Borut Batagelj

Programi kot zaporedja ukazov

Prva uvodna aktivnost

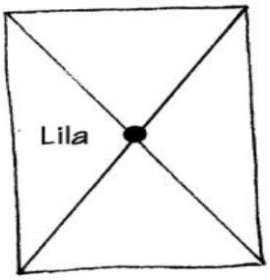
Računalniki vedno naredijo to, kar jim naročimo. Če smo pri dajanju navodil – se pravi programiranju – nepazljivi, so lahko rezultati napačni. Kako se počutita programer in računalnik bomo spoznali v naslednji aktivnosti, ko bomo prevzeli vlogo programerja s tem, da bomo dajali navodila za risanje, ter vlogo računalnika, ko bomo poskušali narisati sliko po navodilih drugega.

S pomočjo te aktivnosti spoznamo kako težko je podajati dobra navodila in kako smešni so lahko rezultati ohlapnih navodil. Na ta način lahko učencem prikažemo, zakaj pišemo programe v posebnih **računalniških jezikih**, ki programerja silijo v točno izražanje.

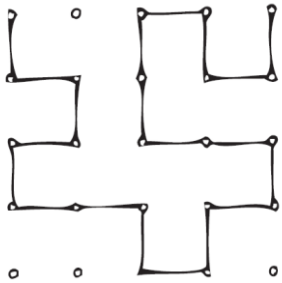
Primer enostavne naloge

Udeleženci naj vzamejo papir in pisalo. Preberemo jim spodnja navodila, oni pa sproti rišejo.

1. Narišite majhen krog sredi papirja.
2. Pobarvajte krog.
3. Potegnite črto iz zgornjega levega kota papirja skozi krog v spodnji desni kot.
4. Potegnite črto iz zgornjega desnega kota papirja skozi krog v spodnje levi kot.
5. Napišite svoje ime v trikotnik levo od kroga na sredini papirja.
6. Udeleženci naj pokažejo, kaj so narisali.
7. Na tablo narišem, kar bi morali, glede na zgornja navodila narisati.



Slika 9



Na takšen način lahko učenci prakticirajo še druge slike (slika 9), s tem da en otrok narekuje navodila za risanje, drugi pa rišejo. Več primerov slik ter še druge zanimive aktivnosti s področja računalništva lahko najdete na portalu: *http://vidra.si*.

Takšno igro vsakodnevno igrajo tudi programerji in računalniki. Programer ima podobno nalogo kot učenec, ki opisuje sliko. Tudi programer daje navodila računalniku in šele potem vidi, kaj je računalnik na osnovi teh navodil naredil. Največji problem v igri niso nenatančna, temveč dvoumna navodila. Zaradi tega so lahko nastale različne slike, ki niso bile takšne, kot bi morale biti.

Ker so človeški jeziki (slovenščina, angleščina, stara grščina ...) preveč ohlapni, nejasni, dvoumni, za programiranje računalnikov uporabljamo posebne jezike, ki so jasnejši in programerja silijo v točno izražanje.

Teoretično ozadje opisane aktivnosti

Računalniku dajemo navodila v obliki **programov**. Vsak program opravlja določeno nalogo. Programi so napisani v jezikih, ki imajo omejen nabor ukazov. Različni programski jeziki so primerni za različne naloge: nekateri so primernejši za programe, ki tečejo na spletu, drugi jeziki so znani po tem, da je mogoče v njih zelo hitro programirati manj zmogljive programe, spet v tretjih je programiranje težje in počasnejše, zato pa so programi, napisani v njih, zelo hitri.

Ne glede na izbrani jezik mora biti programer previden in zelo točno povedati računalniku, kaj bi rad od njega. Računalnik bo vedno dobesedno izpolnil ukaze (kadar bo to mogoče, seveda), pa čeprav je rezultat lahko smešen.

Programerji morajo biti natančni, saj ima lahko že drobna napaka v programu resne posledice. Predstavljajte si, kaj se lahko zgodi zaradi napake v programu, ki krmili jedrsko elektrarno, prižiga luči na železniških semaforjih ali vozi letalo! Napakam v programih rečemo hrošči v čast hrošču (točneje molju), ki so ga našli v enem prvih elektronskih računalnikov iz štiridesetih let prejšnjega stoletja. Odstranjevanju hroščev iz teh ogromnih računalnikov so rekli razhroščevanje (*debugging*) in tudi današnji programerji razhroščujejo svoje programe, pri čemer uporabljajo posebna programska orodja, ki jim pravijo razhroščevalniki.

Druga uvodna aktivnost: Moj prijatelj Robotek[1]

Glavni cilj je osvetliti tehnike programiranja in predstaviti potrebo po funkcijah. Otroci bodo s pomočjo v naprej določenega "robotskega jezika" ugotovili kako voditi drugega, da bo izvršil določeno nalogo brez, da bi mu nalogo prej opisal. Otroci se bodo naučili povezave med simboli in akcijami, med drugim tudi zelo pomembne veščine pri programiranju: razhroščevanju. Na koncu aktivnosti lahko predstavimo še uporabo podprogramov v obliki funkcij.

Otroci se bodo naučili:
- vsakodnevne aktivnosti pretvoriti v navodila,
- pisanje navodil – programiranja s pomočjo simbolov,
- razumevanja, zakaj moramo biti pri programiranju natančni,
- pregledovanju napačno napisanih navodil – programov = razhroščevanje,
- uporabnosti funkcij in parametrov.

Slovar pojmov:
- **algoritem**: navodila kako izvesti nalogo
- **programiranje**: prepis akcij v simbolični jezik
- **razhroščevanje**: poiskati in popraviti napako v programu
- **funkcija**: del kode, ki jo lahko uporabimo znova in znova
- **parameter funkcije**: dodatna informacija, ki jo podamo funkciji, da jo lahko prilagodimo

Simboli, ki predstavljajo ukaze, ki jih robotek pozna:
↑: poberi lonček
↓: spusti lonček
→: korak (pol dolžine lončka) naprej
←: korak (pol dolžine lončka) nazaj
↻: obrni lonček desno za 90 stopinj
↺: obrni lonček levo za 90 stopinj

Otroke najprej vprašamo o robotih. Če so jih že videli, se jih dotaknili? Ali roboti res slišijo in vidijo? Ali znajo govoriti? Ali res razumejo, kar rečejo?
Roboti počnejo natanko to kar smo jim s pomočjo navodil ukazali naj naredijo. Da naredijo določeno nalogo morajo imeti roboti seznam navodil (ki jih rečemo tudi algoritem), ki jih lahko izvedejo.

Samo iz zgoraj predstavljenih šestih simbolov bodo učenci dajali navodila robotu, da bo zgradil določen stolp iz lončkov.
Igra je zelo preprosta:
1) Izberemo enega učenca, ki bo predstavljal robota.
2) Izbranega učenca pošljemo stran, da ne bo videl kaj ostali počnejo.
3) Ostali učenci bodo v vlogi programerja:
    - izberejo poljubno sliko stolpa,

[1] Aktivnost je povzeta po tečaju My robotic friends skupine Thinkersmith (http://thinkersmith.org) predstavljeni na tednu programiranja 2013.

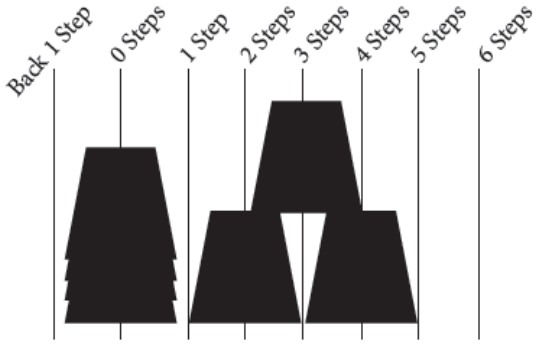
- zapišejo algoritem, kako bo robot zgradil stolp,
- algoritem zapišejo s pomočjo šestih simbolov.

4) Ko programerji zaključijo, pokličejo robota in mu dajo navodila-program.
5) Robot prebere program v obliki simbolov in ga prevede v premike.
6) Učenci naj opazujejo in v primeru napake lahko izvajanje prekinejo
   - napako naj popravijo in program ponovno predajo robotu.

Pri igri je potrebno strogo spoštovati pravila:
- programerji morajo vse poteze prevesti v šest simbolov,
- lončki naj bodo pri robotu in naj ne bodo na razpolago programerjem,
- ko se robot vrne ni dovoljeno pogovarjanje.

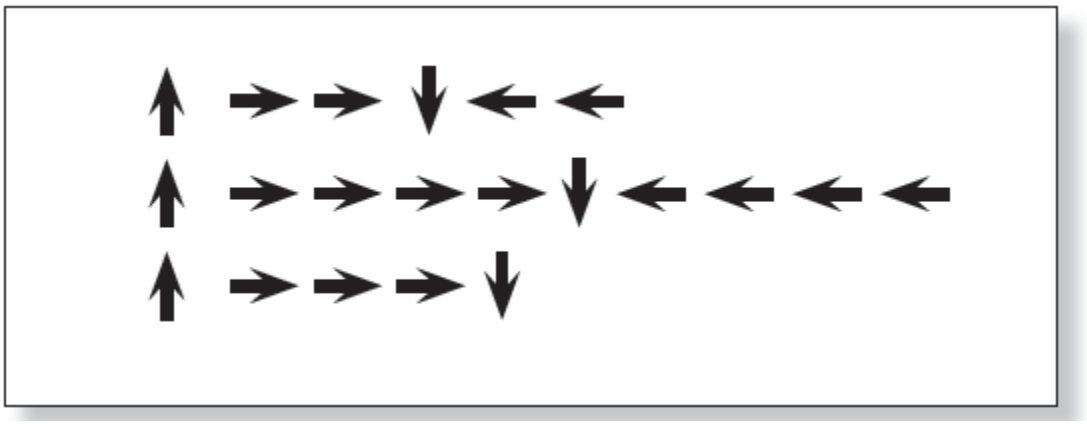
Najprej skupaj sestavimo program za stolp na sliki 1. Opozoriti moramo na to, da se premikamo naprej in nazaj za pol širine lončka (slika 2).

Slika 1: Primer stolpa iz 3 lončkov.



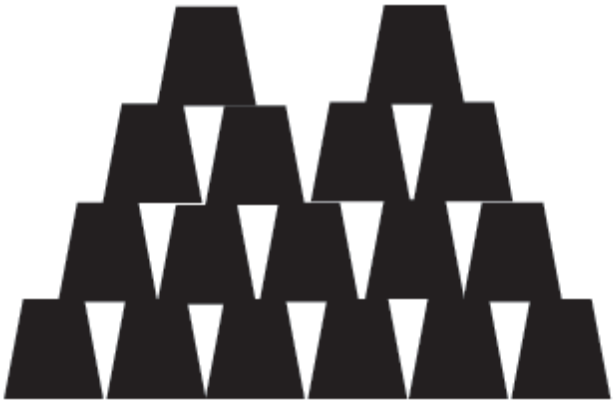
Slika 2: Dolžina koraka naprej in nazaj.

Možna rešitev:



Nato naj otroci po skupinah izberejo novega sošolca, ki bo robot in ostali kot programerji sestavijo program za nov stolp. Ko se robot vrne mu dajo program in v tišini spremljajo sestavljanje. Če ugotovijo napako lahko prekinejo izvajanje in napako popravijo.

Da jih napeljemo na uporabo funkcij jim ponudimo primer stolpa z veliko lončkov (slika 3).



Slika 3: Stolp iz 17 lončkov.

Otroci sami opazijo, da lahko program skrajšajo, če več puščic zapišejo kot ponavljanje:


. To je v bistvu naloga funkcij, da omogočijo da se del kode pokliče večkrat.
Izzovimo jih ali mogoče opazijo več ponavljajočih se blokov, mogoče samo z različnim številom ponovitev. Ugotovimo lahko, da lahko dvig lončka in potem število premikov in spust lončka in nato enako število premikov nazaj zapišemo kot funkcijo za katero vpeljemo nov simbol:

, kjer *x* pomeni število premikov naprej in nazaj in v kontekstu funkcije je to parameter, ki ga podamo funkciji in s tem jo lahko spremenimo.

Programiranje za otroke

Zgornji aktivnosti lahko nadaljujemo tako, da sedaj poskušamo računalniku podati navodila, da izriše želeno sliko na ekran ali premakne figurico na določeno mesto na ekranu. Seveda je izbira pravega programskega jezika za učence, ki niso vešči programiranja, ključnega pomena. Na srečo imamo veliko izbiro tako imenovanih vizualnih orodij za programiranje, kjer ukaze enostavno skladamo skupaj, kot so otroci tega že navajeni pri kockah Lego.

Eno najbolj poznanih in razširjenih okolij je gotovo programsko okolje Scratch (*http://scratch.mit.edu*).

Scratch

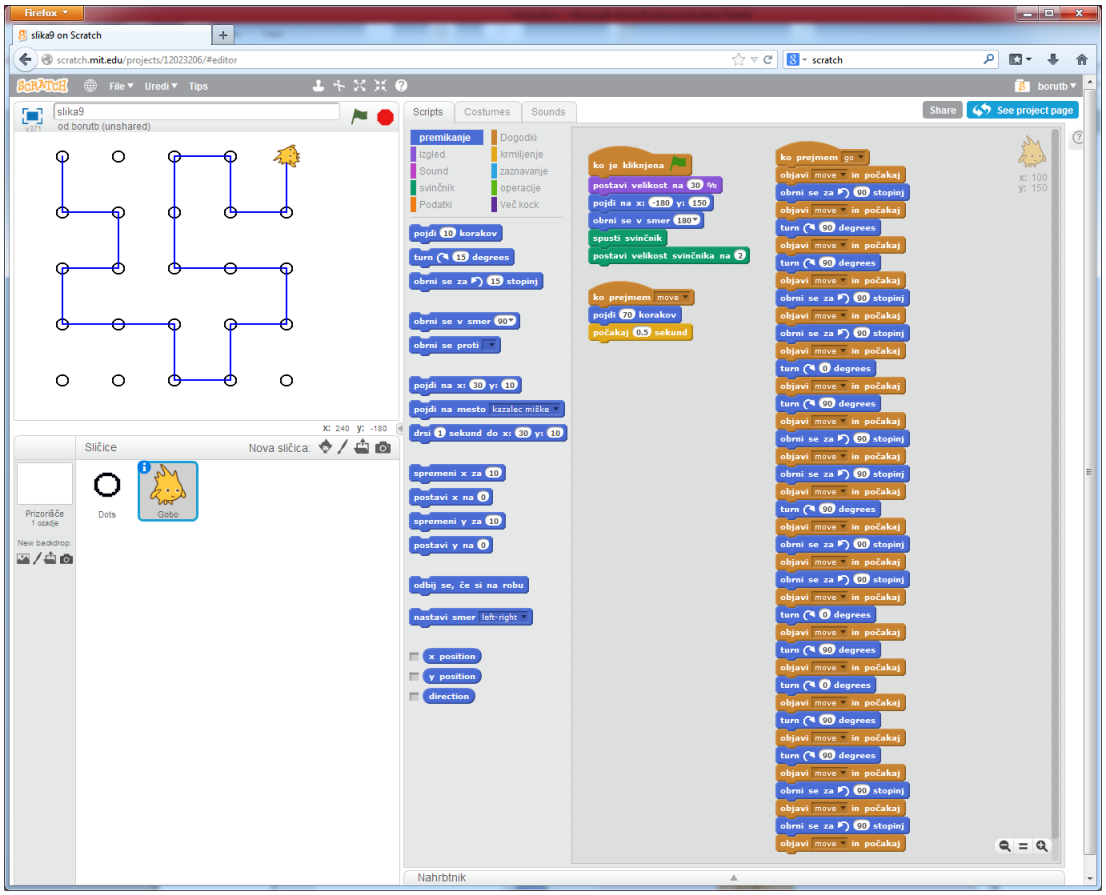
Prednost programskega jezika Scratch je v tem, da je uporabniški vmesnik preveden v slovenščino, čeprav ima prevod še nekaj pomanjkljivosti. Tudi literaturo že lahko najdemo v slovenskem jeziku, na internetu pa imamo ogromno že narejenih programov, ki jih lahko spreminjamo. Kot je značilno za vizualna okolja, v Scratchu ni tipkanja, ni prevajalnika in podobnih strašljivih orodij in pravzaprav ne moremo napisati programa, ki ne bi deloval. Programirate namreč tako, da v program vlečete gradnike, ki program sestavljajo. Gradniki so različnih vrst – od premikalnih (premakni predmet, zavrti se ...) do upravljalnih (ponavljaj n-krat, ponavljaj dokler), sestavljate pa jih lahko le na način, ki zagotavlja pravilno delovanje. Ukaz lahko odložite le na mesto, kjer ima smisel, pri tem pa vam pomaga oblika programskih gradnikov, ki nakazuje, kam določen ukaz spada.

Glavni elementi Scratcha so bitja in predmeti, ki jim v angleščini pravimo *sprite*. Na odru spremljamo premikanje figur, igre in animacije. Položaj figure na odru je določen s pomočjo koordinatnega sistema. Figuro lahko narišeš sam ali pa uporabiš katerekoli slike s svojega računalnika. Figuri lahko napišeš **program**. Figuri lahko zamenjaš videz. Vsaka figura ima lahko tudi svoj seznam **zvokov.** V sklopu multimedije lahko uporabljaš: urejevalnik slik, kamero in snemalnik zvoka. Ukazi so razdeljeni v osem skupin, ki združujejo vsebinsko podobne ukaze. Ukazi znotraj skupine so iste barve.

Koncepti, ki jih Scratch podpira: zaporedje ukazov, zanke, pogojni stavki, spremenljivke, tabele, odzivi na dogodke, vnos podatkov preko tipkovnice, naključna števila, logične operacije. Težje pa boste predstavili podprogram, rekurzijo, dedovanje, definiranja lastnih razredov ter branje in pisanje iz datoteke.

Za izdelavo programov ne potrebujete nameščanja nobenih dodatnih programov, ker razvojno okolje deluje kar v spletnem brskalniku. Tako izdelane programe lahko poganjate v razvojnem okolju ali pa jih zapakirate v spletno aplikacijo.

Primer programa v Scratchu, ki zriše sliko 9, naše prve aktivnosti:

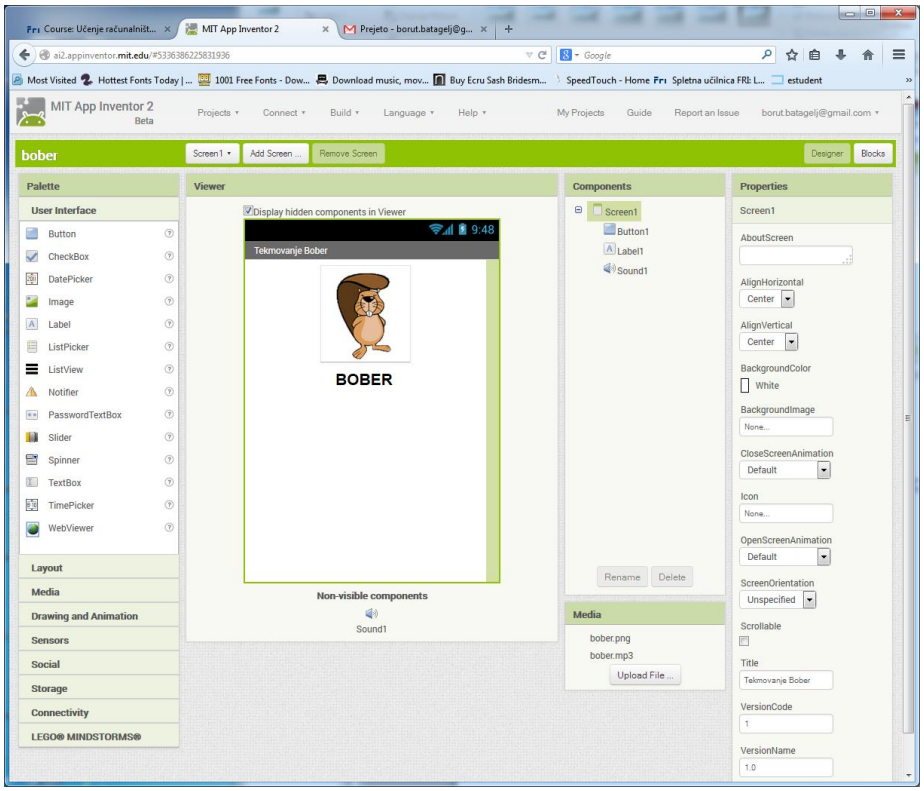


Program: ***slika9.sb2***, izvorna kodo najdete na spletni učilnici FRI (https://ucilnica.fri.uni-lj.si/course/view.php?id=303)

App Inventor

Če vas je okolje Scratch navdušilo, potem boste okolje App Inventor oboževali (slika 4). S pomočjo Scratcha namreč lahko izdelujete programe za računalnik, App Inventor pa vam omogoča, da izdelate čisto pravo igrico za svoj telefon ali tablico, ki uporablja Android. Pri programiranju imate na voljo vse dodatne senzorje, ki jih prenosna naprava omogoča: kamero, mikrofon, ekran na dotik, GPS, senzor premikov. Tako lahko izdelate zelo zanimive interaktivne programe.

Program: ***bober.apk***, izvorno kodo najdete na spletni učilnici FRI (https://ucilnica.fri.uni-lj.si/course/view.php?id=303)



Slika 4: Primer aplikacije za mobilne telefone Android v razvojnem okolju App Inventor.

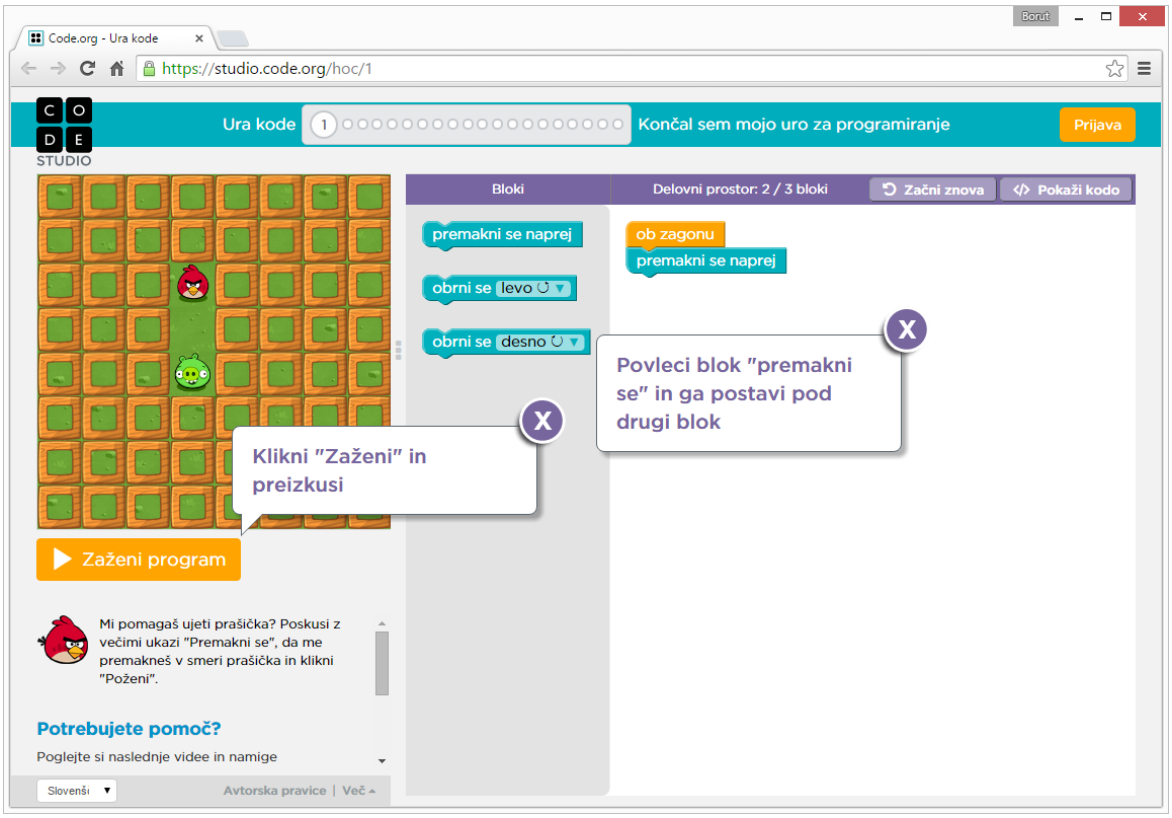
Za najmlajše

Ker je tekmovanje Bober po novem namenjeno tudi našim najmlajšim bobrčkom v 2 razredih osnovnih šol je zelo pomembno, da jih na zabaven način preko igric naučimo logičnega in postopkovnega razmišljanja.

Za ta namen dobimo na spletni strani **Code.org** kar nekaj aktivnosti. Code je ameriška neprofitna organizacija, katere cilj je omogočiti izobraževanje s področja računalništva v vseh šolah čim večjemu številu otrok, ne glede na spol, barvo in poreklo. V okviru kampanje **Ura kode** (angl. Hour of code) ponuja privlačne brezplačne tečaje osnov programiranja v različnih programskih jezikih.

V okviru tega projekta lahko otroci s pomočjo igrice jeznih ptičev napišejo svoj prvi program, ki vodi ptiča do sovražnih prašičev (slika 5). Na takšen način otroci spoznajo osnovne koncepte programiranja, kot so ponavljanje (brezpogojno, pogojno), pogoji, podprogrami, spremenljivke, itd. S tem ko sestavljajo bloke v ozadju sestavljajo svoj program, ki si ga lahko tudi ogledajo. Takšne aktivnosti lahko otroke, predvsem mlajše na zabaven način uvedejo v programiranje in kasneje v bolj kompleksna okolja, kot sta prej omenjeni orodji Scratch in App Inventor.

Še več aktivnosti, ki uporabljajo koncept vizualnega programiranja s pomočjo blokov lahko najdete na: *https://blockly-games.appspot.com/*.



Slika 5: Programiranje za najmlajše v okviru projekta Ura kode.

ScratchJr

Programsko okolje ScratchJr (Scratch Junior) je poenostavljena različica okolja Scratch namenjena otrokom starim od 5 do 7 let. Okolje je na voljo brezplačno za tablične računalnike s sistemom iOS ali Android. Otroci lahko z združevanjem različnih grafičnih blokov z navodili ustvarjajo svoje interaktivne zgodbe in igre. V preprostem grafičnem urejevalniku lahko narišejo svoje junake ali spremenijo obstoječe, dodajo zvoke in govor in celo uporabijo svoje slike s pomočjo vgrajene kamere. Nato s pomočjo programskih elementov naredijo, da junaki oživijo; se premikajo, skačejo, plešejo, pojejo (slika 6).

Slika 6: Programsko okolje ScratchJr namenjeno najmlajšim programerjem.

Teorija: programski jeziki

Za prvo programabilno napravo bi lahko šteli Jacquardove mehanske statve iz leta 1801, s katerimi je omogočal tkanje različnih vzorcev z različnimi programi (luknjane kartice). Pravi program in programiranje pa se je začelo z zasnovo računalniškega modela, ki ga je predlagal John von Neumann leta 1964. Pred tem je bilo možno računalnike reprogramirati samo s pomočjo žic, priključkov ali stikal. V spominu so bili shranjeni samo podatki, ne pa ukazi. Za vsak problem je bilo tako potrebno prežičiti celoten računalnik. Prvi elektronski splošno namenski računalnik - ENIAC je imel na primer 6000 stikal, ki jih je bilo potrebno spremeniti za drugo nalogo. Von Neumann je predlagal, da bi bili ukazi, ki kontrolirajo računalnik, zapisani poleg podatkov v spominu. Tako bi bilo za rešitev novega problema potrebno samo preurediti ukaze, namesto razporejati žice ali stikala – to pomeni, napisati nov program. Tako lahko rečemo, da je programiranje, kot ga poznamo dandanes, izumil von Neumann.

V nadaljevanju si bomo pogledali, kakšen sploh je računalnik, kot si ga je zamislil von Neumann in ki se še danes uporablja, ter kaj sploh je to program. Nato se bomo sprehodili skozi zgodovino in poskušali razumeti, kako so višje-nivojski programski jeziki sploh nastali. Zanimalo nas bo, kako jih računalnik pravzaprav razume, če pa so na nivoju našega jezika, računalnik pa kot vemo, pozna samo enice in ničle.

Računalnik

Računalnik je naprava, ki izvaja računanje in procesira podatke. Računalnik dela pod kontrolo **programa** – množice navodil, ki povejo, kaj mora računalnik delati. Strojna oprema opisuje elektroniko in mehanične dele računalnika. Programska oprema pa so programi, ki kontrolirajo strojno opremo.

Računalnik sestavljajo: izhodne naprave (tiskalnik, ekran, zvočniki), vhodne naprave (tipkovnica, miška, mikrofon, skener, kamera), primarni pomnilnik (hrani podatke in programe, podatki se izgubijo, ko ugasnemo računalnik) in sekundarni pomnilnik (trdi disk, CD, tračne enote, USB ključi, SD kartice), centralna procesna enota CPU, ki skrbi za izvajanje ukazov, aritmetično logična enota ALU, skrbi za računanje in primerjanje, preko signalov sporoča tudi drugim napravam.

Poznamo dve vrti programov: aplikativni in sistemski. Aplikativni skrbijo za določeno nalogo. Sistemski pa naredijo računalnik sploh uporaben. Najpomembnejši takšen program je operacijski sistem (OS).

Program

Računalniški program je nabor **navodil**, ki usmerjajo obnašanje računalnika. Programiranje je umetnost in znanost oblikovanja in pisanja programov. Je kreativna in zabavna aktivnost reševanja problemov. Svojo rešitev lahko preizkusite v obliki izvajajočega programa.

Od strojnega jezika do višje-nivojskih jezikov

Danes se večinoma programira v višje-nivojskih jezikih kot so Java, C++ ali Python. Programski jezik označimo kot višje nivojski, če imajo stavki programa pomen tudi v naravnem jeziku. Vsi našteti programski jeziki imajo na primer pogojni stavek IF, ki ga uporabimo kot: IF pogoj THEN akcija. Če je izpolnjen določen pogoj, izvedi sledečo akcijo. Nekateri programski jeziki imajo lastnosti, ki jih naredijo primerne za pisanje specifičnih programov: COBOL za komercialne programe, FORTRAN za inženirje in znanstvenike ter C in C++ za programe operacijskega sistema. Poleg tega pa uporabljajo notacijo in simbole, ki so ljudem razumljivi. Aritmetične operacije so na primer predstavljene z operatorji +, -, * in /, tako da lahko zapišemo v programu izraz: (a+b)/2.

Pojavi pa se težava, ker računalniki takšnega izraza neposredno ne razumejo. Če hočemo, da bo računalnik razumel tako navodilo, moramo program prevesti v računalniku razumljiv **strojni jezik**, ki ga razume CPU (Centralno procesna enota) oziroma mikroprocesor. Vsak procesor ime svoj strojni jezik, zato tudi imamo programe za različne naprave (operacijske sisteme). Takšni programi se imenujejo **platformsko odvisni programi**.

V splošnem strojni jezik bazira na binarni kodi, ki ima dve stanji, to je 0 ali 1. Tako so vsi ukazi in podatki predstavljeni z binarnimi kodami. Seštevanje dveh števil na primer predstavimo z ukazom opcode: ADD (011110), ki prejme 3 operande, ki predstavljajo lokacije v spominu, kjer se podatki nahajajo:

prvo število na 1. lokaciji: 110110, drugo na 2. lokaciji: 111100 in 3. lokacija predstavlja mesto, kamor se shrani izračunan podatek. Tako bi lahko omenjeno seštevanje napisali v strojnem jeziku kot:

011110 110110 111100 111101

Na prvih računalnikih, ko še ni bilo višje nivojskih programskih jezikov, je bilo potrebno programirati na nivoju ničel in enic. Iskanje napak v takšnih programih je bilo zelo težavno.

Danes ne rabimo več skrbeti glede strojnega jezika, ker lahko uporabimo poseben program, da prevede višje-nivojsko ali **izvorno kodo** programa v strojni jezik ali objektno kodo, ki je edina koda, ki jo lahko izvršimo ali poženemo s pomočjo računalnika. V splošnem rečemo takšnemu programu prevajalec (angl. translator). Tako lahko z ustreznim prevajalcem za jezik Java ali C pišemo programe, kot da bi računalnik razumel ta jezik.

Prevajalci izvorne kode obstajajo v dveh različicah. **Tolmači** ali **interpreterji** (angl. interpreters) prevajajo vrstico za vrstico in izvršijo kodo vrstice preden prevedejo naslednjo vrstico. **Prevajalniki** (angl. compilers) pa prevedejo celoten program v izvršljiv program. Zaradi tega so učinkovitejši, odkrivanje napak pa je težje. Dandanes imamo vse več programskih jezikov (Java, Python), ki združujejo prednosti enih in drugih in tako interpretirajo izvorno kodo v vmesno kodo, ki jo potem prevedejo s pomočjo prevajalnikov.

Zbirni jezik

Leta 1950 se je programiralo v zbirnem jeziku. V primerjavi s strojnim jezikom je bilo to kar zadovoljivo programsko okolje. Ljudje, ki so programirali, so bili tehnično usmerjeni, poznali so delovanje računalnika in so lahko s pisanjem programov prihranili čas izvajanja, kar je pri takratnih računalnikih veliko pomenili, saj so bili viri zelo omejeni.

V naslednjih desetletjih se je pojavila potreba, da bi tudi netehnični ljudje pisali programe. Pojavila se je potreba po višje-nivojskih jezikih. V istem času je tudi računalnik postal zmogljivejši kar je omogočilo, da je bilo daljše izvajanje sprejemljivo. Tudi računalniški viri niso bili več tako omejeni.

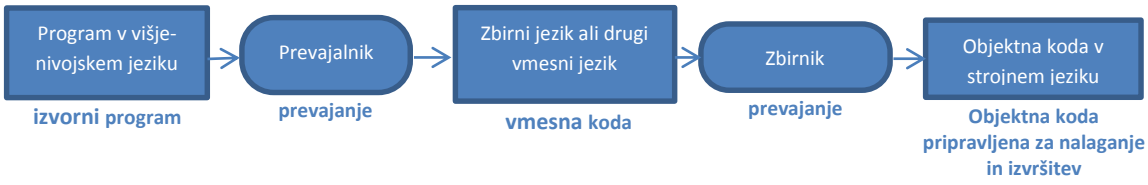
V zbirnem jeziku - zbirniku moramo paziti in poznati lokacijo v spominu. Vsak ukaz se prevede v strojni ukaz. Še več, program napisan za stroj X ne teče na drugem stroju, ki ima drugačne ukaze. Vse to so glavne pomanjkljivosti zbirnega jezika.

Visoko-nivojski jeziki odpravijo te pomanjkljivosti:

- Programer ne rabi skrbeti za nalaganje podatkov iz spomina oziroma vedeti, kje se ti podatki v spominu nahajajo.
- Ne rabi poznati nalog na najnižjem nivoju, ampak se lahko skoncentrira na reševanje problema na višjem nivoju, kot je v našem primeru sešteti dve števili v spremenljivko C.
- Programi so prenosljivi in niso odvisni od naprave.
- Izrazi so na nivoju naravnih jezikov in uporabljajo standardne matematične notacije.

Visoko-nivojske programske jezike imenujemo tudi jeziki tretje generacije, ki nakazujejo napredek iz strojnega jezika (prva generacija) v zbirni jezik (druga generacija). S tem smo dosegli nov nivo abstrakcije, ki nas še bolj oddalji od nizko nivojskih elektronskih komponent naprave.

Tako kot smo za prevod iz kode zbirnika v strojno kodo potrebovali zbirnik (angl. assembler), sedaj potrebujemo nov prevajalnik (angl. compiler), ki prevede izvorno kodo višjega programskega jezika v vmesno kodo. Ta vmesna koda se nato prevede s pomočjo zbirnika v objektno kodo.



Večina visoko-nivojskih jezikov je tako imenovanih **postopkovnih jezikov**. Za te jezike je značilno, da program sestavlja zaporedje ukazov, ki se izvršujejo in rešijo določeno nalogo. To sovpada z arhitekturo von Neumanovega računalnika, ki opisuje zaporedje ciklov: prenos + izvajanje. Tako so tudi osnovne operacije postopkovnih jezikov predvsem shranjevanje in pridobivanje vrednosti. Različni jeziki se razlikujejo tako po pravilih, kako morajo biti stavki napisani (sintaksa oz. skladnja), kot tudi po pomenu pravilno napisanih stavkov (semantika).

Sintaksa programskega jezika enolično **določa obliko dovoljenih izrazov** v danem programskem jeziku. Če program ni napisan v skladu s sintaktičnimi pravili jezika, prevajalnik odkrije napake v programu in uporabnika z ustreznimi sporočili na to opozori.

Semantika programskega jezika enolično **določa interpretacijo (pomen) izrazov** v danem programskem jeziku. Na splošno prevajalnik ne more odkriti semantičnih napak v programu, saj ne ve, kaj je uporabnik s programom hotel opisati. Do določene mere je semantika določena s sintakso programskega jezika. Semantične napake, vezane na sintakso jezika, lahko prevajalnik odkrije (npr. spreminjanje vrednosti konstante).

Namenski programski jeziki

Čeprav imajo omenjeni postopkovni programski jeziki (COBOL, FORTRAN, Pascal, C++, C#, JAVA, Python) močna področja, vsi veljajo za splošno namenske programske jezike. Poznamo pa tudi jezike, ki so namenjeni za določeno področje: podatkovne baze (SQL), spletne strani (HTML, JavaScript).

Druge paradigme programiranja

Paradigma postopkovnega programiranja pravi, da zaporedje ukazov posredujemo računalniku. Vsak ukaz dostopa ali spremeni vsebino v spominu računalnika. Če računalnik zaporedno izvaja ukaze, je končna rešitev zadnje stanje v spominu.

Pravzaprav programiranje sestoji iz dveh delov: najprej načrtovanje algoritma-postopka, potem pa zapis nedvoumnih operacij kot zaporedje ukazov. Pri postopkovnem načinu moramo na rešitev naloge gledati kot na reševanje **korak po koraku**. Tudi pri objektnem programiranju način ostaja enak, samo da so koraki porazdeljeni med posamezne podnaloge razredov.

Poznamo še druge načine programiranja – programiranje, ki bazira na drugih paradigmah. Tako bi lahko primerjali učenje postopkovnega jezika z učenjem nemščine, španščine , italijanščine (različni, a podobni jeziki) in se sedaj hočemo naučiti arabsko, japonsko ali znakovni jezik – jeziki ki so popolnoma drugačni po obliki, strukturi in abecedi.

Funkcijsko programiranje

Pri funkcijskem programiranju vsako nalogo opišemo s pomočjo funkcije. Tukaj je funkcija mišljena kot matematična funkcija: f(x)=2*x. Funkcija vzame argument ali več argumentov in vrne rezultat. Poznamo primitivne funkcije, ki so del jezika. Druge lahko napišemo sami. Pri klicanju funkcij velikokrat gnezdimo tudi funkcije v samo funkcijo, tako da je **rekurzivni način** prevladujoči način pri funkcijskem programiranju.

Logično programiranje

Pri funkcijskem programiranju se oddaljimo od implicitnega podajanja navodil za posamezen korak, ki ga mora računalnik narediti. Namesto tega določimo transformacije podatkov - funkcije in njihovo kombinacijo, ki nas pripelje do rešitve.

Pri logičnem programiranju gremo še korak naprej s tem, da ne podamo natančno, kako naj bo naloga rešena. Enostavno samo naštejemo **dejstva**, ki veljajo, in potem logični program sklepa nadaljnja. Logičnim programskim jezikom pravimo tudi deklarativni jeziki (v nasprotju z ukaznimi jeziki), ker v programe namesto ukazov zapisujemo veljavna dejstva.

Program sestavljajo **dejstva** in **pravila**. Programer zgolj pove dejstva in pravila določenega področja, ne potrebuje pa podajati navodil računalniku korak po koraku, kako pride do odgovora na določeno povpraševanje. V nasprotju z ukaznimi jeziki pri deklarativnih jezikih opišemo KAJ naj program naredi in ne KAKO naj to naredi.

Paralelno programiranje

Čisto za konec omenimo še **paralelno programiranje**, ki dandanes, ko imamo večjedrne procesorje oziroma več računalnikov povezanih v mreže, pridobiva na pomenu. Naloga paralelnih programskih jezikov je čim bolj zaposliti vse procesne enote. Pojavlja pa se tudi vse večja težnja, da se paralelizem vključi v same prevajalnike, ki zaporedno napisan program sami porazdelijo med razpoložljive procesorske enote.

Programi kot zaporedja ukazov v sklopu tekmovanja BOBER

Večina nalog sega na področje postopkovnega programiranja – postopkovnih programskih jezikov. To se pravi, da računalniku posredujemo zaporedje ukazov. Potem pa računalnik zaporedno izvaja ukaze korak po koraku. Pri tem vsak ukaz spreminja stanje in končna rešitev je zadnje stanje.

Naloge iz področja programiranja lahko razdelimo v 4 skupine.

1. V prvo skupino spadajo naloge, ki **podajajo zaporedje ukazov**, ki mu moramo **slediti**, da rešimo nalogo. Slediti moramo torej programu – navodilom. Nekatere naloge ponujajo več možnih rešitev med katerimi moramo poiskati tisto, ki je najbolj optimalna (najkrajša, najhitrejša)

2. V drugo skupino spadajo naloge pri katerih moramo takšno **zaporedje ukazov zapisati** sami, da potem napisan program reši nalogo.

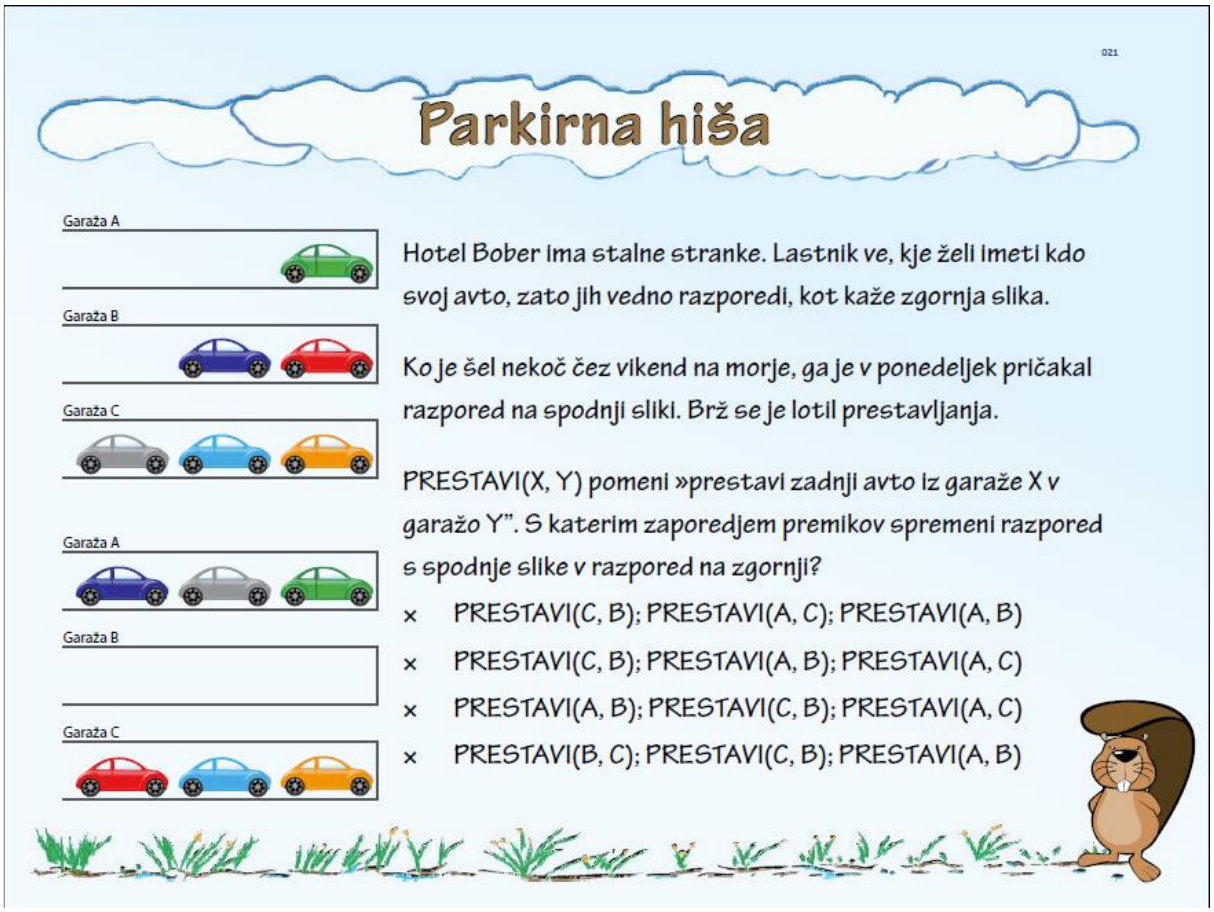
3. Pri bolj zahtevnih nalogah ukazi niso trivialni ali pa je naloga težja zaradi samega razumevanja notacije. Tako lahko v tretjo skupino uvrstimo naloge, ki za razliko od prejšnjih skupin ne podajajo natančnega opisa ali zaporedja ukazov temveč samo **opisujejo postopek** iz katerega moramo sami razvozlati zaporedje ukazov.

4. V ločeno skupino pa lahko uvrstimo naloge, ki zahtevajo od nas naprednejše sledenje postopku. Pri teh nalogah moramo razumeti algoritem, pravila igre in razmisliti o tem, kako ga optimalno uporabiti oziroma v okviru pravil igre **poiskati optimalno rešitev**. Pri teh nalogah ni dovolj, da najdemo rešitev ampak moramo poiskati najbolj optimalno rešitev.

V nadaljevanju bomo spoznali nekaj primerov takšnih nalog. Naloge so zbrane iz seznama nalog iz preteklih let (http://dajmi.fri.uni-lj.si/bober/bober.pdf) ter nalog iz šolskega in državnega tekmovanja v letu 2013/14 (http://dajmi.fri.uni-lj.si/bober/2013-osnovna-sola.pdf) in letu 2014/15 (http://dajmi.fri.uni-lj.si/bober/2014-knjizica.pdf). Naloge so smiselno razdeljene po zgoraj opisanih skupinah.

1. Zaporedje ukazov – sledenje programu

Naloge podajo **zaporedje ukazov**, ki mu moramo slediti, da rešimo nalogo.



Nalogo lahko rešimo, da poskušamo zapisati zaporedje ukazov, ki bo rešilo dani problem ali pa **sledimo ukazom** in si zapisujemo vmesna stanja med izvajanjem.

Imamo	PRESTAVI(C,B)	PRESTAVI(A,C)	PRESTAVI(A,B)		Končno stanje:
A:V,S,Z B: C:R,M,O	A:V,S,Z B:R C:M,O	A:S,Z B:R C:V,M,O	A:Z B:S,R C:V,M,O	X	A:Z B:V,R C:S,M,O

Imamo	PRESTAVI(C,B)	PRESTAVI(A,B)	PRESTAVI(A,C)		Končno stanje:
A:V,S,Z B: C:R,M,O	A:V,S,Z B:R C:M,O	A:S,Z B:V,R C:M,O	A:Z B:V,R C:S,M,O	=	A:Z B:V,R C:S,M,O

Imamo	PRESTAVI(A,B)	PRESTAVI(C,B)	PRESTAVI(A,C)		Končno stanje:
A:V,S,Z B: C:R,M,O	A:S,Z B:V C:R,M,O	A:S,Z B:R, V C:M,O	A:Z B:R,V C:S,M,O	X	A:Z B:V,R C:S,M,O

Imamo	PRESTAVI(B,C)	PRESTAVI(C,B)	PRESTAVI(A,B)		Končno mora:
A:V,S,Z B: C:R,M,O	A:V,S,Z B: C:R,M,O	A:V,S,Z B:R C:M,O	A:S,Z B:V,R C:M,O	X	A:Z B:V,R C:S,M,O

Računalniško ozadje - razhroščevanje

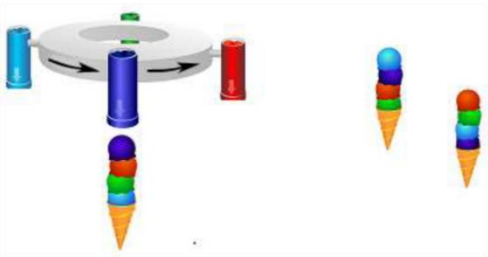
Če ste že preizkušali programe, se vam je morda že kdaj zgodilo, da program ni naredil točno tistega, kar ste hoteli. To se dogaja pogosto. Pri razhroščevanju ugotavljamo kaj računalnik počne (korak po koraku) in kaj moramo storiti, da bo delal, kar bi hoteli mi. To je lahko precej zahtevno. (Iz lastnih izkušenj vam lahko povem, da lahko tudi najmanjšo napako iščemo cel teden ali več.)

Naloga iz tekmovanja 2013/14: ***Avtomat za sladoled***

Avtomat za sladoled

V slaščičarni Sladki hlod imajo avtomat za sladoled. Vanj postavijo kornet in vrtljiva glava vanj naloži štiri kepice. Vrstni red kepic je vedno enak: vijolični vedno sledi modra, modri zelena in tako naprej. S katero kepico začne, je odvisno od tega, kako je obrnjen, ko podstavijo kornet.

Slika kaže ravnokar pripravljeni sladoled, ki se začne z modro kepico in še dva druga sladoleda, pripravljena s tem avtomatom.



Pred slaščičarno ližejo sladoled štirje bobrčki. Le eden je kupil sladoled pri Sladkem hlodu. Kateri?

A. B. C. D.

Avtomat za sladoled je kot program, ki "izpisuje" kepice sladoleda. Tako, kot je potrebno v nalogi odkriti vrstni red kepic, bi nas lahko pri programu zanimalo, kaj bo izpisal in v kakšnem vrstnem redu.

Če pogledamo ponujene rešitve, lahko vidimo da se vsi sladoledi začnejo z rdečo. Zaradi vrtenja avtomata rdeči sledijo vijolična, modra in zelena, tako kot pri sladoledu A. Pri reševanju je potrebno tudi paziti, da začnemo zaporedje pri prvi kepici spodaj tako kot se sladoled naklada in ne pri vrhnji kepici.

Podobne naloge iz prejšnjih let: *Skladiščenje hlodov*, *Stroj za prestavljanje krožnikov* (sklad)

Kje je Franci?

Sledimo opisu poti in sicer iz vsake hiše. Za razliko od naloge Parkiranje (glej naslednje poglavje) imamo tukaj več možnih rešitev, zato sledimo postopku. Razlika od naloge Parkiranje in naslednje naloge z Bagrom je tudi, da tukaj nimamo ukaza naprej, ki določa premik do naslednjega križišča ampak se samo odločamo v posameznem križišču kam bomo šli.

Nalogo lahko rešimo tudi tako, da gremo ritensko od šole in beremo pot nazaj. Pri tem moramo paziti, da ustrezno zamenjamo levo v desno in obratno. Na prvi ali drugi način ugotovimo, da Franci živi v hiši 2.

Bager

Pri tej nalogi sledimo ukazom. Ugotovimo, da nas več različnih postopkov pripelje do rešitve. Izbrati moramo najbolj optimalno. Učenec sledi izvajanju programa. Pri tem izloči programa (postopka) A in D, ki imata napačna 2 oziroma 4 ukaz. Razumeti mora tudi različne čase trajanja ukazov in s tem izločiti program C, ki je zaradi daljših obratov počasnejši. Torej je najbolj optimalna rešitev B.

Naloga iz tekmovanja 2013/14: *Čudežni tuneli, Robotrot (iščemo napako v programu), Pretakanje vode, Popotna darila, Rože*
Naloga iz tekmovanja 2014/15: *Pokvarjena ura, Preurejanje, za srednje šola: Sumljive naprave, Gobelin, Pot skozi labirint*

Natakar

Natakar Bob je dobil naročilo za jagodni sok, limonin sok, pomarančni sok in vodo (H_2O). Ko je napolnil kozarce, je opazil, da so označeni (slika sadja) in da ni uporabil pravih kozarcev za pravi sok.

- V kozarcu 1 je namesto jagodnega pomarančni sok.
- V kozarcu 2 je namesto limoninega soka jagodni sok.
- V kozarcu 3 je namesto pomarančnega soka limonin sok.
- V kozarcu 4 je voda.

Sklenil je popraviti napako. Na voljo ima le te štiri kozarce. Sokov ne sme mešati ali jih zliti v umivalnik. V umivalnik sme, če hoče, izliti le vodo.

Izberi pravo zaporedje prelivanja sokov.

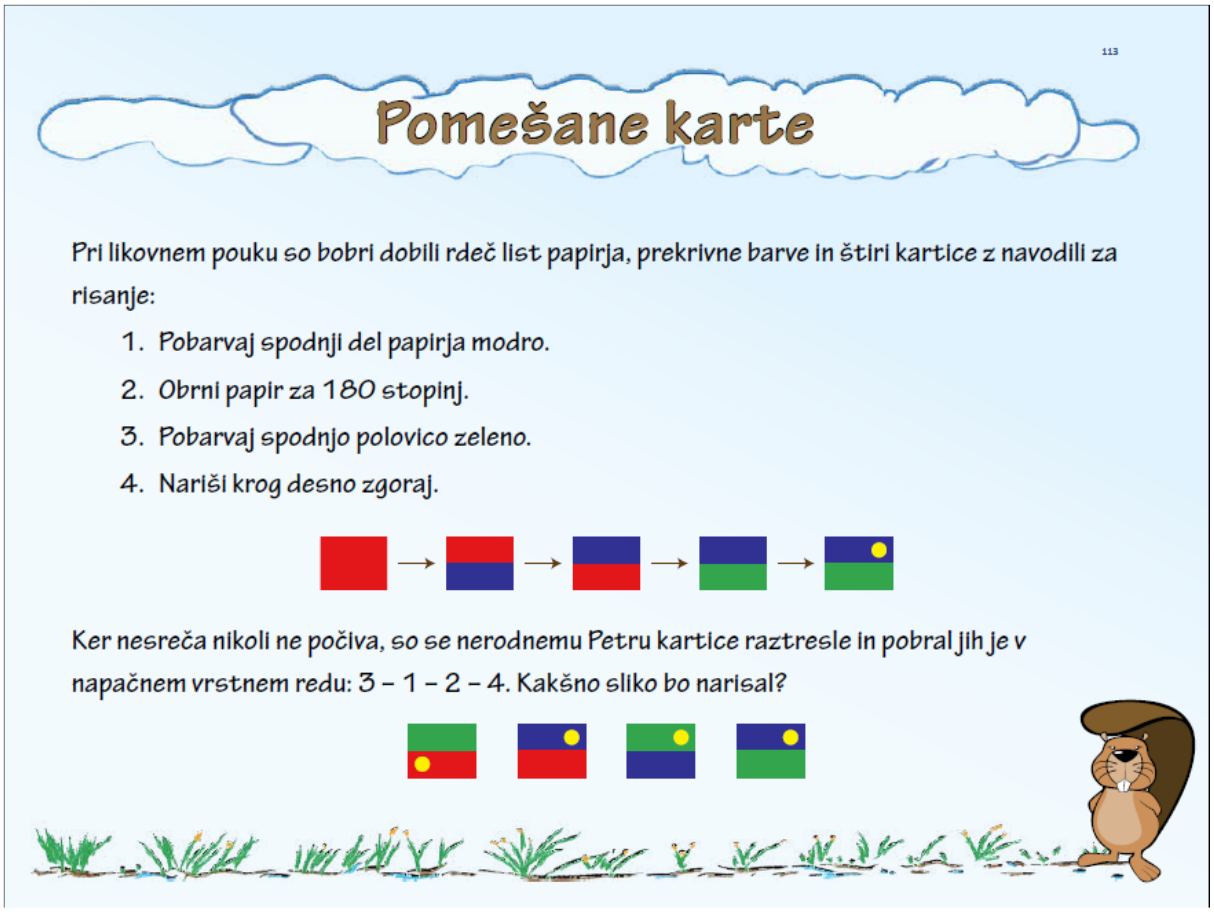
A.	B.	C.	D.
2 prelije v 1	4 izlije v umivalnik	2 prelije v 1	4 izlije v umivalnik
3 prelije v 2	1 prelije v 4	3 prelije v 2	2 prelije v 1
1 prelije v 3	2 prelije v 1	2 prelije v 1	3 prelije v 2
	3 prelije v 2		1 prelije v 3
	4 prelije v 3		natoči vodo v 4
	natoči vodo v 4		

Tukaj imamo na voljo 4 različne programe. Naša naloga je, da sledimo izvajanju programov, kot programer, ki išče napako v programu.

Takoj lahko izločimo postopka A in B, ker se prelivanje začne ne, da bi izlili vodo v umivalnik. Tudi da je rešitev D napačna kmalu ugotovimo, ker v drugem koraku prelivamo iz polnega kozarca 2 v polni kozarec 1. V pravilni rešitvi B v vsakem koraku prelivamo v tisti kozarec, ki ga v prejšnjem koraku izpraznimo.

Računalniško ozadje - spremenljivke

Lahko si predstavljamo da so kozarci v spremenljivke programa. Nekakšne posode v katere shranjujemo vrednosti, številke. Najprej moramo narediti prostor (izlijemo vodo v umivalnik). Nato prestavimo sok iz enega od preostalih treh kozarcev v prostega. V naslednjih treh korakih vedno napolnimo prazen kozarec s pravim sokom. Ko končamo, ostane četrti kozarec prazen, v katerega ponovno natočimo vodo.



Računalniško ozadje - vrstni red ukazov

Tukaj imamo zopet seznam navodil – ukazov, ki mu moramo slediti, da izdelamo sliko. Enostavno sledimo postopku in končno stanje je narisana slika.

Kaj se lahko še naučimo iz omenjene naloge? Spoznamo, da je vrstni red ukazov lahko zelo pomemben. Pri nalogi smo videli, da je drugačen vrstni red pri pomešanih kartah dal drugačen končni rezultat – drugačno sliko.

Podobne naloge: *Slika iz štampiljk, Ugašanje* (vrstni red ni pomemben)
Za najmlajše (naloge 2014/15): *Nalepke, Zobne krtačke*

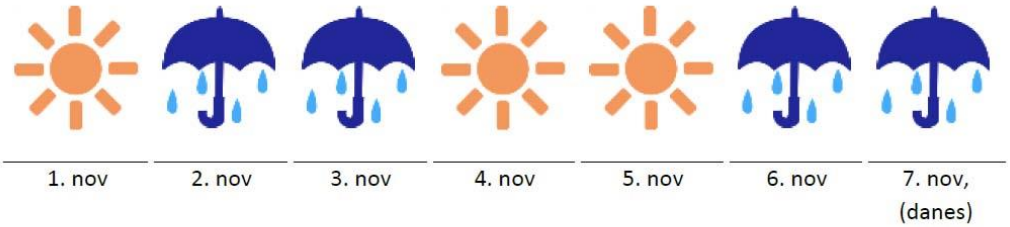
Igrišče

Šolsko, 4. in 5. razred

Bober Marko se takole odloči, kje se bo igral:

- Če je sončno, gre plavat v reko.
- Če dežuje, vendar je bilo včeraj sončno, se igra v hiši.
- Če dežuje že natančno dva dni zapored, se igra na rečnem bregu.
- Če dežuje že vsaj tri dni zapored, se ne igra.

Oglej si, kakšno je bilo vreme v zadnjem tednu. Danes smo sedmega novembra. Kje se bo igral?



1. nov	2. nov	3. nov	4. nov	5. nov	6. nov	7. nov, (danes)

Računalniško ozadje – pogojni stavek

Ena od osnovnih sestavin računalniških programov je pogojni stavek, s katerim določimo, da naj se nek del programa izvede le, kadar drži določen pogoj. Ko se Marko odloča, kje se bo igral, izvaja preprost "program".
Trenutno stanje je: dežuje.
1. stavek: Pogoj ni izpolnjen ker ni sončno.
2. stavek: Tukaj morata biti izpolnjena dva pogoja: prvič mora deževati in drugič je bilo včeraj sončno. Med pogojema je veznik in (angl. AND), ki določa, da je pogoj v celoti izpolnjen samo, če sta izpolnjena oba pogoja. V tem primeru ni izpolnjen drugi pogoj.
3. stavek: Pogoj, da dežuje dva dni je izpolnjen, saj dežuje drugi dan zapored.
4. stavek: Pogoj ni izpolnjen, ker ne dežuje tri dni zapored.

Pri pogojnem izvajanju programa lahko imamo še vezni člen ali (angl. OR), ki določa da mora biti izpolnjen vsaj en izmed naštetih pogojev.

Pri nalogi ponovno sledimo ukazom, ki izriše cvet. Tudi tukaj lahko vidimo, da je zaporedje ukazov zelo pomembno: rumena – oranžna – zelena – rdeča. Spoznamo pa še en koncept programiranja in to je ponavljanje.

Računalniško ozadje - ponavljanje

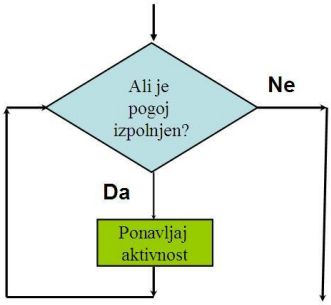
Velikokrat se srečamo z izvajanjem opravil, kjer moramo večkrat ponoviti isto aktivnost. Dober primer takšnega opravila je peka palačink. Za pripravo palačinke moramo najprej v posodo za peko naliti olje, nato v posodo nalijemo mešanico mleka, jajc in olja, palačinko nato pečemo najprej na eni strani, nato jo obrnemo in popečemo še na drugi strani. Ko je palačinka pečena, jo damo na krožnik, namažemo in zvijemo. Postopek peke palačink ponavljamo dokler nismo spekli želeno število palačink ali dokler nam ne zmanjka mase za palačinke.

Podobno opravilo je tudi prenašanje opek iz enega mesta na drugo. Takšen prenos opek lahko opišemo z naslednjim postopkom (psevdokodom):

dokler nisi preložil vse opeke
    vzemi opeko
    prenesi opeko
    spusti opeko

V splošnem lahko vsako ponavljanje opravila zapišemo na način:

dokler (pogoj)
    stavek



To lahko razumemo kot: Dokler je pogoj izpolnjen: Izvajaj stavek. Ko pogoj ni več izpolnjen: Izvedi stavek, ki sledi strukturi ponavljanja. Običajno imamo na razpolago še zanko, ki ji podamo kolikokrat se ponovi (zanka **for**).

Podobne naloge: *Bobri predejo mrežo* (razumevanje notacije)
Naloga iz tekmovanja 2014/15: *Vzorci iz hlodov, Kvadrati (srednja šola)*

2. Zapis korakov

Nalogo moramo zapisati kot zaporedje korakov, ki reši dani problem: v tem primeru pripeljat avto do garaže. Zaporedju ukazov lahko rečemo tudi program.

Pri tej nalogi moramo paziti, ker imamo tudi ukaz naprej (N), ki avto dejansko premakne do drugega ovinka.

Pravilen zapis ukazov: N, L, N, D, N, L, N,D, N

Iz zaporedja ukazov lahko ugotovimo, da se del ukazov ponavlja (podčrtan del). Tako, da lahko v okviru tega spregovorimo tudi o konceptu podprograma, ki je pri programiranju zelo pomemben.

Računalniško ozadje - podprogrami

Če opazimo ponavljajoče bloke zaporedja ukazov v programu lahko program zapišemo z manj ukazi tako, da ponavljajoči blok zapišemo v tako imenovan podprogram. Ta podprogram nato kličemo iz glavnega programa. Tako bi lahko zgornji program zapisali kot:

MAIN: P1, P1, N

P1: N,L,N,D

Običajno podprogrami zaokrožujejo določeno nalogo. V različnih programskih jezikih jih imenujemo procedure, funkcije, rutine, podprogrami. Pri objektnem programiranju jih imenujemo metode in

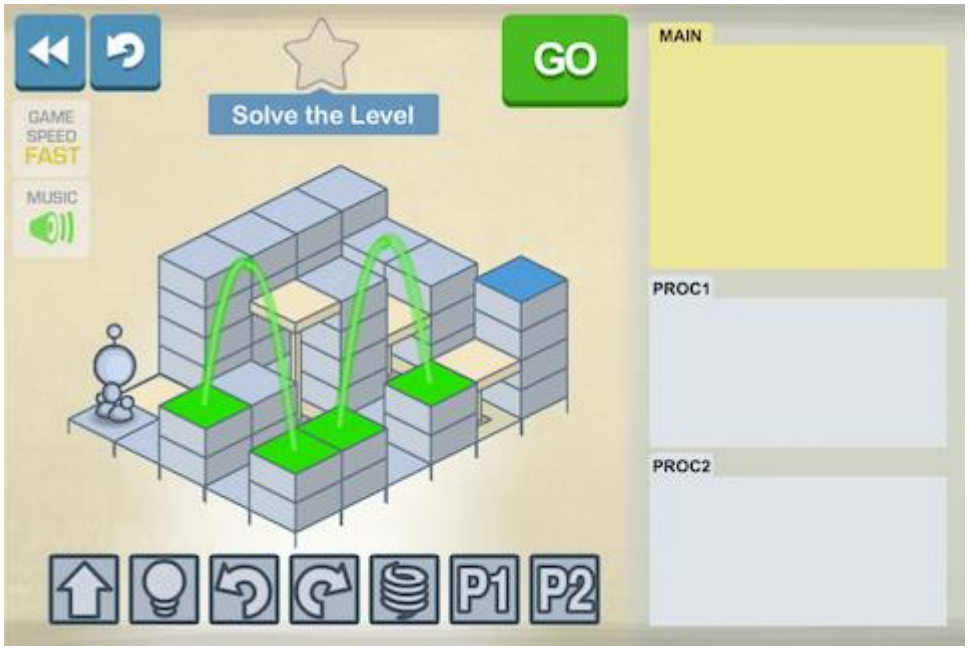
vezane na posamezne objekt. Lahko jih definiramo v programu ali pa v knjižnicah, tako da so na voljo tudi drugim programom. Podprogram je lahko napisan tako, da sprejme parametre – podatke iz glavnega programa in vrne vrednost glavnemu programu.

Podobne naloge, ki od nas zahtevajo zapis ukazov: *Žabin sprehod*, *Pot do cveta* (binarno drevo: levo ali desno)

Naloga iz tekmovanja 2013/14: *Barvobot*
Naloga iz tekmovanja 2014/15: *Risanje, Zidni robot*

Zanimivo: Igrica **Light bot** (http://light-bot.com/), ki nas uči programiranja ter spoznavanja konceptov, kot so podprogram, pogoji, zanke: Na voljo za iOS, Android, Windows, Kindle in Flash (slika 7).

Obstaja tudi podobna brezplačna različica **Light bot 2**: http://www.silvergames.com/light-bot-2 in podobna brezplačna različica za mobilne naprave Android **MYBOT.**



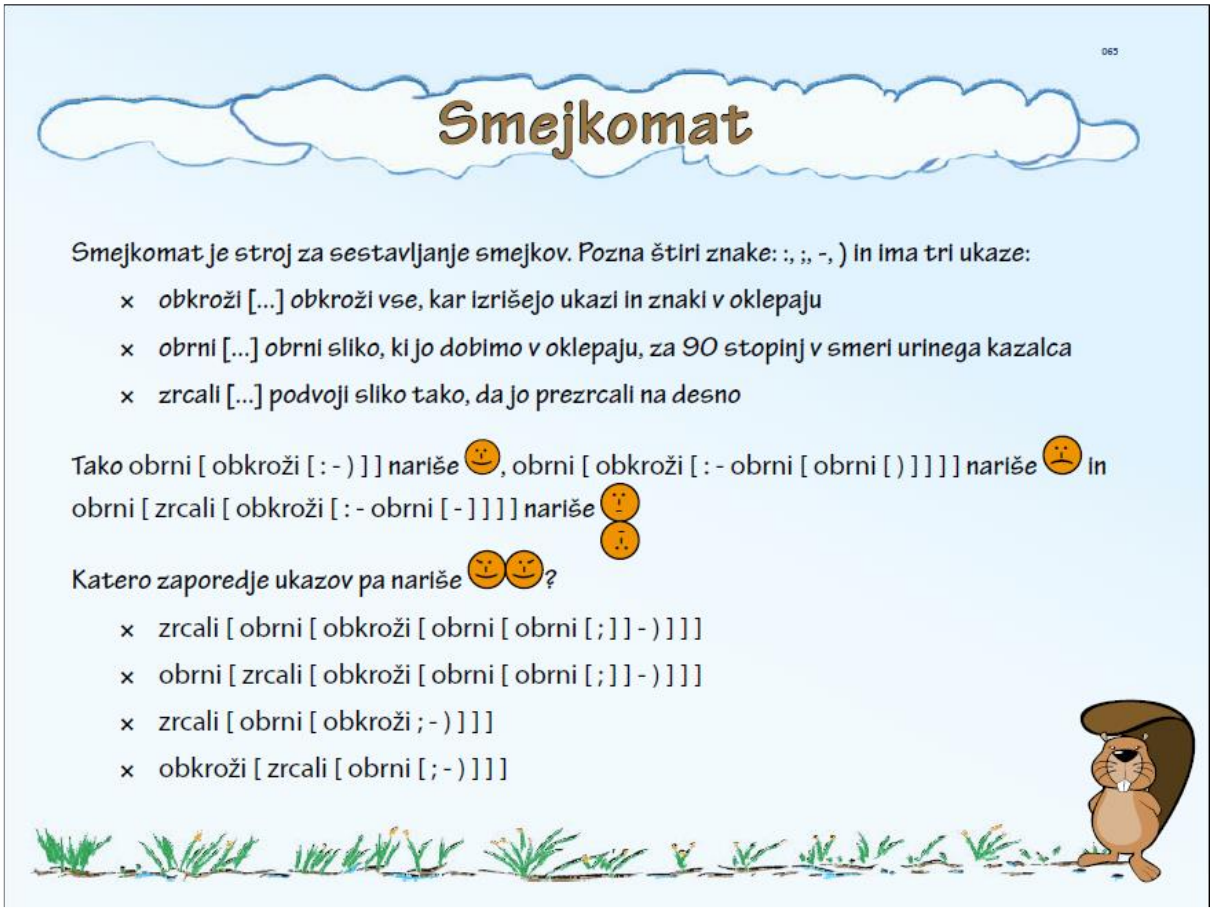
Slika 7: Igrica Light bot, ki nas uči osnov programiranja.

Naloga iz tekmovanja 2013/14: **Labirint**

Zelo podobna naloga, ki so jo učenci, ki so preigravali igrico Light bot brez težav rešili. Do rešitve lažje pridemo, če si sami zapišemo zaporedje ukazov, ki pripelje robota do zaklada.

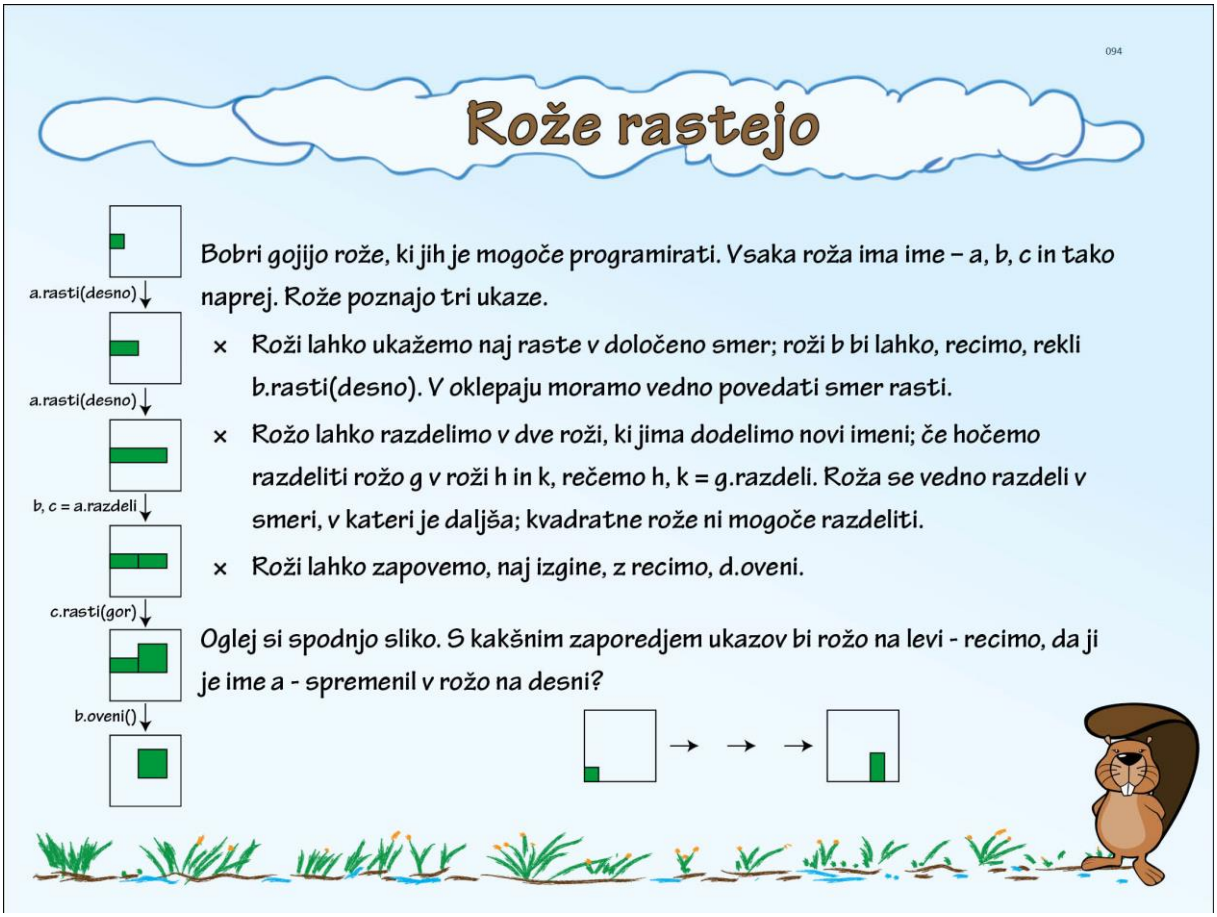
Paziti moramo, ker v navodilih piše, da ga moramo pripeljati tudi nazaj. Lahko si zopet zapišemo ukaze ali pa prepišemo prejšnje zaporedje ukazov v obratnem vrstnem redu s tem da zamenjamo ukaz za naprej z nazaj in gor z dol in obratno. Oziroma naloga je še lažja, ker samo ena rešitev uspešno pripelje robota na cilj.

3. Razumevanje opisa



Računalniško ozadje - funkcijsko programiranje

Pri tej nalogi imamo ravno tako ukaze samo, da so ti ugnezdeni. To nalogo lahko uvrstimo v 3 skupino, ker nimamo zapisanega pravega zaporedja ukazov – kot v postopkovnih programih ampak je program zapisan bolj v obliki funkcijskega programiranja. Znotraj posamezne funkcije imamo drugo funkcijo.



a.rasti(desno); a.rasti(desno); b,c = a.razdeli; b.oveni; d, e = c.razdeli; d.oveni; e.rasti(gor)

Računalniško ozadje - *objektno programiranje*

Ozadje naloge je programiranje. Točneje, oblika programiranja, ki mu pravimo objektno programiranje; rože so "objekti", ki jim lahko, s pomočjo metod, "naročamo", kaj naj storijo.

Podobne naloge: *Pirhi, Dvoštevila, Ujemi barvo*
Naloga iz tekmovanja 2013/14: *Jabolka*

Da rešimo nalogo, moramo razumeti "program", po katerem dela Adam. V programu se skrivajo tudi pogoji stavki (IF…ELSE…): če (if) leva košara še ni prazna naredi nekaj, drugače (else) naredi nekaj drugega.

Odtisi stopinj

Državno, 6. - 9. razred in srednja šola

Bobri s tacanjem rišejo drevesa. Nanja se dobro spoznajo, zato so jim nadeli imena.

1-drevo narediš takole:

```
Naredi 1 korak naprej in naredi odtis stopinje.
Naredi korak nazaj.
```

2-drevo narediš takole:

```
Naredi 2 koraka naprej, na vsakem koraku naredi
odtis stopinje.
Obrni se desno in naredi 1-drevo.
Obrni se levo in naredi 1-drevo.
Naredi 2 koraka nazaj.
```

Kako narediš 3-drevo, lahko že uganemo:

```
Naredi 3 korake naprej, na vsakem koraku naredi
odtis stopinje.
Obrni se desno in naredi 2-drevo.
Obrni se levo in naredi 2-drevo.
Naredi 3 korake nazaj.
```

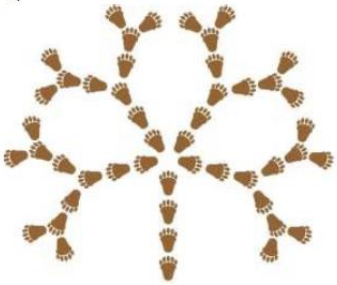
Kako je videti 4-drevo?

A)

B)

C)

D)

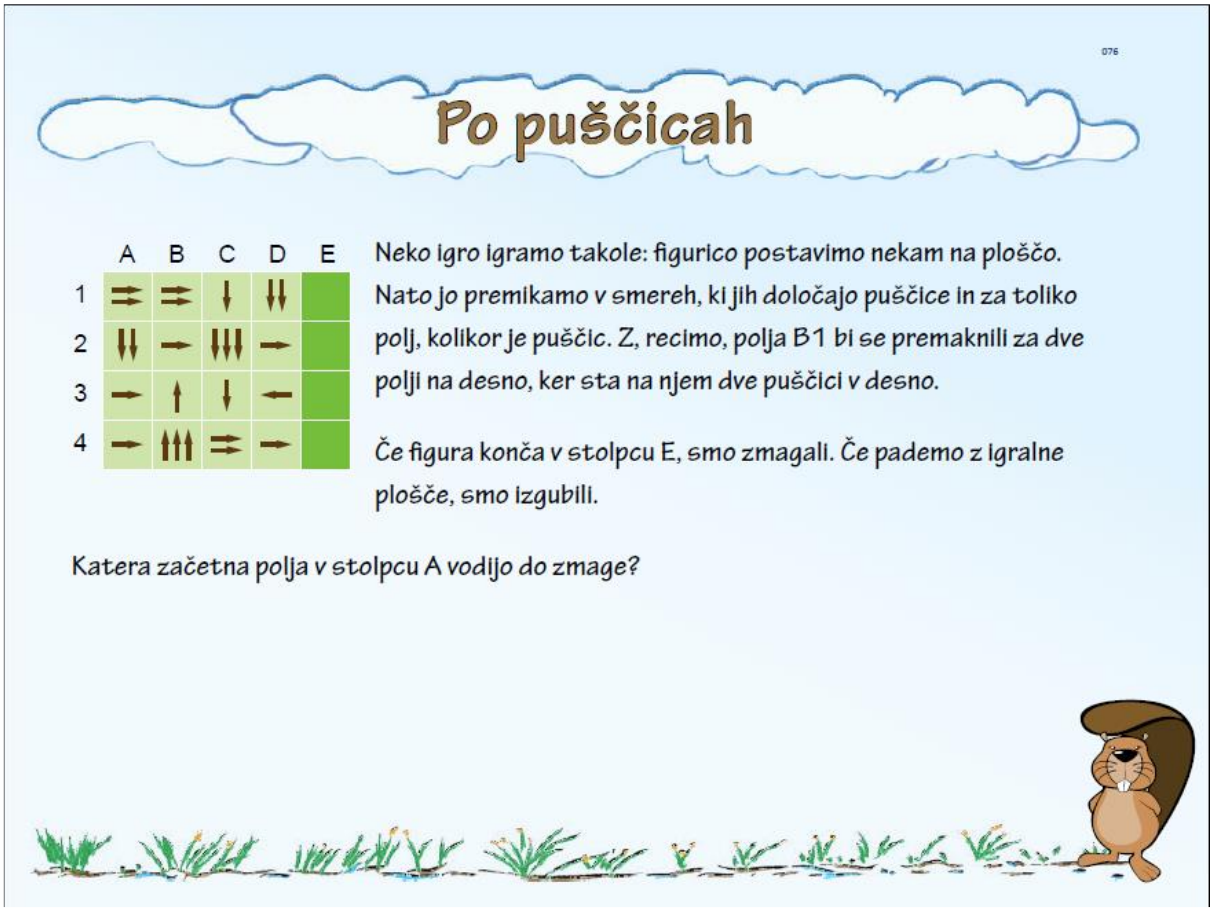


Računalniško ozadje - rekurzija

Vzorec, ki mu sledijo bobri, je zanimiv, ker lahko z njim narišemo poljubno veliko drevo. X-drevo je sestavljeno iz X korakov naprej in dveh (X-1)-dreves, ki ju lahko sestavimo iz X-1 korakov in dveh (X-2) dreves in tako naprej, dokler ne pridemo do 1-dreves. Takšnim načinom opisovanja pravimo "rekurzivni opis" in ga pogosto uporabljamo pri programiranju, saj nam lahko olajša razmišljanje o kakih zapletenih problemih s tem, da jih spremeni v manjše, preprostejše.

Podobna naloga iz tekmovanja 2014/15: *Povej svoje ime*

4. Poiskati moramo rešitev



Ker nas naloga sprašuje samo za stolpec A lahko enostavno preverimo za vse kvadratke v temu stolpcu. Če pa bi hoteli preveriti za vse kvadratke na polju sledimo postopku:

Za vse kvadratke, ki še niso označeni:

> Pot katera pripelje do cilja označimo, drugače jo prečrtamo.
>
> Če pridemo na označen kvadratek, pot končamo.

Lahko tudi obratno: Pogledamo kateri kvadratki nas pripeljejo na cilj in jih označimo. Potem preverimo za vse označene kvadratke, kateri kvadratki pripeljejo do njih.

Ker vidimo, da je prvi vagon (skrajno levi) na 1. in 2. tiru napačen bomo morali premakniti vse vagone na sredinski tir. Torej rabimo 16 premikov oziroma 16 kovancev.

Pri naslednji različici naloge: *Zmeda na postaji (2),* ko imamo omejitev samo petih vagonov na srednjem tiru moramo med premikanjem pospravljati vagone na druga tira. Naj bo O tir z oranžno lokomotivo, Z tir z zeleno lokomotivo in D desni tir. Vsak premik označimo kot <vir><ponor><število vagonov>. Rešitev: ZD1, OD4, DZ1, DO2, ZD2, DO1, ZD2, DO1, DZ4. To nam prinese v najboljšem primeru 18 premikov.

Nalogo bi z računalnikom rešili tako, da bi zapisali vse možne vmesne rešitve v graf stanj in potem poiskali najkrajšo pot v njem. Postopek reševanja je predstavljen v nadaljevanju na primeru Brodnikovega problema.

Podobne naloge: *Preskakovanje, Labirint*
Naloga iz tekmovanja 2014/15: *Srečevanje*

Računalniško ozadje - Grafi in iskanje v globino in širino

Vsako nalogo moramo prepisati v stanja. Običajno imamo vedno začetno stanje iz katerega hočemo doseči neko končno stanje. Pri vsaki nalogi imamo podana navodila, kako preidemo iz enega stanja v drugo. Če povežemo stanja, ki si sledijo ena v drugo dobimo povezan graf.

Iskanje rešitve običajno poteka tako, da začnemo v začetnem stanju in potem obiskujemo vmesna stanja, dokler ne pridemo do rešitve. Če pridemo do končnega stanja od koder ne moremo več naprej se vrnemo nivo višje in preiskujemo naslednjo vejo, dokler ne pridemo do rešitve. Celotna pot od začetka do rešitve je končna rešitev, ki nam da postopek.

Rešitev lahko iščemo na več načinov. Tako poznamo iskanje v globino, kjer v vsakem koraku pogledamo eno stanje naprej – se spuščamo vedno en nivo globlje.

Pri preiskovanju v širino pa najprej pogledamo vsa možna stanja iz začetnega stanja nato pa gremo in pogledamo vsa stanja, ki so en nivo nižje in tako naprej.

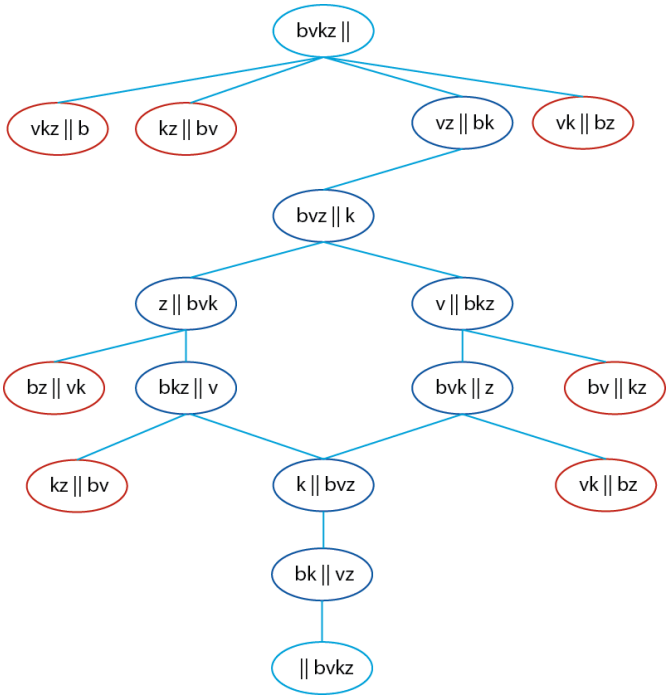
Brodnikov problem

Poglejmo si opisana iskanja na dobro poznani uganki

Brodnik mora na nasprotni breg reke spraviti kozo, volka in zelje. Pri tem lahko v čolnu pelje le eno od obeh živali ali le zelje. Poleg tega ne sme na istem bregu pustiti koze same z volkom ali koze same z zeljem.

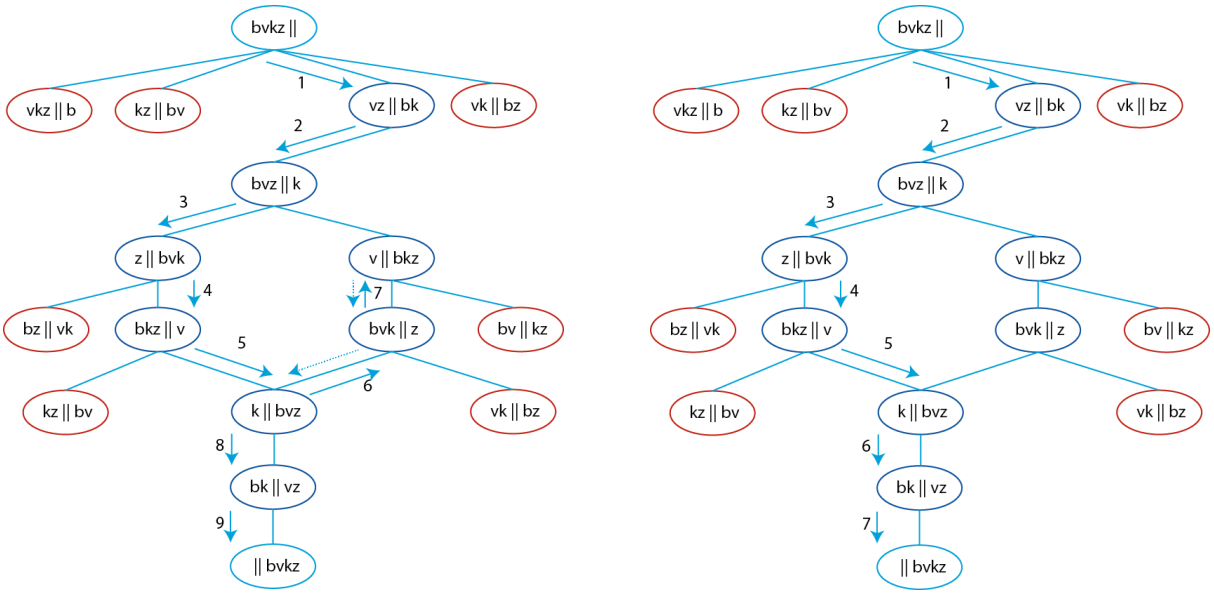
Najprej moramo prepisati uganko v povezan graf možnih stanj (slika 1). Dve stanji bomo povezali, če lahko z enim prevozom preidemo iz tega stanja v drugo. Pri reševanju problema (brodnika, volka, koze in zelja = bvkz) bomo začeli z začetnim stanjem in poskušali doseči ciljno stanje (vsi na drugi stani reke) tako, da se bomo sprehodili po vmesnih stanjih. Stanja lahko zapišemo tako, da

udeležence zapišemo v seznam prvih črk (bvkz ||), reko med njimi pa predstavimo z dvema navpičnicama. Cilj je vse pripeljati na drugo stran (|| bvkz).



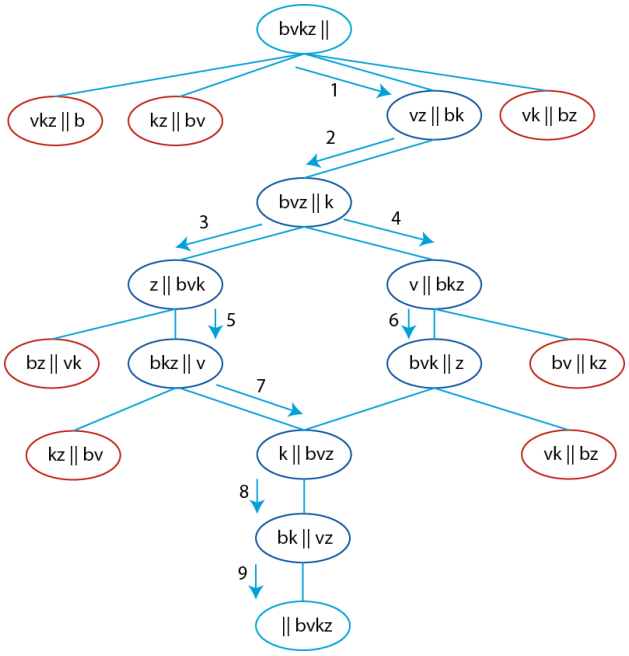
Slika 1: Graf možnih stanj za Brodnikov problem

Pri iskanju v globino začnemo z začetnim stanjem in nato v naslednjem dovoljenem (vz || bk). Pri preiskovanju si zapomnimo katera stanja smo že obiskali. Če pridemo, do konca in to še ni rešitev se vrnemo nazaj. Tako nadaljujejo z naslednjim stanjem, ko se brodnik vrne (bvz || k). Nadaljujemo v desni veji (z|| bvk) in tako naprej. Ko pridemo do stanja, ko imamo samo še kozo na levi strani (k||bvz) imamo dve možnosti navzdol ali pa levo. Za preiskovanje je to vse naslednik trenutnega stanja, ki ga še nismo obiskali. V primeru, če obiščemo stanje (bvk || z) se bomo pri stanju (v || bkz) morali vrniti, ker smo naslednje stanje že obiskali. Obe nakazani rešitvi prikazujeta sliki 2 in 3.



Sliki 2 in 3: Postopka preiskovanja v globino

Pri preiskovanju v širino pa najprej preiščemo vse naslednike. V našem primeru gremo iz stanja (bvz || k) najprej pogledati desno in nato še levo in tako naprej. Postopek iskanja prikazuje slika 4.



Slika 4: Preiskovanje v širino.