Blockly, Pišek in poučevanje programiranja

Blockly, Pišek and teaching programming

Gregor Anželj
Gimnazija Bežigrad
Peričeva ulica 4
Ljubljana
Gregor.Anzelj@gimb.org

Gregor Jerše
UL FRI
Večna pot 113
Ljubljana
gregor.jerse@fri.uni-lj.si

Matija Lokar
UL FMF
Jadranska ulica 19
Ljubljana
matija.lokar@fmf.uni-lj.si

POVZETEK

V prispevku sta opisana dva projekta, ki sta namenjena predvsem tistim učiteljem računalništva in informatike tako v osnovni kot tudi v srednji šoli, ki se odločajo, s katerim programskim jezikom uvesti osnovne pojme programiranja.

Prvi projekt je prosto dostopen e-učbenik Slikovno programiranje, ki s pomočjo jezika Blockly učenca vodi v prve korake pri programiranju. Drugi projekt pa je spletna storitev Pišek, ki na avtomatski način preverja pravilnost v Blocklyu zapisano rešitev različnih problemov.

V uvodnih razdelkih so predstavljeni razlogi za uporabo slikovnih jezikov in sistemov za avtomatsko preverjanje.

Ključne besede

Poučevanje, programiranje, spletna storitev, slikovni jeziki, avtomatsko preverjanje

ABSTRACT

The paper describes two projects designed primarily for those primary and secondary computer science teachers who decide which programming language to use at introducing the basic concepts of programming.

The first project is a freely accessible e-textbook Visual Programming, which, with the help of the Blockly, guides the learner to make the first steps in programming. The other project is the web service Pišek, which automatically checks the correctness of the solutions coded in Blockly.

The introductory sections provide grounds for the use of visual languages and systems for automatic assessment.

Keywords

Teaching, programming, web service, visual languages, systems for automatic assessment

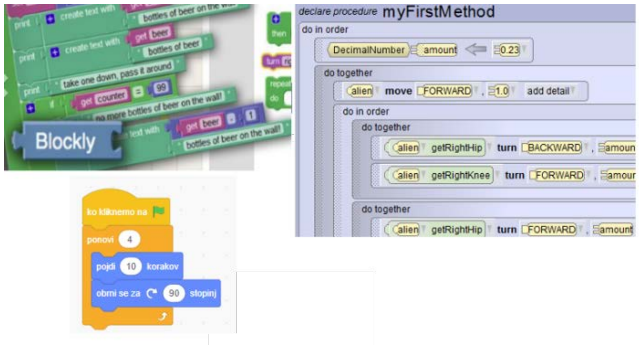
1. UVOD

Pomemben cilj pri učenju programiranja je učenje oziroma razumevanje temeljnih konceptov. Drugi cilj, vendar mogoče še pomembnejši, je mlajše učence navdušiti za računalništvo in informatiko (RIN) in jim pokazati, da je učenje RIN zabavno. Pogosto je ta cilj še pomembnejši od učenja konceptov ([1]).

Zato je zelo pomembna izbira okolja, v katerem se začetnik prvič sreča s programiranjem. V zadnjem času vrsta strokovnjakov proučuje, ali niso za začetnike morda najprimernejša tako imenovana Grafična blokovna okolja za programiranje ali GBOP ([2]). To so okolja, ki vsebujejo grafično okolje za izvajanje programov (krmiljenje figur). Te pa krmilimo s pomočjo slikovnih ali grafičnih programskih jezikov. Slikovni ali grafični programski jezik (ang. Visual programming language - VPL) je po [3] vsak programski jezik (ali okolje), ki uporabniku omogoča ustvarjanje programov tako, da uporabnik premika, ali razporeja elemente slikovnega programskega jezika.

Prednost slikovnih programskih jezikov je v tem, da omogočajo ustvarjanje ali gradnjo sintaktično pravilnih programov, saj lahko uporabnik združuje elemente na točno določen način. S tem se izogne sintaktičnim napakam, kar posledično olajša učenje programskega jezika začetnikom. Ti se lahko namesto s sintakso, t.j. slovnično pravilnostjo programa, ukvarjajo s postopkom, ki bo privedel do rešitve problema.

Obstaja cela množica slikovnih programskih jezikov, ki jih lahko uporabljamo za različne namene. Najbolj znani slikovni programski jeziki za učenje programiranja so: Alice, App Inventor, Blockly, Lego Mindstorms NXT, Pencil Code, Scratch, Snap! in številni drugi.



Slika 1: Nekaj primerov slikovnih jezikov

Da so razvoj in raziskave na tem področju zelo živahne, priča že dejstvo, da se tudi poimenovanje tovrstnih jezikov še ni ustalilo. Tako različni avtorji, ko opisujejo jezike, kot jih vidimo na Slika 1 govorijo o Visual programming languages, Block-based programming languages, Visual block programming languages, Graphical programming languages, pa še kak drug izraz srečamo. Tudi v slovenščini še ni ustaljenega poimenovanja (grafični jeziki, jeziki s kockami, blokovni jeziki, slikovni jeziki…).

Glavna prednost, ki naj bi jo prinašala uporaba slikovnih jezikov, je ta, da zmanjšuje nivo kognitivne obremenitve [4], ki mu je

izpostavljen učenec, ko mora reševati določen problem s pomočjo pisanja kode.

Teorija kognitivne obremenitve pravi, da se pri učenju srečamo s tremi oblikami obremenitve (napora). Ti so (primeri oblik so povzeti po [5]):

- nujna (intrinsic): obremenitev, ki nastopa zaradi samega problema (na primer razumevanje, kaj je spremenljivka)
- vgradna (german): obremenitev, ki je potrebna za to, da se nečesa dolgotrajno naučimo – vgradimo v svoje znanje (na primer vedenje, da kontrolna spremenljivka ob vsaki izvedbi zanke dobi novo vrednost)
- tuja (extraneous): obremenitev, da razumemo navodila za reševanje – splošno vse v izobraževalnem gradivu, kar ni neposredno povezano s samim problemom (drugačna razporeditev ukazov v primeru kot je v učenčevem orodju)

Osnovni učinek učenja nam prinaša vgradna obremenitev. Zato želimo pri poučevanju doseči, da bo učenec vlagal kar se da velik napor v to obremenitev. Kot pravijo Yousoof in drugi v [6], se vse tri obremenitve seštevajo. Ker je velikost prve, nujne obremenitve pri posamezniku več ali manj nespremenljiva, skupen napor, ki ga lahko vložimo v učenje, pa več ali manj omejen, je nujno, da, če želimo dati večji delež drugi, tretjo zmanjšamo. Ker so navodila za reševanje (uporaba samega okolja za programiranje) pri slikovnem programiranju enostavnejša za razumevanje kot pri uporabi tekstovnega programiranja in urejevalnikov in je s tem tuja obremenitev manjša (glej. npr. [7], [8], [9]), je slikovno programiranje tako primerno za začetnika.

Kot piše Mark Guzdial v [10], obstaja več študij, ki kažejo, da je znanje slikovnih programskih jezikov prenosljivo na tekstovno usmerjene jezike. Tako Chris Hundhausen ([11]), Shuchi Grover ([12], [13]) David Weintrop ([14], [15], [16]) in drugi opozarjajo na enostavnost, s katero učenci prenesejo znanje (npr. glede spremenljivk, iteracijskih struktur in pogojev) iz slikovnih jezikov (kot so Blockly, Alice, Scratch in drugi) v tekstovne jezike (npr. Java ali Python). Doktorska disertacija Davida Weintropa ([14]) pove veliko že v prvih besedah naslova "Modality Matters." Weintrop v disertaciji poroča, da se učenci učijo več in hitreje, ko uporabljajo slikovne jezike kot takrat, ko uporabljajo tekstovne. Vendar pa je pri spoznavanju določenih konceptov računalništva in informatike praktično nujen prehod k tekstovnim jezikom. Prav tako raziskave kot tudi naše osebne izkušnje kažejo, da določene skupine učencev lažje in bolje napredujejo ob uporabi tekstovnih jezikov.

Ker je programiranje veščina, se jo učenci lahko naučijo le z veliko vaje. O tem pričajo številne raziskave, med drugim [17]. Učitelji morajo pripraviti veliko nalog, jih razdeliti učencem, sproti preverjati njihov napredek in jim po potrebi pomagati. To še posebej pride do izraza pri poučevanju programiranja začetnikov. Pri odkrivanju sintaktičnih napak v slikovnih jezikih ni večjih težav, saj jih načeloma ni. Večja težava je s semantičnimi napakami. Če pomoč učitelja ni takoj na voljo, to močno upočasni napredek učencev, saj ne vedo, kako naprej. Pogosto se tudi zgodi, da učenci zaradi odsotnosti pomoči nalogo rešijo narobe ali pomanjkljivo in se tega niti ne zavedajo. Ker pa je velika večina začetniških napak preprosto rešljivih, nam lahko priskočijo na pomoč sistemi za avtomatsko preverjanje programskih rešitev. Več o uporabi tovrstnih sistemov, med drugim tudi slovenskega, razvitega na UL FMF, si lahko preberemo v [18].

Kot ugotavljajo Papadakis in drugi v [9], je poučevanje programiranja zapletena naloga, ki je še toliko bolj zahtevna, ko poučujemo uvod v programiranje. V skupnosti učiteljev na globalni ravni poteka živahna debata o najboljših pristopih pri poučevanju uvoda v programiranje. Zato smo tudi učiteljem v slovenskih šolah želeli dati na voljo učna gradiva in pripomočke, ki bi jim omogočala, da se bodo po lastni presoji odločali, ali bi pri poučevanju uporabljali slikovne ali tekstovne programske jezika. Menimo namreč, da učni položaj v razredu najbolje oceni vsak učitelj samo. V skladu s svojo strokovno presojo se odloči o primernem pristopu. Seveda pa mora v ta namen imeti ustrezna sredstva (predvsem učna gradiva in pripomočke).

Zato smo pripravili različico e-učbenika za prve korake v programiranje z naslovom Slikovno programiranje. Po vzoru spletnega sistema za avtomatsko preverjanje Projekt Tomo ([19, 20, 21]) smo želeli tudi v naš prostor prinesti podoben sistem, ki pa bo dovoljeval uporabo slikovnega jezika Blockly. Sistem razvijamo na spletnem naslovu http://pisek.acm.si

2. BLOCKLY

Brodnik in drugi ([22]) ugotavljajo, da je, za razliko od večine razvitih držav v Evropi in svetu, kjer imajo obvezni pouk računalništva in informatike v različnih oblikah že v osnovni šoli, pri nas pouk računalništva in informatike obvezen le v 1. letniku srednje šole. To posledično pomeni, da se pri nas s poučevanjem osnov programiranja organizirano srečujejo šele dijaki. V starosti, ko bi dejansko že morali izvesti prehod iz slikovnih programskih jezikov na tekstovne programske jezike, imajo naši dijaki težave z osnovnimi koncepti programiranja.

Slikovna programska okolja, kot je na primer Blockly, omogočajo dijakom, da ustvarjajo programe na način, ki je bolj dostopen kot v okoljih za tekstovno programiranje. Ta okolja, namenjena izobraževanju, dijakom omogočajo programiranje brez ovire sintaktičnih napak, ki je prisotna v tradicionalnih tekstovnih programskih jezikih.

Različica e-učbenika za prve korake v programiranju uporablja slikovni jezik Blockly. Z uporabo slikovnega programskega jezika se izognemo trem oviram pri učenju programiranja, kot jih navajajo Bau in drugi ([8]):

1. Učenje in pomnjenje besedišča oziroma ključnih besed nekega programskega jezika je težko. Bloki zmanjšujejo te težave, saj je izbiranje bloka iz določene kategorije enostavnejše, kot pomnjenje ključne besede: pri uporabi blokov se opiramo na prepoznavanje, namesto priklica.

   Tipičen tekstovni programski jezik vsebuje med 100 in 200 ključnih besed. Scratch vsebuje okoli 130 različnih blokov. Prepoznavanje bloka (v ustrezni kategoriji) je bistveno hitrejše oziroma učinkovitejše, kot pomnjenje in priklic ene izmed 100-200 ključnih besed.

2. Programsko kodo je težko uporabljati, saj novim programerjem predstavlja veliko kognitivno obremenitev, ko se učijo sintakse novega programskega jezika. Bloki zmanjšujejo to obremenitev, saj kodo razdrobijo v manjše število elementov, ki imajo vsak svoj pomen.

Oglejmo si primer sintakse zanke for v programskem jeziku JavaScript:

```
for (var i = 0; i < 50; i++) { _ _ _ }
```

Če hočemo razumeti težavnost, s katero se soočajo novi programerji, si moramo predstavljati, da ta delček kode vsebuje pet besed (`for var i i i`), deset ločil (`( = ; < ; + + ) { }`) in dve števili (`0` in `50`). Skupaj to pomeni 17 različnih podatkov. Študije zmožnosti pomnjenja ugotavljajo, da smo ljudje zmožni pomnjenja okoli 7 različnih podatkov. Iz tega sledi, da je pomnjenje 17 ločenih podatkov preveč za novinca.

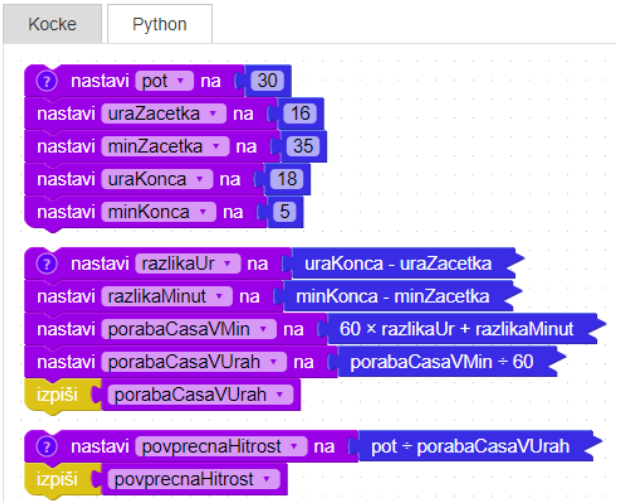
Izkušeni JavaScript programerji s to kodo seveda ne bi imeli popolnoma nobenih težav, saj so se naučili prepoznavati programsko kodo v večjih kosih. V tem primeru bi bil en podatek, da gre za običajno zanko `for`, z običajno sintakso. Drugi podatek pa bi bil število 50, kot zgornja meja izvajanja zanke.

Bloki lahko zmanjšujejo kognitivno obremenitev novih programerjev, saj jim pomagajo pri branju oziroma prepoznavanju večjih kosov programske kode.

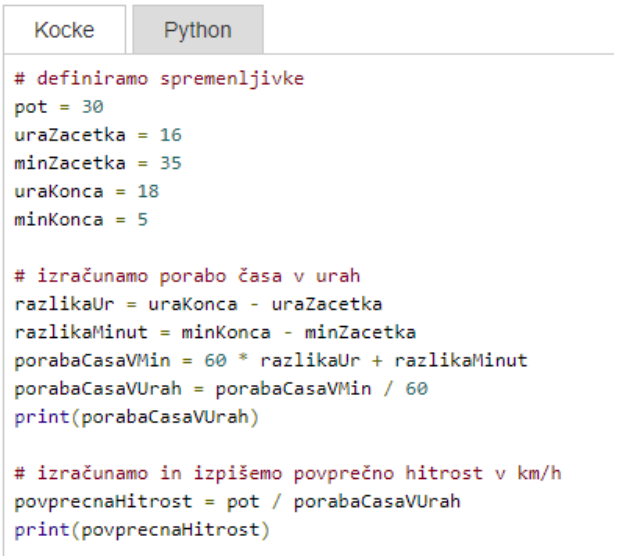
3. Vsi programerji so nagnjeni k delanju napak, ko sestavljajo oziroma pišejo programsko kodo. Bloki pomagajo novim programerjem pri zmanjševanju napak, saj omogočajo sestavljanje le sintaktično pravilne kode (na primer, dveh nezdružljivih konceptov ne moremo povezati, saj so bloki narejeni tako, da jih ne moremo sestaviti skupaj).

Tako se lahko dijaki posvetijo razumevanju problema, ki ga želijo rešiti oziroma želijo napisati program, ki bo rešil problem. Kot pravijo Bau in drugi ([8]), da organizacija programske kode v obliki blokov pomaga novim programerjem, da se ukvarjajo s tem, kaj koda pomeni, namesto da bi se morali ukvarjati s tem, kako kodo pravilno zapisati.

Okolje Blockly omogoča tudi to, da se ob sestavljanju blokov (Slika 2) generira tudi ustrezna tekstovna programska koda (Slika 3) v programskem jeziku Python [23],. Tako lahko novi programerji primerjajo isto kodo, sestavljeno s pomočjo blokov in zapisano v tekstovnem programskem jeziku. Kasneje jim to omogoča lažji in hitrejši prehod na uporabo le tekstovnih programskih jezikov.



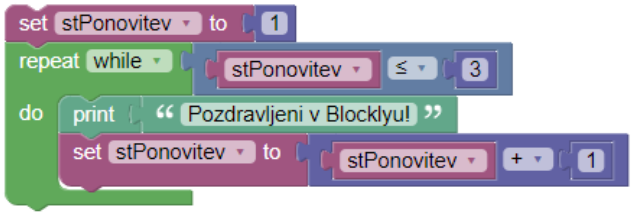
Slika 2: Koda v Blocklyu



Slika 3: Zapis kode v Pythonu

Blockly je knjižnica, okolje oziroma ogrodje, ki omogoča gradnjo slikovnih programskih jezikov. Sam po sebi ni GBOP, saj načeloma ne omogoča krmiljenja figur.

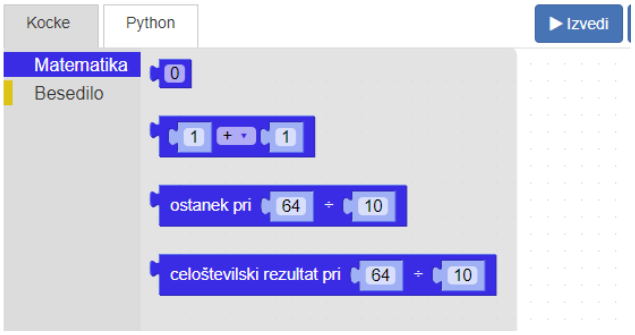
Dostopen je na naslovu https://developers.google.com/blockly/



Slika 4: Program v Blocklyu

Dobrodošla možnost, da v okolju za programiranje (blokovnem vmesniku) enostavno omejimo bloke, ki so na voljo za programiranje (Slika 5)

S tem omogočimo zmanjšanje tuje obremenitve



Slika 5: Uporaba omejenega nabora blokov/kock

Številni raziskovalci se ukvarjajo z vprašanjem o možnostih in morebitnih težavah pri prehodu iz blokovnih jezikov (glej npr.

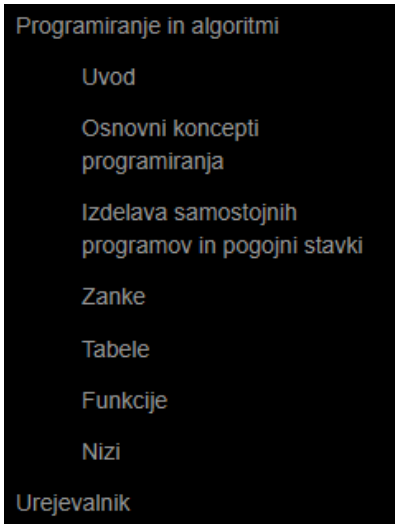
[16]). Kaže, da prijemi, kjer je omogočen sočasen prikaz slikovne in tekstovne kode, zmanjšajo težave, ki nastopajo ob tem prehodu ([1], [9], [10], [13], [15], [16])

3. UČBENIK

Na osnovi prvega dela e-učbenika za informatiko v gimnaziji [24] je nastaja še e-učbenik Slikovno programiranje ([3]) .

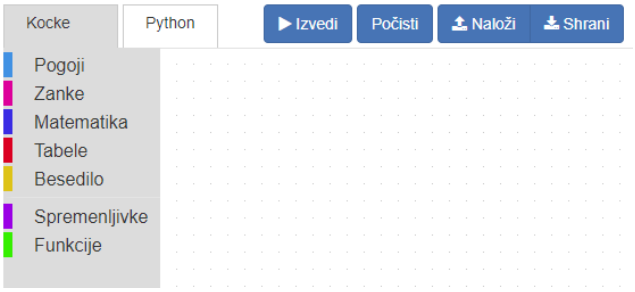
Slika 6: E-učbenik Slikovno programiranje

E-učbenik vsebinsko več ali manj sledi e-učbeniku [24], le da je namesto programskega jezika Pythona uporabljen jezik Blockly. Kot je razvidno iz kazala (Slika 7), učbenik pokriva osnovne koncepte, ki jih srečamo pri začetnem učenju programiranja – zaporedje ukazov, vejitev, zanke, funkcije in osnovne strukture.



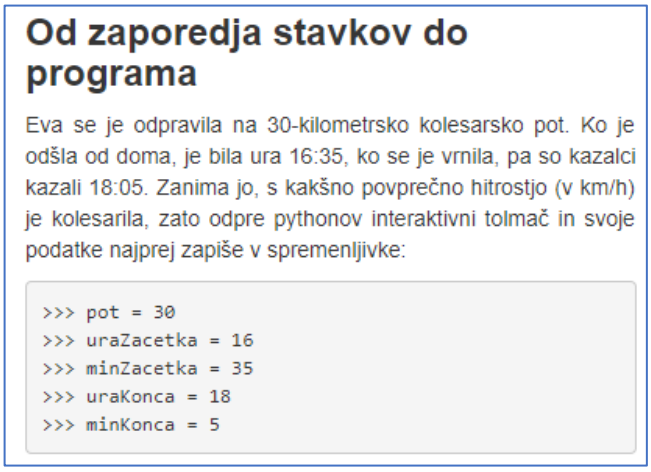
Slika 7: Kazalo e-učbenika Slikovno programiranje

Za uporabnike Blocklya bo koristen tudi spletni urejevalnik, ki omogoča izvajanje, nalaganje in shranjevanje poljubnega programa



Slika 8: Urejevalnik za Blockly

Kot vidimo iz primerjave ustreznih strani iz obeh e-učbenikov na Slika 9 in Slika 10, je v e-učbeniku Slikovno programiranje uporabljeno več ali manj isto besedilo kot v [24] , seveda pa so vsi zgledi predstavljeni v Blocklyu.



Slika 9: Izsek iz e-učbenika za informatiko

Slika 10: Izsek iz učbenika Slikovno programiranje

Zanimiva izkušnja pri nastajanju e-učbenika Slikovno programiranje je ta, da večji posegi v samo besedilo v [24] niso potrebni. Po našem mnenju to priča o kvalitetni zasnovi e-učbenika [24], saj priča, da je v [24] uporabljeni programski jezik zgolj

orodje za ponazoritev določenih programskih konceptov in ni to npr. učbenik za učenje programskega jezika Python.

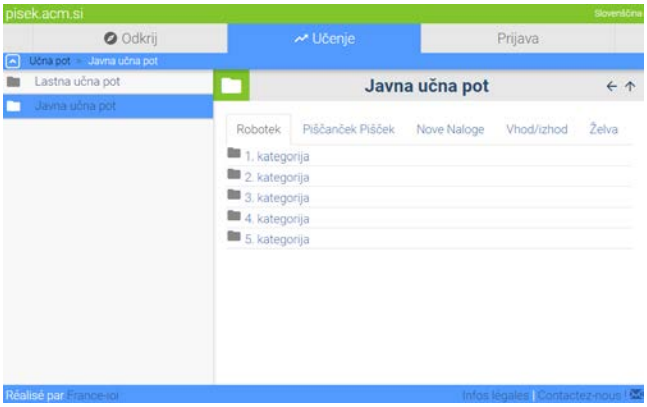
Ob predstavitvi e-učbenika skupini učiteljev v sklopu projekta NAPOJ3 [25] je več učiteljev izjavilo, da bodo v šolskem letu 2018/19 poskusno uporabili Blockly pri poučevanju dijakov, ki se bodo prvič srečali s programiranjem. Zanimivo bo spremljati izkušnje. Predvideno je, da bo o tem veliko govora tudi v sklopu slovenske skupnosti učiteljev računalništva NAPOJ [25].

4. SPLETNI SISTEM PIŠEK

Kot smo omenili v uvodu, je pri učenju zelo koristno, če si učenec pri reševanju nalog lahko pomaga s sistemi za avtomatsko preverjanje programskih rešitev. Za učenje programskega jezika Python obstaja v našem šolskem prostoru spletni sistem za avtomatsko preverjanje Projekt Tomo, ki je bil med učitelji zelo dobro sprejet [18]. Zato smo želeli pripraviti podoben sistem, ki pa bi dovoljeval uporabo slikovnega jezika Blockly. To je bil precejšen izziv, saj v Projektu Tomo ni mogoče preverjati pravilnosti grafičnega izhoda. Ker pa veliko nalog v slikovnih jezikih temelji ravno na grafiki, ni bilo mogoče le nadgraditi Toma z novim jezikom.

Obstajajo številni sistemi za avtomatsko preverjanje, ki podpirajo tekstovne programske jezike. Med tistimi redkimi, ki podpirajo slikovne jezike pa izstopa sistem Algorea, ki so ga zasnovali v francoskem združenju France-IOI (http://www.france-ioi.org). Poleg Francozov ga uporabljajo tudi Nemci. Oboji ga uporabljajo tako za učenje programiranja kot tudi za izvedbo osnovnošolskih in srednješolskih tekmovanj v slikovnih jezikih. Žal pa je za njuno uporabo nujno poznavanje francoskega ali nemškega jezika. Sistema sta dostopna na spletnih naslovih http://concours.castor-informatique.fr in https://wettbewerb.jwinf.de.

Zato smo sistem Algorea pred kratkim za potrebe slovenskih učiteljev priredili, poimenovali Pišek in ga postavili na spletni naslov http://pisek.acm.si/.



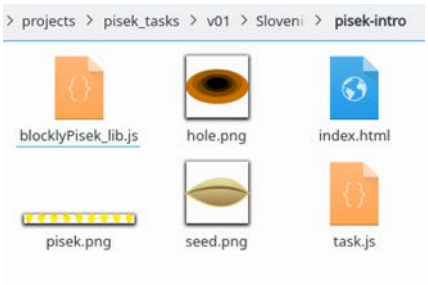
Slika 11: http://pisek.acm.si

Opozoriti velja, da priredba še ni povsem končana. Zato se v določenih delih še pojavljajo neprevedeni izrazi, določeni deli (npr. Odkrij) pa ne delujejo.

Sistem je odprte narave in ga za učenje in testiranje uporablja vsakdo, za sestavljanje svojih nalog pa je treba pridobiti pravice. Sistem uporabnikom s temi pravicami - recimo jim učitelji - omogoča dodajanje novih nalog in stavljenje že obstoječih nalog v sklope. Ti se nato lahko ponudijo uporabniku v testiranje ali pa iz njih pripravi tekmovanje.

Ker je sestavljanje nalog za povprečnega učitelja informatike zahtevno opravilo, smo, kot je vidno iz zgornje zaslonske slike, že pripravili zajeten kupček nalog v slovenščini. Osnovna zbirka nalog je nastala v sklopu študentskega projekta v okviru akcije Študentski inovativni projekti za družbeno korist (ŠIPK) [24] ProNAL. Ker je projekt v teku, se besedila nalog in njihov videz spreminjajo dnevno, prav tako nastajajo nove naloge.

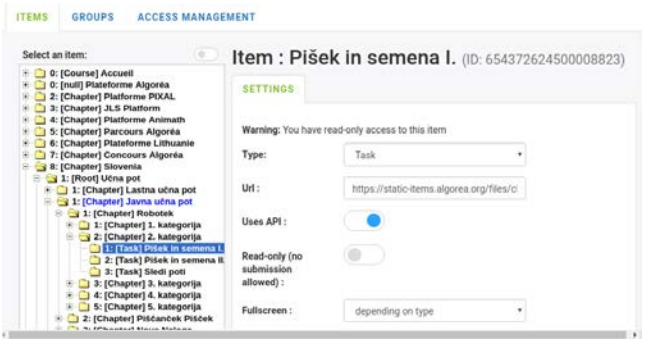
Naloge se lahko uporabijo takšne, kot so, ali pa jih uporabimo kot osnovo za izgradnjo novih nalog. Za sestavljanje novih nalog mora učitelj poznati vsaj osnove programskega jezika Javascript in mora biti vešč dela s HTML datotekami. Kot je razvidno iz Slika 12 so naloge sestavljene iz glavne strani (index.html), gradiv (slike) in programskega dela (task.js) ter knjižnice za delo s programskim jezikom Blockly (blocklyPisek_lib.js).



Slika 12: Sestavni deli naloge

Sestavljavec spreminja le datoteki index.html in task.js. V prvi datoteki zapiše besedilo naloge in nastavi njen videz, v drugi pa v programskem jeziku Javascript opiše programerski del: kateri bloki bodo uporabniku na voljo, kako je videti slika na zaslonu, testni program za preverjanje rešitve … Točno potrebne datoteke so seveda odvisne tudi od tipa problema, ki ga sestavljamo.

Ko je naloga sestavljena, jo uvozimo v sistem in uvrstimo v učne poti. V sistemu imamo v poglavju Slovenia dve učni poti: javno in lastno. V lastno učno pot lahko vsak dodaja naloge zase, naloge pod javno učno potjo pa so vidne vsem. Učne poti so razdeljene na poglavja, ki se lahko gnezdijo, vsako poglavje pa ima eno ali več nalog. Na spodnji sliki lahko vidimo trenutno organizacijo slovenske javne učne poti: razdeljena je na poglavja glede na tip naloge, znotraj poglavja pa so naloge urejene po zahtevnosti. Samo strukturo učne poti, poglavij in nalog lahko sestavljavec poljubno spreminja, kar mu omogoča veliko fleksibilnosti pri izdelavi gradiv.



Slika 13: Datotečni sistem nalog

Že sestavljene naloge so sestavljene na osnovi blokovnega programskega jezika Blockly in so nekaj tipov.

Naloge z robotom oz. piščančkom: piščančka premikamo naokrog po zaslonu s pomočjo ukazov v blokih, pri čemer mora opraviti točno določena opravila. Naloge se po težavnosti stopnjujejo od najbolj preprostih (izvedi nekaj premikov) do zahtevnih, pri katerih mora učenec obvladati koncepte spremenljivk, zank, tabel in celo rekurzije. V zgornjem delu zaslona pri nalogah običajno vidimo

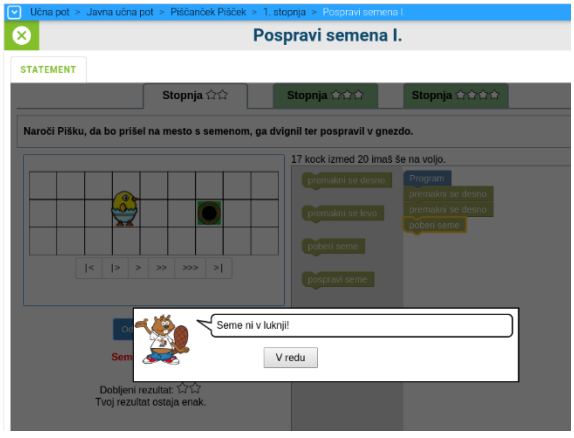
težavnosti naloge, ki so na voljo. Pod njimi sledi navodilo, pod njim pa po vrsti situacija pri nalogi, bloki na voljo in prostor za odlaganje blokov.

Slika 14: Primer naloge

Učenec iz prostora za odlaganje blokov le-te z miško potegne v prostor za odlaganje blokov in jih sestavi v program. Program lahko s pomočjo puščic pod situacijo izvede z različnimi hitrostmi (po korakih, počasi, normalno, hitro). Ko je učenec zadovoljen s programom, ga s klikom na gumb »Oddaj program« odda testnemu programu, ki ga izvede in učencu sporoči rezultat. Program je bodisi sprejet

Slika 15: Uspešno rešena naloga

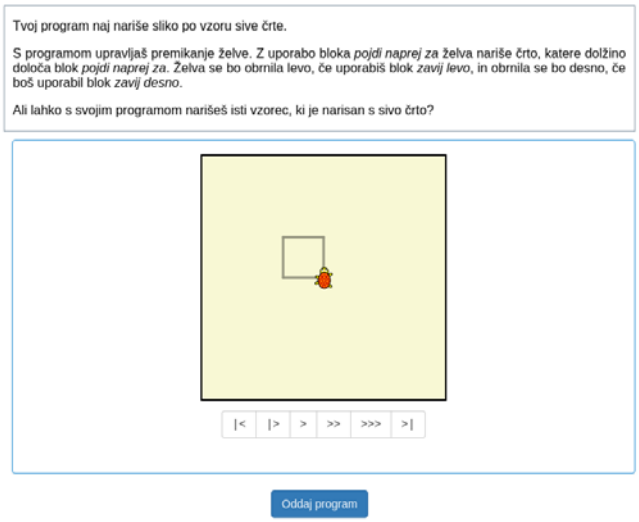
bodisi zavrnjen.



Slika 16: Neuspešno rešena naloga

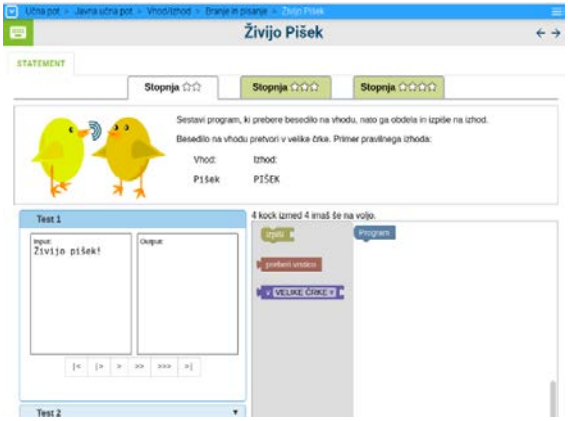
Naloge z logo želvo: verjetno je vsem učiteljem dobro poznan koncept želvje grafike, ki ga je vpeljal Papert s programskim jezikom LOGO. Imamo grafični objekt (želvo), ki ga s pomočjo preprostih ukazov premikamo po zaslonu. Pri tem za sabo (lahko) pušča sled in s tem ustvarja določene grafične vzorce. Tudi v jeziku Blockly lahko delamo naloge z uporabo Logo Želve. Pri tem na enak način kot pri prejšnjem tipu naloge vlečemo bloke na polje za odlaganje blokov in jih sestavimo v program.

V sistemu Pišek je na voljo več po težavnosti urejenih nalog, ki zahtevajo risanje vzorca s pomočjo želvje grafike. Preproste naloge vključujejo risanje preprostih vzorcev, kot je npr. kvadrat, pri težjih vzorcih pa mora učenec uporabiti zanke in spremenljivke.



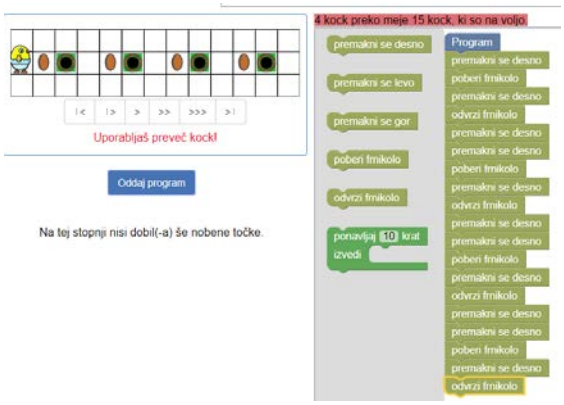
Slika 17: Naloga z želvjo grafiko

Naloge z vhodno – izhodnimi podatki: pri nalogah tega tipa mora učenec prebrati neko besedilo na vhodu, ga obdelati in v pravilni obliki izpisati na izhod. Tovrstne naloge so praviloma zahtevnejše, saj zelo hitro zahtevajo uporabo spremenljivk.



Slika 18: Naloga z branjem/izpisovanjem

Nalogo lahko dodatno otežimo s tem, da omejimo število blokov/kock, ki jih pri programiranju lahko uporabi učenec. Tako program na Slika 19 sicer uspešno reši problem, a porabi preveč kock. Zato rešitev ni sprejeta.



Slika 19: Omejitev števila uporabljenih kock

V nadaljevanju projekta nameravamo sistem Pišek in e-učbenik Slikovno programiranje povezati na podoben način, kot sta povezana E-učbenik za informatiko v gimnaziji [24] in sistem Projekt Tomo [19]. Ena od učnih poti v Pišku bo vsebovala vse naloge iz e-učbenika, v samem e-učbeniku pa bodo povezave do ustreznih nalog v Pišku.

Glavni poudarek pri aktivnostih, ki potekajo, je v sam sistem dodati kar se da veliko ustreznih nalog. V nadaljevanju pa bi sistem radi približali sistemu Projekt Tomo predvsem glede možnosti učiteljevega spremljanja učenčevega napredka in možnosti, da vsak učitelj sam oblikuje izbor nalog, ki jih bo ponudil učencem.

5. VTISI UPORABNIKOV

Kot smo omenili, se tako sistem Pišek, kot e-učbenik se še intenzivno razvijata. Vseeno pa smo zbrali nekaj odzivov tistih, ki smo jim ob različnih priložnostih predstavili oba projekta.

Tako je Ana Cencelj z OŠ Griže zapisala *"Učbenik Slikovno programiranje mi je zelo všeč. Zagotovo ga bom uporabila pri pouku NIP Računalništvo in krožku. Dva možna pogleda kode (kocke in Python) omogočata nadgradnjo znanja. Vključeni primeri so zanimivi in vsakodnevni, tako da jih učenci razumejo. Nabor nalog je dovolj velik za diferenciacijo pouka.*"

Svetovalec za računalništvo iz Zavoda za šolstvo Radovan Krajnc meni, da "*Učbenik podaja snov, ki bi jo moral v naši tehnološko napredni (informacijski) družbi predelati in razumeti vsak odrasel posameznik s srednješolsko izobrazbo.*" in "*Učitelji v osnovni šoli bi lahko učbenik uporabljali pri delu z nadarjenimi učenci (ki bi si izbrali področje računalništva kot področje, ki ga želijo razvijati) ter pri izbirnem predmetu računalništvo pri posameznih temah, ki bi jih predelovali z učenci.*" ter "*Pri informatiki v gimnaziji je uporaben cel učbenik, sploh če bi se učitelj odločil, da bo tej temi namenil kakšnih 30 ur. Prihodnost učbenika je precej odvisna od tega, kakšen status bo pridobilo računalništvo v slovenskem šolstvu. Menim, da prav s tem učbenikom lahko argumentiramo in prepričujemo odgovorne, da smo že zreli in pripravljeni na spremembe.*" Glede sistema Pišek pa je napisal "*… bo ponujal učiteljem odlično orodje za sistematično in načrtno razvijanje znanj s področja algoritmov, programiranja in reševanja problemov učencev. Ne vem, koliko bo imel učitelj vpogled v lastne poti učencev, vsekakor pa bodo lahko učenci z ustrezno podporo učitelja spremljali svoj razvoj in razumevanje osnovnih pojmov s tega področja. Uporabnost Piška bo tudi v tem, da bo nekako »nakazoval«, kaj je tisto minimalno znanje, ki bi ga naj učenci imeli. V kombinaciji z ostalimi orodji in igračami (fizično računalništvo) bo olajšano učenje računalništva, vsekakor pa bo treba učiteljem nuditi podporo pri rabi teh orodij. Zaradi izgleda in velike nazornosti je sistem primeren za vse učence (tudi za učence prvega vzgojno izobraževalne obdobja) v osnovni šoli.*"

Marko Kikelj, profesor Informatike na Gimnaziji Jesenice, pravi "*V prvi letnik gimnazije večina dijakov pride brez predznanja reševanja računalniških problemov. Če jih takoj zasujem s tekstovnim programiranjem, se pojavijo težave s sintakso programskega jezika. S tem smo se oddaljili od osnovnega cilja: naučiti dijake računalniškega razmišljanja.*

Slikovno programiranje mi pomaga prebroditi te težave. Zato pozdravljam tako učbenik Slikovno programiranje kot projekt Pišek. Uporabljal bom oba. Učbenik bodo dijaki uporabljali predvsem doma za utrjevanje znanja. Naloge iz Piška pa bodo reševali pri vajah v šoli."

Profesorica Informatike ene slovenskih gimnazij pravi "*Učbenik bom uporabljala v prvem letniku, ker je primeren za uvodne ure programiranja, sistem Pišek pa bom preizkusila v drugem letniku za ponovitev in osvežitev znanja iz prvega letnika.*", spet druga, prav tako iz gimnazije pa "*Pišek se mi zdi zanimiv za razlago zank in funkcij in ga bom uporabljala pri izbirni uri informatike. Blockly pa bom preizkusila v 1. letniku namesto Scratcha pri eni skupini. Zanima me, kakšen bo odziv ...*"

6. ZAKLJUČEK

Tovrstna orodja in gradiva bodo, kot kaže, v prihodnosti nujna, saj je tudi v zavest širše javnosti prišlo spoznanje, da je vsaj osnovno znanje programiranja dandanes izjemno koristno. Dobra in čim bolj raznolika orodja nam pomagajo pri tem, da je učenje programiranja zanimivo in dosegljivo za kar najširši krog. Kot kažejo izkušnje iz tujine, je trenutno glavna težava pri širši vpeljavi učenja programiranja ta, da primanjkuje dobrih učiteljev programiranja. Seveda pa noben e-učbenik s še večjo interaktivnostjo in noben sistem za avtomatsko preverjanje ne more nadomestiti učitelja, lahko pa mu pomaga, da se laže sooči z večjimi skupinami.

7. VIRI

[1] O. Meerbaum-Salant, M. Armoni in M. (. Ben-Ari, „Learning computer science concepts with scratch,“ v *Proceedings of the Sixth international workshop on Computing education research (ICER '10)*, New York, 2010.

[2] D. Weintrop, A. K. Hansen, D. B. Harlow in D. Franklin, „Starting from Scratch: Outcomes of Early Computer Science Learning Experience,“ v *Proceedings ICER'18*, Espoo, Finland, 2018.

[3] G. Anželj, J. Brank, A. Brodnik, L. Fürst in M. Lokar, „Slikovno programiranje; E-učbenik za uvod v programiranje,“ 2018. [Elektronski]. Available: https://lusy.fri.uni-lj.si/ucbenik/prog/index.html. [Poskus dostopa 27 8 2018].

[4] P. Chandler in J. Sweller, „Cognitive Load While Learning to Use a Computer Program,“ *Cognitive Psychology*, 1996.

[5] G. Wilson (ed.), „Teaching Tech Together - Cognitive Load,“ Lulu.com, 2018. [Elektronski]. Available: http://teachtogether.tech/en/load/.

[6] M. Yousoof, M. Sapiyan in K. Kamaluddin, „Measuring Cognitive Load - A Solution to Ease Learning of

Programming,“ v *Proceedings of World Academy of Science Engineering and Technology* , 2007.

[7] B. J. Ericson, J. D. Foley and J. Rick, “Evaluating the Efficiency and Effectiveness of Adaptive Parsons Problems,” in *Proceeding ICER ’18*, Espoo, Finland, 2018.

[8] D. Bau, J. Gray, C. Kelleher, J. Sheldon in F. Turbak, „Learnable programming: blocks and beyond,“ *Commun. ACM* , Izv. 60, št. 6, pp. 72-80, 2017.

[9] S. Papadakis, M. Kalogiannakis, V. Orfanakis in N. Zaranis, „The Appropriateness of Scratch and App Inventor as Educational Environments for Teaching Introductory Programming in Primary and Secondary Education,“ *Int. J. Web-Based Learn. Teach. Technol.*, Izv. 12, št. 4, pp. 58-77, 2017.

[10] M. Guzdial, „Teaching Two Programming Languages in the First CS Course,“ Blog@CACM, 22 May 2018. [Elektronski]. Available: https://cacm.acm.org/blogs/blog-cacm/228006-teaching-two-programming-languages-in-the-first-cs-course/fulltext#.

[11] C. D. Hundhausen, S. F. Farley in B. J. L., „Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study.,“ *ACM Trans. Comput.-Hum. Interact.*, 2009.

[12] S. Grover in S. Basu, „Measuring student learning in introductory block-based programming: Examining misconceptions of loops, In Variables, and Boolean Logic,“ *ACM Press*, 2017.

[13] S. Grover, R. Pea in S. Cooper, „Designing for deeper learning in a blended computer science course for middle school students,“ *Comput. Sci. Educ.*, Izv. 25, št. 2, pp. 199-237, 2015.

[14] D. Weintrop, „Modality Matters: Understanding the Effects of Programming Language Representation in High School Computer Science Classrooms (Dissertation),“ NORTHWESTERN UNIVERSITY, 2016.

[15] D. Weintrop in U. Wilensky, „Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms,“ *ACM Trans. Comput. Educ.*, Izv. 18, št. 1, pp. 3:1--3:25, 2017.

[16] D. Weintrop, H. Killen in B. Franke, „Blocks or Text? How Programming Language Modality Makes a Difference in Assessing Underrepresented Populations,“ v *ICLS 2018*, London, 2018.

[17] A. T. Corbett in J. R. Anderson, „Locus of feedback control in computer-based tutoring: impact on learning rate, achievement and attitudes,“ v *In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '01).* , New York, 2001.

[18] G. Jerše in M. Lokar, „Uporaba sistema za avtomatsko preverjanje nalog Projekt Tomo pri učenju programiranja,“ v *Vzgoja in izobraževanje v informacijski družbi - VIVID 2017 : zbornik referatov*, Ljubljana, 2018.

[19] UL FMF, „Projekt Tomo,“ 2010 - 2018. [Elektronski]. Available: https://www.projekt-tomo.si.

[20] M. Pretnar, „GitHub - Projekt Tomo,“ 2017. [Elektronski]. Available: https://github.com/matijapretnar/projekt-tomo.

[21] M. Pretnar in M. Lokar, „A Low Overhead Automated Service for teaching Programming,“ v *Proceedings of the 15th Koli Calling International Conference on Computing Education Research*, Koli, Finland, 2015.

[22] RINOS, „Snovalci digitalne prihodnosti ali le uporabniki?,“ 2018. [Elektronski]. Available: https://fri.uni-lj.si/sl/novice/novica/uporabniki-ali-snovalci-digitalne-prihodnosti.

[23] N. Fraser, „Ten things we've learned from Blockly,“ v *Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond) (BLOCKS AND BEYOND '15)*, Washington, 2015.

[24] G. Anželj, J. Brank, A. Brodnik, P. Bulić, M. Ciglarič, M. Đukić, L. Fürst, M. Kikelj, A. Krapež, H. Medvešek, N. Mori, M. Pančur in P. Sterle, Računalništvo in informatika v2.11; E-učbenik za informatiko v gimnaziji, 2017.

[25] A. Brodnik, R. Capuder in M. Lokar, „NAPOJ-3 - MU: Gradnja skupnosti,“ 2018. [Elektronski]. Available: https://moodle.lusy.fri.uni-lj.si/course/view.php?id=59. [Poskus dostopa 27 8 2018].

[26] Javni sklad Republike Slovenije za razvoj kadrov in štipendije, „Študentski inovativni projekti za družbeno korist (ŠIPK),“ 2018. [Elektronski]. Available: http://www.sklad-kadri.si/si/razvoj-kadrov/studentski-inovativni-projekti-za-druzbeno-korist-sipk/.