Prihvaćanje i korištenje Scratcha od strane početnika u učenju programiranja

Martinović, Ana

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

PRIHVAĆANJE I KORIŠTENJE SCRATCHA OD STRANE POČETNIKA U UČENJU PROGRAMIRANJA

Ana Martinović

Split, ožujak 2022.

SVEUČILIŠTE U SPLITU
PRIRODOSLOVNO MATEMATIČKI FAKULTET

DIPLOMSKI RAD

PRIHVAĆANJE I KORIŠTENJE SCRATCHA OD STRANE POČETNIKA U UČENJU PROGRAMIRANJA

Studentica:

Ana Martinović

Mentor:

doc. dr. sc. Nikola Marangunić

Split, ožujak 2022.

Temeljna dokumentacijska kartica

Sveučilište z Splitu
Prirodoslovno-matematički fakultet
Odjel za informatiku
Ruđera Boškovića 33, 21000 Split, Hrvatska

Diplomski rad

PRIHVAĆANJE I KORIŠTENJE SCRATCHA OD STRANE POČETNIKA U UČENJU PROGRAMIRANJA

Ana Martinović

Računalno programiranje razvija vještine rješavanja problema i logičkog zaključivanja i njegova integracija u sve obrazovne razine je nužna. Međutim učenje programiranja nije jednostavno i predstavlja izazov i za učenike i za nastavnike. Zbog svoje jednostavnosti Scratch se koristi kao uvod u programiranje, a računalne vještine koje učenici razviju u Scratchu mogu se kasnije primijeniti u drugim programskim jezicima. U ovom diplomskom radu istraživala se povezanost četiri varijable TAM modela, percipirane korisnosti, percipirane lakoće korištenja, stava prema korištenju i namjere korištenja programskog jezika Scratch, kako bi se otkrio stav i namjera korištenja Scratcha među početnicima u programiranju. Prema kvantitativnim rezultatima može se jasno reći da su svi učenici voljeli programiranje i željeli poboljšati svoje programiranje. Rezultati su, između ostalog, pokazali i kako učenici smatraju Scratch jednostavnim za korištenje i korisnim, ali ipak ne dovoljno „ozbiljnim“ za učenje drugih tekstualnih programskih jezika.

Ključne riječi: Model prihvaćanja tehnologije (TAM), vizualno programiranje, strategije poučavanja, srednjoškolski učenici, računalno razmišljanje

Rad je pohranjen u knjižnici Prirodoslovno-matematičkog fakulteta, Sveučilišta u Splitu

Rad sadrži: 54 stranica, 15 grafičkih prikaza, 13 tablica i 44 literaturnih navoda. Izvornik je na hrvatskom jeziku.

Mentor: Dr. sc. Nikola Marangunić, docent

Ocjenjivači: Dr. sc. Nikola Marangunić, docent

Dr. sc. Divna Krpan, docent

Dr. sc. Jelena Nakić, docent

Rad prihvaćen: 17. ožujak 2022.

Basic documentation card

University of Split
Faculty of Science
Department of Computer Science
Ruđera Boškovića 33, 21000 Split, Croatia

Diploma Thesis

PERCEIVED ACCEPTANCE AND USE OF SCRATCH AMONG BEGINNERS OF LEARNING PROGRAMMING

Ana Martinović

Computer programming develops problem-solving and logical reasoning skills and its integration into all levels of education is essential. However, learning programming is not easy and is a challenge for both students and teachers. Because of its simplicity, Scratch is used as an introduction to programming, and the computer skills that students develop in Scratch can later be applied in other textual based programming languages. In this thesis, the relationship between four variables of the TAM model, perceived usefulness, perceived ease of use, attitude towards use and intentions to use the Scratch was investigated to reveal the attitude and intention to use Scratch among beginners in programming. According to the quantitative results, it can be clearly said that all students loved programming and wanted to improve their programming. The findings showed, that students find Scratch easy to use and useful, but still not "serious" enough to prepare them for „real programming“ in text based programming languages.

Keywords: Technology Acceptance Model (TAM), visual programming, learning strategies, high school students, computational thinking

Thesis deposited in the library of Faculty of Science, University of Split

Thesis consists of: 54 pages, 15 figures, 13 tables and 44 references, original in: Croatian

Mentor: Nikola Marangunić, Ph.D. *Assistant Professor*

Reviewers: Nikola Marangunić, Ph.D. *Assistant Professor*

Divna Krpan, Ph.D. *Assistant Professor*

Jelena Nakić, Ph.D. *Assistant Professor*

Thesis accepted: March 17th 2022

IZJAVA

kojom ja, Ana Martinović, studentica Prirodoslovno-matematičkog fakulteta Sveučilišta u Splitu, kao autorica diplomskog rada s naslovom: Prihvaćanje i korištenje Scratcha od strane početnika u učenju programiranja:

Izjavljujem da sam diplomski rad izradila samostalno pod mentorstvom doc. dr. sc. Nikole Marangunića. U radu sam primijenila metodologiju istraživačkog rada i koristila literaturu koja je navedena na kraju rada. Tuđe spoznaje, stavove, zaključke, teorije i zakonitosti koje sam izravno ili parafrazirajući navela u radu citirala sam i povezala s korištenim bibliografskim jedinicama sukladno odredbama Pravilnika o diplomskom radu Prirodoslovno-matematičkog fakulteta Sveučilišta u Splitu.

Rad je pisan u duhu hrvatskog jezika.

Sadržaj

Uvod

Kao profesor informatike u srednjoj strukovnoj školi često se susrećem sa pitanjem zašto moram učiti programirati. Programiranje je svojevrsna „teretana za mozak“ koja razvija niz tehničkih, kognitivnih i socijalnih vještina. Učenje programiranja nije isključivo namijenjeno samo budućim programerima već bi sva djeca u školi trebala imati priliku učiti programirati kako bi ih pripremili za poslove budućnosti, obavljanje zadataka i rješavanje problema u digitalnom okruženju.

Učenje programiranja nije jednostavno i predstavlja izazov i za učenike i za nastavnike [1]. Glavni izazovi početnika u programiranju su osmišljavanje rješenja problema, odnosno izrada algoritma, te potom “prevođenje“ napisanog algoritma na programski jezik. Usvajanje algoritamskih struktura i krute sintakse programskih jezika zahtjeva visoku razinu apstraktnog razmišljanja, te vrijeme i trud. Kako bi učenike motivirali na usvajanje takvog znanja, učitelji moraju prilagoditi kontekst programiranja interesima učenika koji su okruženi tehnologijom i odrastaju uz nju. Na motivaciju učenja programiranja utječe i odabir prvog programskog jezika. Jezik za poučavanje programera početnika mora biti intuitivan, bez kompliciranе sintakse i zbunjujuće semantike.

Scratch je vizualni programski jezik osmišljen za početno učenje programiranja. Prvenstveno je usmjeren na djecu i tinejdžere s namjerom prenošenja računalnog razmišljanja i fokusiran na naglašavanje aspekata rješavanja problema [2]. Alat ima intuitivno sučelje i djeca bez predznanja iz programiranja mogu jednostavno stvarati interaktivne igre i priče, razvijati maštu i dijeliti svoje projekte.

U Republici Hrvatskoj, prema novom kurikulumu od školske godine 2018/2019, predmet Informatika uveden je kao obavezan predmet za pete i šeste razrede osnovne škole, dok učenici sedmih i osmih razreda biraju žele li upisati Informatiku kao izborni predmet. U srednjoj školi, ovisno o tipu škole, Informatika je obavezan predmet u barem jednom razredu. Provedba predmeta Informatika kao izbornog u završnim razredima osnovne škole ima za posljedicu heterogenost učenika po znanju i sposobnostima u srednjoj školi, pa se tako u istom razredu mogu nalaziti učenici bez predznanja programiranja kao i učenici koji su već „iskusni programeri“. U razredu od 24 učenika u prosjeku polovica razreda je odabrala informatiku sve četiri godine tijekom osnovnoškolskog obrazovanja. Na pitanje

što misle o programiranju i kako su se snalazili, odgovor je gotovo uvijek isti, programiranje je teško.

Istraživanje u ovom radu provedeno je među učenicima prvih i drugih razreda četverogodišnje strukovne škole kojima je „Rješavanje problema pomoću računala“ jedna od nastavnih cjelina u izvedbenom godišnjem kurikulumu. Prije uvoda u proceduralni programski jezik Python, implementiran je Scratch kako bi se učenici na jednostavan i zabavan način upoznali s osnovnim principima programiranja, razvojem algoritama i rješavanjem problema. Cilj istraživanja je izmjeriti prihvaćanje vizualnog programskog jezika u srednjoj školi analizirajući utječu li percipirana korisnost, stav prema korištenju i percipirana lakoća korištenja na namjere učenika za stvarnom upotrebom Scratcha. Istraživanje je utemeljeno na modelu prihvaćanja tehnologije (eng. Technology Acceptance Model, TAM).

Istraživačka pitanja na kojima se temelji ovaj rad bila su:

1) U kojoj mjeri percipirana korisnost utječe na stav učenika da koristi Scratch programiranje?

2) U kojoj mjeri percipirana lakoća korištenja utječe na namjeru učenika da koristi Scratch programiranje?

3) U kojoj mjeri stav prema korištenju Scratcha utječe na namjeru učenika da koristi Scratch?

1. Programiranje

Živimo u digitalnom dobu i računala su dio naše svakodnevnice. Djeca tehnologiju prihvaćaju na „prirodan“ način, nemaju strah od nje i većinu slobodnog vremena provode uz tehnologiju. Vrijeme koje provode uz tehnologiju treba iskoristiti i usmjeriti na učenje. „Najbolji način za to je učenje programiranja od najranije dobi, ali na način koji je djeci „prirodan“.„[3]. Programirati znači komunicirati i upravljati računalom, stvarati vlastite sadržaje, rješavati probleme, ispravljati pogreške. Programirati danas znači biti pismen [4].

U svojoj knjizi „Mindstorms: Children, Computers and Powerful Ideas“ Papert ističe kako djeca kroz aktivnost programiranja uče računalo kako razmišljati pri tome otkrivajući i kako sami razmišljaju. Seymour Papert bio je jedan od vodećih autora Logo jezika. Pod utjecajem svog mentora Jean Piageta istraživao je nove metode poučavanja i adekvatne tehnologije kao sredstvo kognitivnog razvoj djeteta. Vjerovao je kako računalni sadržaji mogu pomaknuti granice koje razdvajaju konkretnu od formalne faze i time ubrzati stvaranje kognitivnih struktura kod djece ranije dobi. Prema Papertu, Logo je prijateljski jezik jer se komunikacija djece i „kornjače“ odvija upotrebom jednostavnih i lako razumljivih engleskih riječi. Komunikacija sa računalom kod djece stvara osjećaj kontrole i izgrađuje samopouzdanje. Mogućnost vizualizacije razvoja programskog koda omogućava i vizualno praćenje vlastitih kognitivnih aktivnosti, te pomaže u razvoju apstraktnog načina razmišljanja [5].

Programiranjem se razvija algoritamski i modularni način razmišljanja. Svaki problem i zadatak bez obzira na svoju veličinu i kompleksnost može se razložiti na manje i lakše zadatke koji se mogu jednostavno riješiti. Vježbanjem modularnog razmišljanja razvijamo strategije za rješavanje problema koje se mogu primijeniti i u drugim životnim područjima.

Programiranje uključuje planiranje, jezičnu preciznost, stvaranje i testiranje različitih hipoteza, predviđanje kao i mnoštvo drugih vještina koje reflektiraju deduktivno razmišljanje [6]. Različitim istraživanjima je dokazano da programiranje razvija specifične vještine pojedinaca, kao prevođenje problema zadanog prirodnim jezikom na način prikladan za zapisivanje programskim jezikom, zadavanje jasnih uputa, zaključivanje analogijom i generalizacijom, planiranje i kreativnost. Rješavanjem kognitivno zahtjevnih

zadataka učenik razvija meta-kognitivne sposobnosti koje predstavljaju temelj uspješnosti u daljnjem školovanju.

„Zato programiranje nije samo vještina koju je poželjno svladati već alat koji nas uči kako učiti“ [3].

1.1. Problemi u procesu učenja programiranja

„Pitanje zašto je programiranje učenicima teško je univerzalno i neovisno o državi i njenom obrazovnom sustavu te o uzrastu učenika ili odabranom programskom jeziku“ [7].

Programiranje je višeslojna vještina. Učenici moraju usvojiti algoritamski način rješavanja problema, razumjeti način na koji računalo radi, imati predodžbu o tome kako će se program izvršavati itd. Početnik u programiranju nema izgrađen model računala tj. ne poznaje princip rada računala i njegove mogućnosti [8]. „Problemi za mnoge učenike počinju već u ranoj fazi učenja kada trebaju riješiti konkretne probleme iz stvarnog svijeta na algoritamski način“ [9]. Tan i sur. zaključili su da je mogući uzrok problema s učenjem programiranja nedostatak jasno izgrađenih mentalnih modela memorije računala. Većina studenata imala je problem s konceptima vezanim za memoriju kao što su pohrana i manipulacija varijablama u glavnoj memoriji računala [10]. Milne i Rowe (2002) zaključili su da mnogi studenti nisu bili sposobni razviti jednostavan mentalni model tijeka izvođenja programa [11].

Još jedan razlog poteškoća programera početnika je i sintaksa programskih jezika koja nije jednostavna. Mnoge pogreške i zablude u programiranju nastaju kao posljedica prijelaza s prirodnog na programerski jezik [12]. Sintaksa programskog jezika je skup pravila i simbola koji programeru omogućuju pisanje ispravno strukturiranih programa. Često su mješavina engleskog jezika i različitih simbola, kao što su točke sa zarezom, zagrade i drugi elementi koji zahtijevaju određenu preciznost. Programski jezik nije prirodan i ne funkcionira na način na koji ljudi razmišljaju. Izražavanje osmišljenih rješenja kao programa zahtijeva poznavanje sintakse i „uvježbano oko“ pa mnogi početnici u programiranju budu obeshrabreni upravo krutom sintaksom.

U poučavanju programiranja često se koriste tradicionalni oblici nastave. Lekcije su fokusirane na dijelove programskog koda i sintaksu programskog jezika. U procesu rješavanja zadataka koriste se dijelovi programskog koda usvojeni kroz teorijski dio. Problem nastaje kod usvajanja i kombiniranja tih dijelova u smislenu cjelinu jer učenik

kroz lekcije ne usvaja općenite strategije primjene programiranja. Oni se više usredotočuju na rješavanje malih dijelova koda, a ne na planiranje i testiranje cjelovitog rješenja. Isto tako, učenici ne predstavljaju homogenu skupinu s obzirom na sposobnosti i različite stilove učenja. Mnogi istraživači u području edukacije smatraju stilove učenja važnim faktorom procesa učenja. Učinkovitost u učenju je veća kod učenika čiji je stil učenja u skladu s načinom prezentacije sadržaja kojeg uče.

Dio problema može ležati i u već usvojenom znanju učenika. Vrlo dobar primjer su simboli + i = koji imaju različita značenja u matematici i programiranju. Znak = u programiranju označava pridruživanje, a u matematici predstavlja znak jednakosti. U IF izjavama znak = označava uspoređivanje što stvara dodatnu zbunjenost kod učenika. Kognitivne poteškoće programera početnika stvara i brojanje od nula jer brojanje u realnom životu započinje od jedan i intuitivno je jasno da nulu ne treba brojati. Za stvaranja dobrih temelja za razvoj ispravnih koncepata bitno je na vrijeme otkriti učeničke miskoncepcije (učenički koncept koji nije u skladu sa znanstvenom spoznajom) i suočiti ih s njima [13].

Usvajanje vještine programiranja značajno ovisi i o motivaciji učenika. Programiranje zahtijeva dugotrajan rad, disciplinu i strpljivost. Svaki problem koji se rješava programiranjem, pokretanjem napisanog programa može raditi ili ne. Programiranje zahtijeva kontinuirano vježbanje no ne mora značiti da će svi imati jednak uspjeh s obzirom na broj uloženih sati. Svaki pojedinac pokušava objasniti uspjeh ili neuspjeh pridodavanjem odnosno atribucijom koji mogu biti unutarnji ili vanjski, kontrolirani ili ne. U situaciji učenja odnosno poučavanja važno je pomoći učeniku pri razvoju vlastitog objašnjenja truda. Ako osoba ima atribucijsku sposobnost onog trenutka kada osjeti da ima poteškoće u procesu učenja, smanjit će prikladno ponašanje u učenju. Početnički neuspjeh može rezultirati padom motivacije i odustajanjem od daljnjeg ulaganja truda.

1.2. Edukacijske teorije u poučavanju programiranja

Poučavanje u računalnoj znanosti zasniva se na općim edukacijskim teorijama. U nastavku rada opisane su teorija obrade informacija, teorija mentalnih modela i konstruktivizam u kontekstu unapređenja poučavanja programiranja.

1.2.1. Teorija obrade informacija

Učenjem stječemo novo znanje o konceptima predmetnog područja, međusobnom odnosu tih koncepata te njihovoj primjeni u rješavanju problema. Tijekom učenja programiranja učenik usvaja znanje o svojstvima programskog jezika, razvija vještine oblikovanja programa te stječe vještinu rješavanja problema. Cilj učenja programiranja je stjecanje vještine prijenosa stečenog znanja pri učenju novih formalnih sustava.

Kognitivni ili spoznajni stil je način na koji pojedinac opaža, pamti, misli i rješava probleme. Kognitivni stil pojedinca je ujedno i njegov stil učenja. Kognitivni sustav dijeli se na kratkotrajnu ili radnu memoriju ograničenog kapaciteta i dugotrajnu memoriju vrlo velikog kapaciteta.

Ljudi primaju nove informacije iz vanjskog svijeta i one djeluju na jedan ili više osjetilnih sustava. Senzorno pamćenje omogućuje identifikaciju oblika, predmeta i pojava. Tu se informacije u nepromijenjenom obliku zadržavaju kratko vrijeme od pola do dvije sekunde. Ako informacije nisu proslijeđene u radnu memoriju nakon prestanka djelovanja podražaja one nestaju.

Radna memorija predstavlja drugu fazu pamćenja koja se odvija na svjesnoj razini. Kapacitet radne memorije iznosi 7+/-2 nepovezane informacije. Grupiranjem, smislenim povezivanjem i prikladnim sažimanjem moguće je i taj kapacitet znatno povećati. U radnoj memoriji informacije se kodiraju i organiziraju, donose se odluke i oblikuju reakcije. Ako su pristigle informacije vrijedne i korisne na duži rok, slijedi njihovo organiziranje u semantičke sklopove koji se zatim ponavljanjem pohranjuju u dugotrajnoj memoriji.

U dugotrajnoj memoriji nalazi se pohranjeno mnoštvo informacija koje čine bazu našeg znanja i iskustva uopće. Dugotrajna memorija ima neograničen kapacitet i gotovo sve informacije koje čine naše znanje ostaju tu tokom cijelog života. Zaborav se očituje u nemogućnosti njihova pronalaska. Kod smislenog pamćenja informacije se organiziraju u semantičke sklopove, a zatim ponavljanjem prenose i integriraju u odgovarajuće mreže informacija koje se već nalaze u dugotrajnoj memoriji. Kod mehaničkog pamćenja informacije se u izvornom obliku, neobrađene, ponavljanjem prenose u dugotrajno pamćenje.

U dugotrajnoj memoriji od većeg je značaja smislena organizacija informacija koja omogućuje njihovu prikladnu pohranu. Procesom smislenog učenja, učenik dolazi u

kontakt sa sadržajem o kojem ne posjeduje znanje u svojoj dugotrajnoj memoriji. Proces započinje prihvaćanjem novih informacija u radnoj memoriji nakon čega slijedi pretraživanje dugotrajne memorije radi pronalaženja starih informacija s kojima može povezati nove informacije. Postojeće znanje se aktivira prelaskom iz dugotrajne memorije u radnu te njegovim kombiniranjem s novim informacijama. Ako nove i stare informacije imaju bar jedno zajedničko obilježje, stare informacije prenose se u radnu memoriju. U radnoj memoriji vrednuju se mogućnosti povezivanja starih informacija po njihovim obilježjima. Neke stare informacije se odbacuju ako su neprikladne za ispravno povezivanje, dok se ispravne kombiniraju sa dolazećim informacijama. Kad ne postoji znanje prikladno za povezivanjem s novim, smisleno učenje nije moguće pa učenik mehanički pamti nove informacije i takvo znanje postaje izolirano u njegovoj dugotrajnoj memoriji. U nekim se situacijama može ostvariti i pogrešno povezivanje novog i postojećeg znanja, dajući novim informacijama značenje kakvo nemaju. Do pogrešnog povezivanja može doći zbog podudaranja nekih manje bitnih obilježja starih i novih informacija, dok se značajna obilježja razlikuju. Uloga učitelja je osmisliti i provesti proces učenja koji će omogućiti ispravno povezivanje starog i novog znanja, ali i otkriti pogrešno znanje i ostvariti preduvjete ispravka.

Poznavanje teorije obrade informacija može pomoći učenicima u boljem upravljanu svojim učenjem, ali isto tako može pomoći učiteljima u izradi boljih i učinkovitijih nastavnih materijala.

1.2.2. Teorija mentalnih modela

Prvi istraživač koji je upotrijebio pojam mentalni model bio je Craik 1943. godine [14], ali je za daljnji razvoj i unapređenje teorije mentalnih modela zaslužno rođenje kognitivne znanosti. Prema teoriji mentalnih modela ljudi koriste mentalne reprezentacije (modele) kako bi strukturirali informacije o kojima zaključuju. Proces zaključivanja ovisi o tri faze zaključivanja. U prvoj fazi, fazi razumijevanja, ljudi koriste svoje opće znanje i znanje o jeziku kako bi konstruirali mentalni model. U drugoj fazi se izvodi zaključak iz mentalnog modela stvorenog u prethodnoj fazi. U trećoj fazi pokušavaju se konstruirati alternativni modeli koji negiraju zaključak. Zaključak je ispravan ukoliko nema alternativnih modela.

Konstrukcija mentalnih modela je ograničena kapacitetom radne memorije te je broj mentalnih modela koje pojedinci stvaraju obično malen.

Učenici u pravilu ne pristupaju nastavi programiranja kao „tabule rase“ već posjeduju određeno znanje s kojim mogu povezati nove informacije. Istraživanje mentalnih modela koje učenici posjeduju također je važno za pripremu učitelja i oblikovanje nastavnih materijala.

Teorija mentalnih modela naslanja se na teoriju konstruktivizma čijim se pristupom izazivaju postojeće ideje i uvjerenja, a s ciljem stvaranja održivih mentalnih modela.

1.2.3. Konstruktivizam

Konstruktivizam je teorija učenja koja predstavlja oprečnost mehaničkom učenju i pasivnom usvajanju znanja. Učenik samostalno i aktivno gradi razumijevanje, spoznaju i znanje na već postojećem znanju. Kako bi nove informacije dobile smisao učenici moraju stvarati veze između novih i starih informacija. Prema Piagetovoj teorija kognitivnog razvoja proces učenja odvija se kroz asimilaciju tj. prihvaćanje novih informacija u postojeće strukture znanja i akomodaciju tj. prilagođavanje postojećih struktura znanja s novim informacijama

Učitelj je osoba koja induktivnim metodama vodi učenika kroz njegov proces učenja. Uloga učitelja je pripremiti okolinu koja potiče samostalni rad i učenje i u potpunosti odgovara potrebama svakog učenika. U svom istraživanju konstruktivizma, Ben-Ari je predložio smjernice praktične primjene konstruktivizma u nastavi programiranja [8]:

- učitelji moraju artikulirati spoznajne promjene koje žele postići kod učenika i strukturirati nastavne aktivnosti i metode s ciljem postizanja tih promjena,
- učitelji moraju biti svjesni vlastitog prethodnog znanja na kojem su gradili stručnost jer učenici moraju imati iste temelje na koje će se kasnije nadograđivati složeniji sadržaji,
- nastava mora biti organizirana na način da učenicima osigurava dovoljno prilika za osobnu refleksiju (analizu grešaka) i učenje kroz socijalnu interakciju.

Konstruktivizam u poučavanju programiranja mora uzeti u obzir dvije karakteristike koje se ne pojavljuju u prirodnim znanostima [8]:

1. Učenik nema razvijen mentalni model računala. Ako učenici nemaju vlastiti model o nekoj temi, učitelj mora osigurati razumijevanje hijerarhije modela. To znači da

se model računala, CPU, memoriju, I/O jedinice moraju eksplicitno naučiti i raspravljati o njima, a ne ih ostaviti slučajnoj konstrukciji.

2. Grafičko korisničko sučelje se često uči kao intuitivno i pristupačno ali mnogi korisnici ga nerado uče. Ikone bi trebale biti intuitivne i analogija prikazanog objekta i objekta koji ikona zapravo predstavlja, bi trebala biti savršena. Međutim, mnoga istraživanja dokazuju upravo suprotno i mnogi početnici imaju problema sa shvaćanjem što se zapravo događa unutar računala kad kliknemo na tu ikonu. Ben-Ari navodi primjer ikone „paste“. Prvo, ikona bi trebala predstavljati riječ „zalijepi“. Drugo, riječ „zalijepi“ bi trebala biti povezana sa operacijom „umetni kopiju materijala koja se trenutno nalazi u među-spremniku na mjesto označeno kursorom u trenutno otvorenom dokumentu.“ Kako bi korisnik prepoznao ovu operaciju ona mora imati izgrađeni mentalni model koji omogućava razumijevanje četiri različita koncepta iz prethodne rečenice. Ikona je vizualna prezentacija koja mnogo ne doprinosi izgradnji ispravnog mentalnog modela.

Mnogi fenomeni u informatičkom obrazovanju mogu biti objašnjeni s konstruktivizmom. Frustracija i pretpostavka da je informatika teška pridonosi činjenici da model mora biti vlastito izgrađen iz početka. Rad na računalu može biti ne ohrabrujući za učenike koji vole više socijalni stil učenja. Često učenici započinju prerano pisati programe što vodi do niza pokušaja i pogreški. Učenik koji isključivo koristi metodu “probaj pa ćeš vidjeti što će se dogoditi“ ne stvara održivi mentalni model. Stvaranje održivih mentalnih modela u programiranju zahtjeva sposobnost kreiranja i testiranja različitih apstraktnih hipoteza.

1.3. Strategije poučavanja programiranja

Učenje programiranja kompleksna je aktivnost koja značajno ovisi o kognitivnim sposobnostima pojedinca. Programer početnik nema prethodno programersko znanje, ali poznaje područja iz kojih problemski zadaci dolaze. Početnik bi trebao znati riješiti zadani problem na papiru, ali ne posjeduje znanja i vještine oblikovanja algoritma čijim izvršavanjem se rješava isti problem. Prepoznavanje pojmova na koje se fokusiraju programeri početnici pomoći će učiteljima odrediti pravilan tempo nadogradnje njihovog znanja.

Neki učitelji računalne znanosti, pogotovo neiskusni učitelji, smatraju kako je dobro imati popis poteškoća sa kojima se učenici često susreću u nastavi programiranja kao i strategije

koje mogu koristiti za podučavanje tih sadržaja i izbjegavanje istih. Prema njima, imati znanje o poteškoćama moglo bi promijeniti način na koji podučavaju. Na primjer, mogu birati vježbe zbog kojih će se poteškoće pojaviti tijekom nastave i tako da mogu pružiti podršku učenicima u rješavanju problema. Pravilnim pristupom organiziranju nastave može se poboljšati pedagoško znanje za bolje poučavanje i posljedično bolje usvajanje programiranja [15].

Mnoga istraživanja bave se utjecajem rasporeda nastavnih sljedova na znanje koje su početnici usvojili. Istraživanja su pokazala da učenje od osnovnog prema apstraktnom ima najveću učinkovitost te najnižu ocjenu složenosti. U poučavanju programiranja prijelaz od jednostavnog prema kompleksnijem treba biti postupan. Osnovne koncepte je potrebno temeljito usvojiti prije prelaska na složenije probleme.

Ujedno, svaka tema programiranja usko je povezana. Učenici se mogu izgubiti u sljedećoj temi ako dobro ne savladaju prethodnu temu. Programeri početnici moraju provesti više vremena vježbajući osnovna znanja i jedino kontinuirana praksa može pomoći učenicima zadržati znanje.

Bez motivacije, nastava programiranja je osuđena na neuspjeh. Motivaciju u nastavi potrebno je stalno pratiti i sagledati sve faktore koji utječu na njen smjer, dinamiku i karakter. Posebnu pažnju potrebno je posvetiti učenicima i njihovim unutrašnjim čimbenicima koji ih pokreću na aktivno i stvaralačko sudjelovanje u nastavnom procesu. Sastavni dio motivacijskog ciklusa čine i drugi faktori kao što su program nastave, organizacijska forma nastave i strategije vođenja u učenju i poučavanju. Suvremeni oblici vođenja nastave idu u smjeru detaljnog preispitivanja sadržaja koje će učenici proučavati te inzistiraju na što većem njihovom angažmanu u rješavanju konkretnih nastavnih pitanja. Na taj način moguće je djelovati na faktore od kojih zavisi učenička motiviranost. Učitelj je najvažniji motivacijski faktor u nastavi jer je odgovoran za njene tokove i ishode. Postoji niz različitih aktivnosti koje učitelj može primijeniti kako bi povećao motivaciju u razredu. Za povećanje intrinzične motivacije učitelj treba objasniti zašto je usvajanje određenog sadržaja važno, stvoriti ili održati znatiželju, omogućiti različite aktivnosti i poticaje, postaviti ciljeve učenja, povezati učenje i potrebe učenika te im pomoći pri razvoju plana djelovanja. Predstavljanjem jasnih očekivanja, davanjem potpore i vrijednih nagrada učitelj povećava ekstrinzičnu motivaciju. Općenito, učitelji trebaju koristiti što više intrinzičnih prijedloga. Znajući da neće svi učenici biti adekvatno motivirani njima, treba imati na umu da će ekstrinzični prijedlozi biti uspješni samo dok su učenici pod kontrolom učitelja.

1.3.1. Kognitivni konflikt

Strategija kognitivnog konflikta temelji se na konstruktivizmu i teoriji mentalnih modela i danas predstavlja jednu od ključnih strategija poučavanja. Teorija kognitivnog konflikta prvi put je opisana u knjizi „Teorija kognitivne disonance“ Leona Festingera 1957. godine. „Ljudi preferiraju dosljednost, slaganje i ravnotežu između svojih kognitivnih elemenata jer im to daje osjećaj razumnosti i predvidljivosti.“ [16]. Ukoliko dođe do nesklada, osoba osjeća psihološku neugodu koja djeluje kao motivirajuće stanje i potreba za smanjivanjem neslaganja primjenom različitih strategija.

Primjenom ove strategije u poučavanju izaziva se učenika da samostalno uočava pogreške u svom razumijevanju. Kognitivni konflikt je stanje u kojem učenik uočava razlike i neslaganja između dijelova njegove kognitivne strukture ili kognitivne strukture i vanjskih informacija. Uočavanjem pogrešaka u razumijevanju učenik mijenja pogrešne mentalne modele i time održava dosljednost između kognitivnih elemenata. U ovom slučaju, učenje nije samo dodavanje novog znanja na postojeće znanje već uključuje promjenu postojećeg znanja.

Ponekad učenici mogu novim informacijama dati drugo značenje koje nije u skladu sa znanstvenim konceptima. Utvrđeno je da su postojeće kognitivne strukture učenika često u sukobu sa znanstvenim. Učenik može zadržati i nastaviti koristiti postojeće znanstveno neutemeljene mentalne modele pri interpretaciji novih informacija. Postojeće koncepcije otporne su na promjene i učenicima nije jednostavno odbaciti svoje ideje i usvojiti nove. Isto tako, učenici često nisu ni svjesni pogrešnih koncepcija [17]. Stoga je važno koncipirati poučavanje na način koji učenicima pomaže prvo otkriti pogrešne koncepcije, a potom izgraditi nove znanstveno utemeljene. Maier [18] naglašava, “….najbolji način da se riješe ili spriječe miskoncepcije je direktno suočavanje učenikovih miskoncepcija s iskustvom koje uzrokuje neravnotežu praćenu akomodacijom.“

Strategija kognitivnog konflikta zahtijeva ispitivanje učenikovih prethodnih znanja, davanje kontradiktornih informacija s ciljem izazivanja promjene te procjenu promjene između prethodnih uvjerenja i trenutnih.

1.3.2. Kognitivno naukovanje

Naukovanje je pojam ukorijenjen u obrtništvu ili trgovini i u ovoj strategiji poučavanja se koristi kao metafora za poučavanje simboličkih umnih vještina kroz vođeno iskustvo [19].

Tijekom duge povijesti učenje i poučavanje se isključivo baziralo na naukovanju. Djeca su učila govoriti, izrađivati robu ili raditi u polju. Nove vještine i znanja usvajali su prvo promatrajući svoje roditelje ili odrasle sudionike zajednice, a potom su pod njihovim nadzorom uvježbavali te iste vještine.

Istraživanjem prednosti koje ima poučavanje naukovanjem, Brown i Collins (1989) dodavanjem riječi kognitivno približili su pojam naukovanje modernim temama kao što su čitanje, pisanje i matematika. Riječ naukovanje naglašava usmjerenost kognitivnog naukovanja na poučavanje načina na koji stručnjaci obavljaju složene zadaće. Riječ kognitivno naglašava fokus na razvijanju kognitivnih vještina i procesa umjesto fizičkih.

Kao strategija koja naglašava angažman učenika umjesto pasivnog učenja, kognitivno naukovanje je našlo veliku primjenu u poučavanju programiranja. Učitelj formulira problem, gradi misaonu strukturu, uvježbava učenike, pomaže u radu, potiče na razmišljanje te zadaje dodatne zadatke koji učenike vode do rješenja početnog problema. Učenici u autentičnim okolnostima promatraju rad nastavnika, te prakticiraju i otkrivaju vlastite vještine.

Faze poučavanja su modeliranje (engl. *modeling*), skaliranje (engl. *scaffolding*) te nestajanje (engl. *fading*). U fazi modeliranja učitelj pruža učenicima model procesa kako stručnjak obavlja zadani zadatak. Na primjer, u poučavanju programiranja učitelj prilikom rješavanja zadatka programiranjem glasno razmišlja i objašnjava odluke donesene u postupku razmišljanja. Verbalizacijom razmišljanja daje primjer učenicima na koji način pristupiti rješavanju određenog problema. Nakon faze modeliranja slijedi faza skaliranja. U ovoj fazi učenici rješavaju zadatak, a učitelj promatra njihovu izvedbu. Učitelj pomaže učenicima pružajući im naznake i kratke informacije koje ih navode na ispravno rješenje. U ovoj fazi izuzetno je važan način pomaganja i učitelj se treba suzdržavati od davanja jasnih odgovora. Uz učiteljevo vođenje učenici u ovoj fazi uspijevaju izvršiti kompleksne zadatke koji su još uvijek izvan dometa njihovih sposobnosti. Kako učenici stječu potrebne vještine za rješavanje kompleksnih zadataka i približavaju svoju izvedbu učiteljevoj, napušta se faza skaliranja i prelazi se u fazu nestajanja.

Povezanost ove tri faze pomaže razvijanju vještina i znanja karakterističnih za iskusnog programera. Faza modeliranja uvodi učenika u domenu programiranja, dok faza modeliranja i nestajanja pomažu učeniku „prerasti“ ovisnost o učiteljevoj pomoći.

1.3.3. Vizualizacija

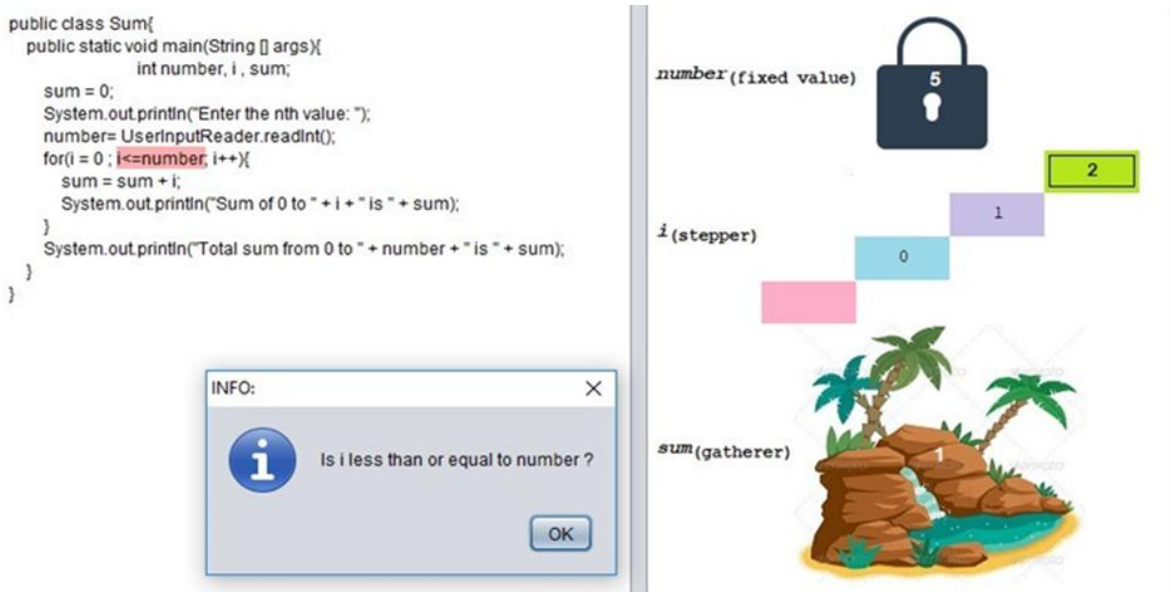
Snažan edukacijski utjecaj u nastavi programiranja ima upotreba alata vizualizacije. Algoritmi i strukture podataka su osnovni građevni blokovi računalnih procesa i tehnika. Algoritam predstavlja skup operacija koje se izvode kako bi se postigao određeni cilj ili izvršio zadatak. To su dinamični procesi, procesi koji se mijenjaju i statične slike ne mogu prenijeti ovu promjenu kao što to mogu animirane slike.

Vizualizacijom možemo grafički ilustrirati različite koncepte u računalnoj znanosti. Općenito, možemo reći da vizualizacija omogućava konkretiziranje značenja nekog apstraktnog koncepta. Vizualizaciju možemo klasificirati [20] na algoritamsku vizualizaciju (dinamična vizualizacija algoritama koji se implementiraju) i programsku vizualizaciju (dinamična vizualizacija trenutno implementiranih programa).

Vizualizacija olakšava učenje jer pomaže u stvaranju mentalnih modela apstraktnih pojmova. Ako pretpostavimo da ljudska spoznaja nije u potpunosti verbalna već i vizualna, onda bi vizualna iskustva trebala biti učinkovita u poticanju učenja. Nadalje, kako mnogi fenomeni nisu samo vizualni već i dinamički, animacije bi također trebale biti učinkovita pomoć u učenju. Kako bi sustav bio učinkovit, učitelj mora poznavati mentalne modele učenika te mentalne modele koji trebaju biti produkt poučavanja. Ovo je teško proizvesti jer mentalni modeli učenika nisu jednaki, a ciljevi učenja nisu predstavljeni kao željeni mentalni modeli nego u terminima „pokrivanja materijala“ [21].

Uloga koju algoritamska vizualizacija ima je osiguravanje „neograničene“ količine podataka nad kojima učenici mogu gledati izvršenje algoritma. Primjenom vizualizacija učenici bi trebali razumjeti opće principe (i ulogu) algoritama i objasniti kako radi, prepoznati bitna svojstva određenih koncepata, implementirati algoritam koristeći neki programski jezik i testirati, razumjeti ponašanje algoritma u najboljem i najgorem slučaju. Osim učenikovog razumijevanja primjenom vizualizacija može se mjeriti i učenikov napredak, vrijeme učenja i zadovoljstvo učenika.

Tijekom posljednjih 17 godina istraživanja su pokazale da programeri početnici imaju problema u razumijevanju uloge varijable i njenog položaja u memoriji računala tijekom izvršavanja programa. Grafički prikazi varijabli povezani s kulturnim okruženjem učenika pomažu im vizualizirati izvođenje programa u memoriji [22].



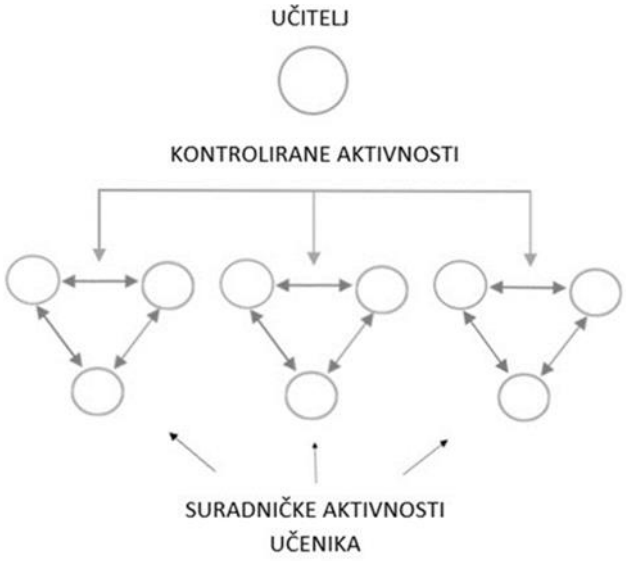
Slika 1.1 - Izvođenje programa "Zbroj"

Slika 1.1 prikazuje primjer sustava „uloga varijable“ koji omogućuje učenicima promatrati mijenjanje vrijednosti varijable tijekom izvođenja programa. U svakom algoritmu svaka se varijabla grafički prikazuje prema svojoj ulozi. Drugim riječima, dvije varijable koje imaju istu ulogu biti će predstavljene istom ikonom ili slikom kako bi učenici lakše razumjeli da varijable moraju "igrati" specifične uloge za postizanje određenog rezultata. Program "Zbroj" izračunava ukupni zbroj od broja 0 do drugog broja "n", koji upisuje korisnik. Svrha programa „Zbroj“ je pomoći početnicima u boljoj vizualizaciji kako petlja djeluje i kako varijable međusobno djeluju kako bi stvorile smisleni ishod. Jednom kada korisnik odabere primjer programa i klikne na gumb "Pokreni", ključne riječi iz programa se istaknu. Nakon što se istakne svaka ključna riječ, pojavljuje se informativna poruka koja objašnjava što je ključna riječ. Izraz "i <= broj" je istaknut i poruka objašnjava što to znači. Broj ili trenutne vrijednosti varijabli prikazane su ikonom u svakoj točki tijekom izvođenja programa. Izlaz i ulaz programa se mogu vidjeti u području teksta ispod područja animacije.

1.3.4. Suradničko učenje

Konstruktivistički proces učenja najbolje funkcionira u socijalnom okruženju jer učenici imaju mogućnost uspoređivati i dijeliti svoje ideje sa drugima. Suradničko učenje kroz male grupne aktivnosti ili razredne diskusije daje učenicima priliku artikulirati svoje znanje i učiti od ostalih. Poticanje učenika na međusobnu suradnju i razmjenjivanje

informacija osnažuje osjećaj zajedništva i pripadnosti što pridonosi razvoju samopouzdanja, kritičkog mišljenja, komunikacijskih i organizacijskih sposobnosti.



Slika 1.2 Model suradničkog učenja i poučavanja

Korijen suradničkog učenja je u teoriji učenja koju je razvio Lev Vygotsky. On razvija koncept "zona proksimalnog razvoja" koji je ključan za razvoj složenih vještina (Slika 1.3). Prema toj teoriji složeni koncepti (poput vještine programiranja) se razvijaju kroz zajedničke socijalne aktivnosti prije nego što se razviju u vještine koje se mogu samostalno primijeniti.

Slika 1.3 Zona idućeg (proksimalnog razvoja)

Kako je suradničko učenje u biti bilo kakvo učenje koje uključuje dva ili više sudionika u tom procesu ono obuhvaća mnoštvo različitih tehnika s brojnim varijacijama.

Najpoznatija i najčešće korištena suradnička tehnika u učenju programiranja je programiranje u paru. Programiranje u paru je jedan od temelja metodologije ekstremnog programiranja. Ona je nastala kao odgovor na krizu razvoja softvera korištenjem klasičnih razvojnih metodologija, a s ciljem ubrzanja razvoja softvera kroz iterativni proces [23]. Ova tehnika programiranja vrlo brzo se proširila i u obrazovni kontekst. Kod programiranja u paru dva učenika razvijaju program koristeći jedno računalo. Jedna osoba je "driver" i on oblikuje i unosi kod. Druga osoba je "navigator" i on kontrolira oblikovanje algoritma i ispravlja greške u njemu. Međusobnom komunikacijom učenici u paru razmjenjuju znanje, raščlanjuju problem i brže dolaze do rješenja. Učenje nastaje u situaciji kad učenici pokušavaju razriješiti konfliktne ideje. Istraživanja o njenom korištenju u učenju programiranja pokazuje brojne pozitivne efekte u kvaliteti koda, učinkovitosti, zadovoljstvu i brzini, ali i ne pokazuje značajno poboljšanje u ocjenama [24]. Programiranje u paru najčešće je korištena i najviše istraživana tehnika suradničkog učenja u programiranju.

U kontekstu učenja programiranja korištena je i tehnika suradničkog podučavanja (engl. *peer instruction*). To je pedagoška tehnika u kojoj studenti najprije pojedinačno odgovaraju na pitanja. Nakon odgovaranja slijedi rasprava koju učitelj vodi sa cijelim razredom. Studenti imaju pozitivan stav prema ovoj tehnici koja unapređuje razumijevanje koncepata iz programiranja, te studenti na testiranju imaju bolje rezultate iz gradiva koje su učili na taj način [25].

Obrnuta učionica (engl. *flipped classrom*) je jedna od najpopularnijih metoda poučavanja u novije vrijeme. Obrnuta učionica spada u tehnike suradničkog učenja i svoju uspješnu primjenu našla je i u poučavanju programiranja. Ideja obrnute učionice je samostalno upoznavanje sa gradivom kod kuće, a u školi slijedi primjena naučenog. Učenici imaju pristup video i drugim materijalima prije predavanja, a u učionici kroz praktični rad primjenjuju usvojene koncepte uz učiteljeve smjernice. Obrnuta učionica naslanja se i na strategiju multimedijalnog poučavanja. Upotreba multimedijskih resursa uključuje prezentacije slike, video zapise i *tutorijale* koji su vrlo korisni za razumijevanje apstraktnih ideja na mnogo lakši način. Aktivno učenje odvija za vrijeme nastave, a pasivni dio se može odraditi samostalno. U istraživanju utjecaja obrnute učionice u nastavi programiranja [26] pratio se razvoj percepcije studenata kroz kvalitativne povratne informacije. Studenti su u velikoj mjeri podržali upotrebu ove strategije poučavanja koja se održala do kraja semestra. Mali broj studenata je smatrao da pristup nije učinkovit, te izrazio želju za

komponentom predavanja dok je većina studenata ipak prepoznala potrebu za kratkim predavanjima u nastavi, posebno za naprednije programske teme. Istraživanjem je zaključeno da obrnuta učionica unapređuje sposobnost kritičkog razmišljanja i razvija samopouzdanje za primjenu računalnog programiranja kao alata i izvan učionice.

1.3.5. Rješavanje problemskih zadataka

Poučavanje tehnikom rješavanja problema može značajno unaprijediti izvedbu programiranja [27]. Rješavanje problema je jedna od centralni aktivnosti koju obavljaju iskusni programeri kao i oni koji tek uče programiranje. Međutim, programeri početnici se često suočavaju s poteškoćama prilikom analize problema i konstruiranja rješenja.

Rješavanja problema možemo podijeliti u faze razumijevanja problema, razmatranja alternativnih načina rješavanja problema, odabir načina rješavanja problema, dekompozicija problema na manje dijelove, pisanje algoritma prema navedenim pod-zadacima, provjeravanje ispravnosti algoritma, određivanje efikasnosti algoritma te razmišljanje i analiza prethodnih procesa.

„Teške“ faze nalaze se između razumijevanja problema i njegovog rješenja. Faze koje se nalaze između mogu se promatrati kao proces otkrivanja, pa se procesi rješavanja problema ponekad tretiraju kao kreativni procesi (umjetnost).

Razumijevanje problema je prva faza rješavanja problema i može početi prepoznavanjem ulaza i odabirom izlaza. Identifikacija ulaza zapravo ukazuje na razumijevanje problema.

Dekompozicijom problema na manje probleme stvaramo pregled strukture problema i odnosa između njegovih dijelova. Bistrenje korak po korak je od-gore-prema-dolje (engl. *top-down*) metodologija jer se odvija od općeg prema specifičnom. Alternativni pristup je od-dole-prema-gore (engl. *bottom-up*) metodologija koja se odvija od specifičnog prema općem. Glavna razlika leži u smjeru mentalnih procesa koji vode proces izrade rješenja.

Pisanje algoritama počinje prepoznavanjem sličnih problema čije je rješenje već poznato, prilagodbom uzorka za rješenje trenutnog problem te njegovim integriranjem u rješenje. U većini slučajeva, potrebno je kombinirati različite uzorke za razvoj potrebnog rješenja. Kada su suočeni s nepoznatim problemom, učenici često ne znaju kako početi rješavati problem i doživljavaju poteškoće u prepoznavanju sličnosti između problema i prenošenju ideja s prije riješenih probleme na nove [28]. Učenicima dodatan problem stvara i ne prepoznavanje biti problema i određivanju njegovih dijelova i odnosa među njima. U

takvim slučajevima učenici često iznova razviju rješenje, počevši od nule. Te poteškoće obično nisu rezultat nekih miskoncepcija, već mogu biti posljedica loše organizacije algoritamskog znanja.

Nakon što je rješenje konstruirano, treba provjeriti njegovu ispravnost. Slično mnogim konceptima računalne znanosti, provjeravanje ispravnosti rješenja ima i teorijske i tehničke aspekte. U srednjoj školi dovoljno je, u većini slučajeva, ispitati ispravnost rješenja s tehničkog aspekta (s reprezentativnim ulazima).

Jedna od povezanih aktivnosti s provjeravanjem rješenja je ispravljanje pogreški (engl. *debugging*). Proces ispravljanja pogrešaka može poboljšati učenikovo razumijevanje programiranja.

Razmišljanje se odnosi na preispitivanje i analizu metoda prethodnih mentalnih procesa. Razmišljanje je važan alat u procesu učenja općenito i pripada kognitivnim procesima višeg reda, kao što je na primjer proces rješavanja problema. Razmišljanje pruža učenicima priliku učiniti korak natrag i razmišljati o vlastitom razmišljanju i na taj način unaprijediti vlastite vještine rješavanja problema.

Kombiniranje strategija obrnute učionice i problemskog rješavanja zadataka omogućava upotrebu informacijske i komunikacijske tehnologije u procesu učenja i poučavanja, te podržava učenje u smislu autentičnog konteksta, više perspektiva kroz timski rad i suradnju. Obrnuta učionica uključuje gledanje obrazovnih videozapisa prije nastave i rješavanje praktičnih zadataka tijekom nastave, podržavajući na taj način razvoj vještina programiranja. Problemsko rješavanje zadataka omogućuje aktivnosti učenja kroz grupni rad i pomaže učenicima razvijanje vještina kao što su samo-usmjerenje, suradnja, kreativnost i inovacija.

Istraživanje učinkovitost ovakvog kombiniranog pristupa poučavanju [29] pokazuje povećanu učinkovitost i unapređenje znanja posebno za slabije studente. Istraživanje je razmatralo tri tipa poučavanja, tradicionalni pristup, tehniku obrnute učionice i kombinirani pristup. Tri navedene strategije primijenjene su na brojne programske teme. Element obrazovanja istražen je putem upitnika kojim se procjenjivalo okruženje učenja, angažman, te zadovoljstvo učenika tijekom predavanja. Analiza ankete pokazuje da kombinirani pristup pomaže u obrazovanju učenika i pruža ugodnije iskustvo učenja.

2. Vizualno programiranje

Mnogo truda je uloženo u razvoj alata za vizualno programiranje koji kroz zabavno okruženje imaju cilj smanjiti tjeskobu i strah koji su često povezani s učenjem programiranja. Ti alati namijenjeni su prvenstveno djeci, ali i učenicima srednjih škola i studentima na uvodnim kolegijima programiranja. Vjeruje se da upotreba vizualnih alata može povećati motivaciju za nastavak studija programiranja [30].

Općenito, vizualno programiranje opisuje programsko okruženje u kojem su vizualnim ikonama, blokovima, predstavljeni elementi računalnog programa kao što su petlje, varijable, uvjetna grananja itd. Na taj način programer može brzo napredovati u pisanju koda bez učenja sintakse cijelog programskog jezika.

Prije razvoja Scratcha, postojala su druga programska okruženja, kao što su *Karel the Robot*, Logo i Alice, a sva su pokušavala učiniti učenje programiranja jednostavnijim i intuitivnijim.

Robotom, po imenu Karel, moglo se upravljati zadavanjem jednostavnih naredbi. Ovaj jednostavan princip poučavanja programiranja osmišljen je tijekom sedamdesetih godina na sveučilištu Stanford i bio je jedan od prvih vizualnih jezika.

Na sličnom principu rada temelji se i programski jezik Logo. Jednostavnim naredbama upravlja se kretanjem kornjače, grafičkog elementa na ekranu. Logo ima tekstualno okruženje i svaki redak koda mora biti u skladu sa pravilima programskog jezika prije nego što se program može pokrenuti. U tom slučaju se više pažnje posvećuju ispravnom pisanju sintakse, dok se manji naglasak stavlja na semantičko značenje.

Alice je besplatan interaktivni alat za 3D programiranje koji pomaže učenicima upoznati se s objektno-orijentiranim konceptima programiranja u obliku kreiranja videoigara ili animacija u vizualnom programskom okruženju. Autor Alice, Randy Paush, postavio je učenika u ulogu redatelja filma koji animira objekte u virtualnom svijetu. Prema zadanom scenariju se postavlja scenografija, određuju objekti koji će imati glavnu i sporedne uloge te zatim akcije koje će se odvijati u priči. Naredbe upravljaju postavljenim objektima i slažu se mišem po principu povuci-i-spusti.

Scratch smanjuje složenost sintakse Alice i samim time ima određene pedagoške prednosti.

2.1. Scratch

Scratch, jedan od najraširenijih vizualnih programskih jezika, razvijen 2003. godine kao projekt LifelongKindergarten grupe na Tehnološkom institutu države Massachusetts (Massachusetts Institute of Technology, MIT). Istraživačka grupa LifelongKindergarten u suradnji s tvrtkom Lego uočila je razvoj dječje kreativnosti i mašte u igri sa kockicama pa su odlučili napraviti vizualni programski jezik koji podsjeća na slaganje kockica.

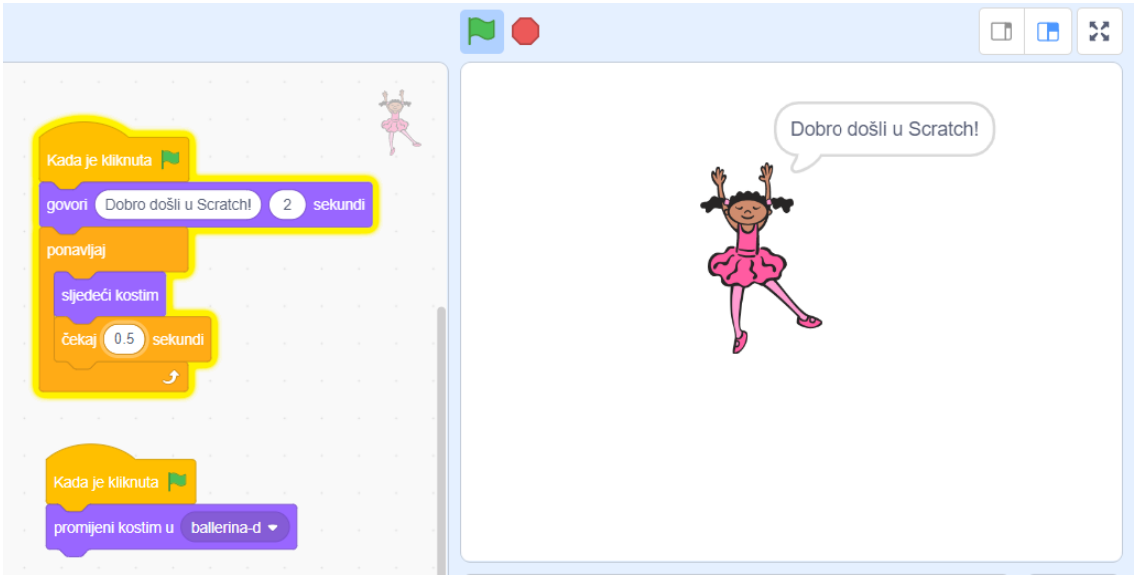
Zbog svoje jednostavnosti Scratch se koristi kao uvod u programiranje, a računalne vještine koje učenici razviju u Scratchu mogu se kasnije primijeniti u drugim programskim jezicima poput Python-a i Jave [31][32]. Primjenom Scratcha u edukaciji potiče se učenje temeljeno na iskustvu. Učenici uče, rješavaju probleme, međusobno surađuju, realiziraju svoje ideje te kritički razmišljaju. Te aktivnosti podrazumijevaju otvorenu komunikaciju, analizu i kontinuirano učenje. Svoju primjenu Scratch može imati i u drugim školskim predmetima. Učiteljima može poslužiti kao alat za izradu kvizova, interaktivnih programa, multimedijalnih te animiranih prezentacija za prikaz različitih sadržaja.

Od javnog pokretanja u svibnju 2007., web stranica Scratch (http://scratch.mit.edu) postala je velika zajednica kreativnih korisnika koji programiranjem izražavaju svoje ideje [4]. Danas se Scratch aktivno koristi u gotovo 150 zemalja diljem svijeta i preveden je na 50 svjetskih jezika među kojima je i hrvatski jezik. Zbirka napravljenih i objavljenih projekata iznimno je raznolika: video igrice, znanstvene simulacije, interaktivne prezentacije, rođendanske čestitke, virtualne ture kao i mnoge druge. Scratch je besplatan alat koji se može instalirati na bilo koji računalni sustav ili mu se može pristupiti preko web poveznice http://scratch.mit.edu. Verzija koja je korištena u ovom radu je aktualna *on-line* verzija Scratch 3.0.

Za početak rada u Scratchu novi korisnik mora proći kroz postupak registracije i stvaranja računa na kojem će biti vidljivi svi njegovi projekti. Nakon uspješno obavljene registracije korisnik može započeti sa programiranjem i dijeljenjem svojih radova sa cjelokupnom Scratch zajednicom.

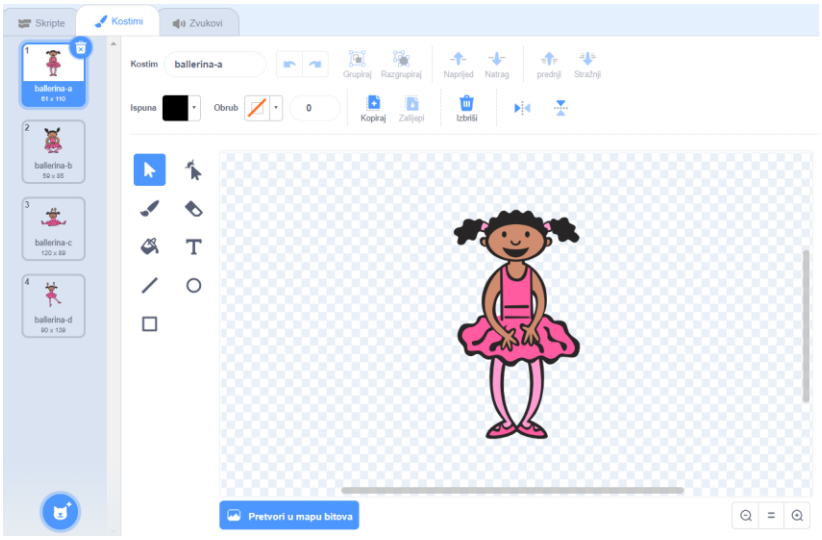
Programi u Scratchu slažu se poput LEGO kockica. Naredbe su napravljene u obliku grafičkih blokova i tipkovnica se koristi samo u slučaju kad se želi upisati broj ponavljanja određene naredbe ili ispisati poruku. Scratch omogućuje programiranje povlačenjem-i-spuštanjem blokova (engl. *drag and drop*) na predviđeno mjesto unutar programskog

okruženja. Nakon pravilnog slaganja blokova program se pokreće klikom na zelenu zastavicu i izvršavaju se sve naredbe unutar bloka za početak (Slika 2.1).



Slika 2.1 - Izvođenje programa u Scratchu

Blokovi naredbi koji se nalaze na radnoj površini, a ne nalaze se unutar bloka za početak neće se izvršiti. Pokretanjem programa izvršavanje se vidi na pozornici u obliku promjene izgleda ili kretanja 2-D grafičkog lika (engl. *sprite*), ovisno o zadanim naredbama. Likovi mogu biti različite životinje, osobe ili objekti poput automobila, lopte, kuće, jabuke itd. Dodavanjem različitih kostima (engl. *costumes*) mijenjamo izgled odabranom liku. Svaki lik ima jedan ili više kostima. Kako bi animirali lik potrebno je složiti naredbe u cjelinu koju nazivamo skripta. Pomoću skripti likovi na pozornici razgovaraju, kreću se, reproduciraju glazbu.



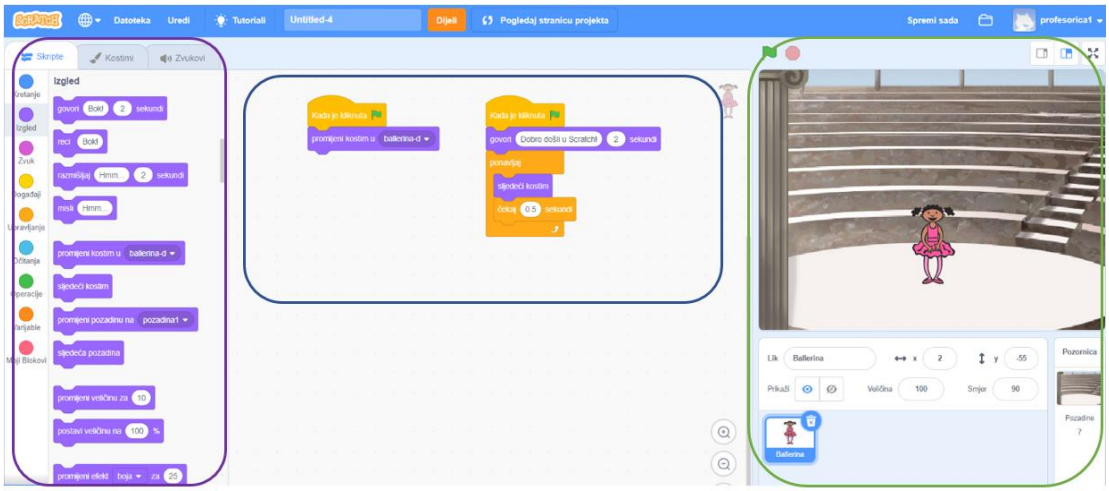
Slika 2.2 - Grafički uređivač

Scratch ima i opciju crtanja i uređivanja vlastitih likova i pripadajućih kostima što pridonosi kreativnom izražavanju učenika (Slika 2.2).

Pozadine određuju kontekst igrice koju stvaramo i Scratch nudi bogati izbor već gotovih pozadina raspoređenih u kategorije. Osim već ponuđenih možemo nacrtati ili učitati svoju te na nju dodavati različite likove.

Pozornica na kojoj se program izvodi je podijeljena na x i y os i ishodište se nalazi u sredini pozornice. Koordinate lika na pozornici možemo očitati u x-y prikazu ispod pozornice. Položaj lika možemo mijenjati povlačenjem miša ili upisujući željene vrijednosti za x i y koordinatu.

Sučelje za razvoj projekata nalazi se u jednom prozoru i dijeli se na tri fiksna elementa. Na desnoj strani sučelja nalazi se pozornica, prostor na kojem se izvodi program. Korisnik ima mogućnost povećanja pozornice preko cijelog ekrana. Ispod pozornice nalazi se popis likova i pozadina koji se mogu postaviti na pozornici. Na lijevoj strani nalazi se popis skripti i pripadajući blokovi naredbi. U sredini je prostor za slaganje naredbi (Slika 2.3).



Slika 2.3 - Scratch okruženje

Grafički blokovi, naredbe, su grupirani tematski i svaka grupa ima drugačiju boju. Oblik blokova pokazuje način spajanja pa je korisniku odmah jasno koji blokovi mogu biti samo na početku ili samo na kraju kao i oni koji mogu biti samo u sredini. Na taj način eliminira se mogućnost sintaksnih pogreški.

2.2. Pregled istraživanja primjene Scratcha u poučavanju programiranja

Scratch je programsko okruženje koje omogućuje korisnicima, uglavnom u dobi između 8 i 16 godina, učenje i razumijevanje temeljnih koncepata programiranja kroz razvijanje svrhovitih i smislenih projekta poput igara ili animacija [2].

Programiranjem u Scratchu korisnici mogu pozitivno razvijati kompetencije kao što su rješavanje problema i vještine kritičkog mišljenja, koje su ključne za 21. stoljeće [33].

Armoni, Mareebaum-Salant, Ben-Ari (2015) razmatrali su prijelaz s učenja programiranja u vizualnom Scratch okruženju na učenje programiranja u profesionalnom tekstualnom programskom jeziku (C# ili Java) u srednjoj školi. Utvrdili su da je programsko znanje i iskustvo učenika koji su naučili Scratch olakšalo učenje naprednijeg gradiva. Učenicima je bilo potrebno manje vremena za učenje novih tema, imali su manje poteškoća u učenju i imali su bolje razumijevanje većine koncepata. Ujedno uočena je viša razina motivacije, samoučinkovitosti i interes za računalnu znanost. Ovo istraživanje opravdava poučavanje programiranja općenito, a posebno vizualnog programiranja u srednjim školama [32].

Scratch je vrlo korisna okolina jer se programiranje izvodi izgradnjom blokova jednostavnih naredbi, a ne pisanjem tekstualnih naredbi. Theodorou i Kordaki (2010) dizajnirali su i razvili poznatu računalnu igricu Super Mario u Scratch okruženju kako bi srednjoškolcima približili temeljne koncepte u programiranju, poput pojma varijable [34].

Učenje temeljeno na igricama, učenje temeljeno na problemu i vizualno programiranje mogu pomoći učenicima ostvariti bolje rezultate u uvodnim tečajevima programiranja. Razvoj igara u stvarnom životu uz programsko okruženje Scratch poboljšava i vještine programiranja i motivaciju učenika [35].

Vještina rješavanja problema uključuje shvaćanje problema, analizu problema, pronalaženje rješenja i pisanje programa, provjeru rješenja testiranjem i izmjenu programa prema rezultatu testa [36]. Lai i Yang (2011) proveli su istraživanje kako bi procijenili učinak vizualnog programiranja na logičke sposobnosti učenika i vještine rješavanja problema. U istraživanju u kojem su sudjelovali učenici šestih razreda primijetili su značajno poboljšanje vještine rješavanja problema. Zaključeno je kako povezivanje vizualnog učenja programiranja i strategija rješavanja problema može poboljšati vještine rješavanja problema kod učenika.

U sličnom istraživanju do malo drugačijih rezultata došli su Jiang i Li (2021). Cilj ovog istraživanja bio je također analizirati učinke učenja jezika Scratch na vještine računalnog mišljenja (kreativnost, algoritamsko mišljenje, kooperativnost, kritičko mišljenje i rješavanje problema) učenika osnovnih škola. Proveli su istraživanje s 336 učenika petih razreda. Rezultati istraživanja pokazuju kako postoji značajna razlika u vještinama kreativnosti, kooperativnosti i kritičkog mišljenja. Međutim, u ovom istraživanju Scratch nije postigao značajne razlike u vještinama rješavanja problema i algoritamskog razmišljanja. Učitelji bi trebali Scratch kombinirati s drugim predmetima, kao što su matematika i robotsko programiranje kako bi učenicima pružili smislenije probleme programiranja koji razvijaju vještine algoritamskog razmišljanja [37].

Wilson i Moffat (2010) bilježili su kognitivni napredak učenika i afektivni utjecaj svake lekcije tijekom 8 tjedana poučavanja u Scratch programskom okruženju. Istraživanje je provedeno s učenicima osnovnih škola u dobi od 8 do 9 godina. Rezultati su pokazali umjereni kognitivni napredak učenika i ugodnije iskustvo za učenike, čineći učenje programiranja pozitivnim iskustvom, suprotno frustraciji i tjeskobi koji tako često karakteriziraju uobičajeno iskustvo učenja [31].

Učenici percipiraju Scratch kao lagan i zabavan alat koji motivira učenje programiranja i potiče želju za upisivanjem studija programiranja [38].

Scratch je implementiran kao preliminarno programsko okruženje kako bi se poboljšalo iskustvo u programiranju, motivacija i uspjeh studenata računalnog inženjerstva [39]. U fokusu istraživanja bilo je prihvaćanje i korištenje programskog okruženja Scratch razvijajući teorijski model temeljen na Modelu prihvaćanja tehnologije (TAM). Istraživanje na uzorku od 186 studenata pokazalo je značajnu povezanost percipiranog uživanja s percipiranom korisnošću i stavovima. Nadalje, rezultati pokazuju kako je samoučinkovitost značajno povezana s percepcijom lakoće korištenja.

U Južnoj Africi učenici 10. razreda (10. razred je ekvivalent našem prvom razredu srednje škole) kroz Scratch usvajaju osnovne koncepte programiranja, a u 11. razredu (11. razred je ekvivalent našem drugom razredu srednje škole) prelaze na Javu, programski jezik više razine. Istraživanje, vođeno Modelom prihvaćanja tehnologije (engl. Technology Acceptance Model, TAM), pokazalo je da učenici 10. razreda smatraju Scratch lakim za korištenje i korisnim, a učenici 11. razreda smatraju da je jednostavan za korištenje, ali koristan samo u učenju uvodnih koncepata programiranja. Scratch pomaže učenicima

razumjeti logiku i rješavanje problema, ali ne pomaže im dovoljno u pripremi za korištenje programskog jezika više razine kao što je Java [40].

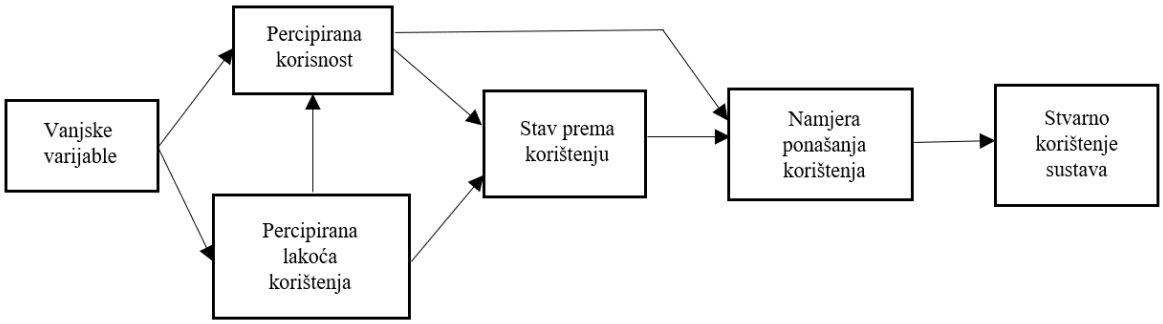
Scratch se može koristiti i *on-line* i *off-line* verziji, pa je vrlo koristan za proces učenja na daljinu (engl. Distance Learning Process, DLP). Nakon obuke učitelja za rad u programu Scratch, provjerena je njihova razina prihvaćanja nove tehnologije. Mjereni pokazatelji bili su percipirana korisnost, percipirana lakoća korištenja, namjera korištenja i stvarna upotreba tehnologije te samoučinkovitost. Iz rezultata provedene analize može se vidjeti da namjera korištenja utječe na stvarnu upotrebu tehnologije. Uočena korisnost i percipirana lakoća korištenja zajedno utječu na namjeru stvarne upotrebe dok uočena lakoća korištenja i samoučinkovitost zajedno imaju značajan učinak na percipiranu korisnost [41].

3. Model prihvaćanja tehnologije

Model prihvaćanja tehnologije (engl. *Technology Acceptance Model*, TAM) teorija je informacijskih sustava koja modelira kako korisnici prihvaćaju i koriste tehnologiju. Stvarna upotreba sustava krajnja je točka u kojoj ljudi koriste tehnologiju i brojni čimbenici utječu na njihovu odluku o tome kako i kada će je koristiti.

Percipirana korisnost je kognitivna procjena hoće li usvajanje nove tehnologije utjecati na radnu izvedbu pojedinca. Percipirana korisnost utječe na stav pojedinca prema korištenju novih tehnologija jer ljudi imaju tendenciju formirati pozitivne stavove prema novim tehnologijama ako vjeruju da će biti korisna za ono što žele raditi. Percipirana korisnost također izravno utječe na namjeru ponašanja za korištenje sustava. Pojedinci razvijaju namjeru za korištenje tehnologije ako su uvjereni da će ona pozitivno utjecati na radnu uspješnost kao što su povišice ili unapređenja. Motivacija za dobivanjem uvjetnih nagrada nadilazi osobne osjećaje koje pojedinci imaju prema tehnologiji [42].

Percipirana lakoća korištenja utječe na stav prema korištenju i na percipiranu korisnost. Kad je tehnologija jednostavna za upotrebu korisnik ima snažno uvjerenje u svoju sposobnost korištenja sustava, što rezultira entuzijastičnijim stavom prema sustavu. Lakoća korištenja također izravno utječe na izvedbe pojedinca, budući da će nova tehnologija vjerojatno dovesti do završetka zadatka uz manje napora [42].



Slika 3.1 - TAM model

Prema Davisu i sur. (1989), percipirana korisnost i percipirana lakoća korištenja imaju značajan utjecaj na stav, što zauzvrat utječe na namjeru stvarnog korištenja. Gledano u smislu TAM okvira, stvarna upotreba Scratcha može biti određena namjerom učenika da koristi Scratch, a namjera je određena ukupnim stavom učenika prema Scratchu. Percepcija

korisnosti i lakoće korištenja oblikuju njihov stav (Slika 3.1). U istraživanju u ovom radu, percipirana korisnost je prikazana kao stupanj u kojem učenik vjeruje da upotreba programskog jezika Scratch poboljšava njegovu sposobnost programiranja. Percipirana lakoća korištenja označava stupanj vjerovanja da je Scratch jednostavan za rad bez ulaganja velikog kognitivnog napora. Vanjske varijable kao što je društveni utjecaj važan su čimbenik za određivanje stava. Isto tako, percepcija se može promijeniti ovisno o dobi i spolu jer su svi različiti.

4. Metoda istraživanja

Cilj ovog istraživanje je otkrivanje povezanosti 4 varijable TAM modela, percipirane korisnosti, percipirane lakoće korištenja, stava i namjere korištenja programskog jezika Scratch među učenicima jedne srednje strukovne škole u dobi od 15 do 17. godina. U skladu s navedenim ciljem istraživanja postavljene su sljedeće hipoteze:

H1 – Motivacijska varijabla percipirana korisnost značajan je prediktor varijable stav prema korištenju. Varijable su pozitivno povezane.

H2 – Motivacijska varijabla percipirana lakoća korištenja značajan je prediktor varijable namjere korištenja. Varijable su pozitivno povezane.

H3 – Varijabla stav prema korištenju značajan je prediktor namjere korištenja. Varijable su pozitivno povezane.

4.1. Opis istraživanja i sudionici

Istraživanje je provedeno tijekom redovne nastave računalstva u školskoj godini 2021./2022. u prirodnom okruženju bez prisustva trećih osoba. Istraživanje je obuhvatilo tri prva razreda (15 – 16 godina) i tri druga razreda (16 - 17 godina) srednje strukovne škole. Provedba istraživanja je trajala 2 tjedna i odvijala se u nekoliko etapa: upoznavanje sa Scratch okruženjem, demonstracija razvoja igrice, samostalan rad učenika, međusobni dijalog i konstruktivno komentiranje vlastitih i tuđih igrica te ispunjavanja izlaznih TAM upitnika.

Sudionici u istraživanju su početnici u programiranju. Kako se nastava tijekom prethodne školske godine odvijala većinom online zbog nepovoljnih uvjeta uzrokovanih Covid virusom, nastavna cjelina programiranja nije provedena iako je planirana godišnjim izvedbenim kurikulumom, tako je većini učenika u obje dobne skupine ovo bio prvi susret s programiranjem.

Na početku provedbe nastavne cjeline „Rješavanje problema pomoću računala“ uveden je Scratch za upoznavanje učenika s pojmom algoritma. Svi sudionici su bili izloženi istom tretmanu u terminu po dva sata tjedno kao blok sat. U poučavanju je primijenjen model

kognitivnog šegrtovanja. Tijekom prvog sata nastavnica je provela učenike kroz proces registracije i kreiranja korisničkog računa u Scratchu, predstavila Scratch okruženje, objasnila skripte, naredbe, događaje i kontrolne strukture, te pokazala igricu koju će učenici izrađivati. Učenici su na prvom satu, prateći upute nastavnice, započeli sa izradom vlastitog lika i pripadajućeg kostima kao i pozadine za cijelu igricu. Tijekom sljedeća tri školska sata učenici su izradili vlastite animacije, u početku uz povremenu pomoć nastavnice, a zatim potpuno samostalno. Tablica 4.2 prikazuje detaljno izrađen tijek istraživanja kao i tijek uvođenja novih naredbi i koncepata programiranja. Kroz izradu jednostavne igrice učenici su se upoznali sa sva tri algoritama, slijeda, ponavljanja i uvjetnog grananja, te pojmom varijable. Tema odabrane igrice je bliska učenicima i ciljano prilagođena uzrastu sudionika i interesima učenika kako bi se povećala motivacija. Vizualni izgled naredbi u Scratchu omogućava intuitivno usvajanje značenja pojedinih naredbi i slaganje jednostavnih skripti. Zbog vizualnog izvršavanja programa učenici su mogli pratiti izvođenje programa, prepoznati eventualne greške u izvršavanju, te ih na primjeren način ispraviti. Nakon upoznavanja s temeljnim konceptima programiranja planiran je prelazak na proceduralni jezik Python koji je ujedno i zadani programski jezik po godišnjem izvedbenom kurikulumu.

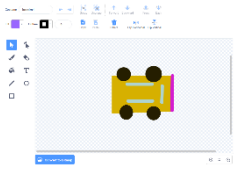
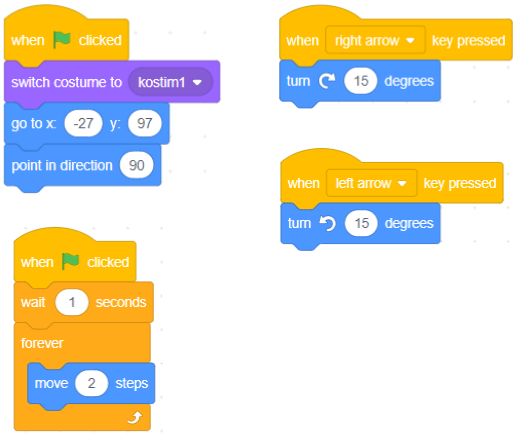
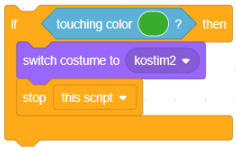
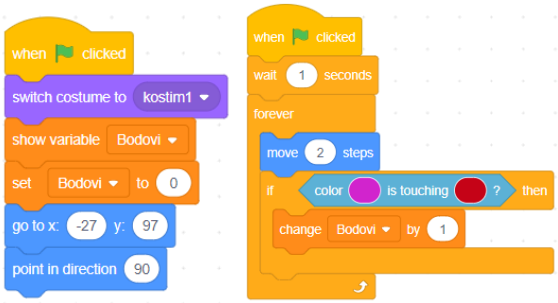
Izlaznim TAM upitnicima izmjerilo se prihvaćanje Scratcha, percipirana korisnost, percipirana lakoća korištenja, stav prema korištenju te namjera korištenja. Istraživanje je obuhvatilo i kvalitativne i kvantitativne tehnike prikupljanja podataka. Otvoreni tip pitanja omogućio je razumijevanje prikupljenih kvantitativnih podataka. Kvantitativni podatci korišteni su za analizu povezanosti između četiri varijable TAM-a (percipirane korisnosti, percipirane lakoće korištenja, stava prema korištenju i namjere korištenja) kako bi se odgovorilo na postavljena istraživačka pitanja.

Za istraživanje je planirana ciljna populacija od 140 sudionika. Tablica 4.1 prikazuje uzorak sudionika po svakom upitniku koji su u konačnici ispunjeni i vraćeni.

Tablica 4.1 - Uzorak sudionika u istraživanju

Upitnik	Broj učenika	Dječaci	Djevojčice
Percipirana korisnost	78	77	1
Percipirana lakoća korištenja	72	71	1
Stav prema korištenju	76	75	1
Namjera korištenja	71	70	1

Tablica 4.2 - Tijek istraživanja

<table>
<tr><th>tjedan</th><th>sat</th><th>Tema</th><th>Novi koncepti i naredbe</th><th>Opis</th></tr>
<tr><td rowspan="2">1.</td><td>1.</td><td>Utrka</td><td>Lik (sprite), kostimi (costumes), skripte (code), crtanje lika, kostima i pozadine
   </td><td>Upoznavanje s Scratch okruženjem, likovima, kostimima, skriptama i naredbama. Izrada vlastitog lika i pripadajućeg kostima te izrada pozadine igrice u grafičkom uređivaču.</td></tr>
<tr><td>2.</td><td>Utrka</td><td>Položaj lika (x, y-koordinate), smjer (direction, turn), promjeni kostim (switch costume), čekaj (wait), idi (move), događaji (kad je zelena zastavica pritisnuta, kad je strelica lijevo pritisnuta, kad je strelica desno pritisnuta)
Primjer koda:
  </td><td>Uvođenje algoritma slijeda i algoritma ponavljanja izradom simulacije kretanja automobila po nacrtanoj stazi. Upravljanje kratanjem lika pomoću strelica lijevo i desno na tipkovnici.</td></tr>
<tr><td rowspan="3">2.</td><td>1.</td><td>Utrka</td><td>Ako (if), uvjet (touching color ___?), zaustavi izvršavanje skripte (stop this script)
  </td><td>Uvođenje uvjetnog izvršavanja programa.</td></tr>
<tr><td>2.</td><td>Utrka</td><td>Varijable, prikaži varijablu (show variable), postavi vrijednost varijable na 0 (set ____ to 0), promijeni vrijednost varijable za 1(change ____ by 1).
Primjer koda:
  </td><td>Uvođenje pojma varijable potrebno je za dobijanje bodova u igri.</td></tr>
<tr><td colspan="3">Ispunjavanje 4 TAM upitnika</td></tr>
</table>

4.2. Instrumenti

Za dobivanje vrijednosti svih varijabli predviđenih TAM modelom u ovom istraživanju korišteni su slijedeći instrumenti mjerenja:

1. Upitnik percipirane korisnosti ispituje procjenu učenika o stupnju korisnosti koje će učenje programiranja u Scratch-u imati na njegovu ukupnu sposobnost programiranja. Upitnik se sastoji od 6 afirmativnih tvrdnji i jednog pitanja otvorenog tipa. Tablica 4.3 prikazuje opis tvrdnji kao i tip pitanja.

Tablica 4.3 - Upitnik percipirane korisnosti

Pitanje	Tvrdnje	Tip pitanja
P1	Scratch unapređuje moje logičko razmišljanje.	Likertova skala (1-5)
P2	Scratch unapređuje moju sposobnost rješavanja problema.	Likertova skala (1-5)
P3	Scratch je koristan za razumijevanje temeljnih koncepata programiranja.	Likertova skala (1-5)
P4	Rad u Scratchu povećava moju motivaciju za učenjem programiranja.	Likertova skala (1-5)
P5	Scratch povećava učinkovitost u programiranju.	Likertova skala (1-5)
P6	Scratch razvija moju kreativnost.	Likertova skala (1-5)
P7	Ako imaš neki svoj komentar u vezi programiranja u Scratchu slobodno ga napiši.	Otvoreni tip pitanja

2. Upitnik percipirane lakoće korištenja ispituje procjenu učenika o stupnju lakoće korištenja i snalaženja u programskom okruženju Scratch. Upitnik se sastoji od 6 afirmativnih tvrdnji. Tablica 4.4 prikazuje opis tvrdnji kao i tip pitanja.

Tablica 4.4 - Upitnik percipirane lakoće korištenja

Pitanje	Tvrdnje	Tip pitanja
P1	Scratch je jednostavan za korištenje.	Likertova skala (1-5)
P2	Izgled Scratch sučelja je jednostavan i intuitivan.	Likertova skala (1-5)
P3	Scratch je koristan za razumijevanje temeljnih koncepata programiranja.	Likertova skala (1-5)
P4	Upute za kreiranje koda su jednostavne i razumljive.	Likertova skala (1-5)
P5	Lako je postati vješt u Scratch programiranju.	Likertova skala (1-5)
P6	U Scratchu mogu programirati i bez pomoći nastavnika.	Likertova skala (1-5)

3. Upitnik stav prema korištenju mjeri učenikov ukupan stav prema Scratch programiranju. Na njegov stav utječu percipirana korisnost i percipirana lakoća korištenja. Tablica 4.5 prikazuje opis tvrdnji ovog upitnika.

Tablica 4.5- Upitnik stav prema korištenju

Pitanje	Tvrdnje	Tip pitanja
P1	Scratch ima pozitivan utjecaj na moje programerske vještine.	Likertova skala (1-5)
P2	Želio/željela bih više znati o programiranju.	Likertova skala (1-5)
P3	Učenje programiranja u Scratchu je dobra ideja.	Likertova skala (1-5)

4. Upitnik namjere korištenja sadrži tri afirmativne tvrdnje (Tablica 4.6). Prema Davisu i suradnicima [42], percipirana korisnost i percipirana lakoća korištenja imaju značajan utjecaj na korisnikov stav, što u konačnici utječe i na njegovu namjeru za stvarnom upotrebom tehnologije.

Tablica 4.6 - Upitnik namjere korištenja

Pitanje	Tvrdnje	Tip pitanja
P1	Preporučiti ću Scratch drugima.	Likertova skala (1-5)
P2	Planiram i u budućnosti programirati u Scratchu.	Likertova skala (1-5)
P3	Motiviran/a sam i dalje učiti programiranje.	Likertova skala (1-5)

Upitnici sadrže zatvoreni i otvoreni tip pitanja. Zatvorena pitanja koristila su Likertovu skalu ocjena od 1 (u potpunosti se ne slažem) do 5 (u potpunosti se slažem) kako bi se steklo razumijevanje o mišljenjima učenika o učinkovitosti Scratcha. Tvrdnje u upitnicima su potvrdno formulirane pa veća ocjena odgovara većem slaganju sa tvrdnjom. Otvoreni tip pitanja korišten je za bolje razumijevanje percepcije učenika o Scratchu.

4.3. Prikupljanje podataka

Ovo istraživanje provedeno je u školskom okruženju tijekom redovnog izvođenja nastave iz predmeta računalstvo. Učenici prvih i drugih razreda imali su jednake uvjete, te nisu imali dodatne domaće zadaće ili bilo kakve dodatne obaveze izvan redovne nastave. Istraživanje je naturalističko jer se sudionici nalaze u svom prirodnom okruženju u kojem je istraživač ujedno i nastavnik te samim tim dio tog prirodnog okruženja. Nastava se odvijala po godišnjem izvedbenom kurikulumu i od učenika nije bio potreban svjestan pristanak. Obzirom da se radi o naturalističkom istraživanju [43] [44] željelo se eliminirati svjesno mijenjanje ponašanja sudionika koje bi moglo utjecati na valjanost rezultata istraživanja u konačnici. Sudionici su potpuno anonimno odgovarali na upitnike i analizom podataka nije moguće utvrditi njihov identitet. Ispunjavanje upitnika bilo je dobrovoljno te sudionici nisu bili motivirani dobivanjem plusa ili ocjene za aktivnost. Cilj je bio da učenici što iskrenije odgovore na pitanja i ukoliko žele napišu komentare. Isto tako, tijekom istraživanja nije bilo ocjenjivanja znanja.

5. Rezultati i rasprava

Za analizu kvantitativnih podataka korištena je deskriptivna statistička analiza i tehnika korelacijske analize. Vrijednosti deskriptivne statistike svih varijabli, broj ispitanika u istraživanju (N), aritmetička sredina (M), Mod, minimalna i maksimalna izmjerena vrijednost varijable prikazani su u tablici ispod (Tablica 5.1). Učenici su ispunjavali upitnike u on-line obliku nakon tretmana u Scratchu. Kako je odgovaranje na upitnike bilo potpuno dobrovoljno, broj učenika (N) je različit za svaki upitnik jer dio učenika nije ispunio sva četiri upitnika.

Tablica 5.1 - Deskriptivna statistika promatranih varijabli

Varijabla	N	As	Mod	Min	Max
Upitnik percipirane korisnosti	78	3,52	4	1	5
Upitnik percipirane lakoće korištenja	72	4,15	4	1	5
Upitnik stava prema korištenju	76	3,96	4	1	5
Upitnik namjere korištenja	71	3,49	3	1	5

Za mjerenje pouzdanosti upitnika korišten je koeficijent Cronbach alpha, čija prihvatljiva vrijednost za empirijska istraživanja iznosi od 0,7 ili više. Upitnici percipirana korisnost (Cronbach α=0,913) i percipirana lakoća korištenja (Cronbach α=0,864) spadaju u grupu visoko pouzdanih dok upitnik namjera korištenja spada u grupu granično pouzdanih (α=0,685). Za tri pitanja upitnika stav prema korištenju koeficijent pouzdanosti je niži od prihvatljivog (α=0,500) što se može pripisati malom broju pitanja u upitniku.

Tablica 5.2 - Rezultati analize pouzdanosti upitnika

Varijabla	α
Upitnik percipirane korisnosti	0,913
Upitnik percipirane lakoće korištenja	0,864
Upitnik stava prema korištenju	0,500
Upitnik namjere korištenja	0,685

Pearsonov koeficijent korelacije korišten je za mjerenje snage i smjera povezanosti između četiri varijable modela prihvaćanja tehnologije. Odabrana je Pearsonova korelacija jer se dobiveni kvantitativni podaci statistički značajno ne razlikuju od normalne raspodjele.

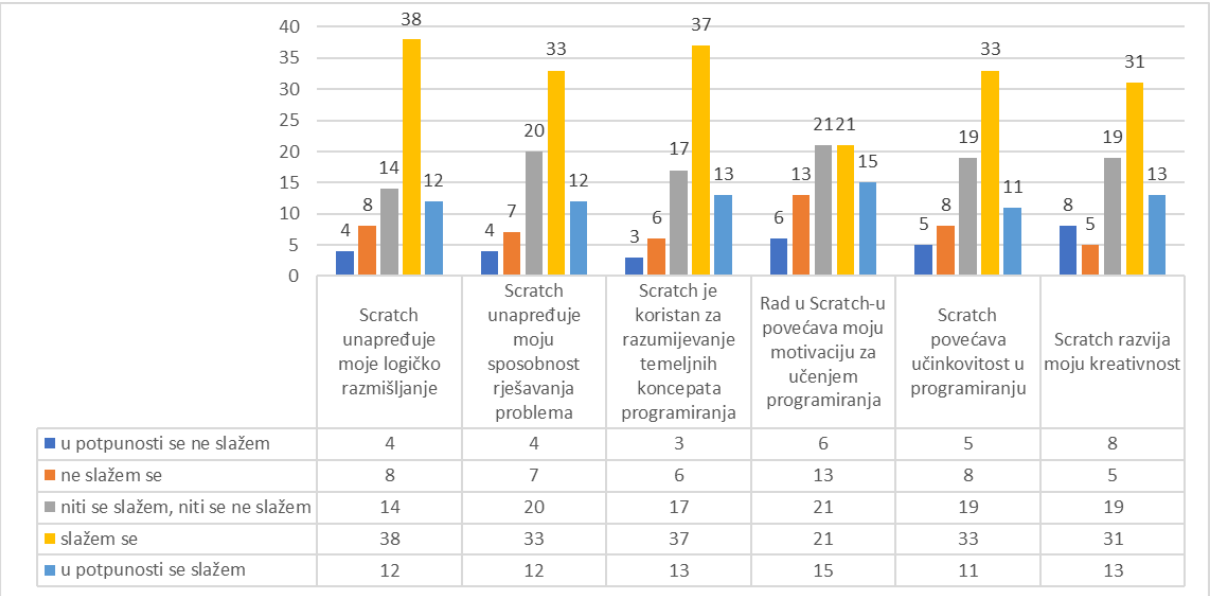
5.1. Upitnik percipirane korisnosti

Tablica 5.3 prikazuje deskriptivne podatke za svako pitanje iz upitnika percipirane korisnosti. Deskriptivni podaci su prikazani kao aritmetička sredina svakog pojedinog odgovora i vrijednost odgovara koji se najčešće ponavlja (Mod) te minimalna i maksimalna moguća vrijednost odgovora.

Tablica 5.3 - Deskriptivna statistika upitnika percipirane korisnosti

Pitanje	As	Mod	Min	Max
Scratch unapređuje moje logičko razmišljanje.	3,6	4	1	5
Scratch unapređuje moju sposobnost rješavanja problema.	3,5	4	1	5
Scratch je koristan za razumijevanje temeljnih koncepata programiranja	3,7	4	1	5
Rad u Scratch-u povećava moju motivaciju za učenjem programranja	3,3	4	1	5
Scratch povećava učinkovitost u programiranju	3,4	4	1	5
Scratch razvija moju kreativnost	3,4	4	1	5

Frekvencije odgovora na svako pitanje prikazane su grafički na slici ispod (Slika 5.1).



Slika 5.1 - Frekvencije svih odgovora upitnika percipirane korisnosti

korisnost Scratcha u pogledu svih tvrdnji osim tvrdnje “Rad u Scratch-u povećava moju motivaciju za učenjem programiranja” gdje je među odgovorima bio gotovo jednak broj nepovoljnih odgovora, ukupno 52% (27% je bilo neutralno i 25% se ne slaže ili se u potpunosti ne slaže s tvrdnjom) i povoljnih odgovora ukupno 48% (ili slaže se ili u potpunosti se slaže s tvrdnjom). Objašnjenje toga može biti činjenica da je programiranje danas jedno od najtraženijih, visoko plaćenih zanimanja u svijetu i učenici koji imaju ambiciju učiti „prave“ programske jezike su doživjeli programsko okruženje Scratch inferiorno i „dječje“ u smislu učenja programiranja. U prilog toj pretpostavci ide i većina komentara na neobavezno pitanje otvorenog tipa koje je napisalo pet učenika. Komentari su prikazani u obliku kako su ih učenici napisali.

- *„scratch je jako dosadan i jednostavan program.“*
- *„Scratch je zabavan program za programiranje ali nije bas puno ucinkovit.“*
- *„dobar za pocetnike“*
- *„Djecja aplikacija za programiranje“*
- *„vrh“*

Gotovo jednak broj ispitanika se složio i u potpunosti složio s tvrdnjama “Scratch unapređuje moje logičko razmišljanje“, “Scratch je koristan za razumijevanje temeljnih koncepata programiranja”, “Scratch poboljšava moje sposobnosti rješavanja problema” i „Scratch povećava učinkovitost u programiranju“ pa se može izvući zaključak da su učenici uvidjeli korisnost učenja programiranja u Scratchu. Prema Davisu i suradnicima [42] percipirana korisnost ima značajan utjecaj na oblikovanje stava prema korištenju Scratcha, a što zauzvrat utječe i na njihovu namjeru stvarnog korištenja.

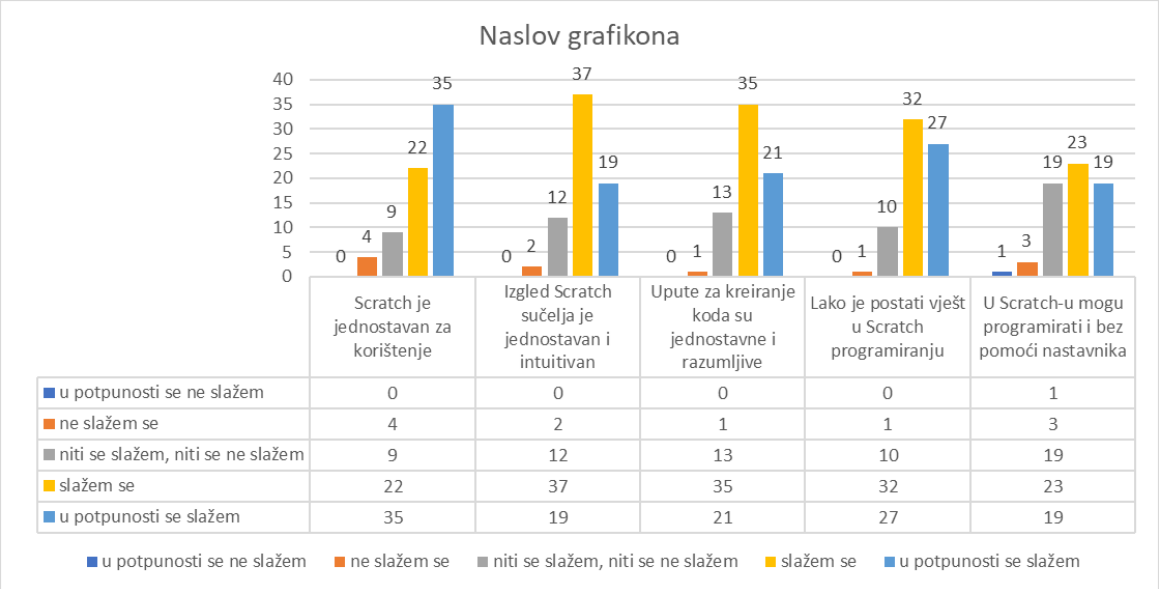
5.2. Upitnik percipirane lakoće korištenja

Tablica 5.4 prikazuje deskriptivne podatke za svako pitanje iz upitnika percipirane lakoće korištenja. Prikazani podatci su aritmetička sredina svakog pojedinog odgovora i vrijednost odgovara koji se najčešće ponavlja (Mod) te minimalna i maksimalna moguća vrijednost odgovora.

Tablica 5.4 - Deskriptivna statistika upitnika percipirane lakoće korištenja

Pitanje	As	Mod	Min	Max
Scratch je jednostavan za korištenje	4,2	5	2	5
Izgled Scratch sučelja je jednostavan i intuitivan	4,0	4	2	5
Upute za kreiranje koda su jednostavne i razumljive	4,0	4	2	5
Lako je postati vješt u Scratch programiranju	4,2	4	2	5
U Scratchu mogu programirati i bez pomoći nastavnika	3,8	4	1	5

Najčešći odgovor na pitanja ovog upitnika bio je odgovor slažem se (Tablica 5.4).



Slika 5.2 - Frekvencije odgovora upitnika percipirane lakoće korištenja

Učenici imaju uglavnom pozitivne odgovore (u potpunosti se slažu ili slažu) u pogledu tvrdnji jednostavnosti korištenja što ukazuje na činjenicu da je većini učenika Scratch jednostavan za korištenje (Slika 5.2). Jedina razlika u ovom pozitivnom nizu odgovora je posljednja tvrdnja „U Scratchu mogu programirati i bez pomoći nastavnika“ gdje je 36%

učenika dalo nepovoljan odgovor (19 učenika je ostalo neutralno, 3 učenika se nisu složila te 1 učenik se u potpunosti nije složio). Ova razlika nije neočekivana, budući da je većina učenika prvi put bila izložena programiranju u Scratchu, pa je logično da su sigurniji u programiranju uz pomoć nastavnika.

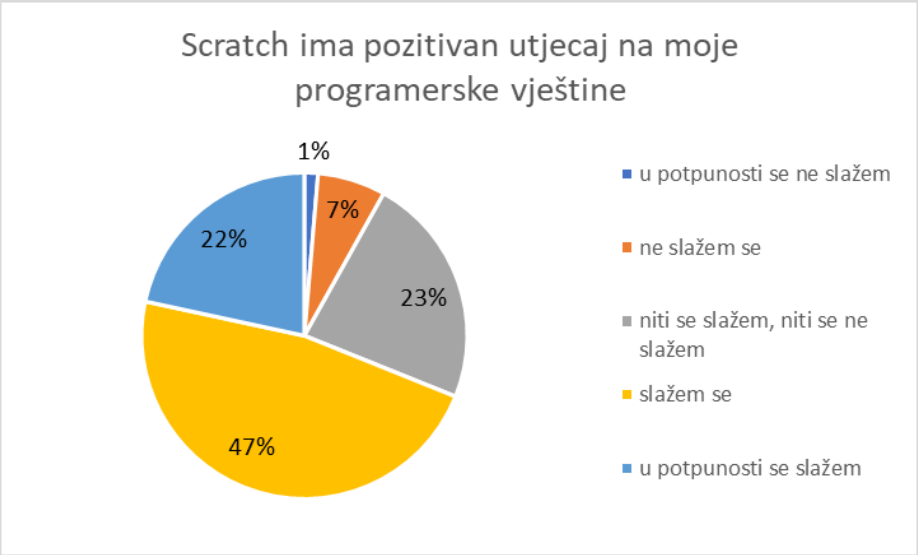
5.3. Upitnik stav prema korištenju

Tablica 5.5 prikazuje deskriptivne podatke za upitnik stav prema korištenju. Deskriptivni podaci su prikazani kao aritmetička sredina svakog pojedinog odgovora i vrijednost odgovara koji se najčešće ponavlja (Mod) te minimalna i maksimalna moguća vrijednost odgovora.

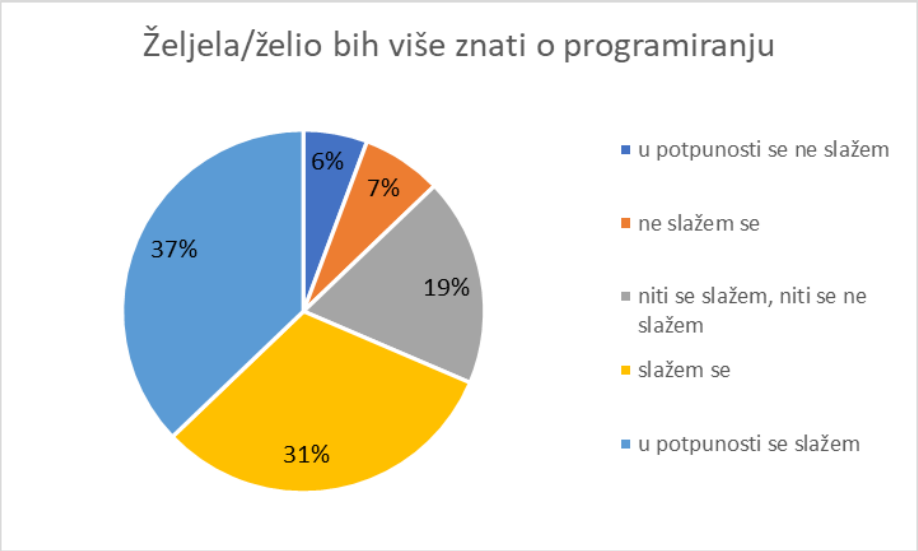
Tablica 5.5 - Deskriptivna statistika upitnika stav prema korištenju

Pitanje	As	Mod	Min	Max
Scratch ima pozitivan utjecaj na moje programerske vještine	3,8	4	1	5
Željela/želio bih više znati o programiranju	3,8	5	1	5
Učenje programiranja u Scratch-u je dobra ideja	3,7	4	1	5

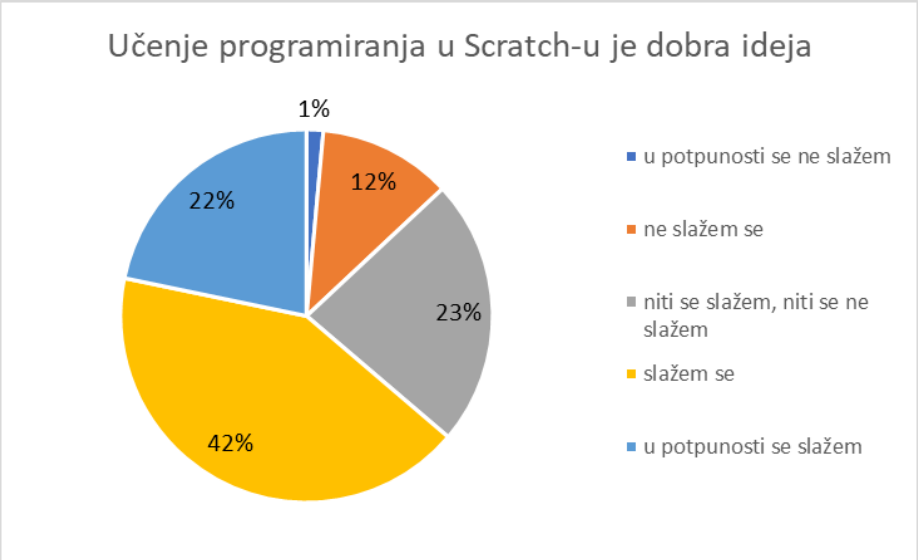
Većina učenika ima pozitivan stav prema korištenju Scratcha. Najčešći odgovori na sva tri pitanja ovog upitnika su ili slažem se ili u potpunosti se slažem.



Slika 5.3 - Rezultati odgovora na pitanje pozitivnog utjecaja Scratcha na programerske vještine učenika



Slika 5.4 - Rezultati odgovora vezanih uz motivaciju nastavka učenja programiranja



Slika 5.5 - Mišljenje učenika o učenju programiranja u Scratchu

Ukupni pozitivni odgovori (u potpunosti se slažem i slažem se) na svako pojedino pitanje iz ovog upitnika se nalaze u rasponu od 64% do 69% (Slika 5.3), (Slika 5.4), (Slika 5.5). Nepovoljni odgovori (niti se slažem, niti se ne slažem, ne slažem se i u potpunosti se ne slažem) nikad nisu bili u većini, s manje od 40% ukupno. Učenici su većinom imali pozitivne odgovore u upitnicima percipirane korisnosti i percipirane lakoće korištenja, dvije motivacijske varijable koje imaju značajan utjecaj na stav, pa je ovakav rezultat i očekivan.

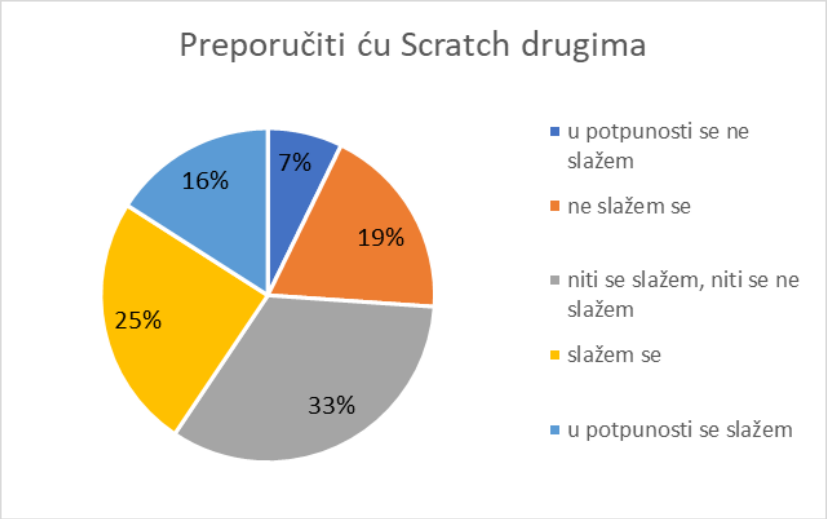
5.4. Upitnik namjere korištenja

Tablica 5.6 prikazuje deskriptivne statističke podatke za namjeru prema korištenju Scratcha. Deskriptivni podaci su prikazani kao aritmetička sredina svakog pojedinog odgovora i vrijednost odgovara koji se najčešće ponavlja (Mod).

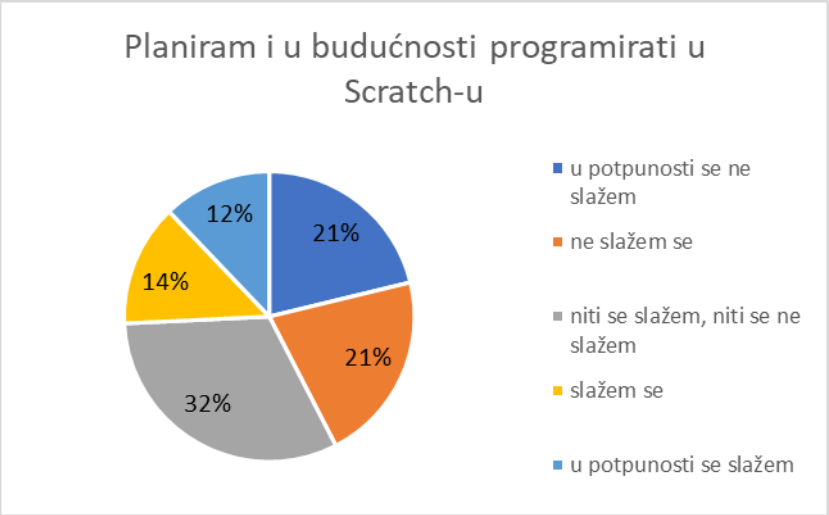
Tablica 5.6 - Deskriptivna statistika upitnika namjere korištenja

Pitanje	As	Mod	Min	Max
Preporučiti ću Scratch i drugima	4,0	3	1	5
Planiram i u budućnostiprogramirati u Scratchu	2,7	3	1	5
Motiviran/a sam i dalje učiti programiranje	3,9	5	1	5

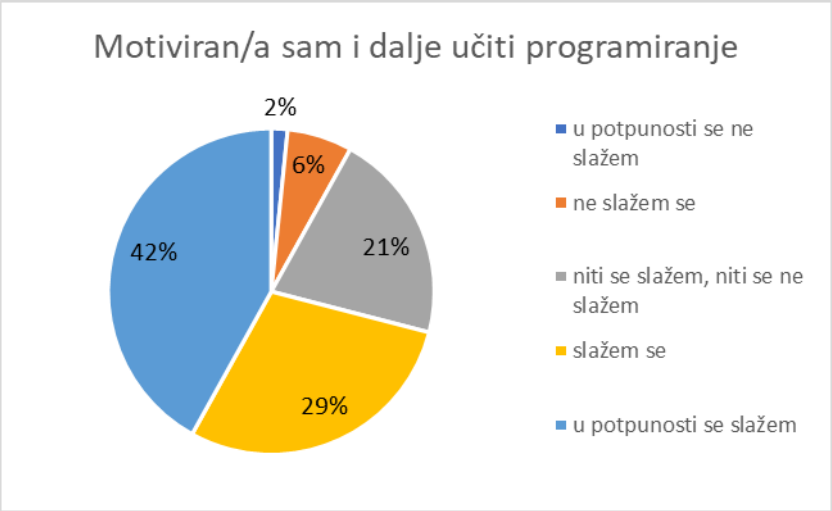
Prema rezultatima može se primijetiti da je najčešći odgovor na prva dva pitanja - niti se slažem, niti se ne slažem što znači da su učenici neutralni u svojoj namjeri korištenja Scratcha. Na slikama ispod grafički su prikazani postotci odgovora na svako pojedino pitanje iz ovog upitnika.



Slika 5.6 - Mišljenje učenika o preporuci Scratcha svojim vršnjacima



Slika 5.7 - Mišljenje učenika o namjeri korištenja Scratcha



Slika 5.8 - Motivacija za nastavak programiranja

71% ispitanika se složilo ili u potpunosti složilo s izjavom da su motivirani i dalje učiti programiranje (Slika 5.8). Međutim, bilo je mnogo nepovoljnih odgovora na izjave "Planiram i u budućnosti programirati u Scratchu" (samo 26% se složilo ili u potpunosti složilo) i "Preporučiti ću Scratch drugima" (samo 41% se složilo ili u potpunosti složilo). Za te dvije tvrdnje većina ispitanika je ostala ili neutralna ili se nije složila. 32% ispitanika su neutralna dok se 42% nije složilo ili u potpunosti složilo s tvrdnjom "Planiram u budućnosti programirati u Scratchu" (Slika 5.7). 33% ispitanika je neutralno, dok se 26% ispitanika nije složilo s tvrdnjom "Preporučiti ću Scratch drugima" (Slika 5.6). Dakle, to sugerira da učenici početnih razreda srednje škole ipak percipiraju Scratch kao „dječji alat

za programiranje“ te su svjesni ograničenja u mogućnostima Scratcha i sukladno tome su shvatili da ga ne namjeravaju koristiti u budućnosti.

5.5. Korelacijska analiza upitnika

Pearsonov koeficijent korelacije korišten je za mjerenje snage i smjera za sve četiri TAM varijable. Pearsonov koeficijent može varirati od +1 do -1. Dvije varijable su u pozitivnoj korelaciji ako između njih postoji direktna, jednosmjerna veza. U jednosmjernom odnosu, male vrijednosti jedne varijable odgovaraju malim vrijednostima druge varijable, velike vrijednosti odgovaraju velikim. Dvije varijable su u negativnoj korelaciji ako između njih postoji inverzna veza.

Tablica 5.7 - Korelacijska analiza upitnika

Korelacije

		Stav prema korištenju	Percipirana lakoća korištenja	Percipirana korisnost	Namjera korištenja
Stav prema korištenju	Pearson-ov koeficijent korelacije	1	.102	.302*	.154
	Statistička značajnost		.405	.012	.207
	Broj ispitanika (N)	69	69	69	69
Percipirana lakoća korištenja	Pearson-ov koeficijent korelacije	.102	1	.038	.279*
	Statistička značajnost	.405		.759	.020
	Broj ispitanika (N)	69	69	69	69
Percipirana korisnost	Pearson-ov koeficijent korelacije	.302*	.038	1	.139
	Statistička značajnost	.012	.759		.253
	Broj ispitanika (N)	69	69	69	69
Namjera korištenja	Pearson-ov koeficijent korelacije	.154	.279*	.139	1
	Statistička značajnost	.207	.020	.253	
	Broj ispitanika (N)	69	69	69	69

*. Correlation is significant at the 0.05 level (2-tailed).

Dobiveni rezultati (Tablica 5.7) pokazuju postojanje statistički značajne pozitivne povezanost (.302*) između uvjerenja učenika da je Scratch koristan i njihovog stava prema

korištenju. Ova statistički značajna povezanost pokazuje da je percipirana korisnost Scratcha značajno utjecala na njihov stav o Scratchu.

Uvjerenje ispitanika o lakoći korištenja Scratcha također je značajno koreliralo (.279*) s namjerom ponašanja učenika da ga koriste. To potvrđuje postojanje linearne povezanosti između lakoće korištenja Scratcha i stvarne namjere korištenja Scratcha.

Dobivene korelacije su u skladu sa TAM modelom i dosadašnjim istraživanjima. Percipirana lakoća korištenja utječe na namjeru ponašanja. Percipirana korisnost utječe na stav prema korištenju i time su potvrđene prve dvije postavljene hipoteze ovog istraživanja.

Međutim, varijabla stav nije značajno korelirala (.154) s namjerom korištenja, što ukazuje da stav učenika o Scratchu nije značajno utjecao na njihovu namjeru da koriste ga koriste. Time je treća hipoteza samo djelomično potvrđena jer je povezanost između varijabli ipak pozitivna.

6. Zaključak

Provedeno istraživanje imalo je za cilj otkriti kako srednjoškolci u dobi od 15 do 17 godina prihvaćaju programiranje u Scratchu istražujući povezanost četiri varijable TAM modela, percipiranu korisnost, percipiranu lakoću korištenja, stav prema korištenju i namjeru korištenja.

Rezultati istraživanja pokazuju da je percipirana lakoća korištenja značajan čimbenik u učenikovoj namjeri stvarnog korištenje Scratcha. Analiza rezultata također je pokazala da je percipirana korisnost značajno utjecala na učenikov stav prema korištenju Scratcha.

Kad se odgovori prikupljeni iz zatvorenih i otvorenih anketnih pitanja razmatraju zajedno, može se zaključiti da učenici početnih razreda srednje škole doživljavaju Scratch jednostavnim za korištenje i korisnim, ali ipak nedovoljno ozbiljnim alatom za učenje programiranja. Takvo mišljenje učenika o Scratcha se može pripisati činjenici da su ti učenici bili izloženi drugim programskim jezicima tijekom osnovnoškolskog obrazovanja i mnogi su došli do uvjerenja da Scratch ne pruža potrebno znanje, sintaktičko i računalno, potrebno za „pravo“ programiranje u proceduralnim jezicima.

Dakle, ukupni podaci prikupljeni ovim istraživanjem sugeriraju pozitivan utjecaj Scratcha na razvijanje logičke vještine i vještine rješavanja problema i povećanu motivaciju za daljnjim učenjem programiranja u nekom konkretnijem programskom jeziku, a što je ujedno krajnji cilj i motiv uvođenja vizualno-blokovskog jezika na početku nastavne cjeline programiranja kao posredovanog transfera iz jednog konteksta programiranja u drugi.

Ograničenje ovog istraživanja je kratko vrijeme provođenja tretmana. Cijelo istraživanje je trajalo dva tjedna, odnosno četiri školska sata. Dodatno ograničenje je sudjelovanje manjeg broja učenika pa bi bilo interesantno provesti istraživanje na većem broju učenika i razreda u različitim srednjim školama.

Kako je nastavnica računalstva ujedno i provodila istraživanje imala je i ulogu sudjelujućeg opažača. Najvažniji zaključak je da Scratch omogućuje uvođenje temeljnih koncepata programiranja u puno manje vremena za razliku od tekstualnih programskih jezika. U ovom istraživanju učenici su se upoznali s algoritmom slijeda, ponavljanja i

grananja tijekom četiri školska sata. S algoritmom slijeda i ponavljanja već tijekom prvog sata programiranja kroz simulaciju pokretanja i upravljanja likom. Dodatne funkcionalnosti kao što su situacije kad lik (autić) dodirne zelenu boju (pređe rub automobilske staze) ili kad lik dodirne crnu boju (dođe do cilja) ostvarene su s algoritmom grananja. Pojam varijable, kao jedan od najtežih koncepata programiranja, uveden je već na četvrtom školskom satu kroz skupljanje bodova u igrici. Svi navedeni koncepti su učenicima predstavljeni kroz kontekst igrice pa samim tim je njihova uloga vrlo konkretna i jasna i učenik razumije program. Rezultat izvršavanja naredbi vidljiv je kroz kretanje likova i učenici su planirali rješenja programa, promatrali izvršavanje programa, samostalno uočavali i otklanjali pogreške. Scratch okruženje omogućuje usmjeravanje pažnje na rješavanje problema bez suočavanja sa problemom krute sintakse. Grupe naredbi su kodirane bojom i oblikom pa je svaka naredba intuitivno jasna, čak i bez dodatnog pojašnjavanja. Na taj način su i učenici sa slabije razvijenim apstraktnim razmišljanjem aktivno sudjelovali u nastavi dok su samostalniji učenici imali više vremena istraživati dodatne naredbe kojima mogu unaprijediti svoju igricu. Scratch, također, potiče suradnju među učenicima pa su oni aktivno komunicirali, razmjenjivali i uspoređivali ideje, međusobno pomagali u rješavanju problema pri pisanju programa.

U tradicionalnoj nastavi programiranja nastavnik izlaže programske pojmove, a učenici primjenjuju stečeno znanje rješavajući zadane zadatke. Razredne skupine često su heterogene s obzirom na prethodno znanje, kognitivne sposobnosti, stilove učenja i motivaciju. Brojnost i raznolikost skupine predstavlja otežavajuće okolnosti za nastavnike tradicionalnih navika poučavanja.

Zahvaljujući mnogobrojnim istraživanjima tijekom posljednjih nekoliko desetljeća postignut je napredak u razumijevanju loših rezultata početnika u učenju programiranja i predložene su nove strategije poučavanja i programska okruženja koja pokazuju napredak u izvedbi učenika. Odabir pojedine strategije poučavanja namijenjene početnicima treba prilagoditi kontekstu i okruženju učenja te samom učeniku. Uzimajući u obzir različite karakteristike svakog pojedinog učenika, želja konstruktivističkog pristupa poučavanju programiranja je približiti „teško“ gradivo većem broju učenika. Samim tim konstruktivistički pristup za nastavnike predstavlja puno više posla i organizacije, a za učenike puno veći osobni angažman pa možemo zaključiti kako konstruktivistički pristup, prije svega, zahtijeva promjenu svijesti o ulozi koju pojedinac ima u razredu, kako učenik tako i nastavnik.

Kada se sagledaju razmišljanja srednjoškolaca obuhvaćenih ovim istraživanjem, može se jasno reći da učenici vole programiranje i žele učiti programirati. Isto tako, većina učenika je doživjela Scratch jednostavnim za korištenje i još važnije, korisnim za razvoj sposobnosti rješavanja problema i logike. Ovakvi nalazi potiču potrebu za razvojem multimedijskih aktivnosti i okruženja koji bi, posebno početnicima u učenju programiranja, olakšali prijelaz u svijet „pravog“ programiranja u tekstualnom programskom jeziku. S druge strane, samo osiguravanje takvog okruženja nije dovoljno za učinkovito poučavanje. Zadatke i izazove za učenike treba oblikovati prema njihovoj dobi i interesima, ali i kontekstu vremena u kojem živimo pri čemu istraživanja u edukaciji, dokumentirana najbolja praksa, standardi kurikuluma, profesionalni razvoj i podrška nastavnicima su također potrebni.

Literatura

[1] M. Hassinen and H. Mäyrä, "Learning programming by programming: A case study," in *ACM International Conference Proceeding Series*, 2006, vol. 276.

[2] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Trans. Comput. Educ.*, vol. 10, no. 4, 2010.

[3] N. Bubica, M. Mladenović, and I. Boljat, "Programiranje kao alat za razvoj apstraktnog mišljenja," 2013.

[4] M. Resnick *et al.*, "Scratch: Programming for Everyone," *Commun. ACM*, vol. 52., 2009.

[5] S. Papert, *Mindstorms: Children, Computers and Powerful Ideas*. 1993.

[6] R. S. Nickerson, "Computer programming as a vehicle for teaching thinking skills," *Think. J. Philos. Child.*, vol. 4, pp. 42–48, 1983.

[7] R. D. Pea, "Language-Independent Conceptual 'Bugs' in Novice Programming," *J. Educ. Comput. Res.*, vol. 2, no. 1, pp. 25–36, 1986.

[8] M. Ben-Ari, "Constructivism in Computer Science Education," in *Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education*, 1998, pp. 257–261.

[9] A. J. Gomes and A. J. N. Mendes, "Problem Solving in Programming," in *PPIG*, 2007.

[10] P. H. Tan, T. Yee, and siew-woei Ling, "Learning Difficulties in Programming Courses: Undergraduates' Perspective and Perception," 2009, pp. 42–46.

[11] I. Milne and G. Rowe, "Difficulties in Learning and Teaching Programming—Views of Students and Tutors," *Educ. Inf. Technol.*, vol. 7, pp. 55–66, 2002.

[12] J. Bonar and E. Soloway, "Uncovering Principles of Novice Programming," in *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1983, pp. 10–13.

[13] M. Ben-Ari, "Constructivism in Computer Science Education," *SIGCSE Bull.*, vol. 30, no. 1, pp. 257–261, 1998.

[14] K. J. W. Craik, *The nature of explanation*. Oxford, England: University Press, Macmillan, 1943.

[15] Y. Bosse, D. Redmiles, and M. A. Gerosa, "Pedagogical Content for Professors of Introductory Programming Courses," 2019.

[16] L. Festinger, *A theory of cognitive dissonance*. Stanford University Press, 1957.

[17] P. W. Hewson and M. A. Hewson, "The role of conceptual conflict in conceptual change and the design of science instruction.," *Instr. Sci.*, vol. 13, no. 1, pp. 1–13, 1984.

[18] R. E. Mayer, "Should There Be a Three-Strikes Rule Against Pure Discovery

Learning?," *American Psychologist*, vol. 59, no. 1. American Psychological Association, Mayer, Richard E.: Department of Psychology, University of California, Santa Barbara, Santa Barbara, CA, US, 93106-9660, mayer@psych.ucsb.edu, pp. 14–19, 2004.

[19] J. S. Brown, A. Collins, and P. Duguid, "Situated Cognition and the Culture of Learning," *Educ. Res.*, vol. 18, no. 1, pp. 32–42, 1989.

[20] T. Naps *et al.*, "Exploring the Role of Visualization and Engagement in Computer Science Education," *SIGCSE Bull.*, vol. 35, pp. 131–152, 2003.

[21] A. Moreno, N. Myller, E. Sutinen, and M. Ben-Ari, "Visualizing Programs with Jeliot 3," in *Proceedings of the Working Conference on Advanced Visual Interfaces*, 2004, pp. 373–376.

[22] M. E. Rana and M. Hosanee, "A Refined Approach for Understanding Role of Variables in Elementary Programming," *J. Adv. Res. Dyn. Control Syst.*, vol. 10, pp. 238–248, 2018.

[23] K. Beck, "Embracing Change with Extreme Programming," *Computer (Long. Beach. Calif).*, vol. 32, no. 10, pp. 70–77, 1999.

[24] M. Corney, D. Teague, and R. N. Thomas, "Engaging Students in Programming," in *Proceedings of the Twelfth Australasian Conference on Computing Education - Volume 103*, 2010, pp. 63–72.

[25] L. Porter, S. Garcia, J. Glick, A. Matusiewicz, and C. Taylor, "Peer Instruction in Computer Science at Small Liberal Arts Colleges," in *Proceedings of the 18th ACM Conference on Innovation and Technology in Computer Science Education*, 2013, pp. 129–134.

[26] C. E. Davenport, "Evolution in Student Perceptions of a Flipped Classroom in a Computer Programming Course," *J. Coll. Sci. Teach.*, vol. 47, pp. 30–35, 2018.

[27] V. Allan and M. Kolesar, "Teaching Computer Science: A Problem Solving Approach that Works," *ACM SIGCUE Outlook*, vol. 25, 1998.

[28] O. Muller and B. Haberman, "Supporting abstraction processes in problem solving through pattern-oriented instruction," *Comput. Sci. Educ.*, vol. 18, pp. 187–212, 2008.

[29] A. Chis, A.-N. Moldovan, L. Murphy, and C. Muntean, "Investigating Flipped Classroom and Problem-based Learning in a programming module for computing conversion course," *Educ. Technol. Soc.*, vol. 21, pp. 232–247, 2018.

[30] O. Meerbaum-Salant, M. Armoni, and M. Ben-Ari, "Learning Computer Science Concepts with Scratch," *Comput. Sci. Educ.*, vol. 23, pp. 239–264, 2013.

[31] A. Wilson and D. Moffat, "Evaluating Scratch to introduce younger schoolchildren to," *Proc. 22nd Annu. Work. Psychol. Program. Interes. group-PPIG2010, Sept. 19-22, 2010*, 2012.

[32] M. Armoni, O. Meerbaum-Salant, and M. Ben-Ari, "From scratch to 'Real' programming," *ACM Trans. Comput. Educ.*, vol. 14, no. 4, 2015.

[33] F. O. Marques and M. T. Marques, "No problem ? No research, little learning ...Big problem!," *J. Syst. Cybern. Informatics*, vol. 10, no. 3, 2012.

[34] C. Theodorou and M. Kordaki, "Super Mario : a Collaborative Game for the

Learning of Variables in Programming," *Building*, vol. 2, no. 4, 2010.

[35] D. Topalli and N. E. Cagiltay, "Improving programming skills in engineering education through problem-based game projects with Scratch," *Comput. Educ.*, vol. 120, 2018.

[36] A. F. Lai and S. M. Yang, "The learning effect of visualized programming learning on 6th graders' problem solving and logical reasoning abilities," in *2011 International Conference on Electrical and Control Engineering, ICECE 2011 - Proceedings*, 2011.

[37] B. Jiang and Z. Li, "Effect of Scratch on computational thinking skills of Chinese primary school students," *J. Comput. Educ.*, vol. 8, no. 4, 2021.

[38] L. Permatasari, R. A. Yuana, and D. Maryono, "Implementation of Scratch Application to Improve Learning Outcomes and Student Motivation on Basic Programming Subjects," *J. Informatics Vocat. Educ.*, vol. 3, no. 1, 2020.

[39] I. Arpaci, P. O. Durdu, and A. Mutlu, "The Role of Self-Efficacy and Perceived Enjoyment in Predicting Computer Engineering Students' Continuous Use Intention of Scratch," *Int. J. E-Adoption*, vol. 11, no. 2, 2019.

[40] M. Marimuthu and P. Govender, "Perceptions of Scratch Programming among Secondary School Students in KwaZulu-Natal, South Africa," *African J. Inf. Commun.*, no. 21, 2018.

[41] G. Gunadi and I. K. Sudaryana, "ANALISA TINGKAT PENERIMAAN APLIKASI SCRATCH MENGGUNAKAN TECHNOLOGY ACCEPTANCE MODEL (TAM)," *Infotech J. Technol. Inf.*, vol. 7, no. 1, 2021.

[42] F. Davis and F. Davis, "Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology," *MIS Q.*, vol. 13, pp. 319-, 1989.

[43] K. Cohen, L., Manion, L., &Morrison, *Research Methods in Education (7th ed.)*. 2011.

[44] Monika Mladenović, "Poučavanje početnog programiranja oblikovanjem računalnih igara," 2019.

Sažetak

PRIHVAĆANJE I KORIŠTENJE SCRATCHA OD STRANE POČETNIKA U UČENJU PROGRAMIRANJA

Računalno programiranje razvija vještine rješavanja problema i logičkog zaključivanja i njegova integracija u sve obrazovne razine je nužna. Međutim učenje programiranja nije jednostavno i predstavlja izazov i za učenike i za nastavnike. Zbog svoje jednostavnosti Scratch se koristi kao uvod u programiranje, a računalne vještine koje učenici razviju u Scratchu mogu se kasnije primijeniti u drugim programskim jezicima. U ovom diplomskom radu istraživala se povezanost četiri varijable TAM modela, percipirane korisnosti, percipirane lakoće korištenja, stava prema korištenju i namjere korištenja programskog jezika Scratch, kako bi se otkrio stav i namjera korištenja Scratcha među početnicima u programiranju. Prema kvantitativnim rezultatima može se jasno reći da su svi učenici voljeli programiranje i željeli poboljšati svoje programiranje. Rezultati su, između ostalog, pokazali i kako učenici smatraju Scratch jednostavnim za korištenje i korisnim, ali ipak ne dovoljno „ozbiljnim“ za učenje drugih tekstualnih programskih jezika.

Ključne riječi

Model prihvaćanja tehnologije (TAM), vizualno programiranje, strategije poučavanja, srednjoškolski učenici, računalno razmišljanje, vještine rješavanja problema

Summary

PERCEIVED ACCEPTANCE AND USE OF SCRATCH AMONG BEGINNERS OF LEARNING PROGRAMMING

Computer programming develops problem-solving and logical reasoning skills and its integration into all levels of education is essential. However, learning programming is not easy and is a challenge for both students and teachers. Because of its simplicity, Scratch is used as an introduction to programming, and the computer skills that students develop in Scratch can later be applied in other textual based programming languages. In this thesis, the relationship between four variables of the TAM model, perceived usefulness, perceived ease of use, attitude towards use and intentions to use the Scratch was investigated to reveal the attitude and intention to use Scratch among beginners in programming. According to the quantitative results, it can be clearly said that all students loved programming and wanted to improve their programming. The findings showed, that students find Scratch easy to use and useful, but still not "serious" enough to prepare them for „real programming“ in text based programming languages.

Keywords

Technology Acceptance Model (TAM), visual programming, learning strategies, high school students, computational thinking, problem-solving skills

Popis slika

Popis tablica

Skraćenice

TAM	*Technology Acceptance Model*	Model prihvaćanja tehnologije