Poučavanje početnog programiranja oblikovanjem računalnih igara

Mladenović, Monika

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

Monika Mladenović

POUČAVANJE POČETNOG PROGRAMIRANJA OBLIKOVANJEM RAČUNALNIH IGARA

DOKTORSKI RAD

Split, 2019.

PRIRODOSLOVNO-MATEMATIČKI FAKULTET

Monika Mladenović

POUČAVANJE POČETNOG PROGRAMIRANJA OBLIKOVANJEM RAČUNALNIH IGARA

DOKTORSKI RAD

MENTOR: prof. dr. sc. Marko Rosić

Split, 2019.

FACULTY OF SCIENCE

Monika Mladenović

TEACHING PROGRAMMING FOR NOVICES BY DESIGNING COMPUTER GAMES

DOCTORIAL THESIS

SUPERVISOR: professor Marko Rosić, Ph. D.

Split, 2019.

ZAHVALE

Prije svih, iskreno se zahvaljujem svom mentoru, prof. dr .sc. Marku Rosiću na ukazanom povjerenju i podršci tijekom izrade ovog rada.

Hvala članovima povjerenstva: izv. prof. dr. sc. Krešimiru Pavlini, izv. prof. dr. sc. Ivici Boljatu, doc. dr. sc. Nikoli Maranguniću, izv. prof. dr. sc. Branku Žitku i doc. dr. sc. Hrvoju Kaliniću na svim komentarima, vremenu i trudu potrošenom na vrednovanju ovog rada.

Veliko hvala mojim dragim kolegicama i kolegi: Žani Žanko, prof., Mili Ozretić, prof. i Tomislavu Grubišiću, prof. koji su pomogli te rado sudjelovali u istraživanjima. Zahvaljujem i svim učenicima koji su sudjelovali u istraživanjima.

Posebno se zahvaljujem dragoj kolegici i suputnici na cijelom ovom putovanju Žani Žanko. Draga moja ZZ, iskreno se zahvaljujem na zajedničkom vremenu, podršci, suradnji, dugim i konstruktivnim raspravama, razgovorima… Bez tebe bi sve ovo bilo nezamislivo!

Za kraj, najveće hvala mojoj obitelji na stalnoj i bezrezervnoj podršci, razumijevanju, strpljenju… Dragi moji Saša, Klara i Nora beskrajno hvala na svemu!

TEMELJNA DOKUMENTACIJSKA KARTICA

Sveučilište u Splitu Doktorska disertacija
Prirodoslovno-matematički fakultet
Poslijediplomski sveučilišni studij
„Istraživanje u edukaciji u području prirodnih i tehničkih znanosti“

POUČAVANJE POČETNOG PROGRAMIRANJA OBLIKOVANJEM RAČUNALNIH IGARA

Monika Mladenović

Prirodoslovno-matematički fakultet

Ruđera Boškovića 33, 21 000 Split, Hrvatska

Sažetak

Programiranje je apstraktno i teško, pogotovo za početnike na osnovnoškolskoj razini školovanja. Za „klasično“ poučavanje programiranja uglavnom se koriste tekstualni programski jezici, što dodatno otežava učenje početnicima zbog naglašenih problema sintakse. Osim toga, dodatni problem je kontekst programiranja koji se uglavnom svodi na rješavanje matematičkih problema, što pripadnicima digitalnog doba smanjuje motivaciju. Ublažavanje problema sintakse, ali i pomak konteksta programiranja sa matematičkih problema na npr. oblikovanje igara može se postići korištenjem vizualnih programskih jezika primjerenim uzrastu. Prema navedenom postavljena su tri osnovna cilja istraživanja: a) utvrditi utjecaj na poučavanje programiranja, kod početnika u osnovnoj školi, oblikovanjem igara u blokovski orijentiranom, vizualnom programskom jeziku, primjerenom dobi učenika; b) utvrditi utjecaj promjene konteksta učenja s matematičkih problema prema oblikovanju računalnih igara na razumijevanje koncepta petlje i motivaciju učenika u osnovnoj školi; c) osmisliti model poučavanja programiranja početnika u osnovnoj školi uz uporabu alata i zadataka primjerenim dobi učenika. Prema ciljevima osmišljeno je istraživanje u četiri faze koje se provodilo tijekom četiri školske godine u osnovnim školama. Rezultati cjelokupnog istraživanja pokazali su kako učenici postižu bolje rezultate kada se za poučavanje programiranja kod početnika u osnovnoj školi koristi kontekst oblikovanja igara te kada se za programiranje koristi vizualni-blokovski programski jezik primjeren dobi. Također je utvrđeno kako je stav prema programiranju bolji u odnosu na „klasično“ poučavanje programiranja. Rezultati istraživanja mogu biti smjernice za razvoj novih načina poučavanja programiranja početnika u osnovnim školama.

(201 stranica, 38 slika 79 tablica, 160 literaturnih navoda, jezik izvornika: hrvatski). Rad je pohranjen u Sveučilišnoj knjižnici u Splitu, Ruđera Boškovića 31, Split te u Nacionalnoj i sveučilišnoj knjižnici, Ul. Hrvatske bratske zajednice 4, Zagreb.

Ključne riječi: poučavanje programiranja; programski jezici; igre; početnici u programiranju; informatika

Mentor: prof. dr. sc. Marko Rosić

Ocjenjivači:
1. izv. prof. dr. sc. Krešimir Pavlina
2. izv. prof. dr. sc. Ivica Boljat
3. doc. dr. sc. Nikola Marangunić
4. izv. prof. dr. sc. Branko Žitko
5. doc. dr. sc. Hrvoje Kalinić

Rad prihvaćen: 2. srpnja 2019.

BASIC DOCUMENTATION CARD

University of Split Doctoral Thesis

Faculty of Science
Doctoral programme
"Education Research in Natural and Technical Sciences"

TEACHING PROGRAMMING FOR NOVICES BY DESIGNING COMPUTER GAMES

Monika Mladenović

Faculty of Science

Ruđera Boškovića 33, 21 000 Split, Hrvatska

Abstract

Programming is abstract and hard, especially for beginners at the primary level of education. The traditional way of teaching programming is based on using text-based programming languages which further complicates learning for beginners by emphasizing syntax problems. Besides that, additional problem is the programming context which is mostly based on solving math problems, reducing the motivation for members of the digital age. Syntax problems can be reduced, but also context can be shifted from solving math problems towards the design of computer games, by using visual programming languages appropriate for target age.
Per the above, three main goals were set: a) determine the appropriateness and effectiveness of using visual programming language for designing computer games for programming novices in the elementary school; b) determine the effectiveness of the understanding of repeating algorithm by designing games using visual programming languages; c) offer teaching model for programming novices at the elementary school level by using appropriate tools and tasks. According to set goals, the research is organized in four stages during four school years in elementary schools. The results of the revealed that students gain a better understanding of programming concepts by using visual programming languages appropriate for students age compared to the traditional way of teaching programming. Besides, students motivation toward programming is also improved by programming games using visual programming languages. According to the results, the new didactic strategy that will enrich the methodological knowledge will be proposed.

(201 pages, 38 figures 79 tables, 160 references, original in Croatian)
Thesis is deposited in the University Library of Split, Ruđera Boškovića 31, Split and National and University Library, Ul. Hrvatske bratske zajednice 4, Zagreb

Keywords: teaching programming; programming languages; games; programming novices; informatics

Supervisor: professor Marko Rosić, Ph. D.

Reviewers:
1. associate professor Krešimir Pavlina, Ph. D.
2. associate professor Ivica Boljat, Ph. D.
3. assistant professor Nikola Marangunić, Ph. D.
4. associate professor Branko Žitko, Ph. D.
5. assistant professor Hrvoje Kalinić, Ph. D.

Thesis accepted: 2nd July 2019

SADRŽAJ

POPIS SLIKA I TABLICA

Popis slika

Popis tablica

1 UVOD

Glavna značajka poslova budućnosti bit će, (i već jest), visoka uporaba tehnologije, rješavanje problema i kompleksne komunikacije (Levy & Murnane, 2005). Poveznica svega spomenutog je računalna znanost i programiranje kao njezin dio. Programiranje je način rješavanja problema pomoću računala koje se može prenijeti na područja izvan granica same računalne znanosti (Hassinen & Mäyrä, 2006a), stoga bi djeca trebala imati priliku učiti programirati, ne nužno kako bi postali programeri, već kako bi vježbali rješavanje problema (Casey, 1997).

Učenje programiranja postalo je popularno krajem 60-ih i u 70-im godinama prošlog stoljeća u „eri osobnih računala“ (eng. Personal computers - PC) kada su računala postala dostupnija svima. Dostupnost računala otvorila je cijeli novi svijet koji je do tada nije bio poznat većini ljudi. Programiranje je teško, pogotovo kada se govori o prvim programskim jezicima. Programski jezici, kao i strani jezici, imaju svoju sintaksu koja mora biti potpuno točna kako bi program ispravno radio. Problem programiranja i programskih jezika nije vezan samo uz sintaksu već i uz semantiku. Kako bi program ispravno radio potrebno je prvo osmisliti algoritam pa ga tek onda „prevesti“ na programski jezik. U odnosu na samu sintaksu jezika, zahtjevniji su osmišljavanje i izrada algoritma. Dakle, problem početnika u programiranju je dvojak, prvi problem predstavlja samo rješavanje problema, odnosno izrada algoritma, a drugi problem svodi se na ispravno pisanje programa u nekom programskom jeziku. Algoritamsko razmišljanje zahtijeva apstrakciju, što može predstavljati problem kod početnika, pogotovo kod djece osnovnoškolske dobi, kod kojih se apstraktno razmišljanje, u većini slučajeva, još nije razvilo. Postoje istraživanja koja potvrđuju kako je za programiranje, kao i za rješavanje problema, potrebna visoka razina apstraktnog razmišljanja (Papert, 1980; White & Sivitanides, 2003a), ali i ona koja potvrđuju kako se programiranjem, odnosno rješavanjem problema pomoću računala, može razvijati apstraktno mišljenje (Bubica, Mladenović, & Boljat, 2013; Fessakis, Gouli, & Mavroudi, 2013; Hassinen & Mäyrä, 2006b; Liao & Bright, 1991; Papert, 1980) što je naizgled paradoksalno. Obje su izjave točne, ali visoka razina apstraktnog razmišljanja nije jedini preduvjet za uspjeh u programiranju kao što ni niža razina apstraktnog razmišljanja ne mora biti ograničavajući faktor kako bi postali programeri. Motivacija može imati ključnu ulogu u programiranju, a značajan utjecaj na motivaciju može imati kontekst programiranja. Programiranje se još uvijek uglavnom poučava na „tradicionalan“ način na koji se poučavalo u samim počecima, kad se računalno programiranje usko vezivalo uz rješavanje matematičkih problema. Danas, kad su učenicima računala sastavni dio života, rješavanje matematičkih problema učenicima je uglavnom odbojno. Samim time motivacija za

programiranjem opada što se očituje u padu zanimanja za programiranje u cijelom svijetu (Denning & McGettrick, 2005; Robins, Rountree, & Rountree, 2003b; Uludag, Karakus, & Turner, 2011). Neka su istraživanja pokazala kako čimbenici poput motivacije i zanimanja za područje utječu na učinkovitost pri učenju programiranja (Zainal et al., 2012). Neuspjehom pri povezivanju programiranja, odnosno računalne znanosti, s interesima učenika te tradicionalni pristup poučavanju programiranja očito ne uspijeva motivirati učenike za odabir zanimanja iz područja računalne znanosti (Forte & Guzdial, 2005; Uludag et al., 2011). Nameće se pitanje: Koji bi kontekst programiranja učenike motivirao za programiranje? Jedan od potencijalnih kandidata je sigurno programiranje računalnih igara.

Iz svega navedenog, nameću se ključna pitanja kada se govori o početnicima u učenju programiranja: i) Na kojoj razini školovanja početi učiti programiranje?; ii) Koji programski jezik odabrati za početno poučavanje programiranja?; iii) Koji kontekst programiranja odabrati?

Cilj je ovog istraživanja utvrditi kako na kognitivnu i afektivnu domenu utječe primjena korištenja vizualnih blokovskih programskih jezika u kontekstu programiranja igara u odnosu na „tradicionalan“ pristup koji uključuje programiranje u tekstualnim programskim jezicima u kontekstu rješavanja matematičkih problema. Temeljem rezultata predložit će se model poučavanja početnika u osnovnoj školi korištenjem odgovarajućeg pristupa koji je u skladu s kognitivnim razvojem učenika.

2 TEORIJSKI OKVIR

2.1 Status informatike i programiranja u društvu

Brojna istraživanja pokazuju kako je u prošlom i ovom desetljeću prisutan trend opadanja zanimanja povezanih s učenjem programiranja i računalne znanosti općenito, što se također čini paradoksalno. Čini se kako je postojalo puno veće zanimanje za učenje programiranja u 1970-im godinama, kada računala nisu bila dostupna svima, nego u današnje vrijeme u kojem nije moguće zamisliti svakodnevni život bez računala. Zašto je to tako? Za navedeno postoji nekoliko vjerojatnih razloga:

(i) Jedan od razloga sigurno je izostanak uvođenja nastave Informatike (naziv uvriježen u Europi dok se u SAD-u naziva Računalnom znanosti) kao obaveznog dijela osnovnoškolskih kurikula, što vrijedi za većinu Europskih zemalja i SAD. U SAD-u je još 2003. godine donesen K-12 kurikul (Jones, Mccowan, & Stephenson, 2003) sa smjernicama uvođenja koji zapravo nikada nije proveden. U Europi je situacija slična, ali nije identična u svim zemljama. Izvješća ACM-a, kao krovne organizacije računalnog obrazovanja u svijetu, upozoravaju kako se propušta prilika za kreiranjem računalno obrazovanih građana te kako umjesto toga dobivamo samo konzumente računalne tehnologije (Gander et al., 2013). Od samog početka računalnu znanost čini jedinstvena kombinacija matematike, inženjerstva i prirodnih znanosti, svih triju, a ne samo jedne (Denning, 2009b). Iz navedenog se može zaključiti kako računalna znanost, sama za sebe, predstavlja integrirani STEM u samom temelju, sa multidisciplinarnim vezama između područja računarstva i STEM-a (Badawy et al., 2013). U temeljima STEM-a nalazi se rješavanje problema koje se, kao što je već navedeno, može vježbati upravo programiranjem. Istraživanja pokazuju kako učenici oblikuju mišljenje o računalnoj znanosti već u ranoj dobi (Yardi & Bruckman, 2007), te za svoju buduću profesiju najčešće biraju područje s kojim su se susreli do 8. razreda osnovne škole (Tai, 2006), pa je očita šteta koja nastaje propuštanjem uvođenja informatike u osnovnu školu s obzirom da se većina učenika sa računalnom znanosti ne susreće do srednje škole (Maltese & Tai, 2010). Postoje i protivnici takvih stajališta, a to su uglavnom razne grupacije koje ne priznaju informatiku kao samostalnu disciplinu već je vide kao inženjerski dio matematike, tehnike i slično, nazivajući je „znanosti umjetnog“ („science of the artificial“) (Denning, 2013), a računala doživljavaju samo kao alat što pokazuje nerazumijevanje same znanosti. Djikstra je kao jedan od pionira i vodećih autoriteta u području računalne znanosti, dao lijepu usporedbu računala i računalne znanosti kad je izjavio da je računalo za informatičara što i teleskop za astronoma (Denning, 2010). Osim toga Informatika se često izjednačava s programiranjem (Denning, 2009a) što se često koristi kao argument

protivnika uvođenja informatike s pitanjem „Trebaju li svi učiti programirati?“. U debatama oko kurikula Informatike postavljaju se pitanja kao što su: Koja je svrha takvog kurikula: stvoriti digitalne građane, konzumente tehnologije, informatičke profesionalce? Sva ova pitanja vode u začarani krug u kojem Informatika nije redovan predmet dok s druge strane ne možemo ponuditi odgovore na sva ta pitanja dok ne vidimo rezultate uvođenja predmeta (Webb et al., 2017). Činjenica je kako se danas Informatika smatra znanošću, a programiranje je samo jedan od njezinih temeljnih područja koje bi trebalo učiti od najranije dobi (Denning, 2013; Webb et al., 2017).

(ii) Drugi vjerojatan razlog za smanjenje zanimanja povezanih s učenjem programiranja i računalnih znanosti su kontekst i način poučavanja programiranja koji se nisu promijenili od samih početaka još iz 1970-ih. Istraživanja o učenju i poučavanju su doživjela procvat nakon drugog svjetskog rata (industrijsko doba). U novom socijalnom ustroju koji se dogodio nakon rata, obrazovanje i učenje počeli su utjecati na društvo što je prepoznato na svim (društvenim, političkim, sociološkim i dr.) razinama. Svjetovne škole vezane su uz razvoj industrijskog društva. Za industrijski način proizvodnje trebali su radnici koji znaju pisati i računati, dok su sve ostale individualne značajke bile manje važne. Danas živimo u digitalnom dobu u kojem su potrebne drugačije sposobnosti. Od vremena informacijskog doba pomak prema digitalnom dobu se, kako u ekonomiji tako i u obrazovanju, događaju promjene koje su međusobno ovisne (Reigeluth, 2016). Industrija više nije ključna profesija, broj "bijelih ovratnika" nadmašio je broj "plavih ovratnika", obitelj više nije u centru društva, sve je više samohranih roditelja, uvažavaju se razlike među ljudima. Što se tiče učenja i poučavanja, došlo je do promjena u teoriji učenja te u razumijevanju inteligencije. Do kraja industrijskog doba dominantni biheviorizam zamijenjen je kognitivizmom pa kasnije konstruktivizmom, a tadašnje shvaćanje inteligencije temeljeno isključivo na matematičkoj i lingvističkoj inteligenciji promijenilo se uvažavanjem koncepta višestruke inteligencije (Armstrong & Association for Supervision and Curriculum Development., 2009) koja se temelji na prepoznavanju devet vrsta inteligencije kod čovjeka. Sposobnost zaključivanja i matematičke sposobnosti, za koje je zadužena lijeva strana mozga, bile su važnije u prethodnim razdobljiima. U digitalnom dobu postala je važnija kreativnost, sposobnost prilagodbe i vizualizacije za što je zadužena desna strana mozga (Pink, 2007). Globalizacijom tržišta i uzletom velikih korporacija došlo je do promjena u društvu koje zahtijevaju promjene u obrazovanju. Daljnjim napretkom tehnologije i virtualizacijom proizvoda i usluga porastao je značaj "intelektualnog kapitala" u stvaranju i održavanju vrijednosti u poslu. Učenje i obuka prepoznati su kao važni faktori za zadržavanje

kompetitivnosti ljudi i organizacija (Fox, 2003). Već sada postoji velika razlika između ishoda učenja i vještina koje su potrebne u okolini koja se ovako brzo mijenja. Današnje škole nisu se značajno promijenile od industrijskog doba, a kao ishodi učenja traže se potpuno druge kompetencije (Prensky, 2010). Glavna značajka sljedeće generacije poslova bit će povećana uporaba tehnologije, rješavanje problema i kompleksne komunikacije (Levy & Murnane, 2012). Primjerice, programer je mogao biti netko tko ima sposobnosti linearnog razmišljanja i izvršavanja rutina, dok se danas od programera očekuje kreativnost (desni mozak), a ne isključivo kreiranje algoritama (lijevi mozak). Očito je kako se nešto mora promijeniti u obrazovanju s obzirom da je sposobnost programiranja u zapadnom svijetu postala samo vještina, te se sada puno više cijene sposobnosti analize, dizajniranja sustava, gledanja cjelokupne slike (eng. *big picture*) i druge kreativne sposobnosti (Pink, 2005). Unatoč preporukama da se u učenju što više koristi lijeva i desna strana mozga većina nastavnih programa računalne znanosti napravljena je za „lijevi" mozak (Tsai, Huang, & Zeng, 2006). Današnje doba je digitalno doba koje je znatno drugačije od dosadašnjih. Prijelazi između prethodnih razdoblja tekli su inkrementalno, dok se pri prijelazu na digitalno doba dogodio diskontinuitet. Uzrok ovakvog diskontinuiteta nalazi se u ubrzanim promjenama digitalnih tehnologija od kraja 20. stoljeća. Današnji učenici predstavljaju prvu generaciju koja je odrasla s novim tehnologijama. Oni su cijeli život okruženi računalima, mobitelima, video igrama, digitalnim glazbenim uređajima, video kamerama, i drugim uređajima i igračkama digitalnog doba. Digitalni urođenici preferiraju učenje kroz igru u odnosu na „ozbiljan rad" (Prensky, 2001). Pojavom digitalnog doba pojavljuje se i pojam informatičke ili digitalne pismenosti kao sposobnosti korištenja računala i različitih računalnih alata. Informatička pismenost (Office of Technology Assessment, 1984) danas predstavlja minimum pojmova koje bi trebalo poznavati, a većina današnjih poslova zahtjeva ne samo informatičku pismenost već i *informatičku okretnost* (Council & others, 1999) koja podrazumijeva ne samu uporabu različitih informatičkih alata već i sposobnost sustavnog i kreativnog razmišljanja potpomognutog tehnologijom. S obzirom na sve navedeno kao i na sveprisutnost računalne znanosti u životu, informatika i računalno razmišljanje su vještine koje bi svi trebali savladati. Računalno razmišljanje bila bi četvrta „analitička sposobnost", uz čitanje, pisanje i aritmetiku, koja bi za učenje trebala biti dostupna svima (Wing, 2008), a programiranje bi trebala biti vještina koju bi svi trebali usvojiti (Uludag et al., 2011)(Perlis, 1962). Unatoč navedenom, u svijetu postoji trend opadanja interesa za učenjem računalne znanosti (Uludag et al., 2011) (Denning & McGettrick, 2005) dok istodobno raste potražnja za takvim poslovima (Litecky, Prabhakar, & Arnett, 2006)(Gupta & Houtz, 2000). Zbog nerazmjera ponude i potražnje u području računalne

znanosti jasno je kako su nužne promjene. Razlog pojave nerazmjera možda se krije u činjenici što su današnja djeca digitalni urođenici, a način poučavanja računalne znanosti i programiranja se nije znatno promijenio, odnosno nije prilagođen takvoj populaciji. U ovom radu obrađuje se poučavanje programiranja kroz igru te kroz programiranje igara kao načinima prilagodbe učenju digitalnih urođenika.

(iii) Treći vjerojatni razlog za smanjenje zanimanja povezanih sa učenjem programiranja je i odabir programskog jezika za početno učenje. Prvi jezici za poučavanje bili su LOGO i BASIC, nastali još početkom 1960-ih godina, koji se još uvijek u Republici Hrvatskoj koriste kao službeni jezici za poučavanje u osnovnim školama. Ti su jezici služili svrsi, ali u današnje vrijeme, za današnje učenike više nisu primjereni. Postoji nova generacija jezika izrađena upravo za početnike, a temelje se na ideji mini-jezika. Takvi jezici trebali bi biti što jednostavniji kako bi omogućili početnicima što lakše učenje programiranja, te bi se po mogućnosti trebali temeljiti na mikrosvjetovima, što je bila osnovna filozofija Papertovog programskog jezika Logo. Upravo na toj filozofiji temelji se nova generacija vizualnih programskih jezika osmišljenih za učenje programiranja. Ideja tih jezika je prije svega umanjiti ili potpuno eliminirati problem sintakse jezika koja je često izvor frustracije, pogotovo kod početnika. Kako je već spomenuto, prvi korak izrade programa je izrada algoritma te je ideja bila izraditi programske jezike za početnike i to od najranije dobi, u kojima će naglasak biti upravo na rješavanju problema, odnosno na semantici umjesto na sintaksi. Osim toga, programiranje se još uvijek percipira kao „muško“ zanimanje, tim više kad se uzme u obzir kako se programiranje još uvijek uglavnom poučava na „tradicionalan“ način, na koji se poučavalo u samim počecima poučavanja programiranja, koji se uglavnom odnosi na rješavanje matematičkih problema u proceduralnom tekstualnom programskom jeziku. Novim vizualnim programskim jezicima, osim što se naglasak stavlja na semantiku umjesto na sintaksu, pomiče se i kontekst programiranja prema programiranju igara, priča, i slično, što se pokazalo podjednako uspješno kod dječaka i djevojčica (Kelleher, Pausch, & Kiesler, 2007). Primjeri takvih jezika su Scratch, Alice i Greenfoot koji se prije svega razlikuju prema ciljanoj populaciji, pa je tako Scratch primjeren za djecu od 7 do 16 godina što odgovara osnovnoj školi, Alice je primjeren za uzrast srednje škole, a Greenfoot za preddiplomsku razinu studija. U ovom radu su provedena istraživanja u osnovnim školama te je Scratch bio najprimjereniji za tu svrhu.

2.2 Informatika i programiranje na osnovnoškolskoj razini obrazovanja u Republici Hrvatskoj

U Republici Hrvatskoj je informatika do školske godine 2018./2019. bila izborni predmet u osnovnoj školi od 5. do 8. razreda te obavezan predmet u barem jednom razredu srednje škole, ovisno o školi. Nastavni plan i program za predmet Informatika (Ministry of science education and Sports of the Republic of Croatia, 2005) je prema Hrvatskom nacionalnom obrazovnom standardu (HNOS) bio važeći od 2006. do 2018. godine. U tom se periodu Informatika kao izborni predmet sigurno nije mogao provoditi u mjeri u kojoj bi se provodio kao redovni predmet jer se svake godine učenik može upisati ili ispisati s izbornog predmeta. Istraživanje provedeno na 1462 učenika osmih razreda osnovne škole u Republici Hrvatskoj pokazalo je kako su učenici uglavnom ekstrinzično motivirani za upis izbornog predmeta Informatika i to radi igranja računalnih igara i dobivanja dodatnih bodova za upis u srednju školu. Osim toga, pokazalo se kako su učenici iz zagrebačke regije više intrinzično motivirani u odnosu na riječku i dalmatinsku regiju, te da su učenici iz dalmatinske regije više socijalno motivirani za upis izbornog predmeta informatike (S. Mladenović, Žanko, & Mladenović, 2015). Što se tiče motivacije za učenje predmeta Informatika i programiranja kao dio kurikula pokazalo se kako su dječaci više motivirani od djevojčica (M. Mladenović, Žanko, & Mladenović, 2014; Žanko, Mladenović, & Mladenović, 2014), što je u skladu sa svjetskim trendom poimanja programiranja kao „muškog“ zanimanja.

Kao posljedica provedbe predmeta Informatika kao izbornog u osnovnim školama, nastao je problem provedbe predmeta Informatika u srednjoj školi, jer se program izvodi s pretpostavkom učenika početnika informatike što nije slučaj kod svih učenika, pa se tako u istoj grupi nalaze učenici koji možda već 4 godine uče informatiku i oni kojima je to prvi doticaj sa računalom. Očita je heterogenost takvih grupa po znanju, ali i sposobnostima, pa će učenicima s predznanjem gradivo biti lagano dok će onim pravim početnicima biti teško.

Od školske godine 2018/2019 predmet Informatika uveden je kao obavezan predmet za pete i šeste razrede osnovne škole prema novom kurikulu, prema kojem je „Računalno razmišljanje i programiranje“ jedna od četiri osnovne domene (Ministrarstvo znanosti i obrazovanja, 2018). Kako je ovo prva godina provođenja obaveznog predmeta Informatika još se ne može govoriti o rezultatima uvođenja. Problemi vezani uz provedbu obaveznog predmeta Informatika su prije svega tehnička opremljenost škola i veličine grupa. Naime, nisu sve škole podjednako tehnički opremljene, a grupe učenika nisu ograničene tehničkim mogućnostima škole. Iz toga razloga može nastati više problema. Jedan od njih je mogućnost dijeljenja računala između više učenika

za vrijeme sata Informatike, što nije poželjno. Drugi, vjerojatno i veći problem, je što u grupi može biti od 14 do 28 učenika što za praktičnu provedbu nastave Informatike također nije poželjno. Dakle, novonastali problemi Informatike kao obaveznog predmeta su tehnička ograničenja, velike grupe učenika te razlike u motivaciji i sposobnostima učenika, što je posebno naglašeno u cjelini Računalno razmišljanje i programiranje.

Istraživanje u ovoj disertaciji provedeno je za vrijeme dok se Informatika provodila kao izborni predmet prema nastavnom planu i programu HNOS (Ministry of science education and Sports of the Republic of Croatia, 2005) u kojem je programiranje bilo jedna od deset osnovnih cjelina predmeta Informatika koji se provodio svake godine od petog do osmog razreda prema spiralnom kurikulu (Bruner, 1960). Provedba cjeline Programiranje uključuje mogućnost odabira učitelja između dvaju pristupa, A ili B . Pristup A oslanja se na kornjačinu grafiku, dok se Pristup B oslanja na proceduralno programiranje. Teme za svaki od pristupa su prikazane u tablici 2.1.

Tablica 2.1 - Teme cjeline programiranja prema HNOS-u

	Teme A (kornjačina grafika)	Teme B (proceduralni jezik)
5. razred	1. Osnovne naredbe programskog jezika 2. Ponavljanje niza naredbi 3. Uporaba petlje za crtanje niza likova 4. Ulazne vrijednosti procedura 5. Uporaba više ulaznih vrijednosti 6. Odluke u programu	1. Pojam algoritma 2. Dijagram tijeka 3. Naredbe za ulaz i izlaz podataka
6. razred	1. Crtanje kocke i kvadra 2. Višestruke kornjače 3. Tipovi podataka: numerički, znakovni, liste 4. Algoritmi koji koriste različite tipove podataka	1. Uporaba naredbi za grananje i bezuvjetni skok 2. Algoritmi s uporabom petlje 3. Uporaba naredbi za petlju bez logičkog uvjeta
7. razred	1. Koordinatna grafika 2. Zadatci o pravilnim mnogokutima 3. Izrada grafičkog sučelja	1. Uporaba naredbe za petlju s logičkim uvjetom 2. Crtanje ravnih linija i pravokutnika 3. Crtanje kružnice
8. razred	1. Grafičke naredbe u programu 2. Procedure i programske funkcije 3. Primjena programiranja u nastavi matematike 4. Primjena programiranja u fizici i kemiji	1. Potprogrami 2. Primjena programiranja u nastavi matematike 3. Primjena programiranja u fizici i kemiji

Iako su teme zadane nisu striktno zadani programski jezici, unatoč tome što se za A pristup uglavnom koristio Logo, za B pristup BASIC, te kasnije Python kao programski jezik izbora.

Prema zadanim temama vidljiv je različit pristup osnovnim algoritmima: slijed, grananje i ponavljanje. Vidljivo je kako se prema pristupu A sva tri osnovna algoritma obrađuju već u petom razredu dok se prema pristupu B u petom razredu navedeni algoritmi obrađuju izvan konteksta programskog jezika (Teme B, 5. razred, 1. pojam algoritma), dok se u programskom jeziku obrađuje samo algoritam slijeda, a grananje i ponavljanje u 6. razredu. Zbog toga, učitelji nisu u mogućnosti svake godine mijenjati pristup već su se uglavnom oslanjali na pristup

odabran u petom razredu. Dodatan problem je izbornost predmeta te se učenici mogu upisati i ispisati svake školske godine pa je postojala mogućnost da se i na višim godinama pojave grupe učenika bez predznanja programiranja, što otežava održanje nastave prema spiralnom kurikulu.

2.3 Programiranje

Programiranje je područje koje je izazov za učenje i poučavanje (Milne & Rowe, 2002). To je područje koje se smatra teškim za učenje, razumijevanje i savladavanje, pogotovo za početnike (Gomes & Mendes, 2007; Robins, Rountree, & Rountree, 2003a; Winslow, 1996), bez obzira na dob (Guzdial, 2004), a većina djece u osnovnim školama ima negativno mišljenje o računalnoj znanosti (Violino, 2009). Programiranje je računalom potpomognuto rješavanje problema, otklanjanje grešaka, razvijanje logičkog i računalnog razmišljanja, što podrazumijeva razvoj strategija za rješavanje problema koji se mogu odnositi i na ne-programerska područja. Stoga se može reći da programiranje mijenja način razmišljanja (Resnick et al., 2009). Učenje programiranja podrazumijeva ne samo učenje novih pojmova već i vježbanje vještine primjene naučenog za rješavanje problema u novim situacijama (Hassinen & Mäyrä, 2006b). Iako se područje poučavanja programiranja aktivno istražuje, najbolja metoda za učenje programiranja još nije pronađena (Hassinen & Mäyrä, 2006b).

Neka istraživanja pokazuju kako faktori poput stava, motivacije i pojačanog interesa za programiranje utječu na uspješnost u učenju programiranja (Zainal et al., 2012). Bez povezivanja programiranja i koncepata računalne znanosti s različitim interesima učenika ostajemo pri klasičnom poučavanju programiranja koje ne motivira većinu učenika i obeshrabruje ih u nastavku školovanja usmjerenog prema računalnoj znanosti (Forte & Guzdial, 2005; Uludag et al., 2011). „Klasični“ alati za programiranje, poput BASIC-a, osim što su vizualno neprivlačni današnjoj djeci, zahtijevaju potpuno sintaktički (i semantički) ispravno napisan program, što je često izvor frustracija i demotivirajući faktor za učenje programiranja. Kako program mora sintaktički biti potpuno ispravan učenici više pažnje pridaju sintaksi nego semantičkom značenju (Lewis, 2010).

Nakon početnog entuzijazma za programiranjem, koji je postojao 1970-ih i 1980-ih s pojavom osobnih računala, danas takvog entuzijazma gotovo da nema. Danas je jedan jednostavan pametni telefon snažniji od svih sveučilišnih računala 1970-ih. Djeca su uključena u računalno okruženje od malih nogu i očekuju interakciju s računalom pa to područje više nije usko vezano samo uz matematiku i matematičke sposobnosti (Wing, 2008). Programiranje se uglavnom doživljava kao tehničku aktivnost primjerenu za manji broj ljudi. Što se dogodilo s entuzijazmom? Postoji nekoliko čimbenika koji su utjecali na pad entuzijazma za programiranje (Resnick et al., 2009):

- Rani alati za programiranje bili su presloženi za korištenje te dosta djece ne bi uspjelo savladati sintaksu
- Programiranje se obično vezivalo uz aktivnosti koje nisu vezane uz interese djece (uglavnom matematički problemi kao što je traženje prostih brojeva i sl.)
- Programiranje se odvijalo na način da se u slučaju pojave pogrešaka teško samostalno našlo rješenje

U sljedećim poglavljima će se opisati svaki od ključnih pojmova vezanih uz programiranje.

2.3.1 Programiranje i apstrakcija

Apstrakcija je temelj matematike, znanosti i inženjerstva uopće, koja igra ključnu ulogu u izradi modela za analizu te u izradi inženjerskih rješenja. Prema Piagetovoj teoriji kognitivnog razvoja postoje četiri faze kognitivnog razvoja čovjeka: senzomotorna (0-2 g.), predoperacijska (2.-7. g), faza konkretnih operacija (7.-11.g) i formalnih operacija (>12 g) (Piaget, 1952). Neka istraživanja pokazuju kako samo 34% adolescenata dolazi do faze formalnih operacija, dok ni kod odraslih taj postotak nije veći (Fusco, 1981), što znači da je većina osnovnoškolaca razredne nastave na konkretnoj ili na pred-formalnoj razini razmišljanja dok su studenti preddiplomskog studija sazreli u logičkom zaključivanju (Sekiya & Yamaguchi, 2013) te su dosegli fazu formalnih operacija. Ovakva razlika upućuje na mogućnost poticanja razvoja apstraktnog mišljenja kod većeg broj učenika prilagođavanjem kurikula učeničkim kognitivnim razinama. Dio odraslih ljudi nikada ne dođe do faze formalnih operacija jer se nisu našli u okruženju koje od njih to zahtijeva, što potiče na propitivanje: Može li se apstrakcija vježbati? (Kramer, 2007) Prema Piagetovim istraživanjima došlo se do zaključka da učenici trebaju konkretna iskustva kako bi razumjeli apstrakciju (Turkle & Papert, 1992).

Piaget je postavio temelje konstruktivizma - teorije učenja. Mnogi psiholozi novijih generacija nastavili su istraživanja na Piagetovim temeljima teorije kognitivnog razvoja pri čemu su došli do novih spoznaja od kojih neke nisu u skladu s Piagetovim. To se prije svega odnosi upravo na teoriju kognitivnog razvoja pri čemu Piaget smatra kako faze kognitivnog razvoja direktno ovise o dobi dok je Bruner došao do drugačije teorije, nastale pod utjecajem Piageta. Prema Bruneru (Bruner, 1966), proces učenja događa se u tri faze: akcijska (temeljena na akcijama), ikonička (temeljena na ikonama-slikama) i simbolička (temeljena na jeziku-sintaksi).

Akcijskom fazom smatra se manipulacija konkretnim objektima, na primjer lego kockicama. Manipulacija slikama objekata može se smatrati alatom za ikoničku fazu učenja. Programski

jezici koji odgovaraju ovoj fazi su blokovski programski jezici jer su naredbe prikazane u obliku slagalica. Osim toga u ovoj fazi vizualizacije i dijagrami mogu pomoći u razumijevanju novih pojmova. Manipulacija reprezentacijom objekata uklapa se u simboličku fazu u kojoj učenik prepoznaje simbole koji predstavljaju neki objekt ili akciju. Programiranje u tekstualnim programskim jezicima, smatra se alatom za ovu fazu (slika 2.1).

Slika 2.1 - Programski jezici prema Brunerovoj reprezentaciji stvarnosti

Iako ovakva podjela podsjeća na faze kognitivnog razvoja, one to nisu. Prema Bruneru početni način razmišljanja se ne napušta, već se može koristi i kasnije, ako se pokaže korisnim u određenim situacijama. U kontekstu programiranja to znači da i u odrasloj dobi ima smisla pri uvođenju novog koncepta programiranja učenike voditi kroz sve tri faze kako bi bolje razumjeli najvišu, simboličku razinu, što je u skladu s principom „od konkretnog prema apstraktnom“. Dob, naravno, ipak ima utjecaj, ali na vrijeme uporabe određenih alata, pa će se kroz akcijsku i ikoničku faza puno brže proći u odrasloj nego u osnovnoškolskoj dobi, u kojoj bi se trebalo duže zadržati upravo na akcijskoj, odnosno ikoničkoj. Osim toga, poučavanje programiranja se najčešće odvija prema spiralnom kurikulu koji je osmislio Bruner. Spiralni kurikul (Bruner, 1960) se odnosi na ideju poučavanja kod koje se osnovna jezgra područja učenja svake godine ponavlja i širi, od jednostavnijih ideja prema složenijima, što je u skladu sa Brunerovom idejom da se i apstraktni koncepti mogu učiti i u ranijoj dobi, te Piagetovoj teoriji akomodacije novog znanja. Kod programiranja se navedeno može odnositi i na uporabu programskih jezika, kod koje zaista nije primjereno početi s tekstualnim programskim jezikom dok se ne usvoje osnovni, pozadinski koncepti.

Sukladno navedenim teorijama, problemi za programere početnike nastaju već u početnoj fazi učenja programiranja kada trebaju razumjeti i primijeniti apstraktne koncepte programiranja,

često prikazane tekstualnim (simboličkim) programskim jezikom. Poučavanje osnovnoškolaca početnika u programiranju trebalo bi podrazumijevati učenje i strategije učenja koje se temelje na principu od konkretnog prema apstraktnom (Dann & Cooper, 2009a), odnosno od akcijske i ikoničke faze prema simboličkoj. Učenici lakše razumiju ono što vide, u suprotnom moraju uložiti puno veći napor, što može rezultirati visokom razinom frustracije, smanjenom motivacijom i na kraju odustajanjem. Jedna od ozbiljnijih poteškoća za učenje programiranja je činjenica da su programski jezici simbolički pa zahtijevaju visoku razinu apstrakcije (Moser, 1997). Programiranje je samo po sebi apstraktno i samim time teško za razumijevanje, ali je u isto vrijeme dobar alat za vježbu i razvoj apstraktnog razmišljanja. Cilj učenja programiranja ne mora biti stvaranje programera već vježba za poticanje učenja i apstraktnog mišljenja.

Iz svega navedenog nameću se tri važne postavke za računalnu znanost: sveprisutnost računalne znanosti, važnost konteksta, pomak od konkretnog prema apstraktnome. Pitanje pomaka od konkretnog prema apstraktnome je za računalnu znanost ključna jer je sama disciplina apstraktna. Konkretna iskustva su najučinkovitiji način učenja kada se pojavljuju u kontekstu relevantne konceptualne strukture (Dann & Cooper, 2009b).

2.3.2 Programiranje i rješavanje problema

Čovjeku je još od najranije dobi „urođeno“ rješavanje problema na svoj način, ali se često oslanja na pomoć drugih koji imaju više iskustva, što se može nazvati „uspinjanjem“ (eng. *scaffolding*). U navedenom procesu učitelj je taj koji pomaže učeniku doći do rješenja zadanog problema. Tako se pokazalo da učitelj djeluje potičući pažnju, smanjujući stupnjeve slobode u optimalnim granicama, usmjeravajući učenika prema rješavanju problema, kontrolirajući frustraciju te demonstrirajući rješenje kada ga učenik može prepoznati (Wood, Bruner, & Ross, 1976).

Rješavanje problema u samom je temelju programiranja jer se programiranje smatra rješavanjem problema. Rješavanje problema odnosi se na proces suočavanja s novom situacijom, stvaranjem veza među činjenicama, identificiranjem cilja i istraživanjem mogućih strategija za postizanje cilja (Szetela & Nicol, 1992). Računalno razmišljanje, koje se odnosi na rješavanje problema, je centralni aspekt računalne znanosti i programiranja koji zahtijeva apstraktno razmišljanje i omogućuje kreativno rješenje za dani problem. Štoviše, računalno razmišljanje je skup procesa rješavanja problema proizašlih iz informatike (računalne znanosti), ali primjenjivih u bilo kojoj domeni (Yadav, Stephenson, & Hong, 2017). Rješavanje problema i programiranje ne smiju se izjednačiti s informatikom iako jesu njezin temeljni dio (Denning, 2009a).

Programiranje zahtijeva visoku razinu sposobnosti rješavanja problema (White & Sivitanides, 2003b), ali se uz to programiranjem može vježbati rješavanje problema (Fessakis et al., 2013; Hassinen & Mäyrä, 2006c; Liao & Bright, 1991; Papert, 1980). Prethodna izjava može zvučati kontradiktorno jer ako netko nema dovoljno visoko razvijenu sposobnost rješavanja problema tada će programiranje biti teško i frustrirajuće što će najvjerojatnije rezultirati odustajanjem. Možemo zaključiti kako će učenici s visoko razvijenom sposobnosti rješavanja problema imati više uspjeha u programiranju, bez obzira na kontekst programiranja i korišteni programski jezik, dok će učenici s nižom sposobnošću rješavanja problema imati problema s istim.

Problemi nemaju uvijek jedinstveno rješenje. Neki problemi imaju cijeli niz mogućih rješenja koja se nazivaju algoritamskim rješenjima. Odabrana rješenja koja najbolje odgovaraju zadanom problemu nazivaju se algoritmom (Sprankle & Hubbard, 2008). Prilikom rješavanja problema pomoću računala jedno od najtežih zadataka je pisanje programskog koda. Zbog toga su odabir programskog jezika, konteksta programiranja te način postavljanja i vođenja prema rješavanju problema ključni u poučavanju programiranja pri čemu veliku ulogu ima učitelj. Uporabom odgovarajućih programskih jezika i konteksta programiranja, primjerenima dobi i kognitivnom razvoju učenika, može se poboljšati motivacija i uspjeh u programiranju.

2.3.3 Programiranje, mentalni modeli i *pojmovni stroj*

Programiranje zahtijeva više različitih vrsta „mentalnih modela“ što se može bitno razlikovati od poznavanja samog programskog jezika. Jasno razumijevanje mentalnog modela problema koji se rješava trebalo bi prethoditi pisanju samog programa. Stoga se, najčešće u prvim godinama učenja programiranja na fakultetima, programerski kolegiji temelje na rješavanju problema kod kojih se značajke jezika predstavljaju studentima u kontekstu rješavanja specifičnog problema (Robins et al., 2003a).

Mnoga istraživanja su pokazala kako centralnu ulogu ima model (apstrakcija) računala, često nazvan „*pojmovnim strojem*“ (eng. „*notional machine*“) (Du Boulay, 1986; Sorva, 2013). Apstrakcije su kreirane s razlogom – cilj zamišljenog stroja je objasniti izvršavanje programa. *Pojmovni stroj* je skup obilježja računala i njegove uloge kao izvršitelja programa u određenom programskom jeziku ili nizu sličnih programskih jezika. Stoga je *pojmovni stroj* vezan uz programski jezik. Du Boulay (Du Boulay, 1986) tvrdi kako *pojmovni stroj* treba biti jednostavan te po mogućnosti podržan konkretnim alatom koji omogućuje uvid u promatrani model. Ukratko treba osigurati „staklenu kutiju“ umjesto „crne kutije“ kako se program često doživljava. Sve navedeno odnosi se na poučavanje programiranja uglavnom studenata preddidplomske razine, početnika u programiranju, koji uglavnom imaju razvijeno apstraktno

mišljenje. Čak i za taj uzrast Du Boulay zapravo preporučuje učenje programiranja po principu od konkretnog prema apstraktnom. Iz svega navedenog jasno je s kojim se sve problemima susreću osnovnoškolski programeri početnici koji još nemaju razvijeno apstraktno mišljenje.

2.4 Programeri početnici i osnovnoškolci početnici

Izraz „programeri početnici“ najčešće se odnosi na studente preddiplomskog studija. Možemo li i osnovnoškolce programere početnike obuhvatiti istim izrazom? Prema svemu prethodno navedenom sigurno ne možemo.

Kako je već navedeno, programiranje zahtijeva razvijenu određenu razinu apstrakcije kako bi učenici mogli razumjeti programske koncepte koje uče. Programiranje je kompleksan proces za koji je, prema Du Boulayu (Du Boulay, 1986), potrebno pet međusobno ovisnih domena, ali i potencijalnih izvora poteškoća. To su: 1) generalna *orijentacija*; što se odnosi na programe, odnosno na ono što oni jesu i što se sve mogu činiti; 2) *zamišljeni stroj*; model računala koji se odnosi na izvršavanje programa; 3) *notacija*, sintaksa i semantika određenog programskog jezika; 4) *strukture*, shema i/ili planovi potrebni za izradu programa; 5) *pragmatičnost*, odnosi se na sposobnosti planiranja, razvoja, testiranja, otklanjanja grešaka u programu itd. (Robins et al., 2003a). Svi navedeni problemi međusobno su isprepleteni i poprilično zahtjevni, pogotovo za početnike, kada se moraju suočiti sa svim poteškoćama odjednom, već na samom početku.

Kada sve navedene poteškoće stavimo u kontekst učenika osnovnih škola, jasno je kako je „klasičan“ način poučavanja programiranja, koji se uglavnom primjenjuje na preddiplomskoj razini, neprimjeren i pretežak. Osim što osnovnoškolci još nemaju dovoljno razvijenu razinu apstrakcije, ni rješavanje problema nije dovoljno izvježbano, nemaju dovoljno iskustva i razvijenih mentalnih modela u koje se koncepti programiranja mogu uklopiti.

Prilikom poučavanja programiranja nastavnici trebaju voditi računa o nekoliko ključnih pitanja: 1) Tko su učenici?; 2) Što poučavamo?; 3) Kako poučavamo? (S. Mladenović, Krpan, & Mladenović, 2016). Danas se programiranje uči od vrtićke pa do odrasle dobi, stoga pojmom „programera početnika“ ne možemo obuhvatiti sve dobi učenika i pristupiti im istim načinom poučavanja („Tko su učenici?“). Zatim je potrebno definirati sposobnosti i znanje potrebno za učenje programiranja kako bi se ono moglo prilagoditi skupini koju se poučava („Što poučavamo?“). Na kraju je potrebno odlučiti kako poučavati. Na primjer: koju paradigmu odabrati (proceduralno, objektno-orijentirano i slično.), strategije poučavanja ili njihovu kombinaciju, odnosno koji kontekst programiranja koristiti (učenje temeljeno na igrama, projekti temeljeni na rješavanju problema…). Na pitanja „Što“ i „Kako“ lakše je odgovoriti

na sveučilišnoj razini jer je izbor često uvjetovan zahtijevima poslova za koje se studenti školuju, dok su odgovori za niže razine školovanja složeniji i značajnije ovise o faktoru dobi.

Uvažavajući sve navedeno može se zaključiti kako pod širokim pojmom programera početnika ne možemo svrstati baš sve početnike. Osnovnoškolski početnici u programiranju su posebno osjetljiva skupina kojoj pravilnim usmjeravanjem možemo olakšati programiranje, ali ih i zainteresirati za isto kako bi, po mogućnosti, nastavili školovanje u tom smjeru jer programiranje sve više postaje sastavni dio većine poslova današnjice, te je za očekivati da će sposobnosti potrebne za programiranje sve više biti potrebne u budućnosti (Levy & Murnane, 2005).

2.4.1 Kontekst programiranja

Kontekst programiranje odnosi se na domenu problema koja definira što se programira. Uobičajeni, „tradicionalni“ kontekst programiranja odnosi se na matematiku i rješavanje problema na način na koji se to radilo još od 70-ih godina prošlog stoljeća. Spuštanjem poučavanja na niže razine školovanja te napretkom tehnologije, prije svega medija, takav kontekst programiranja nije primjeren današnjim učenicima osnovnih škola, ali ni starijim učenicima. Rezultat pogrešnog odabira konteksta programiranja može rezultirati ne samo slabim usvajanjem pojmova kod učenika koji ne savladavaju programiranje s lakoćom, već i padom motivacije te odustajanjem i nezainteresiranošću za programiranje čak i kod onih učenika koji imaju visoko razvijene sposobnosti za programiranje. Dakle, primjereno odabran kontekst programiranja može olakšati učenicima učenje programiranja, ali i pozitivno djelovati na motivaciju (M. Mladenović, Krpan, & Mladenović, 2017, 2016).

Novi konteksti programiranja koji su se pojavili razvojem tehnologije i pojavom novih programskih okruženja odnose se na programiranje igara, pričanje priča, izradu animacija i slično. Pomicanjem konteksta programiranja s matematičkih problema prema interesima učenika možemo pozitivno utjecati na kognitivnu i afektivnu domenu učenika.

2.4.2 Programski jezici za početnike

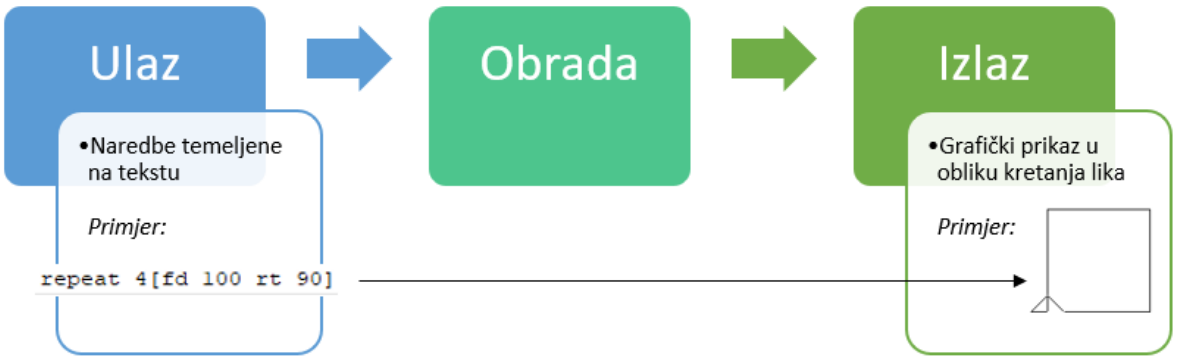
Programski jezici za početnike trebali bi imati jednostavnu sintaksu kako bi se pojednostavio proces učenja dok bi u isto vrijeme trebali biti moćan način za uvođenje programiranja učenicima. Takvi jezici se nazivaju mini-jezicima (Brusilovsky, Calabrese, Hvorecky, Kouchnirenko, & Miller, 1997). Fokus u početnoj fazi učenja programiranja trebao bi biti na semantičkoj strukturi programa umjesto na sintaksi programskog jezika.

Logo i BASIC se smatraju prvim mini-jezicima napravljenim za poučavanje programiranja, ali s različitim početnim idejama. Logo je primarno osmišljen za uvođenje programiranja osnovnoškolskom uzrastu kroz geometriju. Logo se temelji na konstrukcionizmu (Papert & Harel, 1991). Konstrukcionizam je u isto vrijeme teorija učenja i strategija poučavanja. Temelji se na dvije vrste „konstrukcije“: i) učenje je aktivan proces u kojem učenik aktivno konstruira znanje iz iskustva, (ova ideja temelji se na Piagetovoj teoriji konstruktivizma); ii) ljudi „konstruiraju“ novo znanje s određenom učinkovitošću kada su uključeni u konstruiranje smislenih produkata što se može odnositi na slaganje LEGO kockica, robota i/ili računalnih programa (Resnick, 1996). Najvažnije je učenikovo aktivno sudjelovanje u izradi nečega smislenog sebi i/ili svijetu oko sebe.

Temeljem sintakse programskog jezika, programski se jezici za početnike mogu podijeliti u tri grupe: vizualne-tekstualne, vizualne-blokovske, te proceduralne-tekstualne programske jezike.

2.4.2.1 Vizualni-tekstualni programski jezici

„Kornjačina grafika“ (eng. *Turtle graphics*) Logo (naziv nastao od grčke riječi *logos* što znači *riječ, misao, plan*) programski jezik (Papert, 1980) smatra se jednim od prvih mini-jezika. Logo, oblikovan krajem 60-ih godina prošlog stoljeća, temelji se na ideji mikrosvijeta u kojem učenik, kao programer, kontrolira kretanje nekog lika, ili više njih. Lik može biti kornjača, robot ili neki drugi po volji odabran lik. Glavna značajka takvih jezika su tekstualne naredbe kao ulaz dok je grafički prikaz rezultat izvršavanja programa koji može biti primjerice, pomicanje lika u mikrosvijetu. Primjer takvog programskog jezika (Logo) može se vidjeti na slici 2.1.



Slika 2.2 - Vizualno-tekstualni programski jezik

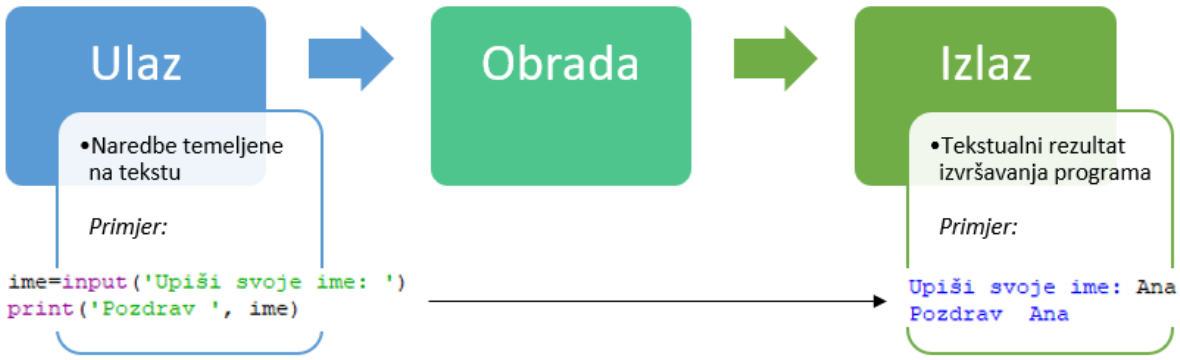
Značajka vizualno-tekstualnih programskih jezika je jednostavna tekstualna sintaksa s grafičkim rezultatom izvršavanja programa, najčešće prikazana u mikrosvijetu. Takvi se jezici smatraju primjerenima za prve programerske korake kod djece. „Kornjačina grafika “ programskog jezika Logo je prvi takav jezik. Ideja Logo jezika jako je dobro prihvaćena, toliko

da je iz istih postavki nastao cijeli niz novih vizualnih programskih jezika. Čak je i Python, programski jezik temeljen na tekstu, uključio *turtle* biblioteku tako omogućavajući korištenje „kornjačine grafike“ na način na koji se ista koristi u Logu.

Kontekst programiranja vizualno-tekstualnih programskih jezika najčešće je geometrija, kroz konkretno iskustvo programera početnika, u obliku programiranja lika koji se kreće u mikrosvijetu pritom crtajuću različite geometrijske likove. Vizualno izvršavanje programa učenicima omogućuje iskustvo „od konkretnog prema apstraktnom“ na način da se tekstualne naredbe izvršavaju crtanjem likova, što učenik može vidjeti i doživjeti u mikrosvijetu kroz konkretno iskustvo.

2.4.2.2 Proceduralni-tekstualni programski jezici

Gotovo u isto vrijeme kad je nastao Logo, nastao je i BASIC (akronim od eng. Beginner's All-purpose Symbolic Instruction Code), tekstualni programski jezik za proceduralno programiranje. Pojavom mikroračunala, sredinom 70-ih godina prošlog stoljeća, BASIC je postao popularan. Glavno obilježje tekstualnih programskih jezika za proceduralno programiranje su tekstualne naredbe kao ulaz, ali i tekstualni izlaz, jer je rezultat izvršavanja programa također tekstualan. Primjer takvog programskog jezika (Python) se može vidjeti na slici 2.3.



Slika 2.3 - Proceduralni-tekstualni programski jezici

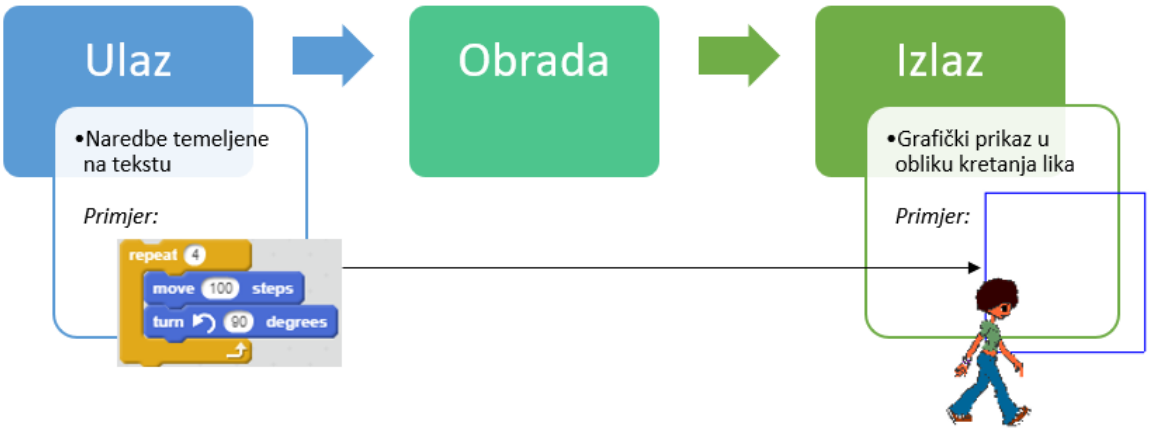
BASIC se smatra pretečom novijih, sličnih programskih jezika. U današnje vrijeme programski jezik Python najčešće se koristi za poučavanje proceduralnog programiranja u tekstualnom programskom jeziku za početnike zbog jednostavne sintakse, te komercijalne popularnosti.

Kontekst programiranja za proceduralno programiranje u tekstualnim programskim jezicima još uvijek se uglavnom temelji na rješavanju matematičkih problema. Kako se, i ulaz, i izlaz izvršavanja programa temelje na tekstu, potrebna je sposobnost apstraktnog razmišljanja kako

bi se usvojili koncepti programiranja pri poučavanju i učenju programiranja. Zbog toga, unatoč jednostavnoj sintaksi, tekstualni programski jezici vjerojatno nisu prikladni za niže razrede predmetne nastave osnovne škole.

2.4.2.3 Vizualni-blokovski programski jezici

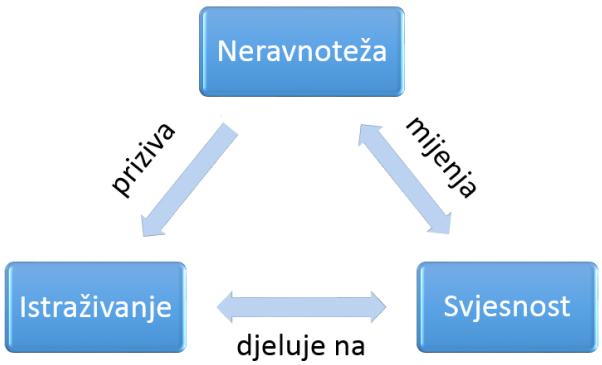
U zadnja dva desetljeća pojavila se nova generacija programskih jezika oblikovanih za učenje i poučavanje programiranja namijenjenih mlađim naraštajima, već od vrtićke dobi. Zajednička poveznica za navedene jezike je „Logo filozofija“ prema kojoj su napravljeni, a odnosi se na programiranje u nekom mikrosvijetu koji predstavlja konkretan svijet kojim se može manipulirati. Osnovna razlika takvih jezika, u odnosu na Logo, je prikaz instrukcija, kod kojih su ulaz i naredbe predstavljeni blokovima kombiniranim u obliku slagalice, dok je izlaz grafički prikaz u obliku kretanja lika.. Primjer takvog programskog jezika (Scratch) može se vidjeti na slici 2.4.



Slika 2.4 - Vizualni-blokovski programski jezici

Glavna značajka takvih, vizualnih-blokovskih, programskih jezika, kao što su Scratch, Alice, Micro:bit i slični, je manipulacija vizualnim objektima, u obliku djelova slagalice, umjesto pisanja tekstualnih naredbi. Pokazalo se kako vizualne ilustracije olakšavaju učenje (Levin, 1981) što se može postići vizualizacijom naredbi u programiranju, na primjer naredbama u obliku slagalica kao u blokovskim programskim jezicima. Na taj moguće je olakšati poteškoće u razumijevanju kompleksnih koncepata programiranja kod učenika. Zbog toga se vjeruje kako programski jezici temeljeni na blokovima od učenika zahtijevaju konkretnu razinu kognitivnog razvoja za razliku od tekstualnih programskih jezika koji zahtijevaju kognitivnu fazu formalnih operacija.

Osim toga, značajka programskih jezika temeljenih na blokovima je pomicanje konteksta programiranja od rješavanja matematičkih problema prema programiranju igara, animacija, priča i slično. Pomicanjem konteksta programiranja na navedeni način, učenici mogu lako istraživati primjenu različitih programskih konstrukata. Istraživanjem tijekom programiranja učenici imaju više informacija i iskustva za razumijevanje zadanog problema što može rezultirati različitim formama kognitivnog rasta (eng. cognitive scaffolding) od kodiranja do povratne informacije tijekom, na primjer, igranja igre. Ovaj proces uklapa se u kognitivnu neravnotežu (D'Mello & Graesser, 2012; Yuen & Liu, 2010). Postavke okruženja programiranja, kao što su programski jezik, kontekst programiranja i zadaci ili primjeri, trebali bi dovesti učenika do razine frustracije optimalne za motivaciju. Osim kognitivne neravnoteže tijekom učenja se javljaju i kognitivni procesi istraživanje i svjesnost. Neravnoteža, istraživanje i svjesnost povezani su procesi kojima se dolazi do viših razina razmišljanja i rješavanja problema što ilustrira Slika 2.5.



Slika 2.5 - Odnos neravnoteže, istraživanja, svjesnosti (Yuen & Liu, 2010)

Postavke okruženja programiranja trebala bi biti dovoljno izazovna za izazivanje optimalne razine frustracije. Preteško postavljene postavke mogu voditi do prevelike frustracije i preopterećenja učenika, dok prelagano postavljene postavke mogu dovesti do dosade.

Štoviše, programiranjem u vizualno-blokovskim programskim jezicima, osnovni pojmovi programiranja mogu se lako uvesti kroz programiranje igara te na taj način upoznati učenike sa svim osnovnim pojmovima programiranja već u prvim predavanjima. Ovakve postavke učenja programiranja: kontekst programiranja (programiranje igara, priča, animacija i slično) u blokovskim programskim jezicima omogućuju učenje otkrivanjem (Bruner, 1961). Situacije rješavanja problema su okidači učenja otkrivanjem, kad se učenik oslanja na vlastito iskustvo i prethodno znanje u svrhu rješavanja novonastalog problema interakcijom s okruženjem, istraživanjem i manipulacijom objektima. Tijekom programiranja igre, učenik mora postaviti

scenarij igre u formi algoritma kako bi isprogramirao igru. Nadalje, učenik mora rukovati s naredbama, u obliku slagalica, za konstrukciju programa. U slučaju neispravnog izvršavanja programa učenik mora naći pogrešku istraživanjem programa. Tijekom ovog procesa učenici aktivno koriste koncepte programiranja bez da su ih eksplicitno svjesni. Na učitelju je da tijekom nastavnog procesa učenicima otkrije korištene pozadinske pojmove programiranja.

2.4.2.4 Postavke okruženja programiranja

Temeljem ulaza, izlaza i konteksta programiranja programska okruženja mogu se svrstati u tri skupine: vizualni-blokovski, vizualni-tekstualni te proceduralni-tekstualni.

Tablica 2.2 prikazuje prijedlog programskih okruženja prema odgovarajućoj kongnitivnoj razini učenika.

Tablica 2.2 - Programska okruženja i razine kognitivnog razvoja

	Razine kognitivnog razvoja prema Piagetu		
Programski jezici/okruženje	**Konkretna**	**Pred-Formalna**	**Formalna**
Vizualni -> blokovski (Primjer: Scratch, Alice, Hour of code, Microsoft MakeCode micro:bit editor)	Optimalno	Nije izazovno	Nije izazovno
Vizualni -> tekstualni (Primjer: Logo, Python-uz korištenje turtle biblioteke, Greenfoot)	Preopterećenje	Optimalno	Nije izazovno
Proceduralni –> tekstualni (Primjer: Python, BASIC)	Preopterećenje	Preopterećenje	Optimalno

Ako sadržaj učenja prelazi kognitivnu razinu učenika, može doći do kognitivnog preopterećenja što može dovesti do manjka samopouzdanja, smanjene motivacije i uspjeha učenika. Stoga bi, pri odabiru odgovarajućeg programskog jezika za osnovnoškolce početnike u programiranju, učitelj trebao uzeti u obzir njihovu dob te kognitivne karakteristike učenika. Izborom odgovarajućeg programskog okruženja učitelj može potaknuti učenje koncepata programiranja što može pridonijeti smanjenju razvoja pogrešnih predodžbi (miskoncepcija), te povećati motivaciju i zadovoljstvo učenika. U suprotnom, uvođenjem neodgovarajućeg programskog okruženja, učenici u ranoj fazi mogu odustati od programiranja zbog razvoja viših razina frustracija i preopterećenja. Na primjer, ako učitelj uvede proceduralni tekstualni programski jezik prerano za dob učenika, većina učenika bi, zbog visoke razine frustracije, mogla odustati od programiranja zbog kognitivnog preopterećenja. Isto tako, ako se, na primjer, uvede vizualni

blokovski programski jezik u dobi u kojoj su ga učenici već „prerasli“, razina frustracije bi mogla postati preniska što može rezultirati dosadom i nedovoljnom razinom izazova.

Ipak, ako se uzmu u obzir i Brunerove faze reprezentacije stvarnosti (Bruner, 1966) ne može se generalizirati i tvrditi kako blokovski programski jezici nemaju smisla u učenju programiranja u odrasloj dobi, na primjer na preddiplomskoj razini. Ne treba smetnuti s uma kako je pristup „od konkretnog prema apstraktnom“ i dalje najpoželjniji, što u ovom slučaju znači kako bi se na preddiplomskoj razini vizualni-blokovski programski jezik mogao iskoristiti za uvod u pozadinske apstraktne pojmove, te bi prijelaz na tekstualne programske jezike išao puno brže nego na nižim razinama obrazovanja. Optimalno bi bilo, na primjer na prediplomskoj razini uvesti kompleksne pojmove programiranja kroz sve tri faze: akcijsku, ikoničku i simboličku, ali naravno prijelaz prema simboličkoj razini bit će značajno brži od onoga na osnovnoškolskoj razini.

Kako se istraživanje ovog rada provodi u osnovnim školama, vizualni blokovski programski jezici smatraju se primjerenim za poučavanje programiranja, te će se takvi jezici u sljedećem poglavlju detaljnije opisati.

2.4.3 Programski jezici za početno učenje programiranja osnovnoškolaca

Ljudima je prirodno razmišljati od konkretnog prema apstraktnom, pa je upravo po tom principu Papert sa suradnicima razvio programski jezik Logo koji je prilagođen djeci, jer se naredbe koje se zadaju računalu vide kroz konkretne, vizualne rezultate što povećava i afektivnu domenu (Papert, 1980). Papert je tvrdio kako programski jezici trebaju imati „nizak pod“ (eng. *Low floor*) (početak treba biti lak), „visoki strop“ (eng. *High ceiling*) (s vremenom treba postojati mogućnost izraditi sve kompleksnije projekte), i „široke zidove“ (eng. *Wide walls*) (treba podržati različite tipove projekata primjerene osobama različitih interesa i stilova učenja). Zadovoljavanje trojke *niski pod/visoki strop/ široki zidovi* nije lako (Resnick et al., 2009). Logo se smatra prvim jezikom prilagođenim za početnike koji je potakao razvoj drugih takvih jezika koji se nazivaju mini-jezicima. Značajke mini-jezika su mali broj instrukcija i njihova jednostavnost kako bi olakšali prve korake u programiranju (Brusilovsky et al., 1997). U većini mini-jezika učenici uče programirati kroz upravljanje nekim fizičkim ili virtualnim likom. Fizički lik predstavljen je fizičkim uređajem, koji je u početku Loga predstavljen kornjačom, ili danas robotom. Virtualni lik predstavljen je računalnim modelom lika u mikrosvijetu kojim se upravlja vrlo jednostavnom sintaksom. Mini-jezici u novije vrijeme nastali po uzoru na Logo, su vizualni programski jezici poput Scratch, Alice, Greenfoot, GameMaker, Kodu, NXTG (Lego Mindstorms) i drugi. Takvi jezici pružaju mogućnost stvaranja priča, oblikovanja video

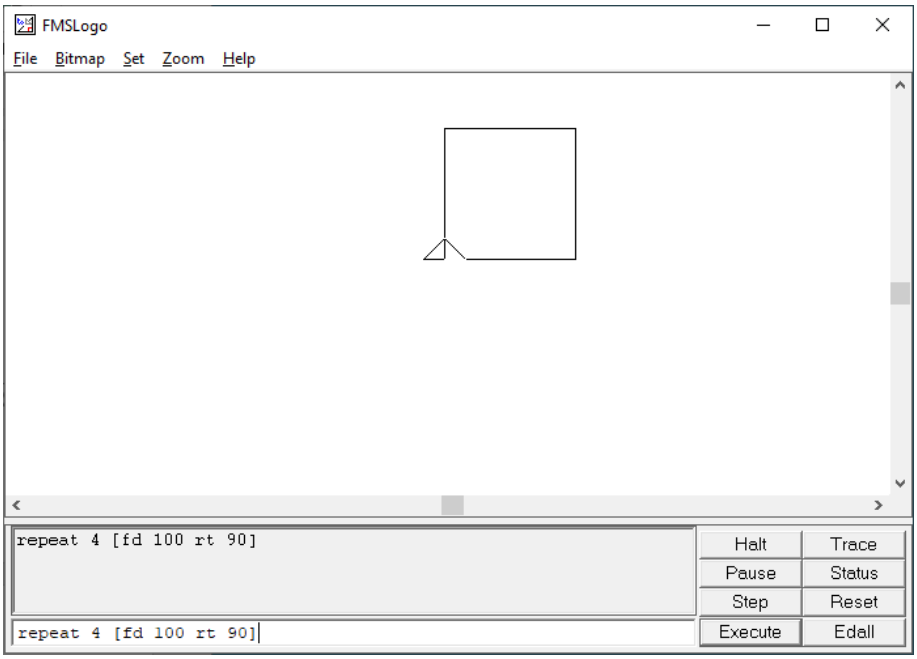
igara, izrada scenarija i slično. Vizualni programski jezici korisniku pružaju iskustvo programiranja od konkretnog prema apstraktnom u određenom kontekstu. Inače teški, kompleksni koncepti programiranja se kroz ovakve jezike doživljavaju preko konkretnih događaja, pa se lakše prenose u apstrakciju, na primjer apstraktni pojmovi poput varijabli, petlji i slično u vizualnom okruženju pružaju konkretno iskustvo kroz vizualizaciju pojedinih koncepata. Navedeni programski jezici zadovoljavaju i potrebe digitalnih urođenika za interakcijom s računalom, vizualno su privlačni te omogućuju programiranje u kontekstu. Kontekst programiranja se na ovaj način pomiče od aktivnosti koje nisu privlačne digitalnim urođenicima i stavlja se u okruženje koje omogućuje razvoj ne samo matematičkih sposobnosti, već potiče kreativnost na drugim poljima, čime se u obzir uzimaju i druge vrste inteligencije, a ne isključivo matematičko-logička. Ovakav način učenja programiranja otvara put učenja programiranja učenicima koji pripadaju digitalnom dobu.

Prethodno navedeni jezici dobri su za početno učenje programiranja, ali ne mogu se nazvati igrama jer nemaju cilj već se mogu nazvati okruženjima u kojima se može igrati i izrađivati igre. Kako bi se povezala navedena dva svijeta, programiranja i igara, nameće se ideja o učenju programiranja programiranjem igara. Jezici koji su navedeni pogodni su i za programiranje igara. Na taj način učenicima je zabavnije programirati, a prebacivanjem težišta sa sintakse na semantičku strukturu programa učenici zapravo nisu svjesni kako vježbaju rješavanje problema, otklanjaju pogreške u programima, izrađuju scenarije, već su uvjereni kako izrađuju računalne igre. Poticanje učenika na učenje jednog dok oni misle da rade nešto sasvim drugo, Pausch je nazvao prijevara glave (eng. „head fake“) (Pausch & Zaslow, 2008).

U provedenim istraživanjima u sklopu ove disertacije koristili su se sljedeći programski jezici: FMSLogo, Python, Scratch i Micro:bit. Glavne značajke navedenih programskih jezika i njihovih sučelja ukratko su opisani u nastavku.

2.4.3.1 LOGO

Logo je programski jezik osmišljen za učenje geometrije, a koji se koristi za poučavanje programiranja (Papert, 1980). Značajke su mu jednostavna sintaksa, brzo izvršavanje programa bez potrebe kompajliranja, te vizualni prikaz izvršavanja programa. Na slici 2.6 se vidi sučelje verzije FMSLogo koja se koristila tijekom istraživanja opisanih u ovoj disertaciji.



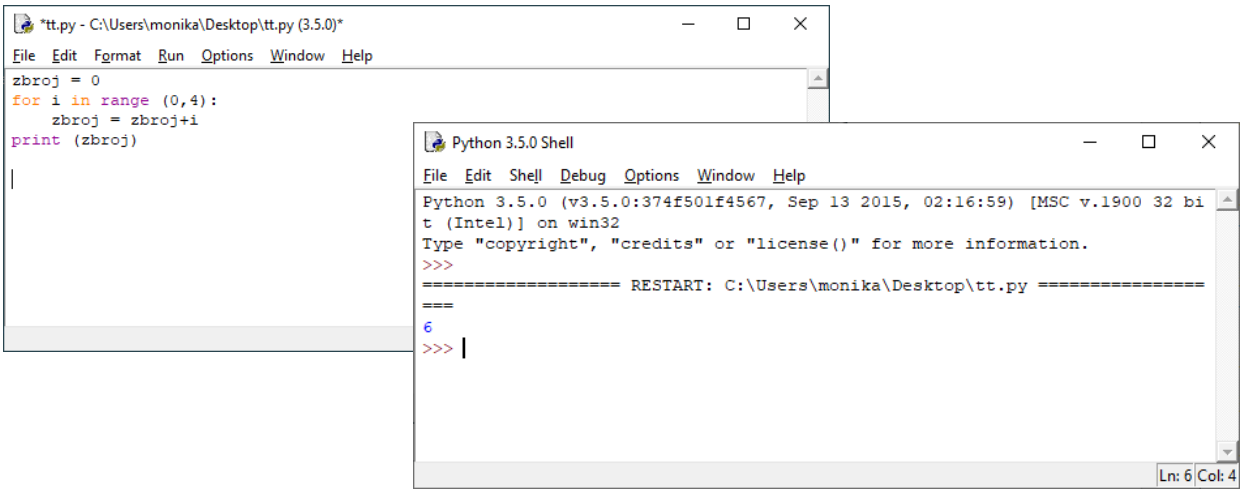
Slika 2.6 - Sučelje programskog jezika FMSLogo

Rezultat izvršavanja naredbi prikazuje se na virtualnom zaslonu točno određene rezolucije koja nije direktno povezana s rezolucijom stvarnog zaslona. Svaka točka ima svoju koordinatu, a svaki objekt, u ovom slučaju „kornjača“ ima svoj smjer kretanja (Foit, 2011). Na početku programa „kornjača“, koja u FMS logu izgleda kao trokut nalazi se na sredini, odnosno na koordinati (0,0) u smjeru 0^{o}. Naredbama se „kornjača“ pomiče po zaslonu te programer vidi grafički rezultat izvršavanja naredbi.

2.4.3.2 Python

Python je skriptni jezik visoke razine kojeg je 1990. godine kreirao Guido van Rossum. Zbog jednostavne sintakse jednostavan je za početnike u programiranju, ali ima visoke mogućnosti, zbog čega se koristi i u komercijalnoj primjeni što mu je prednost. Kako je skriptni jezik tako se programi pišu i izvršavaju jednostavno i brzo bez potrebe kompajliranja. Isto može biti i nedostatak jer se veći programi mogu sporije izvršavati što, zbog veličine programa, nije bitno na osnovnoškolskoj razini učenja programiranja (Grandell, Peltomäki, Back, & Salakoski, 2006).

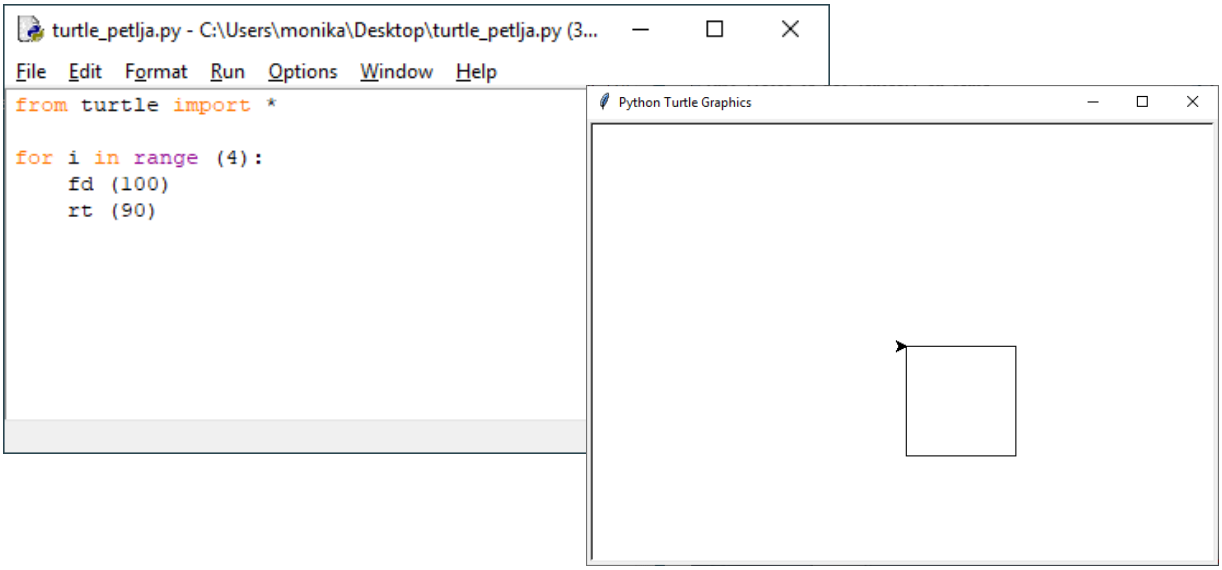
U nastavi tijekom istraživanja se koristilo nekoliko verzija Pythona od kojih su sve iz inačice 3.x jer nema razlike u sintaksi za skup korištenih naredbi. Na slici 2.7 je prikazano tekstualno sučelje IDLE koje dolazi s instalacijom Pythona.



Slika 2.7 - Sučelje programskog jezika Python

Naredbe, funkcije, metode i ključne riječi su obojene što olakšava pronalazak sintaktičkih pogrešaka. Pokretanjem programa otvara se *Python Shell*, tekstualni prozor u kojem se ispisuje rezultat izvršavanja programa.

S Pythonom dolazi i *turtle* biblioteka čijim uključivanjem Python dobija Logo svojstva. Potrebno je na početku programa uključiti *turtle* biblioteku kao što je prikazano na slici 2.8.



Slika 2.8 - Upotreba turtle biblioteke u Pythonu

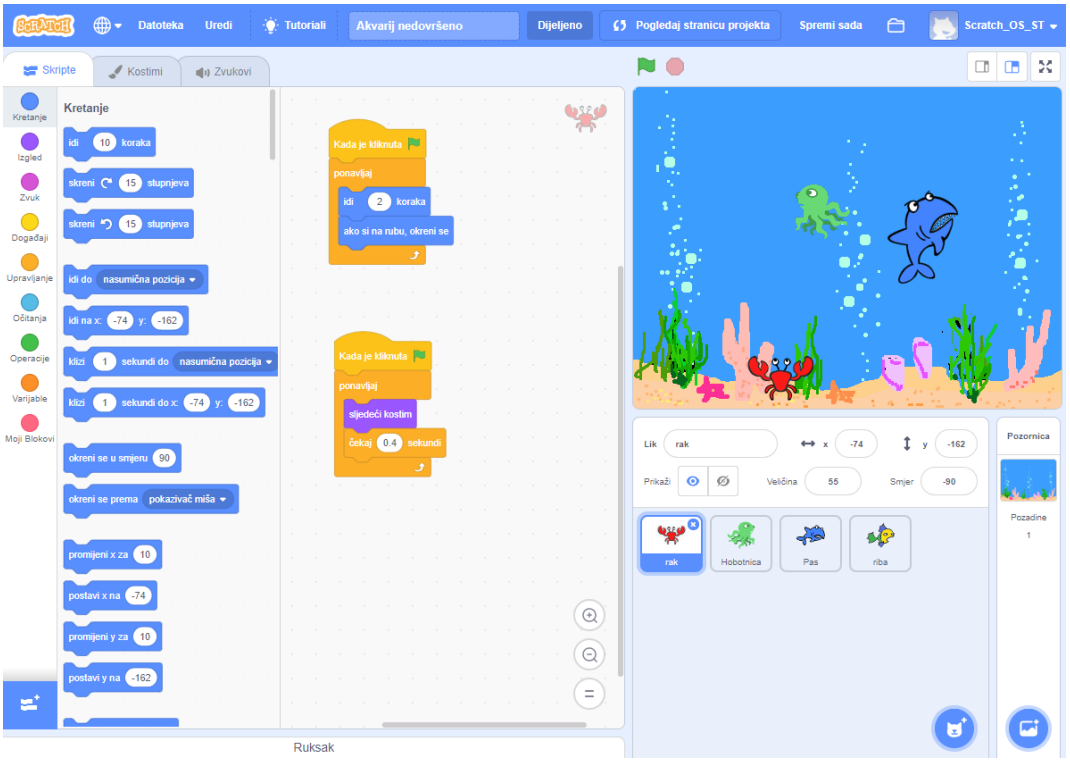
Izvršavanjem programa otvara se *Python Turtle graphics* prozor u kojem se grafički prikazuje rezultat izvršavanja programa. *Python Turtle graphics* je, kao u Logu, virtualni zaslon na kojem svaka točka ima svoju koordinatu, a svaki objekt svoj smjer. U Python kornjači početna koordinata ista je kao kod Loga (0,0), ali je početni smjer 90^{o}.

Razlike u sintaksi između Loga i Pythona, što se tiče naredbi korištenim u istraživanja, odnose se na *for* petlju. Naime, kako u Pythonu ne postoji *repeat* naredba kao u Logu tako se koristi *for* petlja.

2.4.3.3 Scratch

Scratch je jedan od najraširenijih vizualnih-blokovskih programskih jezika nastao 2007. godine na Papertovoj ideji (Papert, 1980), na istoj ideji na kojoj je nastao Logo. Scratch je produkt rada istraživačke grupe *Lifelong Kindergarten* pri *MIT Media Lab* u suradnji s Lego tvrtkom pri izradi *Lego Mindstorms* robota (Resnick et al., 2009).

Postoji nekoliko verzija Scratcha. Verzija koja je korištena u istraživanjima je desktop verzija 1.4. Posljednja, aktualna verzija je on-line inačica Scratch 3.0 (slika 2.9).



Slika 2.9 - Sučelje Scratch 3.0 programskog jezika

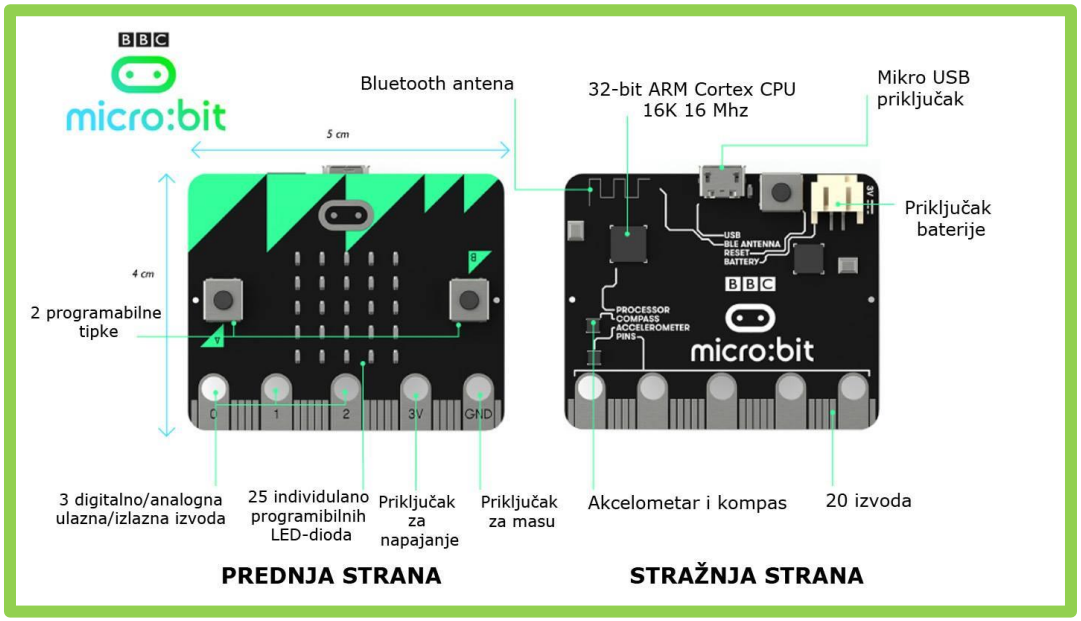
Zbog intuitivnosti i sličnosti između verzija, naredbe i programi koji su se koristili u istraživanju se na isti način mogu koristiti u novijim inačicama.

Naredbe u Scratchu prikazane su u obliku slagalica koje su grupirane tematski po bojama. Osim toga naredbe su lokalizirane na više od 50 jezika među kojima je i hrvatski. Naredbama se kontroliraju likovi prikazani o obliku 2-D grafičkih objekata na pozornici. Korisnik slaže odgovarajuće naredbe u skripte. Pokretanjem programa korisnik vidi rezultat programa u obliku kretanja likova na pozornici čime se na konkretan način prikazuje izvršavanje programa. Važna

značajka je da nije potrebno kompajlirati, odnosno *uredi/pokreni* način rada. Korisnik klikom može pokrenuti program ili samo pojedine skripte (J. Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010).

2.4.3.4 Micro:bit

Micro:bit je programibilni mikrokontroler ili malo računalo, izrađeno je u edukacijske svrhe u Velikoj Britaniji od strane tvrtke BBC 2016. godine (slika 2.10). Zahvaljujući niskoj cijeni, sklopovlju, jednostavnosti i zanimljivosti za učenje programiranja i nastavu, može se primijeniti u cijelom STEM području za što je zapravo i napravljen, te se pokazao vrlo uspješnim pothvatom u Velikoj Britaniji (Ball et al., 2016; Rogers et al., 2017; Tyrén, Carlborg, Heath, & Eriksson, 2018).



Slika 2.10 - Micro:bit

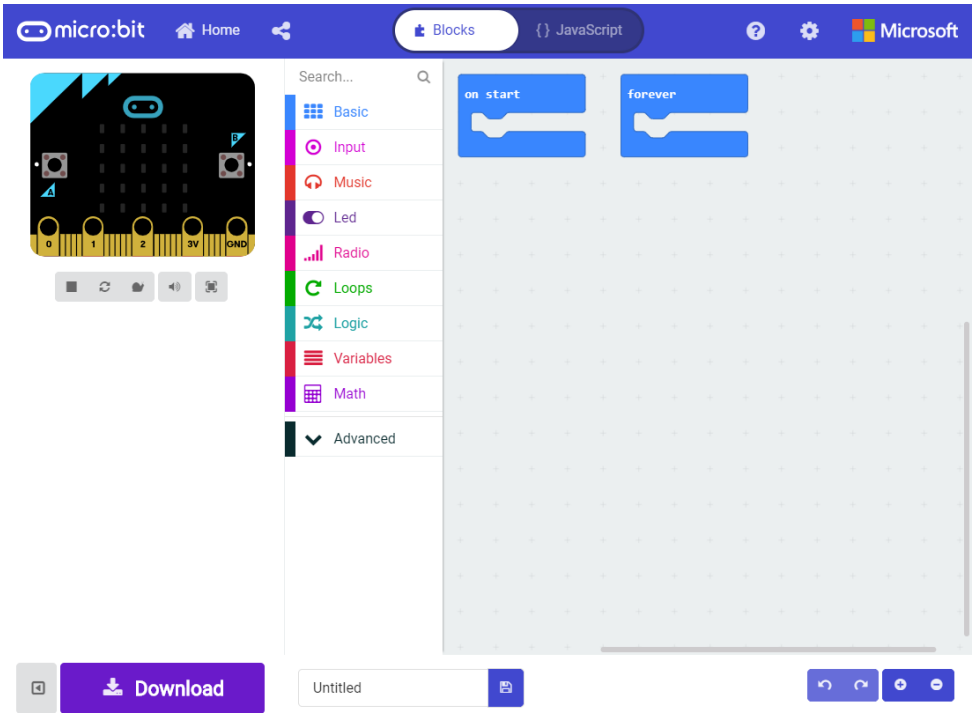
Osnovne značajke sklopovlja su:

- 5x5 LED svjećice
- Dva programibilna dugmeta
- Akcelerometar
- Kompas
- Senzor svjetla
- Senzor temperature
- Blutooth
- Pet priključaka za ulaz i izlaz (I/O)

Micro:bit se može programirati koristeći nekoliko programskih jezika kao što su: Microsoft MakeCode micro:bit editor, javaScript, Python. Od navedenih programskih jezika Microsoft MakeCode micro:bit editor (slika 2.11) predstavnik je vizualnog-blokovskog programskog jezika primjeren za početnike u programiranju. Programiranje fizičkih, opipljivih uređaja ima pozitivan učinak na aktivno učenje i suradnju, potiče kreativnost, omogućuje usmjeravanje na ideje umjesto na ograničenja, poticajno djeluje na manje zastupljene skupine, poput djevojčica, koje rado programiraju (Sentance, Waite, Hodges, MacLeod, & Yeomans, 2017).

Ovakvim uređajem koji je popraćen s više programskih jezika, od blokovskog do tekstualnih, omogućuje se ostvarenje svih faza Brunerove reprezentacije stvarnosti, od fizičke interakcije s uređajem, preko ikoničkog (blokovskog) pa do simboličkog (tekstualnog) programskog jezika.

Zbog svega navedenog se Microsoft MakeCode micro:bit editor, kao blokovski programski jezik, koristio u istraživanju opisanom u ovoj disertaciji.



Slika 2.11 - Sučelje MakeCode micro:bit programskog jezika

Za izradu jednostavnog programa učenik treba odabrati odgovarajuće naredbe koje se nalaze u središnjem dijelu sučelja. Naredbe su grupirane tematski i kodirane bojama. Početni blokovi koji se prikazuju su *on start* i *forever*. Napravljeni programi mogu se vrednovati na simulatoru i preuzeti na računalo kako bi se prebacili na micro:bit uređaj. Osnovni koncepti programiranja: slijed, ponavljanje i grananje mogu se na jednostavan način prikazati učenicima.

2.4.4 Osnovni koncepti programiranja

Algoritam je niz koraka napravljen kako bi se izvršio određeni zadatak, odnosno riješio određeni problem. Izrada algoritma se sastoji od plana rješavanja problema koji se zatim prevodi u niz koraka i konačno u programski jezik. Proceduralni programski jezici koriste programske konstrukte za kontrolu tijeka izvršavanja naredbi. Böhm i Jacopini su uveli klasičnu definiciju prema kojoj se programi mogu prikazati dijagramom tijeka (Böhm & Jacopini, 1966). Temeljem rezultata njihovog rada mogu se definirati tri osnovna algoritma pomoću kojih se može prikazati bilo koji program, bez obzira na njegovu kompleksnost: algoritmi slijeda, grananja i ponavljanja. Osnovni algoritmi, slijeda, grananja i ponavljanja, su temelj za sve složenije programske strukture, stoga, bez savladavanja osnovnih konstrukata teško se mogu razumjeti složeniji programski konstrukti. Zbog navedenog je u početnom učenju programiranja ključno savladati navedene osnovne algoritme.

Algoritam slijeda odnosi se na niz uputa koje se izvršavaju jedna za drugom u zadanom redoslijedu (Brennan & Resnick, 2012). Ovaj algoritam važan je u mnogim disciplinama poput matematike, ali i literarnog izražavanja (Bers, Flannery, Kazakoff, & Sullivan, 2014). Algoritmom slijeda započinje poučavanje, odnosno učenje, programiranja. Ovaj algoritam se najčešće objašnjava primjerima iz svakodnevnog života u obliku, primjerice, algoritma za pranje zubi. U vizualnim blokovskim i tekstualnim programskim jezicima može se započeti s algoritmom slijeda scenarijem za kretanjem određenog lika u nekom mikrosvijetu. Prednost ovakvog pristupa je mogućnost povezivanja svake od naredbi programskog jezika s akcijom lika kojeg se pokreće. Kod tekstualnih programskih jezika u proceduralnom pristupu pri uvođenju algoritma slijeda se najčešće moraju uvesti i naredbe za unos i izlaz, varijable te osnovne aritmetičke operacije.

Algoritam grananja odnosi se na odluke u programu, odnosno na uvjetno izvršavanje programa (Brennan & Resnick, 2012). Uvođenjem ovog algoritma povezuje se rezultat uvjeta, koji može biti istina ili laž, s nastavkom izvršavanja programa što je akcija koja uključuje logičko zaključivanje. U programiranju algoritam grananja realizira se *if-else* strukturom. U vizualnim blokovskim i tekstualnim programskim jezicima uvođenje grananja opet se odvija u nekom mikrosvijetu u kojem učenici vide rezultat izvršavanja programa koja ovisi o odluci, a očituje se kroz konkretnu akciju nekog lika u mikrosvijetu. Predstavljen na ovaj način, algoritam grananja često se kombinira s algoritmom ponavljanja. Kod tekstualnih proceduralnih programskih jezika potrebno je prethodno učenike upoznati s relacijskim operatorima te uvjetima uglavnom vezanim uz uspoređivanje brojeva.

Algoritam ponavljanja odnosi se na ponavljanje jedne ili više akcija (Brennan & Resnick, 2012). Osnovni primjeri su jednostavni, ali kod složenijih je ključno prepoznati uzorak ponavljanja. Algoritam ponavljanja, koji se u programiranju realizira kroz konstrukt petlje, jedan je od temeljnih koncepata programiranja. To je kompleksan i apstraktan koncept, posebno za osnovnoškolske početnike. Stoga je iznimno važno razumijevanje jednostavnih, početnih primjera kako bi se kasnije mogli savladati kompleksniji primjeri. U nastavi to znači kako bi prije prelaska na kompleksnije, učenici trebali savladati jednostavne slučajeve.

Algoritam ponavljanja se u programiranju realizira kroz konstrukt petlje. Postoje dvije osnovne vrste petlji:

- petlje bez logičkog uvjeta (na primjer *repeat* u Logu, *for* u Pythonu i Logu);
- petlje s logičkim uvjetom (na primjer *while* i *repeat-until*)

Petlje bez logičkog uvjeta jednostavnije su za razumijevanje pa se u pravilu uvode i usvajaju prije petlje s logičkim uvjetom.

Kod programskih jezika temeljenih na blokovima, petlje se uvode drugačijim redoslijedom i imaju nešto drugačiju podjelu, ali je pozadinski koncept isti (Ben-Ari, 2011). Kod blokovskih programskih jezika petlje bez logičkog uvjeta se dijele na beskonačne i konačne (ograničeni broj ponavljanja). Pojam beskonačne petlje može se realizirati u tekstualnim programskim jezicima, ali pomoću petlje s logičkim uvjetom.

U tablici 2.3 prikazani su primjeri petlji bez logičkog uvjeta u programskim jezicima Scratch, Logo i Python.

Tablica 2.3 - Primjer petlji bez logičkog uvjeta

Scratch	Logo	Python
	`repeat 10 [ fd 10 ]`	`for i in range (10):` `  fd (10)`

Iako su naredbe analogne u primjerima programskih jezika na tablici, uočljiva je razlika u samoj sintaksi programskih jezika gdje je u blokovskom programskom jeziku *repeat* naredba intuitivna te učenik ne može napraviti sintaksnu pogrešku, do sintakse u Pythonu koja je značajno kompleksnija, pogotovo za početnika osnovnoškolca. Osim toga očita je razlika i u potrebnom predznanju za uvođenje navedenih naredbi. U Scratchu i Logu nije potrebno

prethodno poznavanje nekog koncepta dok je za *for* strukturu u Pythonu potrebno predznanje o konceptu varijable. To je jedan od osnovnih razloga zbog kojeg se algoritam ponavljanja obrađuje posljednji kod proceduralnih programskih jezika dok se kod vizualnih obrađuje nakon algoritma slijeda.

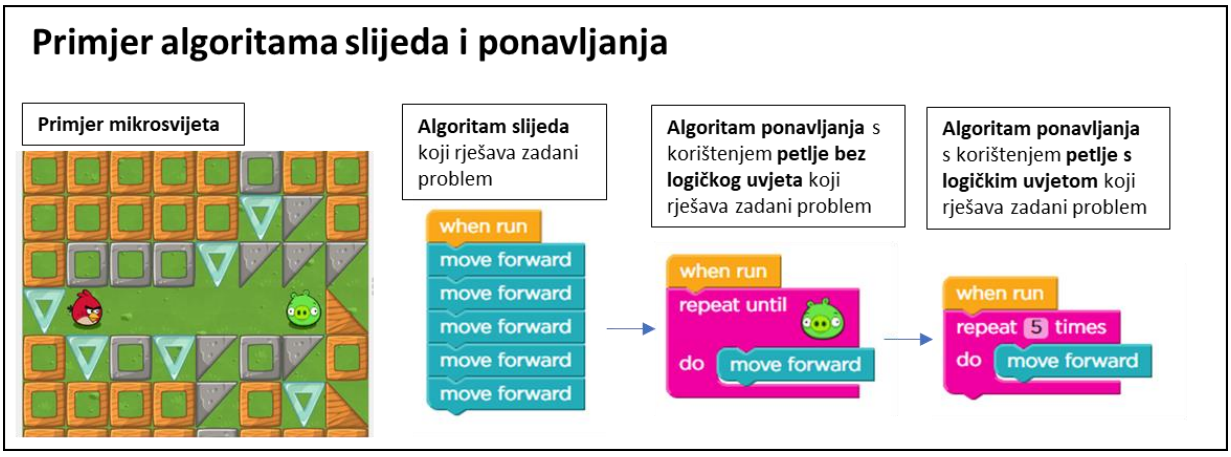
2.4.5 Redoslijed uvođenja osnovnih algoritama

Redoslijed uvođenja osnovnih algoritama ovisi o korištenom programskom jeziku i kontekstu programiranja. Tako je redoslijed uvođenja osnovnih algoritama prema vrsti programskih jezika sljedeći:

- **slijed-grananje-ponavljanje** za proceduralne tekstualne programske jezike,
- **slijed-ponavljanje-grananje** za vizualno-blokovske i vizualno-tekstualne (npr. Logo) programske jezike.

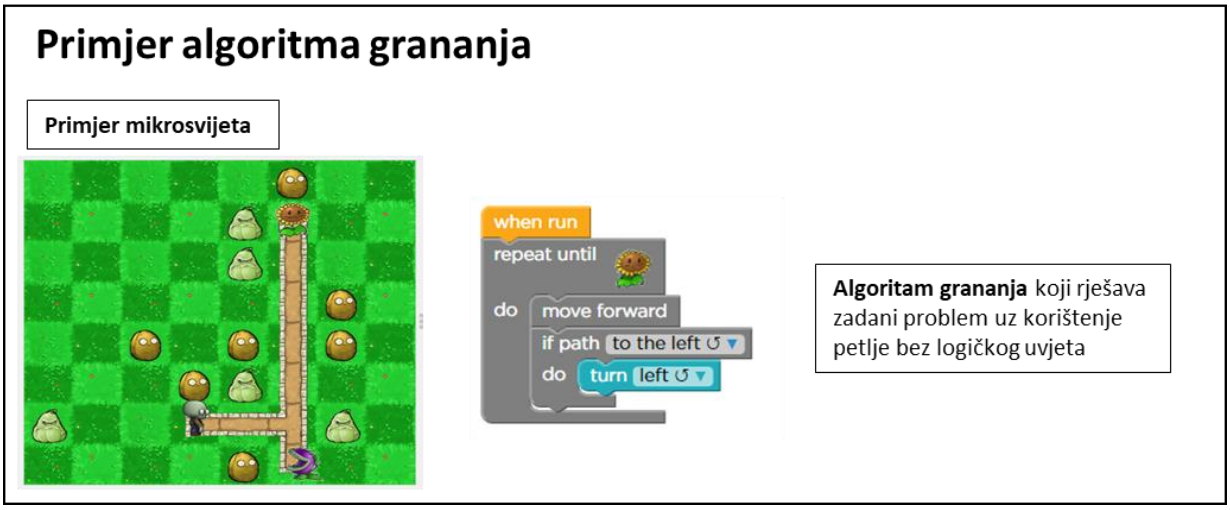
Razlika u redoslijedu uvođenja koncepta petlje između programskih jezika nastala je zbog same prirode programskih jezika na što utječe i kontekst programiranja. Tako je u mikrosvijetu „prirodno“ nakon algoritma slijeda uvesti algoritam ponavljanja jer nam je potreban za, na primjer neprekidno pomicanje nekog lika u mikrosvijetu.

Algoritmom slijeda može se napraviti određeni broj pokreta lika dok se algoritmom ponavljanja može realizirati „beskonačan“ (do prestanka izvršavanja programa ili do isključivanja robota ako je riječ o fizičkom uređaju) ili ograničen broj koraka kretanja lika. Primjer uvođenja algoritma ponavljanja kroz petlju bez uvjetnog izvršavanja i petlju s uvjetnim izvršavanjem prikazan je na slici 2.12.



Slika 2.12 - Primjer uvođenja algoritama slijeda i ponavljanja u vizualnom-blokovskom programskom jeziku

Na slici 2.12 prikazan je primjer vizualnog-blokovskog programskog jezika Hour of code na primjeru igre. Može se uočiti kako se kroz problem istog mikrosvijeta može uvesti algoritam slijeda te pokazati primjenu algoritma grananja i to korištenjem petlje bez logičkog uvjeta i petlje s logičkim uvjetom. Algoritam grananja najčešće se uvodi kao treći u ovakvim jezicima i najčešće se kombinira s algoritmom ponavljanja.

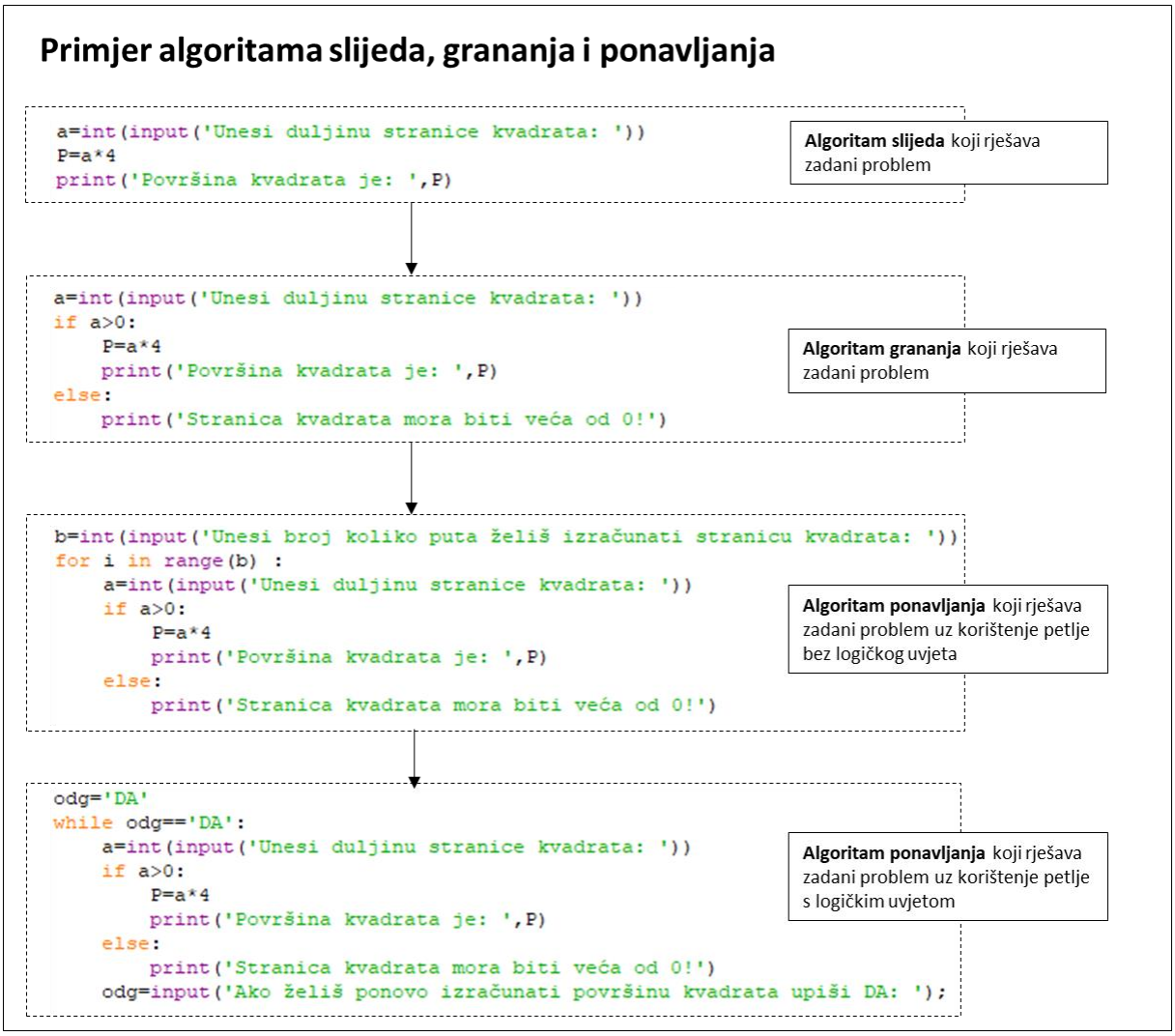


Slika 2.13 - Primjer uvođenja algoritama grananja u vizualnom-blokovskom programskom jeziku

Na slici 2.13 je, kao i u prethodnom slučaju, prikazan primjer vizualnog-blokovskog programskog jezika Hour of code na primjeru igre. Nakon algoritama slijeda i ponavljanja algoritam grananja može se uvesti kroz ispitivanje nekog uvjeta u mikrosvijetu bez potrebe dodatnog uvođenja relacijskih operatora.

Kao što je već rečeno kod tekstualnih programskih jezika redoslijed uvođenja je slijed-grananje-ponavljanje. Dodatni problem pri primjeni algoritma slijeda u programskom jeziku je potreba uvođenja većeg broja naredbi potrebnih za realiziranje algoritama u odnosu na blokovske jezike. Tako su za primjenu algoritma slijeda potrebne naredbe za ulaz, izlaz, varijable, pridruživanje vrijednosti varijabli, cjelobrojni tip podataka te aritmetičke operacije. Za primjenu algoritma grananja potrebno je još uvesti i relacijske operatore. Tipični primjeri za uvođenje osnovna tri algoritma u programskom jeziku Python i potrebne programske strukture prikazane su na slici 2.14.

Kao što se može vidjeti iz primjera zadataka, osim same jednostavnosti blokovskih jezika u odnosu na tekstualne, lako je ostvariv pomak konteksta od rješavanja matematičkih problema, tipičnih za proceduralno programiranje u tekstualnom programskom jeziku, prema igrama, bilo kroz igranje bilo kroz programiranje u blokovskim programskim jezicima.



Slika 2.14 - Primjer uvođenja algoritama grananja u tekstualnom programskom jeziku prema proceduralnom pristupu

Kako je osim samog programskog jezika bitan i kontekst programiranja u sljedećem poglavlju opisuju se igre u kontekstu učenja.

2.5 Učenje i igre

Psiholozi su davno prepoznali igranje kao važnu aktivnost u kognitivnom razvoju i učenju. Piaget je opisao igru kao integralni dio kognitivnog razvoja djece (Piaget & Play, 1962). Jedan od načina utjecaja igre na kognitivni razvoj djece je aktiviranje shema kod kojeg djeca prenose iskustva iz igre u realan svijet. Iz navedenog je jasno kako se igre mogu lako iskoristiti u edukacijske svrhe, ali na pažljiv način kako ne bi, umjesto najboljeg, dobili „najgore“ od obaju svijetova: igara i učenja (Papert, 2010). Važno je razumjeti kako računalne igre utječu na kognitivni razvoj i učenje (Plass, Homer, & Kinzer, 2015).

Prijelazom na digitalno doba, u odnosu na prethodna, promijenio se način učenja i obrade informacija. Kako su prije digitalnog doba od medija bile dostupne samo knjige i pisani materijali, a kasnije i televizija, koristilo se linearno, deduktivno zaključivanje (Brown, 2000). Zbog velikog utjecaja i uporabe digitalnih tehnologija djeca odrasla u digitalnom dobu imaju sposobnost obrade više informacija istovremeno (eng. paralell processing), sposobnost obavljanja više poslova istovremeno (eng. multitasking), više koriste induktivno zaključivanje, žele brzu i čestu interakciju sa sadržajem i imaju izvanredne vizualne sposobnosti što su sve karakteristike učenja temeljenog na računalnim igrama (Brown, 2000; Oblinger & Oblinger, 2005; Prensky, 2003). Iz navedenog se može zaključiti kako je kod digitalnih urođenika dominantno konstruiranje novih ideja uporabom bilo čega što je pri ruci, što je način razmišljanja u stručnoj literaturi poznat pod nazivom „bricolage“ (Brown, 2000). „Bricolage“ mislioci rješavaju probleme po principu pokušaja i pogrešaka, istraživanja, eksperimentiranja što je utjecajem interneta i digitalne tehnologije postao dominantan, čak i poželjan način razmišljanja (Turkle & Papert, 1992). Iz razmatranih istraživanja proizlazi hipoteza mogućeg budućeg istraživanja kako se upravo u navedenom krije jedan od razloga za smanjenje zanimanja za programiranje koje zahtijeva analitičnost i sustavni pristup rješavanju problema koja nije u skladu s „bricolage“ načinom razmišljanja. Osnovne stilove programiranja odnosno razmišljanja moguće je podijeliti na „tvrdi pristup“ (eng. hard) i „meki pristup“ (eng. soft) koji uključuju nekoliko razlika u pristupu poučavanju programiranja (Sedelmaier & Landes, 2014). Tvrdi pristup podrazumijeva logički, hijerarhijski pristup rješavanju problema te pristup „razmisli dvaput kodiraj jednom“ (Bailey, 2005), dok meki pristup uključuje fleksibilan, ne-hijerarhijski pristup rješavanju problema i otvorenost prema eksperimentiranju. Tvrdi pristup, od učenika, zahtijeva visoku razinu apstrakcije dok meki preferira pristup od konkretnog prema apstraktnom. Istraživanja pokazuju kako žene preferiraju meki pristup dok muškarci preferiraju tvrdi pristup programiranju (Turkle & Papert, 1992). Iz navedenog se može zaključiti kako drugačiji, „mekši“ pristup programiranju ne samo da može ohrabriti digitalne urođenike na učenje programiranja već se može ujednačiti i stav prema programiranju u odnosu na spol. Neka istraživanja su pokazala kako se programiranjem igara i pričanjem priča (eng. storytelling) u programskom jeziku Alice u srednjoj školi djevojčice motiviraju za programiranje (Kelleher et al., 2007).

Područje istraživanja o utjecaju korištenja računalnih igara za učenje nije dovoljno istraženo kako bi se moglo zaključiti ima li korištenje igara ima utjecaj na efekte učenja ili ne (Wouters, der Spek, & Van Oostendorp, 2009). Osim istraživanja na tu temu potrebno je odgovoriti na

pitanje kako uključiti računalne igre u nastavni proces kako bi se maksimizirao učinak učenja (Van Eck, 2006). Iako je posljednjih nekoliko godina uključivanje igara u postupak poučavanja postala aktualna tema o kojoj se puno piše, postoji i bojazan da se lošom primjenom izgubi sve potencijalno dobro. Može se reći kako se nešto takvo i dogodilo, od očekivanja da se uzme najbolje od oba svijeta (igara i učenja) dogodio se obrat koji je rezultirao usvajanjem najgoreg od obaju svjetova (Papert, 2010).

Istraživačko pitanje koje proizlazi iz diskusije je: Koje uvjete igra mora zadovoljavati kako bi bila zabavna te istodobno tijekom igre omogućila odvijanje procesa učenja? Prema Piagetovoj teoriji kognitivnog razvoja znanje se asimilira ili akomodira (Piaget, 1952). Proces asimilacije je jednostavniji, a podrazumijeva potpuno novo znanje koje se prihvaća dok je akomodacija puno zahtjevniji proces pri kojem se novo znanje mora uklopiti u postojeće s kojim može biti u konfliktu pri čemu može doći do kognitivne neravnoteže (Piaget, 1952) što je važno za kognitivni razvoj. Igrom je lako postići stanje kognitivne neravnoteže koje u pravoj mjeri potiče učenje. Igranje igara potiče ciklus postavljanja hipoteza, testiranja i ispitivanja (Van Eck, 2006).

Osim kognitivne neravnoteže tijekom učenja se javljaju i kognitivni procesi istraživanje i svjesnost. Neravnoteža, istraživanje i svjesnost su povezani procesi kojima se dolazi do viših razina razmišljanja i rješavanja problema (Yuen & Liu, 2010).

Prejednostavne igre neće u dovoljnoj mjeri potaknuti sudjelovanje te će brzo postati dosadne. Nasuprot tome preteška igra može kognitivno preopteretiti igrača koji će brzo odustati. Napredovanje u igri mora pratiti ciklus neravnoteže, istraživanja i svjesnosti.

2.5.1 Igre

Igre su, u današnje vrijeme, vrlo prilagodljivi medij koji se može prilagoditi svakoj tehnologiji od neolitika do visoke tehnologije. Postoji velik broj vrsta igara: ratne igre, slagalice, logičke, strateške, društvene i druge (Kirriemuir & McFarlane, 2004).

Potrebno je odrediti što je svim igrama zajedničko, definirati pojam igre i uvjete koje mora zadovoljiti kako bi bila uspješna. Ne postoji jedinstvena definicija igre već postoji više interpretacija tog pojma (Smed, Hakonen, & others, 2003). Jedna od njih je Juulsova definicija koja igru predstavlja u dvije dimenzije. Jedna dimenzija odnosi se na perspektive, a to su: *igra* (svojstva sustava koja definiraju pravila igre), *igrač* (relacija između igre i igrača) i *svijet* (relacija između igre i svijeta) (Juul, 2018). Kada govorimo o igrama koje bi osim zabave trebale igrati i obrazovnu ulogu ovom modelu možemo dodati i *dimenziju znanja* koja predstavlja relaciju između igrača i obrazovne svrhe igre. Druga dimenzija definira što igra

mora zadovoljavati: *pravila*, *mjerljivi ishod*, *vrednovanje ishoda*, *trud igrača* (igra je izazov), *igrač koji je vezan za ishod* (npr. sretan je ako je ishod pozitivan, a nezadovoljan je ako je ishod loš), te *prenosive posljedice* (igra se može igrati sa ili bez posljedica u stvarnom svijetu) (Juul, 2018). Iz svega navedenog igru bi mogli definirati kao formalni sustav temeljen na pravilima s mjerljivim ishodom i varijablama gdje je različit ishod vezan uz različite vrijednosti, a u kojem igrač ulaže napor kako bi utjecao na ishod, te želi ostvariti cilj igre, dok je naučeno (posljedice) za vrijeme igranja po izboru prenosivo u stvarni svijet. Kako bi igru mogli smatrati pedagoškim alatom osim navedenih značajki igra bi još trebala imati mjerljivi pedagoški ishod.

Tablica 2.4 - Uvjeti koji definiraju igru

	igra	igrač	svijet	znanje
1. pravila	x			
2. mjerljivi ishod	x			x
3. vrednovanje ishoda	x			x
4. trud igrača		x		
5. igrač koji je vezan za ishod		x		
6. prenosive posljedice			x	x

U tablici 2.4 prikazani su uvjeti koji definiraju igru. Navedeni uvjeti koje igra treba zadovoljiti nisu vezani uz iste razine pa tako uvjeti navedeni pod 1, 2 i 4 opisuju svojstva igre kao formalnog sustava, 3 definira vrijednosti za moguće ishode igre, 4 i 5 opisuju relaciju između igrača i sustava, a 6 definira vezu između igre i ostatka svijeta.

Costikyan definira igre vrlo jednostavno: "bez cilja to je igračka (eng. „*without a goal, it is a 'toy.*'") te tvrdi da je igra skup definiranih pravila dodanih igrački (Curtis & Lawson, 2002). On je definirao sljedeće elemente igre: cilj, borba (ne nužno natjecateljski oblik što uključuje i suradničke igre), borba sa samim sobom (npr. slagalice, i slični izazovni zadaci koji ne moraju uključivati druge igrače), struktura, smislenost i interaktivna zabava.

Uvjete koje igra treba zadovoljiti kako bi bila zabavna prema LeBlancovoj taksonomiji su: osjet, fantazija, narativnost, izazov, druženje, otkrivanje, izražavanje i podčinjenost (Hunicke, LeBlanc, & Zubek, 2004).

2.5.2 Igre za učenje i poučavanje programiranja

Kako je već navedeno, igre su bitan faktor u kognitivnom razvoju djece. Odrastanjem, djeca igraju sve apstraktnije, socijalnije i više simboličke igre (Piaget & Play, 1962) pri čemu se može uočiti veza s programiranjem kao apstraktnom disciplinom te primjerenosti načina uvođenja

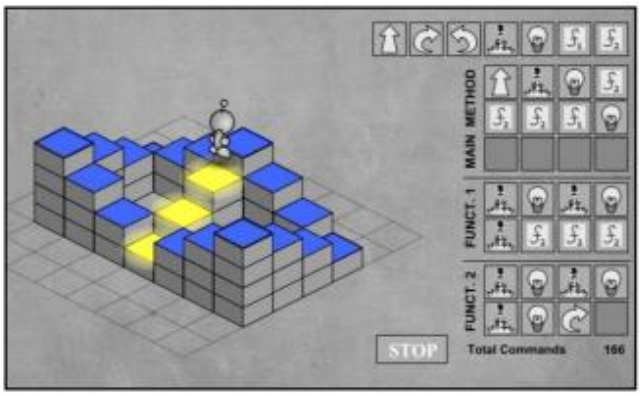
programiranja ovisno o dobi učenika. Na ovaj način, čini se lako povezati programiranje kao aktivnost koja se može učiti kroz igru, programiranjem ili igranjem istih.

Iako su istraživanja pokazala kako se vještine poput rješavanja problema povećavaju i čak prenose tijekom igranja, teško je takve vještine prenijeti izvan igre (Egenfeldt-Nielsen, 2006). Curtis i Lawson su ustanovili tek umjereni prijenos vještine rješavanja problema (Curtis & Lawson, 2002). Prenesene vještine su, čini se, ograničene na nisku razinu prijenosa. Takva istraživanja pokazuju kako nedostaje posredovani transfer (Dann, Cosgrove, Slater, Culyba, & Cooper, 2012), odnosno, igra i (ili) pedagoški pristup u kojem bi se vještine i znanja stečene u igri mogle prenijeti u stvarni svijet, odnosno na koncepte programiranja koje su bile cilj učenja. Postoje dvije kategorije posredovanog transfera. Prva kategorija je „premoštenje“ (eng. „bridging“) u kojem nastavnik pomaže učenicima premostiti put od konteksta u kojem je koncept naučen do novog potencijalnog konteksta. Druga kategorija je „prigrljivanje“ (eng. „hugging“) u kojoj nastavnik kreira situaciju učenja sličnu situaciji u kojoj se transfer očekuje (Dann, Cosgrove, Slater, & Culyba, 2012a). Od igara se ponekad previše očekuje, očekuje se igra u kojoj će učenik igrajući dobiti svo očekivano znanje. Na učitelju je omogućiti prijenos znanja iz igre u željene ciljeve učenja, dakle učitelj je taj koji mora pomoći učeniku premostiti put od igre prema željenom cilju učenja. Primjer za navedeno je istraživanje provedeno igrom *Crystal Island* koje je izvršeno u srednjoj školi *Ocoee* na Floridi gdje su učenici kroz igru učili koncepte iz mikrobiologije zapisivanjem svojih otkrića i hipoteza kroz igru (Ash, 2011). Prestankom igranja ne prestaje učenje, upravo tada je najvažnije vrijeme kada govorimo o korištenju igre u učenju jer treba povezati koncepte naučene u igri i staviti ih u željeni kontekst u čemu učitelj ima ključnu ulogu (McClarty et al., 2012). Provedena meta-analiza na 17 studija pokazala je nužnost podrške učitelja kako bi se koncepti naučeni u igri prenijeli u druge kontekste (Ke, 2011). Iz navedenog se zaključuje kako pedagoška primjena igara u učenju i poučavanju neće zamijeniti učitelje i učionice, ali može zamijeniti neke udžbenike ili laboratorije, te kako igre neće postati nastavna metoda već ih treba koristiti kao nastavno sredstvo.

Područje izrade igara u svrhu poučavanja programiranja i računalne znanosti pomoću igara je relativno novo pa nema dovoljno empirijskih dokaza o učenju pomoći igara u tom području što ukazuje na područje otvoreno za istraživanje i proučavanje. Neke od igara koje se koriste za poučavanje programiranja su LightBot (“Lightbot,” n.d.) i RoboZZle (“RoboZZle,” n.d.). Navedene igre spadaju u skupinu igara slagalica (puzzle) u kojima se naredbe slažu povuci i pusti (eng. *drag and drop*) tehnikom kako bi se pokrenuo robot i izvršio zadani cilj. Za

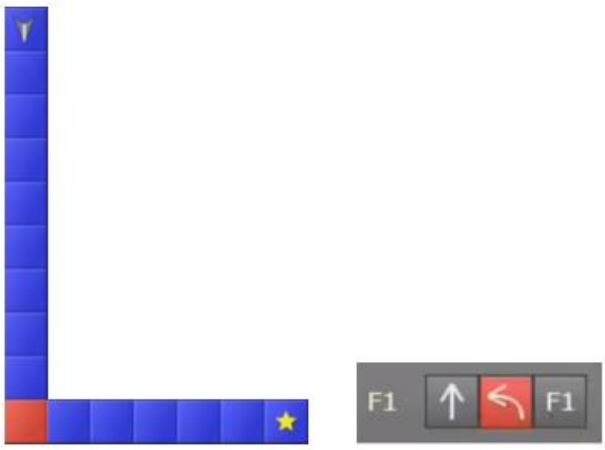
rješavanje zadanog problema igrač mora ponuditi semantički točno rješenje, a u isto vrijeme ne postoji problem sa sintaksom. Napredovanjem kroz igru igrače se potiče na učinkovito korištenje naredbi i na rješavanje sve kompleksnijih problema.

Cilj igre LightBot je robotom, kojim igrač upravlja, posjetiti sva plava polja i osvijetliti ih. Igra počinje s jednostavnim mapama i naredbama (naprijed, lijevo, desno, skoči, uključi lampicu) da bi napretkom kroz igru dobivao sve složenije zadatke što podrazumijeva ograničenost broja naredbi, uvođenje funkcija i rekurzije. Na slici 2.16 prikazan je primjer LightBot okruženja u kojem se koriste funkcije.



Slika 2.15 - LightBot okruženje

Slično je i kod igre RoboZZle, kod koje ima manje naredbi (naprijed, lijevo, desno), ali je podržana mogućnost grananja korištenjem boja za uvjetno izvođenje naredbi. Na slici 2.17 je prikazan primjer labirinta s korištenjem grananja.



Slika 2.16 - RoboZZle okruženje

Početkom 2013. godine pojavila se Hour of Code inicijativa u SAD-u u svrhu promicanja računalne znanosti kod mlađih naraštaja kako bi se demistificiralo cijelo područje (C. Wilson,

2015). U inicijativu su se uključile mnoge poznate ličnosti među kojima je tadašnji predsjednik SAD-a Barack Obama, osnivač Facebooka Mark Zukerberg, osnivač Microsofta Bill Gates i mnoge druge poznate osobe i to ne samo iz tehničkog područja. Tijekom ovih godina broj sudionika i država te jezika na kojima su tečajevi dostupni se eksponencijalno povećavao. Svi tečajevi potpuno su besplatni za korištenje. Između ostalih, tečajevi su dostupni i na hrvatskom jeziku. Platforma na kojoj se svi tečajevi nalaze je Code Studio (C. Wilson, 2015) na kojoj se nalazi cijeli niz tečajeva i aktivnosti dostupnih po kategorijama za određene uzraste i različite tehnologije. Tečajevi se oslanjaju na poznate likove iz animiranih filmova i računalnih igara kao što su: Angry Birds, Frozen, Minecraft i brojni drugi. Početni tečajevi programiranja za osnovnoškolski uzrast, ali i za druge početnike u programiranju, oblikovani su kao vizualno-blokovski programski jezici. U svakom tečaju obrađuju se tri osnovna algoritma: slijed, grananje i ponavljanje kroz manje scenarije. Na slici 2.18 se vidi primjer iz *Hour of Code Angry Bird* aktivnosti u kojoj je cilj kroz 20 razina obraditi spomenute osnovne algoritme.

Slika 2.17 - Hour of Code: Angry Bird

Angry Bird je jednostavna i popularna igra s kojom je upoznata većina osnovnoškolaca te su im samim time okruženje i kontekst već privlačni. Ideju igre nije potrebno posebno objašnjavati jer je cilj dovesti *Pticu (Angry Bird)* do polja na kojem se nalazi *Svinja*. Na slici 2.18 je prikazan primjer uvježbavanja algoritma slijeda kod kojeg se moraju uvesti naredbe odgovarajućim redoslijedom. Na sučelju su dostupne samo potrebne naredbe čime se učenicima ne odvraća pažnja sa suvišnim naredbama. Osim toga sugeriran je broj blokova potrebnih za rješavanje programa što je jako važno kad se dođe do algoritma ponavljanja jer učenici imaju tendenciju riješiti problem algoritmom slijeda koji im je jednostavniji.

Ovakve inicijative zaista su polučile uspjeh kad je u pitanju popularizacija računalne znanosti u svijetu što se može vidjeti na primjeru *Hour of Code* inicijative u kojoj je do sada sudjelovalo preko 38 milijuna učenika. Osim toga postoji veća količina materijala izrađenih za nastavnike kako bi se što više ovakvih sadržaja uklopilo u redovnu nastavu (Theodoropoulos, Antoniou, & Lepouras, 2017).

2.5.3 Programiranje igara ili igranje igara za učenje programiranja?

Računalne igre imaju veliku ulogu kao kulturološko artefakt u razvoju računalnih znanstvenika (Simmons, DiSalvo, & Guzdial, 2012). Međutim, iako se korištenje računalnih igara u svrhu poučavanja informatike pokazalo uspješnim u privlačenju, ali i ostajanju u području informatike na diplomskoj razini, samo igranje igara nije dovoljno za podizanje interesa za informatiku (DiSalvo & Bruckman, 2009). Moguće je na odgovarajući način uvesti niz aktivnosti koje uključuju igranje igara kao pedagoško sredstvo učenja. Na primjer, jedan od prvih koncepata koji se uvode u proceduralnom programiranju je koncept varijabli. Za uvođenje takvog koncepta nije potrebno uključiti učenike u neku veliku igru. Dovoljno bi bilo ponuditi gotove dijelove koda, eventualno s greškom, kako bi kroz aktivnosti unaprjeđenja igre ili traženja greški učili o određenom pojmu (Bayliss, 2009). Kako početnici imaju problema s analizom kôda, ovakve, manje, aktivnosti mogu pomoći u razvoju takvih sposobnosti.

Dakle, odgovor na pitanje naslova poglavlja „Programiranje igara ili igranje igara za učenje programiranja?“ nije jednostrano. Ako igre nisu pažljivo implementirane u svrhu učenja tada neće djelovati na edukacijsku domenu, što je cilj (Kafai & Burke, 2015). To je vjerojatno razlog zbog kojeg postoji nerazmjer u rezultatima učenja korištenjem igara (Kafai & Burke, 2015). Odgovor se vjerojatno krije u kombinaciji igranja i programiranja igara što spada u dva osnovna pristupa u korištenju igara u edukacijske svrhe: instrukcijski i konstrukcijski (Kafai, 2006; Kafai & Burke, 2015). Kod instrukcijskog pristupa u fokusu je igranje igara, dok je u konstrukcijskom stvaranje igara (Papert, 1993). Opet se postavlja pitanje koji je pristup primjeren za osnovnoškolski uzrast, s obzirom na činjenicu kako početnici nemaju nikakvo predznanje o konceptima programiranja pa samo učenje otkrivanjem (instrukcijski pristup) ne može voditi do usvajanja željenih koncepata programiranja. Vjerojatno je za osnovnoškolce početnike najprikladniji pristup kombinacija spomenutih pristupa koji može potaknuti učenje više od samog učenja otkrivanjem (Grover, Pea, & Cooper, 2015).

Dakle, važni faktori u izboru pristupa su: uzrast, aktivnosti te poučavani koncepti. Dok programiranje igara vjerojatno izaziva bolji i dugotrajniji proces učenja, igranje igara primjerenije je za uvođenje kompleksnih, apstraktnih pojmova koji se mogu prikazati po

principu od konkretnog prema apstraktnom. Na primjer, uvođenje pojma algoritma, te uvođenje osnovnih algoritama: slijeda, grananja, ponavljanja, koje se prema HNOS-u uvodi prije ikakvog učenja programiranja, vjerojatno je idealno kroz igranje nekih jednostavnih igara, kao na primjer *Hour of code* aktivnosti (*Angry birds, Lightbot* i slično). Kroz aktivnosti igranja takvih igara, učitelj može na jednostavan, djeci razumljiv i konkretan način, uvesti pojmove poput algoritma slijeda, grananja i ponavljanja. U daljnjem procesu nastave, igranjem neke igre može se definirati scenarij koji predstavlja algoritam za izradu igre, te kroz aktivnosti izrade same igre uvoditi nove pojmove programiranja.

Važno je naglasiti i potencijalne probleme pri korištenju igara u nastavi. Osnovna pogrešna predodžba o očekivanjima u korištenju igara u nastavi je da igre budu same sebi svrha, te da će učenici samo igranjem igre sami usvojiti željene pojmove učenja. Igre nisu same sebi svrha, ako se u nastavi ne koriste u procesu učenja tada njihova uporaba nije opravdana. Igre u nastavi trebale bi učitelju poslužiti kao nastavno sredstvo za uvođenje inače kompleksnih i apstraktnih pojmova. Nije za očekivati da će učenici samo igranjem igara doći do željenih ishoda učenja, učitelj je taj koji ima ključnu ulogu u procesu učenja korištenjem igara.

2.6 Prethodna istraživanja područja

Kako programiranje nije popularno među učenicima, odabir odgovarajućeg programskog jezika kao i kontekst programiranja može biti ključan za motivaciju i stav prema programiranju i računalnoj znanosti uopće. U isto vrijeme učenici bi trebali ovladati osnovnim programskim konceptima koji ne smiju biti zanemareni na račun motivacijskog faktora. Važno je pitanje i zainteresiranost djevojčica za učenje programiranja i računalne znanosti općenito. Svjetski trendovi pokazuju smanjenu zainteresiranost djevojčica za programiranje unatoč tome što se nisu pokazale nikakve razlike u sposobnostima programiranja između spolova (Kelleher et al., 2007). Stanje u Republici Hrvatskoj potvrđuje takve trendove što je pokazalo istraživanje provedeno na 1462 učenika osmih razreda diljem države. Rezultati su pokazali smanjeno zanimanje djevojčica za programiranje unatoč približno jednakoj brojnosti u pohađanju izbornog predmeta informatike (M. Mladenović et al., 2014). Osim toga rezultati su pokazali kako je programiranje jedna od najmanje omiljenih nastavnih cjelina iz kurikula (Žanko et al., 2014). Dominantna motivacija za upis izbornog predmeta informatike su postignuće i ekstrinzična motivacija što potvrđuje trendove koja su pokazala ostala istraživanja o nedovoljnoj intrinzičnoj motivaciji za učenje računalne znanosti (S. Mladenović et al., 2015).

Brojna istraživanja su pokazala kako se programiranjem igara, neovisno o spolu, povećava motivacija učenika te da se na taj način mogu usvojiti određeni koncepti programiranja kroz različita okruženja: korištenjem programskih jezika Alice (L. Werner, Campe, & Denner, 2012) (L. L. Werner, Campe, & Denner, 2005) (L. Werner, Denner, Bliesner, & Rex, 2009), Scratch (J. H. Maloney, Peppler, Kafai, Resnick, & Rusk, 2008) (Malan & Leitner, 2007) (A. Wilson, Hainey, & Connolly, 2013) i Kodu (Fowler & Cusack, 2011). Osim toga uočeno je kako korištenje ovakvih jezika potiče informatičku okretnost (L. Werner et al., 2009), te da učenici kroz rad u Scratchu kao prvom programskom jeziku u školi mogu usvojiti koncepte programiranja (Orni Meerbaum-Salant, Armoni, & Ben-Ari, 2013).

Osim navedenog pokazalo se kako se korištenjem animacija i/ili Scratcha može utjecati na „srednju trećinu“ učenika (Ben-Bassat Levy, Ben-Ari, & Uronen, 2003) (Armoni, Meerbaum-Salant, & Ben-Ari, 2015), odnosno na učenike koji imaju sposobnosti za savladavanje gradiva, ali trebaju poticaj. U ovom slučaju to može biti animacija ili vizualni programski jezik koji omogućuje učenicima iskustvo od konkretnog prema apstraktnom (Dann & Cooper, 2009b), odnosno stavlja ih se u kontekst posredovanog transfera (Dann, Cosgrove, Slater, & Culyba, 2012b) gdje kroz konkretan doživljaj animacije mogu premostiti put prema apstraktnom shvaćanju učenog pojma.

Većina istraživanja u ovom području se, između ostalog zbog nedostatka primjerenih programa u školama, provodi u klubovima ili ljetnim kampovima, dok je manji dio proveden u školskom okruženju. Prednost istraživanja u klubovima i ljetnim kampovima, kao neformalnim okruženjima, je što učenici u takvom okruženju mogu učiti neometano i bez čvrstog vremenskog ograničenja istraživati vlastite interese (Peppler & Kafai, 2007). Nedostatak takvih okruženja je visoka motivacija sudionika, s obzirom da je sudjelovanje dobrovoljno, čime se ne dobiva reprezentativan uzorak učenika. Problem istraživanja u školskom okruženju je nepostojanje predmeta kao dijela nastavnog plana i programa te teškoće u provedbi zbog različitih ograničenja kurikula te vremena izvođenja. Kada su u pitanju istraživanja na osnovnoškolskoj razini stanje je još lošije zbog nedostataka odgovarajućih kurikula te zbog složenosti provedbe istraživanja. Zbog navedenog većina istraživanja u školskom okruženju provedena je na preddiplomskoj i diplomskoj razini.

2.6.1 Istraživanja provedena u vizualnim-blokovskim programskim jezicima

Jedno od prvih, većih istraživanja na temu Scratcha provedeno je u klubu pokraj Los Angelesa u kojem su sudjelovali učenici od 8 do 18 godina s ukupno 536 projekata u Scratchu. Istraživači su proučavali korištenje Scratcha za izradu igara, animacija, glazbenih videa itd. Rezultati su pokazali kako su učenici savladali osnovne programske koncepte u Scratchu, kojeg su doživjeli zabavnim za korištenje (J. H. Maloney, Peppler, Kafai, Resnick, & Rusk, 2008). Cilj sljedećeg istraživanja, koje se provodilo u ljetnoj školi, bila je usporedba stava i ishoda učenja učenika šestih razreda pri korištenju programskih jezika Logo i Scratch. Rezultati tog istraživanja su pokazali kako se korištenjem Loga kod učenika potakao razvoj samopouzdanja i interesa za programiranje, kao i razumijevanje koncepta petlje. Korištenjem Scratcha se pokazao bolji napredak u ishodima učenja kad je u pitanju uvjetno izvršavanje programa (Lewis, 2010).

Brojna druga istraživanja su pokazala kako vizualni-blokovski programski jezici mogu utjecati na afektivnu i kognitivnu domenu učenika. Tako su, prema rezultatima istraživanja Xinogalosa i ostalih, studenti preddiplomskog studija između BlueJ, objectKarel, Scratch, Alice i MIT App Inventor okruženja odabrali Scratch kao naučinkovitiji blokovski jezik za uvod u programiranje na svim obrazovnim razinama (Xinogalos, Satratzemi, & Malliarakis, 2017). Također, u istraživanju koje su proveli Weintrop i Wilensky, učenici srednjih škola smatraju kako je blokovski programski jezik jednostavniji od tekstualnih, ali kako u isto vrijeme imaju manje mogućnosti za razvoj složenijih programa (Weintrop & Wilensky, 2015). Važno je napomenuti kako su učenici koji su sudjelovali u istraživanju imali prethodno iskustvo programiranja u tekstualnom programskom jeziku što je u skladu s otkrićem Malana i Leitnera u kojem se otkrilo

da je manji broj učenika kojima se Scratch nije svidio imalo prethodno iskustvo programiranja u tekstualnom programskom jeziku. Navedeno istraživanje provedeno je na Harvardskoj ljetnoj školi, a cilj je bio utvrditi stav učenika prema programiranju. Učenici su koristili Javu nakon korištenja Scratcha. Rezultati su pokazali da učenici doživljavaju Scratch zabavnim i imali su pozitivan stav prema programiranju nakon korištenja Scratcha. (Malan & Leitner, 2007).

Prema pregledu literature o obrazovanju u računarstvu na osnovnoškolskoj razini istraživači su zaključili kako početnici u programiranju imaju brojne probleme u programiranju u tekstualnim programskim jezicima, ali i kako takve probleme mogu prebroditi osjećajem da rade u profesionalnom alatu (Garneli, Giannakos, & Chorianopoulos, 2015). Programiranje u tekstualnim programskim jezicima može privući učenike s višim sposobnostima rješavanja problema što dolazi do izražaja u istraživanjima u neformalnim okruženjima kao što su kampovi, tečajevi, ljetne škole i slično. Pitanje je koliko istraživanja u neformalnom pružaju pravu sliku jer je samo sudjelovanje dobrovoljno, što znači da je istraživana populacija unaprijed motivirana za učenje programiranja. U školskom okruženju, koje je više realistično, nalaze se učenici različitih sposobnosti i sklonosti određenim područjima. Učenicima je školsko okruženje prirodno, što se odnosi na fizičke (učionica), ljudske (učitelj) i programske (kurikul) postavke (Cohen, Manion, & Morrison, 2011). Kako je istraživanja u školskom okruženju teže provesti s višom razinom znanstvene pouzdanosti, tako je njihov broj manji (Grover, 2017; Kölling & McKay, 2016).

Neka istraživanja su provedena u više realnom, školskom okruženju. U jednom od takvih istraživanja rezultati su pokazali kako se uporabom Loga potiče kreativnost i sposobnost rješavanja problema među učenicima petih razreda u Katoličkoj osnovnoj školi pokraj Jakarte (Pardamean, Evelin, & Honni, 2011). Istraživanje provedeno u Škotskoj, također u školskom okruženju, pokazalo je kako su učenici programiranjem u Scratchu kreirali igre korištenjem programskih koncepata (A. Wilson et al., 2013). Zapaženija istraživanja provedena su u srednjoj školi u Izraelu u kojima su istraživači mjerili ishode učenja prema kombiniranoj SOLO i Bloomovoj taksonomiji. Prema rezultatima prvog takvog istraživanja, provedenim u dva razreda, pokazalo se kako učenici mogu savladati programske koncepte korištenjem Scratcha, te da Scratch može poslužiti kao platforma za poučavanje i učenje računalne znanosti. Unatoč tome, uočene su neke poteškoće pri učenju inicijalizacije, varijabli i istodobnog izvršavanja programa. U kontekstu SOLO taksonomije, za promatrane koncepte, potrebna je barem sposobnost multi-strukturalnog razmišljanja a, u nekim slučajevima, čak i sposobnost povezivanja. Za internalizaciju ostalih koncepata potrebna je osnovna, unistrukturalna

sposobnost (Orni Meerbaum-Salant et al., 2013). Tijekom ovog istraživanja uočeni su i mogući problemi koji se odnose na loše navike programiranja kao što su „bottom-up“ programiranje ili ekstremno sitno raščlanjeno programiranje (O Meerbaum-Salant, Armoni, & Ben-Ari, 2011). Objašnjenje ove pojave može biti programiranje temeljeno na scenariju koje se smatra „prirodnim“ pa takva pojava ne bi trebala biti zabrinjavajuća (Gordon, Marron, & Meerbaum-Salant, 2012). Sljedeće istraživanje istih istraživača u istoj školi provedeno je među pet razreda od kojih su učenici u tri razreda imali prethodno iskustvo programiranja u Scratchu i to tijekom prethodno navedenog istraživanja. Takvi razredi su pripadali eksperimentalnoj skupini dok su dva razreda, u kojima učenici nisu imali prethodnog iskustva u programiranju, predstavljali kontrolnu skupinu. Korištena metodologija uključivala je kvalitativnu i kvantitativnu analizu. Kvantitativna analiza pokazala je statistički značajnu razliku za usvajanje pojma petlje. Osim toga na završnom testu uočene su statistički značajne razlike na najvišoj razini kombinirane SOLO i Bloomove taksonomije koja se odnosi na relacijsko kreiranje. Kako se pojam petlje smatra teškim za razumijevanje već je i dobiveni rezultat istraživanja dovoljan razlog za uporabu Scratcha pri poučavanju programiranja. U kvalitativnoj analizi, iz razgovora sa učiteljima, uočeno je kako je povećana učinkovitost učenika te smanjeno vrijeme potrebno za objašnjavanje novih pojmova, što učiteljima osigurava više vremena za pomoć učenicima kojima je to potrebno, dok ostali učenici imaju priliku samostalno raditi u Scratchu. Kako je istraživanje provedeno na izbornom predmetu uočeno je i dupliranje upisa nakon korištenja Scratcha u nastavi što ukazuje na veću motivaciju za ovakvo učenje programiranja (Armoni et al., 2015). Još jedno važno zapažanje iz istog istraživanja je najveći utjecaj na „srednju trećinu“ učenika, što je u skladu s rezultatima istraživanja Ben-Bassat Levy i ostalih koje je pokazalo kako su od korištenja sustava za animaciju Jeliot najviše profitirali upravo takvi učenici (Ben-Bassat Levy et al., 2003).

Istraživači u Turskoj su 2014. istraživali utjecaj na sposobnost rješavanja problema nakon programiranja u Scratchu među učenicima petih razreda osnovne škole. Nisu potvrđene statistički značajne razlike u završnim testovima, ali su istraživači ostavili mogućnost dobivanja različitih rezultata u različitim kontekstima i dizajnu istraživanja. Otkriveno je i poboljšanje učeničkog samopouzdanja pri rješavanju problema, ipak samo okruženje nije dovoljno kako bi učenje bilo efektivno (Pardamean et al., 2011). Ipak vizualno okruženje može poslužiti za posredovani transfer prema „pravom“ programiranju, ako se koristi na odgovarajući način. Upravo su takvi rezultati dvogodišnjeg istraživanja na preddiplomskoj razini kod kojeg su se pratili rezultati kontrolne skupine koja je učila samo Java programski jezik i eksperimentalne

skupine koja je učila Alice pa Javu, gdje je programski jezik Alice služio za posredovani transfer od konkretnog doživljaja objekata u Alice prema apstraktnim u Javi. Rezultati su pokazali statistički značajnu razliku u rezultatima završnih testova kolegija pri čemu je eksperimentalna skupina bila uspješnija od kontrolne te se tako potvrdila hipoteza kako se vizualno programsko okruženje može uspješno koristiti kao medij za posredovani transfer pri usvajanju, inače kompleksnih, programskih pojmova (Dann, Cosgrove, Slater, & Culyba, 2012b) te kako je pristup poučavanju programiranja od konkretnog prema apstraktnom (Dann & Cooper, 2009b) učinkovit.

2.6.2 Istraživanja na temu igara u poučavanju programiranja

Kafai i Burke (Kafai & Burke, 2015) su dali pregled istraživanja kroz zadnjih 20 godina na temu aktivnosti temeljenih na igrama u različitim okruženjima (formalnim i neformalnim) provedenim na različitim uzrastima učenika. Prema rezultatima pregleda područja igre su poticajno okruženje za učenje koncepata računalne znanosti bez obzira na dob učenika, korišteni programski alat (Scratch, Alice, Greenfoot, itd.), okruženje učenja (formalno ili neformalno) koje pridonose širenju perspektive o informatici i STEM području općenito.

U Izraelu je tijekom tri godine (od 2011. do 2013.) u srednjoj školi provedeno istraživanje na uzrastu od sedmog do devetog razreda prema novom kurikulu iz računalne znanosti koristeći Scratch. Kao rezultat istraživanja napravit će se promjene u načinu poučavanja koncepta petlje koje će se poučavati izradom igre. (Zur-Bargury, Pârv, & Lanzberg, 2013). Rezultati ovog istraživanja potvrđeni su u drugom istraživanju provedenom u Sjevernoj Kaliforniji među 26 učenika od sedmog do osmog razreda koji su sudjelovali u izbornom predmetu „Computers“ (Grover, Cooper, & Pea, 2014).

Istraživanja su, između ostalog, pokazala kako programiranje računalnih igara pozitivno utječe na stav, samopouzdanje, užitak i motivaciju učenika u odnosu prema matematici (Ke & Fengfeng, 2014) i programiranju (M. Mladenović, Rosić, & Mladenović, 2016). Kada su u pitanju kognitivni stilovi učenika otkriveno je kako uporaba igara u nastavi najviše odgovara introvertima i prosudilačkim stilovima učenika (Theodoropoulos et al., 2017).

U Finskoj je provedeno longitudinalno istraživanje na uzrastu 12-16 godina. Kroz tri godine pratili su se učenici koji su pohađali jednotjedne ljetne tečajeve programiranja igara. Istraživanje je pokazalo kako je većina učenika nastavila s programiranjem, te da se njihovo zanimanje za računalnu znanost povećalo, osim toga su s programiranjem igara djevojčice bile podjednako zadovoljne kao i dječaci (Lakanen, Isomöttönen, & Lappalainen, 2012).

Među studentskom populacijom provedeno je istraživanje u sklopu kolegija programiranja 3D igara i to za studente koji studiraju umjetnost, jer se u izradi igara isprepliću umjetnost i programiranje, odnosno lijeva i desna strana mozga. Rezultat kolegija su deseci igara različitih žanrova rađeni kroz projekte u timovima koji su brojili od 3 do 6 studenata. Igre su izrađene pomoću više alata poput Macromedia Directora, Flasha, Jave, Virtoolsa i Quest3Da. Studentske igre bile su jako dobre, a često i bolje od igara koje su izradili studenti računalne znanosti što se objasnilo činjenicom kako studenti studija umjetnosti posjeduju bolje vještine potrebne za izradu sadržaja same igre (Tsai et al., 2006).

Istraživanje u kojem se primijenio početni pristup učenja programiranja programirajući igre (eng. game first approach) pokazalo je kako se takvim pristupom poboljšalo znanje osnovnih programerskih koncepata, povećao ostanak na studiju računalne znanosti te kako je povećano ukupno zadovoljstvo studenata (Leutenegger & Edgington, 2007).

Unatoč uglavnom pozitivnim rezultatima istraživanja o utjecaju igara na učenje, postoje razne debate o korištenju igara u poučavanju, pri čemu je upitno je li, i u kojoj mjeri učinkovito koristiti igre za učenje. Mali broj istraživanja proveden je s ciljem utvrđivanja utjecaja uporabe igara u nastavi te njihove učinkovitosti na učenje. Većina istraživanja odnosi se na utjecaj korištenja igara na stav i motivaciju učenika (Ke, 2011). Jedan od razloga malog broja provedenih istraživanja je i otežana mogućnost mjerenja nakon korištenja igara u nastavi zbog većeg broja varijabli koje utječu na učinkovitost učenja. Utjecaj igre na učenje je jedinstven i višestruk jer utječe na kombinaciju motivacije, razinu sudjelovanja, mjesto sudjelovanja, prilagodljivosti, suradnje te simulacije. Zbog svega navedenoga teško je zaključiti koji je utjecaj same igre te je teško osmisliti instrument kojim bi se izmjerila učinkovitost na samo učenje. Zbog navedenih utjecaja igre na učenje, buduća istraživanja ne bi se trebala baviti pitanjem utječu li igre na učenje već kako što učinkovitije koristiti igre za učenje (McClarty et al., 2012).

3 METODOLOGIJA ISTRAŽIVANJA

U ovom poglavlju opisana je metodologija istraživanja što uključuje opis pojedinosti o strukturi i provedbi istraživanja.

3.1 Predmet i cilj istraživanja

Predmet ovog istraživanja je proučavanje različitih pristupa poučavanju programiranja početnicima u osnovnoj školi u okvirima zadanog nastavnog plana i programa. Poučavanje i učenje programiranja je izuzetno zahtijevno, posebno za početnike, a naročito osnovnoškolce jer zahtijeva visoku razinu apstrakcije. Način poučavanja programiranja uglavnom se temelji na rješavanju matematičkih problema u tekstualnim programskim jezicima što dodatno smanjuje motivaciju učenika za učenjem programiranja. Iz navedenog proizlazi problem istraživanja. Kako su nastavnim planom i programom obuhvaćene teme, ali ne i programski jezici, poučavanje se može provoditi i na drugačiji način od „tradicionalnog“. To se prije svega odnosi na izbor programskog jezika, ali i konteksta programiranja. Navedeni različiti pristupi u poučavanju programiranja u ovom istraživanju odnose se na korištenje samog programskog jezika pri čemu se uspoređivalo uporabu vizualno-blokovskih i tekstualnih programskih jezika. Osim toga istraživao se i utjecaj pomaka konteksta programiranja s „klasičnog“ pristupa temeljenog na rješavanju matematičkih zadataka prema poučavanju programiranja programiranjem igara.

Cilj istraživanja je utvrditi kako na kognitivnu i afektivnu domenu učenika utječe primjena poučavanja oblikovanjem igara u vizualnom programskom jeziku na usvajanje i razumijevanje osnovnih programskih koncepata kod programera početnika u osnovnoj školi.

3.2 Istraživačka pitanja

Temeljem navedenog cilja istraživanja proizašla su sljedeća istraživačka pitanja:

1. Utječe li izbor programskog jezika i konteksta rješavanja problema na uspjeh u rješavanju problema programiranjem?
2. Utječe li izbor programskog jezika i konteksta rješavanja problema na motivaciju za programiranje?

Temeljem istraživačkih pitanja postavljene su sljedeće hipoteze:

1. Učenici će uspješnije savladati osnovne koncepte programiranja korištenjem vizualno-blokovskog programskog jezika Scratch u odnosu na vizualno-tekstualni programski jezik Logo.

2. Postignuće učenika predmetne nastave petih razreda osnovne škole, s obzirom na sposobnosti rješavanja problema, bit će veće ako se koristi poučavanje oblikovanjem igara u Scratchu u odnosu na klasičan način poučavanja koji se primjenjuje u tekstualnom programskom jeziku Python proceduralnim pristupom.
3. Postignuće učenika predmetne nastave osnovne škole, s obzirom na odabrani programski jezik, bit će veće poučavanjem oblikovanjem igara u vizualnom-blokovskom programskom jeziku Scratch, u odnosu na vizualno-tekstualne programske jezike Logo ili Python (uz korištenje kornjačine grafike).
4. Početnici u petom razredu osnovne škole postići će bolje rezultate koristeći vizualno-blokovski programski jezik Scratch od učenika istog uzrasta koji koriste vizualno-tekstualne programske jezike Logo ili Python (uz korištenje kornjačine grafike).
5. Učenici će postići bolji posredovani prijenos pojmova korištenjem metode „premošćivanja“ u odnosu na „metodu prigrljivanja“ pri korištenju vizualnog-blokovskog programskog jezika micro:bit u tekstualni programski jezik Python proceduralnim pristupom.
6. Učenici će biti više motivirani za programiranje u kontekstu programiranja igara u odnosu na „tradicionalni“ kontekst rješavanja matematičkih problema.

3.3 Paradigma i metodologijski pristup

S obzirom da se radi o istraživanju u obrazovanju, primijenjene su različite istraživačke metode. U istraživanjima u obrazovanju ne može se utjecati na veći broj faktora, jer nemamo laboratorijske uvjete, što je slučaj u medicini, fizici, kemiji i sl. Već pri odabiru populacije istraživanja ne mogu se raditi intervencije u smislu biranja određenih učenika, ili skupina što utječe na uzorak koji nije slučajan. Kada se ispituje, na primjer neki razred, tada bi se svi učenici razreda trebali uzeti u obradu, a tek pri obradi podataka mogu se filtrirati podaci učenika koji eventualno odskaču po nekom kriteriju od ostatka razreda. Selektivnim uzorkom populacije učenika narušio bi se prirodni okoliš, te samim tim ponašanje učenika, čime bi se mogla narušiti valjanost istraživanja. Nadalje, opseg istraživanja ograničen je nastavnim planom i programom što utječe na vrijeme provođenja, opseg ispitivanih pojmova te instrumente istraživanja. Zbog navedenih ograničenja istraživanje opisano u ovoj disertaciji napravljeno je u četiri faze, odnosno kroz četiri istraživanja kako bi se napravila triangulacija istraživanja. U svakoj od faza

istraživanja eliminirano je neko od ograničenja iz ostalih istraživanja čime se može dobiti više različitih pogleda na isti istraživački problem.

Svaka faza istraživanja provedena je *in situ*, odnosno u prirodnom, školskom okruženju bez prisustva trećih osoba, što je primjereno za ovu vrstu istraživanja, jer se njime eliminira neprirodno ponašanje sudionika, u ovom slučaju učenika. Osim toga, nastava se odvijala prema redovnom rasporedu, te od učenika nije tražen dodatni angažman u obliku domaćih zadaća kao ni izvanškolskih obveza kako bi svi učenici bili jednako izloženi tretmanu, odnosno redovnoj nastavi. Kako se nastava odvijala prema nastavnom planu i programu nije nužan svjesni pristanak učenika jer nisu dovedeni ni u kakav rizik ni ranjivost samim istraživanjem. Na taj način je njihovo ponašanje prirodnije nego što bi bilo u slučaju svjesnog pristanka jer svjesnošću istraživanja sudionici mijenjaju ponašanje čime se mogu ugroziti rezultati. Povjerljivost i anonimnost je zajamčena jer podaci učenika neće biti otkriveni niti će se analizom podataka moći otkriti identitet učenika. Navedeni pristup se svrstava u naturalističku paradigmu jer se promatrana pojava i sudionici promatraju u svom prirodnom okruženju čime se elimininiraju vanjski utjecaji na rezultate, a sami sudionici nalaze se u prirodnom okruženju čime se eliminira utjecaj vanjskih faktora na istraživanje (Cohen, Manion, & Morrison, 2013). Navedenim pristupom istraživanje ima i praktičnu primjenu jer rezultati neposredno mogu utjecati na provedbu postojeće nastave bez potrebnih većih promjena. Značajno je naglasiti kako predloženi pristup ne znači odustajanje od primjene proceduralnih tekstualnih programskih jezika poput Pythona već se nakon uspješno usvojenih osnovnih koncepata programiranja primjenom posredovanog prijenosa očekuje prijelaz učenika s blokovskog na tekstualni programski jezik.

Istraživanje se može smatrati empirijskim, transverzalnim, razvojnim, ali i akcijskim. Empirijskim se može smatrati jer su se podaci sakupljali iz nastavnog procesa. Transverzalnim jer su se istraživanja provodila tijekom četiri školske godine na različitom uzrastu učenika. Razvojnim jer može utjecati na primjenu novih metoda poučavanja u nastavi informatike. Akcijskim jer je autor sudjelovao u pripremi i realizaciji istraživanja.

Valjanost će se, prema naturalističkim istraživanjima, osigurati prirodnim okruženjem sudionika u kojem je istraživač dio istraživanog svijeta. Osim toga, osigurat će se prostornom triangulacijom, triangulacijom istraživača, triangulacijom metoda te triangulacijom istraživanja. Prostorna triangulacija će se osigurati sudjelovanjem većeg broj škola. Triangulacija istraživača će se postići sudjelovanjem više učiteljica informatike. Triangulacija

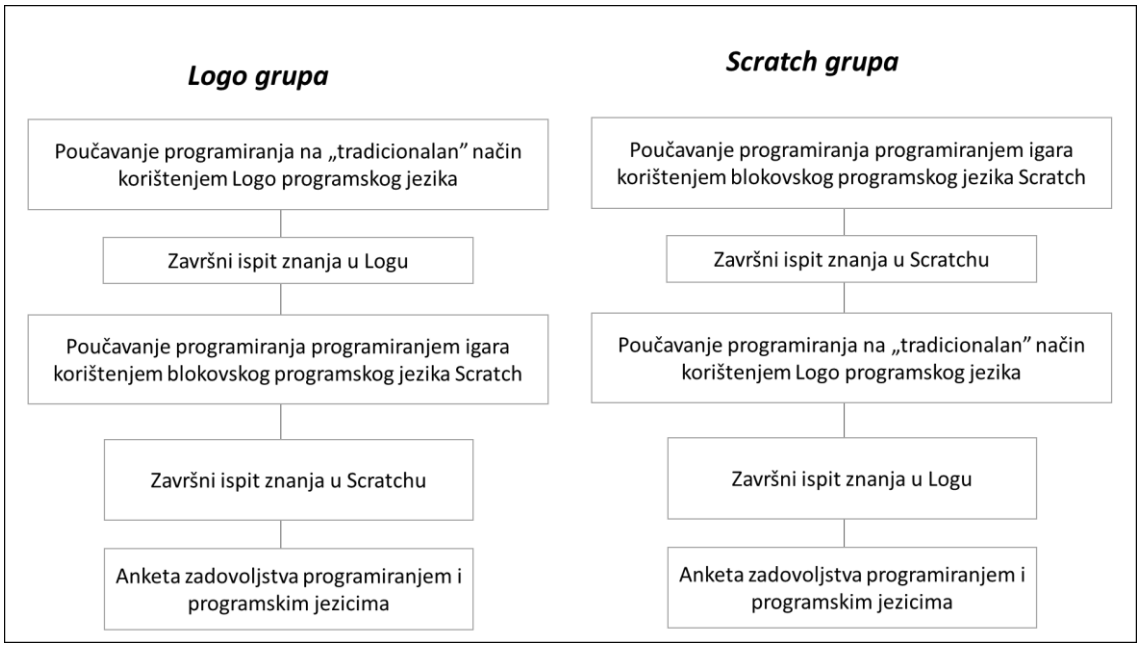
metoda će se postići kvalitativnim i kvantitativnim istraživanjem. Triangulacija istraživanja će se postići s više istraživanja na iste teme, ali s različitim postavkama istraživanja.

3.4 Nacrt istraživanja

Istraživanje je provedeno kao kvazieksperiment s eksperimentalnim i kontrolnim skupinama u četiri faze. Nastavna cjelina programiranja se za vrijeme provedbe istraživanja obrađivala prema već spomenuta dva pristupa, A i B. Prema pristupu A programiranje se poučavalo korištenjem kornjačine grafike, koji uključuje grafičko izvršavanje programa. Prema pristupu B poučavalo se proceduralno programiranje korištenjem tekstualnih programskih jezika. Kako bi se dobio bolji uvid uporabe blokovskih programskih jezika u nastavi u oba pristupa (A i B) tako se i istraživanje provelo u više faza kako bi se istražilo što više pristupa i načina korištenja blokovskih programskih jezika te usporedilo iste s „tradicionalnim“ pristupima. Istraživanja su provedena u školskim godinama: 2013./2014., 2014./2015., 2015./2016. i 2017./2018.

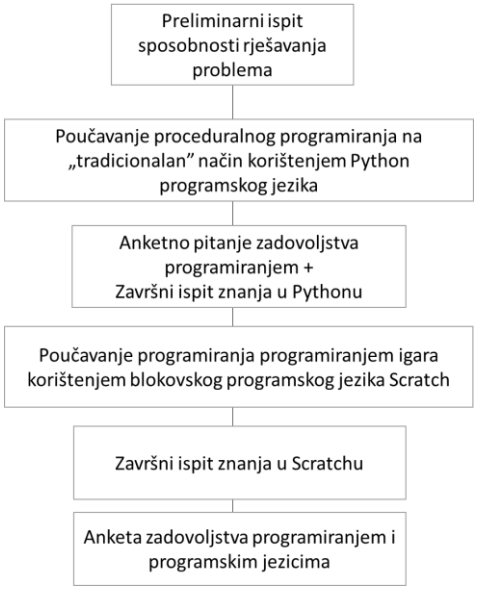
Kvazieksperiment se provodio u 4 faze:

- 1. faza istraživanje – Prva faza istraživanja provedena je nad 23 učenika sedmih razreda (učenici 13-14 godina) jedne osnovne škole, školske godine 2013./2014. Uspoređivala se učinkovitosti korištenja programskih jezika Logo i Scratch. Iako su u pitanju sedmi razredi riječ je o početnicima u programiranju. Zbog toga je odabran pristup A korištenjem kornjačine grafike. Kako se sve teme mogu obraditi i blokovskim jezikom tako su se u istraživanju koristila dva programska jezika: Logo i Scratch. Zbog više zajedničkih svojstava obaju jezika završni ispiti znanja sadrže analogne zadatke. Cilj ovog istraživanja bio je usporediti utjecaj korištenja programskih jezika Logo i Scratch u nastavi na razumijevanje osnovnih programskih koncepata. Na slici 3.1 prikazan je shematski prikaz nacrta prve faze istraživanja.



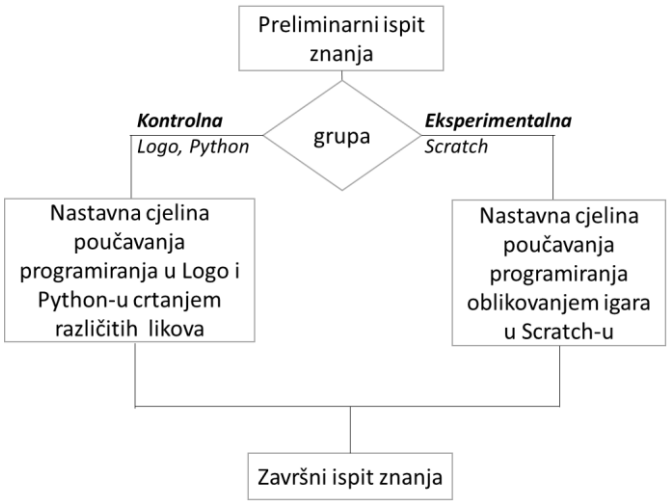
Slika 3.1 - Shematski prikaz nacrta prve faze istraživanja

- 2. faza istraživanja – druga faza istraživanja provedena je tijekom školske godine 2014./2015. nad 54 učenika petih razreda (učenici 11-12 godina) u dvije osnovne škole. U ovom istraživanju se koristio preliminarni ispit za utvrđivanje sposobnost rješavanja problema učenika i završni ispit znanja u Pythonu i Scratchu. Riječ je o petim razredima u kojima su opet svi učenici početnici u programiranju. Ovaj put odabran je B pristup u poučavanju cjeline programiranja. U proceduralnom programiranju teže je uspostaviti direktnu poveznicu između programskih jezika Python i Scratch za razliku od programskih jezika Logo i Scratch. Zbog toga se u ovom istraživanju ispitivao utjecaj korištenja proceduralnog tekstualnog i vizualno-blokovskog programskog jezika na rezultate učenika u završnim testovima, ali i zadovoljstvo učenika programiranjem i programskim jezikom. Na slici 3.2 prikazan je shematski prikaz nacrta druge faze istraživanja.



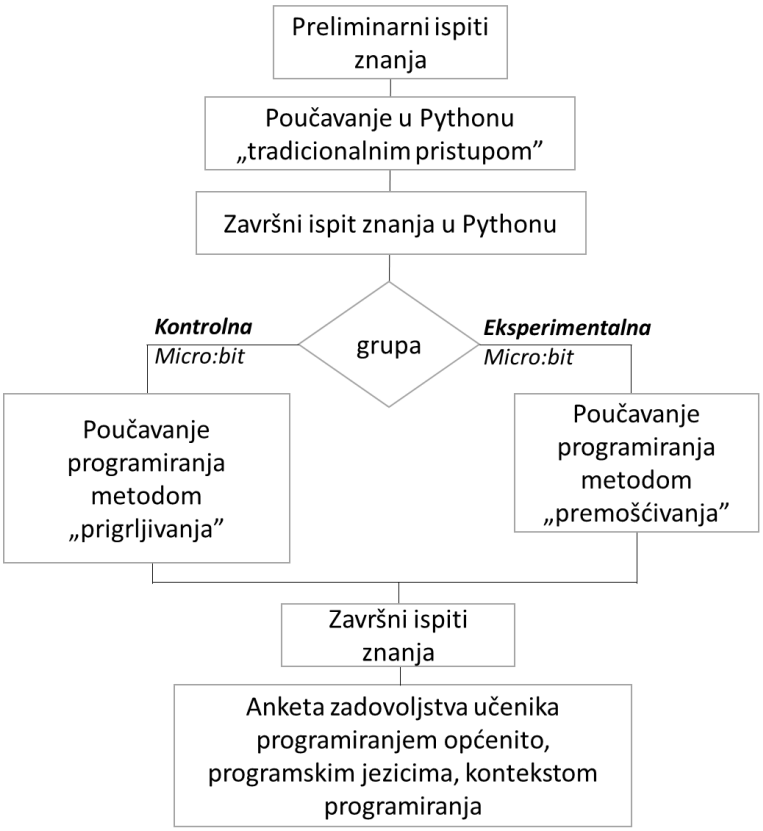
Slika 3.2 - Shematski prikaz nacrta druge faze istraživanja

- 3. faza istraživanja provedena je tijekom školske godine 2015./2016. u tri osnovne škole, na uzorku od 312 učenika od petog do osmog razreda. Nastavna cjelina programiranja obrađivala se prema pristupu A. Kontrolne skupine provodile su „tradicionalno" poučavanje programiranja u tekstualnim programskim jezicima, dok su eksperimentalne skupine provodile poučavanje programiranja korištenjem vizualnih programskih jezika, a neke skupine su osim toga imale i drugačiji kontekst programiranja temeljen na programiranju igara. Ispitanici su testirani prije kako bi se utvrdilo postoje li razlike među skupinama, te nakon intervencije kako bi se provjerili utjecaji različitih načina poučavanja. Na slici 3.3 je prikazan shematski prikaz nacrta treće faze istraživanja.



Slika 3.3 - Shematski prikaz nacrta treće faze istraživanja

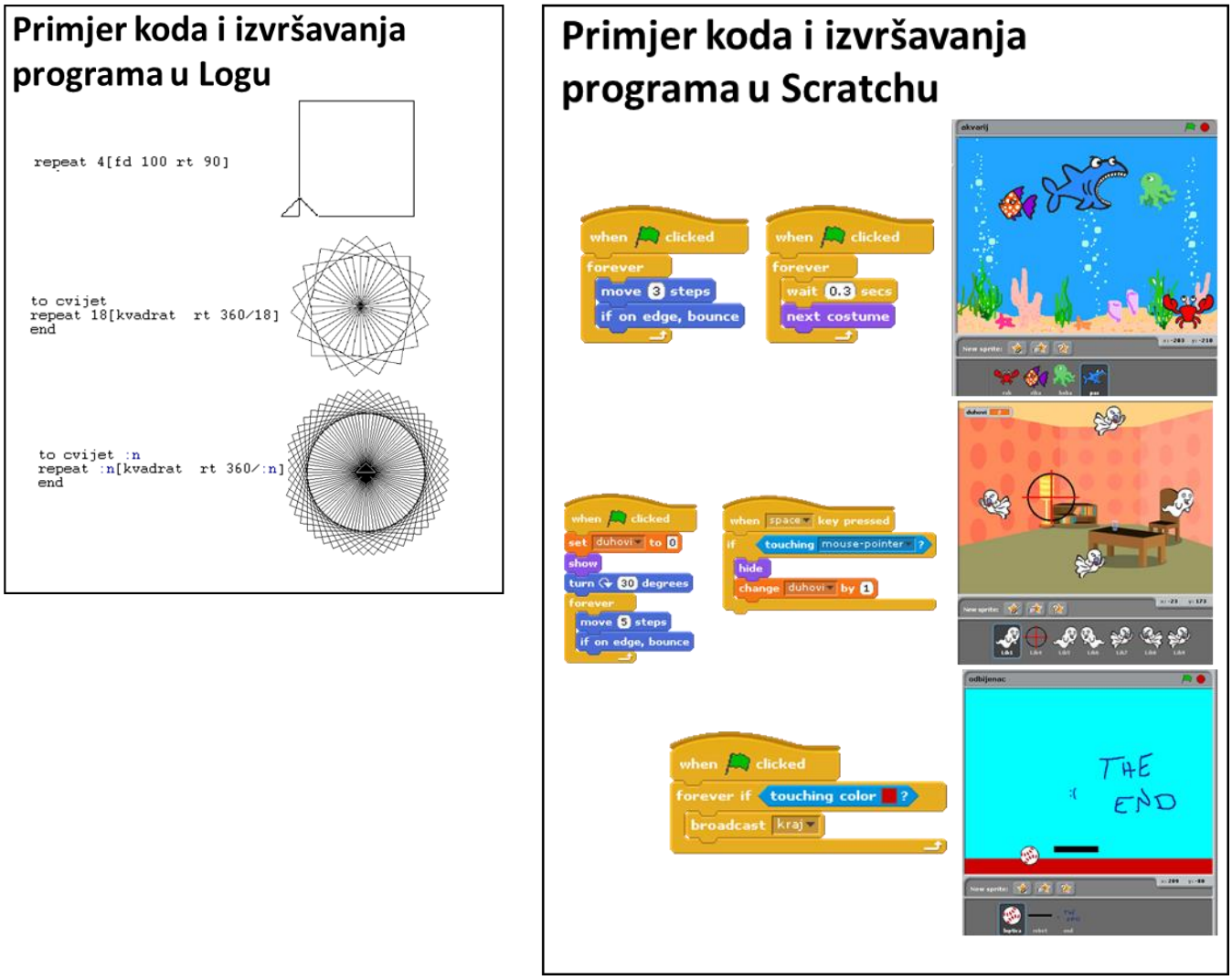
- 4. faza istraživanja je provedena tijekom školske godine 2017./2018. u jednoj osnovnoj školi nad 49 učenika. Nastavna cjelina programiranja obrađivala se prema pristupu B. Svi učenici učili su programiranje korištenjem programskog jezika Python nakon čega se pristupilo eksperimentalnom pristupu poučavanja u kojem su učenici učili programiranje uporabom vizualnog programskog jezika micro:bit, ali na dva različita pristupa. Svi su učenici imali prethodno iskustvo programiranja u Pythonu te su pisali preliminarni ispit znanja. Nakon toga su podijeljeni u dvije eksperimentalne i jednu kontrolnu skupinu koje su se razlikovale u načinu poučavanja micro:bit vizualno-blokovskim programskim jezikom. Kontrolna skupina je programiranje micro:bita učila metodom „prigrljavanja“ dok je eksperimentalna skupina učila metodom „premošćivanja“. Sve grupe rješavale su iste zadatke koristeći različiti pristup. Ispitanici su testirani prije intervencije kako bi se utvrdilo postoje li razlike među skupinama te kako bi se utvrdilo znanje učenika, te nakon intervencije kako bi se provjerio utjecaj različitih načina poučavanja. Osim toga učenici su nakon tretmana ispunjavali anketu zadovoljstva programiranjem, programskim jezikom i kontekstom programiranja. Na slici 3.4 prikazan je shematski prikaz nacrta četvrte faze istraživanja.



Slika 3.4 - Shematski prikaz nacrta četvrte faze istraživanja

Svaka od faza istraživanja bit će detaljnije opisana u sljedećim poglavljima.

Kroz sve četiri faze istraživanja poučavanje tekstualnih programskih jezika provodilo se na „tradicionalan“ način s primjerima i zadacima poput onih koji se nalaze u udžbenicima. U prvoj i trećoj fazi programiranje se poučavanje provodilo prema A pristupu (kornjačina grafika). Na slici 3.5 prikazani su primjeri zadataka u Logu i Scratchu.



Slika 3.5 - Primjeri zadataka u Logu i Scratchu

3.5 Mjerni instrumenti i analiza podataka

Za analizu kvantitativnih podataka koristio se programski paket za statističku obradu podataka SPSS (*Statistical Package for Social Science;* SPSS 20), IBM).

Grafikoni su izrađeni u MS Excel 2016. Deskriptivni podaci su prikazani kao: aritmetička sredina i standardna devijacija. Za unutarnju pouzdanost testova koristio se Cronbach α koeficijent za procjenu pouzdanosti u pisanim testovima. Tumačenje Cronbach α koeficijenta je sljedeće (Cohen et al., 2013):

- veće od 0,90 - vrlo visoko pouzdane
- 0,80–0,90 - visoko pouzdane
- 0,70–0,79 - pouzdane
- 0,60–0,69 - granično/minimalno pouzdane
- <0,60 – neprihvatljivo nisko pouzdane.

Tipovi zadataka korišteni u završnim ispitima znanja su zadaci višestrukog izbora s jednim odgovorom. Ovakvi tipovi zadataka primjereni su za ispitivanje vještine programiranja zbog objektivnosti, bodovanja, veće mogućnosti obuhvata većeg broja tema, jednostavnije analize itd. (Kuechler & Simkin, 2003). Pri izradi zadataka višestrukog izbora vodilo se računa o ponuđenim netočnim odgovorima kao distraktorima. Prema učestalosti odabira pojedinih distraktora u nekim ispitima provedena je analiza utvrđivanja grešaka učenika.

Za utvrđivanje težine zadataka korišten je indeks težine zadataka koji predstavlja prosječnu vrijednost točnih odgovora svih učenika po zadatku. Prema tome indeksi težine mogu biti u intervalu od 0 do 1, odnosno od 0% do 100%. Prema (Cohen et al., 2013) tumačenje indeksa težine je sljedeće:

- <0.33 (33%) – težak zadatak
- 0.33-0.67 (33%-67%) – zadatak umjerene težine
- >0.67 (>67%) – lagan zadatak

4 REZULTATI I RASPRAVA

Kako se istraživanje sastoji od četiri faze, odnosno četiri različita istraživanja, u ovom poglavlju detaljno je opisano svako od istraživanja.

4.1 Usporedba učinka učenika nakon korištenja programskih jezika Logo i Scratch u nastavi

Prva faza istraživanja provedena je tijekom školske godine 2013./2014. nad 23 učenika sedmih razreda jedne osnovne škole. Istraživanje se odnosilo na usporedbu učinkovitosti poučavanja nakon korištenja programskih jezika Logo i Scratch u prepoznavanju naredbi, predviđanju izvršavanja programa, petlji i pisanju programa.

Postavljena je hipoteza:

H1 - učenici će postići bolje rezultate na završnom ispitu znanja u Scratchu nego u Logu

4.1.1 Opis istraživanja

Istraživanje je provedeno tijekom školske godine 2013./2014. u dva sedma razreda osnovne škole tijekom nastave informatike u prirodnom okruženju učenika, bez prisustva trećih osoba. Kako je informatika bila izborni predmet, učenici su se mogli upisati i ispisati s izbornog predmeta nakon svake školske godine, tako učenici obaju razreda nisu imali prethodnog iskustva u programiranju pa je to bio prvi njihov susreta s programiranjem. Jedan razred brojio je 13, a drugi 10 učenika. Informatika se provodila po dva sata tjedno kao blok sat. Umjesto preliminarnog ispita znanja za usporedbu grupa u obzir su se uzele zaključne ocjene 5. i 6. razreda iz matematike prethodnih godina zbog visoke korelacije između sposobnosti programiranja i uspjeha iz matematike (White & Sivitanides, 2003a).

Prema tadašnjem kurikulu informatike, za nastavu programiranja se birao jedan od dva moguća pristupa, ali programski jezici nisu bili definirani. Jedan pristup temeljio se na proceduralnom programiranju, gdje je još uvijek bio dominantan BASIC kao programski jezik. Drugi pristup temeljio se na kornjačinoj grafici, kod koje je dominantan programski jezik bio Logo. Kako su svi učenici bili početnici u programiranju program nastavne cjeline programiranja (pristup A) s kornjačinom grafikom bio je primjereniji jer se rezultat izvršavanja programa vizualizira što ga čini manje apstraktnim za učenike. Scratch se izvrsno uklapa u navedeni pristup te se svi koncepti koji se uče mogu obuhvatiti i Scratchom. Kako učenici nisu imali nikakvo predznanje u programiranju bilo je potrebno na početku cjeline programiranja obuhvatiti i sve koncepte koji su prema programu trebali biti obrađeni. U tablici 4.1 je prikazan opis rada *Logo grupe*.

Tablica 4.1- Program Logo grupe

Logo grupa				
tjedan		**Tema**	**Nove naredbe**	**Novi koncepti**
1.	**LOGO**	Crtanje osnovnih geometrijskih oblika	fd, lr, rt, lt, repeat	Pokretanje kornjače korištenjem osnovnih naredbi; petlje
2.		Crtanje geometrijskih oblika	Procedure	Procedure
3.		Crtanje geometrijskih oblika	Procedure s parametrima	Procedure s parametrima
Logo - završni ispit znanja				
4.	**Scratch**	Simulacija akvarija	Naprijed (move), lijevo (left), desno (right), ponovi (repeat)	Likovi, paralelnost, petlje
5.		Igra *Hvatanje duhova*	if	Uvjetno izvršavanje programa
6.		Igra *Odbijenac*	Šalji poruku (broadcast), Ponovi ako je (repeat if)	Petlja s uvjetnim izvršavanjem
Scratch - završni ispit znanja				
Anketa o zadovoljstvu programiranjem i programskim jezicima				

U tablici 4.2 je prikazan opis rada *Scratch grupe*.

Tablica 4.2- Program Scratch grupe

Scratch grupa				
tjedan		**Tema**	**Nove naredbe**	**Novi koncepti**
1.	**Scratch**	Simulacija akvarija	Naprijed (move), lijevo (left), desno (right), ponovi (repeat)	Likovi, paralelnost, petlje
2.		Igra *Hvatanje duhova*	if	Uvjetno izvršavanje programa
3.		Igra *Odbijenac*	Šalji poruku (broadcast), Ponovi ako je (repeat if)	Petlja s uvjetnim izvršavanjem
Scratch - završni ispit znanja				
4.	**LOGO**	Crtanje osnovnih geometrijskih oblika	fd, lr, rt, lt, repeat	Pokretanje kornjače korištenjem osnovnih naredbi; petlje
5.		Crtanje geometrijskih oblika	Procedure	Procedure
6.		Crtanje geometrijskih oblika	Procedure s parametrima	Procedure s parametrima
Logo - završni ispit znanja				
Anketa o zadovoljstvu programiranjem i programskim jezicima				

U jednom razredu se programiranje obrađivalo prvo u programskom jeziku Logo (*Logo grupa*), pa nakon toga u Scratchu. Druga grupa je imala obrnuti redoslijed, pa su prvo radili u Scratchu (*Scratch grupa*), zatim u Logu. Osim razlike u programskom jeziku promijenjen je i kontekst

programiranja. Tako se Logo obrađivao „tradicionalnim“ pristupom, rješavanjem geometrijskih problema, dok se u Scratchu pristup temeljio na programiranju igre, a već sami scenarij igre predstavljen je kao algoritam.

4.1.2 Instrumenti

Kako je cilj istraživanja utvrditi razlike u uspjehu na završnom ispitu znanja za oba programska jezika, tako su korišteni analogni zadaci završnih testova znanja za oba korištena programska jezika. Zbog navedenog je ograničen broj testiranih koncepata, ali je uspjeh na pojedinačnim zadacima i ukupnom testu usporediv. Važno je napomenuti kako ni jedna grupa nije uvježbavala zadatke koji su se koristili pri ispitivanju. Osim toga, učenici nisu dobivali domaće zadaće ni dodatne zadatke za vježbu. Učenici nisu ocjenjivani za ispit znanja kako bi se, između ostalog, izbjegao dodatan individualan rad kod kuće koji bi mogao utjecati na rezultate istraživanja. S druge strane, učenici su motivirani na način da im je prije pisanja ispita rečeno da mogu, za uspješno odrađen ispit, biti nagrađeni odličnom ocjenom ili plusom.

Za istraživanje su korištena tri instrumenta:

- završni ispit znanja u Logu;
- završni ispit znanja u Scratchu;
- anketa o stavu prema programiranju i svakom od programskih jezika.

Oba završna ispita znanja provedena su nakon završetka procesa učenja. Učenici su ispite pisali u papirnatom obliku. U tablici 4.3 prikazan je opis pitanja završnih ispita znanja kao i tip korištenih pitanja.

Tablica 4.3 - Opis pitanja završnog ispita znanja

zadatak	Ishod	Tip zadatka
Z1	Prepoznavanje naredbi	Uparivanje
Z2	Predviđanje ishoda izvršavanja programa (algoritam slijeda)	• Zadaci višestrukog izbora s jednim odgovorom • Objašnjenje odgovora
Z3	Predviđanje ishoda izvršavanja programa (algoritam ponavljanja)	
Z4	Predviđanje ishoda izvršavanja programa (algoritam ponavljanja – ugniježđena petlja)	
Z5	Pisanje algoritma	Pisanje odgovora (zadatak otvorenog tipa)

S obzirom na apstrakciju istraživanih koncepata težina zadataka je graduirana od najlakšeg prema najtežem. Učenici su uz zadatke Z2, Z3, i Z4 mogli objasniti odgovor pri čemu su točan

odgovor kao i točno objašnjenje nosili po jedan bod. Na taj način se smanjuje utjecaj nasumičnog odabira odgovora. Osim toga, može se utvrditi uzrok netočnih odgovora, odnosno pogrešna shvaćanja koncepata koji se ispituju.

Anketa zadovoljstva koju su svi učenici dobrovoljno i anonimno ispunjavali nakon cijelog tretmana u on-line obliku također je instrument. U tablici 4.4 prikazana su pitanja i tip pitanja u anketi zadovoljstva programiranjem i programskim jezicima.

Tablica 4.4 - Anketa zadovoljstva programiranjem

Pitanje	**Ishod**	**Tip zadatka**
P1	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje	Likertova skala (1-5)
P2	Koji ti se programski jezik više sviđa?	Zadatak višestrukog izbora s jednim odgovorom
P3	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa LOGO.	Likertova skala (1-5)
P4	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Scratch.	Likertova skala (1-5)
P5	Ako imaš neki svoj komentar u vezi programiranja slobodno ga napiši.	Pitanje otvorenog tipa

4.1.3 Analiza podataka

Za utvrđivanje ujednačenosti grupa, umjesto preliminarnog ispita znanja, napravljena je usporedba ocjena iz matematike i hrvatskoga jezika. Zbog malog uzorka korišten je neparametrijski Mann-Whitney U test prema kojem je utvrđeno kako nema razlika između grupa niti po uspjehu iz hrvatskoga jezika (U=64.0, p=0.95) niti po uspjehu u matematici (U=41.0, p=0.07).

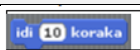
Kako je uzorak ispitanika manji od 50 za utvrđivanje analize distribucije rezultata korišten je Shapiro-Wilk test. Shapiro-Wilk testom utvrđena je normalna distribucija rezultata obaju testova svih učenika (Tablica 4.5).

Tablica 4.5 - Rezultati Shapiro-Wilkovog testa

Grupa		*df*	*p*
LOGO rezultati	Logo grupa	10	0.175
	Scratch grupa	13	0.748
Scratch rezultati	Logo grupa	10	0.056
	Scratch grupa	13	0.842

Z1 – u prvom zadatku učenici su trebali povezati naredbu s odgovarajućim značenjem. Točan odgovor nosio je jedan bod.

Tablica 4.6 - Zadatak 1

Z1 - Logo		Z1 - Scratch	
a) fd 100	1) Idi nazad za 100 koraka		1) Ponavljaj zauvijek
b) rt 100	2) Idi naprijed za 100 koraka		2) Ponovi deset puta
c) lt 100	3) Okreni se desno za 100 stupnjeva		3) Idi naprijed deset koraka
d) bk 100	4) Ponovi 100 puta		4) Okreni se u desno za 100 stupnjeva
e) repeat 100 []	5) Okreni se lijevo za 100 stupnjeva		5) Okreni se u lijevo za 100 stupnjeva

U zadacima Z2, Z3 i Z4 učenici su trebali odabrati jedan od odgovora uz pitanje „Za koliko koraka će se lik pokrenuti nakon izvršavanja sljedeće skripte?“. Osim toga, učenici su trebali objasniti svoj odgovor. Točan odgovor se bodovao jednim bodom, a ispravno objašnjenje s još jednim bodom.

Z2 – u drugom zadatku se ispitivao ishod algoritma slijeda s prepoznavanjem naredbi, odnosno učenici su trebali prepoznati samo naredbe kretanja te izračunati za koliko koraka će se lik pokrenuti. Zadatak je prikazan u tablici 4.7.

Tablica 4.7 - Zadatak 2

Z2 - Logo	Z2 - Scratch	Z2 - odgovori

`fd 100`
`rt 90`
`fd 50`
`fd 50`
`rt 90`
		a) 0
		b) 100
		c) 150
		d) 200
		e) 380

Z3- u trećem zadatku se ispitivala jednostavna petlja, odnosno ishod izvršavanja programa s jednostavnom petljom. Zadatak je prikazan u tablici 4.8.

Tablica 4.8 - Zadatak 3

Z3 - Logo	Z3 - Scratch	Z3 - odgovori
`repeat 10 [fd 10]`		a) 1 b) 10 c) 20 d) 100 e) 1000

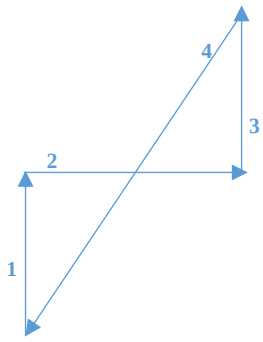
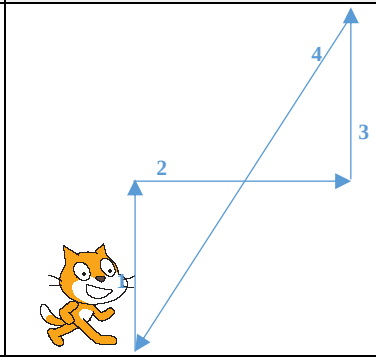
Z4- u četvrtom zadatku ispitivao se ishod izvršavanja programa s ugniježđenom petljom. Zadatak je prikazan u tablici 4.9.

Tablica 4.9 - Zadatak 4

Z4 - Logo	Z4 - Scratch	Z4 - odgovori
`repeat 10 [fd 1 repeat 10 [fd 1]]`		a) 10 b) 20 c) 100 d) 110 e) 1000

Z5- u petom zadatku učenici su trebali napisati algoritam za kretanje lika/kornjače prema zadanom putu, zadanim redoslijedom. Za opis algoritma mogli su birati između Logo ili Scratch sintakse ili algoritam opisati svojim riječima (pseudokod). Zadatak je prikazan u tablici 4.10.

Tablica 4.10 - Zadatak 5

Z5 - Logo	Z5 - Scratch
	

Kako je ukupan broj sudionika <30, unatoč normalnoj distribuciji rezultata, korišteni su neparametrijski testovi zbog nezadovoljavanja uvjeta za uporabu parametrijskih.

Za usporedbu rezultata Logo i Scratch ispita znanja unutar iste grupe korišten je Wilcoxon test umjesto t-testa.

4.1.3.1 Usporedba rezultata na Logo i Scratch ispitima znanja unutar iste grupe

Wilcoxonov test korišten je za usporedbu rezultata Logo i Scratch ispita znanja unutar iste grupe. Rezultati Wilcoxonovog testa su prikazani u tablici 4.11.

Tablica 4.11 - Rezultati Wilcoxonovog testa

		Z1 Scratch – LOGO	Z2 Scratch – LOGO	Z3 Scratch – LOGO	Z4 Scratch – LOGO	Z5 Scratch – LOGO	UKUPNO Scratch – LOGO
Logo grupa	Z	-1	-1,342	-1.342	-2.53	0	-2.263
	p	0.317	0.18	0.18	0.011*	1	0.024*
Scratch grupa	z	-1.414	-0.905	-1.414	-0.577	-1.265	-0.871
	p	0.157	0.366	0.157	0.564	0.206	0.384

Kod *Logo grupe* rezultati Wilcoxonovog testa pokazali su statistički značajnu razliku kod zadatka 4 koji se odnosi na ugniježđenu petlju (z=-2.53, p=0.011). Također postoji statistički značajna razlika kod ukupnog rezultata testova (z=-2.263, p=0.024). Rezultati u Scratch ispitu znanja (Aritmetička sredina postotaka riješenosti: 65.56±35.313) su bolji od rezultata u Logo ispitu znanja (Aritmetička sredina postotaka riješenosti: 48.89±25.769).

Kod *Scratch grupe* nema statistički značajnih razlika ni za jedno pitanje.

Frekvencije odgovora prikazane su u tablici 4.12.

Tablica 4.12 - Frekvencije odgovora

		Z1		Z2		Z3		Z4		Z5	
grupa	bod	Logo	Scratch	Logo	Scratch	Logo	Scratch	Logo	Scratch	Logo	Scratch
Logo grupa	0	1	0	3	1	0	1	10	3	5	5
	1	9	10	3	4	5	0	0	1	1	1
	2	-	-	4	5	5	9	0	6	4	4
Scratch grupa	0	2	0	5	1	1	1	5	7	8	5
	1	11	13	3	8	2	0	3	2	3	5
	2	-	-	5	3	10	12	5	4	2	3

Za Z4, kod kojeg postoji statistički značajna razlika u rezultatima *Logo grupe*, vidljivo je kako nitko nije točno odgovorio na pitanje u Logo ispitu znanja. Iako je bilo ponuđeno pet odgovora nitko nije niti slučajno pogodio točan odgovor. Kod istog zadatka u Scratchu šest učenika je potpuno točno odgovorilo na isto pitanje (zaokružen točan odgovor uz ispravno objašnjenje), a jedan učenik je samo zaokružio točan odgovor bez objašnjenja.

4.1.3.2 Usporedba rezultata na Logo i Scratch ispitima znanja među grupama

Daljnjom analizom istraživalo se postoji li razlika u rezultatima prema grupama. Za tu analizu koristio se neparametrijaski Man-Whitney test. Statistički značajna razlika opet se pokazala za zadatak s ugniježđenom petljom Z4 (U=25.0, p=0.00). Medijan za *Logo grupu* zadatka 4 je 0 dok je za *Scratch grupu* iznosio 1. Kako je već rečeno, kod *Logo grupe* niti jedan učenik nije ponudio točan odgovor na Z4 dok je iz *Scratch grupe* 6 od 13 učenika točno odgovorilo. Rezultati deskriptivne statistike ukupnih rezultata prikazani su u tablici 4.13.

Tablica 4.13 - Deskriptivna statistika za rezultate završnih ispita

Jezik	grupa	N	AS	Median	Mod	Min	Max
Logo ispit	Logo grupa	10	4.40	4.5	7	1	7
	Scratch grupa	13	5.08	5	6	1	9
Scratch ispit	Logo grupa	10	5.90	7	9	1	9
	Scratch grupa	13	5.69	6	4	2	9

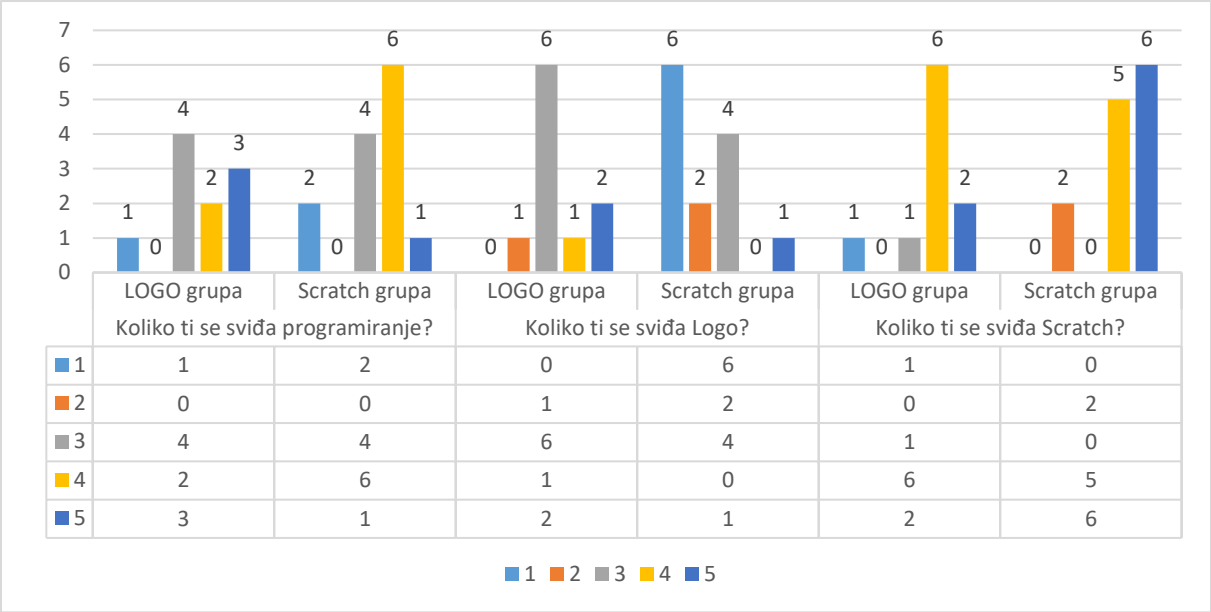
Očito je kako su obje grupe bolje riješile Scratch ispit u odnosu na Logo ispit, s tim da je *Logo grupa* bolje riješila Scratch ispit od *Scratch grupe*, dok je s Logo ispitom situacija suprotna. Ovakvi rezultati su i očekivani jer su se obrađivali isti koncepti u različitim kontekstima i različitim programskim jezicima. Stoga je očekivano da se u drugom programskom jeziku ostvare bolji rezultati od prvog programskog jezika. Ono što je važno je veliki pomak *Logo grupe* u Scratch ispitu u odnosu na Logo ispit.

Iz svih prikazanih rezultata može se prihvatiti H1 i zaključiti kako učenici postižu bolje rezultate u Scratch ispitu znanja u odnosu na Logo ispit znanja.

4.1.3.3 Stav prema programiranju i programskim jezicima

Učenici su anketu o zadovoljstvu programiranjem i programskim jezicima ispunjavali nakon cijelog tretmana, odnosno nakon što su svi obradili Scratch i Logo. Učenici su u tri pitanja birali ocjenu prema Likertovoj skali od 1 (ne sviđa mi se uopće) do 5 (jako mi se sviđa) o stavu prema programiranji i korištenim programskim jezicima. Kako su pitanja postavljena afirmativno (primjerice, Koliko ti se sviđa programiranje?) veća ocjena prema Likertovoj skali odgovara većem slaganju s postavljenim pitanjem.

Na slici 4.1 prikazane su frekvencije odgovora na tri pitanja prema grupama.



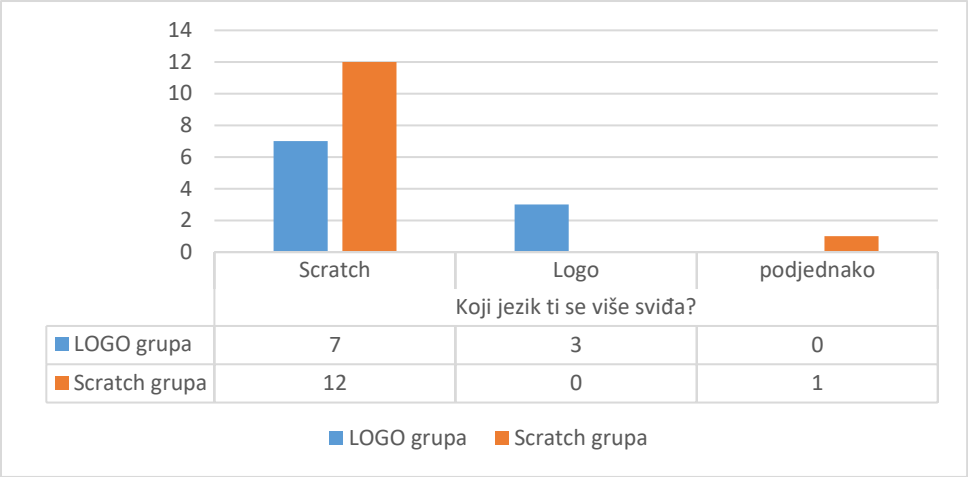
Slika 4.1 - Frekvencije odgovora na anketna pitanja

Iz frekvencija odgovora može se primijetiti kako učenici iz *Logo grupe* imaju nešto lošiji stav prema programiranju, ali im se više sviđa Logo nego učenicima iz *Scratch grupe*. Osim toga učenici koji su prvo programirali u Scratchu daju veće ocjene Scratchu, a manje Logu. Učenici koji su prvo programirali u Scratchu bolje ocjenjuju Scratch u odnosu na Logo. U tablici 4.14 je prikazana deskriptivna statistika za anketna pitanja.

Tablica 4.14 - Deskriptivna statistika za anketna pitanja

	Logo grupa				**Scratch grupa**			
Pitanje	**M**	**Mod**	**Min**	**Max**	**M**	**Mod**	**Min**	**Max**
Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje	3.5	3	1	5	4.0	1	1	5
Ocjeni ocjenom od 1 do 5 koliko ti se sviđa LOGO	3.0	3	2	5	2.0	1	1	5
Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Scratch	4.0	4	1	5	4.0	5	2	5

Kod pitanja izbora jezika (slika 4.2) većina učenika preferira Scratch, ali je važno uočiti kako troje od deset učenika iz *Logo grupe* daje prednost Logu što je u skladu s istraživanjem koje je pokazalo kako učenici koji imaju iskustvo programiranja u nekom tekstualnom programskom jeziku Scratch ne smatraju „pravim“ programskim jezikom (Lewis, 2010).



Slika 4.2 - Frekvencija odgovara za izbor programskog jezika

Kako su učenici u anketi imali i neobavezno pitanje otvorenog tipa za komentiranje programiranja zanimljivo je vidjeti njihove komentare koje je napisalo 7 učenika. Komentari su prikazani u izvornom obliku točno kako su ih učenici napisali.

- *„Bilo bi dobro da scratch ima komande ka logo da moras upisivati bas. ”*
- *„Zanimljivo, zabavno i korisno ”*
- *„logo mi je dosadan, a scratch je mi je zanimljiv. ”*
- *„logo mi je uzasan i dosadan,scratch mi je zanimljiv i zabavan ”*
- *„Zanimljivije radit kao igre na Strachu nego mnogokute,krugove i takve stvari na FMSLogu ”*
- *„Scratch je bolji i ljepsi, a u Loga se moraju pamtit raznovrsne formule na engleskom.”*
- *„Lakše i bolje mi je raditi sa Scratch-om nego s Logom. ”*

Iz komentara učenika može se uočiti kako im je Scratch bio zanimljiviji. Važno je uočiti i problem jezika u tekstualnom programskom jeziku kao što je Logo jer su naredbe skraćeni zapisi engleskih riječi, što učenicima dodatno otežava učenje. Osim toga uočljivo je i nezadovoljstvo rješavanjem matematičkih problema, u ovom slučaju crtanja mnogokuta.

Kako je učiteljica ujedno bila i istraživačica, može se reći da je imala ulogu sudjelujućeg opažača. Najznačajniji dojmovi su da se uporabom Scratcha u manje vremena može obuhvatiti puno više programskih koncepata nego korištenjem Loga i to na učenicima konkretan način, čime učenici sa slabije razvijenim apstraktnim mišljenjem mogu aktivno sudjelovati u nastavi. Osim toga učitelj ima više vremena za rad sa „slabijim“ učenicima jer „bolji“ učenici imaju

velik broj mogućnosti za istraživanje kako unaprijediti svoje igre, animacije, priče ili slično. Uočen je i problem pamćenja naredbi u Logu jer su vezane uz engleski jezik tako da su neki učenici pitali postoje li naredbe na hrvatskome jeziku. Dojam istraživačice je i da je „boljim“ učenicima manje bitno programsko okruženje, ono je puno važnije za ostale učenike.

4.1.3.4 Ograničenja istraživanja, rasprava i zaključak

U ovom istraživanju sudjelovali su učenici sedmih razreda koji su, prema Piagetu, u fazi predformalnih operacija. Svi učenici su bili početnici u programiranju. Rezultati su pokazali kako postoji statistički značajna razlika u rezultatima kad je u pitanju ugniježđena petlja gdje su učenici koji su prvo radili Scratch bili uspješniji u odgovorima. Postoji više vjerojatnih razloga za to. Jedan je sigurno sintaksa s kojom u Scratchu, kao blokovskom jeziku, učenici nemaju problema. Takvo okruženje omogućuje usmjeravanje pažnje na rješavanje problema i to u okruženju koje omogućuje pomicanje konteksta programiranja prema programiranju igara, animacija i slično. Na taj način učenici imaju pozotovniji stav prema programiranju. Scratch omogućuje uvođenje više programskih koncepata u manje vremena. Takve programske koncepte, naučene u blokovskom jeziku, učenici lakše mogu savladati u tekstualnom okruženju.

Ograničenje istraživanja je sudjelovanje manjeg broj učenika i razreda. Istraživanje treba ponoviti na većem broju učenika, većem broju razreda u kojima predaju različiti učitelji kako bi se mogao isključiti utjecaj učitelja na razred. U sljedećem istraživanju će se istraživanje ciljano usmjeriti na koncept petlje kod kojeg se pokazala statistički značajna razlika u korist rezultata ostvarenih u Scratchu. Sve navedeno realizirat će se u 3. fazi ukupnog istraživanja.

4.2 Usporedba korištenja proceduralnog i blokovskog jezika

Cjelina programiranja u osnovnim školama se provodila prema HNOS-u do školske godine 2018./2019. Programiranje se obrađivalo kroz jedan od dva pristupa: A pristup s kornjačinom grafikom, te B pristup uz proceduralno programiranje. Zbog grafičkog rezultata izvršavanja programa, prema pristupu A, učenici mogu konkretno vidjeti vizualan rezultat izvršavanja programa. Na taj način se u manje vremena može obraditi više tema jer je potrebna manja razina apstrakcije u usporedbi s proceduralnim programiranjem. Prethodno istraživanje (M. Mladenović, Rosić, et al., 2016) je potvrdilo neke rezultate drugih istraživanja (Armoni et al., 2015; Dann, Cosgrove, Slater, & Culyba, 2012b; J. H. Maloney, Peppler, Kafai, Resnick, Rusk, et al., 2008; Orni Meerbaum-Salant et al., 2013), gdje se pokazalo kako se korištenjem blokovskih programskih jezika još više može spustiti razina apstrakcije potrebna za programiranje. Proceduralno programiranje je zahtjevnije te zahtijeva veću razinu apstrakcije. Kako su učenici u petom razredu još uvijek većinom u fazi konkretnih ili predformalnih operacija tako je proceduralno programiranje još zahtjevnije. Zbog toga se prema B pristupu programiranja obrađuje manje tema uz isti broj sati kao u A pristupu. U tablici 4.15 su prikazane teme programiranja za 5. razred prema A i B pristupima.

Tablica 4.15 - Teme programiranja za 5. razred

	Teme A (kornjačina grafika)	Teme B (proceduralni jezik)
5. razred	1. Osnovne naredbe programskog jezika 2. Ponavljanje niza naredbi 3. Uporaba petlje za crtanje niza likova 4. Ulazne vrijednosti procedura 5. Uporaba više ulaznih vrijednosti 6. Odluke u programu	1. Pojam algoritma 2. Dijagram tijeka 3. Naredbe za ulaz i izlaz podataka

Osnovni algoritmi: slijed grananje i ponavljanje se prema B pristupu obrađuju samo kroz prvu temu koja nije vezana uz programski jezik, za razliku od A pristupa kod kojeg se sva tri algoritma obrađuju kroz programski jezik. U ovom istraživanju se cjelina programiranja obrađivala prema B pristupu kod kojeg postoji mogućnost obrade prve teme pomoću blokovskog programskog jezika, u ovom slučaju Scratcha. Kako se redoslijed tema može mijenjati tako se pojam algoritma obrađivao i nakon algoritma slijeda (Naredbe za ulaz i izlaz podataka).

Osnovni cilj ovog istraživanja je bio utvrditi postoje li razlike u uspjehu učenika između preliminarnog ispita (PI) rješavanja problema i završnih ispita (ZI) znanja u programskim

jezicima Python i Scratch. Osim toga cilj je istražiti stavove učenika prema programiranju i programskim jezicima. Prema navedenim ciljevima postavljene su sljedeće hipoteze:

H1 – Učenici s većom sposobnosti rješavanja problema će biti uspješniji u završnom ispitu znanja u Pythonu od učenika s nižom sposobnosti rješavanja problema

H2 - Učenici s većom sposobnosti rješavanja problema će biti uspješniji u završnom ispitu znanja u Scratchu od učenika s nižom sposobnosti rješavanja problema

H3 – Stav prema programiranju će biti bolji nakon nastave uz korištenje Scratch nego nakon Pythona.

4.2.1 Opis istraživanja

Istraživanje o usporedbi korištenja proceduralnog i blokovskog jezika provedeno je tijekom školske godine 2014/2015 među 50 učenika u četiri peta razreda (11-12 godina) u dvije osnovne škole bez prisustva trećih osoba. Kako svi učenici nisu bili prisutni u svim fazama istraživanja tako je ukupan broj 50 iako je ukupan broj učenika bio 54. Kako se istraživala ciljana skupinu učenika, tako je uzorak neslučajan i ciljan (Cohen et al., 2013).

Nastava se odvijala prema redovnom rasporedu od dva sata tjedno kao blok sat. U tablici 4.16 su prikazan uzorak sudionika.

Tablica 4.16 - Sudionici istraživanja

Razred	Škola	Broj učenika	Dječaci	Djevojčice
5a	S1	10	5	5
5b	S1	16	13	3
5c	S1	10	8	2
5b	S2	14	8	6
Ukupno	2	50	34	16

Učiteljica u svim razredima je bila ista i to sama istraživačica. Kako su u pitanju peti razredi svi učenici su bili početnici u programiranju bez prethodnog programerskog iskustva. U ovom istraživanju se cjelina programiranja obrađivala prema B programu, odnosno proceduralno programiranje u programskom jeziku Python. Zbog veće razine apstrakcije potrebne za razumijevanje osnovnih programskih koncepata zadane su samo tri teme za peti razred. Kako programski jezici nisu strogo zadani tako se može kombinirati proceduralni jezik, u ovom slučaju Python, i blokovski jezik, u ovom slučaju Scratch. Scratch se koristio za temu „Pojam algoritma“ jer se korištenjem Scratcha mogu lako i brzo uvesti sva tri algoritma kroz konkretne primjere bliske učenicima što se može provesti kroz programiranje igara.

U tablici 4.17 je prikazan dizajn istraživanja.

Tablica 4.17 - Dizajn istraživanja

tjedan		Teme	Novi pojmovi i naredbe	Novi koncepti
Preliminarni ispit rješavanja problema				
1	**Python**	Pojam algoritma	Algoritam: slijed, grananje, ponavljanje	Uvođenje pojma algoritma s primjerima iz svakodnevnog života. Uvod u programski jezik Python.
2		Naredbe za ulaz i izlaz podataka	Varijable, input, print, int	Osnovne naredbe s primjerima u Pythonu.
3		Naredbe za ulaz i izlaz podataka	Aritmetičke operacije (+,-,*, /)	Rješavanje jednostavnih problema korištenjem algoritma slijeda u Pythonu.
4		Naredbe za ulaz i izlaz podataka	If else	Rješavanje jednostavnih problema korištenjem algoritma slijeda u Pythonu uz uvođenje pojma grananja.
Završni ispit znanja u Pythonu, ispunjavanje upitnika o programiranju i Pythonu				
Tri tjedna božićnih praznika				
1	**Scratch**	Pojam algoritma (*Simulacija akvarija*)	Naprijed (move), lijevo (left), desno (right), ponovi (repeat)	Korištenje algoritama slijeda, ponavljanja i grananja. Likovi, paralelnost, petlje
2		Pojam algoritma Igra (*Hvatanje duhova)*	if	Korištenje algoritama slijeda, ponavljanja i grananja. Uvjetno izvršavanje programa
3		Pojam algoritma (Igra *Odbijenac)*	Šalji poruku (broadcast), Ponovi ako je (repeat if)	Korištenje algoritama slijeda, ponavljanja i grananja. Petlja s uvjetnim izvršavanjem
Završni ispit znanja u Scratchu, ispunjavanje upitnika o programiranju i programskim jezicima				

4.2.2 Instrumenti

Za razliku od prethodnog istraživanja gdje su se završni ispiti mogli napraviti s analognim zadacima u dva programska jezika, u ovom slučaju to nije bilo moguće zbog razlike u programskim jezicima. Svi učenici su pisali iste ispite i ankete te se nastava provodila prema istom programu. Učenici nisu uvježbavali tipizirane zadatke koji su bili u ispitu, učenici nisu dobivali domaće zadaće niti dodatne zadatke za vježbu. Učenici su motivirani za ispite znanja na način da im je prije pisanja ispita rečeno da mogu, za uspješno odrađen ispit, biti nagrađeni odličnom ocjenom ili plusom kako ne bi bili demotivirani za ispit koji se nije ocjenjivao. Anketu su ispunjavali dobrovoljno i anonimno.

Podaci su prikupljani u tri faze uz korištenje pet instrumenata:

- Preliminarni ispit (PI) sposobnosti rješavanja problema (prva faza);
- Završni ispit znanja u Pythonu (ZIP) (druga faza);
- Anketa o stavu prema programiranju (druga faza);
- Završni ispit znanja u Scratchu (ZIS) (treća faza);
- Anketa o stavu prema programiranju i programskim jezicima (treća faza).

U prvoj fazi provedbe učenici su prije nastavne cjeline programiranja rješavali preliminarni ispit sposobnosti rješavanja problema. Ispit se pisao na papiru do 45 minuta.

U tablici 4.18 su prikazani zadaci ispita.

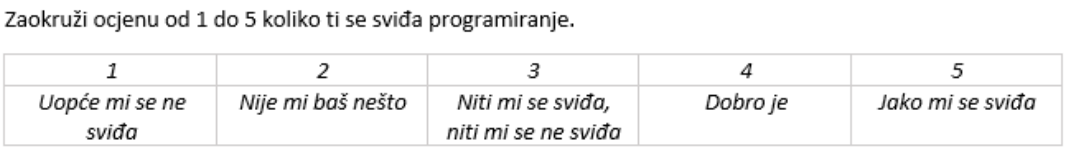
Tablica 4.18 - Preliminarni ispit sposobnosti rješavanja problema

	Zadatak	bod	problem
Z1	Nastavi niz sa sljedeća tri slova: A B A C A D A E __ __ __	*1*	*slijed, prepoznavanje uzorka ponavljanja*
Z2	Nastavi niz s dvije kombinacije slova: JKLMNO JKLMON JKLOMN JKOLMN _____ _____	*2*	*slijed, prepoznavanja uzorka ponavljanja*
Z3	Ako robotu damo upute da ponovi sljedeće radnje 3 puta: idi po 3 koraka naprijed. Koliko će ukupno koraka robot napraviti?	*1*	*ponavljanje*
Z4	Ako riječ REČENICA sadrži manje od 9 slova i više od 3 sloga, napiši prvi slog. U suprotnom napiši suglasnik koji se nalaze najdesnije u riječi.	*1*	*Uvjetno izvršavanje*
Z5	Ako je petak 3 dana prije jučerašnjeg dana. Koji dan će biti sutra?	*1*	*slijed*
Z6	Ivan je teži od Martina, ali lakši od Ante. Napiši njihova imena poredana po težini od lakšeg prema težem.	*1*	*slijed, sortiranje*
Z7	Božić se slavi 25. prosinca. Zamisli da je Božić 2 dana prije četvrtka. Kojeg je dana Božić, a koji će dan biti 4 dana iza Božića i koji će to biti datum.	*3*	*slijed*
Z8	Dražen, Ana i Ivana imaju zanimanja učitelj, prodavač i kuhar. Dražen je niži od Ane, ali viši od Ivane. Prodavač je najviši, a kuhar najniži. Kojeg su zanimanja Dražen, Ana i Ivana? Tko je najviši, a tko najniži?	*4*	*slijed, sortiranje*

Na papiru su trebali i opisati kako su došli do rješenja. Kako su svi učenici početnici u programiranju tako nije bilo potrebo ispitivati njihovo predznanje. Zbog toga je bio cilj ispitati sposobnost rješavanja problema kako bi se kasnije rezultati sa završnih ispita znanja mogli usporediti sa sposobnosti rješavanja problema. Zadaci su preuzeti iz natjecanja Klokan bez granica za uzrast učenika petih razreda. Zbog uzrasta učenika i eventualne frustracije, test je namjerno napravljen s nešto više jednostavnijih zadataka. Na taj način učenici nisu bili

demotivirani za pisanje ovog ispita koji se nije ocjenjivao i bio je dobrovoljan. Osim nešto olakšanog testa učenici su motivirani plusom za aktivno sudjelovanje u ispitu. Za ispit su izabrani zadaci koji ne traže minimalno matematičko znanje već sposobnost rješavanja problema koji uključuje korištenje algoritama. Zadaci su sadržavali probleme: prepoznavanja uzoraka, sortiranja, slijeda, uvjeta te ponavljanja. Kako zadaci nisu bili jednake težine tako su različito i bodovani s čim učenici nisu bili upoznati zbog smanjenja eventualne frustracije.

U drugoj fazi istraživanja učenici su nakon četiri tjedna obrađivanja nastavne cjeline programiranja u Pythonu pisali završni ispit. Završni ispit znanja u Pythonu je napravljen za utvrđivanje mjere znanja nakon ciklusa poučavanja. Učenici su završni ispit znanja u Pythonu pisali na papiru. Prije rješavanja zadataka učenici su, također na papiru, mogli odgovoriti na jedno anketno pitanje o stavu prema programiranju (slika 4.3).


Zaokruži ocjenu od 1 do 5 koliko ti se sviđa programiranje.

1	2	3	4	5
Uopće mi se ne sviđa	*Nije mi baš nešto*	*Niti mi se sviđa, niti mi se ne sviđa*	*Dobro je*	*Jako mi se sviđa*

Slika 4.3 - Anketno pitanje u drugoj fazi istraživanja

Prije treće faze učenici su imali tri tjedna božićnih praznika. U trećoj fazi istraživanja učenici su nakon tri tjedna obrađivanja nastavne cjeline programiranje programiranjem igara u Scratchu, na papiru pisali završni ispit znanja u Scratchu te on-line anketu o zadovoljstvu programiranjem i programskim jezicima (tablica 4.19).

Tablica 4.19- Anketa o stavu prema programiranju i programskim jezicima

	Pitanje	**Ishod**	**Tip zadatka**
Nakon Pythona	P1	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje	Likertova skala (1-5)
Nakon Scratcha	P2	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje	Likertova skala (1-5)
	P3	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Python.	Likertova skala (1-5)
	P4	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Scratch.	Likertova skala (1-5)
	P5	Koji ti se programski jezik više sviđa?	Zadatak višestrukog izbora s jednim odgovorom
	P6	Ako imaš neki svoj komentar u vezi programiranja slobodno ga napiši.	Pitanje otvorenog tipa

Pri izradi ispita znanja pazilo se da budu obuhvaćeni obrađeni koncepti programiranja te da učenici nisu ciljano uvježbavani za ispitne zadatke. Zadaci si bili različitih težinskih razina, od lakših prema težim. Preliminarni ispit znanja je ciljano napravljen lakšim kako se ne bi smanjila motivacija i povećala frustracija učenika pri rješavanju ispita.

Prije analize podataka bilo je potrebno napraviti analizu pouzdanosti ispita. U tablici 4.20 su prikazane karakteristike ispita.

Tablica 4.20 - Karakteristike ispita korištenih u istraživanju

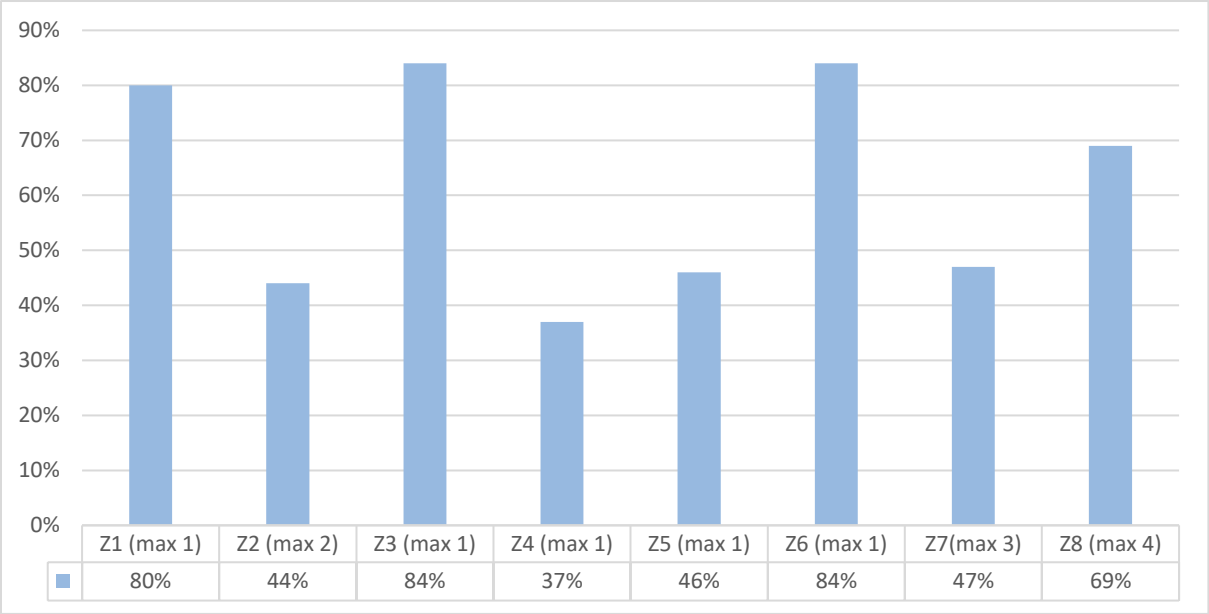
	PI sposobnosti rješavanja problema	ZI Python	ZI Scratch
n	50	50	50
aritmetička sredina	8.22	16.54	7.34
aritmetička sredina (u postocima)	58.71%	59.072%	66.714%
M (Medijan)	10	16	7.5
Mo (Mod)	10	27	8
SD	3.765	7.765	2.273
Minimum	0	0	2
Maksimum	14	28	11
Maksimalan mogući broj bodova	14	28	11
Broj čestica	8	11	11
Cronbach α	0.665	0.861	0.771
Kolmogorov-Smirnov test	0.000	0.197	0.069

Za pouzdanost ispita korišten je Cronbach α koeficijent. Završni ispit za Python spada u kategoriju visoko pouzdanih (Cronbach α=0.861), završni ispit Scratch u visoko pouzdane (Cronbach α=0.771), dok preliminarni ispit sposobnosti rješavanja problema spada u granično pouzdane (0.665). Iako bi se pouzdanost mogla povećati većim brojem čestica, već su navedeni razlozi zbog kojeg je preliminarni ispit rješavanja problema namjerno napravljen lakšim. Zbog toga ne iznenađuje što nema ni normalne distribucije među rezultatima. Preliminarnim ispitom znanja se ne ispituje konkretno znanje već služi za utvrđivanje ujednačenosti skupina (razreda), te za svrstavanje učenika u težinske grupe pa se ovakav rezultat smatra zadovoljavajućim.

Za analizu distribucije rezultata korišten je Kolmogorov-Smirnovljev test jer je uzorak ispitanika veći od 50. Rezultati završnih ispita u Pythonu (p=0.197) i Scratch (p=0.069) su normalno distribuirani.

4.2.2.1 Preliminarni ispit sposobnosti rješavanja problema

Na slici 11 su prikazani indeksi težine zadataka preliminarnog ispita znanja. Kako je indeks težine cijelog ispita blizu 50% (58.71%) ovaj se ispit znanja može svrstati u kategoriju testova umjerene težine (Cohen et al., 2013). Kada se gledaju rezultati pojedinačnih zadataka tada četiri zadataka (Z1, Z3, Z6, Z8) spadaju u lagane zadatke (>67%), dok svi ostali zadaci spadaju u grupu zadataka umjerene težine (33%–67%).



Slika 4.4 - Indeksi težine zadataka PI

Analizom rezultata po zadacima vidi se kako su učenici bili najmanje uspješni u zadatku s uvjetom (Z4). Zadatak je sadržavao i logičku operaciju I (*AND*) što je očito učenicima teže intuitivno prepoznati.

Najbolji rezultati postignuti su u zadacima sa sortiranjem (Z6), čak i u zadatku koji sadrži i dvostruko sortiranje (Z8) rezultati su iznadprosječni.

Što se tiče zadataka s ponavljanjem, zadatak s jednostavnim ponavljanjem (Z3) također spada u najbolje riješene zadatke kao i Z1 s jednostavnim ponavljanjem i prepoznavanjem uzorka. Problemi za učenike nastaju u složenijim zadacima ponavljanja s prepoznavanjem uzorka (Z2) gdje teže uočavaju složeniji uzorak.

Za daljnju analizu učenici su prema rezultatima PI sposobnosti rješavanja problema podijeljeni u tri težinske grupe: *Bolji*, *Srednji* i *Lošiji*. Raspodjela je prikazana u tablici 4.21.

Tablica 4.21 - Distribucija rezultata ispita prema težinskim grupama

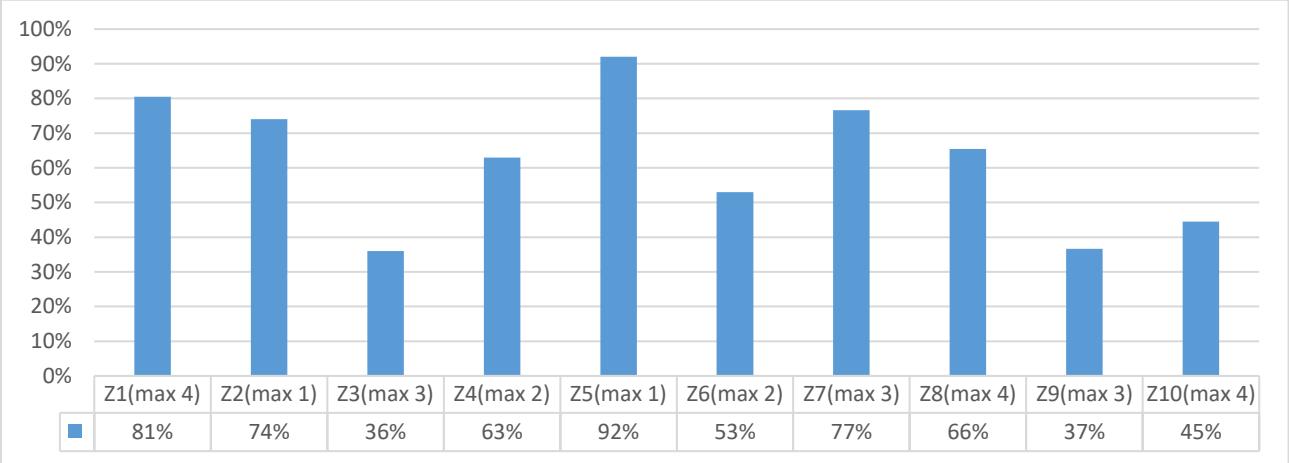
		PI			ZI Python		ZI Scratch	
Grupa	n	Bodovi PI (max.14)	AS	SD	AS	SD	AS	SD
Bolji	15	>=11	11.93	1.223	22.87	5.743	8.47	2.167
Srednji	16	8-11	9.69	0.704	17.31	6.311	7.88	2.094
Lošiji	19	<=7	4.05	2.97	10.89	6.145	6.0	1.886

Legenda:

N – broj učenika
AS – aritmetička sredina
SD – standardna devijacija

4.2.2.2 Završni ispit znanja u Pythonu

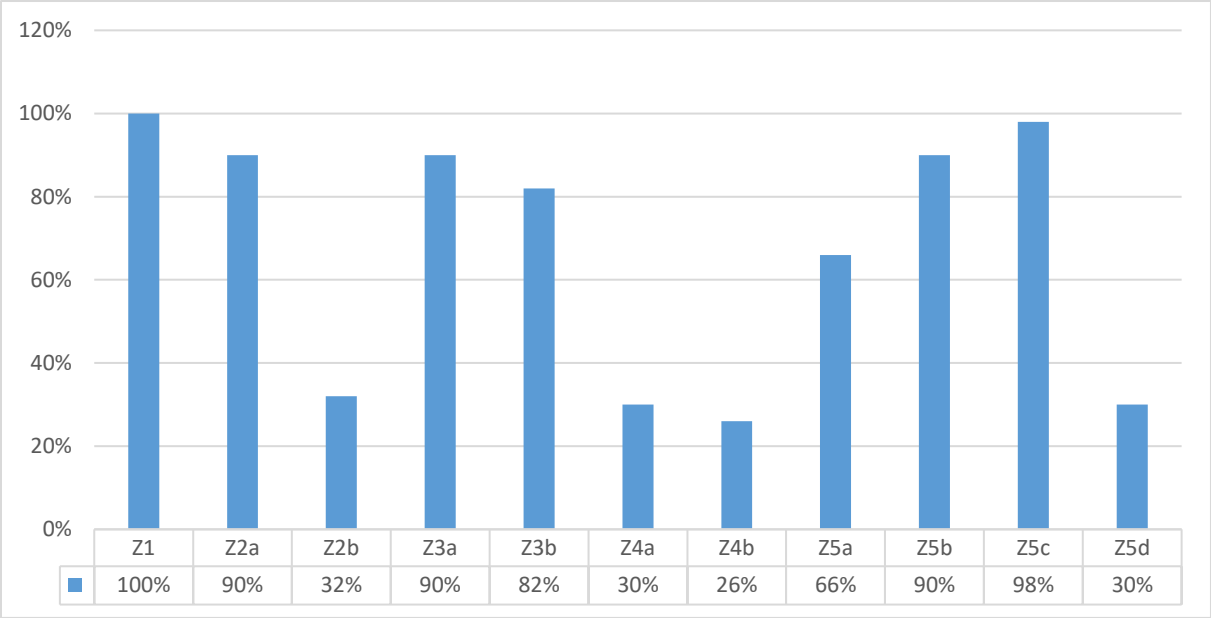
Na slici 4.5 prikazani su indeksi težine zadataka završnog ispita znanja u Pythonu. Prema indeksu težine (59.072%) ovaj ispit znanja može se svrstati u kategoriju ispita umjerene težine. Četiri zadataka (Z1, Z2, Z5, Z4) spadaju u lagane zadatke (>67%), dok svi ostali zadaci spadaju u grupu umjerene težine (33%–67%).



Slika 4.5 - Indeksi težine zadataka ZI Python

4.2.2.3 Završni ispit znanja u Scratchu

Na slici 4.6 su prikazani indeksi težine zadataka završnog ispita znanja u Scratchu. Prema indeksu težine (66.714%) ovaj se test može svrstati u kategoriju testova umjerene težine. Šest zadataka (Z1, Z2a, Z3a, Z3b, Z5b, Z5c) spadaju u lagane zadatke (>67%), zadaci Z2b, Z4b, Z5d spadaju u teške (<33%), dok svi ostali zadaci spadaju u grupu umjerene težine (33%–67%).



Slika 4.6 - Indeksi težine zadataka ZI Scratch

4.2.2.4 Utvrđivanje razlika između grupa

Za daljnju analizu podataka potrebno je utvrditi jesu li ujednačeni rezultati preliminarnog ispita i nezavisnih varijabli spol, razred i škola kako bi mogli nastaviti s daljnjom analizom. Za utvrđivanje razlika korišten je Mann-Whitney U test prema kojem nisu nađene statistički značajne razlike po spolu (U=268.0, p=0.933) ni po školama (U=187.5, p=0.158). Kruskal-Wallis test je korišten za utvrđivanje razlika između razreda s obzirom na rezultate preliminarnog ispita. Ni tu nema statistički značajnih razlika za (χ^2(3)=6.226, p=0.101).

Za utvrđivanje razlika u rezultatima završnih ispita (ZI) Pythona i Scratcha prema spolu i školi korišten je t-test. T-test nije pokazao statistički značajne razlike između spola (t(48)=-1.312, p=0.196) i ZI Python kao ni između spola i ZI Scratch (t(48)=-1.283, p=0.205). T-test također nije pokazao statistički značajne razlike u rezultatima između škola u odnosu na ZI Python (t(48)=0.42, p=0.677) i u odnosu na ZI Scratch (t(48)=0.864, p=0.392). Prema rezultatima jednosmjenog ANOVA testa utvrđeno je da nema statistički značajne razlike između razreda u odnosu na rezultate u ZI Python (F=0.68, p=0.569) i ZI Scratch (F=0.96, p=0.42), odnosno da su rezultati završnih ispita ujednačeni po razredima.

Pri utvrđivanju distribucije rezultata prema težinskim grupama Shapiro-Wilkovim testom utvrđeno je da nema normalne distribucije za težinsku grupu *Bolji* u ZI Python rezultatima (tablica 4.22). Zbog toga se za tu skupinu u daljnjem ispitivanju koristio neparametrijski test Kruskal-Wallis dok se za ostale kombinacije koristio parametrijski test jednosmjerna ANOVA.

Tablica 4.22 – Rezultati Shapiro-Wilkovog testa ZI po težinskim grupama

	ZI Python Shapiro-Wilk p	ZI Scratch Shapiro-Wilk p
Bolji	0.006	0.095
Srednji	0.278	0.253
Lošiji	0.504	0.505

4.2.3 Usporedba rezultata završnih ispita s preliminarnim ispitom sposobnosti rješavanja problema

Za usporedbu rezultata završnih ispita s preliminarnim ispitom sposobnosti rješavanja problema korišten je Man-Whitney U test. U tablici 4.23 su prikazani rezultati testa.

Tablica 4.23 - Rezultati Mann-Whitneyevog testa za ZI po težinskim grupama

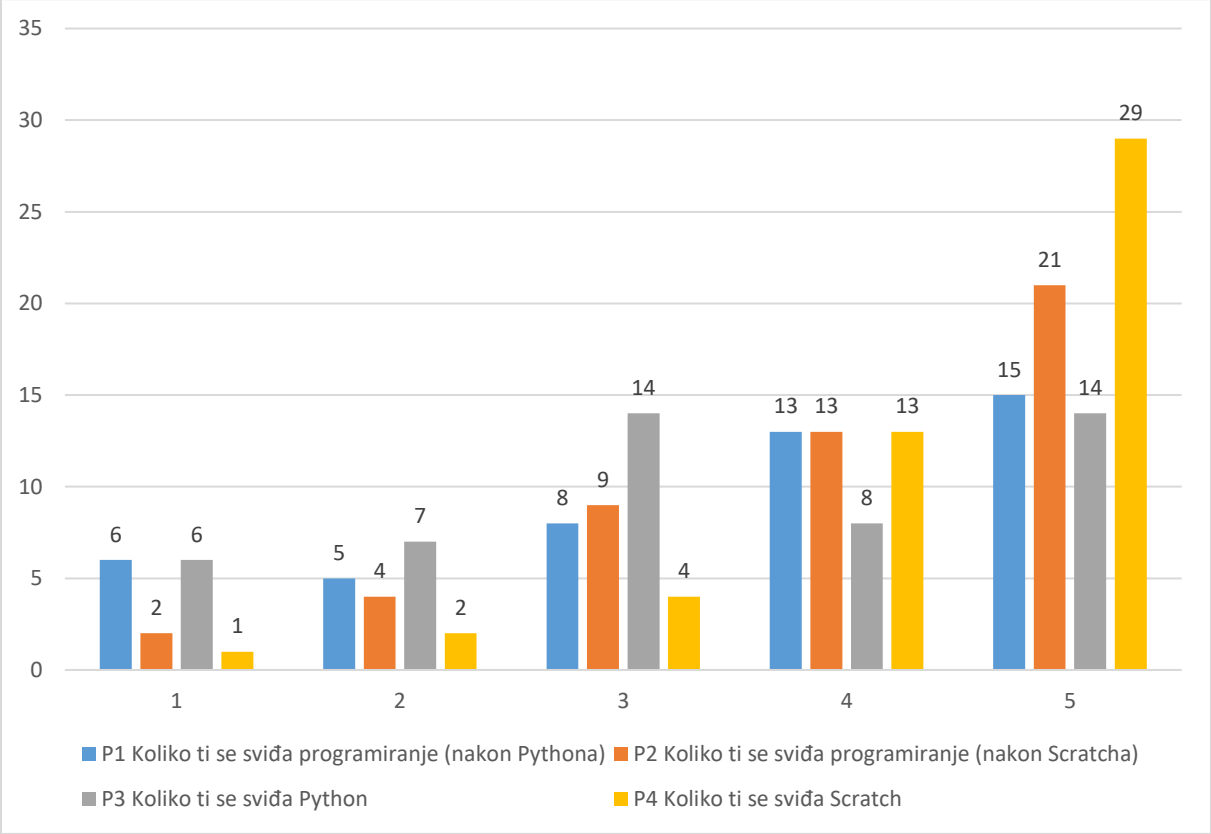
		ZI-Python	ZI-Scratch
Bolji - Srednji	U	62.5	99.5
	z	-2.288	-0.828
	p	0.022*	**0.408**
Bolji - Lošiji	U	24.500	52.000
	z	-4.101	-3.169
	p	0.000*	0.002*
Srednji-Lošiji	U	74.500	76.000
	z	-2.544	-2.548
	p	0.010*	0.011*

Kao što je vidljivo u tablici 4.23, postoje statistički značajne razlike u rezultatima ZI Python između težinskih grupa učenika gdje u pravilu bolje rezultate postižu učenici iz viših težinskih grupa u odnosu na niže. Prema ovim rezultatima možemo prihvatiti hipotezu H1 i zaključiti da učenici s boljom sposobnosti rješavanja problema ostvaruju bolje rezultate u Pythonu.

U slučaju usporedbe rezultata učenika po težinskim grupama u ZI Scratch nema statistički značajne razlike između grupa *Bolji* i *Srednji*. Prema ovim rezultatima se može zaključiti kako sposobnost rješavanja problema nije nužno prediktor uspjeha u Scratchu, odnosno ne možemo zaključiti kako je sposobnost rješavanja problema prediktor uspjeha u Scratchu. Ova pojava se može interpretirati razinom apstrakcije potrebnom za savladavanje vizualno-blokovskog programskog jezika i tekstualnog programskog jezika gdje je za savladavanje tekstualnog programskog jezika potrebna veća razina apstrakcije. Prema navedenom možemo odbaciti hipotezu H2, odnosno prihvatiti nulhipotezu kako učenici s boljom sposobnosti rješavanja problema ne ostvaruju bolje rezultate u Scratchu.

4.2.4 Stav prema programiranju

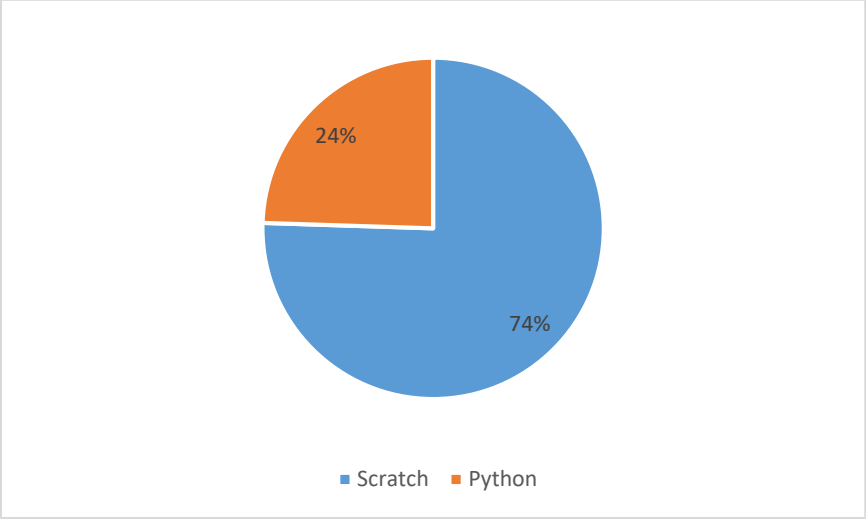
Stav prema programiranju ispitivao se anketnim pitanjima. Na prvo pitanje o stavu prema programiranju učenici su odgovarali, u on-line obliku, nakon tretmana u Pythonu, a prije popunjavanja ispita znanja u Pythonu. Na isto i ostala pitanja učenici su odgovarali nakon tretmana u Scratchu u on-line obliku. Kako je ispunjavanje anketnih pitanja bilo dobrovoljno tako dio učenika nije ispunio anketu, pa je na pitanje nakon prvog dijela tretmana (nakon Pythona) na pitanje P1 odgovorilo 47 učenika, a na ostala pitanja, na koja se odgovaralo nakon cijelog tretmana, odgovorilo 49 učenika. Za usporebe stavova učenika prema programiranju nakon Pythona i nakon Scratcha, te usporedbe ocjena Pythona i Scratcha korišten je Wilcoxonov test. Pri usporedbi stavova prema programiranju rezultat testa je pokazao da postoji statistički značajna razlika u stavovima učenika u prema programiranju nakon korištenja Scratcha i Pythona (Z=-3.735, p=0.000). Frekvencije odgovora na anketna pitanja prikazane su na slici 4.7.



Slika 4.7- Frekvencije odgovora na anketna pitanja

Prema frekvencijama odgovora vidljivo je kako se učenicima više sviđa programiranje nakon Scratcha (P1-P2), u usporedbi sa njihovim dojmom o programiranju nakon Pythona. Osim toga

očita je razlika u stavu prema programskom jeziku Scratch kojeg je 29 od 49 učenika ocijenilo najvećom ocjenom (5).



Slika 4.8 - Rezultat izbora između Scratcha i Pythona kao preferiranog programskog jezika

Što se tiče izbora jezika čak 74% (n=37) učenika preferira Scratch u odnosu na Python kojeg preferira 24% (n=12) učenika (slika 4.8). Važno je napomenuti kako dio učenika koji preferira Python uglavnom pripada učenicima iz težinske skupine *Bolji*, koji su lakše savladavali gradivo u Pythonu, a dio njih je sudjelovao na natjecanju iz programiranja u Pythonu, pa ne iznenađuje da im je taj programski jezik i draži. Navedeno je opet u skladu s rezultatima prethodnog istraživanja kod kojeg se uočilo kako učenici koji imaju iskustvo programiranja u tekstualnom programskom jeziku, te uz to imaju bolje sposobnosti rješavanja problema, više preferiraju „pravi“ programski jezik, odnosno blokovski jezik ne smatraju „pravim“ programskim jezikom.

Što se tiče komentara koji su učenici imali priliku dati nakon cijelog tretmana, od ukupno 49 učenika koji su popunjavali anketu 30 učenika je dalo pisani komentar. Od svih komentara neki se odnose na programiranje općenito, neki na programske jezike, a neki na sve.

Komentari učenika će biti prikazani točno kako su napisani (sa svim gramatičkim i drugim greškama).

Od svih komentara 12 ih je bilo o programiranju općenito s pozitivnim izjavama. Komentari su sljedeći:

- *Programiranje je COOOOOOOOOOOOOOOOOOOOOL!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!*
- *PROGRAMIRANJE JE ZAKON*
- *jako mi se sviđaju oba programiranja i mislim da bi i sljedeće godine trebalo biti tako*

- *Super je,samo bih htio da radimo malo više naredbi*
- *Meni je programiranje jako zanimljivo i puno sam toga naučila o njemu. Nadam se da ćemo ga koristiti i u sljedećoj godini.*
- *Meni se samo svidjelo samo izraćunavanje kalkulatura.*
- *Scratch i Python su odlični,šteta što smo gotovi s programiranjem za 5.razred*
- *misli da programiranja jer naucin nesto novo*
- *U 8. zadatku meni se jednako svidio "Python" i "Scratch" pa sam samo stavio "Python" jer sam bio na natjecanju (školskom i županiskom) iz programiranja (Python). Zato sam stavio "Python",ali mi se i dalje jednako sviđaju oba programska jezika.*
- *DOBRO JE I SVIĐA MI SE I OD SCRATCH I PYTHONA PUNO SMO NAUCILI:). SVE POHVALE PROF. KOJE SU NAS NAUCILE PUNO.BRAVO!!*
- *Sviđa mi Scratch i Pyton ,ali mi je Pyton sviđa za atom više.:)*
- *programiranje mi je super i jako zanimljivo*

Među komentarima se može vidjeti nekoliko njih koji preferiraju Python, odnosno neki učenici su naglasili kako im je Python malo draži. To su upravo oni učenici iz težinske supine *Bolji.*

Među općenitim komentarima o programiranju 8 ih je imalo negativan stav prema programiranju. Komentari su sljedeći:

- *PONEKAD JE SUPER A PONEKAD JE DOSADNO*
- *možda malo težak , ali zna biti zabavan*
- *ja ne volim,PROGRAMIRANJE*
- *jako mi je tezak i dugo traje i nije bas zanimljiv i ne volim ga*
- *Pa, to baš i ne znam zato jer je za mene to nerazumno*
- *MALO JE KOMPLICIRANO.:)*
- *bas je dosadno*
- *samo mi je malo teško*

Učenici s negativnim komentarima su mahom učenici iz skupine *Lošiji*, odnosno učenici kojima je programiranje izuzetno zahtjevno zbog potrebne razine apstrakcije.

Ukupno 10 komentara izrazilo se pozitivno prema programiranju uz isticanje Scratcha kao željenog programskog jezika. Komentari su sljedeći:

- *SCRATCH JE ZAKOOOOON!!!!!!!!!!*
- *svidja mi se al mi je vise drazi scratch nego python*

- *super je scratch, ali nijemi nešto python. Željelabi da sljedece godine imamo više scratch*
- *scratch je puno bolji od pythona*
- *Scratch je super,ali pyton mi nije bas nesto.Volim raditi igrice idoma imam scratch*
- *scratch je superzabavan i zanimljiv.lijepo je vidjeti igru koju si sam napravio.pyithon je malo onako dosadan i težak.*
- *MENI SE PROGRAMIRANJE MALO SVIĐA,ALI SAM SE DOBRO ZABAVIO NA SCRATCHU.*
- *mislim da je puno bolje programiranje u scratchu zato sto je puno zabavnije*
- *Pa programiranje u pytonu mi je bilo malo dosadnije od onog u scratchu , ali mi je scratch jako zabavan i sviđa mi se pa bih ga volila imati kod kuće .*
- *Sviđa mi se i programiranje ali mi se sviđa i scratch nekad bi mi se više dalo praviti igrice a nekad programiranje. ponekad bi mi malo dosadilo kada bi dugo radila ali ipak su i python i scratch i dobri.*

Iz navedenih komentara može se uočiti kako više učenika naglašava kontekst programiranja, te ističu kako su se zabavili programirajući igre u Scratchu što može biti ključno za motivaciju prema programiranju općenito.

4.2.5 Ograničenja istraživanja, rasprava i zaključak

Kao i u prethodnom istraživanju učiteljica je bila istraživačica koja je ujedno imala ulogu i sudjelujućeg opažavača. Opet se može potvrditi kako se korištenjem Scratcha u poučavanju programiranja brzo može uvesti veći broj programskih koncepata učenicima na konkretan način. Algoritmi slijeda, ponavljanja i grananja se u blokovskom jeziku mogu uvesti već na prvoj vježbi , na primjer, simulaciji akvarija, kod koje se za pokretanje likova uvode algoritmi slijeda i ponavljanja, a algoritam grananja se uvodi za dodatne funkcionalnosti kao što je situacija u kojoj morski pas treba „pojesti“ ribicu. Svi navedeni problemi su djeci konkretni i jasni pa je time i uvođenje pozadinskih pojmova programiranja „prirodno“. Kod proceduralnog programiranja vrijedi potpuno drugi pristup. Pojmovi algoritama prvo se uvode kroz primjere iz svakodnevnog života koji se zapravo puno teže prenose u kontekst proceduralnog programskog jezika. Osim toga učenici se bore sa sintaksom tako da se u početku proceduralnog programiranja učenici bore sa sintaksom, rješavanjem problema i apstrakcijom potrebnom da bi se zadani zadaci mogli riješiti programskim jezikom.

U ovom istraživanju pokazalo se da su kod programiranja u tekstualnom programskom jeziku Pythonu koristeći proceduralni pristup učenici s višom sposobnosti rješavanja problema značajno uspješniji. Isto nije slučaj kod programiranja u vizualno-blokovskom programskom jeziku Scratchu gdje sposobnost rješavanja problema nije ključan faktor uspješnosti. Razlog vjerojatno leži u principu „od konkretnog prema apstraktnom“ gdje je kod Scratcha potrebna razina apstrakcije niža u odnosu na Python jer je rezultat izvršavanja naredbi vidljiv kroz kretanje likova u mikrosvijetu te je samim time i otklanjanje grešaka jednostavnije i uočljivije.

Osim toga istraživanje je pokazalo povećanu motivaciju za programiranjem nakon korištenja Scratcha. Najveći problem imali su učenici s nižom sposobnosti rješavanja problema kojima je programiranje samo po sebi iznimno zahtjevno. Ono što je najvažnije, očito se može utjecati na „srednju trećinu“ učenika koji još nisu dosegli visoku razinu apstrakcije, ali mogu savladati osnovne algoritme pri čemu im vizualni-blokovski jezik može pomoći.

Ograničenje ovog istraživanja je nemogućnost bolje usporedbe učinkovitosti između obaju konteksta zbog razlika između korištenih programskih jezika Scratch i Python. Cilj je na kraju učenike uvesti u proceduralno programiranje korištenjem tekstualnog programskog jezika pri čemu se vizualni-blokovski jezik može koristiti za posredovani transfer iz jednog konteksta u drugi. Kako bi se detaljnije utvrdila mogućnost posredovanog transfera, provest će se još jedno istraživanje na tu temu koje će biti obrađeno u četvrtoj fazi ovog cjelovitog istraživanja pri čemu će se odabrati vizualni-blokovski programski jezik u kojem će se moći izraditi analogni zadaci kao u Pythonu.

4.3 Utjecaj korištenja programskog jezika na razumijevanje koncepta petlje

Treća faza istraživanja svojevrsni je nastavak prethodnih dviju faza istraživanja (M. Mladenović et al., 2017; M. Mladenović, Krpan, et al., 2016; M. Mladenović, Rosić, et al., 2016). U prvoj fazi se uspoređivao utjecaj programskih jezika Logo i Scratch na razumijevanje programskih koncepata i uspjeh u završnim ispitima znanja (M. Mladenović, Rosić, et al., 2016). Kako je istraživanje pokazalo statistički značajnu razliku u razumijevanju koncepta petlje tako se u ovom istraživanju usmjerilo upravo na taj koncept. Kako je jedno od ograničenja prvog istraživanja bio mali uzorak, tako se ovim istraživanjem obuhvatio veći broj učenika različitih uzrasta, razreda, škola i nastavnika. Drugo ograničenje bilo je nedostatak preliminarnog ispita prema kojem bi se provjerila ujednačenost grupa. U ovom istraživanju se stoga proveo takav preliminarni ispit sposobnosti rješavanja problema kakav je proveden u drugoj fazi istraživanja. U drugoj fazi istraživanja uspoređivala se učinkovitost učenika u tekstualnom programskom jeziku Python prema proceduralnom pristupu i vizualno-blokovskog jezika Scratch. Zbog razlike u jezicima i pristupima nije se moglo izraditi analogne zadatke u oba programska jezika. U ovom istraživanju se Python kao tekstualni programski jezik obrađivao prema pristupu temeljenom na kornjačinoj grafici pa su se u ovom istraživanju mogli izraditi analogni zadaci za razmatrane jezike. Tako su se u ovom istraživanju koristila tri programska jezika, s jedne strane Scratch kao vizualno-blokovski programski jezik a s druge strane Logo i Python kao tekstualni programski jezici s grafičkim prikazom programa.

Postavljene hipoteze u ovoj fazi istraživanja su:

H1 – učenici (od petog do osmog razreda) koji su učili programiranje koristeći Scratch ostvarit će bolje rezultate na završnom ispitu znanja od učenika koji su koristili Logo i Python

H2 – početnici u programiranju (učenici petih razreda) koji su učili programiranje koristeći Scratch ostvarit će bolji uspjeh na završnom ispitu znanja od učenika koji su koristili Logo i Python

4.3.1 Opis istraživanja

Istraživanje je provedeno tijekom školske godine 2015./2016. u kojem je sudjelovalo 20 razreda iz četiri osnovne škole tijekom redovne nastave izbornog predmeta informatike u prirodnom okruženju bez prisustva trećih osoba. Nastava se odvijala prema redovnom rasporedu od dva sata tjedno kao blok sat. Istraživanjem je obuhvaćen uzrast učenika od petih do osmih razreda (11-15 godina) kojih je ukupno bilo 312. Nastavu je držalo ukupno šest učitelja u svojim razredima. Zbog izbornosti predmeta i različitih uzrasta učenika, neki su učenici imali

prethodno iskustvo programiranja, a neki nisu. Uzorak populacije istraživanja je neslučajan i ciljan jer je istraživala ciljana skupina populacije (Cohen et al., 2013), u ovom slučaju učenici osnovnih škola od petih do osmih razreda koji su upisali izborni predmet informatika. U tablici 4.24 su prikazani podaci o sudionicima.

Tablica 4.24 – Podaci o sudionicima

Škola	Učitelj	Razred	Korišteni programski jezik	Broj razreda	Broj učenika	Dječaci	Djevojčice	Prethodno iskustvo programiranja
OŠ Blatine-Škrape	U1	5	Scratch	2	30	14	16	Ne
OŠ Blatine-Škrape	U1	6	Python	1	17	7	10	Da(Python)
OŠ Mejaši	U2	5	Logo	3	48	22	26	Ne
OŠ Mejaši	U3	6	Logo	4	54	31	23	Da (Logo)
OŠ Mejaši	U4	7	Logo	2	35	18	17	Da (Logo)
OŠ Mejaši	U4	8	Logo	3	48	30	18	Da (Logo)
OŠ Split 3	U6	5	Python	3	51	20	31	Ne
OŠ Bol	U1	6	Scratch	1	16	11	5	Ne
OŠ Bol	U1	7	Scratch	1	13	6	7	Ne
			Ukupno:	20	312	159	153	

U svim razredima nastavna cjelina programiranja odvijala se prema pristupu A. Zbog čega je bilo moguće obuhvatiti tri vrste programskih jezika:

- Logo kao najzastupljeniji prema pristupu A;
- Python korištenjem kornjačine grafike što omogućava korištenje istih zadataka i istog pristupa s vrlo sličnom sintaksom kao kod Loga;
- Scratch kao vizualni-blokovski jezik u kojem se mogu obraditi sve zadane teme pristupa A nastavne cjeline programiranja.

Kako su ovom istraživanju obuhvaćeni razredi od petog do osmog nastava je trebala obuhvatiti teme zajedničke svim razredima, a to su teme petog razreda koje obuhvaćaju algoritme slijeda i ponavljanja. Zbog ograničenog vremena i tema za istraživanje, prema rezultatima prvog istraživanja (M. Mladenović, Rosić, et al., 2016), u kojem se pokazala statistički značajna razlika u razumijevanju koncepta petlje bez logičkog uvjeta, ovo istraživanje usmjerilo se na upravo taj koncept. Kako se kod pristupa A koncept petlje uvodi već u petom razredu i prema programu se koristi kasnije u svim razredima, tako su svi učenici upoznati s navedenim konceptom. U tablici 4.25 se prikazuju naredbe korištene u testovima u tri korištena programska jezika: Scratch, Logo i Python. Već je navedeno kako je sintaksa Loga i Pythona vrlo slična, praktički jedina razlika u korištenim primjerima i zadacima u nastavi je u sintaksi petlje koja je kod Pythona složenija.

Tablica 4.25 - Primjer petlje bez logičkog uvjeta u korištenim programskim jezicima

Scratch	LOGO	Python	Značenje
	repeat 10[…]	for i in range (10): …	Ponovi 10 puta
	fd (100)	fd (100)	Naprijed (eng. **forward - fd**)
	rt (90)	rt (90)	Skreni desno za 90° (eng. **r**ight **t**urn - **rt**)
	lt (90)	lt (90)	Skreni lijevo za 90° (eng. **l**eft **t**urn - **lt**)

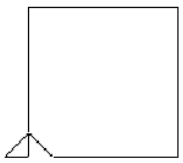
Naredbe u Scratchu su vizualno prikazane u obliku blokova slagalice te kodirane bojama ovisno o funkcionalnosti. Tako su naredbe za kretanje obojene plavom bojom, a za upravljanje tijekom programa žutom bojom. Jasno je kako učenici na ovakav način nemaju problem sa sintaksom te kako je značenje naredbi vrlo intuitivno i bez posebnog objašnjavanja. Zbog toga se fokus učenika sa sintakse lakše usmjerava na rješavanje problema.

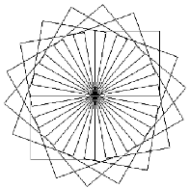
U programskim jezicima Logo i Python naredbe su tekstualne što je u startu dodatan problem učenicima jer moraju točno napisati naredbe kako bi se program uopće mogao izvršiti. Naredbe za kretanje su skraćenice engleskih riječi (*fd, rt, lt*). Već je spomenuto da se Logo i Python malo razlikuju u sintaksi naredbi korištenih u nastavi. Značajna je razlika u sintaksi za petlju bez logičkog uvjeta koja je kod Loga jednostavnija (*repeat[]*) dok je kod Pythona složenija i ne baš intuitivna (*for i in range(10)*).

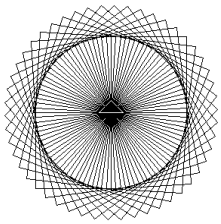
Istraživanje je provedeno kao kvazieksperiment. Osim različitih programskih jezika korišten je i različiti kontekst programiranja. U kontrolnoj skupini se programiranje obrađivalo korištenjem programskih jezika Logo i Python na „tradicionalan“ način, dok se u eksperimentalnoj skupini koristio programski jezik Scratch u kontekstu programiranja jednostavnih igara. Zbog kompleksnosti izvođenja što uključuje velik broj naredbi kao i koncepata programiranja, programiranje igara nije niti moguće izvesti na ovoj razini školovanja u tekstualnim programskim jezicima.

U tablici 4.26 je prikazan dizajn istraživanja za kontrolnu skupinu.

Tablica 4.26 - Dizajn istraživanja za kontrolnu skupinu (Logo i Python)

Logo i Python grupe (Kontrolna skupina)				
tjedan	**Tema**	**Novi pojmovi i naredbe**	**Opis**	**Primjer programa**
1	Preliminarni ispit sposobnosti rješavanja problema (PI)			
2	Crtanje osnovnih geometrijskih likova	fd, bk, rt, lt, repeat *Primjer kôda:* `repeat 4[fd 100 rt 90]`	Uvođenje algoritama slijed i ponavljanje crtanjem jednostavnih geometrijskih likova uz korištenje osnovnih naredbi za kretanje i ponavljanje	

3	Crtanje složenijih likova	Procedure
		
		Primjer kôda s ugniježđenom petljom:
		`repeat 18[repeat 4[fd 100 rt 90] rt 360/18]`
		Primjer kôda s procedurom:
		`to cvijet`
		`repeat 18[kvadrat  rt 360/18]`
		`end`
			Crtanje kompleksnijih likova korištenjem ugniježđenih petlji i procedura.	

4	Crtanje složenijih likova	Procedure s parametrima, varijable
		
		Primjer kôda:
		`to cvijet :n`
		`repeat :n[kvadrat  rt 360/:n]`
		`end`
			Crtanje kompleksnijih likova korištenjem procedura s parametrima	

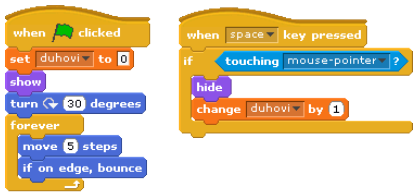
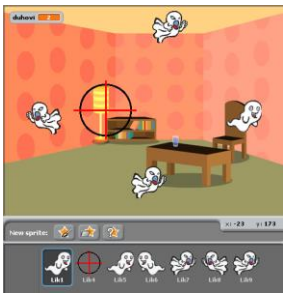
5	Završni ispit znanja (ZI)			

U tekstualnim programskim jezicima radi se na „tradicionalan“ način koji se svodi na crtanje različitih geometrijskih oblika. Pri tome se može vidjeti kako se koncept petlje koristi u svakoj temi te kako se koristi i ugniježđena petlja za crtanje složenijih likova.

Zbog prirode blokovskog programskoj jezika Scratch programiranje igara, simulacija, i slično, se nameće kao prirodan kontekst programiranja. Iz dizajna istraživanja za obje grupe može se uočiti kako se u Scratchu obrađuje veći broj koncepata i naredbi dok se u tekstualnim programskim jezicima dulje zadržava na uvedenim pojmovima. Osim toga, u eksperimentalnoj skupini nema zadataka iz „tradicionalnog“ pristupa.

U tablici 4.27 prikazan je dizajn istraživanja za eksperimentalnu skupinu.

Tablica 4.27 -Dizajn istraživanja za eksperimentalnu skupinu (Scratch)

Scratch grupe (Eksperimentalna skupina)				
tjedan	**Tema**	**Novi pojmovi i naredbe**	**Opis**	**Primjer programa**
1	Preliminarni ispit sposobnosti rješavanja problema (PI)			
2	Simulacija akvarija	Lik (sprite), naprijed (forward), lijevo (left), desno (right), čekaj (wait), sljedeći kostim (next costume), ponavljaj (repeat), istovremeno izvršavanje naredbi *Primjer kôda:* 	Uvođenje algoritama slijeda i ponavljanja izradom simulacije akvarija uz korištenje likova te njihovog pomicanja i istovremenog izvršavanja naredbi (paralelnost)	
3	Igra Hvatanje duhova	Ako (If), varijable, događaji (dodirivanje drugog lika ili miša), prikaži (show) *Primjer kôda:* 	Uvođenje uvjetnog izvršavanja programa programiranjem igre *Hvatanje duhova*. Uvođenje pojma varijable potrebne za brijanje bodova u igri	
4	Igra Odbijenac	Slanje poruke (Broadcast message), petlja s uvjetnim izvršavanjem (ponavljaj ako - forever if) *Primjer kôda:* 	Uvođenje petlje s uvjetnim izvršavanjem, slanje poruka programiranjem igre *Odbijenac*	
5	Završni ispit znanja (ZI)			

4.3.2 Instrumenti

Instrumenti koji su se koristili u ovom istraživanju korišteni su u prethodna dva istraživanja.

Korišteni instrumenti:

- Preliminarni ispit (PI) sposobnosti rješavanja problema;
- Završni ispit znanja u Logo (ZIL) programskom jeziku;
- Završni ispit znanja u Python (ZIP) programskom jeziku;

- Završni ispit znanja u Scratch (ZIS) programskom jeziku;

4.3.2.1 Preliminarni ispit (PI) sposobnosti rješavanja problema

Iako nisu svi učenici početnici u programiranju, nije proveden preliminarni ispit znanja već preliminarni ispit sposobnosti rješavanja problema kako bi se rezultati u završnim ispitima znanja mogli uspoređivati s utvrđenom razinom sposobnosti rješavanja problema. Isti ispit se koristio u drugoj fazi istraživanja (M. Mladenović et al., 2017; M. Mladenović, Krpan, et al., 2016). Ispit je inicijalno oblikovan za uzrast učenika petih razreda te je namjerno napravljen nešto lakšim kako se ne bi smanjila motivacija i povećala frustracija učenika. Važno je da svi učenici imaju isti ispit zbog uspoređivanja grupa, pa iako u ovom istraživanju sudjeluju učenici do osmog razreda ispit je ostavljen isti zbog najmlađih učenika.

4.3.2.2 Završni ispiti znanja

Gradivo obrađeno tijekom tretmana je, prema nastavnom planu i programu, gradivo petog razreda koje se koristi u svim kasnijim razredima. Stoga su svi učenici upoznati s pojmovima i naredbama korištenim tijekom nastave i na završnom ispitu. Završni ispiti znanja su izrađeni s analognim zadacima u sva tri programska jezika. Zbog toga i većeg broja razreda, škola i učitelja test je napravljen s pet zadataka s usmjeravanjem na razumijevanje koncepta petlje. Zadaci nisu bili jednake težine zbog čega je, osim ukupnih rezultata, napravljena analiza svakog od zadataka. Posebno su bili zanimljivi rezultati učenika petih i šestih razreda od koji su većina početnici u programiranju te na sličnoj razini kognitivnog razvoja. Bilo bi zanimljivo na isti način usporediti skupine sedmih i osmih razreda, ali je među takvim učenicima postojao samo manji broj učenika kontrolne skupine te ista analiza nije provedena među sedmim i osmim razredima.

Ispit se sastojao od pet zadataka. U svakom zadatku trebalo je odabrati odgovor koji odgovara rezultatu izvršavanja zadanog kôda. Netočni odgovori osmišljeni su kao distraktori kako bi se moglo utvrditi postojanje pogrešnih predodžbi za ispitivane pojmove. Zadaci i odgovori završnih ispita znanja u razmatranim programskim jezicima prikazani su u tablici 4.28.

Tablica 4.28 - Zadaci i odgovori završnih ispita znanja u programskim jezicima Scratch, Logo i Python

Zadatak	Scratch	Logo	Python	Odgovori
Z1		fd 100 lt 90 fd 50 fd 50 lt 90	fd (100) lt (90) fd (50) fd (50) lt (90)	a) 100 b) 150 c) 180 d) 200 e) 380
Z2		fd 10 repeat 10[fd 10]	fd(10) for i in range (10): fd(10)	a) 10 b) 20 c) 30 d) 100 e) 110
Z3		repeat 10[fd 10 rt 10 fd 10]	for i in range (10): fd(10) rt(10) fd(10)	a) 20 b) 30 c) 40 d) 200 e) 300
Z4		repeat 10[fd 1 repeat 10[fd 1]]	for i in range (10): fd(1) for i in range (10): fd(1)	a) 10 b) 20 c) 100 d) 110 e) 1000
Z5		fd (1) repeat 10[fd 1] repeat 10[fd 1]	fd(1) for i in range (10): fd(1) for i in range (10): fd(1)	a) 20 b) 21 c) 23 d) 110 e) 103

Ni u ovom istraživanju tijekom nastave učenici nisu vježbali tipove zadataka iz ispita. Učenici kontrolne skupine (Logo i Python grupe) koristili su slične primjere pri obrađivanju gradiva što je posebno važno naglasiti za primjer ugniježđene petlje. Za razliku od njih, učenici iz Scratch grupe nisu vježbali slične primjere već su korištene programske koncepte koristili u kontekstu programiranja igara. Takve razlike mogu se uočiti u dizajnu istraživanja obiju skupina (tablice 4.26 i 4.27). Osim toga učenici obiju grupa nisu imali dodatne zadatke u vidu domaće zadaće ni za rad izvan nastave. Ispit nije ocjenjivan, a učenici su za pisanje ispita motivirani s potencijalno ocjenom odličan (za učenike koji budu među najuspješnijima), prihvaćanjem ponuđene druge ocjene ili plusom za aktivno sudjelovanje u ispitu.

4.3.2.3 Utvrđivanje razlika između grupa

Kako je uzorak >50 za utvrđivanje distribucije rezultata korišten je Kolmogorov-Smirnovljev test. Prema rezultatima Kolmogorov-Smirnovljevog testa rezultati preliminarnog ispita nisu normalno raspodijeljeni za preliminarni ispit za Scratch (n=59, p=0.000), i Logo skupine (n=185, p=0.000) dok za Python jesu (n=68, p=0.200). Kod završnih ispita nema normalne razdiobe za Scratch (n=59, p=0.000), Python (n=69, p=0.000) i Logo skupine (n=185, p=0.000). Zbog toga su se u analizi koristili neparametrijski testovi.

Za utvrđivanje razlika između grupa prema nezavisnim varijablama pripadnost skupini prema programskom jeziku s rezultatima na preliminarnom ispitu rješavanja problema korišten je Kruskal-Wallisov test. Rezultati Kruskal-Wallisovog testa pokazali su da nema statistički značajnih razlika između skupina prema programskim jezicima ($\chi^2(2) = 2.237$, p=0. 327) u preliminarnom ispitu sposobnosti rješavanja problema. Prema ovim rezultatima može se utvrditi da su eksperimentalna i kontrolne skupine ujednačene te se podaci mogu dalje analizirati.

4.3.3 Usporedba rezultata završnih ispita u odnosu na korišteni programski jezik

Za usporedbu rezultata završnih ispita u odnosu na korišteni programski jezik korišten je Kruskal-Wallis test. Rezultati su pokazali statistički značajnu razliku između grupa ($\chi^2(2) = 43.323$, p=0.000). Kako bi se utvrdile eventualne razlike između pojedinih grupa korišten je Mann-Whitney U test. Rezultati su prikazani u tablici 4.29.

Tablica 4.29 - Rezultati Mann-Whitney U testa za utvrđivanje razlike između grupa

		PI	Z1	Z2	Z3	Z4	Z5	Ukupno
Scratch - Logo	U	5201.5	3698	3694	3726	4545	4032	2727.5
	p	0.586	0.00*	0.00*	0.00*	0.02*	0.00*	0.00*
Scratch - Python	U	1695	1225	1309	1345	1138	1409	816
	p	0.131	0.00*	0.00*	0.00*	0.00*	0.00*	0.00*
Logo - Python	U	5698	5869	6137	6213	4620	6061	5402
	p	0.25	0.346	0.732	0. 860	0.000*	0.593	0.08

Kao što je vidljivo u tablici, statistički značajne razlike su se pokazale među rezultatima učenika Scratch grupe u usporedbi s rezultatima Logo i Python grupa gdje su učenici iz Scratch grupe ostvarili bolje rezultate. Kod usporedbe Logo i Python rezultata pokazala se statistički značajna razlika za Z4 – zadatak s ugniježđenom petljom, koji je bio najsloženiji.

U tablici 4.30 su prikazane aritmetičke sredine postignutih bodova grupa za svaki zadatak.

Tablica 4.30 - Aritmetičke sredine za svaki zadatak

Zadatak (max. broj bodova)	Z1 (1)	Z2 (2)	Z3 (3)	Z4 (5)	Z5 (4)	Ukupno (15)
Scratch	0.83	1.70	2.14	2.46	3.67	10.78
Logo	0.51	1.04	1.18	1.62	2.62	6.98
Python	0.44	1.00	1.15	0.29	2.47	5.35
Prosječno	0.564	1.17	1.40	1.29	2.74	4.72

Na temelju dobivenih rezultata možemo prihvatiti H1 i zaključiti kako učenici koristeći Scratch postižu bolje rezultate u odnosu na učenike koji su koristili Logo ili Python.

4.3.4 Usporedba rezultata učenika petih i šestih razreda

Za dodatnu analizu su se uzeli rezultati učenika petih i šestih razreda iz više razloga. Prvi je što je većina učenici tog uzrasta (10-12 godina) na istoj razini kognitivnog razvoja. Drugi je što su ili potpuni početnici ili imaju malo iskustva u programiranju.

Od 312 učenika koji su sudjelovali u istraživanju, u navedenu populaciju spada 207 učenika. Detalji sudionika su prikazani u tablici 4.31.

Tablica 4.31 - Uzorak učenika petih i šestih razreda

Škola	Razred	Programski jezik	Učenici	Dječaci	Djevojčice	Prethodno iskustvo programiranja
OŠ Blatine-Škrape	5	Scratch	30	14	16	Ne
OŠ Blatine-Škrape	6	Python	17	7	10	Da (Python)
OŠ Mejaši	5	Logo	46	21	25	Ne
OŠ Mejaši	6	Logo	52	29	23	Da (Logo)
OŠ Split 3	5	Python	46	18	28	Ne
OŠ Bol	6	Scratch	16	11	5	Ne
		Ukupno:	207	100	107	

Prema rezultatima Kolmogorov-Smirnov testa rezultati ne zadovoljavaju normalnu distribuciju ($n=207$, $p=0.00$) pa se za utvrđivanje između grupa koristio test Kruskal-Wallis. Prema rezultatima nema statistički značajne razlike u postignutim rezultatima na preliminarnim ispitima sposobnosti rješavanja problema između grupa ($\chi^2(2)=2.603$, $p=0.272$) pa se grupe mogu tretirati kao ujednačene i nastaviti s analizom.

Za utvrđivanje razlika između uspjeha na završnim ispitima znanja te po zadacima korišten je Mann-Whitney test. Rezultati su prikazani u tablici 4.32.

Tablica 4.32 - Rezultati Mann-Whitney U testa za učenike petih i šestih razreda za utvrđivanje razlike između grupa

		Z1	Z2	Z3	Z4	Z5	Ukupno
Scratch - Logo	U	1752.0	1631.0	1675.0	1679.0	1619.0	1174.0
	p	0.009*	0.001*	0.004*	0.002*	0.000*	0.000*
Scratch - Python	U	973.5	1002.5	1039.5	816.5	1060.5	643.5
	p	0.01*	0.001*	0.004*	0.000*	0.001*	0.001*
Logo - Python	U	2761.5	2989.0	3006.5	2527.5	3045.0	2739.5
	p	0.192	0.693	0.745	0.003*	0.859	0.221

Kod učenika petih i šestih razreda također postoji statistički značajna razlika između rezultata učenika koji su koristili Scratch i rezultata učenika koji su koristili Logo i Python. Između rezultata učenika koji su koristili Logo i Python i u ovoj skupini postoji statistički značajna razlika u rezultatima zadatka 4 (Z4) s ugniježđenom petljom. Grupa koja je radila u Logu je postigla bolje rezultate za navedeni zadatak. U tablici 4.33 su prikazane aritmetičke sredine rezultata za svaki zadatak kao i za ukupne rezultate preliminarnog ispita i završnih ispita.

Tablica 4.33 - Aritmetičke sredine rezultata po zadacima

Zadatak	n	Z1	Z2	Z3	Z4	Z5	ZI Ukupno
Scratch	46	0.8043	0.8478	0.6957	0.50	0.9348	3.7826
Logo	98	0.5816	0.5714	0.4388	0.2449	0.6531	2.4898
Python	63	0.4762	0.5397	0.4127	0.0635	0.6667	2.1587
Ukupno	207	0.6207	0.6530	0.5157	0.2695	0.7515	2.8104

Kao i u drugoj fazi istraživanja, učenici su podijeljeni u težinske grupe temeljem rezultata preliminarnog ispita. Tablica 4.34 prikazuje distribuciju sudionika prema težinskim grupama *Bolji*, *Srednji* i *Lošiji*.

Tablica 4.34 - Razdioba učenika petih i šestih razreda po težinskim skupinama

	Bolji(B)	Srednji (S)	Lošiji (L)
Scratch	19	16	11
Logo	30	33	35
Python	24	14	25
Ukupno	73	63	71

Hi-kvadrat test korišten je za usporedbu rezultata završnih ispita svih programskih jezika po težinskim grupama. Rezultati su pokazali da postoji statistički značajna razlika između sposobnosti rješavanja problema i rezultata učenika na završnim ispitima u Logu ($\chi^2(10)=23.989$, $p=0.008$), Pythonu ($\chi^2(10)=19.006$, $p=0.04$), ali ne i u Scratchu ($\chi^2(10)=13.99$,

p=0.173). Ovi rezultati su u skladu s prethodnom fazom istraživanja te se i u ovom istraživanju može zaključiti kako učenici iz težinskih skupina *Srednji* i *Lošiji* postižu bolje rezultate u Scratchu nego u Logu i Pythonu.

Prema navedenim rezultatima može se prihvatiti H2 i zaključiti kako učenici na istoj razini kognitivnog razvoja, u ovom slučaju petih i šestih razreda, postižu bolje rezultate u završnim ispitima znanja u Scratchu u usporedbi s rezultatima završnih ispita u Logu i Pythonu.

4.3.4.1 Analiza rezultata svakog zadatka

U ovom poglavlju će se analizirati odgovori za svaki od zadataka kako bi se utvrdile najčešće pogreške učenika u svakom od razmatranih programskih jezika. Za svaki zadatak pitanje je glasilo „Koliko će koraka izvršiti lik/kornjača nakon izvršavanja skripte?“. Zadaci su bili tipa višestrukog odgovora s jednim odgovorom. Bilo je ponuđeno po pet odgovora od kojih su četiri distraktori koji se temelje na pogrešnim shvaćanjima. Hi-kvadrat test koristio se za utvrđivanje razlika između programskih jezika i odgovora učenika.

4.3.4.1.1 Zadatak 1 – slijed

U prvom zadatku se ispitivao slijed, odnosno prepoznavanje naredbi za kretanje u skripti s algoritmom slijeda. Rezultati hi-kvadrat testa su pokazali kako postoji statistički značajna razlika između odgovora i programskog jezika (χ^2 (10) =34.057, p=0.000). Zadatak je prikazan u tablici 4.35.

Tablica 4.35 - Prvi zadatak (Z1)

Scratch	Logo	Python	Odgovori

	`fd 100`
	`lt 90`
	`fd 50`
	`fd 50`
	`lt 90`
		`fd (100)`
		`lt (90)`
		`fd (50)`
		`fd (50)`
		`lt (90)`
			a) 100
			b) 150
			c) 180
			d) 200
			e) 380

Točan odgovor je bio pod d) što je odabralo ukupno 59% učenika od čega čak 80.4% učenika iz Scratch grupe, 58.2% učenika Logo grupe te 47.6% učenika Python grupe. Visok postotak točnih odgovora Scratch grupe može se interpretirati vizualnim naredbama kod kojih naredbe skretanja sadrže simbol skretanja pa je sasvim jasno čemu koja naredba služi. Kod programskih jezika Logo i Python naredbe su tekstualne i skraćenice engleskih riječi što je je zahtjevnije. Distribucija odgovora zadatka 1 se može vidjeti u tablici 4.36.

Tablica 4.36 - Distribucija odgovora za Z1

		Bez odgovora	a	b	c	**d**	e	Ukupno
Scratch	Broj	0	1	1	2	**37**	5	46
	Očekivano	1.3	1.8	1.3	.9	**27.6**	13.1	46.0
	%	0.0%	2.2%	2.2%	4.3%	**80.4%**	10.9%	100.0%
Logo	Broj	0	2	4	0	**57**	35	98
	Očekivano	2.8	3.8	2.8	1.9	**58.7**	27.9	98.0
	%	0.0%	2.0%	4.1%	0.0%	**58.2%**	35.7%	100.0%
Python	Broj	6	5	1	2	**30**	19	63
	Očekivano	1.8	2.4	1.8	1.2	**37.7**	18.0	63.0
	%	9.5%	7.9%	1.6%	3.2%	**47.6%**	30.2%	100.0%
Ukupno	Broj	6	8	6	4	**124**	59	207
	%	2.9%	3.9%	2.9%	1.9%	**59.9%**	28.5%	100.0%

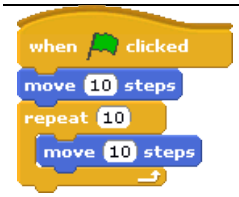
Najčešće zastupljeni netočan odgovor je pod e). Taj odgovor je distraktor koji predstavlja zbroj svih vrijednosti koje se koriste u programu. Ostali odgovori su zastupljeni u malim postocima pa se ne mogu smatrati pogrešnim predodžbama te time ni generalizirati greške.

4.3.4.1.2 Zadatak 2 – jednostavna petlja

Zadatak 2 (Z2) se odnosi na prepoznavanje izvršavanja programa s jednostavnom petljom. Zadatak je prikazan u tablici 4.37.

Tablica 4.37 - Zadatak 2 (Z2)

Scratch	**Logo**	**Python**	**Odgovori**

	`fd 10 repeat 10[fd 10]`	`fd(10)`
		`for i in range (10):`
		`  fd(10)`
			a)10
			b)20
			c)30
			d)100
			e)110

Rezultati hi-kvadrat testa su pokazali da postoji statistički značajna razlika između odgovora i programskog jezika (χ^2 (8) =16.210, p=0.039).

U tablici 4.38 su prikazane distribucije odgovora za zadatak 2.

Tablica 4.38 - Distribucija odgovora za Z2

		a	b	c	d	**e**	Ukupno
Scratch	Broj	1	1	3	2	**39**	46
	Očekivano	2.4	2.0	5.3	7.6	**28.7**	46.0
	%	2.2%	2.2%	6.5%	4.3%	**84.8%**	100.0%
Logo	Broj	5	3	13	21	**56**	98
	Očekivano	5.2	4.3	11.4	16.1	**61.1**	98.0
	%	5.1%	3.1%	13.3%	21.4%	**57.1%**	100.0%
Python	Broj	5	5	8	11	**34**	63
	Očekivano	3.3	2.7	7.3	10.3	**39.3**	63.0
	%	7.9%	7.9%	12.7%	17.5%	**54.0%**	100.0%
Ukupno	Broj	11	9	24	34	**129**	207
	%	5.3%	4.3%	11.6%	16.4%	**62.3%**	100.0%

Točan odgovor je bio e) što je odgovorilo 84.8% učenika Scratch grupe, 57.1% učenika Logo grupe i 54% učenika Python grupe. U Scratchu se naredbe razlikuju u boji i obliku. Naredbe za kretanje su obojane plavom bojom, a za upravljanje žutom bojom pa je vizualno sasvim jasno koja naredba čemu služi što se može vidjeti u tablici 4.38. Stoga ne čudi visok postotak točnih odgovora iz Scratch grupe. Najviše netočnih odgovora je bilo za d) odgovor kojeg je odabralo 21.4% učenika iz Logo grupe, 17.5% iz Python grupe i samo 4.3% iz Scratch grupe. Učenici koji su odabrali ovaj odgovor ignorirali su prvu naredbu, odnosno registrirali su samo rezultat izvršavanja petlje. Kako je pojam petlje složen, učenici koji su programirali u tekstualnom programskom jeziku usmjerili su se samo na petlju i pritom ignorirali prvu naredbu što ukazuje na potrebu uvježbavanja na više različitih primjera. Takav problem je minimalan u Scratchu zbog intuitivnosti korištenja i značenja blokovskih naredbi.

Ostali netočni odgovori su zastupljeni u manjim postocima pa ne upućuju na pogrešna poimanja učenika već na pojedinačne slučajeve.

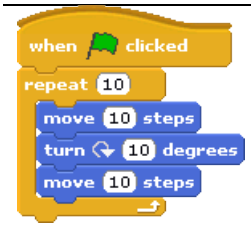
4.3.4.1.3 Zadatak 3 – jednostavna petlja

Zadatak 3 je nešto složenija verzija prethodnog zadatka gdje se ponavljaju tri naredbe od kojih su dvije za kretanje.

Zadatak je prikazan u tablici 4.39

Tablica 4.39 - Zadatak 3 (Z3)

Scratch	Logo	Python	Odgovori

	`repeat 10[fd 10 rt 10 fd 10]`	`for i in range (10):`
		`  fd(10)`
		`  rt(10)`
		`  fd(10)`
			a)20
			b)30
			c)40
			d)200
			e) 300

Rezultati hi-kvadrat testa su pokazali da postoji statistički značajna razlika između odgovora i korištenog programskog jezika ($\chi^2(10) = 20.883$, $p=0.022$).

U tablici 4.40 su prikazane distribucije odgovora za zadatak 3.

Tablica 4.40 - Distribucija odgovora za Z3

		Bez odgovora	a	b	c	**d**	e	Ukupno
Scratch	Broj	0	0	7	3	**32**	4	46
	Očekivano	.2	.9	6.0	4.4	**22.4**	12.0	46.0
	%	0.0%	0.0%	15.2%	6.5%	**69.6%**	8.7%	100.0%
Logo	Broj	0	1	10	11	**43**	33	98
	Očekivano	.5	1.9	12.8	9.5	**47.8**	25.6	98.0
	%	0.0%	1.0%	10.2%	11.2%	**43.9%**	33.7%	100.0%
Python	Broj	1	3	10	6	**26**	17	63
	Očekivano	.3	1.2	8.2	6.1	**30.7**	16.4	63.0
	%	1.6%	4.8%	15.9%	9.5%	**41.3%**	27.0%	100.0%
Ukupno	Broj	1	4	27	20	**101**	54	207
	%	.5%	1.9%	13.0%	9.7%	**48.8%**	26.1%	100.0%

Na točan odgovor d) odgovorilo je 69.6% učenika iz Scratch grupe, 43.9% učenika iz Logo grupe i 41.3% iz Python grupe. Iako je u ovom zadatku udio točnih odgovora nešto manji zbog složenosti zadatka, i dalje je očita razlika između udjela točnih odgovora u Scratchu u usporedbi s Logom i Pythonom.

Najčešći netočan odgovor je pod e). Ovaj odgovor, odnosno distraktor, je služio za otkrivanje nedovoljnog shvaćanja petlje, odnosno, kako je petlja složen programski koncept tako se učenici fokusiraju na samu petlju te pritom ignoriraju sadržaj petlje. U ovom slučaju se unutar nalaze tri naredbe od kojih su dvije za kretanje, a jedna za skretanje. Učenici koji su odabrali ovaj odgovor su samo zbrojili sve brojeve koji se nalaze unutar petlje te ih pomnožili s brojem

ponavljanja. Ovakva greška upućuje na potrebu dodatnog uvježbavanja na više različitih primjera primjene petlje. Odgovor je odabralo 33.7% učenika Logo grupe i 26.1% učenika Python grupe dok je samo 8.7% učenika iz Scratch grupe odabralo ovaj netočan odgovor. Opet je uočljiva velika razlika između točnih i netočnih odgovora u blokovskom i tekstualnim programskim jezicima. Ne čudi što je takav problem u Scratchu manje zastupljen, zbog već spomenutog vizualnog prikaza naredbi. Učestalost ove greške u tekstualnim programskim jezicima opet upućuje na potrebu dodatnog uvježbavanja kako bi učenici mogli jasno pratiti slijed i značenje naredbi. Ostali netočni odgovori su rasuti i zastupljeni s manje od 20% što ne upućuje na sustavna pogrešna shvaćanja već na slučajne pogreške.

4.3.4.1.4 Zadatak 4 – ugniježđena petlja

Zadatak 4 je najsloženiji zadatak koji sadrži ugniježđenu petlju. U tablici 4.41 je prikazan zadatak 4.

Tablica 4.41 - Zadatak 4 (Z4)

Scratch	Logo	Python	Odgovori

	`repeat 10[fd 1 repeat 10[fd 1]]`	`for i in range (10):`
		`  fd(1)`
		`  for i in range (10):`
		`  fd(1)`
			a)10
			b)20
			c)100
			d)110
			e) 1000

Rezultati hi-kvadrat testa su pokazali da postoji statistički značajna razlika između odgovora i korištenog programskog jezika ($\chi^2(10)=44.664$, $p=0.000$). U tablici 4.42 su prikazane distribucije odgovora za zadatak 4.

Tablica 4.42 - Distribucija odgovora za Z4

		Bez odgovora	a	b	c	**d**	e	Ukupno
Scratch	Broj	2	0	10	9	**23**	2	46
	Očekivano	1.3	2.4	20.0	9.6	**11.3**	1.3	46.0
	%	4.3%	0.0%	21.7%	19.6%	**50.0%**	4.3%	100.0%
Logo	Broj	0	7	39	25	**24**	3	98
	Očekivano	2.8	5.2	42.6	20.4	**24.1**	2.8	98.0
	%	0.0%	7.1%	39.8%	25.5%	**24.5%**	3.1%	100.0%
Python	Broj	4	4	41	9	**4**	1	63
	Očekivano	1.8	3.3	27.4	13.1	**15.5**	1.8	63.0

	%	6.3%	6.3%	65.1%	14.3%	**6.3%**	1.6%	100.0%
Ukupno	Broj	6	11	90	43	**51**	6	207
	%	2.9%	5.3%	43.5%	20.8%	**24.6%**	2.9%	100.0%

Odgovori na ovaj zadatak su najraspršeniji od svih zadataka. Točan odgovor d) je odabralo 50% učenika iz Scratch grupe, 24.5% učenika Logo grupe i samo 6.3% učenika iz Python grupe. U ovom slučaju očita je velika razlika između sve tri grupe. Učenici koji su radili u Scratchu nisu imali ni jedan primjer ugniježđene petlje tijekom nastave dok učenici Logo i Python grupe jesu i to za crtanje složenijih likova. Unatoč tome je udio točnih odgovora u Scratchu najzastupljeniji što se opet može objasniti vizualnim prikazom naredbi gdje se, bez uvježbavanja, jednostavno može zaključiti na rezultat izvršavanja složenijih skripti. U tablici 4.41 je vidljiva razlika između sintaksi svih triju jezika. Ono što se dodatno ističe u ovom zadatku je niski udio točnih odgovora u Python grupi. U ostalim zadacima nije bilo većih razlika u točnim odgovorima između Logo i Python grupe. S obzirom da je najveći broj učenika Logo i Python grupe odabrao odgovor b) može se zaključiti kako učenici u ovim skupinama nisu u dovoljnoj mjeri savladali koncept petlje. Netočan odgovor b) je distraktor koji su odabrali učenici koji su ugniježđenu petlju interpretirali kao dvije odvojene petlje što ukazuje na nepotpuno shvaćanje koncepta petlje. Ovaj odgovor je odabralo 21.7% učenika Scratch grupe, 39.8% učenika Logo grupe i čak 65.1% Python grupe. Posebno je zanimljiv velik udio netočnih odgovora Python grupe, pogotovo u odnosu na Logo grupu, koji ukazuje na sustavno pogrešno shvaćanje koncepta petlje u složenijem obliku što je potrebno dodatno analizirati. Veliki udio odabira ovog distraktora u Python grupi se može pripisati sintaksi jezika. Naime, kod Loga se naredbe koje se nalaze unutar svake petlje nalaze unutar uglatih zagrada (`repeat 10[fd 1 repeat 10[fd 1]]`) što naredebu čini jasnijom. U Pythonu se umjesto zagrada za strukture petlji i grananja koriste uvlake što kod čini čitkijim, ali početnicima nejasnijim u složenijim strukturama:

```
for i in range (10):
    fd(1)
    for i in range (10):
          fd(1)
```

Učenici koji su odabrali odgovor d) drugu petlju nisu interpretirali kao petlju unutar petlje već kao novu petlju koja se izvršava nakon što se u potpunosti izvrši prva petlja pa su do odgovora došli na sljedeći način: 10*1+10*1=20. Kod Pythona učenici su dvostruku uvlaku ugniježđene petlje ignorirali.

Drugi najzastupljeniji netočan odgovor je pod c) što je odabralo 19.6% učenika Scratch grupe, 25.5% učenika Logo grupe i 14.3% učenika Python grupe. Zastupljenost netočnih odgovora od

gotovo 20% kod Scratch grupe može upućivati na pogrešno shvaćanje ispitivanog koncepta i na shvaćanje koncepta petlje na niskoj razini. Taj odgovor su odabrali učenici koji su prvu petlju uzeli u obzir kao petlju i pomnožili s brojem ponavljanja druge petlje (10*10).

Ostali odgovori su raspršeni i zastupljeni s manjim udjelom pa se smatra da su rezultat slučajnih odabira, a ne pogrešnih shvaćanja.

4.3.4.1.5 Zadatak 5 – dvije petlje

Zadatak 5 je sadržavao dvije petlje. Zadatak je zapravo sličan prethodnom, ali druga petlja se ovaj put izvršava iza prve. U tablici 4.43 je prikazan zadatak 5.

Tablica 4.43 - Zadatak 5 (Z5)

Scratch	Logo	Python	Odgovori

	`fd (1) repeat 10[fd 1] repeat 10[fd 1]`	`fd(1)`
		`for i in range (10):`
		`  fd(1)`
		`for i in range (10):`
		`  fd(1)`
			a)20
			b)21
			c)23
			d)110
			e) 103

Rezultati hi-kvadrat testa su pokazali da i u ovom zadatku postoji statistički značajna razlika između odgovora i korištenog programskog jezika ($\chi^2(10) = 34.021$, $p=0.000$). U tablici 4.44 su prikazane distribucije odgovora za zadatak 4.

Tablica 4.44 – Distribucija odgovora za Z5

		Bez odgovora	a	**b**	c	d	e	Ukupno
Scratch	Broj	2	0	**43**	0	1	0	46
	Očekivano	1.3	1.8	**33.1**	3.6	4.0	2.2	46.0
	%	4.3%	0.0%	**93.5%**	0.0%	2.2%	0.0%	100.0%
Logo	Broj	0	3	**64**	8	15	8	98
	Očekivano	2.8	3.8	**70.5**	7.6	8.5	4.7	98.0
	%	0.0%	3.1%	**65.3%**	8.2%	15.3%	8.2%	100.0%
Python	Broj	4	5	**42**	8	2	2	63
	Očekivano	1.8	2.4	**45.3**	4.9	5.5	3.0	63.0
	%	6.3%	7.9%	**66.7%**	12.7%	3.2%	3.2%	100.0%
Ukupno	Broj	6	8	**149**	16	18	10	207
	%	2.9%	3.9%	**72.0%**	7.7%	8.7%	4.8%	100.0%

Točan odgovor b) je odabralo čak 93.5% učenika iz Scratch grupe, te 65.3% učenika Logo i 66.7% učenika Python grupe. Rezultati odgovora na ovo pitanje potvrđuje da učenici jesu savladali i razumjeli osnovnu primjenu koncepta petlje. Što se tiče pogrešnih odgovora može se istaknuti odgovor d) koji je odabralo 15.3% učenika Logo grupe, dok je samo 3.2% Python grupe i 2.2% Scratch grupe odabralo isti odgovor. Ovakav udio odgovora u Logo grupi, koji se značajno razlikuje po zastupljenosti u odnosu na Logo i Python grupe, zahtijeva analizu. Naime, učenici su odabrali odgovor koji je bio točan u prethodnom zadatku s ugniježđenom petljom. Ova se pojava opet može pripisati sintaksi jezika jer je ovaj put razlika u položaju jedne zagrade u odnosu na prethodni zadatak.

Ostali odgovori su raspršeni u manjim postocima pa se smatra kako su rezultat slučajnih odabira, a ne pogrešnih shvaćanja.

4.3.4.1.6 Ograničenja istraživanja, rasprava i zaključak

U ovom istraživanju se eliminiralo najveće ograničenje prethodnih dvaju istraživanja koje se odnosi na veličinu uzorka. U ovom istraživanju sudjelovao je veći broj učenika, škola te nastavnika, ali je zato nastalo novo ograničenje. Kao posljedica obuhvaćanja većeg broja učenika različitih uzrasta, razreda, škola i učitelja, istraživanje se provelo u kraćem vremenskom period u odnosu na prethodna dva što je najveće ograničenje ovog istraživanja. Zbog usklađivanja svih sudionika istraživanja tretman je trajalo tri tjedna, odnosno šest školskih sati. Svi učitelji su nastavu provodili u svojim razredima bez prisustva trećih osoba čime se učenicima ne mijenja prirodni okoliš, ali zato nije bilo mogućnosti promatranja samog izvođenja nastave kao i reakcija učenika osim u intervjuima s učiteljima.

Iako se u svim grupama ista količina vremena potrošila na nastavu, te zadaci iz ispita nisu ciljano uvježbavani tijekom nastave, prema rezultatima istraživanja je očito da se u vizualnom blokovskom programskom jeziku lakše i brže uvode novi pojmovi programiranja. Jedan od razloga je problem sintakse koji kod Scratcha praktički ne postoji. Grupe naredbi su kodirane u bojama, te dodatno i oblikom pa je svaka naredba intuitivno jasna, čak i bez dodatnog pojašnjavanja. Kod tekstualnih programskih jezika učenici se osim s rješavanjem problema, što bi trebao biti osnovni problem, prije svega suočavaju s problemom sintakse čime se programiranje dodatno komplicira, pogotovo za obuhvaćeni uzrast. Kao posljedica problema sintakse učenicima je potrebno više uvježbavanja sličnih primjera i zadržavanja na osnovnim slučajevima kako bi mogli u razumjeti uvedene pojmove. Brzim prijelazom na nove, složenije

koncepte, a dok se nisu usvojili prethodni, može biti kontraproduktivno u smislu gubitka motivacije čak i kod učenika koji imaju više razvijeno apstraktno mišljenje od svojih vršnjaka.

Rezultati ove faze istraživanja su u skladu s prethodne dvije faze, a to je da učenici lakše savladavaju osnovne algoritme koristeći vizualni-blokovski programski jezik i bez dodatnog i ciljanog uvježbavanja tipiziranih zadataka. Najveća razlika prisutna je kod najsloženijeg algoritma ponavljanja, odnosno koncepta petlje. Osim toga učenici kroz kontekst programiranja igara s lakoćom vide rezultate izvršavanja programa pa intuitivno mogu riješiti nastale nedostatke u svojim programima. Za razliku od toga, učenici koji su programirali u tekstualnim programskim jezicima imaju problem sa sintaksom na koju troše svoju pažnju pa je manji naglasak na rješavanje problema. Iako su, zbog konteksta programiranja rješavanja matematičkih problema, zadaci koji su se radili u Logo i Python skupinama slični zadacima na završnim ispitima, učenici su ostvarili lošiji uspjeh u odnosu na učenike iz Scratch skupine. Osim toga i u ovom istraživanju se potvrdilo da sposobnost rješavanja problema nije ključan faktor za uspjeh u programiranju u Scratchu dok za uspjeh u programiranju u Logu i Pythonu jest.

4.4 Utjecaj korištenja blokovskog programskog jezika na posredovani prijenos programskih koncepata u kontekst tekstualnog programskog jezika prema proceduralnom pristupu

Četvrta faza istraživanja je nastavak prethodnih faza istraživanja (M. Mladenović, Boljat, & Žanko, 2018; M. Mladenović et al., 2017; M. Mladenović, Krpan, et al., 2016; M. Mladenović, Rosić, et al., 2016). Pokazalo se kako učenici u većoj mjeri usvajaju programske koncepte u blokovskom programskom jeziku. U prethodna tri istraživanja koristio se Scratch kao blokovski programski jezik prema A pristupu u osnovnim školama. Blokovski programski jezik se pokazao učenicima jednostavniji i zanimljiviji što je rezultiralo boljim rezultatima na završnim ispitima znanja. Osim toga pokazali su pozitivniji stav prema programiranju. Kako su djeca u petim i šestim razredima osnovne škole, prema Piagetu, u fazi konkretnih ili predformalnih operacija, pristup od konkretnog prema apstraktnom je poželjan kako bi bolje razumjeli kompleksne, pozadinske koncepte programiranja. Zbog toga je u toj fazi primjereno korištenje blokovskog programskog jezika. Ipak, cilj je da učenici u nekom trenutku prijeđu na „pravi" tekstualni proceduralni jezik što se obrađuje prema pristupu B u osnovnoj školi. U tablici 4.45 su prikazane zadane nastavne teme za 5. i 6. razred prema B pristupu.

Tablica 4.45 - Teme programiranja prema (proceduralnom) B pristupu za 5. i 6. razred

	Teme B (proceduralni jezik)
5. razred	1. Pojam algoritma 2. Dijagram tijeka 3. Naredbe za ulaz i izlaz podataka
6. razred	1. Uporaba naredbi za grananje i bezuvjetni skok 2. Algoritmi s uporabom petlje 3. Uporaba naredbi za petlju bez logičkog uvjeta

Prijelaz iz konteksta blokovskog programskog jezika u kontekst proceduralnog programskog jezika treba provesti uz pravi pristup kako bi učenici mogli „prenijeti" naučene koncept u novi kontekst. Osim toga, pokazalo se kako se širenjem okvira: može potaknuti očekivanja učenika koja učenici kasnije mogu prenositi u nova okruženja; može napraviti poveznicu između dva konteksta; može potaknuti učenike na generalizaciju (Engle, Lam, Meyer, & Nix, 2012). U prijenosu koncepata iz jednog konteksta u drugi pokazala se uspješnim metoda analogije, odnosno korištenje analognih primjera u više situacija učenja (Gentner, Loewenstein, & Thompson, 2003). Korištenje analogija uklapa se u metodu „posredovanog transfera" (eng. *Mediated transfer*) kod koje je cilj učenicima pomoći prenijeti apstraktne koncepte naučene u

blokovskom programskom jeziku, koji im je jednostavniji, u kontekst proceduralnog programskog jezika koji je učenicima apstraktniji i zahtjevniji. Dvije najraširenije tehnike posredovanog prijenosa su „prigrljivanje“ (eng. *hugging*) i „premošćivanje“ (eng. *bridging*) (Dann, Cosgrove, Slater, & Culyba, 2012a; Tabet, Gedawy, Alshikhabobakr, & Razak, 2016). Prema prvom pristupu („prigrljivanje“), u kojoj učitelj stvara situaciju učenja sličnu situaciji u kojoj se transfer očekuje. Prema drugom pristupu („premošćivanje“) učitelj pomaže učenicima premostiti put od konteksta u kojem je koncept naučen do novog potencijalnog konteksta. U ovom pristupu će se koristiti metoda analogije (Gentner et al., 2003) uz korištenje i povezivanje sličnih ili istih primjera u dva programska jezika Python i Micro:bit. Cilj ovog istraživanja je istražiti učinke posredovanog prijenosa između dva navedena pristupa. U tablici 4.46 prikazana je usporedba dvaju različita pristupa posredovanog transfera.

Tablica 4.46 - Metode poučavanja prema pristupima posredovanog transfera

	„Premošćivanje“ (Eksperimentalna skupina)	„Prigrljivanje“ (Kontrolna skupina)
Kontekst učenja	Učitelj direktno povezuje uvođene pojmove u oba okruženja. Prilikom obrađivanja pojmova u novom okruženju učitelj ista rješenja zadatka prikazuje u oba programska okruženja: tekstualnom i blokovskom.	Učitelj ne povezuje direktno koncepte naučene u jednom okruženju već u drugom okruženju obrađuje pojmove u novom kontekstu, u ovom slučaju blokovskom pri čemu na početku nastavne jedinice samo ponovi iste pojmove iz prethodno učenog okruženja bez direktnog povezivanja s novim.
Zadaci	Učitelj koristi iste ili slične zadatke u novom okruženju pri čemu učenici rješavaju zadatke u novom okruženju a učitelj ih direktno uspoređuje s prethodnim okruženjem.	Učitelj koristi iste ili slične zadatke u novom okruženju pri čemu učenici zadatke rješavaju samo u novom okruženju.
Generalizacija	Učitelj pomaže učenicima u povezivanju koncepata u oba okruženja direktnom usporedbom.	Učitelj potiče učenike da sami povezuju koncepte iz oba okruženja na razini ideje umjesto direktne usporedbe.
Analogije	Učitelj pokazuje analogna rješenja iz oba okruženja.	Učitelj potiče povezivanje koncepata pri čemu ne koristi direktne analogije između dva okruženja.

Kako Scratch nije primjeren za proceduralne teme, u ovom istraživanju se koristio Microsoft Block Editor za BBC micro:bit (u daljnjem tekstu će se pisati samo Micro:bit) kao vizualni blokovski jezik i Python kao tekstualni programski jezik. Kako je već navedeno, ovaj put su oba jezika obrađena prema pristupu B koji podrazumijeva proceduralno programiranje.

Iz navedenog su proizašle sljedeće hipoteze:

H1 – Korištenjem posredovanog transfera učenici će postići bolje rezultate u tekstualnom proceduralnom jeziku nakon korištenja blokovskog programskog jezika.

H2 – Metodom „premošćivanja“ učenici će postići bolje rezultate u tekstualnom proceduralnom jeziku nakon korištenja blokovskog programskog jezika.

H3 – Korištenjem konteksta programiranja igara u blokovskom programskom jeziku učenici će imati bolji stav prema programiranju u usporedbi s kontekstom „tradicionalnih“ matematičkih zadataka prilikom obrade nastavne cjeline programiranja.

4.4.1 Opis istraživanja

Istraživanje se provelo rijekom školske godine 2017./2018. u Osnovnoj školi Mejaši u tri šesta razreda (učenici uzrasta 12 godina) tijekom izvođenja izborne nastave predmeta informatika u školskom okruženju. Nastavu je održavala učiteljica na radnom mjestu s višegodišnjim iskustvom u nastavi informatike koja je ujedno i polaznica doktorskog studija „Istraživanje u edukaciji u području prirodnih i tehničkih znanosti“ te je upoznata s metodologijom istraživanja. Istraživanje se usmjerilo na jednu školu i jednu učiteljicu zbog kontrole tijeka same nastave i ispita provedenih u razredima. Svi učenici su u petom razredu također pohađali predmet informatiku. Nastavna cjelina programiranja je u petom razredu obrađena prema proceduralnom, B pristupu. U ovom istraživanju, kao i u prethodnim fazama, uzorak populacije istraživanja je neslučajan i ciljan jer je istraživala ciljana skupina populacije (Cohen et al., 2013). U tablici 4.47 su prikazani podaci o sudionicima koji su u cijelosti bili prisutni tijekom cijelog tretmana.

Tablica 4.47 - Podaci o sudionicima

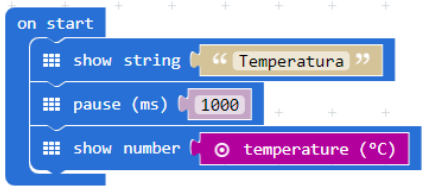
Razred	Skupina	dječaci	djevojčice	Ukupno
6.a	Kontrolna (K)	6	9	15
6.b	Eksperimentalna (E)	7	10	17
6.d	Eksperimentalna (E)	7	8	15
	Ukupno:	20	27	47

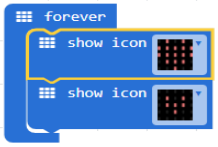
Na kraju istraživanja, ukupan broj učenika je bio 47, iako ih je nešto više sudjelovalo u nekim fazama eksperimenta, ali u obradu su uzeti samo rezultati učenika koji su sudjelovali tijekom cijelog tretmana.

U ovom kvazieksperimentu su od tri razreda dva tretirana kao eksperimentalna, a jedan kao kontrolna skupina. U obje skupine obrađena je nastavna cjelina programiranja prema B pristupu. Nakon svih obrađenih tema učenici su, kroz proces ususstavljivanja gradiva, radili u blokovskom programskom jeziku Micro:bit, ali na različite načine. Učenici su učili programirati u Micro:bitu koristeći zadatke tipične za proceduralno programiranje, ali i programiranjem igre. U kontrolnoj skupini učenicima se nije eksplicitno povezivalo zadatke i situacije s Pythonom dok u eksperimentalnoj jest. U tablici 4.48 je prikazan dizajn istraživanja.

Tablica 4.48 - Dizajn istraživanja

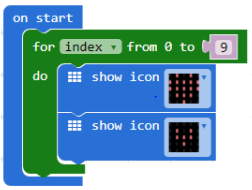
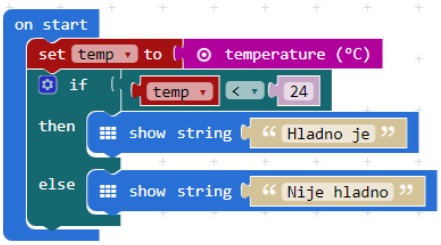
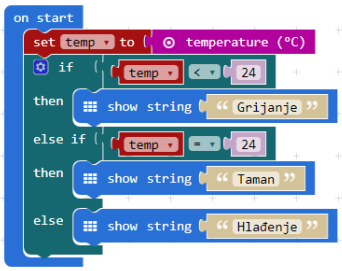
<table>
<tr><td colspan="5">Kontrolna skupina</td></tr>
<tr><td colspan="5">5. razred – obrađena nastavna cjelina programiranja prema B pristup</td></tr>
<tr><td colspan="5">6. razred – obrađena nastavna cjelina programiranja prema B pristup</td></tr>
<tr><td colspan="5">Preliminarni ispit znanja Python (PI-Python)</td></tr>
<tr><td>tjeda n</td><td>Tema</td><td>Novi pojmovi i naredbe</td><td>Opis</td><td>Primjer zadataka</td></tr>
<tr><td>1</td><td>Algoritam slijeda</td><td>On start, show number, show stringShow icon, clear screen, pause, očitavanje senzora za temperaturu i svjetlost

Primjer kôda:
 
</td><td>Upoznavanje s Micro:bitom. Uvođenje algoritma slijeda sa zadacima u blokovskom programskom jeziku.</td><td>Izradi program koji će nakon što se uključi Micro:bit na zaslonu ispisati tekst „Temperatura“ . Nakon jedne sekunde neka prikazuje trenutnu temperaturu prostorije.</td></tr>
<tr><td>2</td><td>Algoritam ponavljanja</td><td>forever, for, varijable, slučajni brojevi

Primjer kôda s beskonačnom petljom:
 

Primjer kôda petlje bez logičkog uvjeta s ograničenim brojem izvršavanja:</td><td>Temeljem algoritma slijeda uveden je algoritam ponavljanja prvo u obliku beskonačne petlje pa nakon toga i petlje bez logičkog uvjeta s ograničenim brojem izvršavanja. Ponovljen je i pojam varijable kroz potrebu „pamćenja“ slučajno odabranog broja.</td><td>a) Limeni iz zemlje Oz je tužan jer nema srce. Napravite program koji će napraviti Limenom srce koje kuca i to točno jedan otkucaj po sekundi.

b) Limeni iz zemlje Oz je tužan jer nema srce. Kako Limeni iz zemlje Oz nije naučio na srce, napravi program da mu točno 10 puta zakuca srce.</td></tr>
</table>

				
3	Algoritam grananja	If-then-else, relacijski operatori *Primjer kôda:* 	Uvedeno je dvostruko grananje u Micro:bitu na način da se koriste senzori za očitavanje temperature i svjetlosti i ispisuje odgovarajuća poruka ovisno o očitanim vrijednostima.	Sobnom temperaturom se smatra 24°C jer nam je tada najugodnije. Napravi program koji će učitati sobnu temperaturu i zapamtiti je u varijabli *temp* pa ispisati: **-Ako je** temperatura ispod 24°C ispisati „Hladno je" -Inače neka ispiše „**Nije hladno**"
4	Višestruko grananje	If-else if-else, on shake događaj, aritmetičke operacije *Primjer kôda:* 	Nastavljanjem na dvostruko grananje iz prethodnog sata uvedena je potreba višestrukog grananja. Osim za slične primjere iz prethodnog sata za očitavanje temperature i razine svjetlosti, učenici su programirali bacanje kockice na Micro:bitu za što im je potreban slučajan broj.	1. Ponekad nam je potrebno ispitati više od jednog uvjeta (kao sada) I kod višestrukog grananja uvijek će se izvršiti najviše jedan uvjet. U svim programskim jezicima postoje naredbe kojima možemo ispitivati više uvjeta. Ako je temperatura ispod 24°C ispisati „Hladno je" Inače ako je temperatura točno 24°C onda ispisati „Taman" Inače neka ispiše „Nije hladno" 2. a) Napravi program koji će pri pokretanju programa ispisati znak kockice prema slučajno izabranom broju. b) Promijeni program tako da se pri svakoj trešnji Micro:bita ispisuje slučajno odabrana vrijednost na „kockici"
5	Ponavljanje	Koristeći naučene pojmove i naredbe u Micro:bitu, učenici su trebali osmisliti algoritam za igru kamen-škare-papir te ga programirati na Micro:bitu		
6	Anketa zadovoljstva programiranjem i programskim jezicima Završni ispit znanja u Micro:bitu (ZI-M)			
7	Završni ispit znanja u Pythonu (ZI-P2)			

4.4.2 Instrumenti

Instrumenti koji su se koristili u ovom istraživanju temelje se na instrumentima korištenim u prethodna tri istraživanja.

Korišteni instrumenti (navedeni redoslijedom korištenja):

- Preliminarni ispit (PI) sposobnosti rješavanja problema;
- Preliminarni ispit znanja u Pythonu (PI-P);
- Anketa zadovoljstva programiranjem, Pythonom, Micro:bitom i programiranjem igara;
- Završni ispit znanja u Micro:bitu (ZI-M);
- Završni ispit znanja u Pythonu (ZI-P);

4.4.2.1 Preliminarni ispit (PI) sposobnosti rješavanja problema

Kao i u prethodnim istraživanjima napravljen je preliminarni ispit rješavanja problema koji su učenici rješavali u petom razredu prije obrađivanja nastavne cjeline programiranje. Preliminarni ispit je korišten na većem uzorku učenika petih razreda jer se koristio kao dio većeg istraživanja.

Kako bi se povećala pouzdanost ispita povećan je broj zadataka. Zbog toga se, u odnosu na prošlo istraživanje, broj zadataka u preliminarnom ispitu znanja povećao s 8 na 12 zadataka. Učenici su i u ovom slučaju ispit pisali dobrovoljno, a bili su motivirani plusom za aktivno sudjelovanje u ispitu.

U ovom preliminarnom ispitu sudjelovalo je ukupno 95 učenika petih razreda (uzrast 11 godina) iz šest razreda u tri osnovne škole u Splitu.

Za pouzdanost ispita korišten je Cronbach α koeficijent prema kojem ovaj preliminarni ispit spada u kategoriju visoko pouzdanih (Crombach α =0.74). Prema rezultatima Kolmogorov-Smirnov testa (p=0.000) podaci ne zadovoljavaju normalnu distribuciju podataka. Zbog toga se za utvrđivanje razlika između grupa koristio neparametrijski Kruskal-Wallis test. Nije utvrđena razlika u rezultatima između škola ($\chi^2(1)=1.631$, p=0.201) ni između razreda ($\chi^2(5)=1.778$, p=0.879). Ovaj ispit je služio za provjeru ujednačenosti između skupina prema sposobnosti rješavanja problema. S obzirom na rezultate može se zaključiti da su skupine ujednačene prema sposobnosti rješavanja problema. U tablici 4.49 su prikazani zadaci preliminarnog ispita posobnosti rješavanja problema.

Tablica 4.49 - Preliminarni ispit sposobnosti rješavanja problema

	Zadatak	bod
Z1	Nastavi niz: 3 4 6 7 9 10 12 13 15 16 ___ ___ ___	*1*
Z2	Nastavi niz: 1 ž 3 z 9 v 27 u 81 ___ ___ ___	*2*
Z3	Nastavi niz s dvije kombinacije slova: JKLMNO JKLMON JKLOMN JKOLMN _____ _____	*2*
Z4	Zbroj Franovih i Filipovih godina iznosi 12. Koliki će biti zbroj njihovih godina za 4 godine?	*1*
Z5	Ana dijeli nekoliko jabuka sebi i petoro svojih prijatelja. Svatko je dobio po polovinu jabuke. Koliko je jabuka Ana podijelila?	*1*
Z6	Ako riječ REČENICA sadrži manje od 9 slova i više od 3 sloga, napiši prvi slog. U suprotnom napiši suglasnik (zatvornik) koji se nalaze najdesnije u riječi.	*1*
Z7	Zamisli da je petak 3 dana prije jučerašnjeg dana. Koji dan će biti sutra?	*1*
Z8	Zamisli da je danas subota. Koji bi dan bio 4 dana nakon prekjučer?	*1*
Z9	Kada Pinokio govori laž, njegov nos se produlji za 6 cm, a kada govori istinu, skrati se za 2 cm. Nakon što njegov nos već bio dug 9 cm, Pinokio je još izgovorio tri lažne i dvije istinite rečenice. Kolika je nakon toga bila duljina njegovog nosa?	*1*
Z10	Ivan je sporiji od Ante, ali brži od Ane. Ana je brža od Petre. Napiši njihova imena od najsporijeg do najbržeg.	*1*
Z11	Dražen, Ana i Ivana imaju zanimanja učitelj, prodavač i kuhar. Dražen je niži od Ane, ali viši od Ivane. Prodavač je najviši, a kuhar najniži. a) Kojeg su zanimanja Dražen, Ana i Ivana? b) Tko je najviši, a tko najniži?	*2*
Z12	Luka ima 12 puta više bombona od Katarine. Ivan ih ima duplo manje od Jasne. Jasna ih ima duplo manje od Luke. Katarina ima 6 bombona. Koliko bombona ima Luka, a koliko Ivan?	*2*

4.4.2.2 Preliminarni ispit (PI-P) i završni ispiti znanja u Micro:bitu (ZI-M) i Pythonu (ZI-P)

Preliminarni ispit u Python (PI-P) i završni ispiti znanja u Micro:bit (ZI-M) i Python (ZI-P) programskim jezicima su napravljeni s analognim zadacima, ali s različitim vrijednostima korištenim u primjerima programa kako bi se mogli uspoređivati rezultati. Svi zadaci su bili tipovi zadataka s jednostrukim odgovorima. Ukupno je bilo 19 zadataka.

- ✓ **Preliminarni ispit znanja u Pythonu** (PI-P) je proveden neposredno nakon obrade svih nastavnih tema programiranja u Pythonu. U šestom razredu se, prema B pristupu, obrađuju sva tri osnovna algoritma: slijed, grananje, ponavljanje te pripadajuće programski konstrukti pa su ispitom obuhvaćeni svi navedeni koncepti.

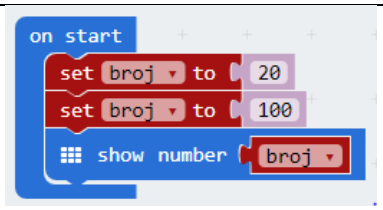
- ✓ **Završni ispit znanja u Micro:bitu** (ZI-M) je proveden neposredno nakon obrađene cjeline korištenjem Micro:bita, s analognim zadacima, ali promijenjenim vrijednostima u zadacima kako bi se izbjeglo odgovaranje temeljeno na prisijećanju.
- ✓ **Završni ispit znanja u Pythonu** (ZI-P) je proveden tjedan nakon završnog ispita u Micro:bitu, s istim zadacima, ali drugačijim korištenim brojevima te nekim redoslijedima odgovora kako bi se izbjeglo odgovaranje prisijećanjem. U svrhu dodatnog ispitivanja razumijevanja koncepata slijeda i petlje, u ZI-P dodano je još devet zadataka otvorenog tipa u kojima učenici nisu imali ponuđene odgovore te u kojima su trebali objasniti dana rješenja.

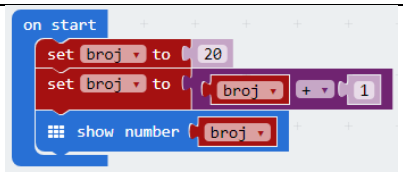
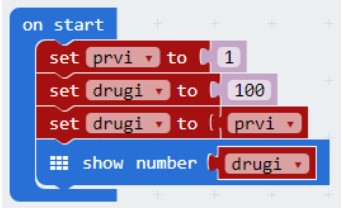
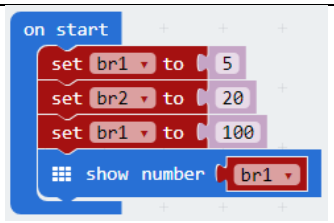
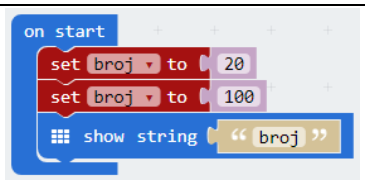
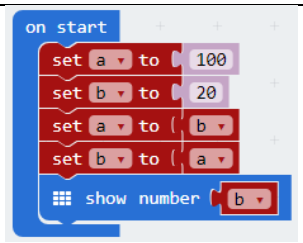
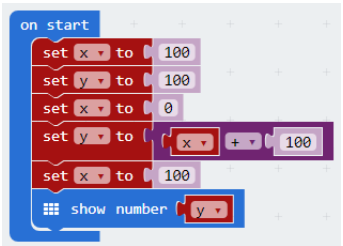
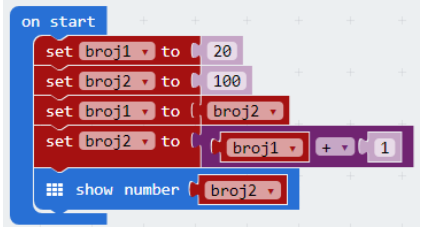
U svrhu valjanosti, s učenicima nije provedena analiza pisanih ispita prije nego su pisali sva tri ispita. Na taj se način izbjeglo memoriranje ispitnih zadataka i odgovaranje u sljedećim ispitima temelejno na rezultatima prethodnih. Svi učenici vježbali su slične primjere tijekom nastavnog procesa, te nisu imali dodatne zadatke za rad izvan nastave u obliku domaćih zadaća. Sami ispiti nisu ocjenjivani, a učenici su za pisanje ispita motivirani s potencijalno ocjenom odličan (za učenike koji budu među najuspješnijima), prihvaćanjem ponuđene druge ocjene ili plusom za aktivno sudjelovanje u ispitu.

Zadaci su grupirani u tri skupine koje odgovaraju ispitivanim algoritmima. U prvoj skupini se ispitivalo razumijevanje algoritma slijeda s naglaskom na pojam varijable. U tablici 4.50 su prikazani zadaci iz prve skupine koji se odnose na algoritam slijeda za sva tri ispita.

Tablica 4.50 - Zadaci iz prve skupine (algoritam slijeda)

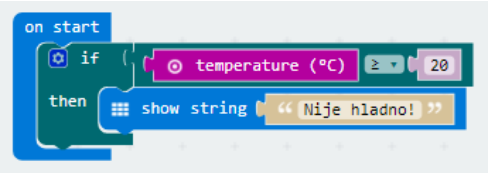
	PI-Python	**PI-Micro:bit**	**ZI-Python**
1. U tablici ispod je 8 zadataka. Pažljivo pogledaj zadatke pa **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa.			

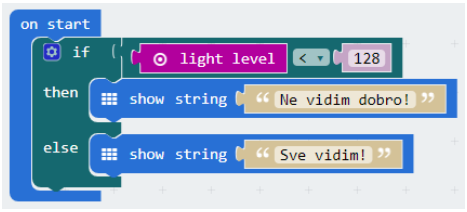
i)	`broj = 100`
	`broj = 20`
	`print (broj)`
			`broj = 2`
			`broj = 1`
			`print (broj)`

ii)	broj = 100 broj = broj+1 print (broj)		broj = 10 broj = broj+1 print (broj)
iii)	prvi = 100 drugi = 1 drugi = prvi print (drugi)		prvi = 1 drugi = 2 drugi = prvi print (drugi)
iv)	br1 = 100 br2 = 20 br1 = 5 print (br1)		br1 = 100 br2 = 5 br1 = 20 print (br1)
v)	broj = 100 broj = 20 print ('broj')		broj = 2 broj = 1 print ('broj')
vi)	a = 20 b = 100 a = b b = a print (b)		a = 10 b = 2 a = b b = a print (b)
vii)	x = 0 y = 0 x = 100 y = x+100 x = 1 print (y)		x = 100 y = 100 x = 0 y = x+100 x = 1 print (y)
viii)	broj1 = 100 broj2 = 20 broj1 = broj2 broj2 = broj1+1 print (broj2)		broj1 = 1 broj2 = 2 broj1 = broj2 broj2 = broj1+1 print (broj2)

U drugoj skupini zadataka se ispitivalo razumijevanje algoritma grananja korištenjem *if-else* strukture. Napravljena su ukupno dva zadatka s tri pitanja što čini ukupno 6 pitanja U tablici 4.51 su prikazani zadaci druge skupine.

Tablica 4.51 - Zadaci druge skupine (algoritam grananja)

<table>
<tr>
<td>PI-P</td>
<td>
a) Napravljen je program u kojem se unosi razina svjetla. Ako je razina svjetla barem 128 vidi se dobro i treba ispisati 'Sve vidim!'. Program je prikazan ispod. Prouči program.
<pre>
razina_svjetla = int(input())
if razina_svjetla >= 128:
        print ('Sve vidim!')
</pre>
b) Napravljen je program u kojem se unosi temperatura. Ako je unesena temperatura ispod 20 treba ispisati 'Hladno', dok za ostale slučajeve treba ispisati 'Nije hladno'. Program je prikazan ispod. Prouči program.
<pre>
temperatura = int(input())
if temperatura < 20:
    print ('Hladno')
else:
print ('Nije hladno')
</pre>
</td>
</tr>
<tr>
<td>ZI-M</td>
<td>
a) Napravljen je program u kojem se očitava temperatura. Ako je temperatura barem 20 toplo je i treba ispisati 'Nije hladno!'. Program je prikazan ispod. Prouči program.


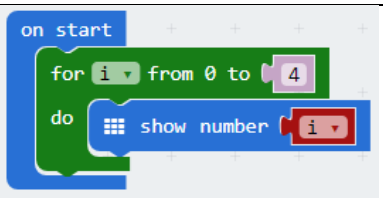
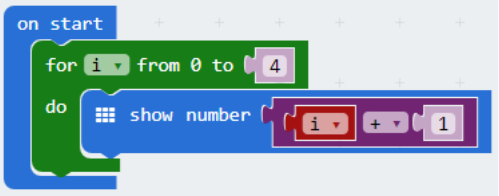
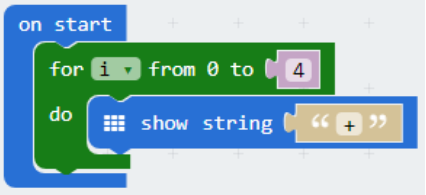
b) Napravljen je program u kojem se očitava razina svjetla. Ako je očitana razina svjetla manja od 128 treba ispisati 'Ne vidim dobro!', dok za ostale slučajeve treba ispisati 'Sve vidim!'. Program je prikazan ispod. Prouči program.


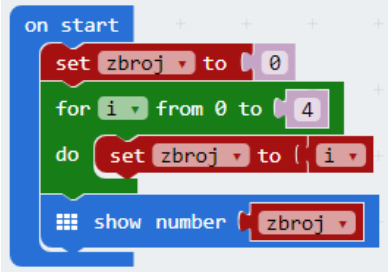
</td>
</tr>
</table>

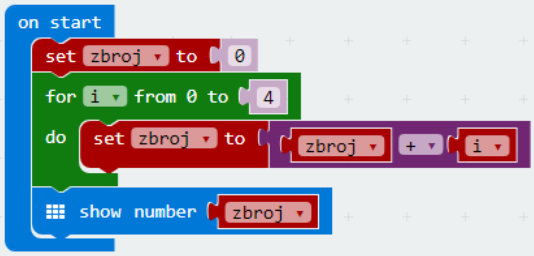
ZI-P	a) Napravljen je program u kojem se očitava temperatura mora. Ako je temperatura mora barem 22 more je ugodno za kupanje i treba ispisati 'Nije hladno!'. Program je prikazan ispod. Prouči program.
	`temperatura = int(input())`
	`if temperatura >= 22:`
	`  print ('Nije hladno!')`
	b) Napravljen je program u kojem se očitava razina svjetla. Ako je očitana razina svjetla manja od 130, osvjetljenje u prostoriji nije dovoljno dobro pa treba ispisati 'Ne vidim dobro!', dok za ostale slučajeve treba ispisati 'Sve vidim!'. Program je prikazan ispod. Prouči program.
	`svjetlo = int(input())`
	`if svjetlo < 130`
	`  print ('Ne vidim dobro!')`
	`else:`
	`  print ('Sve vidim!')`

U trećoj skupini zadataka se ispitivalo razumijevanje algoritma ponavljanja korištenjem *for* strukture. Napravljena su ukupno dva zadatka s tri pitanja što čini ukupno 6 pitanja U tablici 4.52 su prikazani zadaci druge skupine.

Tablica 4.52 - Zadaci treće skupine

	PI-Python	ZI-Micro:bit	ZI-Python
	Za rješavanje sljedećih zadataka prvo prouči program, a zatim **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa		
a)	`for i in range (0,6):` `  print (i)`		`for i in range (0,6):` `  print (i)`
b)	`for i in range (0,6):` `  print (i+1)`		`for i in range (0,6):` `  print (i+1)`
c)	`for i in range (0,6):` `  print ('+')`		`for i in range (0,6):` `  print ('+')`

d)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = i`
	`print (zbroj)`
			`zbroj = 0`
			`for i in range (0,6):`
			`  zbroj = i`
			`print (zbroj)`

e)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = zbroj + i`
	`print (zbroj)`
			`zbroj = 0`
			`for i in range (0,4):`
			`  zbroj = zbroj + i`
			`print (zbroj)`

Završni ispit u Pythonu sadržavao je dodatnih devet zadataka u odnosu na ostale ispite. To su zadaci otvorenog tipa koji su se odnosili na koncepte slijeda i ponavljanja. Grananje je izostavljeno jer je u prethodnim testovima to bio najbolje riješen koncept u svim grupama pa se ispitivanje usmjerilo na koncepte koji su učenicima bili teži. Odabrani su zadaci otvorenog tipa kako bi mogli detaljnije analizirati razumijevanje ispitivanih koncepata u slučaju kada nisu ponuđeni odgovori. Uz odgovor, od učenika se tražilo i objašnjenje ponuđenog rješenja. U tablici 4.53 su prikazani zadaci otvorenog tipa.

Tablica 4.53 - Zadaci otvorenog tipa u ZI-P

<table>
<tr><td colspan="6">4. Za rješavanje sljedećih zadataka prvo prouči program, a zatim napiši odgovor koji bi prikazivao točan rezultat izvršavanja programa.</td></tr>
<tr><td>a)</td><td><pre>br1 = 5
br2 = 100
br1 = 20
print (br1)</pre></td><td>b)</td><td><pre>prvi = 10
drugi = 20
drugi = prvi
print (drugi)</pre></td><td>c)</td><td><pre>x = 10
y = 20
x = 0
y = x+10
x = 1
print (y)</pre></td></tr>
<tr><td>d)</td><td><pre>broj1 = 10
broj2 = 20
broj1 = broj2
broj2 = broj1+10
print (broj2)</pre></td><td>e)</td><td><pre>for i in range (0,7):
    print (i)</pre></td><td>f)</td><td><pre>for i in range (1,4):
    print (i)</pre></td></tr>
<tr><td>e)</td><td><pre>for i in range (0,3):
    print (i+1)</pre></td><td>h)</td><td><pre>zbroj = 0
for i in range (1,4):
    zbroj = zbroj + i
print (zbroj)</pre></td><td>i)</td><td><pre>zbroj = 0
for i in range (0,4):
    zbroj = zbroj + 1
print (zbroj)</pre></td></tr>
</table>

Prva četiri zadatka (*a,b,c,d)* odnosila su se na algoritam slijeda, a ostalih pet (*e, f, g, h, i*) na algoritam ponavljanja. Učenici su na papiru imali prostor za odgovor i za objašnjenje odgovora.

4.4.2.2.1 Metrijske karakteristike ispita

Različit broj učenika sudjelovao je u svakom od ispita jer svi učenici nisu završili cijeli tretman pa njihovi rezultati nisu uzeti u daljnju analizu, ali su za utvrđivanje metrijskih karakteristika ispita svi rezultati uzeti u obradu.

Za utvrđivanje pouzdanosti ispita korišten je Cronbach α koeficijent. Svi završni ispiti spadaju u kategoriju visoko pouzdanih (Cronbach $\alpha > 0.8$), preliminarni ispit sposobnosti rješavanja problema spada u pouzdane (Cronbach $\alpha > 0.7$). U tablici 4.54 su prikazane metrijske karakteristike ispita.

Tablica 4.54 - Metrijske karakteristike ispita

	PI sposobnosti rješavanja problema	PI Python	ZI Micro:bit	ZI Python	ZI Python Petlje	ZI Python Prošireni
n	95	47	47	47	47	47
aritmetička sredina	8.48	10.53	15.28	14.38	5.3617	19.7447
aritmetička sredina (u postocima)	53	55.43	80.40	75.7	51.5133	70.5168
M (Medijan)	9	11	17	16	6	21
SD	3.826	4.128	3.728	3.627	2.47092	5.8327
Minimum	0	3	4	6	0	6
Maksimum	16	18	19	19	9	27
Maksimalan mogući broj bodova	16	19	19	19	9	28
Broj čestica	12	19	19	19	9	28
Cronbach α	0.727	0.821	0.847	0.809	0.797	0.889
Kolmogorov-Smirnov test/Shapiro-Wilk	(K-S) 0.00	(S-W) 0.053	(S-W) 0.00	(S-W) 0.00	(S-W) 0.18	(S-W) 0.01

4.4.2.3 Anketa zadovoljstva programiranjem, Pythonom, Micro:bitom i programiranjem igara

Učenici su ispunjavali anketu u on-line obliku nakon cijelog tretmana, odnosno obrađene nastavne cjeline programiranja korištenjem Pythona pa Micro:bita. Anketu su učenici ispunjavali dobrovoljno, ali ne i anonimno kako bi se mogli usporediti stavovi s postignutim rezultatima. Učenici su motivirani na način da su dobili plus za sudjelovanje. Nije izvršen nikakav utjecaj na učenike u obliku sugeriranja odgovora već se tražilo da što iskrenije daju odgovore te po želji napišu i komentare. Anketu su učenici ispunjavali prije pisanja završnih ispita kako zadaci ne bi utjecali na neposredni dojam nakon programiranja. Tablica 4.55 prikazuje anketu zadovoljstva programiranjem i programskim jezicima.

Tablica 4.55 - Anketa zadovoljstva programiranjem i programskim jezicima

Pitanje	Ishod	Tip zadatka
P1	Koji ti se programski jezik više sviđa? 1) Micro:Bit 2) Python	Zadatak višestrukog izbora s jednim odgovorom
P2	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Python.	Likertova skala (1-5)
P3	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje Micro:Bita.	Likertova skala (1-5)
P4	Kojim bi programskim jezikom učio/la programiranje iduće školske godine? 1) Python 2) Micro:Bit	Likertova skala (1-5)
P5	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje igara (na primjer, kamen-škare-papir) na Micro:Bitu.	Likertova skala (1-5)
P6	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje algoritama na Micro:Bitu (zadaci slični onima u Pythonu).	Likertova skala (1-5)
P7	Što ti se više sviđa: 1) programiranje igre u Micro:bitu 2) Programiranje matematičkih zadataka u Micro:bitu 3) Programiranje matematičkih zadataka u Pythonu	Zadatak višestrukog izbora s jednim odgovorom
P8	Ako imaš neki svoj komentar u vezi programiranja Micro:bita slobodno ga napiši.	Pitanje otvorenog tipa
P9	Ako imaš neki svoj komentar u vezi usporedbe Micro:bita i Pythona slobodno ga napiši.	Pitanje otvorenog tipa

Rezultati obrade ankete će se analizirati u poglavlju s obradom podataka.

4.4.3 Utvrđivanje ujednačenosti između grupa

S obzirom na utvrđenu povezanost rješavanja problema i programiranja, potrebno je utvrditi jesu li grupe ujednačene prema navedenom kriteriju. Prije tretmana učenici su imali predznanje o programiranju i to u Pythonu pa se prema tom kriteriju trebalo utvrditi postoje li razlike između grupa prema navedenom kriteriju. U svrhu utvrđivanja ujednačenosti grupa prema nezavisnim varijablama: spol, pripadnost kontrolnoj (K) ili eksperimentalnoj (E) grupi, u odnosu na preliminarne ispite sposobnosti rješavanja problema (PI) i rezultata u Pythonu (PI-P) korišten je Mann-Whitney neparametrijski test. U tablici 4.56 su prikazani rezultati Mann-Whitney testa.

Tablica 4.56 - Rezultati Mann-Whitney U testa za utvrđivanje razlike između grupa

		Spol	Skupina
Preliminarni ispit sposobnosti rješavanja problema	U	214.50	206.50
	p	0.415	0.654
Preliminarni ispit -Python	U	269.0	237.0
	p	0.983	0.945

Prema rezultatima testa nisu utvrđene statistički značajne razlike ni po jednom kriteriju pa se grupe može tretirati kao ujednačene i nastaviti s analizom podataka.

4.4.4 Usporedba ukupnih rezultata ispita između grupa

S obzirom da postoje dvije skupine – eksperimentalna (E) i kontrolna (K), za utvrđivanje razlika između rezultata završnih ispita korišten je Mann-Whitneyev test. Uspoređivali su se rezultati testova: preliminarni ispit u Pythonu (PI-P), završni ispit u Micro:bitu (ZI-M) i završni ispit u Pythonu (ZI-P). Osim toga, usporedili su se rezultati u završnom ispitu u Pythonu sa zadacima otvorenog tipa (ZI-P-ZOT). Rezultati Mann-Whitneyog testa prikazani su u tablici 4.57.

Tablica 4.57 - Rezultati Mann-Whitneyevog testa

	PI-P	ZI-M	ZI-P	ZI – P- ZOT
Mann-Whitney U	237.000	211.000	212.500	114.50
Z	-0.069	-0.671	-0.633	-2.890
p	0.945	0.503	0.526	**0.004**

Iz rezultata je vidljivo kako je jedina statistički značajna razlika u rezultatima između grupa u zadacima otvorenog tipa u ZI-P. U tablici 4.58 je prikazana deskriptivna statistika rezultata po skupinama za ukupne rezultate po ispitima.

Tablica 4.58 - Deskriptivna statistika po skupinama za ukupne rezultate po ispitima

Ispit	Skupina	N	AS	Median	Mod	Min	Max
PI -Python	K	15	10.53	12.0	6	4	17
	E	32	10.53	10.5	14	3	18
ZI- Micro:bit	K	15	15.47	17.0	19	8	19
	E	32	15.19	17.0	17	4	19
ZI - Python	K	15	14.0	15.0	15	6	19
	E	32	14.56	16.0	17	6	19
ZI – Py - ZOT	K	15	4	4.0	0	0	8
	E	32	6	7.0	7	0	9
ZI – Py - Ukupno	K	15	18	20.0	20	6	27
	E	32	20,5625	23.0	24	6	27

Iz tablice 4.58 proizlazi kako su rezultati učenika ujednačeni po znanju u Pythonu (AS=10.53) prije tretmana (PI-Python). Nakon tretmana se vidi kako je kontrolna (K) skupina nešto bolja od eksperimentalne (E) u Micro:bitu, ali i kako je eksperimentalna (E) skupina nešto bolja ZI-Python. Ako se pogleda napredak u rezultatima preliminarnih i završnih ispita znanja u Pythonu može se zaključiti kako su uz pomoć vizualno-blokovskog programskog jezika Micro:bit svi učenici postigli značajan napredak u usvajanju koncepata slijeda, grananja i ponavljanja.

Kako bi dobili detaljniji uvid u razlike po grupama prema poučavanim konceptima, u daljnjoj analizi će se utvrđivati postoje li razlike među grupama prema ispitivanim konceptima.

4.4.5 Usporedba rezultata ispita po ispitivanim algoritmima između grupa

U ispitima se utvrđivalo razumijevanje tri osnovna koncepta, odnosno algoritma: slijed, grananje i ponavljanje. U svakom ispitu dio zadataka je ispitivao jedan od navedena tri koncepta pa se i rezultati između sva tri ispita mogu usporediti po razini usvojenosti koncepta. Za utvrđivanje razlika između rezultata završnih ispita korišten je Mann-Whitney test. Rezultati su prikazani u tablici 4.59.

Tablica 4.59 - Rezultati Mann-Whitney testa

	Slijed				**Grananje**			**Ponavljanje**			
ispit	PI-P	ZI-M	ZI-P	ZI-P-ZOT	PI-P	ZI-M	ZI-P	PI-P	ZI-M	ZI-P	ZI-P-ZOT
Mann-Whitney U	209.50	206.50	234.00	227.50	225.50	188.50	190.50	210.50	222.00	162.50	65.00
Z	-0.704	-0.843	-0.151	-0.333	-0.342	-1.272	-1.201	-0.707	-0.422	-1.816	-4.073
p	0.481	0.399	0.880	0.739	0.733	0.203	0.230	0.480	0.673	0.069	**0.000**

Jedina statistički značajna razlika između grupa odnosi se na usvajanje koncepta ponavljanja, odnosno petlje i to u zadacima otvorenog tipa. U tablici 4.60 prikazana je deskriptivna statistika rezultata po grupama iz koje se mogu vidjeti postignuća na ispitima po grupama.

Tablica 4.60 - Deskriptivna statistika rezultata za svaki ispitivani koncept po skupinama

Ispit	Skupina	N	AS	Median	Mod	Min	Max
PI -P-Slijed	K	15	5.4	6.0	3	2	8
	E	32	4.91	5.0	7	0	8
PI-P-Grananje	K	15	4.20	5.0	5	0	6
	E	32	4.41	5.0	5	1	6
PI-P-Ponavljanje	K	15	0.93	1.0	0	0	4
	E	32	1.22	1.0	0	0	5
ZI- M-Slijed	K	15	6.80	7.0	8	4	8
	E	32	6.91	8.0	8	1	8
ZI – M- Grananje	K	15	5.33	6.0	6	2	6
	E	32	5.00	5.0	6	0	6
ZI – M - Ponavljanje	K	15	3.33	4.0	5	0	5
	E	32	3.28	4.0	4	1	5
ZI -P-Slijed	K	15	6.53	8.0	0	1	8
	E	32	6.59	8.0	0	2	8
ZI-P-Grananje	K	15	5.27	6.0	6	3	6
	E	32	4.94	5.0	6	2	6
ZI-P-Ponavljanje	K	15	2.20	2.0	2	0	5
	E	32	3.03	3.0	3	1	5
ZI-ZOT- Slijed	K	15	3.33	4.0	4	0	4
	E	32	3.19	4.0	4	0	4
ZI-ZOT- Ponavljanje	K	15	0.67	0.0	0	0	4
	E	32	2.81	3.0	3	0	5

Iz rezultata je vidljivo kako je eksperimentalna (E) skupina postigla, statistički značajno, bolje rezultate u zadacima otvorenog tipa za algoritam ponavljanja u odnosu na kontrolnu (K) skupinu. Kako bi se utvrdio uzrok ovakve razlike, u daljnjem istraživanju će se usporediti rezultati svih učenika između ispita te će se analizirati rezultati po zadacima.

4.4.6 Usporedba rezultata učenika među ispitima

S obzirom da su ispiti u tri ispitivane faze bili isti tako se i rezultati testova svakog učenika mogu uspoređivati. Za navedenu usporedbu se koristio Wilcoxonov test. Uspoređivali su se rezultati ukupnih rezultata završnih ispita kao i rezultati po ispitivanim konceptima.

4.4.6.1 Usporedba ukupnih rezultata ispita

S obzirom da su tri ispita (PI-P, ZI-M, ZI-P) sadržavala analogne zadatke, tako su ukupni rezultati učenika između ispita međusobno usporedivi. Ispiti ukupno imaju 19 zadataka. Rezultati Wilcoxonovog testa za usporedbu ukupnih rezultata svih ispita prikazani su u tablici 4.61.

Tablica 4.61 - Rezultati Wilcoxonovog testa za ukupne rezultate učenika

		SVI		**K-Skupina**		**E-skupina**	
Rezultati	Parovi	Z	p	Z	p	Z	p
Ukupni rezultati	Završni ispit Micro:bit (ZI_M) – Preliminarni ispit Python (PI_P)	-5.917	**0.000**	-3.417	**0.001**	-4.873	**0.000**
	Završni ispit Python (ZI_P) - Preliminarni ispit Python (PI_P)	-5.857	**0.000**	-3.316	**0.001**	-4.873	**0.000**
	Završni ispit Python (ZI_P) - Završni ispit Micro:bit (ZI_M)	-3.798	**0.000**	-2.885	**0.004**	-2.517	**0.012**

Iz tablice je vidljivo kako postoje statistički značajne razlike između ukupnih rezultata preliminarnog ispita i svih završnih ispita, ali i između dva završna ispita i to u svim skupinama. U tablici 4.62 je prikazana deskriptivna statistika ispita gdje se mogu vidjeti razlike između rezultata.

Tablica 4.62 - Deskriptivna statistika za ukupne rezultate

		SVI						**K-Skupina**						**E-skupina**					
Rezultati	Ispit	N	AS	M	Mod	Min	Max	N	AS	M	Mod	Min	Max	N	AS	M	Mod	Min	Max
Ukupni rezultati	PI_P	47	10.53	11.0	14	3	18	15	10.53	12.0	6	4	17	32	10.53	10.5	14	3	18
	ZI_M	47	15.28	17.0	17	4	19	15	15.47	17.0	19	8	19	32	15.19	17.0	17	4	19
	ZI_P	47	14.38	16.0	16	6	19	15	14.00	15.0	15	6	19	32	14.56	16.0	17	6	19

Ukupni prosječni rezultati eksperimentalne (E) skupine (AS=10.53) i kontrolne (K) skupine (AS=10.53) su bili ujednačeni prije tretmana. Očito je poboljšanje rezultata u oba završna ispita između kojih je kontrolna (K) skupina postigla bolji rezultat u Micro:bitu (AS=15.47) u odnosu

na eksperimentalnu (AS=15.19), ali nešto lošiji u Python završnom ispitu (AS=14.00) u odnosu na eksperimentalnu (E) skupinu (AS=15.56). Iz rezultata se može zaključiti kako su svi učenici postigli bolje rezultate na završnom ispitu Pythonu (ZI_P) u odnosu na preliminarni ispit (PI_P) nakon korištenja blokovskog programskog jezika Micro:bit.

Kako bi se utvrdile eventualne razlike u rezultatima prema ispitivanim algoritmima u daljnjoj analizi će se usporediti rezultati prema ispitivanim algoritmima.

4.4.6.2 Usporedba rezultata – algoritam slijeda

U ispitima je bilo ukupno 19 zadataka od čega se 8 odnosilo na algoritam slijeda. U tablici 4.63 su prikazani rezultati Wilcoxonovog testa za zadatke koji se odnose na algoritam slijeda.

Tablica 4.63 - Rezultati Wilcoxonovog testa za rezultate za koncept slijeda

		SVI		**K-Skupina**		**E-skupina**	
Rezultati	Parovi	Z	p	Z	p	Z	p
Slijed	Završni ispit Micro:bit (ZI_M) - Preliminarni ispit Python (PI_P)	-4.841	**0.000**	-2.831	**0.005**	-4.012	**0.000**
	Završni ispit Python (ZI_P) - Preliminarni ispit Python (PI_P)	-4.857	**0.000**	-2.435	**0.015**	-4.200	**0.000**
	Završni ispit Python (ZI_P) - Završni ispit Micro:bit (ZI_M)	-1.473	0.141	-0.962	0.336	-1.101	0.271

Iz tablice se vidi kako postoji statistički značajna razlika između rezultata preliminarnog ispita u Pythonu i obaju završnih ispita dok razlika između završnih ispita obje grupe nije statistički značajna.

Tablica 4.64 - Deskriptivna statistika za rezultate slijeda

		SVI					**K-Skupina**					**E-skupina**				
Rezultati	Ispit	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max
Slijed	PI_P	47	5.0	7	0	8	15	6.0	3	2	8	32	5.0	5	0	8
	ZI_M	47	8.0	8	1	8	15	7.0	8	4	8	32	8.0	8	1	8
	ZI_P	47	8.0	8	1	8	15	8.0	8	1	8	32	8.0	8	2	8

U tablici 4.64 je prikazana deskriptivna statistika za rezultate slijeda gdje se mogu vidjeti razlike između rezultata. Prema podacima u tablici vidljiv je napredak u završnim ispitima u odnosu na preliminarni ispit. Uočljivo je da je eksperimentalna (E) skupina ostvarila nešto bolji rezultat u oba završna ispita od kontrolne (K) skupine iako ne statistički značajne. Iz navedenog se može zaključiti da su učenici postigli bolje rezultate u tekstualnom programskom jeziku Python nakon korištenja blokovskog jezika Micro:bit, odnosno da su bolje usvojili algoritam slijeda.

4.4.6.3 Usporedba rezultata – algoritam grananja

U ispitima se od ukupno 19 zadataka 6 odnosilo na algoritam grananja. U tablici 4.65 su prikazani rezultati Wilcoxonovog testa za zadatke koji se odnose na algoritam grananja.

Tablica 4.65 - Rezultati Wilcoxonovog testa za rezultate za koncept grananja

		SVI		**K-Skupina**		**E-skupina**	
Rezultati	Parovi	Z	p	Z	p	Z	p
Grananje	Završni ispit Micro:bit (ZI_M) - Preliminarni ispit Python (PI_P)	-3.727	**0.000**	-3.002	**0.003**	-2.375	**0.018**
	Završni ispit Python (ZI_P) - Preliminarni ispit Python (PI_P)	-3.283	**0.001**	-2.658	**0.008**	-2.110	**0.035**
	Završni ispit Python (ZI_P) - Završni ispit Micro:bit (ZI_M)	-0.481	0.630	-0.378	0.705	-0.353	0.724

Slično kao i kod rezultata za algoritma slijeda, i kod rezultata algoritma grananja se pokazala statistički značajna razlika između oba završna ispita i preliminarnog ispita. U tablici 4.66 je prikazana deskriptivna statistika za rezultate grananja gdje se mogu usporediti ostvareni rezultati.

Tablica 4.66 - Deskriptivna statistika za rezultate grananja

		SVI					**K-Skupina**					**E-skupina**				
Rezultati	Ispit	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max
Grananje	PI_P	47	5.0	5	0	6	15	5.0	5	0	6	32	5.0	5	1	6
	ZI_M	47	5.0	6	0	6	15	6.0	6	2	6	32	5.0	6	0	6
	ZI_P	47	5.0	6	2	6	15	6.0	6	3	6	32	5.0	6	2	6

Prema podacima iz tablice vidljiv je pomak u rezultatima završnih ispita u odnosu na preliminarni ispit. Ovaj put je kontrolna (K) skupina postigla nešto bolje rezultate u oba završna ispita u odnosu na eksperimentalnu (E) skupinu iako ne statistički značajne. Uočeno je kako su učenici i nakon preliminarnog ispita najbolje rezultate ostvarili u zadacima koji se odnose na grananje.

4.4.6.4 Usporedba rezultata – algoritam ponavljanja

U ispitima se od ukupno 19 zadataka 5 odnosilo na algoritam ponavljanja. U tablici 4.67 su prikazani rezultati Wilcoxonovog testa za zadatke koji se odnose na algoritam ponavljanja.

Tablica 4.67 - Rezultati Wilcoxonovog testa za rezultate za koncept ponavljanja

		SVI		K-Skupina		E-skupina	
Rezultati	Parovi	Z	p	Z	p	Z	p
Ponavljanje	Završni ispit Micro:bit (ZI_M) - Preliminarni ispit Python (PI_P)	-5.733	**0.000**	-3.154	**0.002**	-4.826	**0.000**
	Završni ispit Python (ZI_P) - Preliminarni ispit Python (PI_P)	-5.482	**0.000**	-2.701	**0.007**	-4.785	**0.000**
	Završni ispit Python (ZI_P) - Završni ispit Micro:bit (ZI_M)	-3.056	**0.002**	-2.377	**0.017**	-1.641	0.101

Učenici su postigli statistički značajne razlike u završnim ispitima u odnosu na preliminarni ispit. Za razliku od rezultata slijeda i grananja gdje nije bilo statistički značajnih razlika između rezultata u završnim ispitima, kod rezultata za algoritam ponavljanja postoji statistički značajna razlika između rezultata završnih ispita u Micro:bitu i Pythonu. Kako je utvrđena statistički značajna razlika između rezultata završnih ispita za ukupne rezultate, a nije je bilo kod rezultata za slijed i grananje očito je da je ta razlika nastala za algoritam ponavljanja. U tablici 4.68 je prikazana deskriptivna statistika za rezultate algoritma ponavljanja gdje se mogu usporediti ostvareni rezultati.

Tablica 4.68 - Deskriptivna statistika za rezultate ponavljanja

		SVI					K-Skupina					E-skupina				
Rezultati	Ispit	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max	N	Mdn	Mod	Min	Max
Ponavljanje	PI_P	47	1.0	0	0	5	15	1.0	0	0	4	32	1.0	0	0	5
	ZI_M	47	4.0	4	0	5	15	4.0	5	0	5	32	4.0	4	1	5
	ZI_P	47	3.0	2	0	5	15	2.0	2	0	5	32	3.0	3	1	5

Kao što se pokazalo i u prethodnim fazama istraživanja učenicima je algoritam ponavljanja najsloženiji, što se može potvrditi i rezultatima ovog istraživanja. Već u rezultatima preliminarnog ispita je vidljivo kako su učenici imali najviše problema s algoritmom ponavljanja. Kod rezultata završnog ispita u Micro:bitu u odnosu na preliminarni ispit vidljiv je napredak kod rezultata obiju grupa. Najznačajniji su ipak rezultati završnog ispita u Pythonu gdje je očita razlika u ostvarenim rezultatima između eksperimentalne (E) i kontrolne (K) skupine. Učenici iz kontrolne skupine postigli su statistički značajno lošije rezultate u završnom ispitu u Pythonu u odnosu na ostvarene rezultate u završnom ispitu u Micro:bitu što nije slučaj kod eksperimentalne skupine. Učenici iz kontrolne skupine lošije su riješili završni ispit u Pythonu u odnosu na završni ispit u Micro:bitu iz čega se može zaključiti kako je kod učenika eksperimentalne skupine ostvaren posredovani prijenos znanja premošćivanjem iz blokovskog u tekstualni programski jezik. Moguće je da je navedena razlika nastala upravo zbog različitih tretmana među grupama. U obje grupe se na primjere u Pythonu potrošilo otprilike isto

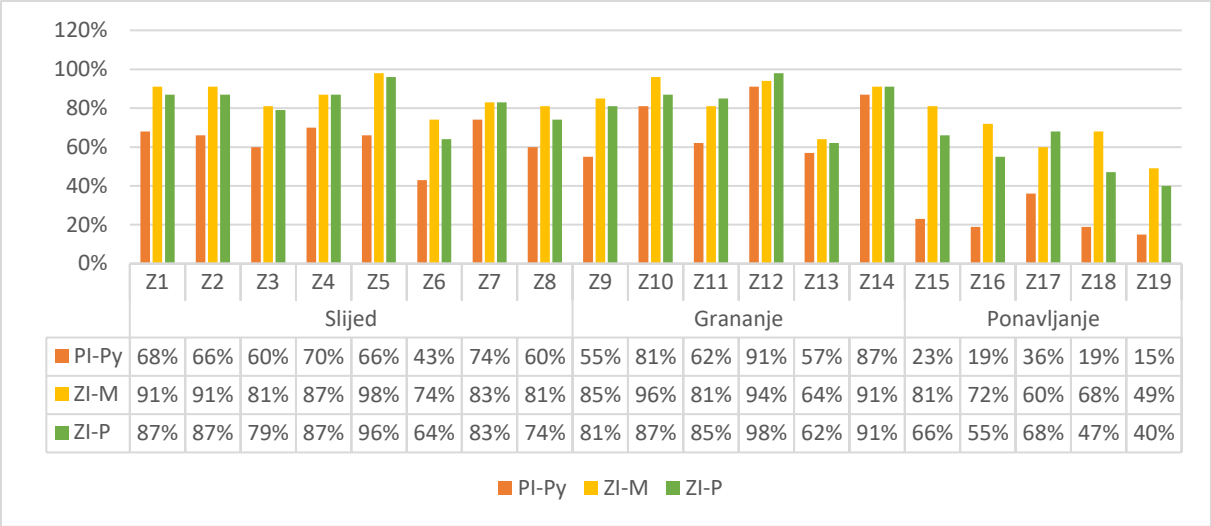
vremena, ali na različite načine. Važno je naglasiti da ni u jednoj grupi učenici nisu samostalno programirali u Pythonu već su im se samo pokazivali primjeri u kontekstu ponavljanja obrađivanih pojmova. Primjeri u Pythonu su u kontrolnoj (K) skupini prikazivani na početku svake nastavne jedinice kao kratko ponavljanje naredbi koje su već obrađivali u Pythonu. Navedeni primjeri nisu eksplicitno povezivani s analognim Python programima. Nakon takvog kratkog ponavljanja slijedila je obrada nastavnih jedinica u Micro:bitu. U eksperimentalnoj (E) skupini su navedeni primjeri u Pythonu eksplicitno povezani s Micro:bit programima tijekom obrade istih naredbi u Micro:bitu. Očito je zbog takvog načina obrade došlo do razlike u rezultatima. Iako je otprilike isti udio vremena utrošen na kratko ponavljanje Python primjera, kod eksperimentalne skupine je direktnim povezivanjem primjera programa u oba okruženja (metoda „premošćivanja“) učitelj direktno stvarao poveznicu između dva okruženja. Kod kontrolne skupine isti primjeri u Pythonu su samo ponovljeni na početku svakog sata bez direktne poveznice („metoda prigljivanja“). Prilikom prikazivanja sadržaja kod druge metode je potreban veći mentalni napor učenika, odnosno potrebna je veća razina apstrakcije za samostalno povezivanje sadržaja. Kako je algoritam ponavljanja apstraktniji u odnosu na druga dva osnovna algoritma, vjerojatno je potrebno učenicima napraviti poveznicu pri povezivanju dva okruženja. Učenici iz kontrolne skupine jesu ostvarili napredak u rezultatima završnog ispita u Pythonu u odnosu na preliminarni ispit u Pythonu, ali su u usporedbi s rezultatima učenika iz eksperimentalne skupine ostvarili statistički značajno lošiji rezultat. Može se zaključiti da direktno povezivanje apstraktnog pojma algoritma ponavljanja, u blokovskom i tekstualnom programskom jeziku, pridonosi boljem razumijevanju algoritma.

Temeljem ovih rezultata može se potvrditi da korištenjem blokovskog programskog jezika uz direktno povezivanje s prethodno učenim tekstualnim programskim jezikom može potaknuti usvajanje algoritma ponavljanja u oba okruženja.

4.4.7 Usporedba rezultata ispita po zadacima

U sljedećoj fazi obrade provedena je usporedba rezultata po zadacima. Sva tri ispita u programskim jezicima su sadržavala analogne zadatke. Jedina razlika u zadacima je u završnom ispitu u Pythonu gdje je osim dijela ispita koji je zajednički s dva prethodna, dodano još devet zadataka otvorenog tipa za ispitivanje razumijevanja algoritama slijeda i ponavljanja.

Na slici 4.9 su prikazani rezultati za svaki zadatak u sva tri ispita.



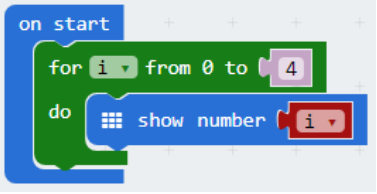
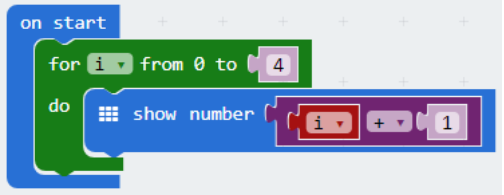
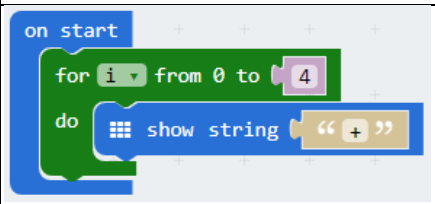
	Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8	Z9	Z10	Z11	Z12	Z13	Z14	Z15	Z16	Z17	Z18	Z19
PI-Py	68%	66%	60%	70%	66%	43%	74%	60%	55%	81%	62%	91%	57%	87%	23%	19%	36%	19%	15%
ZI-M	91%	91%	81%	87%	98%	74%	83%	81%	85%	96%	81%	94%	64%	91%	81%	72%	60%	68%	49%
ZI-P	87%	87%	79%	87%	96%	64%	83%	74%	81%	87%	85%	98%	62%	91%	66%	55%	68%	47%	40%

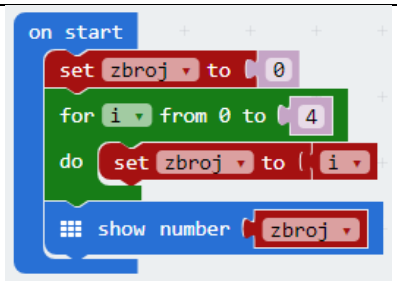
Slika 4.9 – Usporedba rezultata ispita po zadacima

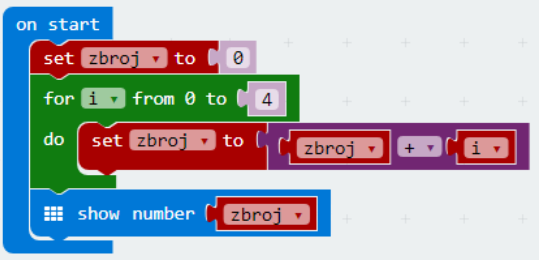
Očita je razlika u rezultatima između preliminarnog ispita u Pythonu (PI-Py) u odnosu na završne ispite u Micro:bitu (ZI-M) i Pythonu (ZI-P). Iako su PI-Py učenici pisali neposredno nakon obrađene nastavne cjeline u Pythonu, prema rezultatima je očito su lošiji rezultati u odnosu na završne testove nakon obrade programiranja pomoću Micro:bita. Iako se naknadno nije više radilo u Pythonu, očit je i pomak u rezultatima u završnom ispitu u Pythonu. Kako su zadaci podijeljeni prema ispitivanim algoritmima: slijed, grananje i ponavljanje, i ovdje se može uočiti da učenici najviše problema imaju s algoritmom ponavljanja te da su rezultati najbolji u Micro:bitu što potvrđuje rezultate istraživanja iz prethodnih faza istraživanja u kojima se pokazala statistički značajna razlika u korist blokovskog programskog jezika kada je u pitanju usvajanje koncepta petlje, odnosno algoritma ponavljanja.

Za usporedbu rezultata između grupa po svakom zadatku napravljen je Mann-Whitney U test. Jedine statistički značajne razlike između rezultata po skupinama su se pokazale za zadatke Z15 ($U=125.00$, $p=0.001$) i Z18 ($U=122.00$, $p=0.002$). S obzirom da su oba zadatka iz skupine ispitivanja algoritma ponavljanja i da se pri usporedbi rezultata između grupa i između rezultata ispita pokazala statistički značajna razlika za algoritam ponavljanja potrebno je dodatno analizirati rezultate takvih zadataka. To su bili zadaci od Z15 do Z19, uz dodatak od devet zadataka otvorenog tipa u završnom ispitu u Pythonu. Zajednički zadaci iz grupe algoritma ponavljanja završnih ispita su prikazani u tablici 4.69.

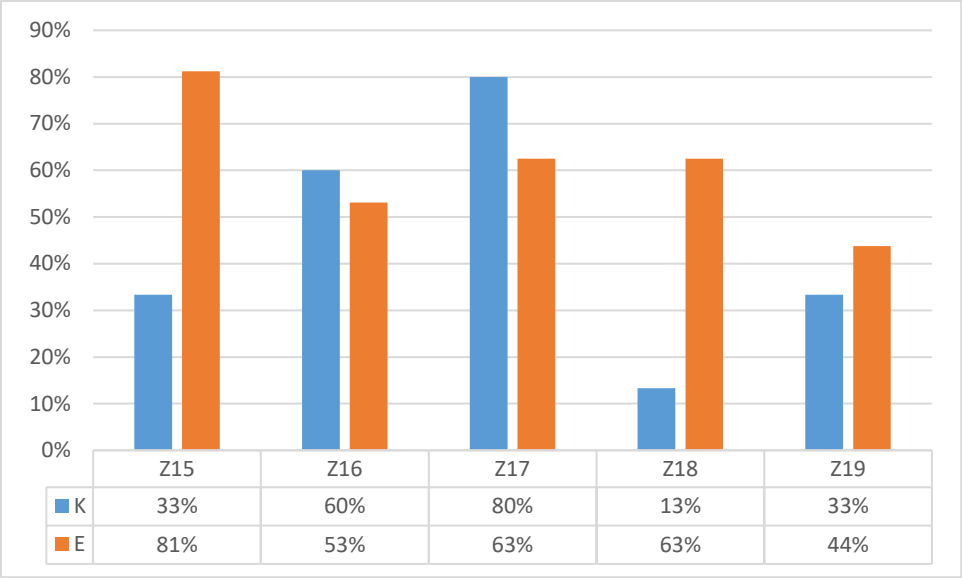
Tablica 4.69 - Usporedba zadataka iz grupe algoritma ponavljanja

	ZI-Micro:bit	ZI-Python
Z15		`for i in range (0,6):` `  print (i)`
Z16		`for i in range (0,6):` `  print (i+1)`
Z17		`for i in range (0,6):` `  print ('+')`

Z18		`zbroj = 0`
		`for i in range (0,6):`
		`  zbroj = i`
		`print (zbroj)`

Z19		`zbroj = 0`
		`for i in range (0,6):`
		`  zbroj = zbroj + i`
		`print (zbroj)`

Na slici 4.10 su prikazane usporedbe rezultata zadataka iz kategorije algoritma ponavljanja po skupinama.



Slika 4.10 - Usporedba rezultata po zadacima iz algoritma ponavljanja po skupinama

Vidljivo je da se najveća razlika nalazi za zadatke Z15 i Z18 gdje se i pokazala statistički značajna razlika između skupina. Učenici eksperimentalne skupine su bili uspješniji u rješavanju navedenih zadataka. U tablici 4.70 je vidljivo da se u zadacima Z15 i Z18 tražio ispis vrijednosti kontrolne varijable. U zadacima Z16 i Z17 se tražio ispit znakova što su bolje, ali ne statistički značajno riješili učenici iz eksperimentalne skupine. Razlog je vjerojatno jasna razlika naredbi *Show number* i *Show string* u Micro:bitu, dok se u Pythonu za ispis svih tipova podataka koristi naredba *print* gdje se za ispis stringa koriste navodnici a za ispis vrijednosti varijable se ispisuje samo naziv varijable. Zbog statistički značajne razlike među grupama u zadacima Z15 i Z18 dodatno će se analizirati odgovori za navedene zadatke u završnom ispita Python. Kako bi se utvrdile najčešće pogreške koristit će se hi-kvadrat test.

4.4.7.1 Analiza zadatka 15

Zadatak Z15 je bio prvi zadatak iz grupe zadataka za algoritma grananja. Zadatak je prikazan u tablici 4.70.

Tablica 4.70 - Zadatak 15

	Python	**Odgovori**
Z15	`for i in range (0,6):` `  print (i)`	a) 0 1 2 3 4 5 b) 0 1 2 3 4 5 6 c) i i i i i d) 1 2 3 4 5 6 e) 1 2 3 4 5

Rezultat hi-kvadrat testa je pokazao da postoji statistički značajna razlika između odgovora i skupine ($\chi^2(10) = 23.756$, $p=0.000$). Distribucija odgovora završnog ispita po skupinama je prikazana u tablici 4.71.

Tablica 4.71 - Distribucija odgovora zadatka 15

		a	b	c	d	e	Ukupno
K	Broj	**5**	6	0	4	0	15
	Očekivano	**9.9**	2.2	.6	1.3	1.0	15
E	Broj	**26**	1	2	0	3	32
	Očekivano	**21.1**	4.8	1.4	2.7	2.0	32
Ukupno	Broj	**31**	7	2	4	3	47

Točan odgovor zadatka 15 je bio pod *a)* što je točno odgovorilo 26 od 47 učenika iz eksperimentalne skupine i 5 od 15 učenika iz kontrolne skupine. Ostali odgovori iz eksperimentalne skupine su raspršeni i zastupljeni manjim brojem što ne upućuje na postojanje pogrešnih predodžbi. Iz kontrolne skupine 6 učenika je odabralo odgovor *b)* što u postotku čini 40% učenika skupine. Ovoliko netočnih odgovora upućuje na pogrešno poimanje petlje u Pythonu. U ovom slučaju odgovor b) je obuhvaćao i završnu vrijednost petlje što bi za Micro:bit zadatak bilo točno, ali ne i za Python. Već je spomenuta specifičnost petlje u Pythonu gdje se petlja izvršava samo do završne vrijednosti petlje, ali ne poprima i završnu vrijednost. Slična je pojava i s odgovorom d) kojeg je izabralo 4 od 15 učenika kontrolne skupine. Ovaj odgovor su odabrali učenici koji su prepoznali da se petlja izvršava pet puta sa zadanim vrijednostima, ali su početnu vrijednost uvećali za jedan. Vjerojatno je da je na ovakve odgovore utjecao upravo tretman kojem su učenici bili izloženi. Eksplicitno povezivanje Micro:bit i Python petlje je očito pomoglo učenicima u razumijevanju samog pojma algoritma ponavljanja pa su usvojili i razlike u sintaksi između dva jezika. Učenici iz eksperimentalne skupine nisu bili izloženi direktnom povezivanju Micro:bit i Python programa te nisu uspjeli sami uočiti razlike u sintaksi jezika pa su zapamtili zadnju koju su radili, onu u Micro:bitu. Zbog toga nisu više bili sigurni u početne i završne vrijednosti kontrolne varijable *i* u *for* petlji Pythonu. Ovakvi rezultati potvrđuju da su učenici ostvarili bolji posredovani prijenos znanja metodom „premošćivanja“ (eng. bridging).

4.4.7.2 Analiza zadatka 18

Učenici su tijekom obrade algoritma ponavljanja rješavali zadatke koji uključuju kumulativno zbrajanje. Zadatak Z18 je zadatak u kojem se ispitivalo utječe li naziv varijable na samu ulogu varijable u programu. Zadatak je prikazan u tablici 4.72.

Tablica 4.72 - Zadatak 18

	Python	**Odgovori**

Z18	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = i`
	`print (zbroj)`
		a) 0
		b) 5
		c) 6
		d) 15
		e) zbroj

Rezultat hi-kvadrat testa je pokazao da postoji statistički značajna razlika između odgovora i skupine ($\chi^2(10) = 12.938$, $p=0.012$). Distribucija odgovora završnog ispita po skupinama je prikazana u tablici 4.73.

Tablica 4.73 - Distribucija odgovora zadatka 18

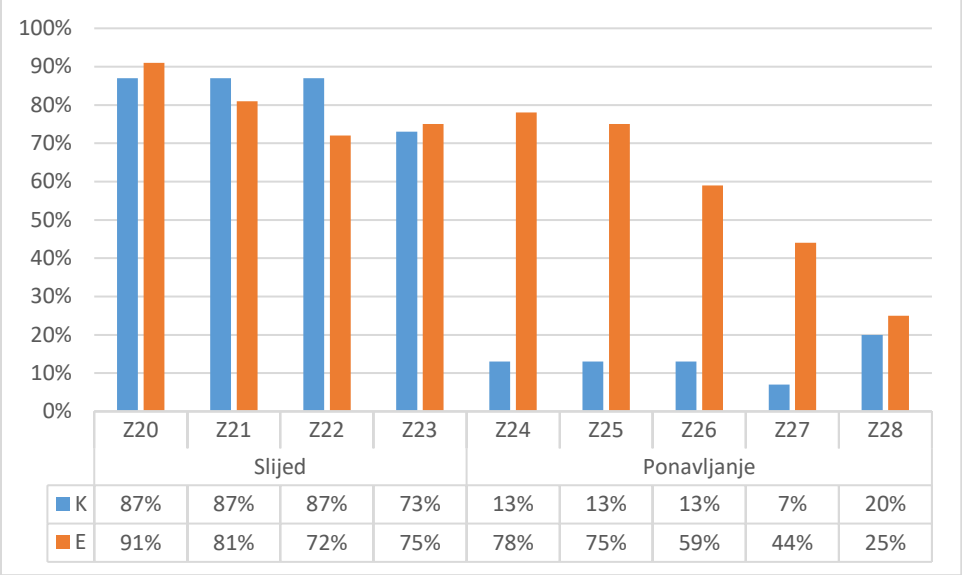
		Bez odgovora	a	**b**	c	d	e	Ukupno
K	Broj	0		**2**	10	2	1	15
	Očekivano	.3		**7.0**	5.1	1.9	0.6	15
E	Broj	1		**20**	6	4	1	32
	Očekivano	.7		**15.0**	10.9	4.1	1.4	32
Ukupno	Broj	1		**2**	10	2	1	15

Točan odgovor ovog zadatka je bio pod b) što je odabralo 15 od ukupno 32 učenika iz eksperimentalne skupine što je nešto manje od pola dok je samo dvoje od 15 učenika iz kontrolne skupine odabralo točan odgovor. Najzastupljeniji netočan odgovor u obje skupine je bio odgovor c) što je odabralo čak 10 od 15 učenika iz kontrolne i 6 od 32 učenika iz eksperimentalne skupine. Većina učenika je ipak prepoznala da varijabla *zbroj* ne sadrži zbroj brojeva na što upućuje naziv varijable, već zadnja vrijednost kontrolne varijable *i*. Razlog za ovako veliku zastupljenost istog netočnog odgovora kontrolne skupine je vjerojatno isti kao i u prethodno opisanom zadatku. Učenici kojima se metodom „prigrljavanja“ nisu direktno povezivali zadaci u Micro:bitu i Pythonu su se više usmjerili na sintaksu konkretnog programskog jezika, a ne na pozadinski koncept pa su, zbog različite strukture *for* petlje u dva korištena jezika, prihvatili onaj koji su zadnji učili.

4.4.7.3 Analiza zadataka otvorenog tipa

Zadaci otvorenog tipa su strukturirani na način da se utvrdi razina usvojenosti algoritama slijeda i ponavljanja. U tu svrhu su se dodatno analizirali rezultati iz navedene grupe zadataka. Iz ZOT

grupe zadataka, prema rezultatima Mann-Whitney testa nema statistički značajne razlike u rezultatima između skupina za koncept slijeda (U=227.50, p=0.739), ali ima za koncept petlje (U=65.0, p=0.00). Na slici 4.11 su prikazani rezultati po zadacima.



Slika 4.11 - Rezultati zadataka otvorenog tipa u ZI-P

Iako su po ostalim rezultatima eksperimentalne (E) i kontrolne (K) skupine ujednačene, očita je razlika u rezultatima sa zadacima koji ispituju petlju, odnosno algoritam ponavljanja. Iz rezultata je vidljivo da je i u ovom slučaju eksperimentalna skupina uspješnije riješila zadatke otvorenog tipa u završnom ispitu u Pythonu što je u skladu s rezultatima ostalih ispita.

4.4.8 Stavovi učenika

Na kraju obrade nastavne cjeline programiranja, odnosno nakon korištenja oba programska jezika, a prije dva završna ispita znanja učenici su ispunjavali anketu o stavovima prema programiranju. Anketa je bila dobrovoljna, ali ne i anonimna. Učenici su poticani na ispunjavanje ankete plusom za sudjelovanje. Od učenika su se tražili iskreni odgovori.

Kako bi se utvrdilo postoje li razlike u stavovima između eksperimentalne i kontrolne skupine korišten je Mann-Whitney test. Rezultat testa nije potvrdio statistički značajnu razliku u stavovima između grupa. Rezultati su prikazani u tablici 4.74.

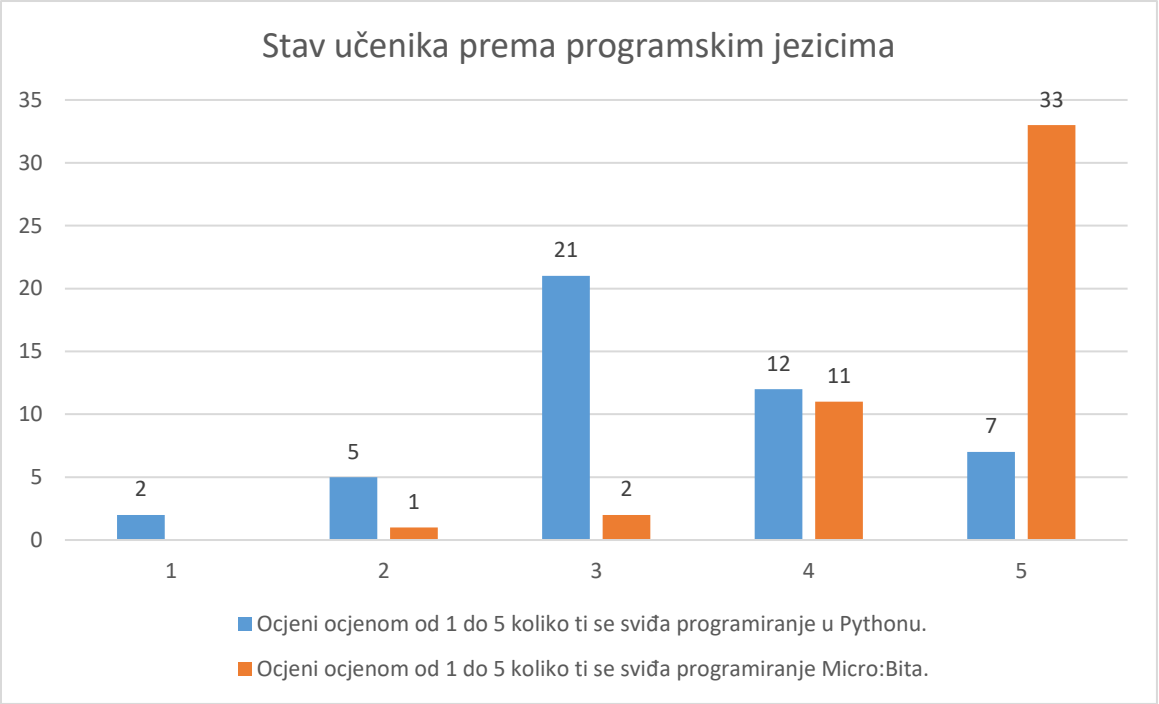
Kako statistički značajna razlika među grupama nije utvrđena svi odgovori se mogu analizirati kao jedne ujednačene grupe.

Tablica 4.74 - Rezultati Mann-Whitneyevog testa za anketu

		Mann-Whitney U	Z	p
P1	Koji ti se programski jezik više sviđa? 1) Micro:Bit 2) Python	200.500	-1.380	0.168
P2	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Python.	231.000	-0.218	0.828
P3	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje Micro:Bita.	223.000	-0.484	0.628
P4	Kojim bi programskim jezikom učio/la programiranje iduće školske godine? 1) Python 2) Micro:Bit	226.000	-0.598	0.550
P5	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje igara (na primjer, kamen-škare-papir) na Micro:Bitu.	220.500	-0.649	0.516
P6	Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje algoritama na Micro:Bitu (zadaci slični onima u Pythonu).	219.500	-0.493	0.622
P7	Što ti se više sviđa: 1) programiranje igre u Micro:bitu 2) Programiranje matematičkih zadataka u Micro:bitu 3) Programiranje matematičkih zadataka u Pythonu	227.500	-0.365	0.715

4.4.8.1 Stavovi učenika prema programskom jeziku

Dva pitanja su se odnosila na ocjenu programskih jezika prema Likertovoj skali od 1 (ne sviđa mi se) do 5 (jako mi se sviđa). Frekvencije odgovora su prikazane na slici 4.12.

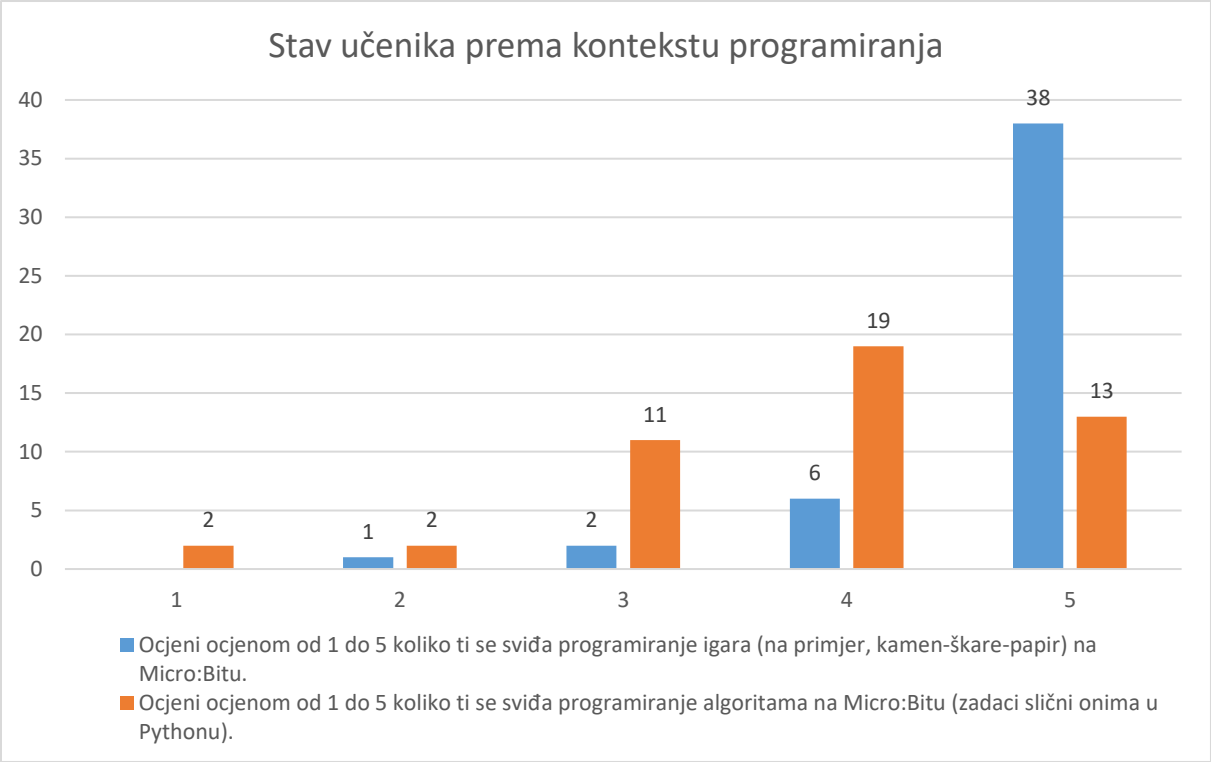


Slika 4.12 - Stavovi učenika prema programskom jeziku

Distribucija učeničkih odgovora sa slike 4.12 implicira da učenici imaju pozitivniji stav prema blokovskom jeziku Micro:bitu nego prema tekstualnom programskom jeziku Pythonu. Ovakvi rezultati u skladu su s rezultatima prve faze istraživanja u kojoj su učenici radije birali blokovski programski jezik nego tekstualni.

4.4.8.2 Stavovi učenika prema kontekstu programiranja

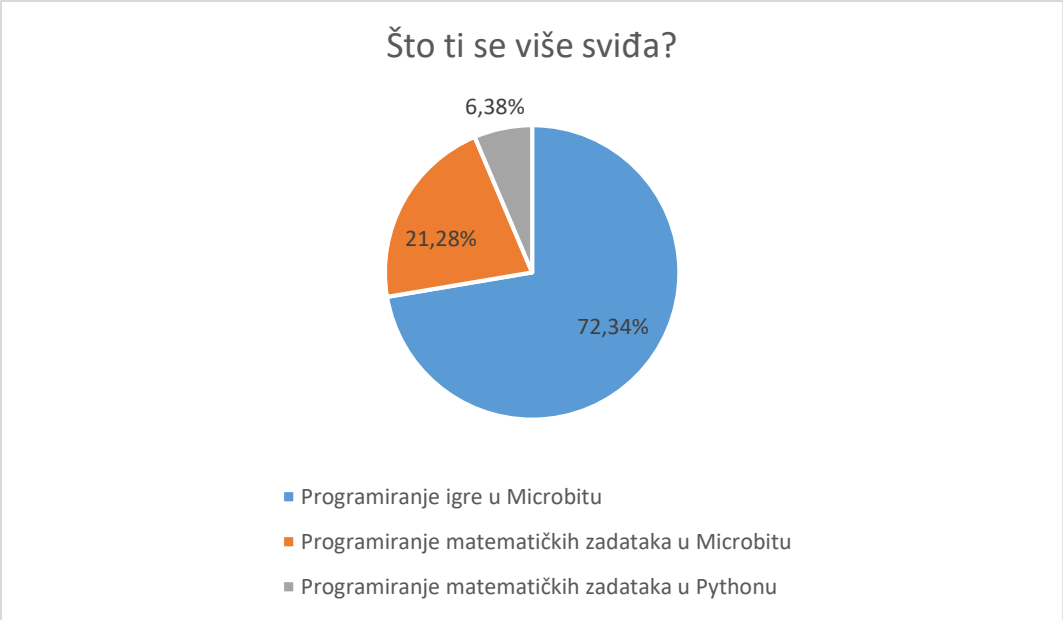
Dva pitanja odnosila su se na ocjenu konteksta programiranja prema Likertovoj skali od 1 (ne sviđa mi se) do 5 (jako mi se sviđa). Jedan kontekst je „tradicionalni“ pristup koji uključuje rješavanje matematičkih zadataka, dok je drugi programiranje igara u blokovskom programskom jeziku. Zbog kompleksnosti sintakse tekstualnih programskih jezika i složenijih programskih struktura koje su potrebne za izradu igara, na razini osnovne škole ne može se provesti programiranje igara u tekstualnom programskom jeziku. Frekvencije odgovora su prikazane na slici 4.13.



Slika 4.13 - Stavovi učenika prema kontekstu programiranja

Prema odgovorima učenika može se uočiti kako učenici preferiraju kontekst programiranja igara u odnosu na „tradicionalne“ zadatke.

Osim ocjene Likertovom skalom učenici su na jedno pitanje mogli odabrati kakav kontekst i programski jezik preferiraju. Rezultati su prikazani na slici 4.14

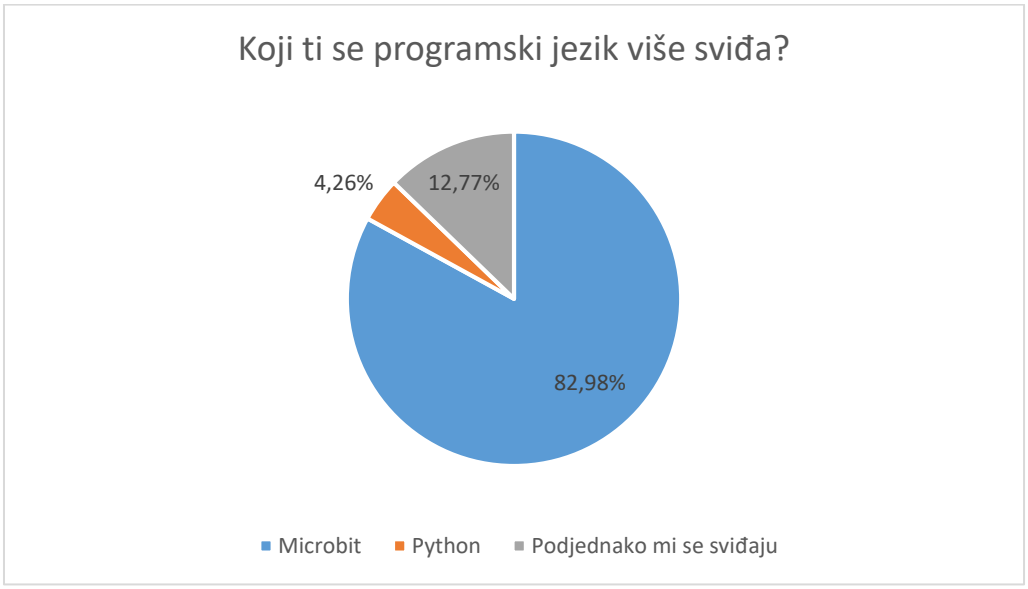


Slika 4.14 - Preferirani kontekst programiranja

Najveći broj učenika (n=34) je odabrao programiranje igre u Micro:bitu. Najmanji broj učenika (n=3) je odabrao „tradicionalan“ pristup u obliku rješavanja matematičkih problema u Pythonu dok je 10 učenika odabralo korištenje „tradicionalnog“ konteksta programiranja, ali u blokovskom programskom jeziku Micro:bit.

4.4.8.3 Preferirani programski jezik

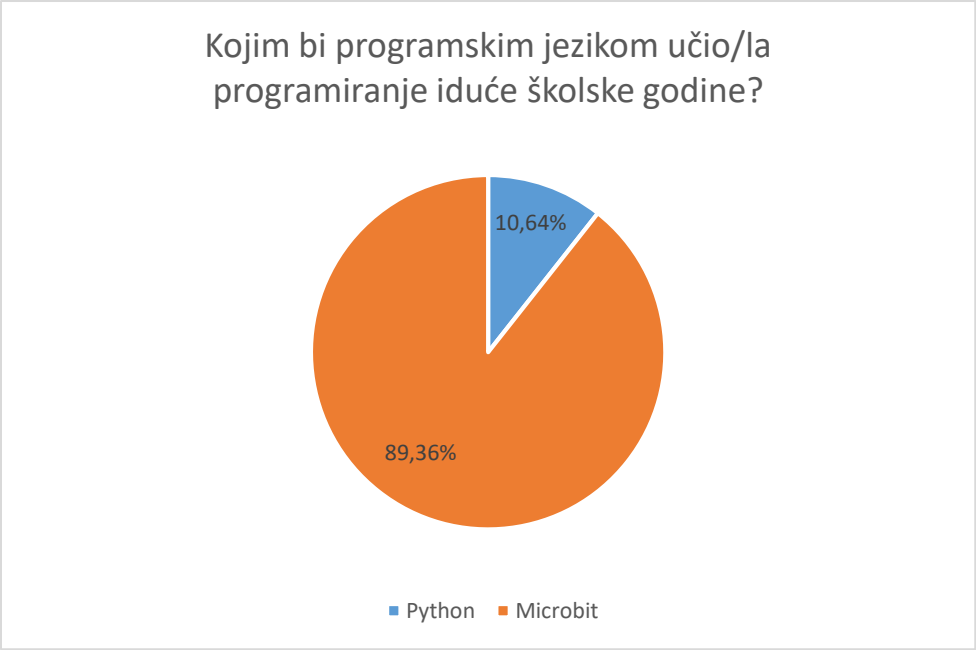
Učenici su na jedno pitanje mogli odabrati koji im se programski jezik više sviđa. Frekvencije odgovora su prikazane na slici 4.15.



Slika 4.15 - Preferirani programski jezik

Gotovo 83% učenika (n=39) je odabralo Micro:bit kao programski jezik koji ima se više sviđa. Najmanji broj učenika je odabrao Python (n=2), dok se šestoro učenika podjednako sviđaju oba jezika.

Učenici su mogli birati koji bi programski jezik koristili u nastavi sljedeće školske godine. Rezultati su prikazani na slici 4.16.



Slika 4.16 - Preferirani programski jezik za iduću školsku godinu

Gotovo 90% učenika (n=42) bi iduće školske godine radije koristili Micro:bit kao programski jezik, a samo 5 učenika više preferira Python kao željeni programski jezik.

4.4.8.4 Komentari

Učenici su u anketi imali i dva pitanja otvorenog tipa za komentare za Micro:bit i za usporedbu Micro:bita i Pythona.

Od ukupno 47 učenika 17 učenika je napisali komentar u vezi Micro:bita od čega ih je čak 14 pozitivno. Komentari, prikazani u izvornom obliku kako su ih učenici napisali, su:

- „*micro bit je zanimljiviji i lakši*“
- „*Micro:bit mi se sviđa.“*
- „*ODLIČAN JE*“
- „*Lakše je razumjet što se događa u programu.*“
- *„Micro bit je zabavan i lako ga zavolis i dobro je sta se mogu radit igre*“
- *„Programiranje u micro bitu mi se svidjelo.“*
- *„Jako mise sviđaju igre i mislim da je to super način da shvatimo kako programiranje funkcionira.“*
- *„Micro:bit mi se jako svidio i preporučio bih da se uvede u program i drugim generacijama“*
- *„Meni se jako sviđa programiranje na Micro:bitu jer je zanimljiv i volim učiti nove stvari“*

- *„super i prakticno“*
- *„Micro:bit mi se više sviđa“*
- *„jako mi se svidjelo“*
- *„Jako mi se sviđa“*

Tri komentara nisu bila s potpuno pozitivnim stavom prema Micro:bitu:

- *„Programiranje u Micro:bitu je jednostavnije, a baš zbog toga mi se više sviđa Python jer je kompliciranije.“*
- *„Sve je OK ali nema puno igara koje se može s micro bitom igrati“*
- *„Dobro je, ali ne može se puno toga raditi.“*

Ovo su komentari učenika koji preferiraju programski jezik Python. Već je zabilježeno (Lewis, 2010) kako učenici koji su već programirali u tekstualnom programskom jeziku i imaju veće sposobnosti rješavanja problema brzo „prerastu“ blokovske programske jezike pa su im tekstualni programski jezici izazovniji. U ovom slučaju u pitanju su upravo takvi učenici,

Jedan komentar nije bio potpuno pozitivan:

- *„svida mi se programiranje i jako je poucno al ponekad dosadno“*

U ovom slučaju u pitanju je učenik koji nema visoke sposobnosti rješavanja problema i ostvario je lošije rezultate na ispitima znanja. Ovakvim učenicima, naročito kad je u pitanju tekstualni programski jezik, programiranje može biti izvor frustracije jer teže savladavaju gradivo pri čemu im brzo postaje dosadno.

Drugo pitanje otvorenog tipa je bilo za komentar o usporedbi Micro:bita i Pythona. Od ukupno četrdeset sedam učenika trienaes ih je napisalo komentar u vezi Micro:bita od čega je deset pozitivnih komentara. Komentari, prikazani u izvornom obliku kako su ih učenici napisali, su:

- *„Ja mislim da je Micro:bit dosta zabavniji od Pythona i meni je zabavnije raditi na Micro:bitu. Meni je lakše shvatiti zadatke na Micro:bitu“*
- *„Sviđa mi se i python, ali micro:bit je bolji.“*
- *„Programiranje u Micro:Bitu je zanimljivije od Pythona.“*
- *„Python je ozbiljni i zato se nekima ne svida“*
- *„Super je što nije velika razlika između njih a Micro:bit je dosta zabavniji.“*
- *„Micro:bit mi je puno draži od Paythona“*
- *„micro bit mi je u usporedbi pythonom puno bolji i zanimljiviji.“*

- *„Micro:bit mi je bolji i jednostavniji“*
- *„micro bit mi se više svidjeo“*
- *„bolji je Micro bit“*

Iz komentara učenika je vidljivo kako im je Micro:bit draži od Pythona. Od učenika koji preferiraju Micro:bit u odnosu na Python većina spada u „srednju trećinu“ po sposobnostima. Takvim učenicima odgovara blokovski programski jezik jer lakše savladavaju koncepte programiranja.

Slično kao i u prethodnom pitanju, dva učenika naglašavaju kako im je Python draži. Opet su u pitanju isti učenici koji spadaju u „gornju trećinu“, odnosno učenici visokih sposobnosti rješavanja problema koji s lakoćom savladavaju programske koncepte pa im je tekstualni programski jezik izazovniji.

- *„Meni je Python bolji jer ima više zadataka oko kojih se može mozgati,a u micro bitu može se sve u jednom danu“*
- *„Python mi je ipak malo bolji jer je veći užitak uspjeha kada programiram nešto i draže mi je tipkanje nego vučenje „kockica".“*

Jedan učenik je izrazio nezadovoljstvo programiranjem. U pitanju je isti učenik koji je i na prethodnom pitanju naglasio kako je programiranje dosadno. To je pripadnik „donje trećine“ učenika, slabijih sposobnosti rješavanja problema kojem programiranje može biti vrlo zahtjevno. Takvim učenicima su tekstualni programski jezici zahtjevniji zbog više razine apstrakcije potrebne za savladavanje gradiva što je vidljivo u komentaru:

- *„zeljela bi da ima manje programiranja ali pythonu je puno teze ucenje“*

4.4.9 Ograničenja istraživanja, rasprava i zaključak

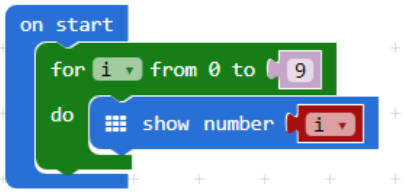
Rezultati ovog istraživanja pokazali su kako se korištenjem blokovskog programskog jezika može poboljšati usvajanje i razumijevanje istih pojmova u tekstualnom programskom jeziku koji je učenicima apstraktniji te samim time i zahtjevniji. Prema rezultatima ovog istraživanja se može prihvatiti H1 i zaključiti kako se korištenjem blokovskog programskog jezika može ostvariti posredovani transfer u kontekst tekstualnog programskog jezika.

Kad je riječ o metodama posredovanog transfera, prema rezultatima istraživanja, pokazalo se da korištenjem metode „premošćivanja“ u nastavi učenici postižu bolji uspjeh u željenom kontekstu. Usporedba rezultata između dviju skupina se pokazala statistički značajna kad je u pitanju usvajanje algoritma ponavljanja koji i je apstraktniji u odnosu na slijed i grananje.

Učenici kojima nisu direktno povezani zadaci iz dva konteksta su se vjerojatno više usmjerili na sintaksu programskog jezika umjesto na za razumijevanje uvedenih pojmova što se odrazilo u oba konteksta.

Eksperimentalna skupina je tijekom rada u Micro:bitu, metodom „premošćivanja“, bila izložena direktnoj usporedbi riješenih zadataka u programskim jezicima Micro:bit i Python na način da su se rješenja zadataka u Micro:bitu direktno povezivala i uspoređivala s analognim rješenjem u Pythonu. Primjer je prikazan u tablici 4.75.

Tablica 4.75 - Usporedba Micro:bit i Python programa

Zadatak *štoperica*: Napravite program na Micro:bitu koji će ispisivati brojeve od 0 do 9.	
Micro:bit	Python
	`for i in range (0,10):` `  print (i)`

U kontrolnoj skupini su se radili isti zadaci, ali samo u Micro:bitu bez eksplicitnog povezivanja s Pythonom. Struktura *for* petlje je u Pythonu specifična u odnosu na druge programske jezike jer završna vrijednost petlje nije obuhvaćena. Na primjer, ako želimo ispisati brojeve do 9 u Pythonu završna vrijednost petlje mora biti veća za jedan broj (10) što nije slučaj u Micro:bitu. Stoga, prema rezultatima istraživanja se može prihvatiti hipoteza H2 te zaključiti da je metoda „premošćivanja“ učinkovitija u razumijevanju apstraktnih programskih koncepata kao što je algoritam ponavljanja, odnosno petlja. Ovo se može objasniti većim razinom apstrakcije koja je učenicima potrebna pri korištenju metode „prigrljivanja“, za razliku od metode „premošćivanja“ gdje se pružanjem direktnih analogija učenicima olakšava posredovani prijenos između dva okruženja.

Stavovi učenika su uglavnom na strani Micro:bita, odnosno radije ga biraju kao programski jezik u odnosu na Python. Uočeno je da mali broj učenika preferira Python. Tu spadaju učenici koji imaju bolje sposobnosti rješavanja problema pa blokovske programske jezike brzo „prerastu“. S druge strane učenici koji imaju lošije sposobnosti rješavanja problema imaju lošiji stav prema programiraju općenito jer brzo postanu frustrirani što je posebno naglašeno u tekstualnim programskim jezicima kao što je Python.

5 ZAKLJUČAK

Programiranje je sastavni dio računalne znanosti koja će, prema procjenama, biti potrebna za većinu poslova budućnosti. U cijelom svijetu je prepoznata važnost upoznavanja učenika s osnovnim konceptima programiranja još od najranije dobi. Kako je programiranje apstraktno te izuzetno zahtjevno, pogotovo početnicima osnovnoškolcima, tako se u posljednja dva desetljeća pokušavaju pronaći drugačiji načini poučavanja programiranja. Tradicionalan način programiranja u kojem se programiranje poučava uglavnom u tekstualnim programskim jezicima, te u kontekstu rješavanja matematičkih problema, očito nije uspješan što se očituje u sve većem padu zainteresiranosti za učenje programiranja i računalne znanosti općenito. Postoji više problema u takvom pristupu poučavanja programiranja. Prvi je sigurno problem sintakse tekstualnih programskih jezika kod kojih se očekuje da svaki program mora biti sintaktički potpuno ispravno napisan kako bi se izvršio. Time se fokus s rješavanja problema, koji bi trebao biti prvi i osnovni korak, pomiče na ovladavanje sintaksom programskog jezika čime se početnici u programiranju suočavaju s višestrukim problemima. Drugi problem je kontekst programiranja gdje je potrebno uvažiti činjenicu kako rješavanje matematičkih zadataka današnjim učenicima nije motivirajuće. Kada se navedeni problemi razmatraju u kontekstu osnovnoškolaca tada su isti još naglašeniji. Učenici osnovnih škola uglavnom još nisu stigli do faze formalnih operacija, odnosno apstraktnog razmišljanja te je za takvu populaciju programiranje još zahtjevnije.

Prvi pokušaji promjene tradicionalnog pristupa poučavanja programiranja rezultirali su novom generacijom programskih jezika dizajniranih posebno za početnike u programiranju. To su vizualno-blokovski programski jezici kao što su: Scratch, Alice, Greenfoot i drugi. Prva i osnovna prednost vizualno-blokovskih jezika je uklanjanje problema sintakse jer su u takvim jezicima naredbe prikazane u obliku slagalica čime se značajno smanjuje ili potpuno eliminira problem sintakse pa se omogućuje fokusiranje na rješavanje problema. Druga prednost je mogućnost pomicanja konteksta programiranja s rješavanja matematičkih problema prema programiranju računalnih igara, animacija, simulacija i slično. Navedene aktivnosti su one kojima su današnji učenici okruženi te im je samim time motivacija za učenje programiranja veća. Treća prednost objedinjuje prethodne dvije, a to je omogućavanje pristupa programiranja od konkretnog prema apstraktnom. Kako se kod takvih programskih jezika programira ponašanje likova u nekom mikrosvijetu, tako je rezultat naredbi vidljiv. Na taj način je konkretno vidljiv rezultat izvršavanja programa pa su razumijevanje rada programa i otklanjanje grešaka jednostavniji.

Postoji veći broj istraživanja koji ispituju utjecaj vizualno-blokovskih jezika na usvajanje koncepata programiranja, ali je veći dio njih proveden u neformalnom okruženju, u kampovima, na tečajevima, ljetnim školama i slično. Od istraživanja u formalnom, školskom okruženju veći dio je proveden na preddiplomskoj ili diplomskoj razini dok se manji broj istraživanja odnosi na osnovnoškolsku razini. Istraživanja u osnovnoj školi puno je teže provesti u odnosu na višu razinu obrazovanja i neformalna okruženja. Jedan od razloga je ograničenje nastavnim planom i programom. U mnogim državama se programiranje ne odvija kroz predmete osnovnih škola. Programiranje je sastavni dio kurikula Informatike u osnovnim školama u Republici Hrvatskoj. Informatika je do školske godine 2017./2018. bila izborna za uzrast od petog do osmog razreda, dok je od školske godine 2018./2019. obavezna za učenike petih i šestih razreda. Kod istraživanja u osnovnim školama postoji ograničenje nastavnog plana i programa te načina provedbe samog istraživanja. Kako ispitivani koncepti moraju biti sastavni dio nastavnog plana i programa tako je broj ispitivanih koncepata ograničen. Osim toga postoji i vremensko ograničenje mogućnosti provedbe kao i odabir nastavnika i škola u kojima bi se istraživanje trebalo provoditi. Zbog svega navedenog teško je provesti istraživanje na velikom broju škola, razreda, učenika te ispitati veći broj koncepata uz istovremeno osiguravanje valjanosti eksperimenta. Zbog navedenih problema pri istraživanju u ovoj disertaciji istraživanje je provedeno u četiri faze tijekom četiri školske godine u kojem je svaka faza zapravo pojedinačno istraživanje. U svakoj od faza istraživanja uklonjeno je neko od ograničenja ostalih istraživanja. Ovakvom provedbom se dobila triangulacija istraživanja čime se pridonosi valjanosti cjelokupnog istraživanja.

Iz navedenih problema u tradicionalnom pristupu poučavanja programiranja proizašli su osnovni ciljevi istraživanja: i) ispitati utjecaj programskog jezika i konteksta rješavanja problema na uspjeh u rješavanju problema programiranjem; ii) ispitati utjecaj izbora programskog jezika i konteksta rješavanja problema na motivaciju za programiranje. Sve faze istraživanja su provedene u formalnom, školskom okruženju, tijekom provedbe nastave informatike. Svo gradivo koje se obrađivao dio je nastavnog plana i programa predmeta. Tijekom svake od faza istraživanja učenici nisu uvježbavali tipove zadataka koji su se koristili u istraživanju. Učenici nisu bili ocjenjivani s obzirom da provedeni ispiti nisu služili za ocjenjivanje. Učenici su poticani za pisanje ispita ocjenom izvrstan za dobro riješene ispite ili plusom za sudjelovanje.

Prva faza istraživanja provedena je tijekom školske godine 2013/2014 u jednoj osnovnoj školi u Splitu među dva sedma razreda (n=23) tijekom nastave informatike. U oba razreda učiteljica

je ujedno i istraživačica. Prema rezultatima istraživanja utvrdilo se da su učenici sedmog razreda uspješniji u usvajanju osnovnih koncepata programiranja korištenjem vizualno-blokovskog programskog jezika Scratch programiranjem igara, u odnosu na vizualno-tekstualni programski jezik Logo rješavanjem matematičkih problema. Najveća, i statistički značajna razlika se pokazala na ugniježđenoj petlji gdje su učenici bilo uspješniji u Scratchu nego u Logu. Osim toga pokazalo se da učenici više preferiraju Scratch nego Logo kao željeni programski jezik.

Druga faza istraživanja je provedena tijekom školske godine 2014/2015 u jednoj osnovnoj školi u Splitu među tri peta razreda (n=49) tijekom nastave informatike. U sva tri razreda učiteljica je istraživačica. Prema rezultatima istraživanja pokazalo se da je, s obzirom na sposobnosti rješavanja problema, uspjeh učenika petih razreda osnovne škole veći pri poučavanju programiranja oblikovanjem igara u Scratchu u odnosu na tradicionalan način poučavanja programiranja u tekstualnom programskom jeziku Python proceduralnim pristupom. Osim toga pokazalo se kako učenici imaju bolji stav prema programiranju nakon korištenja Scratcha te da im je motivacija veća kada programiraju igre.

Treća faza istraživanja je provedena tijekom školske godine 2015./2016. u četiri osnovne škole u Splitu među dvadeset razreda uzrasta od petog do osmog razreda (n=312) tijekom nastave informatike. U ovim istraživanju su potvrđeni rezultati prethodna dva jer je i u ovom slučaju utvrđeno da je uspjeh učenika, s obzirom na odabrani programski jezik, veće poučavanjem oblikovanjem igara u vizualnom-blokovskom programskom jeziku Scratch, u odnosu na vizualno-tekstualne programske jezike Logo ili Python (uz korištenje kornjačine grafike). S obzirom da su sudjelovali učenici od petih do osmih razreda dodatno su se analizirali rezultati učenika petih i šestih razreda jer su svi na otprilike istoj razini kognitivnog razvoja. Opet je utvrđeno da je uspjeh učenika petih razreda i šestih razreda (n=207), s obzirom na sposobnost rješavanja problema, veći korištenjem vizualno-blokovskog programskog jeziku Scratch, u odnosu na vizualno-tekstualne programske jezike Logo ili Python (uz korištenje kornjačine grafike).

Četvrta faza istraživanja je provedena tijekom školske godine 2017/2018 u jednoj osnovnoj škole u Splitu među tri šesta razreda (n=47) tijekom nastave informatike. U sva tri razreda učiteljica je upoznata s metodologijom istraživanja te je nastavu provodila prema materijalima pripremljenim u suradnji s istraživačicom. U ovom istraživanju svi učenici su imali predznanje u Pythonu jer su nastavnu cjelinu programiranja obrađivali koristeći Python prema proceduralnom pristupu. Micro:bit se koristio kao blokovski programski jezik za

ususta vljivanje gradiva iz programiranja. Tijekom nastave su se koristile metode „premošćivanja“ i „prigrljivanja“ u svrhu posredovanog prijenosa znanja iz jednog konteksta u drugi. Prema rezultatima istraživanja svi učenici su nakon korištenja vizualno-blokovskog jezika Micro:bit postigli statistički značajno bolje rezultate u Pythonu čime se može zaključiti da se korištenjem vizualno-blokovskog jezika može pospješiti razumijevanje koncepata programiranja u tekstualnom programskom jeziku. Kad su u pitanju metode posredovanog prijenosa znanja metoda „premošćivanja“ se pokazala, statistički značajno uspješnija na konceptu petlje u odnosu na metodu „prigrljivanja“. Osim toga, učenici su bili više motivirani za programiranje u kontekstu programiranja igara u odnosu na „tradicionalni“ kontekst rješavanja matematičkih problema.

Prema rezultatima cjelokupnog istraživanja može se zaključiti sljedeće:

i) Dokazala se opravdanost poučavanja programiranja kod početnika u osnovnoj školi oblikovanjem igara u blokovski orijentiranom, vizualnom programskom, jeziku primjerenom dobi učenika;
ii) Dokazao se utjecaj na razumijevanje koncepta petlje i motivaciju učenika u osnovnoj školi promjenom konteksta učenja s matematičkih problema prema oblikovanju računalnih igara.
iii) Primjerski zadaci, korišteni u istraživanju, primjereni su za korištenje u nastavi Informatike za nastavnu cjelinu programiranje u osnovnoj školi kao metodički materijali .

Programiranje je vještina koju bi svi učenici trebali imati priliku učiti. Nije cilj sve učenike učiniti programerima, već ih upoznati s računalnim razmišljanjem koje je primjenjivo u gotovo svim aspektima života. Zbog toga bi početno učenje programiranja u osnovnoj školi trebalo biti pristupačno svim učenicima, pri čemu je potrebno voditi računa o primjerenosti programskog jezika fazi kognitivnog razvoja učenika. Poučavanje programiranja uporabom vizualno-blokovskih jezika u kontekstu programiranja igara pokazalo se učinkovitim za istraživanu populaciju. Ovakvim načinom uvođenja i poučavanja osnovnih algoritama programiranja učenicima se olakšava prijelaz u svijet „pravog“ programiranja u tekstualnom programskom jeziku kao ciljnom programskom jeziku.

LITERATURA

Armoni, M., Meerbaum-Salant, O., & Ben-Ari, M. (2015). From Scratch to "Real" Programming. *ACM Transactions on Computing Education*, *14*(4), 1–15. https://doi.org/10.1145/2677087

Armstrong, T., & Association for Supervision and Curriculum Development. (2009). *Multiple intelligences in the classroom*. Alexandria, Va: ASCD.

Ash, K. (2011). Digital gaming goes academic. *Education Week*, *30*(25), 24–28.

Badawy, A.-H. A., Schmitt, K. R. B., Kramer, S. R., Hrapczynski, K. M., Larsen, E. A., Andrew, A. A., … Benson, S. A. (2013). Expectations of computing and other {STEM} students: A comparison for different Class Levels, or (CSE ≠ STEM - CSE). In *2013 {IEEE} Frontiers in Education Conference ({FIE})*. Institute of Electrical {&} Electronics Engineers ({IEEE}). https://doi.org/10.1109/fie.2013.6685120

Bailey, M. W. (2005). IronCode: Think-Twice, Code-Once Programming. In *SIGCSE '05* (pp. 181–185). https://doi.org/10.1145/1047344.1047412

Ball, T., Protzenko, J., Bishop, J., Moskal, M., de Halleux, J., Braun, M., … Riley, C. (2016). Microsoft touch develop and the BBC micro: bit. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)* (pp. 637–640).

Bayliss, J. D. (2009). Using games in introductory courses: tips from the trenches. *ACM SIGCSE Bulletin*, *41*(1), 337–341. https://doi.org/10.1145/1539024.1508989

Ben-Ari, M. (Moti). (2011). Loop Constructs in scratch. *ACM Inroads*, *2*(1), 27. https://doi.org/10.1145/1929887.1929900

Ben-Bassat Levy, R., Ben-Ari, M., & Uronen, P. A. (2003). The Jeliot 2000 program animation system. *Computers & Education*, *40*(1), 1–15. https://doi.org/10.1016/S0360-1315(02)00076-3

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145–157.

Böhm, C., & Jacopini, G. (1966). Flow diagrams, turing machines and languages with only two

formation rules. *Communications of the ACM*, *9*(5), 366–371. https://doi.org/10.1145/355592.365646

Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In *Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada* (Vol. 1, p. 25).

Brown, J. S. (2000). Growing up: Digital: How the web changes work, education, and the ways people learn. *Change: The Magazine of Higher Learning*, *32*(2), 11–20.

Bruner, J. S. (1960). The Process of Education. In *The Process of Education* (pp. 12–13). https://doi.org/10.1017/CBO9781107415324.004

Bruner, J. S. (1961). The act of discovery. *Harvard Educational Review*.

Bruner, J. S. (1966). *Toward a theory of instruction* (Vol. 59). Harvard University Press.

Brusilovsky, P., Calabrese, E., Hvorecky, J., Kouchnirenko, A., & Miller, P. (1997). Mini - languages: A Way to Learn Programming Principles. *Education and Information Technologies*, *2*(1), 65–83. https://doi.org/http://dx.doi.org/10.1023/A:1018636507883

Bubica, N., Mladenović, M., & Boljat, I. (2013). Programming as a tool for the development of abstract thinking. In *" Dohvati znanje"-15. CARNetova korisnička konferencija-CUC 2013*-.

Casey, P. J. (1997). Computer Programming: A medium for teaching problem solving. *Computers in the Schools*, *13*(1–2), 41–51. https://doi.org/10.1300/J025v13n01_05

Cohen, L., Manion, L., & Morrison, K. (2011). *Research Methods in Education*. Oxford, UK: Routledge. https://doi.org/10.4324/9780203720967

Cohen, L., Manion, L., & Morrison, K. (2013). *Research methods in education*. Routledge.

Council, N. R., & others. (1999). *Being fluent with information technology*. National Academies Press.

Curtis, D. D., & Lawson, M. J. (2002). Computer adventure games as problem-solving environments.

D'Mello, S., & Graesser, A. (2012). Dynamics of affective states during complex learning. *Learning and Instruction*, *22*(2), 145–157. https://doi.org/10.1016/J.LEARNINSTRUC.2011.10.001

Dann, W., & Cooper, S. (2009a). Alice 3: Concrete to Abstract. *Communications of the ACM*, *52*(8), 27. https://doi.org/10.1145/1536616.1536628

Dann, W., & Cooper, S. (2009b). Education: Alice 3: Concrete to Abstract. *Commun. ACM*, *52*(8), 27–29. https://doi.org/10.1145/1536616.1536628

Dann, W., Cosgrove, D., Slater, D., & Culyba, D. (2012a). Mediated Transfer : Alice 3 to Java. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 141–146. https://doi.org/10.1145/2157136.2157180

Dann, W., Cosgrove, D., Slater, D., & Culyba, D. (2012b). Mediated Transfer : Alice 3 to Java. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, 141–146. https://doi.org/10.1145/2157136.2157180

Dann, W., Cosgrove, D., Slater, D., Culyba, D., & Cooper, S. (2012). Mediated Transfer: Alice 3 to Java. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 141–146). New York, NY, USA: ACM. https://doi.org/10.1145/2157136.2157180

Denning, P. J. (2009a). The profession of IT: Beyond computational thinking. *Communications of the ACM*, *52*(6), 28. https://doi.org/10.1145/1516046.1516054

Denning, P. J. (2009b). The Profession of IT: Beyond Computational Thinking. *Commun. ACM*, *52*(6), 28–30. https://doi.org/10.1145/1516046.1516054

Denning, P. J. (2010). The great principles of computing. *American Scientist*, *98*(5), 369–372. https://doi.org/10.1145/948383.948400

Denning, P. J. (2013). The science in computer science. *Communications of the ACM*, *56*(5), 35. https://doi.org/10.1145/2447976.2447988

Denning, P. J., & McGettrick, A. (2005). Recentering computer science. *Communications of the ACM*, *48*(11), 15–19.

DiSalvo, B. J., & Bruckman, A. (2009). Questioning video games' influence on CS interest. In *Proceedings of the 4th International Conference on Foundations of Digital Games - FDG '09* (p. 272). New York, New York, USA: ACM Press. https://doi.org/10.1145/1536513.1536561

Du Boulay, B. (1986). Some Difficulties of Learning to Program. *Journal of Educational*

Computing Research, *2*(1), 57–73. https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9

Egenfeldt-Nielsen, S. (2006). Overview of research on the educational use of video games. *Nordic Journal of Digital Literacy*, *1*(03), 184–214.

Engle, R. A., Lam, D. P., Meyer, X. S., & Nix, S. E. (2012). How does expansive framing promote transfer? Several proposed explanations and a research agenda for investigating them. *Educational Psychologist*, *47*(3), 215–231.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87–97. https://doi.org/10.1016/j.compedu.2012.11.016

Foit, K. (2011). The robot programming language interpreter written in the Logo language. *Journal of Achievements in Materials and Manufacturing Engineering*, *45*(2), 194–203.

Forte, A., & Guzdial, M. (2005). Motivation and nonmajors in computer science: identifying discrete audiences for introductory courses. *IEEE Transactions on Education*, *48*(2), 248–253.

Fowler, A., & Cusack, B. (2011). Kodu game lab. In *Proceedings of the 6th International Conference on Foundations of Digital Games - FDG '11* (pp. 238–240). New York, New York, USA: ACM Press. https://doi.org/10.1145/2159365.2159398

Fox, M. (2003). Learning design and e-learning. *An Epic White Paper*.

Fusco, E. (1981). Matching curriculum to students cognitive levels. *Educational Leadership*, *39*(1), 47.

Gander, W., Petit, A., Berry, G., Demo, B., Vahrenhold, J., McGettrick, A., … Meyer, B. (2013). Informatics Education: Europe Cannot Afford to Miss the Boat. *ACM Europe: Informatics Education Report*, (February), 1–21. Retrieved from http://www.fit-in-it.ch/sites/default/files/small_box/Informatics education_final.pdf

Garneli, V., Giannakos, M. N., & Chorianopoulos, K. (2015). Computing education in K-12 schools: A review of the literature. In *Global Engineering Education Conference (EDUCON), 2015 IEEE* (pp. 543–551).

Gentner, D., Loewenstein, J., & Thompson, L. (2003). Learning and transfer: A general role for analogical encoding. *Journal of Educational Psychology*, *95*(2), 393.

Gomes, A., & Mendes, A. J. (2007). Learning to program-difficulties and solutions. In *International Conference on Engineering Education--ICEE* (Vol. 2007).

Gordon, M., Marron, A., & Meerbaum-Salant, O. (2012). Spaghetti for the main course?: observations on the naturalness of scenario-based programming. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education - ITiCSE '12* (p. 198). New York, New York, USA: ACM Press. https://doi.org/10.1145/2325296.2325346

Grandell, L., Peltomäki, M., Back, R.-J., & Salakoski, T. (2006). Why complicate things?: introducing programming in high school using Python. In *Proceedings of the 8th Australasian Conference on Computing Education-Volume 52* (pp. 71–80).

Grover, S. (2017). Assessing algorithmic and computational thinking in K-12: Lessons from a middle school classroom. In *Emerging research, practice, and policy on computational thinking* (pp. 269–288). Springer.

Grover, S., Cooper, S., & Pea, R. (2014). Assessing computational learning in K-12. In *Proceedings of the 2014 conference on Innovation & technology in computer science education - ITiCSE '14* (pp. 57–62). https://doi.org/10.1145/2591708.2591713

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, *25*(2), 199–237. https://doi.org/10.1080/08993408.2015.1033142

Gupta, U. G., & Houtz, L. E. (2000). High school students' perceptions of information technology skills and careers. *Journal of Industrial Technology*, *16*(4), 2–8.

Guzdial, M. (2004). Programming environments for novices. *Computer Science Education Research, 2004*, 127–154.

Hassinen, M., & Mäyrä, H. (2006a). Learning programming by programming. In *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06* (p. 117). New York, New York, USA: ACM Press. https://doi.org/10.1145/1315803.1315824

Hassinen, M., & Mäyrä, H. (2006b). Learning programming by programming. In *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06* (p. 117). New York, New York, USA: ACM Press.

https://doi.org/10.1145/1315803.1315824

Hassinen, M., & Mäyrä, H. (2006c). Learning programming by programming. In *Proceedings of the 6th Baltic Sea conference on Computing education research Koli Calling 2006 - Baltic Sea '06* (p. 117). New York, New York, USA: ACM Press. https://doi.org/10.1145/1315803.1315824

Hunicke, R., LeBlanc, M., & Zubek, R. (2004). MDA: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI* (Vol. 4, p. 1722).

Jones, J., Mccowan, D., & Stephenson, C. (2003). *A Model Curriculum for K–12 Computer Science : Final Report of the ACM K-12 Task Force Curriculum Committee*. *Computer*. Retrieved from https://www.acm.org/education/curric_vols/k12final1022.pdf

Juul, J. (2018). The game, the player, the world: Looking for a heart of gameness. *PLURAIS-Revista Multidisciplinar*, *1*(2).

Kafai, Y. B. (2006). Playing and Making Games for Learning. *Games and Culture*, *1*(1), 36–40. https://doi.org/10.1177/1555412005281767

Kafai, Y. B., & Burke, Q. (2015). Constructionist Gaming: Understanding the Benefits of Making Games for Learning. *Educational Psychologist*, *50*(4), 313–334. https://doi.org/10.1080/00461520.2015.1124022

Ke, F. (2011). A qualitative meta-analysis of computer games as learning tools. In *Gaming and Simulations: Concepts, Methodologies, Tools and Applications* (pp. 1619–1665). IGI Global.

Ke, F., & Fengfeng. (2014). An implementation of design-based learning through creating educational computer games: A case study on mathematics learning during design and computing. *Computers & Education*, *73*, 26–39. https://doi.org/10.1016/j.compedu.2013.12.010

Kelleher, C., Pausch, R., & Kiesler, S. (2007). Storytelling alice motivates middle school girls to learn computer programming. In *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '07* (p. 1455). New York, New York, USA: ACM Press. https://doi.org/10.1145/1240624.1240844

Kirriemuir, J., & McFarlane, A. (2004). Literature review in games and learning.

Kölling, M., & McKay, F. (2016). Heuristic Evaluation for Novice Programming Systems. *ACM Transactions on Computing Education (TOCE)*, *16*(3), 12.

Kramer, J. (2007). Is Abstraction the Key to Computing? *Commun. ACM*, *50*(4), 36–42. https://doi.org/10.1145/1232743.1232745

Kuechler, W. L., & Simkin, M. G. (2003). How well do multiple choice tests evaluate student understanding in computer programming classes? *Journal of Information Systems Education*, *14*(4), 389.

Lakanen, A.-J., Isomöttönen, V., & Lappalainen, V. (2012). Life two years after a game programming course. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education - SIGCSE '12*, 481. https://doi.org/10.1145/2157136.2157280

Leutenegger, S., & Edgington, J. (2007). A games first approach to teaching introductory programming. *Proceedinds of the 38th SIGCSE Technical Symposium on Computer Science Education - SIGCSE '07*, *March*(November 2015), 115. https://doi.org/10.1145/1227310.1227352

Levin, J. R. (1981). On Functions of Pictures in Prose. In *Neuropsychological and Cognitive Processes in Reading* (pp. 203–228). Elsevier. https://doi.org/10.1016/B978-0-12-185030-2.50013-5

Levy, F., & Murnane, R. J. (2005). The New Division of Labor: How Computers Are Creating the Next Job Market. In *Princeton University Press* (pp. 2–10).

Levy, F., & Murnane, R. J. (2012). *The new division of labor: How computers are creating the next job market*. Princeton University Press.

Lewis, C. M. (2010). How programming environment shapes perception, learning and goals. In *Proceedings of the 41st {ACM} technical symposium on Computer science education - {SIGCSE} {\textquotesingle}10*. Association for Computing Machinery ({ACM}). https://doi.org/10.1145/1734263.1734383

Liao, Y.-K. C., & Bright, G. W. (1991). Effects of Computer Programming on Cognitive Outcomes: A Meta-Analysis. *Journal of Educational Computing Research*, *7*(3), 251–268. https://doi.org/10.2190/E53G-HH8K-AJRR-K69M

Lightbot. (n.d.). Retrieved from http://lightbot.com/flash.html

Litecky, C., Prabhakar, B., & Arnett, K. (2006). The IT/IS job market: a longitudinal perspective. In *Proceedings of the 2006 ACM SIGMIS CPR conference on computer personnel research: Forty four years of computer personnel research: achievements, challenges & the future* (pp. 50–52).

Malan, D. J., & Leitner, H. H. (2007). Scratch for budding computer scientists. *ACM SIGCSE Bulletin*, *39*(1), 223. https://doi.org/10.1145/1227504.1227388

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). Programming by choice. *ACM SIGCSE Bulletin*, *40*(1), 367. https://doi.org/10.1145/1352322.1352260

Maloney, J. H., Peppler, K., Kafai, Y., Resnick, M., Rusk, N., Maloney, J. H., … Rusk, N. (2008). Programming by choice. *ACM SIGCSE Bulletin*, *40*(1), 367. https://doi.org/10.1145/1352322.1352260

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The scratch programming language and environment. *ACM Transactions on Computing Education (TOCE)*, *10*(4), 16.

Maltese, A. V., & Tai, R. H. (2010). Eyeballs in the fridge: Sources of early interest in science. *International Journal of Science Education*, *32*(December 2014), 669–685. https://doi.org/10.1080/09500690902792385

McClarty, K. L., Orr, A., Frey, P. M., Dolan, R. P., Vassileva, V., & McVay, A. (2012). A literature review of gaming in education. *Gaming in Education*, 1–35.

Meerbaum-Salant, O, Armoni, M., & Ben-Ari, M. (2011). Habits of programming in Scratch. In *Proceedings of the 16th annual joint conference on Innovation and technology in computer science education* (pp. 168–172). https://doi.org/10.1145/1999747.1999796

Meerbaum-Salant, Orni, Armoni, M., & Ben-Ari, M. (Moti). (2013). Learning computer science concepts with Scratch. *Computer Science Education*, *23*(3), 239–264. https://doi.org/10.1080/08993408.2013.832022

Milne, I., & Rowe, G. (2002). Difficulties in learning and teaching programming—views of students and tutors. *Education and Information Technologies*, *7*(1), 55–66.

Ministrarstvo znanosti i obrazovanja. (2018). *Kurikulum nastavnoga predmeta informatika za osnovne i srednje škole*. Retrieved from https://mzo.hr/sites/default/files/dokumenti/2018/OBRAZOVANJE/Nacionalni-

kurikulumi/informatika/kurikulum_nastavnoga_predmeta_informatika.pdf

Ministry of science education and Sports of the Republic of Croatia. (2005). *The curriculum for primary school*. Zagreb.

Mladenović, M., Boljat, I., & Žanko, Ž. (2018). Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. *Education and Information Technologies*, *23*(4), 1483–1500. https://doi.org/10.1007/s10639-017-9673-3

Mladenović, M., Krpan, D., & Mladenović, S. (2016). Introducing programming to elementary students novices by using game development in Python and Scratch. In *EDULEARN16 Proceedings* (pp. 1622–1629). IATED. https://doi.org/10.21125/edulearn.2016.1323

Mladenović, M., Krpan, D., & Mladenović, S. (2017). Learning programming from Scratch. In *International Conference on New Horizons in Education INTE*.

Mladenović, M., Rosić, M., & Mladenović, S. (2016). Comparing Elementary Students ' Programming Success Based on Programming Environment. *International Journal of Modern Education and Computer Science*, *8*(August), 1–10. https://doi.org/10.5815/ijmecs.2016.08.01

Mladenović, M., Žanko, Ž., & Mladenović, S. (2014). Elementary students' attitude towards programming in the Republic of Croatia. In *CIET*.

Mladenović, S., Krpan, D., & Mladenović, M. (2016). Using Games to Help Novices Embrace Programming: From Elementary to Higher Education. *International Journal of Engineering Education*, *32*(1), 521–531. Retrieved from http://gateway.webofknowledge.com/gateway/Gateway.cgi?GWVersion=2&SrcAuth=ORCID&SrcApp=OrcidOrg&DestLinkType=FullRecord&DestApp=WOS_CPL&KeyUT=WOS:000374234700021&KeyUID=WOS:000374234700021

Mladenović, S., Žanko, Ž., & Mladenović, M. (2015). Elementary students' motivation towards informatics course. *Procedia - Social and Behavioral Sciences*, *174*, 3780–3787. https://doi.org/10.1016/j.sbspro.2015.01.1113

Moser, R. (1997). A Fantasy Adventure Game As a Learning Environment: Why Learning to Program is So Difficult and What Can Be Done About It. *SIGCSE Bull.*, *29*(3), 114–116. https://doi.org/10.1145/268809.268853

Oblinger, D., & Oblinger, J. (2005). Is It Age or IT: First Steps Toward Understanding the Net

Generation. *Educating the Net Generation*, *Chapter 2*(2), 2.1-2.20. https://doi.org/Article

Office of Technology Assessment. (1984). Computerized manufacturing automation employment, education, and the workplace. Washington, D. C.: U.S. Congress.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York, NY, USA: Basic Book, Inc.

Papert, S. (1993). *The children's machine: rethinking school in the age of the computer*. BasicBooks. Retrieved from http://dl.acm.org/citation.cfm?id=139395

Papert, S. (2010). *Does Easy Do It? Children, Games, and Learning*. *Game Developer*. https://doi.org/10.1017/CBO9781107415324.004

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism*, *36*(2), 1–11.

Pardamean, B., Evelin, E., & Honni, H. (2011). The effect of logo programming language for creativity and problem solving. In *Proceedings of the 10th WSEAS international conference on E-Activities* (pp. 151–156).

Pausch, R., & Zaslow, J. (2008). Last Lecture. *Statistics*, *7*, 1–18. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/19064375

Peppler, K. A., & Kafai, Y. B. (2007). From SuperGoo to Scratch: exploring creative digital media production in informal learning. *Learning, Media and Technology*, *32*(2), 149–166. https://doi.org/10.1080/17439880701343337

Perlis, A. J. (1962). The computer in the university. In *Computers and the World of the Future*. Cambridge, MA, USA: MIT Press.

Piaget, J. (1952). *The origins of intelligence in children.* New York, NY, USA: W W Norton & Co. https://doi.org/10.1037/11494-000

Piaget, J., & Play, D. (1962). imitation in Childhood. *NY: Norton*.

Pink, D. H. (2005). *A whole new mind: Moving from the information age to the conceptual age*. Riverhead Books New York.

Pink, D. H. (2007). Revenge of the right brain. *PUBLIC MANAGEMENT-LAWRENCE THEN WASHINGTON-*, *89*(6), 10.

Plass, J. L., Homer, B. D., & Kinzer, C. K. (2015). Foundations of Game-Based Learning.

Educational Psychologist, *50*(4), 258–283. https://doi.org/10.1080/00461520.2015.1122533

Prensky, M. (2001). Digital Natives, Digital Immigrants. *From On the Horizon*, *9*(5).

Prensky, M. (2003). Digital game-based learning. *Computers in Entertainment*, *1*(1), 21. https://doi.org/10.1145/950566.950596

Prensky, M. (2010). *Teaching digital natives: Partnering for real learning*. Corwin Press.

Reigeluth, C. M. (2016). Instructional Theory and Technology for the New Paradigm of Education. *RED. Revista de Educación a Distancia*, *32*, 1–18. Retrieved from http://www.um.es/ead/red/32

Resnick, M. (1996). Distributed Constructionism. *Proceedings of the 1996 International Conference on Learning Sciences*, 280–284. Retrieved from https://dl.acm.org/citation.cfm?id=1161173

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., … Kafai, Y. (2009). Scratch: Programming for All. *Commun. ACM*, *52*(11), 60–67. https://doi.org/10.1145/1592761.1592779

Robins, A., Rountree, J., & Rountree, N. (2003a). Learning and teaching programming: A review and discussion. *Computer Science Education*, *13*(2), 137–172.

Robins, A., Rountree, J., & Rountree, N. (2003b). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education*, *13*(2), 137–172. https://doi.org/10.1076/csed.13.2.137.14200

RoboZZle. (n.d.). Retrieved from http://robozzle.com/

Rogers, Y., Shum, V., Marquardt, N., Lechelt, S., Johnson, R., Baker, H., & Davies, M. (2017). From the BBC Micro to micro:bit and Beyond: A British Innovation. *Interactions*, *24*(2), 74–77.

Sedelmaier, Y., & Landes, D. (2014). Practicing soft skills in software engineering: A project-based didactical approach. In *Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills* (pp. 161–179). IGI Global.

Sekiya, T., & Yamaguchi, K. (2013). Tracing quiz set to identify novices' programming misconceptions. In *Proceedings of the 13th Koli Calling International Conference on*

Computing Education Research - Koli Calling '13 (pp. 87–95). New York, New York, USA: ACM Press. https://doi.org/10.1145/2526968.2526978

Sentance, S., Waite, J., Hodges, S., MacLeod, E., & Yeomans, L. (2017). Creating Cool Stuff: Pupils' Experience of the BBC micro: bit. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (pp. 531–536).

Simmons, S., DiSalvo, B., & Guzdial, M. (2012). Using game development to reveal programming competency. In *Proceedings of the International Conference on the Foundations of Digital Games - FDG '12* (p. 89). New York, New York, USA: ACM Press. https://doi.org/10.1145/2282338.2282359

Smed, J., Hakonen, H., & others. (2003). *Towards a definition of a computer game*. Turku Centre for Computer Science Turku, Finland.

Sorva, J. (2013). *Visual Program Simulation in Introductory Program Education*. *Journal of Chemical Information and Modeling* (Vol. 53). Aalto Univ. School of Science. https://doi.org/10.1007/s13398-014-0173-7.2

Sprankle, M., & Hubbard, J. (2008). *Problem Solving and Programming Concepts* (8th ed.). Upper Saddle River, NJ, USA: Prentice Hall Press.

Szetela, W., & Nicol, C. (1992). Evaluating Problem Solving in Mathematics. *Educational Leadership*, *49*(8), 42–45.

Tabet, N., Gedawy, H., Alshikhabobakr, H., & Razak, S. (2016). From Alice to Python. Introducing Text-based Programming in Middle Schools. https://doi.org/10.1145/2899415.2899462

Tai, R. H. (2006). Career choice. Planning early for careers in science. *Science*, *312*(5777), 1143–1144. https://doi.org/10.1126/science.1128690

Theodoropoulos, A., Antoniou, A., & Lepouras, G. (2017). How do different cognitive styles affect learning programming? Insights from a game-based approach in Greek schools. *ACM Transactions on Computing Education (TOCE)*, *17*(1), 3.

Tsai, M.-H., Huang, C.-H., & Zeng, J.-Y. (2006). Game Programming Courses for Non Programmers. In *Proceedings of the 2006 international conference on Game research and development*. Perth.

Turkle, S., & Papert, S. (1992). Epistemological Pluralism and the Revaluation of the Concrete. *Journal of Mathematical Behavior*, *11*(1), 3–33.

Tyrén, M., Carlborg, N., Heath, C., & Eriksson, E. (2018). Considerations and Technical Pitfalls for Teaching Computational Thinking with BBC micro: bit. In *Proceedings of the Conference on Creativity and Making in Education* (pp. 81–86).

Uludag, S., Karakus, M., & Turner, S. W. (2011). Implementing IT0 / CS0 with Scratch , App Inventor for Android , and Lego Mindstorms. In *Proceedings of the 2011 conference on Information technology education* (pp. 183–189). West Point, NY, USA. https://doi.org/10.1145/2047594.2047645

Van Eck, R. (2006). Digital game-based learning: It's not just the digital natives who are restless. *EDUCAUSE Review*, *41*(2), 16.

Violino, B. (2009). Time to Reboot. *Commun. ACM*, *52*(4), 19. https://doi.org/10.1145/1498765.1498774

Webb, M., Davis, N., Bell, T., Katz, Y., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies*, *22*(2), 445–468. https://doi.org/10.1007/s10639-016-9493-x

Weintrop, D., & Wilensky, U. (2015). To Block or not to Block, That is the Question: Students' Perceptions of Blocks-based Programming. *Proceedings of the 14th International Conference on Interaction Design and Children - IDC '15*, 199–208. https://doi.org/10.1145/2771839.2771860

Werner, L., Campe, S., & Denner, J. (2012). Children learning computer science concepts via Alice game-programming. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education - SIGCSE '12* (p. 427). New York, New York, USA: ACM Press. https://doi.org/10.1145/2157136.2157263

Werner, L., Denner, J., Bliesner, M., & Rex, P. (2009). Can middle-schoolers use Storytelling Alice to make games? In *Proceedings of the 4th International Conference on Foundations of Digital Games - FDG '09* (p. 207). New York, New York, USA: ACM Press. https://doi.org/10.1145/1536513.1536552

Werner, L. L., Campe, S., & Denner, J. (2005). Middle school girls + games programming =

information technology fluency. In *Proceedings of the 6th conference on Information technology education - SIGITE '05* (p. 301). New York, New York, USA: ACM Press. https://doi.org/10.1145/1095714.1095784

White, G., & Sivitanides, M. (2003a). An empirical investigation of the relationship between success in mathematics and visual programming courses. *Journal of Information Systems Education, 14*(4), 409–416. Retrieved from http://www.jise.org/Issues/14/14(4)-409.pdf

White, G., & Sivitanides, M. (2003b). An empirical investigation of the relationship between success in mathematics and visual programming courses. *Journal of Information Systems Education, 14*(4), 409.

Wilson, A., Hainey, T., & Connolly, T. (2013). Evaluation of Computer Games Developed by Primary School Children to Gauge Understanding of Programming Concepts. *Interational Journal of Games-Based Learning*, *3*(1), 93–109. Retrieved from https://pdfs.semanticscholar.org/436c/852c4094a58ff9e629ace758612f58459342.pdf

Wilson, C. (2015). Hour of code---a record year for computer science. *ACM Inroads*, *6*(1), 22.

Wing, J. M. (2008). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, *366*, 3717–3725. https://doi.org/10.1098/rsta.2008.0118

Winslow, L. E. (1996). Programming pedagogy—a psychological overview. *ACM Sigcse Bulletin, 28*(3), 17–22.

Wood, D., Bruner, J. S., & Ross, G. (1976). THE ROLE OF TUTORING IN PROBLEM SOLVING. *Journal of Child Psychology and Psychiatry*, *17*(2), 89–100. https://doi.org/10.1111/j.1469-7610.1976.tb00381.x

Wouters, P., der Spek, E. D., & Van Oostendorp, H. (2009). Current practices in serious game research: A review from a learning outcomes perspective. In *Games-based learning advancements for multi-sensory human computer interfaces: techniques and effective practices* (pp. 232–250). IGI Global.

Xinogalos, S., Satratzemi, M., & Malliarakis, C. (2017). Microworlds, games, animations, mobile apps, puzzle editors and more: What is important for an introductory programming environment? *Education and Information Technologies*, *22*(1), 145–176. https://doi.org/10.1007/s10639-015-9433-1

Yadav, A., Stephenson, C., & Hong, H. (2017). Computational thinking for teacher education. *Communications of the ACM*, *60*(4), 55–62. https://doi.org/10.1145/2994591

Yardi, S., & Bruckman, A. (2007). What is Computing?: Bridging the Gap Between Teenagers' Perceptions and Graduate Students' Experiences. In *Proceedings of the Third International Workshop on Computing Education Research* (pp. 39–50). New York, NY, USA: ACM. https://doi.org/10.1145/1288580.1288586

Yuen, T. T., & Liu, M. (2010). How interactive multimedia authoring transforms object-oriented thinking. In *Proceedings of the 41st ACM technical symposium on Computer science education - SIGCSE '10* (p. 426). New York, New York, USA: ACM Press. https://doi.org/10.1145/1734263.1734408

Zainal, N. F. A., Shahrani, S., Yatim, N. F. M., Rahman, R. A., Rahmat, M., & Latih, R. (2012). Students' perception and motivation towards programming. *Procedia-Social and Behavioral Sciences*, *59*, 277–286.

Žanko, Ž., Mladenović, M., & Mladenović, S. (2014). Students attitude towards informatics curricula. In *7th International Conference of Education, Research and Innovation*.

Zur-Bargury, I., Pârv, B., & Lanzberg, D. (2013). A nationwide exam as a tool for improving a new curriculum. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education - ITiCSE '13* (p. 267). New York, New York, USA: ACM Press. https://doi.org/10.1145/2462476.2462479

PRILOZI

Popis priloga

Prilog 1 – Završni ispit znanja u Logu (1. istraživanje)

Ime i prezime:
Razred:

1. Poveži naredbe i njihovo značenje:

a) fd 100	1) Idi nazad za 100 koraka
b) rt 100	2) Idi naprijed za 100 koraka
c) lt 100	3) Okreni se desno za 100 stupnjeva
d) bk 100	4) Ponovi 100 puta
e) repeat 100 []	5) Okreni se lijevo za 100 stupnjeva

2. Koliko će se koraka napraviti nakon izvršavanja skripte prikazane u tablici ispod (zaokruži točan odgovor)?

a) 100 b) 150 c) 180 d) 200 e) 380

(skripta)	Skiciraj kretanje kornjače nakon izvršavanja naredbi!

`fd 100`
`rt 90`
`fd 50`
`fd 50`
`rt 90`	

3. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
repeat 10 [fd 10]
```

a) 1 b) 10 c) 20 d) 100 e) 1000

Ukratko obrazloži svoj odgovor.

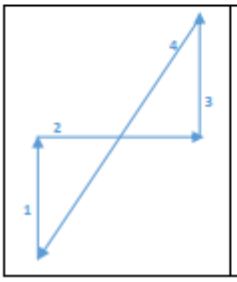
4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
repeat 10 [fd 1 repeat 10 [fd 1]]
```

a) 10 b) 20 c) 100 d) 110 e) 1000

Ukratko obrazloži svoj odgovor.

5. Napiši naredbe koje bi nacrtale crtež na slici ispod. Napomena: stranice 1, 2, i 3 su duljine 100, crtanje počinje iz ishodišta

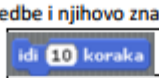


(Prostor za pisanje)

Prilog 2 – Završni ispit znanja u Scratchu (1. istraživanje)

Ime i prezime:
Razred:

1. Poveži naredbe i njihovo značenje:

	a)	1) Ponavljaj zauvijek
	b)	2) Ponovi deset puta
	c)	3) Idi naprijed deset koraka
	d)	4) Okreni se u desno za 100 stupnjeva
	e)	5) Okreni se u lijevo za 100 stupnjeva

2. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

 a) 0 b) 100 c) 150 d) 200	Možeš li skicirati pomoću crta kretanje lika na pozornici nakon izvršavanja skripte na slici gore

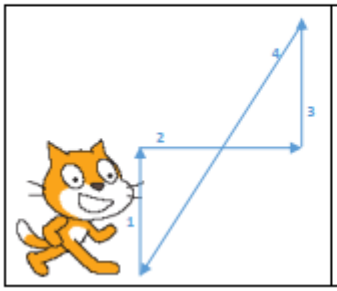
3. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

 a) 1 b) 10 c) 20 d) 100 e) 1000	Ukratko obrazloži svoj odgovor!

4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

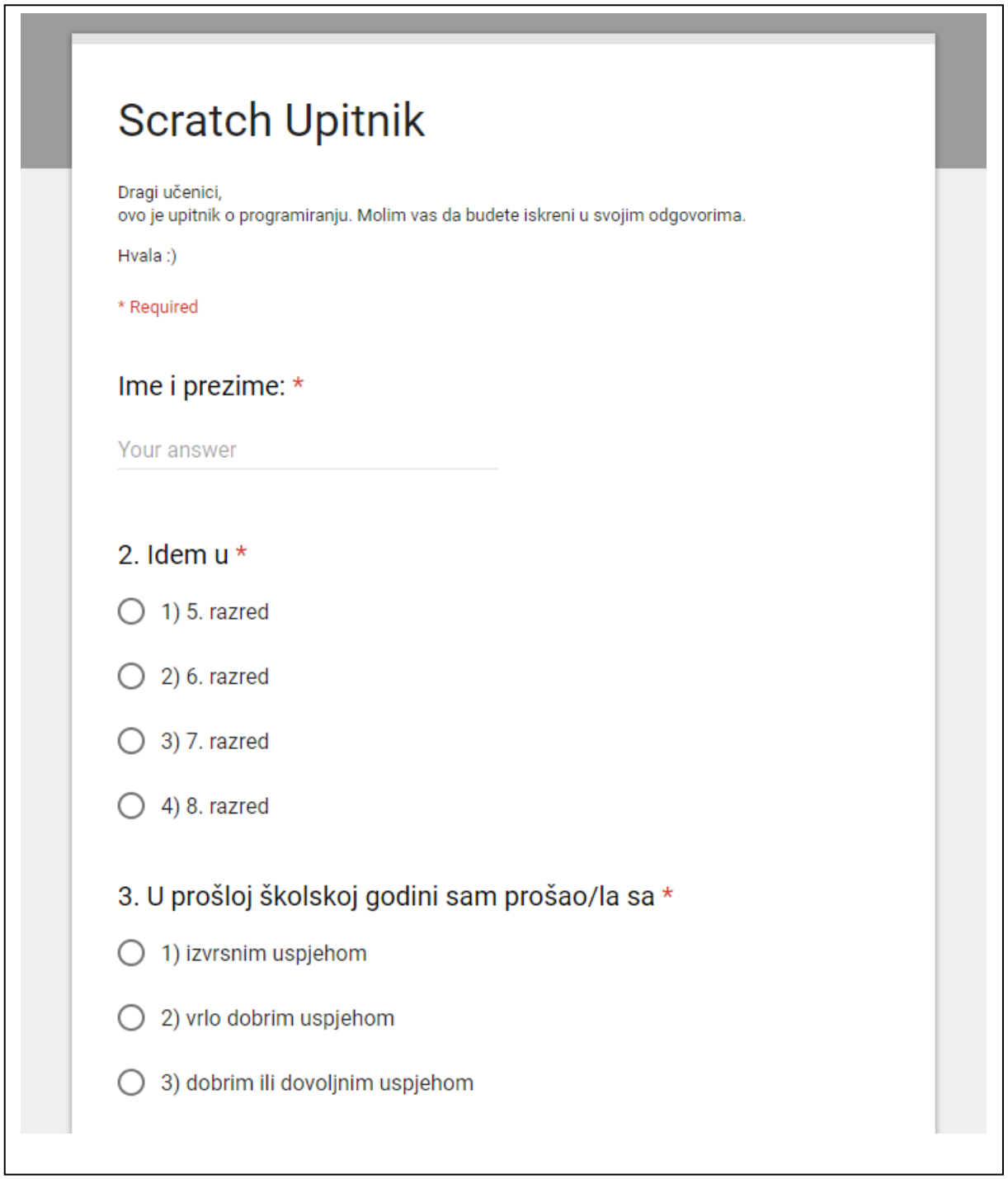
 a) 10 b) 20 c) 100 d) 110 e) 1000	Ukratko obrazloži svoj odgovor!

5. Napiši naredbe koje bi napravile kretanje mačke na pozornici kao na slici ispod (početna pozicija je u ishodištu=0, y=0, pokreti: 1, 2 i 3 su duljine 100 koraka).



(Prostor za pisanje)

Prilog 3 – Anketa o stavu prema programiranju i svakom od programskih jezika (1. istraživanje)



Scratch Upitnik

Dragi učenici,
ovo je upitnik o programiranju. Molim vas da budete iskreni u svojim odgovorima.

Hvala :)

* Required

Ime i prezime: *

Your answer

2. Idem u *

- 1) 5. razred
- 2) 6. razred
- 3) 7. razred
- 4) 8. razred

3. U prošloj školskoj godini sam prošao/la sa *

- 1) izvrsnim uspjehom
- 2) vrlo dobrim uspjehom
- 3) dobrim ili dovoljnim uspjehom

4. Kakav uspjeh očekuješ na kraju ove školske godine? *

- ○ 1) izvrsnim uspjehom
- ○ 2) vrlo dobrim uspjehom
- ○ 3) dobrim ili dovoljnim uspjehom
- ○ 4) Ne znam

5. Već sam programirao/la prije nastave informatike ove godine *

- ○ 1) Da
- ○ 2) Ne

6. Ako je odgovor na prethodno pitanje bio Da, u kojem programskom jeziku?

- ☐ Logo
- ☐ BASIC
- ☐ U nekom drugom jeziku

7. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

7. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

8. Koji ti se programski jezik više sviđa? *

- ○ 1) Scratch
- ○ 2) Logo

9. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa LOGO. *

1	2	3	4	5
○	○	○	○	○

10. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Scratch. *

1	2	3	4	5
○	○	○	○	○

11. Ako imaš neki svoj komentar u vezi programiranja slobodno ga napiši.

Your answer

SUBMIT

Prilog 4 – Preliminarni ispit (PI) sposobnosti rješavanja problema (2. istraživanje)

Pitanja za programere ☺

Dragi učenici, molim vas da odgovorite na pitanja. Kada budete odgovarali na pitanja dobro razmislite, možete se poslužiti prostorom za pisanje na papiru za pomoć (možeš pisati i sa druge strane papira).

Ime i prezime: _____

1. Nastavi niz sa sljedeća tri slova: A B A C A D A E __ __ __

2. Nastavi niz sa dvije kombinacije slova: JKLMNO JKLMON JKLOMN JKOLMN _____ _____

3. Ako robotu damo upute da ponovi sljedeće radnje 3 puta: idi po 3 koraka naprijed. Koliko će ukupno koraka robot napraviti?

4. Ako riječ REČENICA sadrži manje od 9 slova i više od 3 sloga, napiši prvi slog. U suprotnom napiši suglasnik koji se nalaze najdesnije u riječi.

5. Ako je petak 3 dana prije jučerašnjeg dana. Koji dan će biti sutra?

6. Ivan je teži od Martina, ali lakši od Ante. Napiši njihova imena poredana po težini od lakšeg prema težem.

7. Božić se slavi 25. prosinca. Zamisli da je Božić 2 dana prije četvrtka. Kojeg je dana Božić, a koji će dan biti 4 dana iza Božića i koji će to biti datum.

8. Dražen, Ana i Ivana imaju zanimanja učitelj, prodavač i kuhar. Dražen je niži od Ane, ali viši od Ivane. Prodavač je najviši, a kuhar najniži. Kojeg su zanimanja Dražen, Ana i Ivana? Tko je najviši, a tko najniži?

Prilog 5 – Završni ispit znanja u Pythonu (ZIP) (2. istraživanje)

Ime i prezime: _____

1. Poveži naredbe i njihovo značenje

1. input()	a) ispiši
2. print()	b) Pretvori u cijeli broj
3. int()	c) Ako je
4. if	d) unesi

2. Napiši svojim riječima što je to **algoritam**!

3. Navedi su **tri** osnovna algoritma i ako možeš navedi neki jednostavni primjer za svaki?

1. _____
2. _____
3. _____

4. Koji od sljedećih algoritama je ispravno napisan? Odgovori koji je to primjer algoritma

a	1.Uđem u trgovinu 2.Dođem na kasu 3.Platim račun 4.Uzmem žvake
b	1.Dođem na kasu 2.Uđem u trgovinu 3.Platim račun 4.Uzmem žvake
c	1.Uđem u trgovinu 2.Uzmem žvake 3.Dođem na kasu 4.Platim račun

Ovo je primjer algoritma_____

5. Koje su **tri** osnovne faze izvršavanja programa?

a) Ulaz, obrada, izlaz
b) Ulaz, izlaz
c) Izlaz, ulaz, obrada
d) Ulaz, središte, izlaz

6. Zaokruži točno napisan program i navedi koje su greške kod ostalih (možeš ih i zaokružiti).

a	broj1=input(„Unesi prvi broj“) broj2=input(„Unesi drugi broj“) zbroj = int(a)+int(b) print(„Zbroj je: “, zbroj)	
b	broj1=input(„Unesi prvi broj“) broj2=input(„Unesi drugi broj“) zbroj = int(broj1)+int(broj2) print(„Zbroj je: “, zbroj)	
c	broj1=input(„Unesi prvi broj“) broj2=input(„Unesi drugi broj“) zbroj = int(broj1)+int(broj2) print(„Zbroj je: “, broj3)	

7. Zaokruži i napiši sa desne strane koji dijelovi programa koji spadaju u fazu **ulaza**, koji u fazu **obrade**, a koji u fazu **izlaza**?

a=input(„Unesi stranicu a pravokutnika“) b=input(„Unesi stranicu b pravokutnika“) povrsina=int(a)*int(b) print(„Površina pravokutnika je: “, povrsina)	

8. Dopuni sljedeći program tako da bude ispravno napisan.

```
[      ]=input(„Unesi Markov džeparac“)
jana=[      ] („Unesi Janin džeparac“)
ukupno=int([      ])* [    ](jana)
print(„Ukupan iznos Markovog i Janinog džeparca je: “,[      ])
```

9. Nađi tri greške u programu (zaokruži ih) i u desni pravokutnik pravilno napiši program!

a=input(„Unesi stranicu a kvadrata“) povrsina=int(a)*a print=(„Povrsina kvadrata je: “, a)	

10. Napiši algoritam (ili program) za program u koji će se na osnovu unesenog broja mrvica koje je bacila Marica i broj mrvica koje je bacio Ivica ispisati ukupan broj bačenih mrvica.

11. Pogledaj sljedeći algoritam i napiši koji je to algoritam.

```
Ako dobijem 5 iz informatike
        Mogu se igrati
Inače
        Moram još vježbati
```

Ovo je primjer algoritma_____

Prilog 6 – Završni ispit znanja u Scratchu (ZIS) (2. istraživanje)

Ime i prezime:
Razred:

1. Poveži naredbe i njihovo značenje:

	a)	a) Ponavljaj zauvijek
	b)	b) Ponovi deset puta
	c)	c) Idi naprijed deset koraka
	d)	d) Okreni se u desno za 100 stupnjeva
	e)	e) Okreni se u lijevo za 100 stupnjeva

2. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

a) 0
b) 100
c) 150
d) 200

Možeš li skicirati pomoću crta kretanje lika na pozornici nakon izvršavanja skripte na slici gore

3. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

a) 1
b) 10
c) 20
d) 100
e) 1000

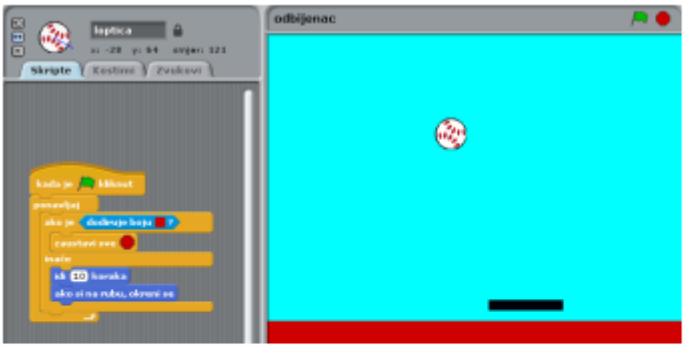
Ukratko obrazloži svoj odgovor!

4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 10
b) 20
c) 100
d) 110
e) 1000

Ukratko obrazloži svoj odgovor!

5. Pogledaj sliku ispod pa odgovori na pitanja.



a) Hoće li se loptica nastaviti kretati nakon što dodirne crvenu boju? Obrazloži odgovor!
b) Što bi se dogodilo kada ne bi bilo naredbe ***ponavljaj***?
c) Čemu služi naredba ponavljaj?
d) Pogledaj skriptu za lopticu ispod i objasni što skripta radi.

Prostor za odgovore:

2

6. Pogledaj pozornicu i skriptu na sljedećoj slici.

a) Pokušaj objasniti kako će se ponašati igrač kada se klikne na zelenu zastavicu.
b) Čemu služi naredba **idi na x=-130 y=-93**?
c) Hoće li se igrač nastaviti kretati nakon što udari loptu? Obrazloži svoj odgovor.
d) Kada Igrač dodirne loptu hoće li se zaustaviti sve ili samo igrač?
e) Ako želimo da se lopta pokrene nakon što igrač razglasi udarac kako bi to napravili? Napiši algoritam.

Prostor za odgovore:

3

Prilog 7 – Prva anketa o stavu prema programiranju (2. istraživanje)

Ime i prezime: _____

Dragi učenici prije testa vas molim da iskreno odgovorite na nekoliko pitanja u vezi programiranja. Vaši odgovori bi trebali pomoći u nastavi za sljedeću godinu kako bi predavanja bila što zanimljivija ☺

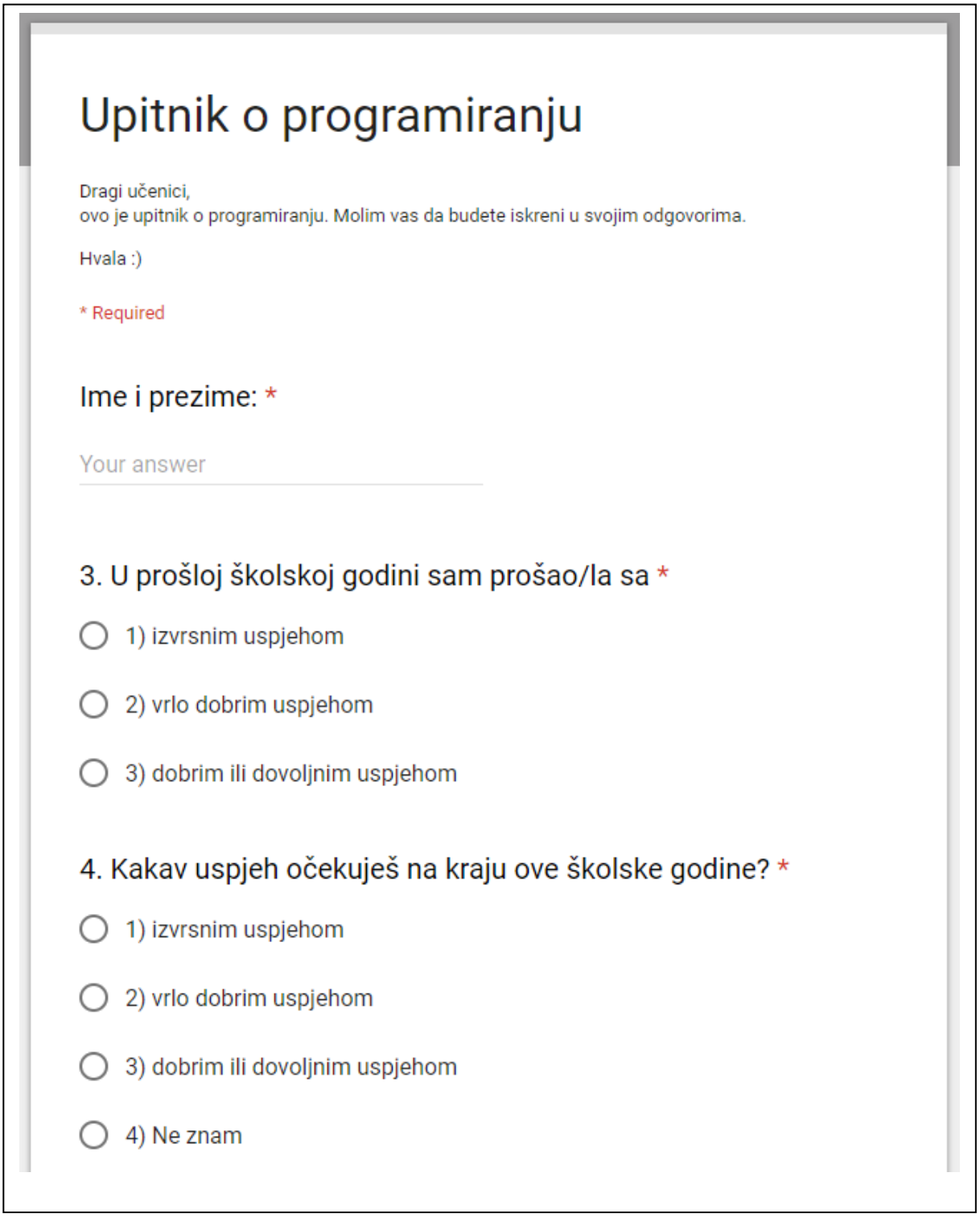
Zaokruži ocjenu od 1 do 5 koliko ti se sviđa programiranje.

1	2	3	4	5
Uopće mi se ne sviđa	*Nije mi baš nešto*	*Niti mi se sviđa, niti mi se ne sviđa*	*Dobro je*	*Jako mi se sviđa*

Možeš li ukratko napisati što ti se sviđa kod programiranja

Možeš li ukratko napisati što ti se **ne** sviđa kod programiranja

Prilog 8 – Druga anketa o stavu prema programiranju (2. istraživanje)



Upitnik o programiranju

Dragi učenici,
ovo je upitnik o programiranju. Molim vas da budete iskreni u svojim odgovorima.

Hvala :)

* Required

Ime i prezime: *

Your answer

3. U prošloj školskoj godini sam prošao/la sa *

- 1) izvrsnim uspjehom
- 2) vrlo dobrim uspjehom
- 3) dobrim ili dovoljnim uspjehom

4. Kakav uspjeh očekuješ na kraju ove školske godine? *

- 1) izvrsnim uspjehom
- 2) vrlo dobrim uspjehom
- 3) dobrim ili dovoljnim uspjehom
- 4) Ne znam

5. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

6. Koji ti se programski jezik više sviđa? *

- ○ 1) Scratch
- ○ 2) Python

7. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Python. *

1	2	3	4	5
○	○	○	○	○

8. Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Scratch. *

1	2	3	4	5
○	○	○	○	○

9. Ako imaš neki svoj komentar u vezi programiranja slobodno ga napiši.

Your answer

SUBMIT

Prilog 9 – Završni ispit znanja u Logu (ZIL) (3. istraživanje)

Ime, prezime, razred:_____

1. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

a) 100 b) 150 c) 180 d) 200 e) 380

```
fd 100
lt 90
fd 50
fd 50
lt 90
```

Skiciraj kretanje kornjače nakon izvršavanja programa. Početni položaj je prikazan ispod.

2. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
fd 10 repeat 10[fd 10]
```

a) 10 b) 20 c) 30 d) 100 e) 110

Ukratko obrazloži svoj odgovor!

3. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
repeat 10[fd 10 rt 10 fd 10]
```

a) 20 b) 30 c) 40 d) 200 e) 300

Ukratko obrazloži svoj odgovor!

4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
repeat 10[fd 1 repeat 10[fd 1]]
```

a) 10 b) 20 c) 100 d) 110 e) 1000

Ukratko obrazloži svoj odgovor!

5. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

```
fd 1 repeat 10[ fd 1] repeat 10[ fd 1]
```

a) 20 b) 21 c) 23 d) 110 e) 103

Ukratko obrazloži svoj odgovor!

Prilog 10 – Završni ispit znanja u Pythonu (ZIP) (3. istraživanje)

Ime, prezime, razred:_____

1. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

a) 100 b) 150 c) 180 d) 200 e) 380

```
fd 100
lt 90
fd 50
fd 50
lt 90
```

Skiciraj kretanje kornjače nakon izvršavanja programa. Početni položaj je prikazan ispod.

2. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 10 b) 20 c) 30 d) 100 e) 110

```
fd(10)
for i in range (10):
    fd(10)
```

Ukratko obrazloži svoj odgovor!

3. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 20 b) 30 c) 40 d) 200 e) 300

```
for i in range (10):
    fd(10)
    rt(10)
    fd(10)
```

Ukratko obrazloži svoj odgovor!

4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 10 b) 20 c) 100 d) 110 e) 1000

```
for i in range (10):
    fd(1)
    for i in range (10):
        fd(1)
```

Ukratko obrazloži svoj odgovor!

5. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 20 b) 21 c) 23 d) 110 e) 103

```
fd(1)
for i in range (10):
    fd(1)
for i in range (10):
    fd(1)
```

Ukratko obrazloži svoj odgovor!

Prilog 11 – Završni ispit znanja u Scratchu (ZIS) (3. istraživanje)

Ime, prezime, razred:_____

1. Koliko će se koraka napraviti nakon izvršavanja sljedeće skripte (zaokruži točan odgovor)?

a) 100 b) 150 c) 180 d) 200 e) 380

Skiciraj kretanje kornjače nakon izvršavanja programa. Početni položaj je prikazan ispod. ▼

2. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 10 b) 20 c) 30 d) 100 e) 110

Ukratko obrazloži svoj odgovor!

3. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 20 b) 30 c) 40 d) 200 e) 300

Ukratko obrazloži svoj odgovor!

4. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 10 b) 20 c) 100 d) 110 e) 1000

Ukratko obrazloži svoj odgovor!

5. Koliko će se koraka napraviti nakon izvršavanja sljedećih naredbi (zaokruži točan odgovor)?

a) 20 b) 21 c) 23 d) 110 e) 103

Ukratko obrazloži svoj odgovor!

Prilog 12 – • Preliminarni ispit znanja u Pythonu (PI-P) (4. istraživanje)

Ime, prezime, razred:_____

1. U tablici ispod je 6 zadataka. U svakom zadatku je prikazano nekoliko programa. Pažljivo pogledaj zadatke pa **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa. U prostor za objašnjenje pokušaj **ukratko objasniti** svoj odgovor.

	Program	Odgovor	Objašnjenje odgovora

a)	`br1 = 100`
	`br2 = 20`
	`br1 = 5`
	`print(br1)`
		a) 100
		b) 20
		c) br1
		d) 5
		e) 105	

b)	`prvi = 100`
	`drugi = 1`
	`drugi = prvi`
	`print(drugi)`
		a) 1
		b) 100
		c) drugi
		d) 1
		e) 101	

c)	`broj = 100`
	`broj = broj+1`
	`print(broj)`
		a) 100
		b) broj
		c) broj+1
		d) 1
		e) 101	

d)	`x=0`
	`y=0`
	`x=100`
	`y=x+100`
	`x=1`
	`print(x)`
		a) 0
		b) 100
		c) 200
		d) 1
		e) x	

e)	`x=0`
	`y=0`
	`x=100`
	`y=x+100`
	`print(y)`
		a) 0
		b) 100
		c) 200
		d) 1
		e) y	

f)	`broj1= 100`
	`broj2 = 20`
	`broj1= broj2`
	`broj2 = broj1+1`
	`print(broj2)`
		a) 21
		b) broj1+1
		c) 20
		d) 120
		e) 121	

2. Za rješavanje sljedećih zadataka prvo pogledaj program pa odgovori što će se ispisati ovisno o unosu podataka.

a) Napravljen je program u koji se unosi razina svjetla. Ako je razinu svjetla barem 128 vidi se dobro i treba ispisati 'Sve vidim'. Program je prikazan ispod. Prouči program.

```
razina_svjetla = int(input())
if razina_svjetla >= 128:
    print ('Sve vidim!')
```

Unos	Odgovor	Objašnjenje odgovora
Što će se prikazati na zaslonu ako za razinu svjetla unesemo 100?	a) 100 b) 128 c) Sve vidim! d) razina_svjetla e) Ništa se neće ispisati.	
Što će se prikazati na zaslonu ako za razinu svjetla unesemo 128?	a) 128 b) Sve vidim! c) razina_svjetla d) Ništa se neće prikazati. e) Ništa od navedenog. Rješenje je:_____	
Što će se prikazati na zaslonu ako za razinu svjetla unesemo 200?	a) 200 b) 128 c) Sve vidim! d) razina_svjetla e) Ništa se neće prikazati.	

b) Napravljen je program u koji se unosi temperatura. Ako je temperatura ispod 20 treba ispisati 'Hladno', dok za ostale slučajeve treba ispisati 'Nije hladno'. Program je prikazan ispod. Prouči program.

```
temperatura = int(input())
if temperatura < 20:
    print ('Hladno')
else:
    print ('Nije hladno')
```

Unos	Odgovor	Objašnjenje odgovora
Što će se prikazati na zaslonu ako za temperaturu unesemo 10?	a) 10 b) 20 c) Hladno d) Nije hladno e) temperatura	
Što će se prikazati na zaslonu ako za temperaturu unesemo 20?	a) 20 b) Hladno c) Nije hladno d) temperatura e) Ništa od navedenog	
Što će se prikazati na zaslonu ako za temperaturu unesemo 30?	a) 30 b) 20 c) Hladno d) Nije hladno e) temperatura	

3. U sljedećim zadacima pogledaj program pa odgovori što će se ispisati ovisno o unosu podataka.

Koje će sve vrijednosti varijabla `i` promijeniti nakon izvođenja sljedećeg programa? Zaokruži točan odgovor.

	Program	Odgovor	Objašnjenje odgovora
a)	`for i in range (0,6):` `  print (i)`	a) 0 1 2 3 4 5 b) 0 1 2 3 4 5 6 c) 1 2 3 4 5 6 d) 1 2 3 4 5 e) 1 2 3 4 5	
b)	`for i in range (0,6):` `  print (i+1)`	a) 0 1 2 3 4 5 b) 0 1 2 3 4 5 6 c) 1 2 3 4 5 6 d) 1 2 3 4 5 e) 1 2 3 4 5	
c)	`for i in range (0,6):` `  print ('+')`	a) + b) 0 1 2 3 4 5 6 c) + + + + + + d) 5 e) 1 2 3 4 5	

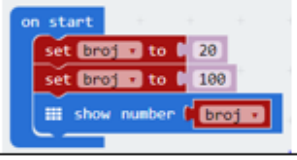
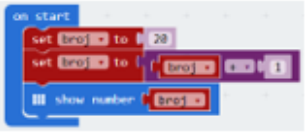
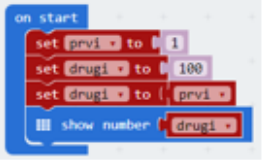
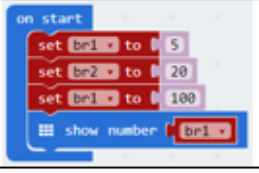
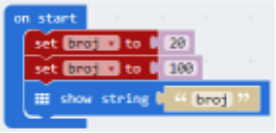
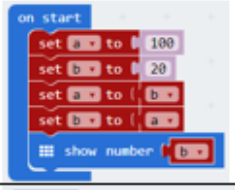
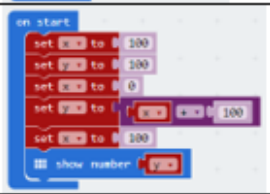
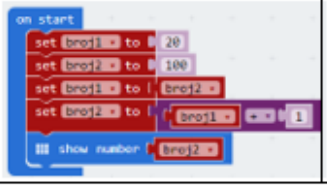
d)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = i`
	`print (zbroj)`
		a) 0
		b) 5
		c) 6
		d) 15
		e) zbroj	

e)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = zbroj + i`
	`print (zbroj)`
		a) 0
		b) 5
		c) 6
		d) 15
		e) zbroj	

Prilog 13 – • Završni ispit znanja u Micro:bitu (ZI-M) (4. istraživanje)

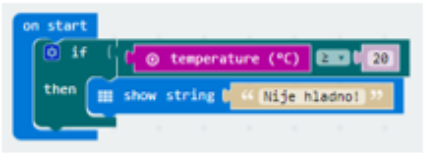
Ime, prezime, razred:_____

1. U tablici ispod je 8 zadataka. Pažljivo pogledaj zadatke pa **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa.

	Program	Odgovor	Prostor za rad
a)		a) 20 b) 100 c) broj d) 120 e) 20 100	
b)		a) 21 b) broj c) broj+1 d) 1 e) 20	
c)		a) 1 b) 100 c) drugi d) 1 100 e) 101	
d)		a) 100 b) 20 c) br1 d) 5 e) 105	
e)		a) 20 b) 100 c) broj d) 120 e) 20 100	
f)		a) 20 b) 100 c) 120 d) a e) b	
g)		a) 0 b) 100 c) 200 d) 1 e) y	
h)		a) 21 b) broj1+1 c) 100 d) 101 e) broj2	

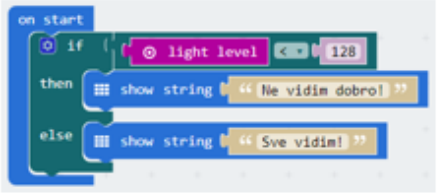
2. Za rješavanje sljedećih zadataka prvo prouči program, a zatim **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat što će se ispisati ovisno o unosu podataka.

a) Napravljen je program u kojem se očitava temperatura. Ako je temperatura barem 20 toplo je i treba ispisati 'Nije hladno!'. Program je prikazan ispod. Prouči program.



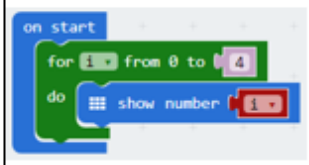
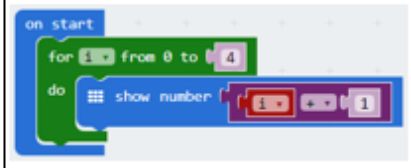
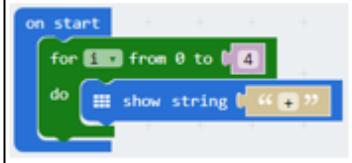
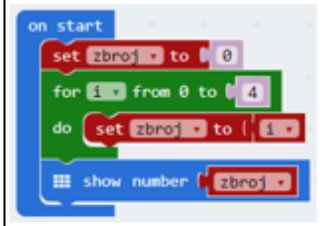
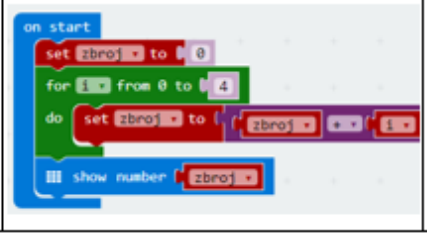
Unos	Odgovor	Prostor za rad
Što će se ispisati na zaslonu ako je izmjerena temperatura 10?	a) 10 b) 20 c) temperature (°C) d) Nije hladno! e) Ništa se neće ispisati.	
Što će se ispisati na zaslonu ako je izmjerena temperatura 20?	a) 20 b) Nije hladno! c) temperature (°C) d) Ništa se neće ispisati. e) Ništa od navedenog.	
Što će se ispisati na zaslonu ako je izmjerena temperatura 30?	a) 20 b) 30 c) temperature (°C) d) Nije hladno! e) Ništa se neće ispisati.	

b) Napravljen je program u kojem se očitava razina svjetla. Ako je očitana razina svjetla manja od 128 treba ispisati 'Ne vidim dobro!', dok za ostale slučajeve treba ispisati 'Sve vidim!'. Program je prikazan ispod. Prouči program.



Unos	Odgovor	Prostor za rad
Što će se ispisati na zaslonu ako je očitana razina svjetla 100?	a) 100 b) 128 c) Sve vidim! d) light level e) Ne vidim dobro!	
Što će se ispisati na zaslonu ako je očitana razina svjetla 128?	a) 128 b) Sve vidim! c) light level d) Ne vidim dobro! e) Ništa se neće ispisati.	
Što će se ispisati na zaslonu ako je očitana razina svjetla 200?	a) 200 b) 128 c) Sve vidim! d) Ne vidim dobro! e) light level	

3. Za rješavanje sljedećih zadataka prvo prouči program, a zatim **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa.

	Program	Odgovor	Prostor za rad
a)		a) 0 1 2 3 4 b) 0 1 2 3 c) i i i i i d) 1 2 3 4 5 e) 1 2 3 4	
b)		a) 0 1 2 3 4 b) 0 1 2 3 c) i+1 i+1 i+1 i+1 i+1 d) 1 2 3 4 5 e) 1 2 3 4	
c)		a) + b) 0 1 2 3 c) + + + + + d) 4 e) 1 2 3 4	
d)		a) 0 b) 4 c) 5 d) 10 e) zbroj	
e)		a) 0 b) 4 c) 5 d) 10 e) zbroj	

Prilog 14 – • Završni ispit znanja u Pythonu (ZI-P) (4. istraživanje)

Ime, prezime, razred:_____

1. U tablici ispod je 8 zadataka. Pažljivo pogledaj zadatke pa **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa. U prostor za objašnjenje pokušaj **ukratko objasniti** svoj odgovor.

	Program	Odgovor	Objašnjenje odgovora

a)	`broj = 100`
	`broj = 20`
	`print (broj)`
		a) 20
		b) 100
		c) broj
		d) 120
		e) 100 20	

b)	`broj = 100`
	`broj = broj+1`
	`print (broj)`
		a) 100
		b) broj
		c) broj+1
		d) 1
		e) 101	

c)	`prvi = 100`
	`drugi = 1`
	`drugi = prvi`
	`print (drugi)`
		a) 1
		b) 100
		c) drugi
		d) 100 1
		e) 101	

d)	`br1 = 100`
	`br2 = 20`
	`br1 = 5`
	`print (br1)`
		a) 100
		b) 20
		c) br1
		d) 5
		e) 105	

e)	`broj = 100`
	`broj = 20`
	`print ('broj')`
		a) 20
		b) 100
		c) broj
		d) 120
		e) 100 20	

f)	`a = 20`
	`b = 100`
	`a = b`
	`b = a`
	`print (b)`
		a) 20
		b) 100
		c) 120
		d) a
		e) b	

g)	`x = 0`
	`y = 0`
	`x = 100`
	`y = x+100`
	`x = 1`
	`print (y)`
		a) 0
		b) 100
		c) 200
		d) 1
		e) y	

h)	`broj1 = 100`
	`broj2 = 20`
	`broj1 = broj2`
	`broj2 = broj1+1`
	`print (broj2)`
		a) 21
		b) broj1+1
		c) 20
		d) 120
		e) broj2	

2. Za rješavanje sljedećih zadataka prvo prouči program, a zatim **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat što će se ispisati ovisno o unosu podataka. Pokušaj **ukratko objasniti** svoj odgovor.

a) Napravljen je program u kojem se unosi razina svjetla. Ako je razina svjetla barem 128 vidi se dobro i treba ispisati 'Sve vidim!'. Program je prikazan ispod. Prouči program.

```
razina_svjetla = int(input())
if razina_svjetla >= 128:
    print ('Sve vidim!')
```

Unos	Odgovor	Objašnjenje odgovora
Što će se ispisati na zaslonu ako za razinu svjetla unesemo 100?	a) 100 b) 128 c) Sve vidim! d) razina_svjetla e) Ništa se neće ispisati.	
Što će se ispisati na zaslonu ako za razinu svjetla unesemo 128?	a) 128 b) Sve vidim! c) razina_svjetla d) Ništa se neće ispisati. e) Ništa od navedenog. Rješenje je:_____	
Što će se ispisati na zaslonu ako za razinu svjetla unesemo 200?	a) 200 b) 128 c) Sve vidim! d) razina_svjetla e) Ništa se neće ispisati.	

b) Napravljen je program u kojem se unosi temperatura. Ako je unesena temperatura ispod 20 treba ispisati 'Hladno', dok za ostale slučajeve treba ispisati 'Nije hladno'. Program je prikazan ispod. Prouči program.

```
temperatura = int(input())
if temperatura < 20:
    print ('Hladno')
else:
    print ('Nije hladno')
```

Unos	Odgovor	Objašnjenje odgovora
Što će se ispisati na zaslonu ako za temperaturu unesemo 10?	a) 10 b) 20 c) Hladno d) Nije hladno e) temperatura	
Što će se ispisati na zaslonu ako za temperaturu unesemo 20?	a) 20 b) Hladno c) Nije hladno d) temperatura e) Ništa se neće ispisati.	
Što će se ispisati na zaslonu ako za temperaturu unesemo 30?	a) 20 b) 30 c) Hladno d) Nije hladno e) temperatura	

3. Za rješavanje sljedećih zadataka prvo prouči program, a zatim **zaokruži slovo** pokraj odgovora koji bi prikazivao točan rezultat izvršavanja programa. Pokušaj **ukratko objasniti** svoj odgovor.

	Program	Odgovor	Objašnjenje odgovora
a)	`for i in range (0,6):` `  print (i)`	a) 0 1 2 3 4 5 b) 0 1 2 3 4 5 6 c) i i i i i i d) 1 2 3 4 5 6 e) 1 2 3 4 5	
b)	`for i in range (0,6):` `  print (i+1)`	a) 0 1 2 3 4 5 b) 0 1 2 3 4 5 6 c) i+1 i+1 i+1 i+1 i+1 i+1 d) 1 2 3 4 5 6 e) 1 2 3 4 5	
c)	`for i in range (0,6):` `  print ('+')`	a) + b) 0 1 2 3 4 5 6 c) + + + + + + d) 5 e) 1 2 3 4 5	

d)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = i`
	`print (zbroj)`
		a) 0
		b) 5
		c) 6
		d) 15
		e) zbroj	

e)	`zbroj = 0`
	`for i in range (0,6):`
	`  zbroj = zbroj + i`
	`print (zbroj)`
		a) 0
		b) 5
		c) 6
		d) 15
		e) zbroj	

4. Za rješavanje sljedećih zadataka prvo prouči program, a zatim **napiši** odgovor koji bi prikazivao točan rezultat izvršavanja programa.

	Program	Napiši odgovor:	Prostor za rad

a)	`br1 = 5`
	`br2 = 100`
	`br1 = 20`
	`print (br1)`		

b)	`prvi = 10`
	`drugi = 20`
	`drugi = prvi`
	`print (drugi)`		

c)	`x = 10`
	`y = 20`
	`x = 0`
	`y = x+10`
	`x = 1`
	`print (y)`		

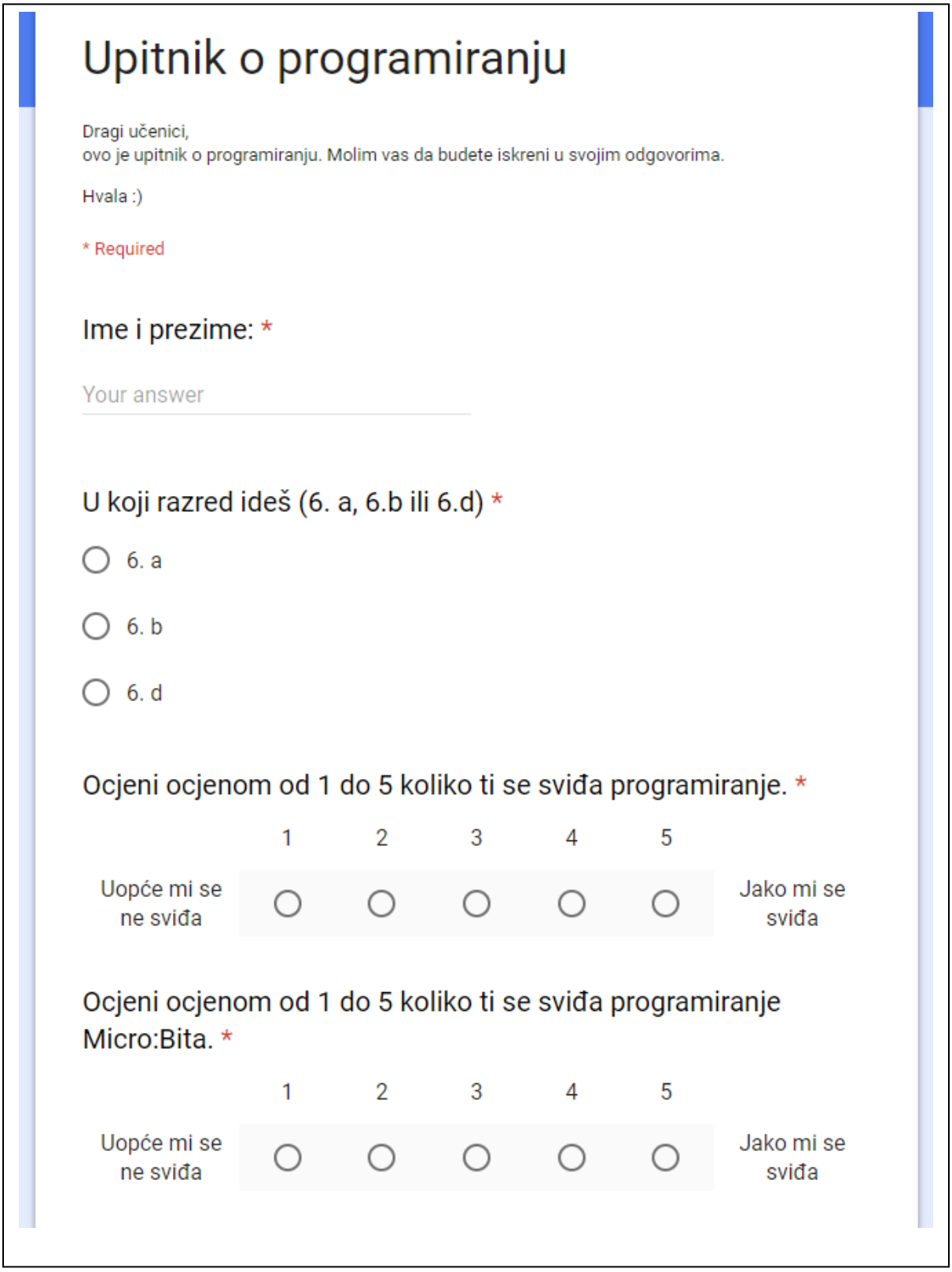
d)	`broj1 = 10`
	`broj2 = 20`
	`broj1 = broj2`
	`broj2 = broj1+10`
	`print (broj2)`		

e)	`for i in range (0,7):` `  print (i)`		
f)	`for i in range (1,4):` `  print (i)`		
g)	`for i in range (0,3):` `  print (i+1)`		

h)	`zbroj = 0`
	`for i in range (1,4):`
	`  zbroj = zbroj + i`
	`print (zbroj)`		

i)	`zbroj = 0`
	`for i in range (0,4):`
	`  zbroj = zbroj + 1`
	`print (zbroj)`		

Prilog 15 – • Anketa zadovoljstva programiranjem, Pythonom, Micro:bitom i programiranjem igara (4. istraživanje)



Upitnik o programiranju

Dragi učenici,
ovo je upitnik o programiranju. Molim vas da budete iskreni u svojim odgovorima.

Hvala :)

* Required

Ime i prezime: *

Your answer

U koji razred ideš (6. a, 6.b ili 6.d) *

- 6. a
- 6. b
- 6. d

Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje Micro:Bita. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje igara (na primjer, kamen-škare-papir) na Micro:Bitu. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

Ocjeni ocjenom od 1 do 5 koliko ti se sviđa programiranje algoritama na Micro:Bitu (zadaci slični onima u Pythonu). *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

Ocjeni ocjenom od 1 do 5 koliko ti se sviđa Python. *

	1	2	3	4	5	
Uopće mi se ne sviđa	○	○	○	○	○	Jako mi se sviđa

Što ti se više sviđa: *

- ○ 1) programiranje igre u Microbitu
- ○ 2) Programiranje matematičkih zadataka u Microbitu
- ○ 3) Programiranje matematičkih zadataka u Pythonu

Koji ti se programski jezik više sviđa? *

- ○ 1) Micro:Bit
- ○ 2) Python
- ○ 3) Podjednako mi se sviđaju

Kojim bi programskim jezikom učio/la programiranje iduće školske godine? *

- ○ 1) Python
- ○ 2) Micro:Bit

Ako imaš neki svoj komentar u vezi programiranja Micro:bita slobodno ga napiši.

Your answer

Ako imaš neki svoj komentar u vezi usporedbe Micro:bita i Pythona slobodno ga napiši.

Your answer

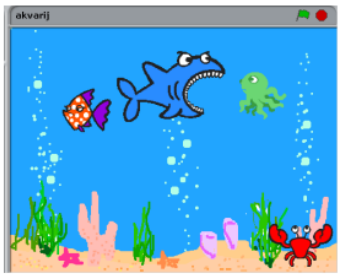
SUBMIT

Prilog 16 – primjer prve lekcije u Scratchu - Akvarij

Akvarij

Napravit ćemo jednostavnu igru akvarij u kojoj će se nalaziti nekoliko likova a morski pas će „jesti“ ribice. Ovo je primjer jednostavne igre prikladne za upoznavanje učenika sa Scratch okruženjem na zanimljiv način. Samu izradu igre poželjno je prikazati kroz izradu scenarija koji zapravo predstavlja algoritam pa tako možemo razraditi scenarij za izradu akvarija.

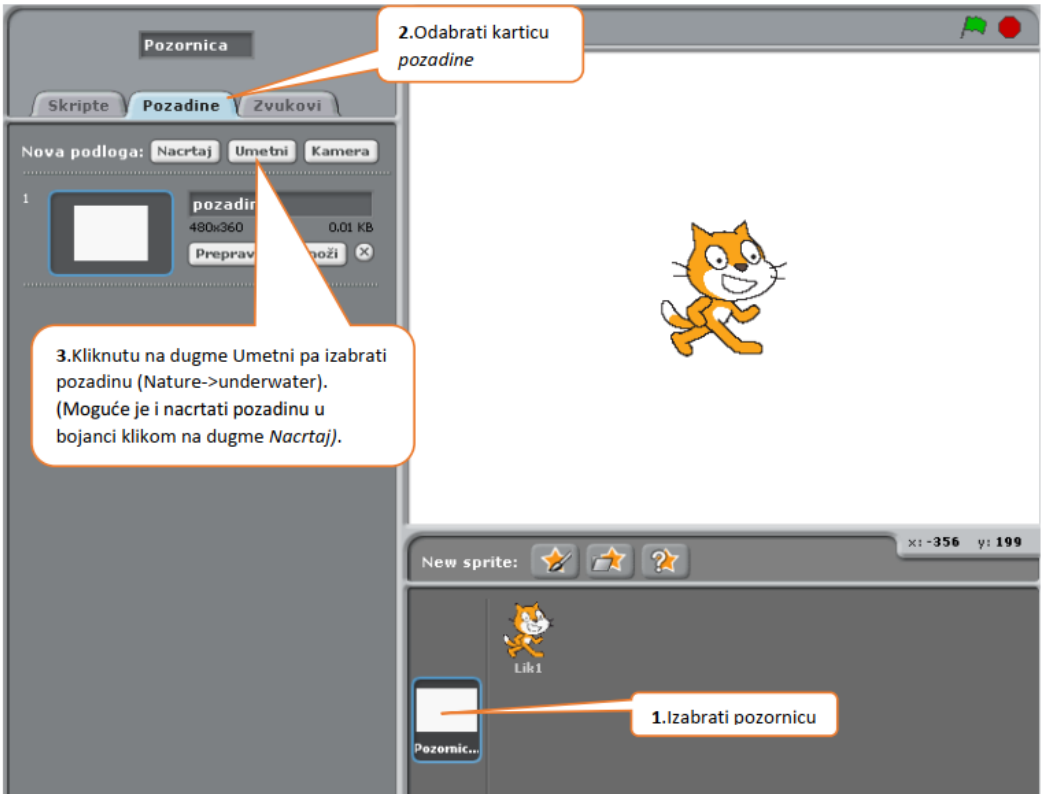
Scenarij:



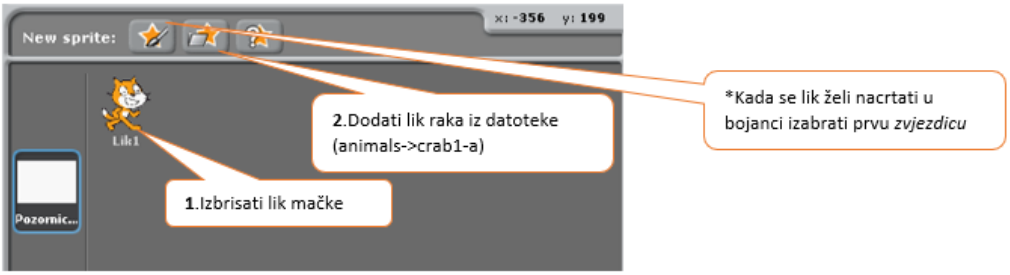
1. Promijeni pozadinu
2. Dodaj lik raka
   a. Uredi kretanje raka
   b. Uredi škljocanje kliješta raka
3. Dodaj lik hobotnice
   a. Uredi kretanje hobotnice
   b. Uredi izgled kretanja hobotnice
4. Dodaj lik morskog psa
   a. Uredi kretanje morskog psa
   b. Uredi otvaranje/zatvaranje usta
5. Dodaj lik ribice
   a. Uredi kretanje ribice
   b. Uredi nestajanje ribice nakon što je dotakne morski pas
6. Umnoži ribice

1. Promjena pozadine

Pozadinu je jednostavni mijenjati ugrađenim pozadinama koje dolaze sa scratchom ili bilo kojom slikom koja je pohranjena na računalu. Slika akvarija se nalazi u ugrađenim pozadinama Scratcha pod *Nature->underwater.*

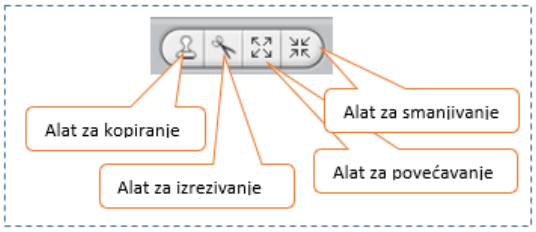


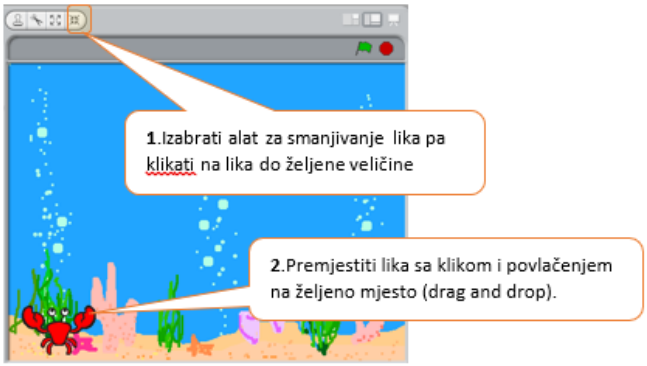
2.Dodavanje likova



Mijenjanje veličine lika

Najčešće je potrebno mijenjati veličinu lika pa tako i raka treba smanjiti i smjestiti na dno akvarija.





Kako ćemo imati više likova dobro je nazivati ih smisleno pa bi našem liku umjesto naziva **Lik1** mogli dati ime **rak**.

2a Kretanje lika

Napravili smo prve dvije točke scenarija a sada je potrebno pokrenuti lika. Upoznali smo naredbe za pokretanje i pa možemo napraviti skriptu.

Pitanja za učenike:

- Što se dogodi nakon izvršavanja skripte?
- Zašto moramo stalno klikati na zastavicu da bi se lik pomaknuo?
- Postoji li neki algoritam kojim bi mogli ponavljati istu naredbu?
- Pogledajte naredbe pod grupom ponavljanje i uočite koju bi naredbu mogli upotrijebiti.
- Koja je razlika između naredbi **ponavljaj** i **ponovi 10**?
- Koja je razlika između sljedećih skripti i koja je naredba ponavljanja nama potrebna?

Koristit ćemo algoritam ponavljanja i beskonačnu petlju.

Pojmovi: ***beskonačna petlja, petlja sa zadanim brojem izvođenja***
Beskonačna petlja uzrokuje beskonačno izvršavanje jedne ili više naredbi.
Petlja sa zadanim brojem izvođenja se izvršava jedne ili više naredbi točno onoliko puta koliko je zadano.

Pitanja za učenike:

- Nakon dodavanje beskonačne petlje kreće li se lik „beskonačno“?

Like se može kreće „beskonačno“ sve dok ima prostora za kretanje, kada dođe do ruba pozornice tada će prestati jer nema više prostora za kretanje. Kako bi nastavi kretanje lik se mora okrenut a tu nam koristi naredba 

Sada skripta izgleda ovako:

Pitanja za učenike:

- Kreće li se lik „normalno“? Što se dogodilo?

Kada lik dođe do ruba i okrene se naredbom ***ako si na rubu, okreni se*** mijenja smjer pa se rak okreće naopako. Taj problem možemo mijenjati bez novih naredbi već podešavanjem lika.

- Rak se kreće brzo. Kako bi mogli usporiti lika, a kako ubrzati?

Postaviti kretanje lika za **2** koraka.

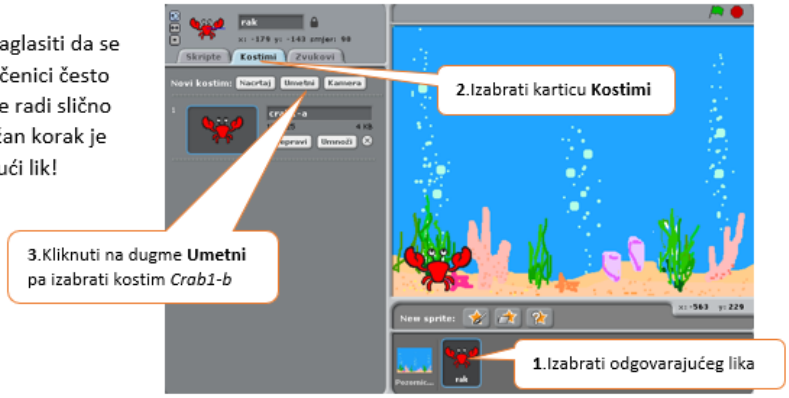
Kako bi zaustavili izvođenje skripte potrebno je kliknuti na stop znak koji se nalazi u gornjem desnom kutu odmah do zastavice za pokretanje. 

2b Mijenjanje kostima

Sljedeći dio scenarija se odnosi na rad kliješta raka. Lik može imati više kostima koji se mogu mijenjati tijekom izvođenja programa. To se može napraviti dodavanjem kostima liku i izradom skripte za izmjenu kostima.

Dodavanje kostima

Prilikom dodavanja kostima naglasiti da se dodaje kostim a ne lik jer to učenici često miješaju. Umetanje kostima se radi slično kao mijenjanje pozadine a važan korak je paziti da je izabran odgovarajući lik!



Sada kada lik ima dva kostima (može ih imati i više) možemo napraviti skriptu za mijenjanje kostima. Iako lik već ima skriptu možemo mu dodati novu. U Scratchu se može više skripti istog lika izvršavati istodobno.

Pojmovi: *istodobno izvršavanje naredbi*
Kada se program izvršava sve skripte lika se **istodobno** izvršavaju.

Pitanja za učenike:

- Pogledajte grupu naredbi izgled i razmislite koju bi naredbu mogli iskoristiti za izmjenu kostima?

Iako se može koristiti naredba ***prebaci na kostim*** zbog jednostavnosti će se koristiti naredba 

Ova naredba je prikladnija jer naš lik mijenja samo dva kostima koja ima.

- Kada ste dodali novu skriptu i naredbu provjerite pokretanjem programa što se dogodilo. Miče li rak kliještima?
- Koliko puta je promijenio kostim?
- Kako bi mogli napraviti da stalno mijenja kliješta?

- Što se sada događa?

Rak prebrzo mijenja kostime pa izgleda kao da se kliješta samo trzaju a neki neće ni primijetiti da se mijenjaju. Nakon naredbe sljedeći kostim potrebno je dati naredbu koja bi napravila da program čeka određeno vrijeme prije sljedeće naredbe.

- Potražite u grupi ***upravljanje*** naredbu koja bi nam mogla pomoći.

Podesiti čekanje na 0.4 sekunde.

Sad kada smo uredili izgled i kretanje raka dodat ćemo i ostale likove.

3 Hobotnica

3a Dodajte lik hobotnice (*animals->octopus1-a*). Lik nazovite **hobotnica** i smanjite je. Napravite da se hobotnica stalno kreće (brzinom 4 koraka), poput raka ali po sredini akvarija. Ne zaboravite podesiti da gleda samo lijevo ili desno da se ne bi okretala naopako.

Pitanje:

- Kreće li se hobotnica „prirodno"? Što bi trebalo promijeniti da se hobotnica kreće u svim smjerovima i koje naredbe bi nam tu mogle pomoći.

Kako bi vidjeli u kojem smjeru se hobotnica kreće uključite opciju smjer u grupi naredbi kretanje. 

Da se hobotnica ne bi kretala samo lijevo ili desno u smjeru 90° (ili -90°) bilo bi dobro na početku izvršavanja skripte promijeniti smjer hobotnice. Tu naredbu bi trebalo izvršiti samo jednom i to na početku skripte. Takav postupak se zove **inicijalizacija**.

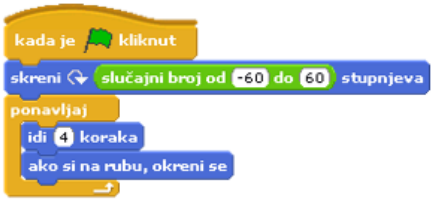
Pojam: *Inicijalizacija*
Inicijalizacija se odnosi na naredbe koje postavljaju neke vrijednosti na počeku programa. To može biti postavljanje lika na određenu poziciju, usmjeravanje lika ili postavljanje varijable na određenu vrijednost na početku programa.

Nama nije važno da se hobotnica kreće baš u nekom određenom smjeru pa nam nije važno .u kojem smjeru i za koliko stupnjeva će hobotnica skrenuti, važno nam je samo da skrene za neki kut. Uzet ćemo naredbu za skretanje u desno i podesit ćemo da skrene za 15°.

Provjerite koje smjerove sada hobotnica poprima. Ako više puta pokrenete program može se dogoditi da se hobotnica opet kreće pod pravim kutom. Kako bi kretanje hobotnice bilo manje predvidivo možemo podesiti da se smjer kretanja hobotnice svaki put izabere slučajnim odabirom (random).

Pojam: *slučajni brojevi*
Korištenjem slučajnog broja program će generirati slučajan broj u rasponu koji korisnik unese. Na taj način možemo dobiti nepredvidivo ponašanje nekih elemenata igre.

Naredba za slučajni broj se nalazi u grupi **operacije**. Postavit ćemo da se slučajni broj bira između -60° i 60° . Umetnut ćemo ga u naredbu za skretanje na mjesto broja stupnjeva.



Provjerite kako se sada kreće hobotnica. Kada zaustavite program uočite u kojem se smjeru nalazi i u kojem se nalazi nakon pokretanja programa. Isključite prikaz smjera hobotnice.

3b Kako bi dobili prirodniji izgled kretanja hobotnice trebali bi joj dodati još jedan kostim Uredite izmjenu kostima hobotnice kao što smo to napravili kod raka. Drugi kostim je *octopus1-b*. Pazite da dodate novi kostim, a ne novi lik!

4 Morski pas

Zadatak:

- Dodajte lik morskog psa (*animals->Shark1-a*). Lika nazovite *pas*. Smanjite veličinu po želji. Podesite da gleda samo lijevo ili desno da se ne bi okretao naopako.
- Uredi kretanje morskog psa tako da se kreće brzinom 3 koraka. Svaki put kada se program pokrene neka morski pas krene u smjeru između 0° i 30° u odnosu na prethodni smjer.
- Uredi izmjenu kostima morskog psa (kostim *Shark1-b*) tako da morski pas otvara i zatvara usta.

5 Ribica

Zadatak:

- Dodajte lik ribice po želji (*fish2*, *fish3* ili *fish4*). Smanjite ribicu tako da bude najmanji lik na pozornici. Ne zaboravite podesiti da gleda samo lijevo ili desno da se ne bi okretala naopako.
- Uredite kretanje ribice tako da se kreće brzinom od 6 koraka. Svaki put neka se pokrene u smjeru između -30° i 30° u odnosu na prethodni smjer.

Za one koji žele više

Ako želimo da ribica nestane kada je dodirne morski pas moramo napravit novu skriptu za ribicu. Dakle:

ako ribica dodirne morskog psa

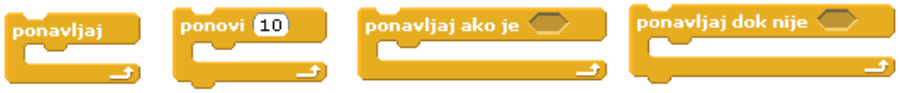
ribica nestaje

Pojam: Uvjet
Uvjet je pitanje na čiji odgovor može biti „Da“ ili „Ne“, odnosno istina ili laž

U našem primjeru uvjet je „ako ribica dodiruje morskog psa“. Odgovor na to pitanje može biti samo da ili ne, odnosno može biti istina ili laž. Kako bi se ovaj uvjet stalno ponavljao potrebno je koristiti petlju. U Scratchu smo spominjali **petlje bez logičkog uvjeta**, a postoje i **petlje sa logičkim uvjetom**.

U našem primjeru uvjet je „ako ribica dodiruje morskog psa“. Odgovor na to pitanje može biti samo da ili ne, odnosno može biti istina ili laž. Kako bi se ovaj uvjet stalno ponavljao potrebno je koristiti petlju. U Scratchu smo spominjali **petlje bez logičkog uvjeta**, a postoje i **petlje sa logičkim uvjetom**.

Pogledajte sljedeće petlje i pokušajte opisati kako svaka od njih radi! Koja petlja nama treba u ovom slučaju?



Uvjetno izvršavanje petlje omogućuje izvršavanje naredbi bez unaprijed definiranog broja izvođenja već do izvršavanja određenog uvjeta.

U našem slučaju potrebna nam je petlja ***ponavljaj ako je*** i za uvjet iz grupe ***očitanja*** umetnuti ***dodiruje*** pas.

Da bi riba nestala potrebna nam je naredba iz skupine izgled.

- Koja naredba nam može pomoći?

Dodajte naredbu sakrij u svoju skriptu pa pokrenite program. Kada ribica nestane zaustavite pa ponovo pokrenite program. Što se dogodilo?

Ribi smo dali naredbu da se sakrije ali nigdje da se pokaže. Znači da je potrebno inicijalizirati ribu na početku programa da se pokaže.

Provjerite je li sada sve u redu sa ponašanjem ribe. Ako je umnoži ribu i napravi 10 kopija iste ribice. To radimo desnim klikom na lika i izborom opcije umnoži.

Igra je gotova.

Prilog 17 – Primjeri zadataka u Micro:bitu

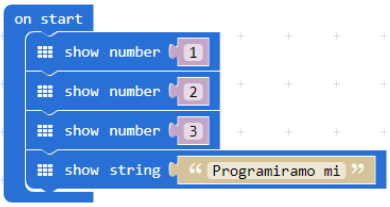
ALGORITAM SLIJEDA

Zadatak: programiranje

Algoritam: **slijeda**

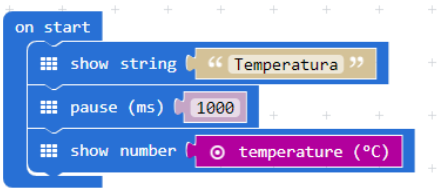
- Ispiši broj **1**
- Ispiši broj **2**
- Ispiši broj **3**
- Ispiši tekst „**Programiramo mi**”

```
print(1)
print(2)
print(3)
print('Programiramo mi')
```



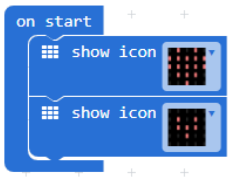
Zadatak: temperatura

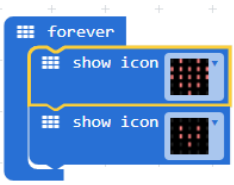
Izradi program koji će nakon što se uključi Micro:bit na zaslonu ispisati tekst „Temperatura“ . Nakon jedne sekunde neka prikazuje trenutnu temperaturu prostorije.



Zadatak: Srce

Limeni iz zemlje Oz je tužan jer nema srce. Napravite program koji će napraviti Limenom srce koje jednom zakuca. Kako bi napravili da mu srce stalno kuca?





Koja je razlika između dvije skupine naredbi?

Što je ispravno i zašto?

ALGORITAM PONAVLJANJA

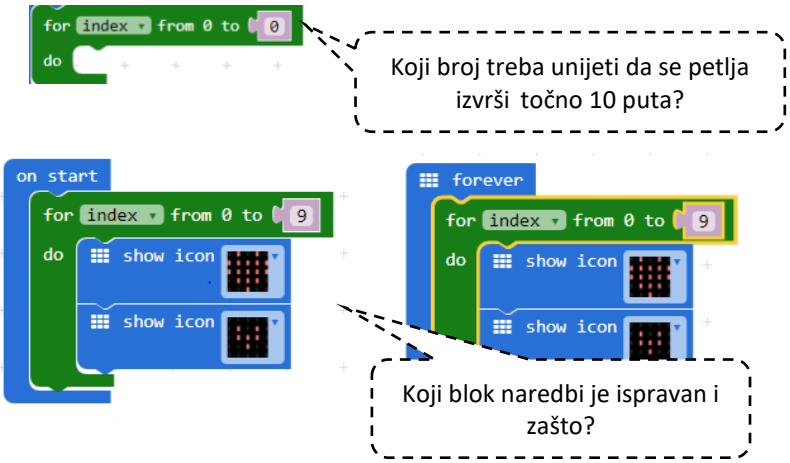
Zadatak: Srce

Limeni iz zemlje Oz je tužan jer nema srce. Kako nije naučio na srce on bi da mu točno 10 puta zakuca.

Algoritam ponavljanja (petlja)

Koja petlja se koristila u Pythonu? `for i in range(0,10):`

Pogledajte grupu naredbi *Loops*



Varijabla

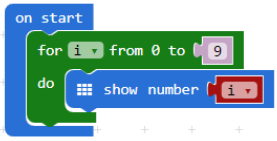
Kako smo najčešće nazivali takvu varijablu u Pythonu?

Možemo i u Micro:bitu raditi varijable (grupa naredbi Variables)

Napravite novu varijablu *i* pa koristite je u petlju umjesto varijable *indeks*

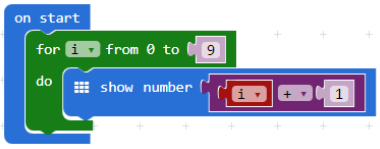
Štoperica

Napravite program na Micro:bitu koji će ispisivati brojeve od 0 do 9.




```
for i in range(0,11):
    print (i)
```

Prepravi program tako da ispisuje brojeve od 1 do 10.

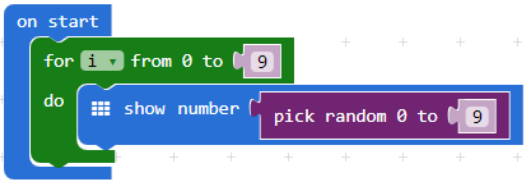


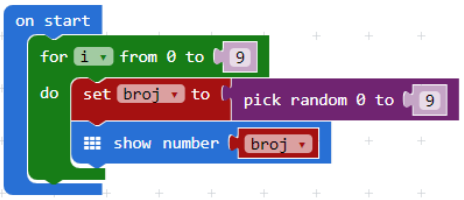

```
for i in range(0,11):
    print (i+1)
```

Slučajni brojevi

Napravi program koji će ispisivati 10 slučajno izabranih brojeva od 0 do 9 jedan za drugim.

Nova naredba: grupa naredbi Math-> Pick random 0 to …





ALGORITAM GRANANJA

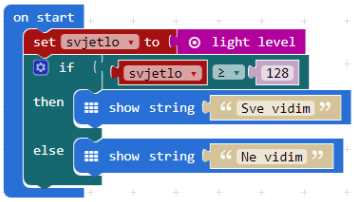
Svjetlo

Napravi program koji će učitavati razinu svjetla i zapamtiti je u varijabli svjetlo pa će ovisno o razini svjetla ispisivati:

Ako je razina svjetla barem 128 neka se ispiše „Sve vidim”

Inače neka ispiše „Ne vidim!”

```
svjetlo = int(input())
if svjetlo >= 128:
    print ('Sve vidim!')
else:
    print('Ne vidim!')
```



Temperatura

Sobnom temperaturom se smatra 24oC jer nam je tada najugodnije.

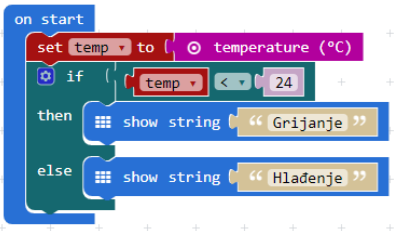
Napravi program koji će učitati sobnu temperaturu i zapamtiti je u varijabli temp pa ispisati:

Ako je temperatura ispod 24oC ispisati „Hladno je”

Inače neka ispiše „Nije hladno”

Kako bi takav program napravili u Pythonu?

```
temp=int(input())
if temp<24:
    print('Grijanje')
else:
    print('Hlađenje')
```



Ponekad nam je potrebno ispitati više od jednog uvjeta (kao sada)

I kod višestrukog grananja uvijek će se izvršiti najviše jedan uvjet.

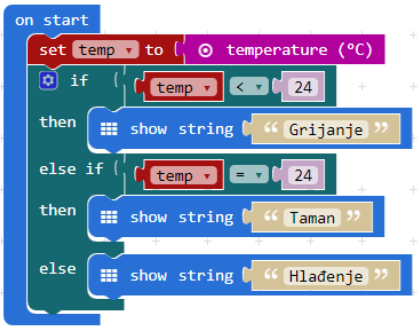
U svim programskim jezicima postoje naredbe kojima možemo ispitivati više uvjeta.

Ako je temperatura ispod 24°C ispisati „Hladno je”

Inače ako je temperatura točno 24°C onda ispisati „Taman”

Inače neka ispiše „Nije hladno”

```
temp=int(input())
if temp<24:
    print('Grijanje')
elif temp==24:
    print('Taman')
else:
    print('Hlađenje')
```

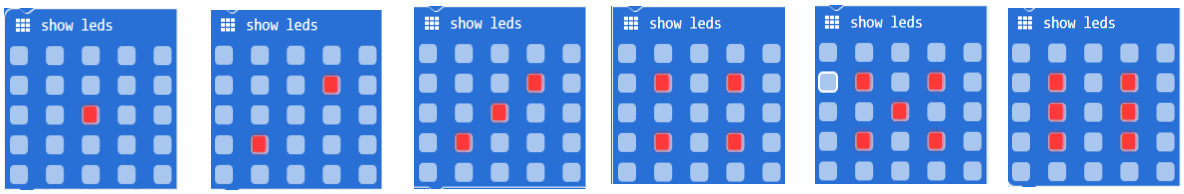


Zadatak: kockica

Pri bacanju kockice ne znamo unaprijed koji ćemo broj dobiti.

Kako možemo dobiti slučajan broj u Micro:bitu?

Kako možemo napraviti ispis znakova kao na kockici?



1 2 3 4 5 6

ŽIVOTOPIS I POPIS JAVNO OBJAVLJENIH RADOVA

Monika Mladenović je rođena 08. kolovoza 1979. u Splitu gdje je završila osnovnu i srednju školu. Na Veleučilištu u Splitu je završila studij računarstva te stekla zvanje inženjerke računarstva. Na Sveučilištu u Splitu pri Sveučilišnom studijskom centru za Stručne studije 2009 godine je stekla zvanje stručne prvostupnica (baccaleurea) inženjerka Informacijske tehnologije. Na Sveučilištu u Splitu pri Prirodoslovno-matematičkom fakultetu je 2011. godine je stekla zvanje Magistra edukacije informatike. Ima više od 16 godina staža na različitim poslovima: Stručni suradnik u nastavi na Veleučilištu u Splitu, Programer u tvrtki Ecsat d.o.o., djelatnica informatičke službe Sveučilišnog studijskog centra za stručne studije, viši stručni suradnik-programer u KBC-u Split. Od 2011. do 2017. je radila u nekoliko osnovnih škola u Splitu: OŠ Split 3, OŠ Blatine-Škrape, OŠ Spinut i OŠ Bol. Od 2011. do 2017 je radila kao vanjski suradnik na Odjelu za informatiku pri Prirodoslovno-matematičkom fakultetu a od 2017. godine je zaposlena kao Asistent za znanstveno područje tehničkih znanosti, polje računarstvo na Prirodoslovno-matematičkom fakultetu u Splitu.

ZNANSTVENI RADOVI

Izvorni znanstveni i pregledni radovi u CC časopisima

Using Games to Help Novices Embrace Programming: From Elementary to Higher Education. // International journal of engineering education. 32 (2016) , 1B; 521-531 (članak, znanstveni)

Znanstveni radovi u drugim časopisima

Žanko, Žana; Mladenović, Monika; Boljat, Ivica. Misconceptions about variables at the K-12 level. // Education and Information Technologies. 24 (2019), 2, 1251-1268 (članak, znanstveni).

Mladenović, Monika; Boljat, Ivica; Žanko, Žana. Comparing loops misconceptions in block-based and text-based programming languages at the K-12 level. // Education and Information Technologies. 23 (2018) , 4; 1483-1500 (članak, znanstveni)

Mladenović, Monika; Krpan, Divna; Mladenović, Saša. Learning programming from Scratch. // The Turkish Online Journal of Educational Technology. 2 (2017) ; 419-427 (članak, znanstveni).

Maleš, Lada; Mladenović, Monika; Mladenović, Saša. Znaju li studenti prve godine što je internet?. // Školski vjesnik : časopis za pedagoška i školska pitanja. 65 (2016) ; 105-117 (članak, znanstveni). URL link to work

Mladenović, Monika; Rosić, Marko; Mladenović, Saša. Comparing Elementary Students' Programming Success based on Programming Environment. // I.J. Modern Education and Computer Science. 8 (2016) , 8; 1-10 (članak, znanstveni).

Znanstveni radovi u zbornicima skupova s međunar.rec.

Mladenović, Monika; Krpan, Divna; Mladenović, Saša. Introducing programming to elementary students novices by using game development in python and scratch // EDULEARN16 Proceedings.2016. 1622-1629 (međunarodna recenzija,objavljeni rad,znanstveni).

Mladenović, Saša; Žanko, Žana; Mladenović, Monika. Elementary Students' Motivation Towards Informatics Course // Procedia - Social and Behavioral Sciences. Elsevier, 2015. 3780-3787 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

Bubica, Nikolina; Mladenović, Monika; Boljat, Ivica. Students motivation for competition in computer science // 8th International Technology, Education and Development Conference (INTED2014) : proceedings.

Valencia : International Academy of Technology, Education and Development IATED, 2014. 288-295 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

Mladenović, Monika; Žanko, Žana; Rosić, Marko. Elementary students' attitude towards programming in the Republic of Croatia // Proceedings of CIET 2014 / Plazibat, Bože ; Kosanović, Silvana (ur.). Split : University of Split, 2014. (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

Žanko, Žana; Mladenović, Monika; Mladenović, Saša. Students attitude towards informatics curricula // ICERI2014 Proceedings. Seville, Spain : ICERI, 2014. 5785-5785 (predavanje,međunarodna recenzija,objavljeni rad,znanstveni).

Drugi radovi u zbornicima skupova s recenzijom

Bubica, Nikolina; Mladenović, Monika; Boljat, Ivica. Programiranje kao alat za razvoj programskog mišljenja // 15. CARNetova korisnička konferencija - CUC 2013 - Zbornik radova / Orlović, Ana (ur.). Zagreb : Hrvatska akademska i istraživačka mreža - CARNet, 2013. (predavanje,domaća recenzija,objavljeni rad,znanstveni).

Stručni radovi, skupovi, predavanja:

Mladenović Monika, „Vizualno blokovsko programiranje“, radionica na Županijskom stručnom vijeću učitelja informatike Splitsko-dalmatinske županije u Splitu, 28.11.2018.

Mladenović Monika, „Početno programiranje za osnovce u Scratchu“, radionica na konferenciji Suvremene tehnologije u obrazovanju – STO, 2017.

Mladenović Monika, „Programski jezici u osnovnim školama“, međužupanijski stručni skup za učitelje/nastavnike pripravnike Informatike/Računalstva i njihove mentore, Prirodoslovno matematički fakultet u Splitu 2017

Mladenović Monika, „Programiranje u Pythonu“, međužupanijski stručni skup za učitelje/nastavnike Informatike/Računalstva iz Dubrovačko-neretvanske, Splitsko-dalmatinske i Šibensko-kninske županije, Prirodoslovno matematički fakultet u Splitu 2015.

Mladenović Monika „Poučavanje programiranja kroz izradu i igranje računalnih igara“, međužupanijski stručni skup za učitelje/nastavnike Informatike/Računalstva iz Dubrovačko-neretvanske, Splitsko-dalmatinske i Šibensko-kninske županije, Prirodoslovno matematički fakultet u Splitu 2015.

Mladenović Monika, „Pedagoška upotreba različitih pedagoških alata za poučavanje programiranja“ Info@edu2, državni Stručni skup za učitelje i nastavnike informatike i računalstva osnovnih i srednjih škola na području Republike Hrvatske, Vodice 2013 (javno predavanje)