). U tom radu opisan je prototip za posredovani prijenos iz vizualnog jezika kojeg je koristila mala skupina studenata bez prethodnog iskustva programiranja u programskom jeziku C#. Studenti su uspjeli izraditi igru u vizualnom okruženju te prenijeti i izvršiti generirani C# kod u novo okruženje Microsoft Visual Studia.

Na temelju ideja vizualnog programskog jezika tipa Scratch i posredovanog prijenosa iz prethodnog istraživanja (Krpan, Mladenović, Zaharija, 2017) razvijeno je 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 kôd u obliku teksta te prikazuje rezultat izvršavanja tog kôda. Na Slici 2 vidljive su sve tri funkcionalnosti na jednostavnom primjeru naslova i odlomka teksta. Dodatno, učenici mogu spremiti datoteke s kôdom (HTML, JavaScript i CSS) te ih pokrenuti u web pregledniku neovisno o samom okruženju i nastaviti raditi u bilo kojem uređivaču kôda.

3.1. Vrednovanje zadaća u vizualnim programskim jezicima

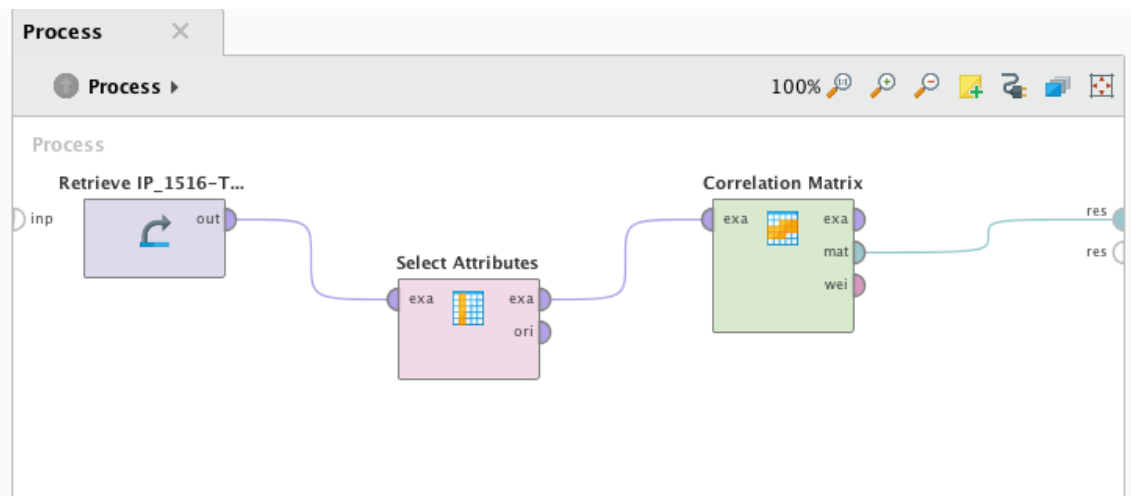
Upotreba Scratch-a i sličnih programskih jezika u nastavi znači da treba vrednovati i ocijeniti zadatke koji se rješavaju u takvim okruženjima. Vrednovanje je jedna od najčešćih zadataka koje nastavnici moraju obaviti, a cilj je da nastavnici utvrde što su učenici usvojili te da učenici dobiju povratnu informaciju o svom znanju (Hazzan, Lapidot, Ragonis, 2015). 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 koje treba prijeći onda nastavnik često treba "odigrati" te razine radi provjere funkcionalnosti.

Studenti su tijekom rada u Scratch-u (Krpan, Mladenović, Zaharija, 2014) imali više zadaća koje je trebalo pregledati i ocijeniti: zadaci na vježbama, zadaci za samostalan rad kod kuće, završni projekt i međuispit. Ukupan broj zadataka po studentu je na kraju bio oko 15-20 ovisno o akademskoj godini i prisutnosti, no primjerice 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 blokova, 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* (Boe, Hill, Len, Dreschler, Conrad, Franklin, 2013). Na temelju

biblioteke razvijen je online alat *Dr. Scratch* kojeg mogu koristiti učenici i nastavnici za provjeru projekata (Moreno-León, Robles, 2015). Prednost automatskog ocjenjivanja može se iskoristiti kod ocjenjivanja domaćih zadaća (Johnson, 2016) gdje npr. učenici mogu sami provjeriti svoju zadaću prije nego je predaju ili nastavnik može brže ocijeniti veliku količinu zadaća, a također se može koristiti za ispitivanje trenda upotrebe nekog programskog koncepta (Aivaloglou, Hermans, 2016). Ovdje je važno napomenuti da se na ovaj način može vidjeti da učenici koriste neke koncepte ili blokove (npr. petlje, odluke i sl.), ali to ne mora značiti da su to zaista usvojili.

Analizirali smo složenost ukupno 55 rješenja zadaća za studente iz ak. godine 2015/16 i 2016/17 pomoću Hairball biblioteke koja ima implementirane funkcije za računanje ciklomatske složenosti (eng. cyclomatic complexity, CC) i Halstead-ove mjere (eng. Halstead's metrics, HM) (Moreno-Leon, Robles, Roman-Gonzalez, 2016). To su standardne mjere složenosti programske podrške. CC ovisi o broju odluka u programu te predstavlja broj neovisnih putanja u programu. To bi lakše mogli predočiti kad bi nacrtali dijagram toka pa prebrojali kojim sve putem možemo doći od početka do kraja programa. S druge strane, HM se računa na temelju broja operatora i operanda te se za razliku od CC fokusira na tok podataka. Potrebno je napomenuti da projekti koje su studenti radili u Scratchu zbog same prirode jezika nisu usporedivi s projektima slične funkcionalnosti iz profesionalnih programskih jezika tako da se ovdje ne ulazi u detalje implementacije spomenutih mjera već se koristi gotova biblioteka i provjereni način ocjenjivanja opisan kod Dr. Scratch-a. Cilj je usporediti ocjenu nastavnika s automatskom ocjenom kako bi se utvrdila sličnost ili povezanost tih ocjena. Pojednostavljeno opisano, pitamo se da li veća ocjena nastavnika odgovara većoj automatskoj ocjeni složenosti?

Korelacija predstavlja međusobnu povezanost između različitih pojava koje promatramo, a koeficijent korelacije je mjera povezanosti. Korelacija se može računati na različite načine (Cohen, Manion, Morrison, 2013) upotrebom statističkih alata ili bez, a za analizu podataka u ovom radu odabrali smo program RapidMiner (https://rapidminer.com/). Razlozi su što je besplatan te omogućuje vizualni prikaz postupka ili modela analize (slika 3).



Slika 3. Model u RapidMiner-u.

Nakon pokretanja modela sa slike 3 dobije se matrica korelacije u obliku tablice. RapidMiner različitim nijansama boje prikazuje jačinu korelacije. Na slici 4 prikazana je matrica korelacija između ručnog ocjenjivanja (*manualGrade*) i ostalih varijabli koje računa

Hairball: Dr. Scratch ocjena (Moreno-León, Robles, 2015), CC, te *timeSec* (HM veličina koja označava potrebno vrijeme za pisanje kôda).

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 4. Korelacija automatskog ocjenjivanja i ocjene nastavnika.

Za ak. god. 2015/16 je postojala korelacija između složenosti (*timeSec*) i ocjene nastavnika (stupac označen s *ManualGrade*) od 0.597, a za ak. god. 2016/17 od 0.522. Dobivene korelacije su male što znači da postoji odnos među varijablama, ali nije značajan.

Zadaci s međuispita sadržavali su precizno definirane zahtjeve, primjerice likovi su imali određene početne pozicije, brzinu kretanja, način interakcije (s drugim likovima, rubom pozornice i korisnikom), uvjete za završetak igre i sl. Nastavnik ručno ocjenjuje ispunjenost postavljenih zahtjeva, odnosno da li likovi u igri izvršavaju ono što je zadano. Dva programa koji dobiju istu ocjenu prema ispunjenosti zahtjeva ne moraju se poklapati i po razini složenosti jer se upotrebom različitog broja odluka i petlji, mijenja složenost iako je funkcionalnost na prvi pogled ista. Više nije uvijek bolje jer su primjerice neki studenti koristili dvije ugniježđene beskonačne petlje ili odluke koje se nikad neće izvršiti ako se likovi zbog pogreške nikad neće dotaknuti i sl. Skripte (dijelovi kôda u Scratchu) se često izvršavaju paralelno. Dio zadatka je bio da korisnik pokreće lika koji hvata loptu. Za svaku uhvaćenu loptu, povećavaju se bodovi. Ako student napravi program tako da i lik i lopta broje bodove na dodir, dogodit će se višestruko povećavanje što nije ispravno rješenje. Nadalje, kod nekih rješenja nije uopće bilo moguće izvršiti ili odigrati igru do kraja jer su se likovi kretali prebrzo ili u pogrešnim smjerovima ili bi nestali s pozornice tako da se nikad ne bi dotakli. Takve stvari općenita biblioteka koju smo koristili za analizu ne može detektirati.

4. 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 opipljivog programiranja već samo dok ne usvoje osnovne koncepte i osnove rada s računalom kako bi se u idućem koraku moglo prijeći na vizualne programske jezike. Odabir vizualnog programskog jezika može pozitivno utjecati na daljnji rad u tekstualnim ili profesionalnim programskim jezicima ukoliko postoji povezanost ili prijenos znanja između tih jezika. U protivnom, učenici ta iskustva promatraju odvojeno.

U ovom radu smo pored osvrta na vrste jezika ispitali automatsko ocjenjivanje projekata izrađenih u Scratchu. Scratch omogućuje stvaranje interaktivnih priča (eng. storytelling) uz upotrebu slika i zvukova, ali uloženi trud u multimedijske sadržaje nije moguće ocijeniti automatski. Također, Scratch omogućuje interakcije likova s korisnikom i ostalim likovima (izrada igara), a pogreške kod kojih je ta

interakcija otežana ili nemoguća čine igru zapravo neupotrebljivom. Ako igranjem igre ne možemo nikako prijeći na iduću razinu onda je ne možemo ni ocijeniti, dok će automatsko ocjenjivanje dati ocjenu. Prema tome, nije se moguće u potpunosti osloniti na automatsko ocjenjivanje takvih projekata i isključiti nastavnika, ali smatramo da se može koristiti kao pomoć ili dodatan kriterij pri vrednovanju zadaća kod slobodnih projekata.

Literatura

Prensky, M. (2001). Digital natives, digital immigrants part 1. *On the horizon*, 9(5), 1-6.

Hernandez, C. C., et al. (2010). Teaching Programming Principles through a Game Engine. *CLEI ELECTRONIC JOURNAL*, 13(3).

Kelleher, C., Pausch, R. (2003). *Lowering the Barriers to Programming: A Survey of Programming Environments and Languages for Novice Programmers*. Defense Technical Information Center, Fort Belvoir, VA.

Ala-Mutka, K. (2004). *Problems in learning and teaching programming-a literature study for developing visualizations in the Codewitz-Minerva project*. Codewitz Needs Analysis, 1-13.

Böhm, C., Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two formation rules. *Communications of the ACM*, 9(5), 366-371.

Robins, A., Rountree, J., Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, 13(2), 137-172.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., Miller, P. (1997). Mini - languages: A Way to Learn Programming Principles. *Education and Information Technologies*, 2(1), 65-83.

Papert, S. (1980). Mindstorms: children, computers, and powerful ideas. New York: Basic Books.

Hazzan, O., Lapidot, T., Ragonis, N. (2015). *Guide to teaching computer science: An activity-based approach*. Springer.

Resnick, M., et al. (2009). Scratch: Programming for All. *Communications of the ACM*, 52(11), 60-67.

Wyeth, P. (2008). How young children learn to program with sensor, action, and logic blocks. *Journal of the Learning Sciences*, 17(April), 517-550.

Piaget, J., Inhelder, B. (2013). *The growth of logical thinking from childhood to adolescence: An essay on the construction of formal operational structures*, vol. 84. Routledge.

Richardson, K. (1998). *Models of Cognitive Development*. Psychology Press.

Flannery, L. P., Silverman, B., Kazakoff, E. R., Bers, M. U., Bontá, P., Resnick, M. (2013). Designing ScratchJr: Support for early childhood learning through computer programming. *Proceedings of the 12th International Conference on Interaction Design and Children*, 1-10.

Bers, M. U., Horn, M. S. (2010). Tangible Programming in Early Childhood: Revisiting Developmental Assumptions Through New Technologies: Childhood in a Digital World. In Berson, M. J., Berson, I. R. (Eds.), *High-Tech Tots: Childhood in a Digital World*. Information Age Publishing. 1-32.

Perlman, R. (1974). *TORTIS: Toddler's Own Recursive Turgle Interpreter System*, Cambridge.

Suzuki, H., Kato, H. (1993). AlgoBlock: a tangible programming language, a tool for collaborative learning. *Proceedings of 4th European Logo Conference*, no. June 2016, 297-303.

Sapounidis, T., Demetriadis, S. (2013). Tangible versus graphical user interfaces for robot programming: exploring cross-age children's preferences. *Personal and Ubiquitous Computing*, 17(8), 1775-1786.

Futschek, G. (2013). Extreme didactic reduction in computational thinking education. *10th World Conference on Computers in Education*, 1-6.

Sweller, J., Ayres, P., Kalyuga, S. (2011). *Cognitive Load Theory*, 1st ed., vol. 10. Springer-Verlag New York.

Perkins, D. N., Salomon, G. (1992). *Transfer of learning, International Encyclopedia of Education*, Second Edition, vol. 2, 1-11.

Krpan, D., Mladenović, S., Zaharija, G. (2014). Vizualni programski jezici u visokom obrazovanju. *16. CARNetova korisnička konferencija - CUC 2014 - Zbornik radova*, Zagreb.

Krpan, D., Mladenović, S., Zaharija, G. (2017). Mediated Transfer from Visual to High-level Programming Language. 40. međunarodni skup za informacijsku i komunikacijsku tehnologiju, elektroniku i mikroelektroniku - MIPRO.

Boe, B., Hill, C., Len, M., Dreschler, G., Conrad, P., Franklin, D. (2013). Hairball: Lint-inspired Static Analysis of Scratch Projects. *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*, 215-220.

Moreno-León, J., Robles, G. (2015). Dr. Scratch: a web tool to automatically evaluate Scratch projects. *Proceedings of the Workshop in Primary and Secondary Computing Education on ZZZ - WiPSCE '15*.

Johnson, D. E. (2016). ITCH : Individual Testing of Computer Homework for Scratch Assignments. *Proceedings of the 47th ACM Technical Symposium on Computer Science Education (SIGCSE '16)*, 223-227.

Aivaloglou, E., Hermans, F. (2016). How Kids Code and How We Know. *Proceedings of the 2016 ACM Conference on International Computing Education Research - ICER '16*, 53-61.

Moreno-Leon, J., Robles, G., Roman-Gonzalez, M. (2016). Comparing computational thinking development assessment scores with software complexity metrics. *IEEE Global Engineering Education Conference, EDUCON*, vol. 10-13-Apri, no. April, 1040-1045.

Cohen, L., Manion, L., Morrison, K. (2013). *Research Methods in Education*, 7th ed. Routledge.

Mediated Transfer in Teaching Programming Using Visual Programming Languages

Abstract

Novice programmers find learning programming difficult and challenging. That perception has a negative effect on motivation, and problems which students face during their first learning attempts only confirm their expectations so they give up. Using simpler syntax and environment is a possible solution to make the first programming experience easier or to lower the threshold towards complex programming languages. It is very important to choose the programming language and the developing environment which are suitable for student's age. In this paper, we describe few examples of didactic tools used for teaching programming to students of different age and also visual programming projects evaluation.

Keywords: *visual programming; didactic reduction; tangible programming; mediated transfer.*