Poučavanje programera početnika u visokom obrazovanju

Divna Krpan
divna.krpan@pmfst.hr

***Sažetak*—Programeri početnici učenje programiranja poistovjećuju s učenjem sintakse programskog jezika, te ne uspijevaju svladati osnovne algoritamske koncepte i vještine rješavanja problema. Poteškoće rezultiraju lošom prolaznosti na predmetima početnog programiranja na fakultetima, te brzim gubitkom motivacije. Odabir početnog jezika neopterećenog sintaksom omogućuje jednostavnije usvajanje osnovnih koncepata programiranja i rješavanje složenijih problema. Obzirom da je u ovom radu fokus na odraslim učenicima, odnosno studentima, cilj je omogućiti prijelaz iz jednostavnog okruženja u profesionalno okruženje za razvoj programske podrške pružajući pri tome primjer dobre prakse budućim nastavnicima informatike kako poučavati učenike programiranju, ali i ostalim nastavnicima kako uključiti programiranje u integrirani STEM pristup.**

Ključne riječi: programeri početnici, poučavanje programiranja, vizualno programiranje, posredovani prijenos

I. Uvod

Učenje i poučavanje programiranja je općenito teško te predstavlja izazov za nastavnike i učenike, bez obzira na uzrast učenika (osnovna škola, srednja škola ili fakultet) [1]. Studenti na fakultetima često imaju poteškoće pri polaganju ispita iz predmeta vezanih za programiranje posebno na predmetima početnog programiranja.

Programiranje se kao značajna stavka pojavljuje u sklopu *informacijske ili računalne pismenosti* [2], a smatra se da je računalna pismenost važna posebno za preddiplomske studije i cjeloživotno učenje [3].

Prisutan je trend grupiranja četiri područja pod akronimom *STEM* (Science, Technology, Engineering, and Mathematics) [4], a obuhvaća znanost, tehnologiju, inženjerstvo i matematiku. STEM je akronim koji stječe popularnost i u Hrvatskom jeziku. Ako se bar dvije ili više navedenih područja na neki način integrira u nastavi, onda se radi o *integriranom STEM-u* [4], [5]. Jednostavan primjer je upotreba računalnih simulacija za učenje pojmova iz fizike, kemije, biologije i sl. Djeca su danas izložena tehnologiji veoma rano [6], [7], [8], a također se pokazuje kako aktivnosti u poučavanju kojima su djeca izložena u ranom djetinjstvu imaju trajniji učinak [9]. U skladu s tim, primjerice dolazi i do inicijative upotrebe robota za poučavanje programiranja [10], [11] kao kombinacije tehnologije i inženjerstva. Djeca na taj način pored koncepata vezanih za programiranje usvajaju i druge kao što su npr. motori, senzori i sl. [12]. Sličan primjer je i učenje programiranja uz Logo programski jezik gdje djeca usvajaju razne matematičke koncepte i rješavanje problema [13]. Papert je kod razvoja Logo programskog jezika shvatio kako djeca uče bolje, odnosno bolje shvaćaju kako programirati kornjaču ako se "zamisle na njenom mjestu" tj. kad crtaju kvadrat i sami pokušaju prošetati u obliku kvadrata [14]. U tim počecima je bila prisutna robotska kornjača, ali se ipak nije uspjela zadržati u učionici pored Logo jezika zbog prevelike cijene i nepouzdanosti. U današnje vrijeme ideju Papert-ove robotske kornjače zamjenjuju Lego roboti.

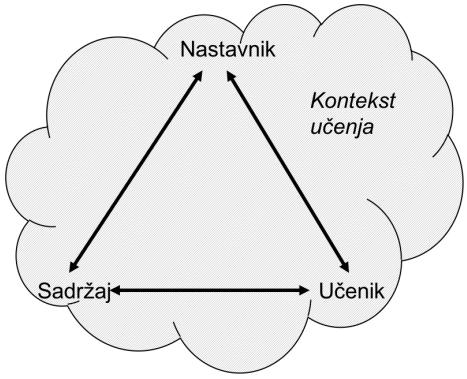
Uvođenje takvog integriranog pristupa pokazuje potencijal, no problem je što nema mnogo nastavnika koji su spremni raditi na taj način ili onih koji imaju potrebne vještine [8], te je potrebno ispitati postojeća iskustva iz nastave [15], [11]. Odmak od uobičajenog načina poučavanja zahtjeva dodatni rad i pripremu za što nastavnici nisu uvijek dovoljno motivirani [16] jer je primjerice već spomenuti primjer upotrebe robota za poučavanje programiranja veliki izazov bez obzira na istraživanja koja pokazuju pozitivan stav nastavnika [17], [18]. Tijekom obrazovanja budućih nastavnika imamo priliku utjecati na njihova znanja, vještine i stavove. Međutim, studenti koji se obrazuju za buduće nastavnike imaju probleme kod početnog programiranja kao i svi ostali studenti [19]. Smatra se da su neki od ključnih problema koje učenici imaju pri učenju programiranja upravo slabo razvijena sposobnost rješavanja problema [20], apstraktna priroda samog jezika, te opterećenje složenom sintaksom. U sljedećem dijelu opisat ćemo sve faktore uključene u proces učenja i poučavanja programiranja.

II. Didaktika i poučavanje programiranja

Postoji izreka da je "didaktika stara kao vrijeme" [21]. Objašnjenje se nalazi u činjenici da od razvoja društva postoji potreba za prijenosom znanja, vještina i iskustava na sljedeću generaciju. Međutim, tijekom vremena su se znanja, vještine i iskustva akumulirali te je postalo sve teže prenositi ih na iduće generacije pa su nastale škole [22]. S pojavom škola, definira se i uloga *poučavatelja*, odnosno *učitelja* ili *nastavnika*. Trojka: *učenik - nastavnik - sadržaj* je prisutna od početka prijenosa znanja, vještina i iskustava, ali se pojam *trokut* pojavljuje 1980-tih godina. Obzirom da se učenje i poučavanje ipak odvija u određenom kontekstu revidirani model uključuje i kontekst (Sl. 1).

Postoji i model *tetraedra* u kojem je tehnologija još jedna dodatna dimenzija [21], no ovdje to nećemo izdvajati iz razloga što je tehnologija prilično integrirana u sadržaj učenja vezan konkretno za programiranje.

Obrazovanje je aktivnost koju inicira jedan ili više aktera s ciljem utjecaja na znanje, vještine i stavove pojedinaca, grupa ili zajednica. Pojam *učenje* odnosi se proces stjecanja



Slika 1. Didaktički trokut

znanja, vještina i stavova. Obrazovanje stavlja naglasak na nastavnika (tehnike i modele poučavanja), a učenje na učenika (karakteristike učenika i strategije učenja) [23]. Opisat ćemo pojedine kutove trokuta u kontekstu programiranja.

Prema [24] *sadržaj* iz didaktičkog trokuta se odnosi na ciljeve učenja, vrednovanje postignuća i stvarni sadržaj koji se uči. *Učitelj/nastavnik* predstavlja proces učenja i poučavanja, te prilagodbu sadržaja (odabira i prikaza) u skladu s tim. *Učenik* predstavlja pretpostavke učenja i proces učenja (konkretno proces promjena). Vanjski uvjeti u kojima se odvija učenje i poučavanje su *kontekst*.

A. Učenik u didaktičkom trokutu

Učenici u visokom obrazovanju su studenti, odnosno odrasle osobe. Djeca i odrasli se razlikuju po karakteristikama, načinu učenja, a to znači da nastavnik u skladu s tim mora prilagoditi i način poučavanja.

Odrasli učenici ne mogu biti pasivni primatelji sadržaja kojeg im nastavnik prenosi. Kod tradicionalne nastave se očekuje da će se učenih prilagoditi kurikulumu, dok bi se kod obrazovanja odraslih kurikulum morao prilagoditi potrebama i interesima odraslih učenika [23]. Svaka odrasla osoba nalazi se u nekom kontekstu koji se odnosi na trenutni život, posao, obitelj, interes i sl. Svaka situacija zahtijeva određene promjene i prilagodbe, te tu na scenu stupa obrazovanje. Odrasla osoba više obraća pažnju na stvari iz stvarnog svijeta i uklanjanje prepreka koje tu nalazi. Smatra se da su odrasli više motivirani za učenje ako postoje neke potrebe ili interesi koje na taj način mogu zadovoljiti. Nastavnik pri tome mora potaknuti odraslog učenika na preispitivanje (eng. inquiriy) umjesto pokušaja prijenosa svog znanja.

Predavački način rada (eng. lecture-based learning, LBL) je najčešće prisutan na fakultetima s odraslim učenicima [25]. Andragogija je znanost koja se bavi proučavanjem učenja odraslih, a nailazi se i na pojem *učenje odraslih* (eng. adult-based learning, ABL). Potrebno je pripremiti okolinu za učenje u kojoj će postojati atmosfera razumijevanja, pomaganja i slobode izražavanja. ABL omogućuje trenutnu primjenu naučenih koncepata i vještina, te također kao i ostali pristupi promovira rješavanje problema i rad u skupinama. Skupine i timovi kod poučavanja odraslih su češće unutar učionice i kraćeg trajanja zbog vremenskih ograničenja. U skladu s tim učitelj mora prilagoditi nastavu.

Andragogijski model (eng. andragogical model) sadrži pretpostavke:

1) *Potreba za znanjem* (eng. need to know) - odrasli moraju znati zašto nešto uče.
2) *Samopoimanje učenika* (eng. learner's self-concept) - poimanje da su sami odgovorni za svoje odluke i akcije (samim tim i za učenje).
3) *Uloga iskustva učenika* - prethodno iskustvo.
4) *Spremnost za učenjem* (eng. readiness to learn) - odrasli su spremni za učenje novog u trenutku kad u stvarnim situacijama dođe do toga.
5) *Orijentacija na učenje* (eng. orientation to learning) - odrasli se ne usmjeravaju prema predmetima i sadržajima kao djeca već prema stvarnim problemima.
6) *Motivacija* - vanjska motivacija (posao, plaća i sl.) može utjecati na odrasle, ali je unutarnja motivacija ključna.

Oblici samostalnog (eng. self-directed), iskustvenog učenja usmjerenog prema rješavanju problema su bitni, te je njihovo razumijevanje od strane nastavnika osnova za povećavanje motivacije kod učenika. Učenici koji su motivirani i koriste odgovarajuće strategije učenja, ostvaruju bolje rezultate, te su uspješniji. Kao posljedica toga, takvi učenici će steći navike koje će im koristiti kod cjeloživotnog učenja. Pomicanje fokusa s učitelja na učenika utječe na nastavu koja se više zasniva na izradi projekata i međusobnoj suradnji učenika.

Motivacija je od iznimne važnosti za učenje programiranja, te posebno unutarnja (intrinzična). U skladu s konstruktivizmom nastala je *teorija uključivanja* (eng. engagement theory) koja nije izravna posljedica nekog teorijskog okvira za učenje već je nastala na temelju iskustva u poučavanju uz pomoć tehnologije, a predstavlja se kao model za učenje [26], [27]. Smatra se da aktivnosti učenika uključuju aktivne kognitivne procese (stvaranje, rješavanje problema, razmišljanje, donošenje odluka i vrednovanje).

Naglasak kod odraslih učenika je na aktivnom učenju i sudjelovanju u nastavi (sudjelovanje u raspravama i učenje u skupinama koje ovdje uključuje i suradnju) [28] gdje nastavnik preuzima ulogu voditelja i pomagača. Poučavanje moderne *Net generacije* (djeca rođena iza 1982 godine) može biti veliki izazov za nastavnika jer su zahtjevni, nestrpljivi, ponekad i nepristojni i s ciničnim stavom prema autoritetu [29]. Međutim, cijene komunikaciju i kad su informirani o tome zašto i kako se nešto uči, te reagiraju drugačije na specifične situacije u odnosu na "tradicionalne" učenike. Primjerice, više će zamjeriti nedostatak iskustva nastavnika i nepripremljenost za nastavu. U početku visokog obrazovanja fakulteti su birali najbolje studente koji su morali biti visoko motivirani kako bi završili studij. Danas je cilj obrazovati što više ljudi čime dolazi i do širokog spektra različitih karakteristika učenika.

Sam proces učenja programiranja zahtjeva od studenata dodatne aktivnosti izvan učionice: čitanje materijala, izrada algoritama, pisanje koda, otkrivanje i uklanjanje grešaka, te traženje pomoći od svojih kolega. Početnicima nedostaje vještina planiranja [30], što se vidi u tome kad počnu pisati program bez prethodno definiranog plana rješenja problema. Pri tome često dolazi do grešaka zbog nepoznavanja jezika, te studenti više nisu u stanju razlikovati iz kojeg razloga njihovo rješenje radi ili ne radi. Trebali bi naučiti i kako procijeniti

svoju uspješnost, te u kojem trenutku i kako zatražiti pomoć od nastavnika ili kolega [31], [32].

Obzirom da se često naglašava kako je učenje programiranja teško uz visoke razine odustajanja, posebno na predmetima početnog programiranja na fakultetima, učenik mora dobiti svu moguću pomoć tako da je nastavnik bitan faktor. Premda se zagovara nastava usmjerena na učenika, ne znači da se uloga nastavnika minimizira već se jednostavno mijenja. Unatoč velikom broju studenata koji odustaje od programerskih predmeta, takvi predmeti u principu i dalje imaju veliki broj studenata.

B. Nastavnik u poučavanju programiranja

U radu [33] opisano je provedeno istraživanje s ciljem ispitivanja kako nastavnici računarstva shvaćaju poučavanje. Pri tome se pokazalo da nastavnici smatraju da se kod nastave usmjerene na učenika razvija razmišljanje učenika i ostvaruju dugoročni ciljevi za razliku od nastave usmjerene na učitelja. Nastava usmjerena na učenika odnosi se na nastavu u kojoj je učenik aktivniji od učitelja ili je aktivan barem u istoj mjeri kao i učitelj, te nikako ne obrnuto [34]. U situacijama kad nastavnik stalno koristi predavački način rada, prenosi činjenice i demonstrira, ostvaruje se najniža razina Bloomove taksonomije (činjenično znanje). Međutim, prema [34]: *"Ima mnogo boljih medija za čuvanje i obradu informacija od dječjih glava!"*, te se predlaže problemska i projektna nastava, što je u skladu s karakteristikama odraslih učenika. Učenici moraju učiti na način da nešto samostalno rade (eng. learning by doing) ili istražuju (eng. learning by discovery). Nedostatak nastave usmjerene na učenika je što takva nastava može zahtijevati prilagođene prostore i opremu kao što učenje programiranja u principu zahtijeva računala, odnosno računalnu učionicu sa svom pripadajućom opremom i troškovima.

Programeri početnici na početku moraju usvojiti veliki broj novih apstraktnih koncepata za najosnovnije programerske vještine [35]. Programiranje zahtijeva kontinuiranu upotrebu vještina rješavanja problema i oblikovanja, a početnici rijetko imaju prethodnog iskustva u tim aktivnostima. Stroga sintaksa (u kojoj se doslovno mora paziti na svaki znak) i semantika otežavaju učenje.

U radu [36] definirana su tri osnovna pristupa poučavanju programiranja:

1) Semiotičke ljestve
2) Taksonomija kognitivnih ciljeva
3) Rješavanje problema

Semiotičke ljestve (eng. semiotic ladder) označavaju pristup s naglaskom na alatima koji se koriste (programski jezik, jezik za modeliranje i sl.). Redoslijed učenja je sljedeći: (i) sintaksa, (II) semantika i (iii) pragmatika. Sintaksa se odnosi na kombiniranje znakova u riječi (naredbe jezika), semantika na njihovo značenje, a pragmatika na primjenu. Svaki sljedeći korak ovisi o prethodnom tako da npr. bez sintakse nije moguće ništa iskazati.

Taksonomija kognitivnih ciljeva (eng. Cognitive objectives taxonomy) podsjeća na Bloom-ovu taksonomiju:

1) *Izvršavanje programa* - učenik pokreće i primjenjuje gotovi program.
2) *Čitanje programa* - učenik ima uvid u kod i prepoznaje i razumije konstrukte.
3) *Izmjena postojećeg programa* - učenik može mijenjati i prilagođavati postojeći program.
4) *Stvaranje programa* - učenik samostalno izrađuje program.

Rješavanje problema (eng. problem solving) je pristup u kojem se smatra da učenici mogu naučiti programirati i proširiti postojeće znanje tijekom rješavanja problema. Ovdje se za razliku od prethodnih pristupa definiraju ulaz i izlaz procesa učenja u obliku znanja i osobnog iskustva koji se stječu. Prema tome, ovo je više model učenja nego strategija poučavanja. Strategija poučavanja bi za ovaj pristup mogao biti korak 3. iz prethodnog pristupa.

Prema razgovoru s nastavnicima koji su sudjelovali u istraživanju za [36], predložen je i pristup učenju programiranja koji se koristi kod razvoja programske podrške: *iterativni razvojni ciklus* (eng. iterative software development cycle).

Prema [37] definiraju se sljedeće kategorije učenja za programiranje:

1) Praćenje
2) Kodiranje
3) Razumijevanje i integracija
4) Rješavanje problema
5) Sudjelovanje

Praćenje (eng. following): učenik prati strukturu nastavne jedinice nastojeći izvršiti pojedine zadatke koji su postavljeni, a koji utječu na ocjenu bez promatranja konteksta ili "šire slike". Učenik se trudi izvršiti ono što se od njega u danom trenutku traži ne povezujući pojedine dijelove u cjelinu što kasnije vodi do problema kod stvaranja složenijih programa.

Kod *kodiranja* (eng. coding) učenik smatra da je najvažnije učenje sintakse, vježba sintaksu i memorira naredbe i dijelove koda kako bi što više naučio, te metodom "pokušaja i pogreške" nastoji uklopiti poznate dijelove koda u novi zadatak. Takvo učenje je površinsko, a posljedica je frustracija učenika kod nemogućnosti rješavanja složenijih problema jer učenik ne može razumjeti zašto dio koda koji je bio ispravan u jednom programu ne radi u drugom kontekstu.

Razumijevanje i integracija odnosi se na gledanje šireg konteksta, učenici žele shvatiti, motivirani su i smatraju da se programi sastoje od sintakse, napisanog koda, ali i općih koncepata nevezanih za jezik. *Rješavanje problema* autori ovdje definiraju kao kategoriju koja ovisi o trećoj kategoriji, ali u fokusu je planiranje. *Sudjelovanje* (eng. Participating) se odnosi na učenike (možda je bolje reći studente) koji se žele baviti programiranjem i visoko su motivirani. Oni se na primjer brinu i o čitljivosti programa, žele ne samo da radi već mora raditi brže, izgledati bolje, traže optimalno rješenje.

Programiranje je ključna vještina u računarstvu, a poučavanje programiranja teško [3]. Teško je identificirati koje su to kognitivne poteškoće koje stvaraju probleme kod učenja, ali i vještine koje čine dobrog programera. Čini se kao da se za poučavanje programiranja ulaže više vremena nego kod ostalih područja, a učenicima je unatoč tome i dalje teško. Činjenica je i da učenje programiranja zahtijeva puno vremena i puno vježbanja [3]. Takvo što se ne uklapa u predavački način rada na fakultetu koji traje jedan semestar.

Za poučavanje programiranja početnika se koriste različite strategije [38]. Neki autori predlažu skrivanje detalja dok učenici ne budu spremni za njih, drugi predlažu i izrađuju računalne sustave, a između raznih iskustava ističe se *učenje rješavanjem problema* (eng. problem-based learning, PBL).

1) Skrivanje detalja: Skrivanje detalja ili *didaktička redukcija* (eng. didactic reduction) se smatra korisnom za učenje složenijih koncepata iz programiranja kao što su to primjerice algoritmi i strukture podataka [39] gdje se učenike potiče na razmišljanje umjesto na memoriranje postojećih algoritama.

2) Učenje rješavanjem problema: Problem je subjektivni doživljaj (spoznajni i emocionalni), odnosno čovjek doživljava problem [22]. Što je problem veći, to su intenzivniji spoznajni i emocionalni doživljaji. Problem izražava određene suprotnosti: poznato - nepoznato, otkriveno - neotkriveno itd. koje čovjek doživljava kao nedostatak kojeg mora nadoknaditi ili prevladati upravo rješavanjem problema. U nastavi se često rješavaju znanstveni problemi (matematički, fizikalni i sl.). Probleme u kontekstu ovog rada moraju rješavati učenici.

Učenje rješavanjem problema (Problem-Based Learning, PBL) uključuje konstruktivističke ideje [40]. PBL započinje 1960tih godina kod poučavanja studenata medicine. Osnovna ideja je da učenik uči rješavanjem problema iz stvarnog svijeta. Dodatna svojstva su: učenje u kontekstu, pojašnjenje znanja kroz društvene interakcije (tijekom rada u skupinama), razvoj sposobnosti kritičkog načina razmišljanja [41]. Učenik mora biti aktivan sudionik procesa učenja i poučavanja. Prvi korak je poticanje učenika na rješavanje problema, odnosno potrebno je kod učenika razviti psihološku potrebu za samostalnim rješavanjem problema [22].

Pristup je usmjeren na učenika, obično se odvija u malim skupinama gdje je učitelj *voditelj* (eng. facilitator) i organizator [42], [35]. Implementacija kurikuluma koji se temelji na PBL zahtijeva veliku angažiranost učitelja [41]. Premda mnogi tvrde da koriste PBL u nastavi, mora biti jasno da se to ne odnosi na male zadatke s nastave koje bi učenici i u tradicionalnoj nastavi rješavali već na problem ili projekt iz stvarnog svijete koji je bitno širi [43]. Postoji potencijalna opasnost kod rada u skupini da jedan sudionik napravi sve ili većinu posla.

Tipovi projekata na kojima se radi razlikuju se po razini kontrole odnosno planiranja nastavnika [42]. Prvi tip projekta s točno definiranim zadaćama (eng. task project) u kojem je svaki korak zadan, te učenik nema nikakvu slobodu. Drugi tip ili razina također ima dobro definirana pravila od strane nastavnika, ali učenik može odrediti na koji način će izvršiti pojedine korake. Rezultat takvog rada su projekti koji su slični i usporedivi. Treći tip projekta koji se smatra *pravim projektom*, a naziva se *problemski projekt* nema precizno zadane pojedine korake niti pravila tako da na kraju svaka skupina ili učenik (ako rade pojedinačno) može napraviti potpuno različit projekt.

Problem ili projekt? Akronim PBL se osim za *učenje rješavanjem problema* (eng. problem based learning) koristi i za *učenje izradom projekta* (eng. project based learning), odnosno *projektnu nastavu*. Prema [44] termin *projektno učenje* (eng. project learning) predstavlja širi pojam od *problemskog*, te je problem nešto što nastavnik preciznije definira, odnosno problem je podskup projekta. Međutim, razlika se često izgubi pa se može objasniti citatom: [44]: *"Dva PBL-a su zapravo dvije strane istog novčića."*.

C. Sadržaj u početnom programiranju

Sadržaj kod učenja i poučavanja početnog programiranja odnosi se na usvajanje osnovnih koncepata kao što su npr. varijable, logički uvjeti, odluke i sl., te kasniju primjenu tih koncepata pri izradi kompletnih programa u odgovarajućem programskom jeziku. Ponekad se *sadržaj* poistovjećuje s *programskim jezikom* koji se koristi za poučavanje. Na primjer, ako pitamo studenta što uči na nekom predmetu iz programiranja ili nastavnika što poučava, vjerojatno će odgovor s obje strane biti naziv programskog jezika. Programski jezik ne bi trebao biti u fokusu odnosno učenje programiranja ne bi trebalo biti vođeno tehničkim karakteristikama ili specifičnostima jezika [24]. Naravno da učenici moraju naučiti sintaksu programskog jezika da bi vidjeli kako njihov program radi no osnovna ideja je da učenici moraju biti u stanju prenijeti naučene koncepte u drugi programski jezik [45], [19].

Odabir programskog jezika za poučavanje početnika je složen problem, posebno ako uzmemo u obzir posljedice koje taj odabir može imati na budući uspjeh učenika. Previše složen jezik će brzo demotivirati učenike. Početni programski jezik za početnike bi morao imati jednostavnu sintaksu, brzu povratnu informaciju i strukturirani način pisanja kako bi programeri početnici stekli navike urednog pisanja programa [46].

Programski jezici koji se traže u stvarnom svijetu prilikom zaposlenja su moćni, no samim tim i prilično složeni. S druge strane, programski jezici pisani za početnike, odnosno za učenje i poučavanje programiranja kao što su Pascal, Logo, Basic i sl. obično nemaju primjenu u stvarnom svijetu. Python je jezik za koji se smatra da je "nešto između", odnosno dovoljno moćan za profesionalne programere, ali i dalje dovoljno jednostavan za početnike.

Iako ponekad nastavnici smatraju kako neke osobe jednostavno ne mogu naučiti programirati, neki autori tvrde da bi većina studenata na početnim predmetima iz programiranja mogla usvojiti osnovna znanja i vještine programiranja premda ne moraju postati stručnjaci [47] no to bi bilo u kontekstu računalne pismenosti. Studenti na predmetima iz početnog programiranja općenito ne ostvare veliki napredak [48], [49], a mogući uzrok je i apstraktna priroda sadržaja kojeg moraju usvojiti [50].

Smatra se da složena sintaksa previše opterećuje početnike pa su stoga razvijeni tzv. *mini jezici* (eng. mini languages) [51], odnosno jezici koji su pojednostavljeni i razvijeni upravo zbog poučavanja. Takvi jezici su ograničeni, ali je cilj omogućavanje prijelaza na složenije ili općenite programske jezike. Programski jezik Logo kojeg je razvio Papert smatra se jednim od prvih takvih mini jezika, a nakon njega nastaju mnogi drugi sa sličnom idejom vizualizacije, te ćemo ih češće naći pod nazivom: *vizualni programski jezici* (eng. visual programming languages).

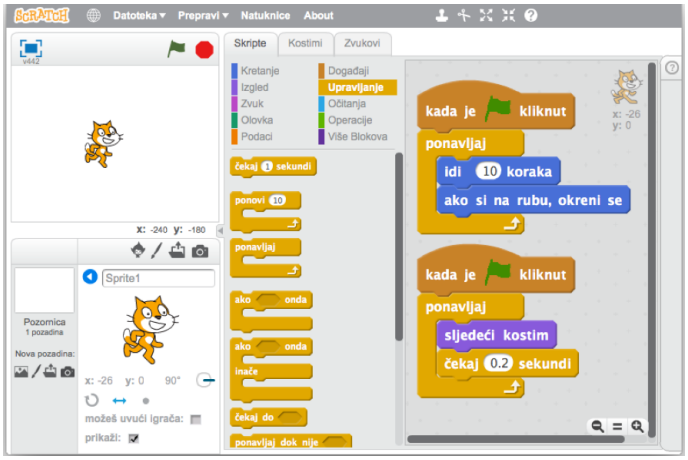
1) Vizualni programski jezici: Vizualizacije se općenito uvod kao pomoć razumijevanju asptraktnih sadržaja [31] počevši od vizualnog prikaza izvršavanja programa do u

potpunosti vizualnih programskih jezika. Početnici ističu kako im je obično sintaksa jezika zbunjujuća i zahtjevna [52]. Učenje sintakse programskog jezika se u početku čini sličnim učenju stranog jezika, ali je teže jer je programski jezik apstraktan, te uz to učenici moraju dodatno svladati okolinu u kojoj programiraju, razumjeti problem i tehnike rješavanja problema, oblikovati konačno rješenje (program) i testirati ga [39].

Vizualni programski jezici potječu još iz 1960-tih [53]. Globalno ih možemo podijeliti na:

1) čiste vizualne programske jezike,
2) hibridne programske jezike.

Kod čistih vizualnih programskih jezika programer stvara ili slaže program od grafičkih elemenata koji predstavljaju elemente programskog jezika (naredbe, varijable i sl.), te se program također izvršava u istoj vizualnoj okolini. Ne postoji prijevod ili prijelaz u tekstualni oblik. Hibridni jezici imaju kombinaciju s tekstualnim oblikom. Pokazuje se u praksi da bolji studenti brže "prerastu" vizualni oblik naredbi, te žele prijeći u tekstualni. Scratch (https://scratch.mit.edu/) (Sl. 2) je primjer čistog vizualnog programskog jezika koji omogućuje usvajanje različite razine koncepata iz programiranja nastalog na idejama programskog jezika Logo.



Slika 2. Scratch 2.0

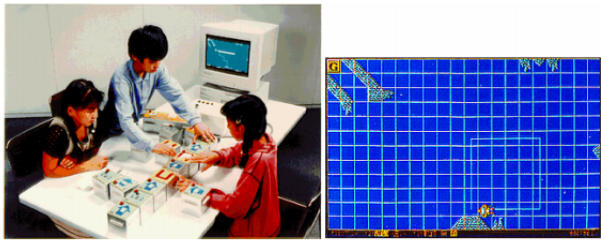
Nedostatak prvih verzija Scratch-a je bio ograničen skup naredbi koje se su se mogle koristiti, te bi napredniji učenici često bili frustrirani nemogućnošću da izraze svoju ideju. Iz tog razloga su razvijani *dijalekti* Scratch-a, kao što je BYOB (akronim od: Build Your Own Blocks) iz kojeg kasnije nastaje ***Snap!***. Osnovna prednost BYOB-a je bila izrada vlastitih blokova čime okruženje više nije bilo ograničeno, te izrada klonova tijekom izvođenja programa, a Scratch je u međuvremenu s verzijom 2.0 također dobio izradu vlastitih blokova kao i klonove. Klonovi su tipičan primjer stvaranja objekata tijekom izvođenja programa čime se olakšava usvajanje osnovnih koncepata iz objektno-orijentiranog programiranja kao što su objekti i klase.

Osim spomenutog Scratch-a, često se koriste i drugi jezici kao što su [54]: Alice (http://www.alice.org), (http://www.greenfoot.org), Kodu (http://fuse.microsoft.com), EToys (http://www.squeakland.org), App Inventor (http://appinventor.mit.edu) i sl. Alice i Greenfoot.

2) Opipljivo programiranje: Prema [55] djeca u dobi od 7-11 godina razvijaju sposobnosti rješavanja problema i logičkog razmišljanja, no postavljanjem problema u odgovarajući kontekst moguće je da djeca u mnogo ranijoj dobi dostignu te sposobnosti [56]. Međutim djeca u predškolskoj dobi još nisu upoznata sa slovima, te čak uz jednostavnu "sintaksu" jezika tipa Scratch u obliku slagalica koje se moraju složiti po obliku ipak moraju čitati što na njima piše. Iz tog razloga se uvode vizualni jezici za niže uzraste sa simbolima kao što je ScratchJr [57] (Sl. 3), te programiranje bez računala (eng. unplugged) odnosno *opipljivo programiranje* (eng. tangible programming) tj. fizički objekti sa simbolima tzv. "pomagala" (eng. manipulatives), a programski jezici s takvim pomagalima nazivaju se *opipljivi programski jezici* (eng. tangible programming languages). Djeca u dobi od 5-8 godina razvijaju veze ili asocijacije između konkretnih i simboličkih prilaza upotrebom "pomagala" (eng. manipulatives) [58]. To mogu biti blokovi, kartice, roboti i sl. [11].

Slika 3. ScratchJr

Jedan od najranijih primjera je *AlgoBlock* s aluminijskim blokovima [59], [60]. Svaki fizički blok je predstavljao naredbu programskog jezika čiji se rezultat prikazivao na zaslonu računala (4).



Slika 4. Rad s AlgoBlock blokovima (lijevo) i prikaz na zaslonu (desno) [60]

Sintaksa kod blokova i kartica se zapravo svodi na to da li se *pomagala* mogu spojiti zajedno, te ovise o obliku, materijalu, boji i sl. [12], [10], [61].

Osim programskog jezika, potrebno je odabrati i okruženje, te paradigmu. Za poučavanje početnog programiranja koriste

se različite paradigme, a najčešće su proceduralna i objektno-orijentirana paradigma [62]. Bez obzira što od ove tri stvari se odabere prije, ostale dvije će ovisiti o tom izboru.

3) Programske paradigme i razvojna okruženja: Programski jezici koji pojednostavljuju sintaksu omogućuju lakše usvajanje temeljnih koncepata programiranja kao što su strukture kontrole tijeka izvođenja programa (eng. control structures): *slijed, grananje i ponavljanje* [63], a s te tri strukture se mogu izraziti ostali algoritmi [64].

Tradicionalni pristup, odnosno pristup kojeg koristi većina je poučavanje proceduralne paradigme u početnim predmetima iz programiranja, tj. poučavanje spomenuta tri osnovna koncepta [62]. Prema toj ideji se objekti uvode tek kasnije.

Studenti koji se obrazuju za buduće nastavnike informatike, a istovremeno su i početnici, odnosno kao odrasle osobe se prvi put susreću s programiranjem, moraju usvojiti ne samo osnovne već i napredne koncepte u relativno kratkom vremenu od jednog semestra po predmetu, te ih naučiti prenijeti na druge. Vrijeme koje studenti imaju na raspolaganju je relativno kratko jer prema [65], [30] potrebno je bar deset godina da bi netko od programera početnika postao stručnjak. Kod obrazovanja nastavnika zapravo nije cilj od njih stvoriti inženjere već ih naučiti konceptima programiranja, kao i primjerima dobre prakse koje će moći primijeniti u svom budućem radu.

Razvojno okruženje ili *okolina* (eng. environment) predstavlja programsku podršku u kojoj se može pisati programski kod tj. naredbe podržanog programskog jezika. Odabir programskog jezika npr. Scratch-a uvjetuje i posljedični odabir programskog razvojnog okruženja (npr. može se birati samo između web okruženja ili lokalne verzije), dok primjerice odabir programskog jezika Python ili C# nude mnogo širi raspon okruženja. Neki nastavnici biraju poučavanje programiranja bez okruženja, odnosno rad u običnom uređivaču teksta, smatrajući da samo okruženje skriva ili komplicira proces programiranja [66]. Takvo što nije moguće za vizualne programske jezike, ali njihova okruženja u pravilu ne bi trebala složena. Mnogi početnici su zaista preplavljeni mogućnostima složenih okruženja, na što ćemo se osvrnuti u idućem dijelu. S druge strane postoji mišljenje da stariji nastavnici biraju "jednostavniju varijantu" jer su oni također učili programirati bez složenih *integriranih razvojnih okruženja* (eng. integrated development environment, IDE) kao što je MS Visual Studio.

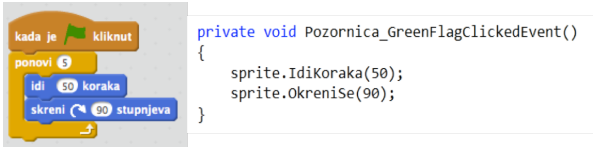
Premda su mnogi oduševljeni Scratch-om, smatra se da je jedan od nedostataka Scratch-a što je težak prijelaz iz tog okruženja u tradicionalno bez nekog posrednog alata koji bi bio poveznica između koncepata koji se usvajaju u Scratch-u i načina na koji se ti koncepti mogu primijeniti u nekom drugom programskom jeziku gdje je važna sintaksa [67], [68] kao što je ideja s prijelazom iz Alice u Java programski jezik [45]. Način rješavanja problema prijelaza iz Scratch-a u C# ćemo opisati u idućem dijelu.

III. Posredovani prijenos

Odabirom vizualnog programskog jezika koji će biti početni jezik također uvjetujemo naknadno učenje i poučavanje programiranja. Primjerice, Alice i Greenfoot su zasnovani na Java programkom jeziku, te bi bilo logično da budu prvi susret s programiranjem, ako učenici kasnije uče Javu. Svaki od ta tri navedena primjera jezika ima određene prednosti ili nedostatke ovisno o kontekstu. Smatra se da Scratch nije početno zamišljen za poučavanje objektno-orijentiranog programiranja (OOP), dok Alice i Greenfoot predstavljaju pravi uvod u OOP obzirom da u potpunosti podržavaju OOP pristup [69]. Greenfoot je moćniji od Scratch-a primjerice za složenije izračune kod kojih je Scratch spor kao i Alice, ali je s druge strane početak rada ili učenja Greenfoot-a teži. Iako se možda taj početni trud isplati za kasnije učenje OOP, odnosno Jave, ovdje se zapravo koncentriramo na usvajanje osnovnih koncepata programiranja temeljenih na tradicionalnom pristupu kojim se poučavaju slijed, grananje i ponavljanje [64] što je u potpunosti pokriveno jednostavnijim pristupom u Scratch-u. Papert je smatrao da programski jezik mora imati *niski prag* (eng. low floor) (odnosno jednostavan početak) i *visok strop* (eng. high ceiling) (mogućnost izrade složenih projekata tijekom vremena) [67]. Dodatno se smatra da bi morao imati i *široke zidove* (eng. wide walls) kako bi podržavao različite interese i stilove. Nije ni malo jednostavno zadovoljiti sva tri uvjeta, ali se Scratch nastoji tome približiti.

Upotrebom vizualnog jezika kao prvog jezika s kojim se studenti upoznaju, izbjegava se složena sintaksa, te se studenti koncentriraju na rješavanje problema koji su u praksi složeniji nego što bi ih bili u stanju riješiti uz neki moćniji programski jezik kao što je Python ili C#. Nakon početnog rješavanja problema uz pomoć vizualnog programskog jezika, studenti u skladu s idejom *posredovanog prijenosa* (eng. mediated transfer) [45] prelaze na složeniji programski jezik.

Studenti za razliku od djece brzo "prerastu" vizualno programsko okruženje Scratch-a koje kod složenijih zadataka postaje nepregledno jer se primjerice ne može pretraživati. Međutim, da bi se mogao ostvariti prijenos naučenih koncepata iz programskog jezika tipa Scratch u moćniji programski jezik kao što je npr. C#, potrebno je pripremiti i prilagoditi okruženje temeljem pristupa *didaktičkog skrivanja* gdje će se neke funkcije ili metode programskog jezika skriti od učenika. Na Slici 5 je prikazan isti program napisan u Scratch-u i u C#-u.



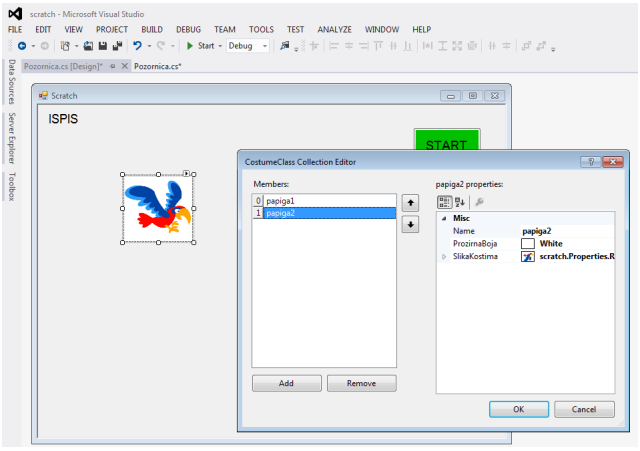
Slika 5. Program napisan u Scratch-u (lijevo) i u C#-u (desno)

Student ili učenik će u C#-u upotrijebiti metodu *Move(50)* kojom će se *lik* (eng. Sprite) pomaknuti za 50 točkica kao što bi to napravio i u Scratch-u, ali ne mora znati kako je ta metoda napravljena. Konkretni primjer metode zahtijeva znanje iz programiranja (kako se pišu metode), te znanje iz matematike obzirom da pomak od 50 točkica ovisi u kojem je smjeru trenutno lik okrenut i koliko će to biti točkica po X, odnosno Y osi (izračunavanje pomoću trigonometrijskih funkcija). Nakon što se student upozna detaljnije s programskim jezikom, onda će biti u stanju razumjeti izradu metode, te će je moći promijeniti ili prilagoditi svojim potrebama.

Smatra se da višak informacija može ometati učenika, te da jednostavno prisutnost nekog elementa na zaslonu računala koji se ne koristi za trenutni problem stvara poteškoće [70] prema teoriji kognitivnog opterećenja (eng. cognitive load theory) [71].

Microsoft Visual Studio je korisničko okruženje kojeg studenti koriste pri programiranju u C#-u. Obzirom da je to također i profesionalno okruženje za razvoj, ima mnoštvo mogućnosti i samim tim *ometala* (eng. distractors) koji mogu zbuniti početnika. Stoga je potrebno početniku dati problem kojeg već zna riješiti u jednostavnom okruženju kao što je Scratch kako bi ga prenio u profesionalno okruženje tipa MS Visual Studio. Ako početnik kod učenja programiranja počne koristiti profesionalno okruženje, onda će kasnije uštedjeti vrijeme potrebno za privikavanje jer će s novim potrebama samo naučiti nove detalje iz okruženja koje mu trebaju [62].

Na Slici 7. prikazan je primjer izrade grafičke aplikacije u prototipu C# okruženja za posredovani prijenos gdje je napravljena *kontrola* (eng. control) *Sprite*. Kontrole su osnovni elementi *windows* grafičkih korisničkih aplikacija s obrascima (eng. Windows Forms Applications).



Slika 6. Prototip C# okruženja za posredovani prijenos

Student u grafičkom okruženju oblikuje likove jednostavnim povlačenjem mišem i odabirom slike, imena lika i sl. te koristi gotove metode za kretanje, animaciju, ... slično kao što to već postoji u Scratch-u. Na taj način student mijenja okruženje i koristi istu logiku rješavanja problema. Drugi primjer skrivanja složenih koncepata je jednostavna metoda *Cekaj(brojSekundi)* koja omogućuje pauzu ili čekanje programa prema upisanom broju sekundi.

```
private void Cekaj(double brojSekundi)
{
    int ms = (int)(brojSekundi * 1000);
    Thread.Sleep(ms);
}
```

Slika 7. Primjer skrivanja

Međutim, unutar metode poziva se složeni koncept programske *niti* (eng. thread). Scratch omogućuje usvajanje koncepata paralelnog izvršavanja skripti [63] (eng. concurrency) jer tako likovi funkcioniraju. Primjerice, ako želimo da papiga istovremeno maše krilima i kreće se, jedna skripta će se baviti mahanjem, a druga pomicanjem lika. To je jako jednostavno riješiti u Scratch-u, no problem paralelnog izvršavanja odnosno *višenitnih aplikacija* (eng. multithreading) u C# je jako složen, te spada u napredno programiranje.

Nakon što studenti nauče koristiti prijelazno okruženje sa Slike 7., malo po malo će im se otkrivati unutrašnji ustroj pojedinih dijelova, te će biti u mogućnosti u potpunosti prijeći na tekstualno okruženje. Studenti bi mogli koristiti Scratch i za brzu izradu prototipa igre.

A. *Odabir problema*

Često nastavnici imaju problema i kod odabira projekta koji bi bio dovoljno blizak problemu iz stvarnog svijeta radi poticanja interesa učenika, ali ipak ne pretjerano složen da ga nisu u stanju riješiti. Na primjer, izrada knjigovodstvenog sustava je projekt iz stvarnog svijeta, no da bi ga učenici mogli izraditi moraju razumjeti i knjigovodstvo. Osim odabira projekta, potrebno je strukturirati jednostavnije probleme kojima učenici vježbaju osnovne vještine prije početka rada na projektu, ali koje su ključne za izradu projekta [72].

Alternativa traženju problema iz stvarnog svijeta, a primjenjiva je na različite dobi je *izrada igara*, obzirom da većina učenika igra ili je bar upoznata s igrama. Uvođenje izrade igara u početne predete iz programiranja bi prema tome trebalo povećati entuzijazam studenata ne samo za programiranje već za računarsvo općenito. *Učenje zasnovano na igrama* (eng. game-based learning, GBL, gamification) se globalno može podijeliti na [19]:

1) učenje kroz igranje obrazovnih igara,
2) izradu igara s ciljem usvajanja koncepata iz programiranja.

Igra zapravo učenicima predstavlja zabavu, a sama izrada igara im je često zanimljiva premda cilj nije zapravo napraviti igru već naučiti programirati. Smatra se da izrada igara omogućuje prilagodbu načina učenja različitim karakteristikama učenika [73]. Svatko će pristupiti rješavanju na svoj način, te će zapravo i učiti na različit način. Smatra se da izrada igara uključuje koncepte iz biheviorizma, kognitivizma i konstruktivizma.

[74] Games have been demonstrated to increase learning (de-Marcos et al., 2014; Gee, 2007), and it is possible that the incorporation of certain game mechanics with clear learning objectives in mind can create an engaging and meaningful experience However, many game mechanics are best applied through use of a computer or virtual world (e.g., immediate feedback, lesson plans tailored to each student). Future research should also consider the unique affordances of interactive technology and create gamified systems that take full advantage of these digital mechanics.

Vizualni programski jezici kao što je npr. Scratch, Byob i sl. omogućuju učenicima različite dobi koncentriranje na rješavanje problema bez opterećenja sa sintaksom, te se može primijeniti i na visoko obrazovanje za početni predmet iz programiranja [75]. Rješavanje problema se provodi i uz pomoć programiranja robota. Upotreba robota pri učenju i

poučavanju programiranja potiče aktivno učenje, a ako se projekt radi u grupama onda potiče i suradnju unutar grupe koja radi na zajedničkom rješavanju problema [76]. Često će se takav tip projekata i raditi u grupi ili timu jer je obično i oprema (broj robota) ograničena, te postoje brojni primjeri primjene rada u grupi [76], [77], [78], [79], ili u paru [80], a navedeni primjeri se uklapaju u integrirani STEM pristup.

Robotika omogućuje učenicima učenje o različitim konceptima (npr. senzori, motor i sl.) iz područja robotike [12], ali kako te robote mogu i programirati onda uče i koncepte iz programiranja. Naravno, tu opet dolazimo do problema programskog jezika kad učenici počinju više pažnje posvećivati samim naredbama. Autori u [81] su izradili programsku podršku koja vizualne ikonice prevodi u programski jezik JavaScript, a ta je karakteristika omogućila učenicima bolje zaključivanje, te nisu bili ograničeni jezikom. Vizualni jezik koji su koristili za programiranje robota je bio prekompliciran za malu djecu, a prejednostavan za više uzraste.

Učenici uz pomoć robota mogu dodatno učiti i koncepte iz matematike i PBL [82]. Učenici vide kako se robot ponaša, odnosno reagira na njihove naredbe, mogu se lakše "staviti na mjesto" robota, odnosno shvatiti zašto ne izvršava ono što su očekivali, te nakon toga ispraviti program.

IV. Zaključak

Učenje i poučavanje programiranja je predmet brojnih rasprava, radova i istraživanja. Općenito se smatra teškim, te se često ističe problem odustajanja na početnim predmetima iz programiranja na fakultetima, ali problemi pri učenju i poučavanju programiranja su prisutni u različitoj dobi učenika. U današnje vrijeme razvoja moderne tehnologije programiranje se smatra dijelom računalne pismenosti, te je trend uključivanja programiranja, robotike, ali i drugih STEM područja u sve niže uzraste. Programiranje je često jedna od poveznica integriranog STEM pristupa.

Studenti na našem fakultetu su budući nastavnici različitih smjerova koji pripadaju STEM područjima, te bi svakako trebali biti upoznati s programiranjem. Većina studenata su ipak budući nastavnici informatike koji osim što sami moraju naučiti programirati moraju naučiti na koji način mogu prenijeti znanje na buduće generacije učenika. Trenutni problem je što je većina studenata koji upisuju početne predmete programiranja na našem fakultetu zapravo bez prethodnog znanja iz programiranja, pa su zapravo odrasle osobe od kojih se očekuje da u vremenu od jednog semestra usvoje koncepte programiranja, sintakse programskih jezika, te okruženja u kojem programiraju.

Pristup poučavanju odraslih osoba je drugačiji od pristupa kod djece, a također današnje generacije studenata pripadaju generacijama koje su odrasle uz tehnologiju. Zahtijevaju više poveznica s problemima iz stvarnog života s kojima se mogu poistovjetiti i aktivno učenje, ali ih je često teško motivirati i brzo gube motivaciju. Kao poseban problem ističe se uvjetovanje naučeno u prethodnom obrazovanju, odnosno često kad se susretnu s nečim što se smatra obrazovanjem jednostavno zauzmu stav "nauči me" [23]. To je stav "zavisnosti" koji je tipičan za djecu, ali tretiranje odraslih kao djece posljedično rezultira unutarnjim konfliktom s potrebom da budu neovisni i samostalni učenici (eng. self-directed learners). Tipičan mehanizam rješavanja takvog konflikta je bijeg iz situacije koja ga uzrokuje pa se smatra da je i to jedan od razloga odustajanja od obrazovanja.

Studenti zapravo moraju biti više samostalni i napraviti prijelaz od zavisnih učenika iz prethodnog školovanja na što više samostalnog rada. Nastavnici nastoje pokrenuti aktivnost kroz rješavanje problema s kojima se studenti mogu poistovjetiti, ali kako bi to primijenili na programiranje, najprije trebaju usvojiti osnovne koncepte kao što su primjerice spomenute algoritamske strukture slijeda, grananja i ponavljanja. Prepreku usvajanju koncepata programiranja često predstavljaju jezik i okruženje u kojem rade. Uklanjanjem problema sintakse pomoću nekog od vizualnih programskih jezika omogućuje se studentima usvajanje osnovnih koncepata programiranja pružajući im pri tome i primjer načina na koji će moći ubuduće poučavati djecu programiranju. Ako pored uklanjanja sintakse uvedemo simbole, odnosno posebne opipljive predmete u obliku kartica ili blokova, onda dodatno snižavamo dobne granice za učenje osnovnih koncepata programiranja. Ovdje treba napomenuti da *opipljivo programiranje* ipak nije primjereno za studente, ali je logičan nastavak pojednostavljenja sintakse i okruženja. Studenti u pravilu u vizualnom okruženju mogu napraviti više nego u tekstualnom, ali bolji studenti znatno brže to okruženje "prerastu" u odnosu na djecu.

Ono što obično nedostaje između početnog jednostavnog jezika kojim bi se studentima mogli približiti osnovni koncepti je poveznica, odnosno prijelaz na složeniji jezik. Pristupom posredovanog prijenosa iz Scratch-a u C# omogućit će se jednostavniji prijelaz u profesionalno okruženje za razvoj programske podrške i složeni profesionalni programski jezik.

Pronalaženje problema i projekata iz stvarnog svijeta s kojima se studenti mogu poistovjetiti nije jednostavno, te se zbog same prirode vizualnog programskog jezika na neki način nameće izrada igara kao logičan odabir obzirom da su igre bogate različitim interakcijama, algoritmima, umjetnom inteligencijom, i sl., a dosta su bliske većini studenata, te se smatra da djeluju motivirajuće. Naravno u vizualnom okruženju je moguće izraditi i simulacije (npr. iz fizike, biologije), poučavati matematiku i sl.

Potrebno je provesti istraživanje kojim će se vrednovati učinkovitost pripremljenog okruženja tijekom primjene u odnosu na usvajanje znanja i motivaciju.

Literatura

[1] K. Turville, G. Meredith, and P. Smith, "Understanding novice programmers: their perceptions and motivations," in *ASCILITE-Australian Society for Computers in Learning in Tertiary Education Annual Conference*, vol. 2012, no. 1, 2012.

[2] K. Williams, "Literacy and computer literacy: Analyzing the nrc's "being fluent with information technology"," *Journal of Literacy and Technology*, vol. 3, no. 1, pp. 1–20, 2003.

[3] G. McAllister and S. Alexander, "Key aspects of teaching and learning in computing science," *A Handbook for Teaching and Learning in Higher Education*, p. 282, 2008.

[4] W. F. McComas, *The language of science education: An expanded glossary of key terms and concepts in science teaching and learning*. Springer Science & Business Media, 2013.

[5] D. R. Heil, G. Pearson, and S. E. Burger, "Understanding integrated stem education: Report on a national study," June 2013.

[6] S. W. Haugland, "Early childhood classrooms in the 21st century: Using computers to maximize learning," *Young Children*, vol. 55, no. 1, pp. 12–18, 2000.

[7] H. Seybert, "Internet use in households and by individuals in 2011," *Eurostat statistics in focus*, vol. 66, pp. 1–8, 2011.

[8] M. Bers, I. Ponte, K. Juelich, A. Viera, and J. Schenker, "Teachers as designers: Integrating robotics in early childhood education," *Information Technology in childhood education*, vol. 1, pp. 123–145, 2002.

[9] A. J. Reynolds, J. A. Temple, S.-R. Ou, I. A. Arteaga, and B. A. White, "School-based early childhood education and age-28 well-being: Effects by timing, dosage, and subgroups," *Science*, vol. 333, no. 6040, pp. 360–364, 2011.

[10] A. Sullivan and M. U. Bers, "Robotics in the early childhood classroom: learning outcomes from an 8-week robotics curriculum in pre-kindergarten through second grade," *International Journal of Technology and Design Education*, vol. 26, no. 1, pp. 3–20, 2016.

[11] M. Elkin, A. Sullivan, and M. U. Bers, "Implementing a robotics curriculum in an early childhood montessori classroom," *Journal of Information Technology Education: Innovations in Practice*, vol. 13, pp. 153–169, 2014.

[12] M. U. Bers and M. S. Horn, "Tangible programming in early childhood: Revisiting developmental assumptions through new technologies," *High-tech tots: Childhood in a digital world*, pp. 49–70, 2010.

[13] D. H. Clements, "The future of educational computing research: The case of computer programming," *Information Technology in Childhood Education Annual*, vol. 1999, no. 1, pp. 147–179, 1999.

[14] T. S. McNerney, "From turtles to tangible programming bricks: explorations in physical language design," *Personal and Ubiquitous Computing*, vol. 8, no. 5, pp. 326–337, 2004.

[15] F. R. Sullivan and M. A. Moriarty, "Robotics and discovery learning: pedagogical beliefs, teacher practice, and technology integration," *Journal of Technology and Teacher Education*, vol. 17, no. 1, pp. 109–142, 2009.

[16] G. Gorozidis and A. G. Papaioannou, "Teachers' motivation to participate in training and to implement innovations," *Teaching and Teacher Education*, vol. 39, pp. 1–11, 2014.

[17] M. Bers, S. Seddighin, and A. Sullivan, "Ready for robotics: Bringing together the t and e of stem in early childhood teacher education," *Journal of Technology and Teacher Education*, vol. 21, no. 3, pp. 355–377, 2013.

[18] E. Cejka, C. Rogers, and M. Portsmore, "Kindergarten robotics: Using robotics to motivate math, science, and engineering literacy in elementary school," *International Journal of Engineering Education*, vol. 22, no. 4, p. 711, 2006.

[19] S. Mladenović, D. Krpan, and M. Mladenović, "Using games to help novices embrace programming: From elementary to higher education," *International journal of engineering education*, vol. 32, no. 1, pp. 521–531, 2016.

[20] L. Ma, J. Ferguson, M. Roper, and M. Wood, "Investigating and improving the models of programming concepts held by novice programmers," *Computer Science Education*, vol. 21, no. 1, pp. 57–80, 2011. [Online]. Available: http://dx.doi.org/10.1080/08993408.2011.554722

[21] M. A. Tchoshanov, *Engineering of Learning: Conceptualizing e-Didactics.* Moscow: UNESCO Institute for Information Technologies in Education, 2013.

[22] V. Poljak, *Didaktika*, 8th ed. Školska knjiga, 1990.

[23] M. S. Knowles, E. F. Holton Iii, and R. A. Swanson, *The adult learner: The definitive classic in adult education and human resource development*. Routledge, 2014. [Online]. Available: http://elsevier.com

[24] J. Bennedsen, "Teaching and learning introductory programming - a model-based approach," PhD Thesis, University of Oslo, Norway, 2008.

[25] M. W. Shreeve, "Beyond the didactic classroom: educational models to encourage active student involvement in learning," *Journal of Chiropractic Education*, vol. 22, no. 1, pp. 23–28, 2008.

[26] G. Kearsley and B. Shneiderman, "Engagement theory: A framework for technology-based teaching and learning." *Educational technology*, vol. 38, no. 5, pp. 20–23, 1998.

[27] W. Pullan, S. Drew, and S. Tucker, "A problem based approach to teaching programming," in *Proceedings of the International Conference on Frontiers in Education: Computer Science and Computer Engineering (FECS)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2013, p. 1.

[28] W. McKeachie and M. Svinicki, *McKeachie's teaching tips*. Cengage Learning, 2013.

[29] L. B. Nilson, *Teaching at Its Best: A Research-Based Resource for College Instructors*. Jossey-Bass Wiley Imprint, 2003, vol. 2nd.

[30] 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.

[31] 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.

[32] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, "A multi-national, multi-institutional study of assessment of programming skills of first-year cs students," *ACM SIGCSE Bulletin*, vol. 33, no. 4, pp. 125–180, 2001.

[33] R. Lister, A. Berglund, I. Box, C. Cope, A. Pears, C. Avram, M. Bower, A. Carbone, B. Davey, and M. de Raadt, "Differing ways that computing academics understand teaching," in *Proceedings of the ninth Australasian conference on Computing education-Volume 66*. Australian Computer Society, Inc., 2007, pp. 97–106.

[34] M. Matijević, "Između didaktike nastave usmjerene na učenika i kurikulumske teorije," *U: Zbornik radova Četvrtog kongresa matematike. Zagreb: Hrvatsko matematičko društvo i Školska knjiga*, pp. 391–408, 2010.

[35] E. Nuutila, S. Törmä, and L. Malmi, "Pbl and computer programming—the seven steps method with adaptations," *Computer Science Education*, vol. 15, no. 2, pp. 123–142, 2005.

[36] J. J. Kaasbøll, "Exploring didactic models for programming," in *NIK 98–Norwegian Computer Science Conference*, 1998, pp. 195–203.

[37] C. Bruce, L. Buckingham, J. Hynd, C. McMahon, M. Roggenkamp, and I. Stoodley, "Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university," *Transforming IT education: Promoting a culture of excellence*, pp. 301–325, 2006.

[38] T.-C. Yang, G.-J. Hwang, S. J. Yang, and G.-H. Hwang, "A two-tier test-based approach to improving students' computer-programming skills in a web-based learning environment." *Educational Technology & Society*, vol. 18, no. 1, pp. 198–210, 2015.

[39] G. Futschek, "Extreme didactic reduction in computational thinking education," *Learning while we are connected*, vol. 3, pp. 1–6, 2013.

[40] S. Psycharis, E. Makri-Botsari, and G. Xynogalas, "The use of educational robotics for the teaching of physics and its relation to self-esteem," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008, pp. 132–142.

[41] S. B. Fee and A. M. Holland-Minkley, "Teaching computer science through problems, not solutions," *Computer Science Education*, vol. 20, no. 2, pp. 129–144, 2010.

[42] E. De Graaf and A. Kolmos, "Characteristics of problem-based learning," *International Journal of Engineering Education*, vol. 19, no. 5, pp. 657–662, 2003.

[43] J. Kay, M. Barg, A. Fekete, T. Greening, O. Hollands, J. H. Kingston, and K. Crawford, "Problem-based learning for foundation computer science courses," *Computer Science Education*, vol. 10, no. 2, pp. 109–128, 2000.

[44] J. Larmer, "Project-based learning vs. problem-based learning vs. x-bl," *Retrieved March*, vol. 8, p. 2024, 2014.

[45] W. Dann, D. Cosgrove, D. Slater, D. Culyba, and S. Cooper, "Mediated transfer: Alice 3 to java," in *Proceedings of the 43rd ACM technical symposium on Computer Science Education*. ACM, 2012, pp. 141–146.

[46] L. Grandell, M. Peltomäki, R.-J. Back, and T. Salakoski, "Why complicate things?: introducing programming in high school using python," in *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52*. Australian Computer Society, Inc., 2006, pp. 71–80.

[47] M. E. Caspersen, "Educating novices in the skills of programming," PhD Dissertation, Department of Computer Science, University of Aarhus, 2007.

[48] M. Kölling and J. Rosenberg, "Blue -A Language for Teaching Object-Oriented Programming BLUE – A LANGUAGE FOR TEACHING OBJECT-ORIENTED PROGRAMMING," in *Proceedings of the 8th Australasian Conference on Computing Education*. Australian Computer Society, Inc., 2006, pp. 71–80.

[49] M. Guzdial, "Programming environments for novices," *Computer science education research*, vol. 2004, pp. 127–154, 2004.

[50] E. Lahtinen, K. Ala-Mutka, and H.-M. Järvinen, "A study of the difficulties of novice programmers," in *ACM SIGCSE Bulletin*, vol. 37, no. 3. ACM, 2005, pp. 14–18.

[51] 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.

[52] C. Kelleher and R. Pausch, "Lowering the barriers to programming: A survey of programming environments and languages for novice programmers," CMU-CS-03-137. Carnegie Mellon Univ Pittsburgh PA School of Computer Science, 2003. 76, Tech. Rep., 2003.

[53] M. Boshernitsan and M. S. Downes, "Visual programming languages: A survey," Computer Science Division (EECS), Tech. Rep., 2004.

[54] E. R. Kazakoff, A. Sullivan, and M. U. Bers, "The Effect of a Classroom-Based Intensive Robotics and Programming Workshop on Sequencing Ability in Early Childhood," *Early Childhood Education Journal*, vol. 41, no. 4, pp. 245–255, 2013.

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

[56] K. Richardson, *Models of Cognitive Development*. Psychology Press, 1998. [Online]. Available: https://books.google.hr/books?id=qYQrsrb6in8C

[57] 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*. ACM, 2013, pp. 1–10.

[58] D. H. Clements and S. McMillen, "Rethinking" concrete" manipulatives," *Teaching children mathematics*, vol. 2, no. 5, pp. 270–279, 1996.

[59] H. Suzuki and H. Kato, "Interaction-level support for collaborative learning: Algoblock—an open programming language," in *The first international conference on Computer support for collaborative learning*. L. Erlbaum Associates Inc., 1995, pp. 349–355.

[60] H. Kato, K. Yamazaki, H. Suzuki, H. Kuzuoka, H. Miki, and A. Yamazaki, "Designing a video-mediated collaboration system based on a body metaphor," in *Proceedings of the 2nd international conference on Computer support for collaborative learning*. International Society of the Learning Sciences, 1997, pp. 148–156.

[61] P. Wyeth, "How young children learn to program with sensor, action, and logic blocks," *Journal of the Learning Sciences*, vol. 17, no. 4, pp. 517–550, 2008. [Online]. Available: http://dx.doi.org/10.1080/10508400802395069

[62] R. Mason, "Designing introductory programming courses: the role of cognitive load," 2012.

[63] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Learning computer science concepts with scratch," *Computer Science Education*, vol. 23, no. 3, pp. 239–264, 2013.

[64] 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.

[65] L. E. Winslow, "Programming pedagogy—a psychological overview," *SIGCSE Bull.*, vol. 28, no. 3, pp. 17–22, Sep. 1996. [Online]. Available: http://doi.acm.org/10.1145/234867.234872

[66] M. De Raadt, R. Watson, and M. Toleman, "Language trends in introductory programming courses," in *Proceedings of the 2002 Informing Science+ Information Technology Education Joint Conference (InSITE 2002)*. Informing Science Institute, 2002, pp. 229–337.

[67] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman *et al.*, "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[68] M. Koorsse, "An Evaluation of Programming Assistance Tools to Support the Learning of IT Programming: A Case Study in South African Secondary Schools," Ph.D. dissertation, Nelson Mandela Metropolitan University, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360131514002735

[69] I. Utting, S. Cooper, M. Kölling, J. Maloney, and M. Resnick, "Alice, greenfoot, and scratch–a discussion," *ACM Transactions on Computing Education (TOCE)*, vol. 10, no. 4, p. 17, 2010.

[70] R. Mason and G. Cooper, "Distractions in programming environments," in *Proceedings of the Fifteenth Australasian Computing Education Conference-Volume 136*. Australian Computer Society, Inc., 2013, pp. 23–30.

[71] J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*, 1st ed., ser. Explorations in the Learning Sciences, Instructional Systems and Performance Technologies 1. Springer-Verlag New York, 2011, vol. 10.

[72] A. Soares, F. Fonseca, and N. L. Martin, "Teaching introductory programming with game design and problem-based learning." *Issues in Information Systems*, vol. 16, no. 3, 2015.

[73] G. I. Bíró, "Didactics 2.0: A pedagogical analysis of gamification theory from a comparative perspective with a special view to the components of learning," *Procedia-Social and Behavioral Sciences*, vol. 141, pp. 148–151, 2014.

[74] M. D. Hanus and J. Fox, "Assessing the effects of gamification in the classroom: A longitudinal study on intrinsic motivation, social comparison, satisfaction, effort, and academic performance," *Computers & Education*, vol. 80, pp. 152–161, 2015.

[75] D. Krpan, S. Mladenović, and G. Zaharija, "Visual programming languages in higher education," in *16. CARNetova korisnička konferencija-CUC 2014*, 2014.

[76] J. Arlegui, E. Menegatti, M. Moro, and A. Pina, "Robotics, computer science curricula and interdisciplinary activities," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008, pp. 10–21.

[77] S. Atmatzidou, I. Markelis, and S. Demetriadis, "The use of lego mindstorms in elementary and secondary education: game as a way of triggering learning," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008.

[78] K. Papanikolaou, S. Frangou, and D. Alimisis, "Teachers as designers of robotics-enhanced projects: the terecop course in greece," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008, pp. 100–111.

[79] P. Petrovič and R. Balogh, "Educational robotics initiatives in slovakia," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008, pp. 122–131.

[80] D. Alimisis, "Teacher education on robotics-enhanced constructivist pedagogical methods," *School of Pedagogical and Technological Education, Athens*, 2009.

[81] E. Micheli, M. Avidano, and F. Operto, "Semantic and epistemological continuity in educational robots' programming languages," in *Proceedings of the TERECoP Workshop Teaching with robotics, Conference SIMPAR*, 2008.

[82] C. Rogers and M. Portsmore, "Bringing engineering to elementary school," *Journal of STEM Education: innovations and research*, vol. 5, no. 3/4, p. 17, 2004.