

**ACM TEKMOVANJE V  
PROGRAMIRANJU Z DELČKI**

**PIŠEK 2021/22**

**1. šolsko tekmovanje  
(delovna verzija)**

**Naloge in rešitve**

**Februar 2022**

Izbor, priredba in preoblikovanje tekmovalnih nalog: Programski svet tekmovanja

Razvoj tekmovalnega sistema: ACM Slovenija in UL FMF v sodelovanju s France-IOI

# KAZALO NALOG

Tekmovalna kategorija: OŠ 4. – 6. r. ZAČETNIKI

GUSARJI IŠČEJO ZAKLAD .....	1
NEŽA BARVA.....	3
ČEBELICA ROZI OPRAŠUJE .....	5
PIŠEK RIŠE .....	7
ZAJČEK POBIRA KORENČKE.....	9
VRT SONČNIC .....	10

Tekmovalna kategorija: OŠ 4. – 6. r. NAPREDNI

PEŠČENI PLANET .....	12
ŠTIRJE ZAKLADI.....	13
AVTO NA DALJINCA .....	15
JEŽEK SE PRIPRAVLJA NA ZIMO.....	16
REGRATOVA LUČKA .....	18
SNEŽAKI IN KORENČKI .....	20

Tekmovalna kategorija: OŠ 7. – 9. r. ZAČETNIKI

SEJALNI ROBOT .....	22
PIŠEK GLEDA TEKMO .....	24
JURE V NEW YORKU .....	26
MIŠKA ZBIRA SIR.....	28
POVPREČNA DOLŽINA POTI V ŠOLO .....	30
PIŠEK VADI POŠTEVANKO .....	32

Tekmovalna kategorija: OŠ 7. – 9. r. NAPREDNI

ŠIFRIRANJE SPOROČIL.....	34
VEVERIČKA IN OZIMNICA .....	36
BARVANJE OGRAJE.....	38
ZMAJČEK IN TLAKOVANJE .....	39
MIŠKA IŠČE SIR .....	41
REGRATOVA LUČKA.....	43

Tekmovalna kategorija: SŠ ZAČETNIKI

ROBOTEK BOJAN RIŠE .....	45
ISKANJE POGREŠANEGA .....	47
BARVANJE OGRAJE .....	49
SLAŠČIČAR STANE .....	50
CEZARJEVA ŠIFRA .....	52

Tekmovalna kategorija: SŠ NAPREDNI

KRTKOVA DRUŽINA .....	54
EKSPERIMENT S SODIMI ŠTEVILI .....	56
NAJTEŽJI ZABOJ .....	58
ODNAŠANJE SMETI .....	60
DRUŽABNA IGRA .....	62
TRIGLAV .....	64
KOCHOVA ČRTA .....	66
Rešitve .....	68

## GUSARJI IŠČEJO ZAKLAD

OŠ 4. – 6. r. ZAČETNIKI

Gusarji so se znašli sredi morja, kjer je veliko otokov. A le na enem je zaklad. Sestavi program, ki jih bo pripeljal do pravega otoka, da bodo odšli domov z zakladom.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej sliko in poišči otok z zakladom.
- Hitro ugotoviš, da lahko ladjo do tja pripelješ na različne načine. Giblješ se od kvadratka do kvadratka. Ker pri tej nalogi ni nobene omejitve glede števila delčkov, ki jih lahko uporabiš, lahko program napišeš popolnoma poljubno. Svetujemo ti, da poiščeš čim krajšo pot, da bo tudi program krajši. Dve izmed mnogih možnosti gibanja sta označeni na spodnji sliki. Ko se odločiš, katero pot boš ubral, delčke nanizaj v pravo zaporedje.
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Pravilne so tudi druge rešitve, saj je pravih poti zelo veliko.

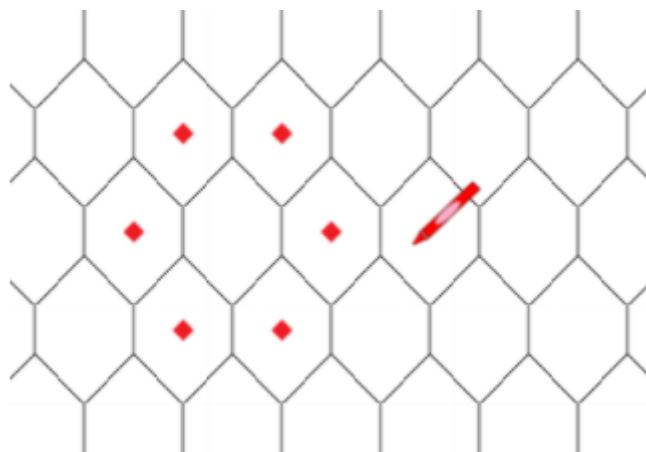


[Povezava do rešitev](#)

## NEŽA BARVA

OŠ 4. – 6. r. ZAČETNIKI

Neža ima zelo rada pobarvanke, še posebno tiste, kjer so priložena navodila, ki točno povedo, kako naj barva. Žal pa so tokratna navodila pomešana. Pomagaj ji in zloži delčke tako, da bo pobarvala polja z rdečimi oznakami.

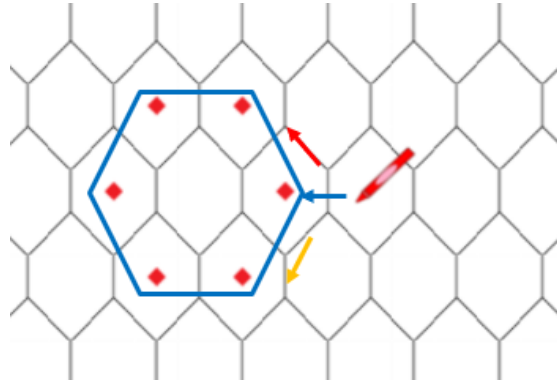


### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej sliko in razmisli, kaj naloga zahteva od tebe. Ugotoviš lahko, da moraš voščenko najprej premakniti za eno polje v levo, nato pa začneš s premikanjem po stranicah 6-kotnika, pri čemer vsako tudi pobarvaš rdeče. Premikaš se lahko v smeri urnega kazalca, ali pa ravno obratno.
- Na delovni površini so razmetani vsi delčki, ki jih potrebuješ. Čeprav bi se po roži lahko premikal v kateri koli smeri, ti delčki na delovni površini nakazujejo rešitev v nasprotni smeri urnega kazalca (rdeča puščica na spodnji sliki). Zdaj, ko veš, katero pot boš ubral, le nanizaj delčke v pravo zaporedje.
- Nalogo lahko rešiš tudi tako, da barvaš v smeri urnega kazalca (oranžna puščica na spodnji sliki), a v tem primeru je treba spremeniti delčke premikanja.
- Delo si lahko tudi precej zapleteš, če iz knjižnice uporabiš dodatne delčke. Tako lahko začneš barvati na poljubnem polju., Vendar bo program v tem

primeru precej daljši, za to pa boš porabil tudi precej časa. Naloga ima torej veliko pravih rešitev.



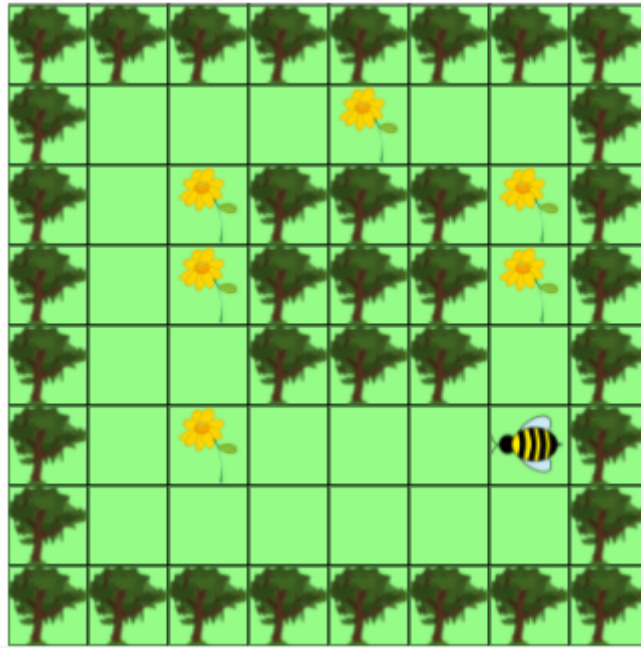
[Povezava do rešitev](#)



## ČEBELICA ROZI OPRAŠUJE

OŠ 4. – 6. r. ZAČETNIKI

Čebelica Rozi mora oprašiti vse rože na travniku. Priskoči ji na pomoč. Čebelica opraši rožo tako, da poleti na njeno polje.



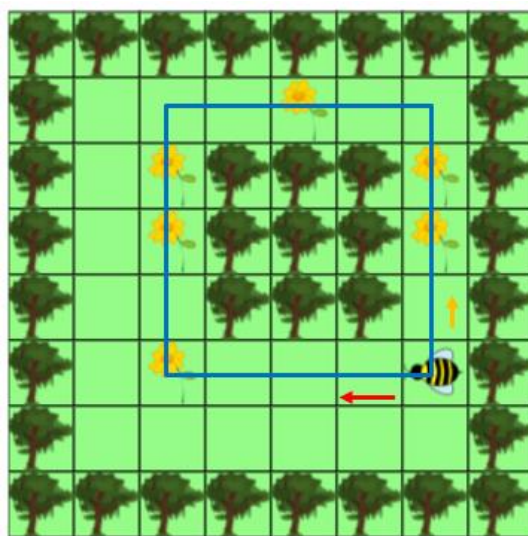
### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej sliko in razmisli, kaj naloga pričakuje od tebe. Ugotoviš lahko, da so rože razporejene v obliki kvadrata. Tvoja naloga je, da vodiš čebelico Rozi po kvadratu tako, da bo prišla do vsake rože. Premikaš se lahko v poljubni smeri – torej v smeri urnega kazalca (tako kot je Rozi postavljena na začetku – lažja rešitev), lahko pa tudi v obratni smeri (torej najprej premakneš Rozi navzgor – težja rešitev). Obe možnosti sta pravilni.
- Ker ima naloga omejitve delčkov, lahko hitro ugotovimo, da nam naloge ne bo uspelo rešiti le z nizananjem delčkov v pravilno zaporedje. Uporabiti moramo torej zanko. Možni sta dve različni rešitvi:
- Namesto, da delček **poleti naprej** nanizamo štirikrat, torej na vsaki stranici kvadrata, lahko uporabimo zanko, ki se ponovi 4-krat. Na koncu vsake stranice pa uporabimo delček **obrne se desno**. Žal naloge na tak

način ne moremo rešiti, če se želimo premikati v obratni smeri, saj imamo na voljo 1 delček premalo.

- Ko imamo pred sabo to kodo, lahko ugotovimo, da se tudi tukaj 4-krat ponovi enak del kode, torej lahko uporabimo kar gnezdeno zanko – »zanko v zanki«. V tem primeru lahko uporabimo tudi premikanje v obratni smeri.
- Najprej poskusi nalogo rešiti samostojno, rešitve z nekaj več razlage pa lahko najdeš na spodnji povezavi.



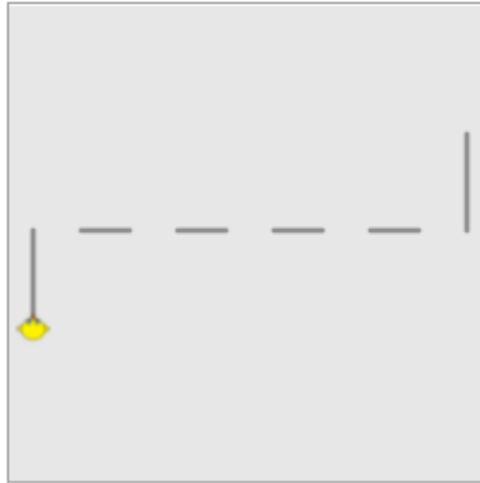
[Povezava do rešitev](#)

## PIŠEK RIŠE

OŠ 4. – 6. r. ZAČETNIKI

Pišek želi postati slikar. Pomagaj mu narisati preprost vzorec iz črt dolgih 10 oz. 20 enot.

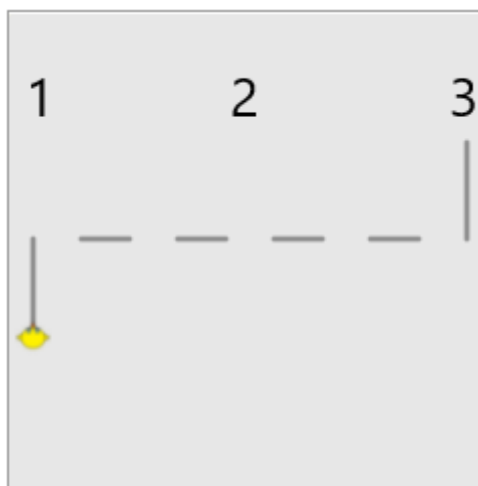
Na voljo imaš omejeno število delčkov, zato spretno uporabi zanko.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej sliko in razmisli, kaj naloga pričakuje od tebe. Nalogo lahko razdelimo na tri dele: 1) risanje leve navpične črte; 2) risanje vodoravnih prekinjenih črt; 3) risanje desne navpične črte.



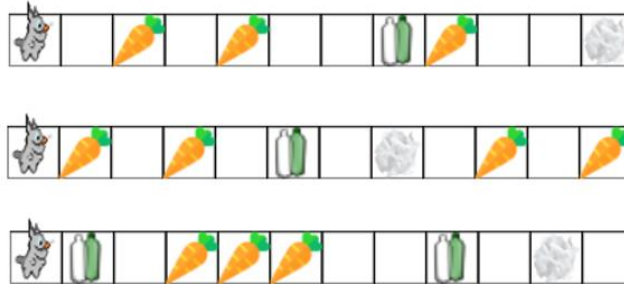
- Obe navpični črti lahko narišemo zelo preprosto – le v delček **pojdi naprej za** moramo vnesti pravo vrednost. Ta je že podana v navodilih, zato jih še enkrat dobro preberi.
- Večji zalogaj bo risanje vmesnih vodoravnih prekinjenih črt. Tukaj moramo zelo paziti, kdaj je pisalo dvignjeno in kdaj spuščeno. Prav tako nam omejitev delčkov nakazuje na to, da naloge ne bomo mogli rešiti zgolj z nizanjem delčkov, ampak moramo uporabiti zanko. Po premisleku lahko vidimo, da vzorec sestavljajo 4 vodoravne črte z vmesno prekinitvijo. Vse 4 črte so si popolnoma enake, torej lahko brez težav uporabimo zanko.
- Zadnji premislek, ki ga moramo narediti, je, ali se bo naša zanka začela s črto (v tem primeru pred zanko naredimo premik z dvignjenim svinčnikom), ali pa s premikom brez risanja (v tem primeru pa dodamo še en premik brez risanja za zanko). Obe rešitvi sta seveda pravilni.
- Najprej poskusi nalogo rešiti samostojno, rešitve z nekaj več razlage pa lahko najdeš na spodnji povezavi.

[Povezava do rešitev](#)

## ZAJČEK POBIRA KORENČKE

OŠ 4. – 6. r. ZAČETNIKI

Zajček lahko na svojem sprehodu naleti na različne predmete. Ker je zajec, naj na svoji poti pobira samo korenčke. Bodi pozoren, naloga ima več testov.



### Povezava do naloge

### Ideja reševanja

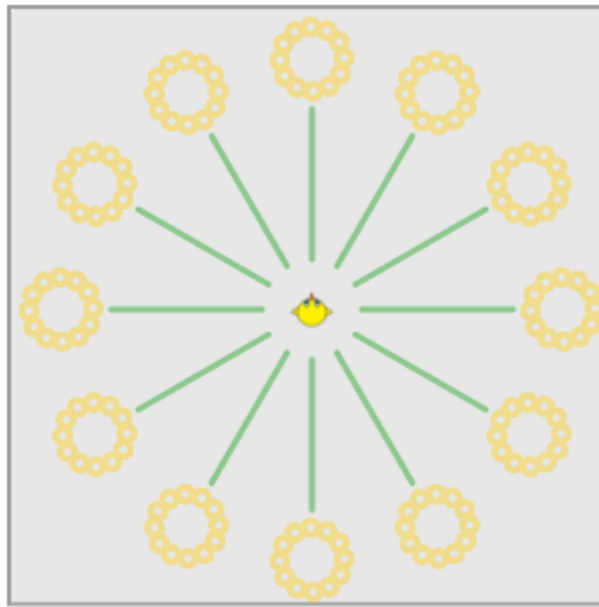
- Najprej si dobro oglej vse tri teste primere. Ugotoviš lahko, da so si precej podobni. Vedno je zajček na prvem izmed 12 polj, na katerih pa so lahko različni predmeti. Pravilna rešitev bo le tista, ki bo rešila vse tri primere. Nalogo lahko razdelimo na dva dela: 1) premikanje po vseh poljih; 2) preverjanje, ali je na polju korenček.
- Za premikanje po vseh poljih bi lahko nanizali 11 delčkov **premakni se naprej**. Ker pa je to zelo zamudno, lahko uporabimo zanko, ki omogoči, da se premikanje ponovi 11-krat.
- Za preverjanje pogoja, ali je na polju korenček, moramo uporabiti pogojni stavek. Ker moramo to preveriti čisto na vsakem polju, tudi pogojni stavek zapišemo v zanki.
- Najprej poskusi nalogo rešiti samostojno, rešitve z nekaj več razlage pa lahko najdeš na spodnji povezavi.

### Povezava do rešitev

## VRT SONČNIC

OŠ 4. – 6. r. ZAČETNIKI

Pišek obožuje sončnice, zato se je odločil, da si bo zasadil vrt sončnic v obliki ure. Stoji na sredini vrta. Popravi in dopolni program tako, da bo spretno zasadil dvanajst sončnic, ki bodo vse od središča vrta oddaljene enako (100 korakov). Stebla so dolga 60 korakov. Upoštevaj, da če Pišek naredi naprej -10 korakov, da to pomeni, da je šel nazaj 10 korakov.



### Povezava do naloge

### Ideja reševanja

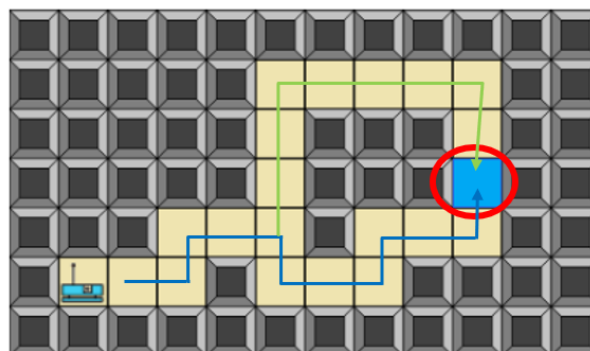
- Najprej si dobro oglej program, ki je že pripravljen. Če ga poženemo, vidimo, da dela nekaj napak: 1) ne nariše stebel; 2) ne vrača se v izhodišče; 3) riše le 10 cvetov namesto 12-ih. Popravljanja se bomo torej lotili počasi po korakih.
- Najprej rešimo napako v risanju stebel. V zanki vidimo, da je že narejen premik z dvignjenim pisalom, ki ga Pišek naredi pred in za stebлом. Vmes bomo torej narisali steblo tako, da bomo spustili pisalo in se premaknili za 60 korakov naprej. Po steblu pa moramo znova dvigniti pisalo in se premakniti za nekaj korakov naprej. Če smo zelo pozorni, lahko vidimo, da začetni in končni premik z dvignjenim pisalom nista 15 korakov, temveč 20 (navodila nam narekujejo izračun:  $100 - 60$ ). Na koncu narišemo sončnico.

- Ko Pišek nariše eno sončnico, se mora vrniti nazaj v izhodišče. Uporabimo lahko kar trik, ki nam ga povedo navodila: Piška premaknemo za -100 korakov. Lahko pa bi uporabili tudi dva delčka, torej se zavrteli za 180 stopinj in šli nato 100 korakov nazaj. Če uberemo to rešitev, moramo paziti, ker se Pišek ne bo po vrnitvi obrnil za 30 stopinj v levo, kot nam pravi že vnaprej podana rešitev.
- Zadnjo napako lahko popravimo zelo enostavno, le spremenimo število ponovitev v zanki iz 10 na 12.
- Najprej poskusi nalogo rešiti samostojno, eno izmed več pravih rešitev pa lahko najdeš na spodnji povezavi.

[Povezava do rešitev](#)

## OŠ 4. – 6. r. NAPREDNI

- Najprej si dobro ogledaj sliko in poišči polje z vodo.
- Hitro ugotoviš, da lahko robota do tja pripelješ na dva različna načina. Ker pri tej nalogi ni nobene omejitve delčkov, lahko program napišeš povsem poljubno. Svetujemo ti, da poiščeš čim krajšo pot, da bo tudi program najkrajši. Na spodnji sliki je krajša pot označena z modro barvo. Ko se odločiš, katero pot boš ubral, delčke nanizaj v pravo zaporedje.
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Pravilne so tudi druge rešitve.



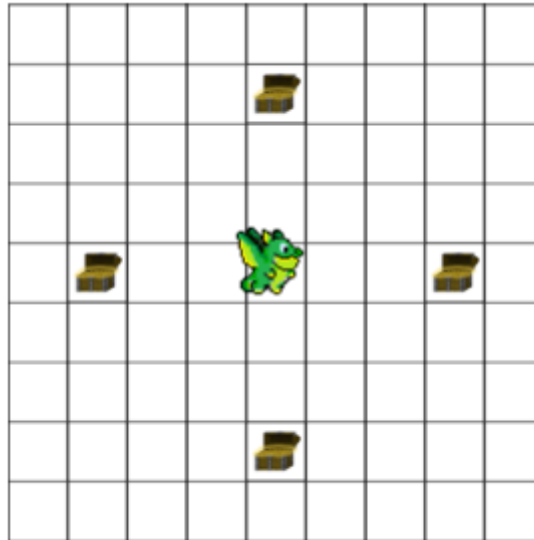
## 12



## ŠTIRJE ZAKLADI

OŠ 4. – 6. r. NAPREDNI

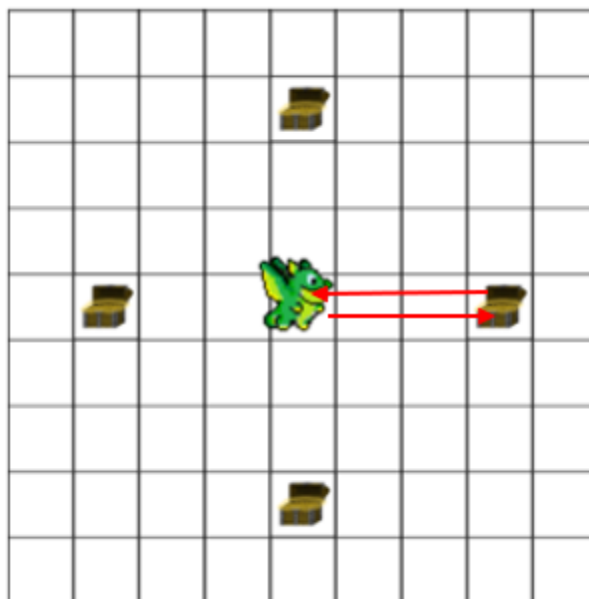
Prijatelj Pišek je za zmajčka pripravil igro Skriti zaklad. Na sliki je zemljevid, kjer se vidi, da so skriti štirje zakladi. Pomagaj zmajčku in mu napiši navodila za premikanje tako, da bo prišel na vse 4 kvadratke z zakladom. A pazi! Ker lahko uporabiš le 12 delčkov, boš moral uporabiti zanke.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej sliko in si zamisli pot, ki jo bo zmajček prehodil. Možnosti je zelo veliko, a ker je število delčkov omejeno, moraš poiskati takšno, ki se bo čim bolj ponavljala, da boš lahko uporabil gnezdeno zanko.
- Ena izmed možnosti je prikazana na sliki spodaj. Z uporabo zanke se premaknemo 3 polja naprej, nato se obrnemo in se znova z uporabo zanke premaknemo 3 polja nazaj do sredine. Tu pa se lahko obrnemo levo ali desno, in enak postopek ponovimo še v preostalih 3 smereh. Ker se ponavlja ista koda, lahko uporabimo gnezdeno zanko.
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Možne so seveda tudi druge rešitve.



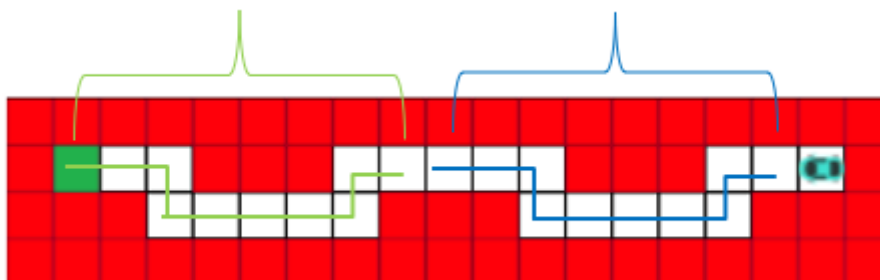
[Povezava do rešitev](#)

## OŠ 4. – 6. r. NAPREDNI

A 10x10 grid world environment. The start cell is green (top-left). The goal cell is blue (top-right). Obstacles are black cells forming a vertical bar on the left and a horizontal bar at the bottom. The path from start to goal is marked with yellow cells.

## Ideja reševanja

- Ker si pri tej nalogi zelo omejen s številom delčkov, ki jih imaš na voljo, moraš obvezno uporabiti gnezdeno zanko. Dobro si oglej sliko in poskusi najti del poti, ki se ponovi.
- Ko najdeš pot, ki se bo podvojila, opišeš prvi del poti (pri premiku za 4 polja uporabi zanko), nato pa programsko kodo le ponoviš tako, da jo vstaviš v gnezdeno zanko.
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Zaradi ostre omejitve delčkov je to edina možna rešitev.

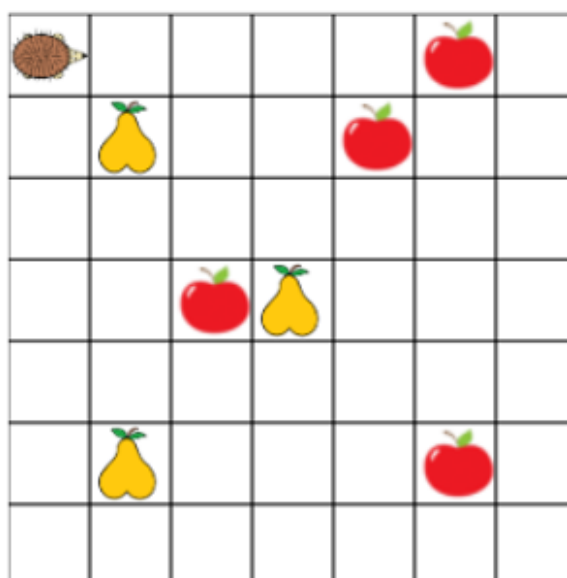
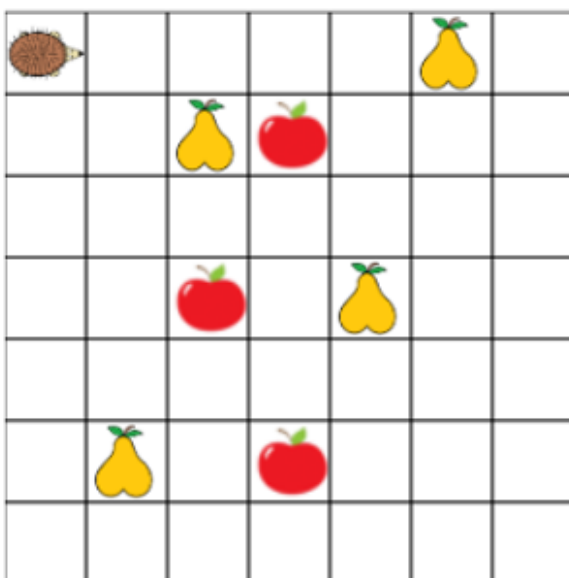


## 15

## JEŽEK SE PRIPRAVLJA NA ZIMO

OŠ 4. – 6. r. NAPREDNI

Ježek ve, da bo zima huda. Zato se odpravi na travnik, kjer ležijo jabolka in hruške. Kako naj se premika, da bo pobral vse sadeže? Sadeži so v TESTU1 in TESTU2 razporejeni na različne načine.



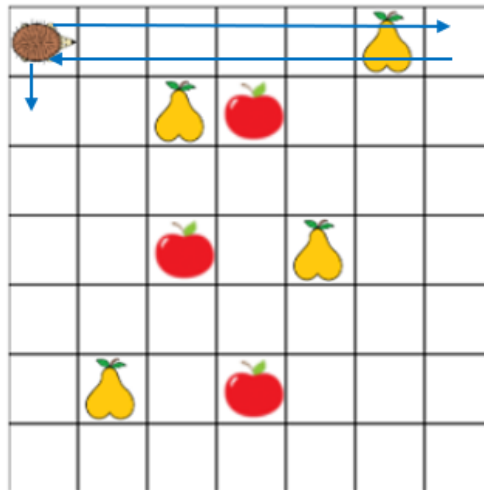
### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglej oba testna primera. Opaziš lahko, da so sadeži v obeh testih razporejeni drugače, skupno pa jima je to, da nikoli niso v prvem oz. zadnjem stolpcu in v zadnji vrstici, kar nam nekoliko olajša reševanje naloge. Program lahko vsebinsko razdelimo na tri dele: 1) sprehod ježka skozi celotno mrežo; 2) iskanje in pobiranje sadežev.
- Najprej se lotimo pisanja kode, ki bo ježka premikala po mreži. Za premike v desno lahko uporabimo preprosto zanko. Na koncu vrstice se mora ježek obrniti in se vrniti na začetek. V zadnji fazi pa se mora premakniti za vrstico navzdol, za kar moramo uporabiti obračanje v levo. Pomembno je, da ježka obrnemo tako, da gleda v desno smer, ker le tako lahko preprosto uporabimo gnezdeno zanko, in sicer enak postopek ponovimo za 6 vrstic.
- Ker imamo 2 testna primera, moramo napisati univerzalno rešitev. V ta namen uporabimo pogojne stavke, in sicer, če je ježek na sadežu, ga

pobere. Ježek lahko sadeže pobira, ko se premika v desno, ali pa ko se vrača na začetek vrstice.

- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Možne so tudi druge rešitve.

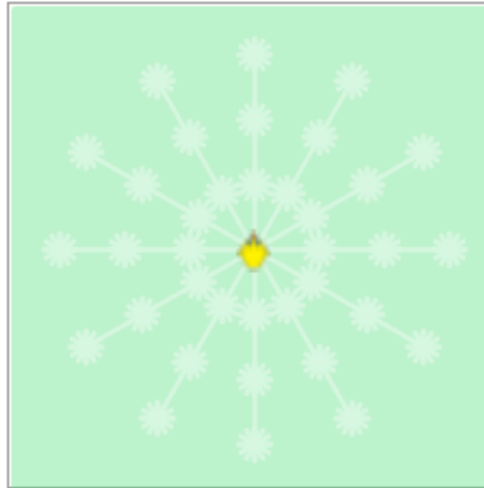


[Povezava do rešitev](#)

## REGRATOVA LUČKA

OŠ 4. – 6. r. NAPREDNI

Pišek ima zelo rad regratove lučke. Pomagaj mu jo narisati. Raca mu je že posodila štopiljko, ki nariše eno padalo regratove lučke. Pišek je začel risanje, vendar njegov program ni pravilen. Pomagaj mu in popravi program tako, da bo Pišek narisal takšno sliko, kot jo vidiš spodaj.



### Povezava do naloge

### Ideja reševanja

- Najprej si oglejmo program in ga preizkusimo. Vidimo, da ima nekaj napak, nekaj korakov pa mu še manjka: 1) manjkajo povezave med padali posameznega kraka regratove lučke; 2) program nariše le enega od 12 krakov.
- Najprej popravimo program tako, da bo pravilno narisal en krak. Premikanje začnemo s spuščnimi pisalom in ga ne dvigamo. Razdalja 40 korakov med padali je že pravilno nastavljena. Nato se mora Pišek obrniti za 180 stopinj in se z dvignjenim svinčnikom vrniti v središče lučke (premakne se za 120 korakov). Najlažje je, da Piška obrnemo za 180 stopinj tako, da gleda v smer že narisane kraka (ni pa nujno).
- Uporabimo gnezdeno zanko in ponovimo risanje krakov 12-krat. Po vsakem kraku se moramo zavrteti za  $(360:12 = 30)$  30 stopinj v levo ali pa desno smer.

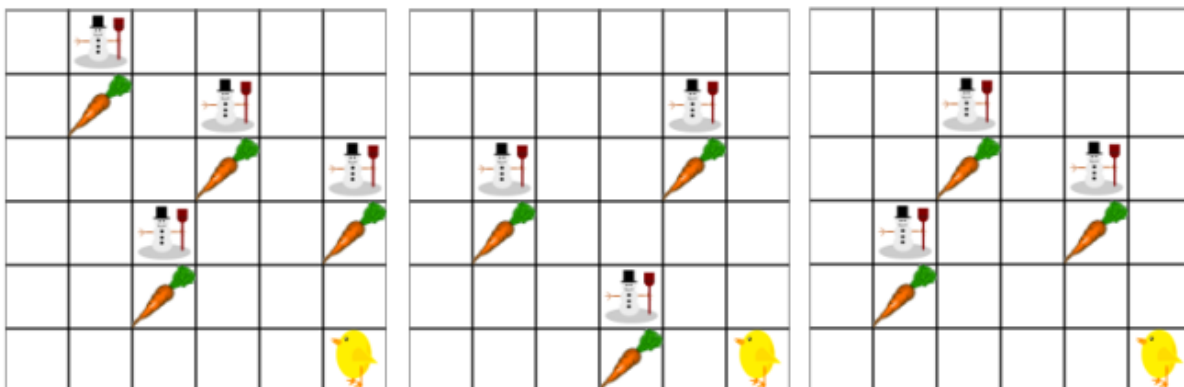
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi.  
Možne so tudi druge rešitve.

[Povezava do rešitev](#)

## SNEŽAKI IN KORENČKI

OŠ 4. – 6. r. NAPREDNI

Pišek je opazil, da so snežaki izgubili svoje nosove. Pomagaj mu odnesti korenčke do vsakega snežaka, tako, da bo vsak snežak imel svoj nos. Korenčki ležijo vedno pred snežaki. Pišek lahko nosi le en korenček naenkrat.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglejmo vse tri testne primere. Opazimo lahko, da se korenčki nikoli ne nahajajo v zgornji vrstici in levem stolpcu, kar nam bo nekoliko olajšalo pisanje programa. Problem lahko razbijemo na dva manjša: 1) premikanje Piška po celi mreži; 2) iskanje in premikanje korenčkov.
- Najprej napišimo program, ki bo Piška premikal po celi mreži. Za premike po vrsticah uporabimo zanko. Premaknemo se torej 5-krat levo; 5-krat desno in enkrat gor; nato pa to ponovimo 5-krat z uporabo gnezdene zanke.
- Odločimo se, v kateri smeri bomo iskali korenčke, npr. ko se premikamo v levo (v obeh smereh ni treba). Ker imamo več testnih primerov, moramo napisati univerzalno rešitev, torej bomo uporabili pogojni stavek. A biti moramo zelo pozorni. Ni dovolj, da le preverjamo, ali je na polju korenček – preveriti moramo, da ni morda na istem polju tudi snežak, kajti v tem primeru bomo korenček snežaku vzeli. Uporabiti moramo torej pogojni stavek z dvema pogojema (v navadni pogojni stavek dodamo delček **in**). In ker ne smemo biti na polju s snežakom, pred senzor **na snežaku** dodamo delček **ne**. Ko imamo pogoj napisan, napišemo še, izvedbo: torej



Pišek vzame korenček, se premakne polje navzgor, postavi korenček na snežaka in se vrne nazaj. S slike lahko vidimo, da je snežak vedno natanko eno polje nad korenčkom, zato tega ni treba preverjati.

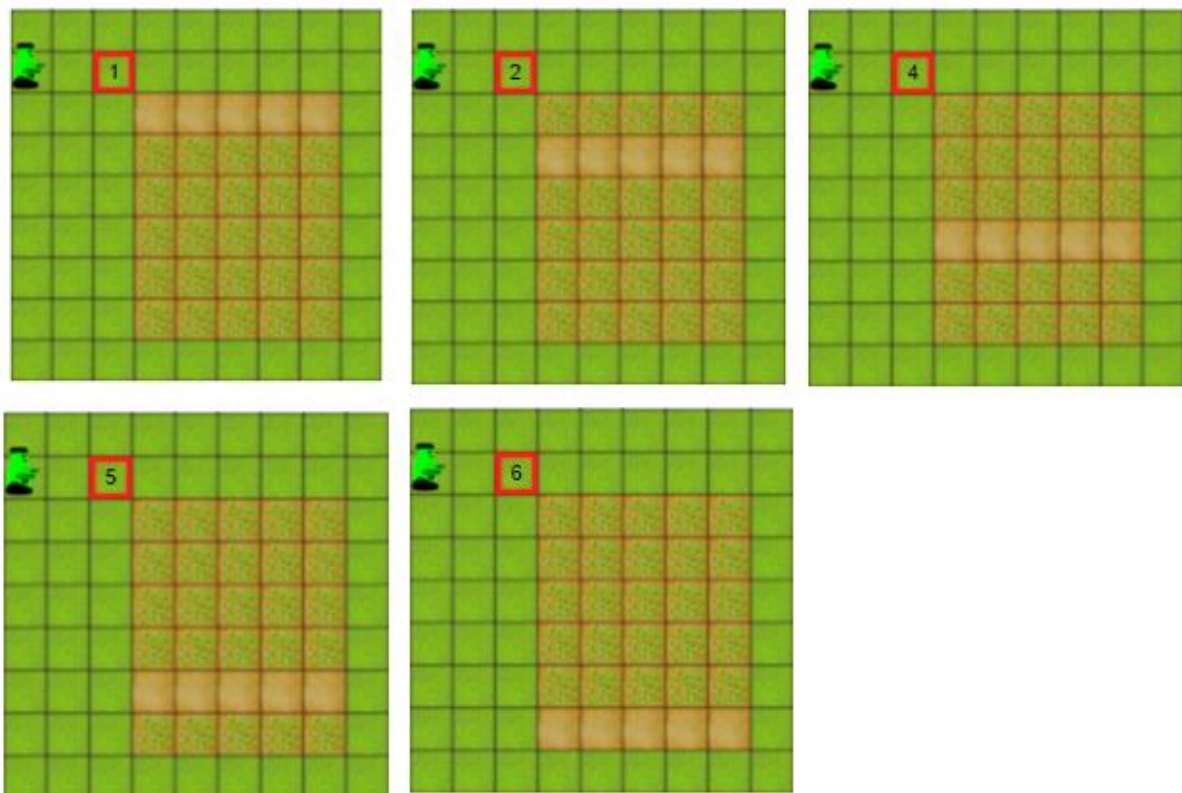
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Možne so tudi nekoliko drugačne rešitve.

**[Povezava do rešitev](#)**

## SEJALNI ROBOT

OŠ 7. – 9. r. ZAČETNIKI

Kmet Marko ima novega sejalnega robota. Zanj je poskusil napisati programsko kodo, ki bi ga usmerjala tako, da bi posejal nova semena v točno določeni vrstici polja. Popravi Markov program tako, da bo sejalni robot prebral število v rdečem polju in nato posadil semena v ustrezni vrstici polja. Dobro si oglej vse testne primere. Del programske kode je že napisan, popravi in dopolni jo.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglejmo vse testne primere. Opazimo lahko, da je rdeče polje vedno na istem mestu. Prav tako je njiva vedno enake oblike, le robot mora posaditi semena v drugi vrstici. Problem si razbijemo na tri manjše: 1) premik do rdečega polja; 2) iskanje prave vrstice; 3) sejanje semen.
- Najprej popravimo program, da se bo robotek premaknil do rdečega polja.
- Zanka, ki je že napisana v programu: **ponavljaj številka na polju krat**, bo robotka premaknila na začetek prave vrstice, zato vanjo

napišemo le delček **premakni se naprej**. Pred njo pa moramo postaviti delček **obrne se desno**, kajti tako se bo robotek res premikal po stolpcu navzdol toliko polj, kot je zapisano na polju s številko.

- Zdaj smo robotka pripeljali tik pred začetek prave vrstice in ga moramo le še usmeriti, da bo pravilno posadil vsa semena. Robotek se mora obrniti v levo tako, da gleda v smer njive. Nato pa z uporabo preproste zanke v vsako od naslednjih petih polj posadimo seme: robotek se premakne naprej in posadi seme.
- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Možne so tudi nekoliko drugačne rešitve.

[Povezava do rešitev](#)

## PIŠEK GLEDA TEKMO

OŠ 7. – 9. r. ZAČETNIKI

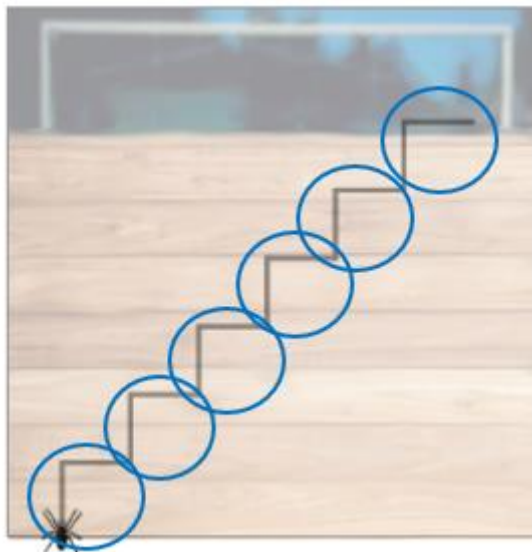
Pišek bi si rad ogledal nogometno tekmo. A kaj, ko nima denarja za vstopnico. No, na srečo je našel mesto ob ograji, kjer bi si lahko brezplačno ogledal tekmo, a je premajhen. Prijatelj pajek mu bo priskočil na pomoč tako, da bo spletel trdne vrvne stopnice z dolžino stranice 100 korakov. Da bo pajek vedel, kaj in kako, je Pišek začel pisati program, ki bo stopnice narisal na ograjo. Začel je ustvarjati program, a je naletel na težave. Pomagaj mu in popravi program tako, da bo narisal stopnice, kot jih vidiš na sliki.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglejmo že vnaprej pripravljeno programsko kodo in jo preizkusimo. Opazimo lahko nekaj pomanjkljivosti: 1) dolžina stranice je prekratka; 2) pajek nariše kvadrat in ne stopnico; 3) število ponovitev v zanki je premajhno.
- Najprej popravimo dolžino koraka na 100, kot je zapisano v navodilih.
- Da pajek nariše stopnico, se mora po prvi stranici obrniti v desno, po drugi pa v levo.
- Število ponovitev moramo povečati na 5.
- Nalogo poskusi rešiti samostojno, v pomoč ti je lahko spodnja slika. Eno izmed pravih rešitev pa najdeš na spodnji povezavi.

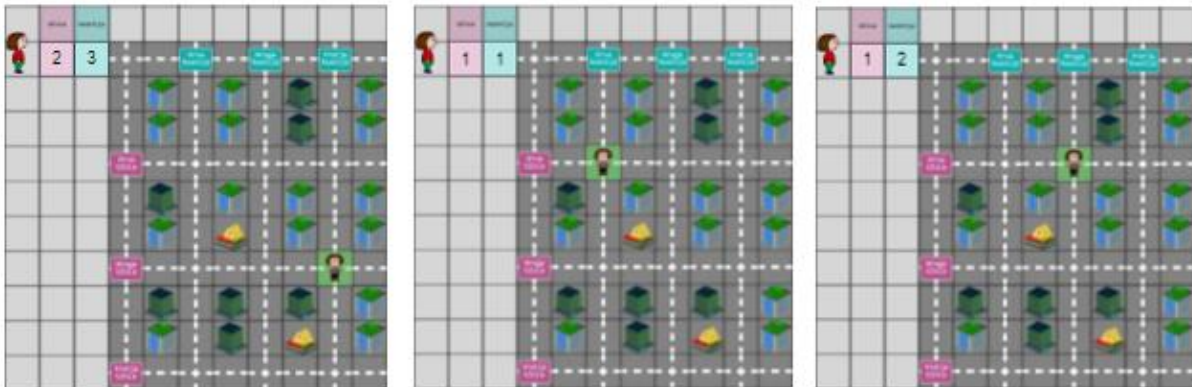


[Povezava do rešitev](#)

## JURE V NEW YORKU

OŠ 7. – 9. r. ZAČETNIKI

Jure je odšel na prvo potovanje v tujino, in sicer je v New Yorku obiskal svojo prijateljico Elo. Ker bo prvič v tako velikem mestu, ga skrbi, da se bo izgubil. Ela mu bo v sporočilu zapisala v križišču katere ulice in avenije ga bo počakala. Ker Jure na letalu nima signala, bi si rad že vnaprej pripravil navodila, kako naj najde Elo, ne glede na to, katero kombinacijo ulice in avenije mu bo sporočila. V New Yorku avenije tečejo v smeri sever-jug, ulice pa v smeri zahod-vzhod. Dobro si oglej vse testne primere, nato pa popravi in dopolni program tako, da bo pripeljal Jureta v pravo križišče.



### Povezava do naloge

### Ideja reševanja

- Najprej dobro preberimo navodila in si oglejmo vse tri testne primere. Ugotovimo lahko, da so vsi trije primeri enaki, le Ela se nahaja v različnem križišču. Napisati bomo morali torej univerzalno kodo. Da si bo program lahko zapomnil vrednosti v roza in modrem polju, bomo morali uporabiti spremenljivko. Razčlenimo nalogo na več korakov: 1) premaknimo se do polja ulica in shranimo vrednost v spremenljivko **ulica**; 2) premaknimo se do polja avenija in shranimo vrednost v spremenljivko **avenija**; 3) premaknimo se toliko polj proti jugu, da dosežemo pravo ulico; 4) premaknimo se toliko polj proti vzhodu, da dosežemo pravo avenijo.
- V prvi fazi se premaknemo eno polje proti vzhodu in nastavimo vrednost spremenljivke »ulica« na vrednost, ki je zapisana na polju.

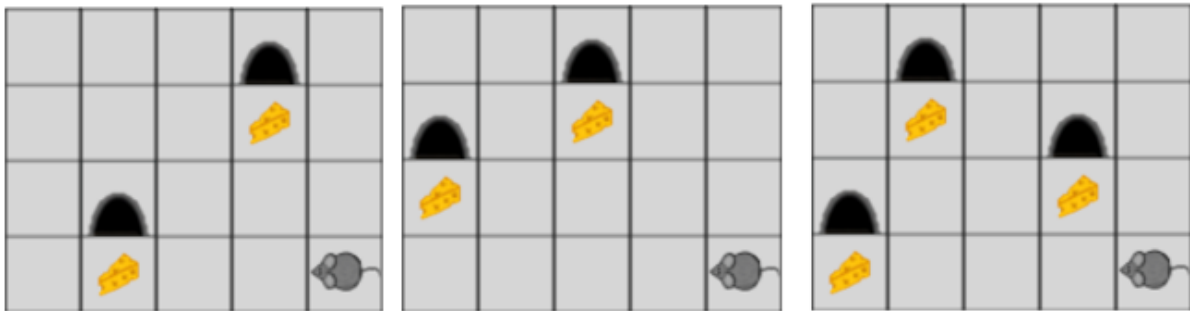
- Premaknimo se še eno polje proti vzhodu in enak postopek ponovimo še za avenijo. Premaknimo se še eno polje proti vzhodu, da bo pisanje programa lažje (torej stojimo v križišču »ničte« ulice in »ničte« avenije).
- Dobro si oglejmo sliko. Vidimo, da so sosednje ulice oddaljene za tri polja. Da bomo dosegli naslednjo ulico, se bomo torej premaknili proti jugu za tri polja. To bomo ponovili tolikokrat, kot je vrednosti, ki smo jo shranili v spremenljivko `ulica`.
- Podobno storimo tudi za avenijo. Vidimo, da so sosednje avenije oddaljene za dve polji. Da bomo dosegli naslednjo avenijo, se bomo torej premaknili proti vzhodu za dve polji. To bomo ponovili tolikokrat, kot je vrednost spremenljivke `avenija`.
- Nalogo poskusi rešiti samostojno. Eno izmed pravih rešitev pa najdeš na spodnji povezavi.

[Povezava do rešitev](#)

## MIŠKA ZBIRA SIR

OŠ 7. – 9. r. ZAČETNIKI

Miška ima za svoj domek pripravljenih več vhodov in sicer dva ali tri. Neki dan je opazila, da pred vsakim vhodom leži košček sira. Usmeri jo, da odnese vse koščke v luknjo. Pri tem naj sir nosi čim krajši čas.



### Povezava do naloge

### Ideja reševanja

- Najprej si dobro oglejmo vse tri testne primere. Opazimo lahko, da se sirčki nikoli ne nahajajo v zgornji vrstici in desnem stolpcu, kar nam bo nekoliko olajšalo pisanje programa. Problem lahko razbijemo na dva manjša: 1) premikanje miške po celi mreži; 2) iskanje in premikanje sirčkov.
- Najprej napišimo program, ki bo miško premikal po celi mreži. Za premike po vrsticah uporabimo preprosto zanko. Premaknemo se torej 4-krat levo; 4-krat desno in enkrat gor; nato pa to ponovimo 3-krat z uporabo gnezdene zanke.
- Odločimo se, v kateri smeri bomo iskali koščke sira, npr. ko se premikamo v levo (v obeh smereh ni treba). Ker imamo več testnih primerov, moramo napisati univerzalno rešitev, torej bomo uporabili pogojni stavek. A biti moramo zelo pozorni, kajti ni dovolj, da le preverjamo, ali je na polju sir. Preveriti moramo tudi, da ni morda na istem polju tudi luknja, kajti v tem primeru bomo sirčke vzeli iz luknje. Uporabiti moramo torej pogojni stavek z dvema pogojema (v navadni pogojni stavek dodamo delček **in**). In ker ne smemo biti na polju z luknjo, pred senzor **v luknji** dodamo delček **ne**. Ko imamo pogoj napisan, napišemo še izvedbo: torej miška pobere sirček, se premakne polje navzgor, odloži sirček v luknjo in se vrne nazaj. S



slike lahko vidimo, da je luknja vedno natanko eno polje nad sirčkom, zato tega ni potrebno preverjati.

- Nalogo poskusi rešiti samostojno, rešitev pa najdeš na spodnji povezavi. Možne so tudi nekoliko drugačne rešitve.

**Povezava do rešitev**

## POVPREČNA DOLŽINA POTI V ŠOLO

OŠ 7. – 9. r. ZAČETNIKI

10 učencev je za domačo nalogo izmerilo, koliko metrov meri njihova pot v šolo. Podatke so zbrali v spodnji tabeli. Pomagaj jim izračunati, koliko metrov je v povprečju oddaljena šola od njihovih domov. Pazi, tvoj program mora delovati za vse testne primere, torej za različne šole.

Povprečno dolžino njihovih oddaljenosti izračunamo tako, da seštejemo oddaljenosti vseh učencev ter nato to število, delimo s številom vseh učencev (v tem primeru delimo z 10).

Vhod: 150 800 1200 740 2600 350 900 650 780	Izhod:	Vhod: 100 1800 3300 4700 2500 1400 1000 950 2400	Izhod:	Vhod: 700 500 650 450 500 780 350 520 4200	Izhod:
------------------------------------------------------------------------	--------	-----------------------------------------------------------------------------	--------	-----------------------------------------------------------------------	--------

### Povezava do naloge

### Ideja reševanja

- Najprej dobro preberimo navodilo in razčlenimo nalogo na posamezne korake. Napisati moramo program, ki bo prebral vseh 10 števil, iz njih izračunal povprečno vrednost in jo na koncu izpisal. Da bomo to lahko naredili, nujno potrebujemo spremenljivko. Nalogo lahko rešimo zgolj z uporabo ene spremenljivke, lahko pa si ustvarimo sami še eno.
- Najprej definiramo spremenljivko **vsota** in nastavimo njeno začetno vrednost na 0. Po vsaki ponovitvi bomo vrednost spremenljivke povečali za število, ki ga bo program prebral v vrstici. Ker v navodilih piše, da je podatke o dolžini poti vpisalo 10 učencev, to ponavljamo 10-krat z uporabo preproste zanke.
- Na koncu uporabimo spremenljivko **povprecna\_vrednost**, ki jo definiramo na začetku programa z vrednostjo nič. Po končani zanki vrednost delimo z 10 (namig, kako izračunati povprečno vrednost, je zapisan v navodilih). V zadnjem koraku izpišemo vrednost spremenljivke **povprecna\_vrednost**).

- Nalogo poskusi rešiti samostojno, eno izmed mnogih rešitev pa najdeš na spodnji povezavi. Nalogo lahko rešiš na veliko različnih načinov, tudi z manj delčki.


[Povezava do rešitev](#)


## PIŠEK VADI POŠTEVANKO

OŠ 7. – 9. r. ZAČETNIKI

Pišek je med počitnicami nekoliko pozabil poštevanko. Ker je učenje med gibanjem vedno bolj zabavno, to počne tudi Pišek. Pišek se je najprej premaknil po mreži do obeh faktorjev, ki si ju je zapomnil, nato pa je v modro polje vpisal njun produkt. Ugotovil je, da je pri množenju naredil nekaj napak.

Napiši program, ki bo preveril njegove izračune in polja s pravilnimi produkti pobarval z zeleno, tista z napačnimi pa z rdečo barvo.

						
	2	·	10	=	20	
	5	·	3	=	12	
	8	·	7	=	62	
	6	·	4	=	24	
	9	·	7	=	62	

						
	7	·	8	=	56	
	10	·	9	=	90	
	5	·	6	=	30	
	2	·	4	=	6	
	1	·	2	=	2	

### Povezava do naloge

### Ideja reševanja

Najprej dobro preberimo navodilo in si oglejmo oba testna primera. Ugotovimo lahko, da sta oba testa podobna, razlikujeta se le v različnih računih. To pomeni, da bomo morali napisati univerzalni program, za kar potrebujemo spremenljivke. Razčlenimo nalogo na posamezne korake:

1. Pišek se mora premakniti po celotni mreži;
  2. program mora prebrati oba faktorja in ju shraniti vsakega v svojo spremenljivko;
  3. program izračuna produkt faktorjev v vsaki vrstici;
  4. preverimo, ali je izračunan produkt enak tistemu, ki je zapisan v mreži in polje pobarvamo z zeleno/rdečo barvo.
- Najprej napišimo program za premikanje po mreži. Pišek se bo premaknil 1 polje navzdol, 5 polj v desno in nato 5 polj v levo. To pa bo ponovil za vseh 5 vrstic, torej uporabimo gnezdeno zanko. Ta program bomo v

naslednji fazi sicer nekoliko spremenili zaradi branja podatkov, a je dobra osnova.

- Najlažje je, da Pišek prebere in si zapomni podatke, ko se premika v desno. Zato nekoliko predelamo kodo za premikanje v desno. Pišek se premakne v desno za 1 polje, in si v spremenljivko **faktor1** shrani vrednost na polju. Nato se premakne do drugega faktorja in si shrani njegovo vrednost v novo spremenljivko (ustvarimo si jo sami – npr. **faktor2**).
- V spremenljivko **produkt** shranimo zmnožek obeh faktorjev, za kar potrebujemo delčke iz sklopa »Matematika«.
- Pišek se na koncu premakne do polja s produktom. Uporabimo pogojni stavek (delček **če ..., izvedi ..., sicer ...**) in preverimo, ali je številka na polju enaka vrednosti, ki smo jo shranili v spremenljivki **produkt**; če je, polje pobarvamo zeleno, sicer pa rdeče.
- Nalogo poskusi rešiti samostojno, eno izmed mnogih rešitev pa najdeš na spodnji povezavi. Nalogo lahko rešiš na veliko različnih načinov, tudi z manj delčki. Uporabiš lahko več in pa tudi manj spremenljivk. Tu smo predstavili le eno izmed možnosti.

[Povezava do rešitev](#)

## ŠIFRIRANJE SPOROČIL

OŠ 7. – 9. r. NAPREDNI

Piškovi bandi je policija začela prestrezati sporočila. Zato so se dogovorili, da bodo svoja sporočila šifrirali tako, da bodo v vsaki besedi obrnili vrstni red črk. Pomagaj jim in sestavi program, ki jim bo sporočila šifriral.

Vhod: mačka	Izhod:
----------------	--------

### Povezava do naloge

### Ideja reševanja

- Na začetku programa preberi besedilo, ki je na vhodu. V ta namen uporabi delček `preberi vrstico` ter prebrano vrednost shrani v ustrezno spremenljivko, npr. v spremenljivko `besedilo`.
- Glede na delčke, ki so na voljo, boš spoznal, da je pri izpisu šifriranega besedila treba izpisovati črko po črko in, da bo treba uporabiti zanko `za`, ki se bo ponovila tolikokrat, kot je dolžina besedila.
- Da boš ugotovil dolžino besedila, uporabi delček `dolžina` in vrednost, ki predstavlja dolžino besedila, shrani v spremenljivko `dolzina`.
- Sedaj uporabi zanko `za` (oz. zanko »for«). Glavna ideja pri nalogi je ta, da zanko `za` (s spremenljivko `i`) ne izvajaš od 1 do `dolzina`, kjer se `i` spreminja za korak 1; temveč, da zanko `za` izvajaš nazaj oz. od `dolzina` do 1 (kjer se `i` spreminja tudi za korak 1). Ob vsaki ponovitvi zanke iz besedila izpišeš in vrneš črko na `i`-tem mestu (oz. `i`-tem indeksu).
- Ker boš zanko `za` izvajal »nazaj«, pa to pomeni, da boš sprva izpisal zadnjo črko (saj je indeks `i = dolzina`), nato predzadnjo črko (saj je indeks `i`

=  $\text{dolžina} - 1$ ) ... in ob zadnji ponovitvi zanke **za** še prvo črko (ko je indeks  $i = 1$ ).


[Povezava do rešitev](#)

## VEVERIČKA IN OZIMNICA

OŠ 7. – 9. r. NAPREDNI

Veverica je zelo skrbna. Na zimo se dobro pripravi, saj si naredi zalogo lešnikov. Ne hrani vseh lešnikov na enem mestu, ampak različno velike kupčke lešnikov skrije na različna mesta po gozdovih (Test 1 in Test 2). Koliko je lešnikov na posameznem kupčku, povedo številke v mreži. Senzor **številka polja** pove to številko. Če na polju ni številke, senzor vrne 0.

A kaj, ko ne ve, ali si je za zimo pripravila že dovolj veliko zalogo lešnikov. Pomagaj ji in seštej, koliko lešnikov je do zdaj shranila v posameznem gozdu. To število vpiši v polje, ki ima zdaj vrednost 0.

		2			3		
	1			2			
		1				1	
			3		3		
		2		2			
0							

### Povezava do naloge

### Ideja reševanja

- Sprva premisli, kako boš premikal veverico. Glede na postavitev veverice in mest kupčkov z lešniki, je eden izmed načinov, kako lahko premikaš veverico, naslednji: »7x premik desno → 7x premik levo → (in nato) 1x premik navzdol.« Vse to pa je treba »5x ponoviti«.
- Za seštevanje števila lešnikov v gozdu uporabi (že definirano) spremenljivko **sestevek**, ki jo pred začetkom premikanja veverice nastavi na vrednost 0 (lešnikov).



- Ko boš veverico premikal po prej opisani poti, je treba pred vsakim premikom v desno, pridobiti številko polja s senzorjem **številka polja** in to vrednost (oz. število lešnikov) prešteti k skupnemu seštevku vseh lešnikov oz. k spremenljivki **sestevek**. Pri spreminjanju seštevka lahko uporabiš delček **spremeni** ali **nastavi**.
- Za konec vrednost spremenljivke **sestevek** (ki torej predstavlja seštevek vseh lešnikov v gozdu) vpiši na spodnje levo polje, ki ima na začetku vrednost 0.

**Povezava do rešitev**

## BARVANJE OGRAJE

OŠ 7. – 9. r. NAPREDNI

Za robota potrebujemo program, s katerim bo pobarval deščice ograje. Barva deščic ograje se mora menjati v zaporedju: rdeča, zelena, modra. Za rdečo deščico naj bo zelena, za zeleno modra in za modro zopet rdeča, itd. Prva deščica je že pobarvana. Sestavi program, po katerem bo na podlagi prve deščice robot pravilno pobarval ograjo do konca.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je nekaj delčkov – za idejo – podanih že vnaprej. Ideja je, da na začetnem polju preberemo barvo prve deščice, nato pa posamezni barvi priredimo določeno številko. Tako za »rdečo barvo« v spremenljivki **stevilka** priredimo vrednost 0. Zeleni in modri barvi pa bi lahko, priredili vrednosti 1 in 2. Torej, če bo na začetnem polju »zelena barva« bo **stevilka** imela vrednost 1, v kolikor bo pa na začetnem polju »modra barva«, pa bo **stevilka** imela vrednost 2.
- V nadaljevanju je nato pripravljena zanka, ki se 10-krat ponovi – saj imamo 10 polj. Ob vsaki izvedbi zanke pa se premaknemo naprej; povečamo vrednost spremenljivke **stevilka** za 1; ter uporabimo več pogojnih stavkov.
- V pogojnih stavkih pa je treba preveriti vrednost spremenljivke **stevilka** in nato, glede na njeno vrednost, pobarvati polje z ustrezno barvo.
- Glede na predlagano idejo je v pogojnih stavkih smiselno uporabiti delček **ostanek pri `stevilka` ÷ 3**, kjer preverimo ali je ta ostanek enak 0, 1 ali 2.

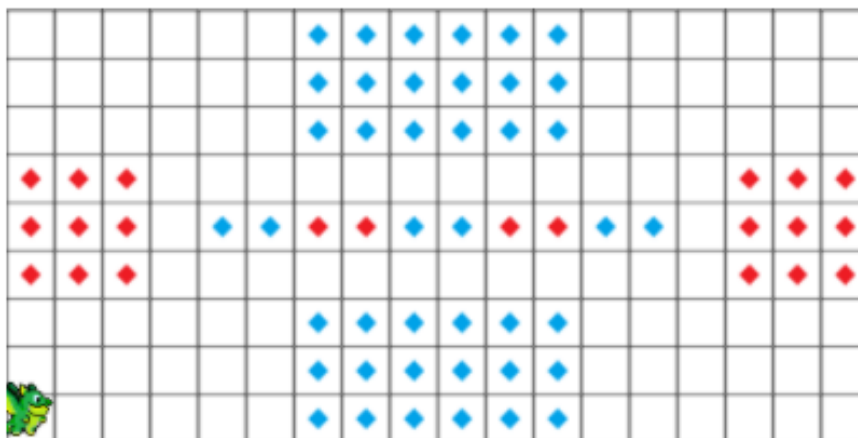
### Povezava do rešitev

## ZMAJČEK IN TLAKOVANJE

OŠ 7. – 9. r. NAPREDNI

Zmajček bi rad polepšal svoje dvorišče z barvnimi ploščicami. Trenutno so vse ploščice na dvorišču bele. Z barvnimi pikami si je označil, kam bo položil modre in kam rdeče ploščice. Če na ploščici ni pike, jo bo pustil belo. Dopolni funkcijo **Preveri in tlakuj**, ki naj preverja, ali je na posamezni ploščici pika, in če je, katere barve ploščico mora tam položiti in jo nato naj položi.

Nato dopolni še program, ki bo vsako ploščico dvorišča preveril s pomočjo funkcije in ustrezno tlakoval.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je treba dopolniti (glavni) program in funkcijo **Preveri in tlakuj**.
- Za začetek začnimo z dopolnitvijo funkcije **Preveri in tlakuj**. To funkcijo bomo zapisali tako, da bomo s pomočjo te funkcije preverili barvo »barvne pike« in nato glede na (prebrano) barvo pike, položili ploščico ustrezne barve. V funkciji bomo zato uporabili dva pogojna stavka, s katerima bomo preverili, če je barvna pika modra ali rdeča.
- V samem glavnem programu pa je treba sprogramirati premikanje Zmajčka. Pri premikanju Zmajčka je treba paziti, da prav na vsakem polju dvorišča preverimo barvo »barvne pike« in položimo mogočo ploščico

(zaradi mogočih ploščic na robovih dvorišča) oz. »kličemo« predhodno definirano funkcijo **Preveri in tlakuj**.

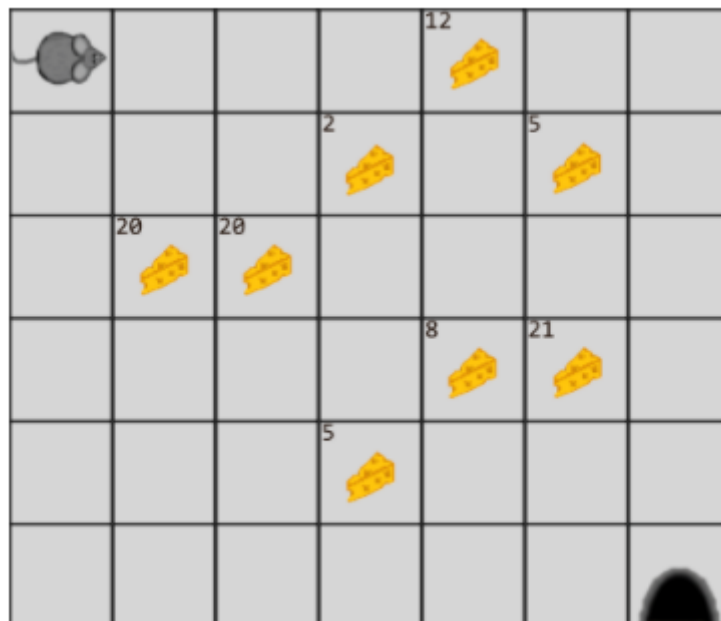
**Povezava do rešitev**

## MIŠKA IŠČE SIR

OŠ 7. – 9. r. NAPREDNI

Miške po skednju vedno hodi po istih poteh. Njena pot je že zapisana v obliki ukazov, ki jih vidiš na delovni površini. Ko se premika, naleti na koščke sira, ki pa imajo različno število lukenj. Miška bi rada v svojo luknjico odnesla sir, ki ima največ lukenj. Dopolni njen program gibanja tako, da bo na koncu v luknji košček z največ luknjami.

Seveda miška lahko naenkrat nese le en košček. Število lukenj koščka, ki ga nese, je v spremenljivki `stevilo_lukenj`. Na začetku je v spremenljivki `stevilo_lukenj` kar vrednost 0. Da miška ugotovi število lukenj, uporabi senzor `koliko_lukenj`. Če na polju ni sira, ta senzor vrne 0. Miška pobere sir z ukazom `poberi sir`. Vendar mora pred tem, če že nosi kak košček sira, tega odložiti.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je treba dopolniti (glavni) program. Če zaženemo začetni program, opazimo, da je miška že sprogramirana tako, da preišče celotno mrežo, vendar v svojo luknjo ne prinese sira z največ luknjami. To pomeni,

da je v predlaganem programu napaka pri iskanju sira z največ luknjami (oz. v pogojnem stavku).

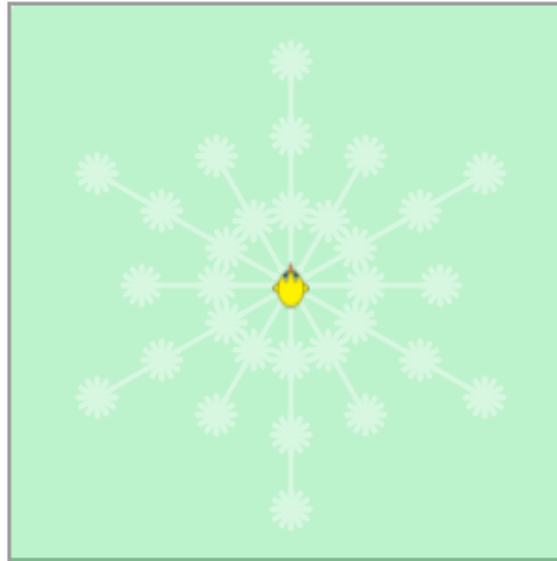
- Prav tako imamo v začetnem programu že definirano spremenljivko **stevilo\_lukenj**, v katero bomo shranili številsko vrednost, ki bo ponazarjala, »koliko lukenj ima sir, ki ga trenutno nosimo«. Ker na začetku ne nosimo nobenega sira je vrednost te spremenljivke nastavljena na 0.
- Kot piše v navodilu naloge, so »posebni delčki« za to nalogo senzor **koliko\_lukenj** s katerim preverimo koliko lukenj ima sir na trenutnem polju; ter delčka **poberi\_sir** in **odloži\_sir**, s katerim lahko sir poberemo ali odložimo.
- Pri nalogi je treba paziti, da poberemo sir le takrat, kadar smo na polju s sirom oz. sir odložimo, kadar miška že nosi sir.
- Ideja je, da v pogojni stavek zapišemo pogoj s katerim bomo preverili, ali je vrednost v **koliko\_lukenj** večja od vrednosti v **stevilo\_lukenj**; kar z drugimi besedami pomeni, da ima sir na trenutnem polju več lukenj, kot sir, ki ga miška trenutno nosi. Vendar preden sir z manj luknjami odložimo in tistega z več luknjami poberemo; je treba nastaviti vrednost spremenljivke **stevilo\_lukenj** na novo vrednost. Šele nato odložimo »trenutni« sir in »novega« poberemo.
- Pri odlaganju sira pa je treba paziti, da na začetku, miška ne nosi nobenega sira, kar pomeni, da »sira«, ob prvi najdbi sira z več luknjami, ne odložimo oz. ga odložimo le takrat, ko je **stevilo\_lukenj** različno (!=) od 0 (oz. ko sir že nosimo).

[Povezava do rešitev](#)

## REGRATOVA LUČKA

OŠ 7. – 9. r. NAPREDNI

Pišek ima zelo rad regratove lučke. Program, ki nariše regratovo lučko, je delno že narejen. Pripravljen imaš funkcijo `padalo`, ki izriše eno `padalo` regratove lučke. Tudi funkcija `vrstica` je že sestavljena. Manjka samo program, ki nariše več vrstic v regratovi lučki. Pomagaj Pišku dokončati program.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je treba Pišku pomagati dokončati (glavni) program. Če zaženemo začetni program, opazimo, da Pišek nariše eno vrstico s pripadajočimi »padali«. To vrstico nariše funkcija `vrstica` s parametroma `n` (kateri določa število »padal«) in `m` (ki določa število »krilc« v padalu), ki med izvajanjem pokliče tudi funkcijo `padalo`.
- S pomočjo sklepanja (ali poskusi), lahko ugotovimo, da če bomo za parameter `n` uporabili število `2`, da bo Pišek narisal vrstico z dvema padaloma.
- Pri krajši vrstici je treba paziti, da se nazaj premaknemo za 80 korakov in ne 120 korakov.
- Izmenično funkcijo `vrstica` uporabimo za risanje daljše in krajše vrstice s padali. Ker imamo šest ponovitev dolgih in kratkih krilc, vso zadevo zapišemo v zanko, ki se 6x ponovi.

[Povezava do rešitev](#)

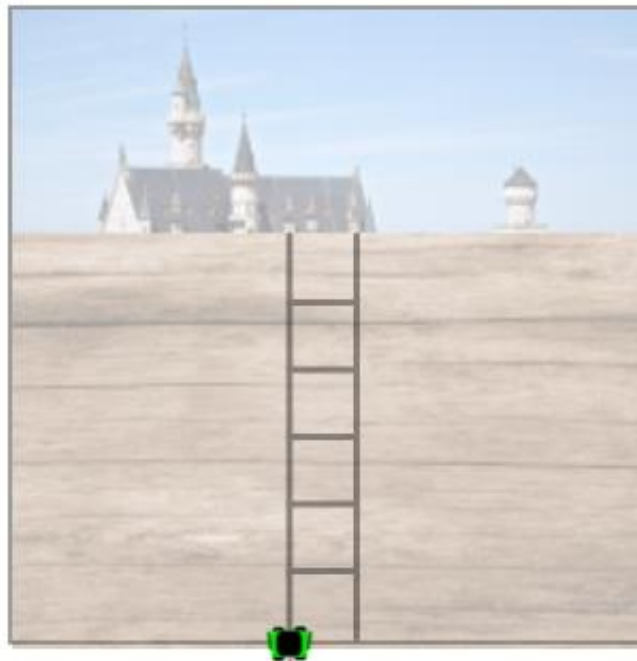


## ROBOTEK BOJAN RIŠE

SŠ ZAČETNIKI

Robotek Bojan bi rad videl grad, ki je skrit za ograjo. Ker je premajhen, išče lestev, a je ne najde. Potem pa si reče: »Saj sem robotek Bojan, ki zna vse, kar potrebuje, narisati. Torej lahko naredim program, ki bo lestev narisal na ograjo«.

Začel je ustvarjati program, a je naletel na težave. Pomagaj mu! Lestev naj bo široka 100 korakov. Pazi pa, da ne boš podvajal črt!



### Povezava do naloge

### Ideja reševanja

- Robotek Bojan lahko lestev nariše na več različnih načinov. V nadaljevanju bo predstavljena ideja reševanja, kjer robotek Bojan nariše lestev tako, da sprva nariše dve navpični (daljši) črti, ki sta dolžini 600 korakov; nato pa še petkrat doda vodoravne prečke, ki so široke 100 korakov
- Da bo Bojan narisal navpično prečko (glede na njegov začetni položaj in smer), se mora s spuščnim pisalom premakniti za 600 korakov naprej (oz. navzgor). Ker smo končali z risanjem, se z dvignjenim pisalom premaknimo na drugo navpično prečko, ki jo bomo narisali navzdol. Če delčke ustrezno zapišemo; lahko celotno zadevo zapišemo v zanko, ki se dvakrat ponovi.

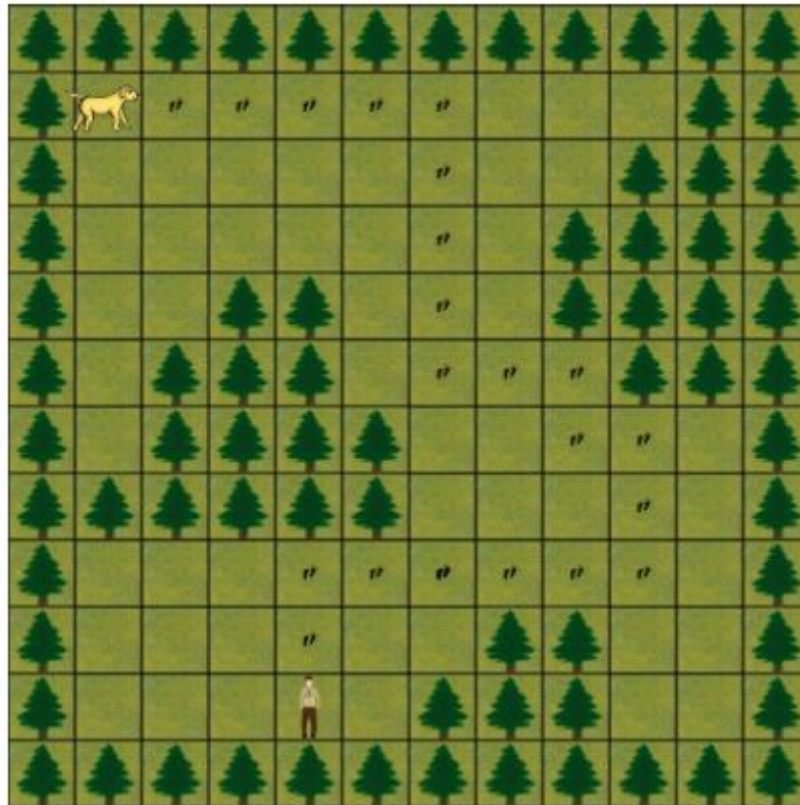
- Ko smo narisali navpični črti, pa je treba narisati še vodoravne prečke. Pri risanju vodoravnih prečk je treba paziti, da je pisalo spuščeno le takrat, kadar rišemo vodoravno prečko; ob premikanju pa je pisalo vselej vzdignjeno.

[Povezava do rešitev](#)

## ISKANJE POGREŠANEGA

SŠ ZAČETNIKI

Nada je reševalna psička, ki pomaga pri iskanju izgubljenih ljudi. Na podlagi sledi oz. stopinj naj Nada najde v gozdu izgubljeno osebo.



### Povezava do naloge

### Ideja reševanja

- Nalogo je treba rešiti od začetka, prav tako pa opazimo, da je treba biti pri uporabi delčkov ekonomičen, saj jih lahko uporabimo le 15.
- »Posebni delčki« za to nalogo so senzorji **na polju je oseba**, **na polju spredaj je oseba** in **na polju spredaj je sled**.
- Naloga Nade je, da hodi po sledi, vse dokler ne pride do polja, kjer je izgubljena oseba. Da bomo to dosegli je treba je uporabiti zanko **ponavljaj dokler ni**, ki se izvaja, dokler ne pridemo na polje z izgubljeno osebo. Znotraj te zanke je treba ugnezditi še zanko **ponavljaj medtem ko**, ki se izvaja dokler je na polju spredaj sled. In če je na polju

spredaj sled, se Nada premakne naprej. Ko pa na polju spredaj ni več sledi pa sta možnosti dve:

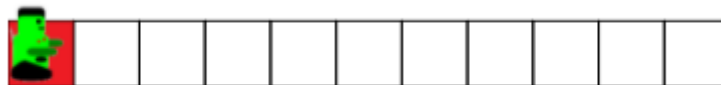
1. da je na polju spredaj že izgubljena oseba ali,
  2. da smo izgubili sled in da se je treba obrniti v levo ali desno.
- Če je na polju spredaj oseba (1. možnost), se premaknemo naprej in zanka se bo zaključila. Sicer (2. možnost) se mora Nada obrniti levo (ali desno), da poišče novo sled. Če pa tudi tam ni sledi, se mora Nada obrniti še naokoli, kjer pa sled zagotovo je.

[Povezava do rešitev](#)

## BARVANJE OGRAJE

SŠ ZAČETNIKI

Potrebujemo program, s katerim bo robot barval deščice ograje. Barva deščic ograje se mora menjati v zaporedju: rdeča, zelena, modra. Za rdečo deščico naj bo zelena, za zeleno modra in za modro zopet rdeča, itd. Prva deščica je že pobarvana. Dopolni program, s katerim bo robot, na podlagi prve deščice, pravilno pobarval ograjo do konca.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je nekaj delčkov – za idejo – podanih že vnaprej. Ideja je, da na začetnem polju preberemo barvo prve deščice, nato pa posamezni barvi priredimo določeno številko. Tako za »rdečo barvo« v spremenljivki **stevilka** priredimo vrednost 0. Zeleni in modri barvi pa bi lahko, priredili vrednosti 1 in 2. Torej, če bo na začetnem polju »zeleno barvo« bo **stevilka** imela vrednost 1, če bo pa na začetnem polju »modra barva«, pa bo **stevilka** imela vrednost 2.
- V nadaljevanju je nato pripravljena zanka, ki se 10-krat ponovi – saj imamo 10 polj. Ob vsaki izvedbi zanke pa se premaknemo naprej; povečamo vrednost spremenljivke **stevilka** za 1; ter uporabimo več pogojnih stavkov.
- V pogojnih stavkih pa je treba preveriti vrednost spremenljivke **stevilka** in nato, glede na njeno vrednost, pobarvati polje z ustrezno barvo.
- Glede na predlagano idejo je v pogojnih stavkih smiselno uporabiti delček **ostanek pri `stevilka` ÷ 3**, kjer preverimo ali je ta ostanek enak 0, 1 ali 2.

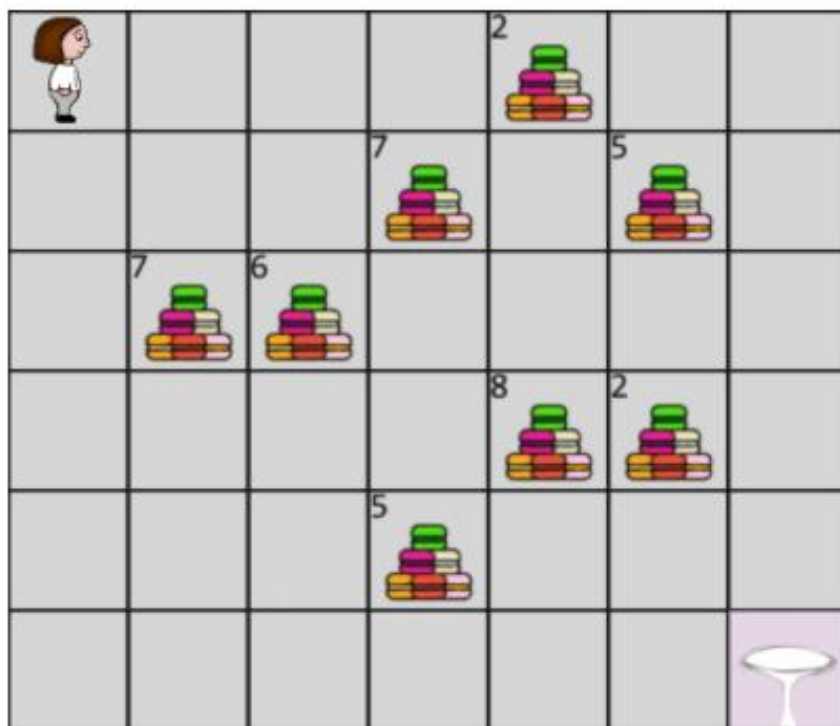
### Povezava do rešitev

## SLAŠČIČAR STANE

SŠ ZAČETNIKI

Slaščičar Stane mora iz skladišča prinesiti najvišjo piramido, sestavljeno iz makronov. Njegova pot je že zapisana v obliki ukazov, ki jih vidiš na delovni površini. Ko se premika, naleti na piramide, ki imajo različno višino. Stane bi rad na končno polje pripeljal najvišjo piramido. Zato dopolni njegov program gibanja tako, da bo na koncu na končno polje odložil najvišjo piramido.

Seveda lahko Stane naenkrat nosi le eno piramido. Višina piramide je zapisana v spremenljivki `visina_piramide`. Na začetku nima v roki nobene piramide, zato je v spremenljivki `visina_piramide` kar vrednost 0. Da ugotoviš višino piramide na polju, imaš na voljo senzor `višina_piramide`. Če na polju ni piramide, ta senzor vrne 0. Če želiš vzeti piramido, ki je na polju, uporabi ukaz `naloži_piramido`. A če Stane že kaj nosi, naj prejšnjo piramido najprej odloži.



### Povezava do naloge

### Ideja reševanja

- Pri nalogi je treba dopolniti (glavni) program. Če zaženemo začetni program, opazimo, da je Stane že sprogramiran tako, da preišče celotno skladišče, vendar na zadnje polje še ne prinese najvišje piramide. To

pomeni, da je v predlaganem programu napaka pri iskanju najvišje piramide (oz. v pogojnem stavku).

- Prav tako imamo v začetnem programu že definirano spremenljivko **visina\_piramide** (številsko vrednost), ki ponazarja »kolikšna je višina piramide, ki jo Stane trenutno nosi«. Ker Stane na začetku ne nosi še nobene piramide z makronov je vrednost te spremenljivke nastavljena na 0.
- Kot piše v navodilu naloge, so »posebni delčki« za to nalogo senzor **višina piramide**, s katerim preverimo kolikšna je višina piramide na trenutnem polju; ter delčka **naloži piramido** in **odloži piramido** z uporabo katerih, lahko pobereмо ali odložimo piramido z makronov.
- Pri nalogi je treba paziti, da pobereмо piramido makronov le tedaj, kadar smo na polju, kjer se nahaja »piramida makronov« oz. piramido makronov odložimo, kadar Stane že predhodno nosi piramido makronov.
- Ideja je, da v pogojni stavek zapišemo pogoj s katerim bomo preverili, ali je **višina piramide** (senzor) večje od **visina\_piramide** (spremenljivka); kar z drugimi besedami pomeni, da Stane stoji na polju, kjer je višina piramide večja od višine piramide iz makronov, ki jo Stane trenutno nosi v rokah. Vendar preden Stane »piramido z manjšo višino« odloži in tisto z večjo višino pobere (zamenja), je treba nastaviti vrednost spremenljivke **visina\_piramide** na novo vrednost. Šele nato odložimo »trenutno« piramido iz makronov in »ta višjo« pobereмо.
- Pri odlaganju piramide z makronov pa je treba paziti, da Stane na začetku, ne nosi še nobene piramide in smemo odložiti »piramide makronov«, ob prvi najdbi »višje piramide makronov«; oz. piramido makronov odlaga Stane šele takrat, ko je **stevilo\_lukenj** različno (!=) od 0 (oz. ko piramido že nosi).

### Povezava do rešitev

## CEZARJEVA ŠIFRA

SŠ ZAČETNIKI

Pišek se je odločil, da bo pri komunikaciji s prijatelji šifriral sporočila in uporabljal t. i. Cezarjevo šifro.

*Cezarjeva šifra* je šifriranje, pri katerem na podlagi šifrirnega ključa vsaki črki v besedilu določimo drugo črko. Šifrirni ključ določi, za koliko mest se zamakne »tajna abeceda« glede na »običajno abecedo«.

Primer. Če bi želeli zašifrirati besedilo: »SAMO BREZ PANIKE« s šifrirnim ključem 3, bi to besedilo bilo zašifrirano takole: »UČPS DTHB ŠČRLNH«; oz. črka »S« (z zamikom 3) bi postala črka »U«; črka A (z zamikom 3) bi postala črka »Č«; itn.

Pomagaj mu napisati program, ki mu bo omogočal zašifrirati poljubno besedilo, sestavljeno iz (velikih) črk. Najprej preberi zamik, potem pa besedilo. Nato pa izpiši ustrezno predelano besedilo.

Pri tem ti bo v pomoč funkcija `cezarjeva_sifra`.

<p>Vhod:</p> <p>3</p> <p>SAMO BREZ PANIKE</p>	<p>Izhod:</p>
-----------------------------------------------	---------------

### Povezava do naloge

### Ideja reševanja

Najprej si je dobro ogledati, kaj počne funkcija `cezarjeva_sifra`. Ta dobi dva podatka, `c` in `k`. V spremenljivki `c` je znak, ki ga želimo spremeniti, v `k` pa šifrirni ključ. Preizkusimo delovanje funkcije na nekaj primerih. V ta namen ustvarimo dve spremenljivki `znak` in pa `ključ`, ter programu dodamo vrstici `nastavi ključ na 5` in `nastavi znak na C`. Sedaj izpišemo rezultat z `izpiši` in klicem funkcije `cezarjeva_sifra` s parametroma `znak` in `ključ`. Sistem bo seveda protestiral, da tvoja rešitev ni pravilna, vendar nas zanima le, kaj se



napiše pod »Vaš odgovor«. V podanem primeru bo izpisal »G«, ker je »G« za 5 mest v abecedi oddaljen od C. Spremenimo še vrednost **znak** v »U«. Sedaj pričakujemo, da bomo dobili »B«. Ko se premikamo od »U« v desno za 5x, po 3 premikih »pridemo« do »Ž« (no zares ne, ker se celoten premik izvede naenkrat), nato pa do »A« in »B«. In res nas Pišek opozori, da je naš odgovor »B«. Vidimo torej, da nam funkcija vrne znak, ki nadomesti obstoječi znak v besedilu.

Kako je funkcija prišla do tega ustreznega znaka? Poiščemo, na katerem mestu v spremenljivki **abeceda** je znak, ki ga spreminjamo. Nato z uporabo šifrirnega ključa izračunamo, na katerem mestu bo spremenjeni znak. Pri tem najprej prištejemo **k - 1**, nato pa uporabimo ostanek pri deljenju, saj se lahko zgodi, da bomo pri premikanju »padli« čez konec (kot smo pri zgornjem primeru pri »U«) in moramo ponovno začeti od začetka.

Sedaj, ko vemo, kaj počne funkcija, je pot do rešitve enostavna.

1. Z delčkom **preberi celo število** nadomestimo našo 5, saj bomo šifrirni ključ zdaj prebrali.
2. Ustvarimo novo spremenljivko **besedilo** in v tretji vrstici našega programa spremenljivko **znak** spremenimo v **besedilo**, namesto znaka »u« pa ji priredimo z **preberi vrstico** prebran tekst za šifriranje.
3. Ustvariti moramo še eno spremenljivko, kamor bomo shranjevali celotno šifrirano besedilo, recimo **novob**. Na začetku ji priredimo kar prazen niz, ki ga najdemo med delčki »Besedilo«
4. Zdaj se le še zapeljemo preko celotnega besedila in vsak znak s klicem funkcije pretvorimo v novega in ga dodamo k **novob**. Potrebovali bomo delček **za ... od ... do ... s korakom ...**, kjer bo začetek **1**, konec dobimo z **dolžina(besedilo)**, korak pa je seveda **1**. Da dobimo znak, ki ga hočemo pretvoriti, pa nam prav pride delček **iz besedila vrni črko št.**
5. Na koncu, izven zanke, le še izpišemo to, kar smo nabrali v spremenljivko **novob**.

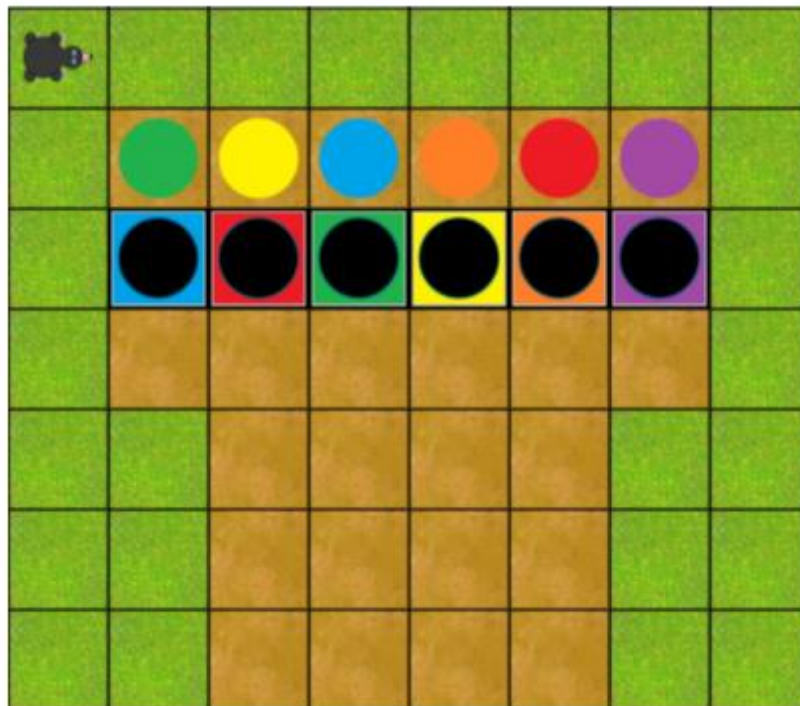
[Povezava do rešitev](#)

## KRTKOVA DRUŽINA

SŠ ZAČETNIKI

Šestčlanska družina krtov je šla na potep. Sredi poti so se spomnili, da so pozabili zapreti vrata svojih rogov. Eden od krtov se mora vrniti in premakniti barvne kamne na pravo odprtino. Pomagaj mu in ustvari program, ki bo krtka vodil tako, da bo vsak barvni kamen zaprl rov iste barve. V pomoč imaš senzorje, ki preberejo barvo kamna in barvo rova.

Pazi, barvo kamna mora krt preveriti s senzorjem, preden ga pobere. Podobno velja za rove. Pozorno si oglej vse testne primere.



### Povezava do naloge

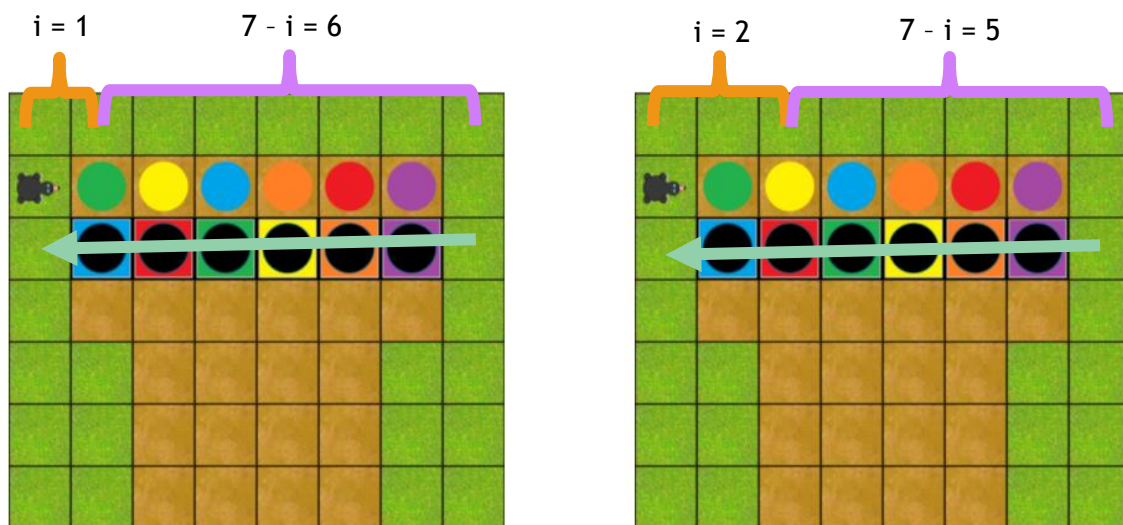
### Ideja reševanja

Ena izmed rešitev naloge je lahko naslednja. Predpostavimo, da krtka pobira kamne po vrsti – od leve proti desni. Potem ko krtka pobere enega izmed kamnov se nato premika v desno – vse do konca vrstice s kamni; nato pa se premakne navzdol, v vrstico z rovi. Ko je krtka v drugi vrstici, pa med premikanjem v levo, išče ustrezni rov, ki ga bo zaklenil s pravkar pobranim kamnom.

Ker imamo šest kamnov in rogov je potrebno napisati zanko po kateri bo krtka šestkrat (v smeri urinega kazalca) zaokrožil okoli kamnov in rogov. Bistveno pri

vsem tem je, da krtek natančno »ve« kateri kamen mora v nekem trenutku pobrati (številko kamna lahko shranimo v spremenljivko **i**).

Npr. če predpostavimo, da krtek stoji v prvem stolpcu in mora pobrati prvi kamen, se mora sprva premakniti za en korak v desno, s senzorjem **barva kamna** prebrati barvo kamna (ter to barvo shraniti v spremenljivko), nato pa pobrati kamen in iti še šest korakov v desno, vse do konca vrstice s kamni; če mora pobrati drugi kamen, se mora sprva premakniti v desno za dva koraka, prebrati barvo kamna ter kamen pobrati in nato iti naprej še pet korakov v desno, itn.



V drugi vrstici, kjer so rovi njegove družine, pa krtka zgolj vrnemo za sedem korakov nazaj v levo, sproti pa s pogojnim stavkom preverimo če je krtek na ustreznem rovu, torej če je barva kamna enaka barvi rova in če je, s pravkar pobranim kamnom zaklenemo rov.

Vse to je najlažje zapisati z zanko **za**.

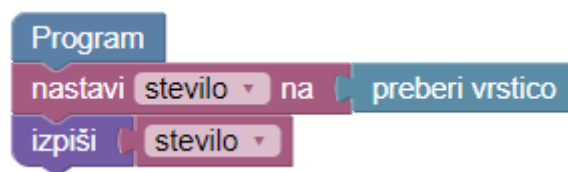
[Povezava do rešitev](#)

## EKSPERIMENT S SODIMI ŠTEVILI

SŠ NAPREDNI

Pišek je vprašal deset svojih prijateljev po številih med 1 in 100. Zanimalo ga je, ali jih bo ravno polovica odgovorila s sodim številom. Pomagaj Pišku prešteti soda števila in izpiši »Da« v primeru, da jih je res polovica, v nasprotnem primeru pa »Ne«.

Na delovni površini je napisan program, ki prebere naslednje število iz seznama in ga izpiše.



Trenutni program glede na dane podatke (Test 1) izpiše število 2, moral pa bi izpisati »Ne«. Pazi, Pišek je ponovil vajo s prijatelji 3 krat!

Input:	Output:
2	
12	
42	
23	
42	
82	
22	
41	
21	

### Povezava do naloge

### Ideja reševanja

Pri nalogi je treba narediti naslednje.

Na začetku programa je treba nastaviti spremenljivko **stevec** na vrednost 0. Z uporabo te spremenljivke bomo preštevali soda števila, ki so jih povedali Piškovi prijatelji.

Nato pa je treba zapisati zanko, ki se bo »10x krat« ponovila (saj imamo 10 vrstic števil). Ob vsaki iteraciji oz. za vsako prebrano število pa je treba preveriti, če je število sodo ali ne. Sodost števila preverim tako, da preverimo, če je ostanek pri

deljenju z 2 enak 0 in če je, to pomeni, da je to število sodo (oziroma, pri tej nalogi lahko uporabimo tudi delček **0 je sodo**). In v tem primeru povečamo **stevec** za +1.

Ko se izvajanje zanke zaključi, pa s pogojnimi stavkom preverimo, ali je bilo natanko 5 števil sodih oz. nas zanima, če je **stevec == 5** in če je izpišemo »Da«; sicer pa »Ne«.

[Povezava do rešitev](#)

## NAJTEŽJI ZABOJ

## SŠ NAPREDNI

Viličar se premika po skladišču. Pri tem naleti na zaboje, ki so različno težki. Viličar bi rad na končno polje pripeljal najtežji zaboj. Zato napiši program gibanja viličarja tako, da bo na koncu na končno polje odložil najtežji zaboj.

Seveda je lahko na viličarju naenkrat le en zaboj. Njegova teža je zapisana v spremenljivki `teza_na_vilicarju`. Na začetku nima naloženega paketa, zato je v spremenljivki `teza_na_vilicarju` kar vrednost 0. Da ugotoviš težo zaboja na polju, imaš na voljo senzor `teža_zaboja`. Če na polju ni zaboja, ta senzor vrne 0. Če želiš na viličarja naložiti zaboj, ki je na polju, uporabi ukaz `poberi_zaboj`. A če viličar že nosi zaboj, ga mora prej odložiti.

### Povezava do naloge

### Ideja reševanja

Reševanja naloge se lahko lotimo takole ...

Sprva napišimo program, s katerim bomo viličarja premikali po skladišču.

Viličarja lahko premikamo takole: »6x premik v desno → 6x premik v levo → (in nato) 1x premik navzdol.« Vse to pa ugnezdimo v zanko, ki jo 5x ponovimo, saj imamo 5 vrstic zabojev.

Ko se zanka zaključi, pa viličarja premaknemo le še 6x premikov v desno, kjer na koncu, odložimo, najtežji zaboj.

Zdaj pa še razmislimo, kako bomo v skladišču poiskali najtežji zaboj. Na začetku imamo podano spremenljivko `teza_na_vilicarju` (številsko vrednost), ki ponazarja »kolikšna je trenutna teža najtežjega zaboja«. Ker viličar na začetku ne nosi še nobenega zaboja, je vrednost te spremenljivke nastavljena na 0.

Kot piše v navodilu naloge, so »posebni delčki« za to nalogo senzor `teža_zaboja`, s katerim preverimo, kolikšna je višina piramide na trenutnem polju; ter delčka po `poberi_zaboj` in `odloži_zaboj` z uporabo katerih, lahko poberemo ali odložimo zaboj.

Pri nalogi je treba paziti, da poberemo, ali odložimo zaboj le tedaj, kadar smo na polju, kjer se nahaja zaboj in zaboj odložimo, le takrat kadar viličar že predhodno nosi zaboj.

Ideja je, da v pogojni stavek zapišemo pogoj s katerim bomo preverili, ali je **teža zaboja** večja od **teza\_na\_vilicarju** (spremenljivka); kar z drugimi besedami pomeni, da viličar stoji na polju, kjer je teža zaboja večja od teže zaboja, ki jo viličar trenutno ima. Vendar preden viličar svoj trenutni zaboj odloži in tistega z večjo težo vzame je treba nastaviti vrednost spremenljivke **teza\_na\_vilicarju** na novo vrednost. Šele nato odložimo »trenutni« zaboj in »težjega« poberemo.

Pri odlaganju zabojev pa je treba paziti, da viličar na začetku, ne nosi še nobenega zaboja in smemo odložiti zaboj, ob prvi najdbi »težjega zaboja«; oz. viličar zaboj prvič odloži šele takrat, ko je **teza\_na\_vilicarju** različna (!=) od 0 (oz. ko viličar zaboj že nosi).

[Povezava do rešitev](#)


## ODNAŠANJE SMETI

SŠ NAPREDNI

V Piškovi tričlanski družini vsi člani zelo radi odnašajo smeti. Ker pa mora biti odnašanje pravično, posamezna odnašanja zapisujejo v mrežo, ki je velikosti 5 x 6. Tako vsak član, ki je odnesel smeti, vpiše začetnico svojega imena v prsto polje. Torej Miha zapiše »M«, Anja zapiše »A« in Pišek »P«. Ko pa se mreža dokončno zapolni, pa bi radi hitro prešteli, kolikokrat je kdo odnesel smeti. Tako bodo izvedeli, kdo še ni bil dovolj priden!

Pomagaj Pišku ustvariti program, ki v prazno vrsto v mrežo izpiše ustrezno število odnašanj smeti. Najprej naj izpiše, kolikokrat je odnesel smeti Miha, potem to stori še za Anjo in na koncu še za Piška.

Opomba. Črke, ki so zapisane na rdečih poljih, služijo kot legenda za izpis in niso podatki, ki jih je treba prebrati.



	M	A	A	P	M	P	
	A	P	P	M	A	P	
	M	A	M	P	A	P	
	M	A	M	P	M	A	
	A	A	M	A	A	P	
	0	0	0				
	M	A	P				

[Povezava do naloge](#)

**Ideja reševanja**



Tovrstne naloge se lahko lotimo na več načinov, saj pri reševanju lahko uporabimo spremenljivke, tabele ali pa tudi slovarje.

Osnovna ideja je enaka. Pregledati bomo morali vsa polja mreže ter sproti šteti, kolikokrat je posameznik odnesel smeti. ZA štetje pa lahko uporabimo bodisi spremenljivke, bodisi slovarje ali pa tabele.

V nadaljevanju je najprej podana rešitev z uporabo spremenljivk.

Na začetku programa ustvarimo tri spremenljivke `m`, `a` in `p`, ki jim kot začetno vrednost nastavimo 0. Kot smo napovedali, jih bomo uporabili za štetje odnašanj smeti vsakega od naših treh junakov. Ker imamo opraviti s petimi vrsticami, ki imajo po 6 polj, bomo uporabili gnezdeno zanko. Zunanja se bo ponovila 5x (šla bo po vrsticah). V notranjosti bomo šli najprej desno in pri tem šteli odnašanja. V ta namen si bomo pomagali s pogojem `črka polja == »X«`, kjer je `X` seveda `M` ali `A` ali `P`. Nato pa se bomo še vrnili nazaj na začetek vrstice, ter šli še vrstico dol.

Po koncu dvojne zanke smo ravno v vrstici, kjer moramo zapisati rezultate. Premaknemo se desno, ter pri `nastavi stevilko polja na` uporabimo najprej `m`, po premiku v desno še `a` in potem še `p`.

Lepša je rešitev z uporabo slovarja. Z njegovo pomočjo se znebimo vseh treh pogojnih stavkov. Uporabili bomo slovar, ki bo kot ključ imel črke `»M«`, `»A«` in `»P«`, pripadajoče vrednosti pa bodo števila odnašanj.

Celotna koda ostane enaka, le da na začetku namesto treh spremenljivk ustvarimo slovar s tremi ključi in pripadajočimi vrednostmi 0. Nato pa namesto pogojnih stavkov uporabimo delček `nastavi vrednost ključa ...`, kjer kot ključ uporabimo kar `črka polja`. Ostale podrobnosti pa so razvidne iz rešitve na povezavi spodaj.

[Povezava do rešitev](#)

## DRUŽABNA IGRA

## SŠ NAPREDNI

Tina je poizkušala napisati program, ki bi ji pomagal pri družabni igri. Vsakič, ko je na vrsti, mora povedati novo pozitivno celo število, za katero velja, da je vsota njegovih pravih deliteljev večja od števila samega.

Število 12 je prvi primer takega števila. Njegovi pravi delitelji so 1, 2, 3, 4 in 6, kar je skupaj 16. Zapisala je program, ki bi ji povedal, ali vneseno število ustreza zgornjim pogojem ali ne, vendar je naredila nekaj napak. Popravi njen program.

Input: 12	Output:
--------------	---------

### Povezava do naloge

### Ideja reševanja

Do prve napake bomo prišli kar hitro. Očitno je naloga funkcije **vsota** ta, da izračuna vsoto pravih deliteljev števila. In če je to res, moramo DA izpisati takrat, če je prebrano število manjše od vrednosti, ki jo vrne klic te funkcije. V Tininem programu moramo torej pogoj `stevilo == vsota` spremeniti v `stevilo < vsota`. Mimogrede – Tini naj še prišepnemo, da ni najbolj pametno spremenljivki dati isto ime kot funkcija. Sicer formalno ni nič narobe, a različne stvari naj imajo različna imena.

In že s to spremembo uspešno prestanemo prvi test. A drugega žal ne. Torej bo še nekaj narobe! Za podatek 12 smo torej pravilno izračunali, da ima vsoto večjo kot 12, za podatek 14 pa ne. Koliko pa sploh dobimo? Programu dodajmo na konec še izpiši `vsota`, da bomo videli, koliko naračuna funkcija `vsota`. In res, za vhodni podatek 14 dobimo 91, čeprav so pravi delitelji le 1, 2 in 7 (torej je njihova vsota 10). In če izvedemo ponovno še `Test1` (ki je zdaj sicer napačen, ker pač vrne preveč vrstic), vidimo, da smo kot vsoto deliteljev izračunali 66, kar je tudi

narobe, saj bi morali naračunati 16. Torej je to, da smo prej dobili pravilen odgovor le naključje! Ta nas uči, da če program vrne pravilen rezultat, to še ne pomeni, da je res pravilen. S preizkušanjem na primerih lahko le ugotovimo, da program ni pravilen. Škoda...

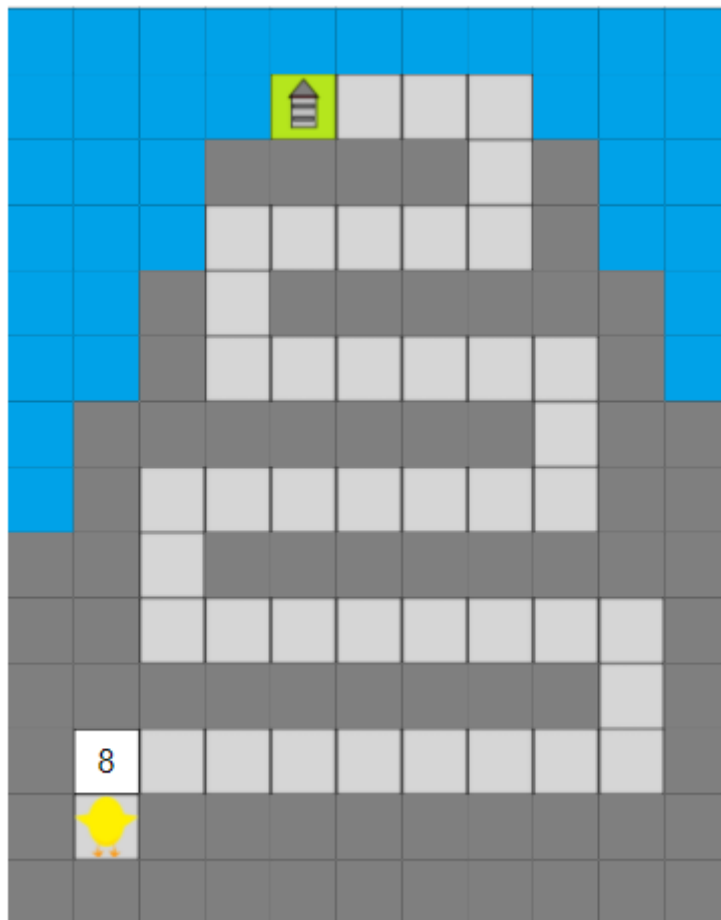
No, vrnimo se k iskanju napak. Očitno funkcija **vsota** vrne prevelike vsote. In če pogledamo kodo, vidimo, da pravzaprav izračuna kar  $1 + 2 + 3 + \dots + n - 1$ , torej sešteje vse možne delitelje, ne le tiste, ki to res so. In ta premislek nas vodi k ustreznemu popravku. Dodati moramo še pogojni stavek, kjer bomo **i** prišteli le, če je delitelj števila. Kdaj pa **i** deli stevilo – takrat, kadar je ostanek pri deljenju **stevilo** z **i** enak 0. Zdaj pa je popravek preprost in Tinin program deluje pravilno. Seveda ne smemo pozabiti odstraniti zadnjega stavka z izpisom vsote, ki smo ga prej dodali.

[Povezava do rešitev](#)

## TRIGLAV

## SŠ NAPREDNI

Pišek bi rad na vrhu Triglava opazoval sončni vzhod. Ker bo hodil v temi, potrebuje natančna in ne predolga navodila v obliki programa. Ker je poti na Triglav več, moraš pripraviti program tako, da bo lahko prišel po vsaki od opisanih treh poti (testni primeri) do vrha. V pomoč je dan podatek o številu korakov, potrebnih, da Pišek prehodi prvi vodoravni del. Pišek je že pravilno napisal večji del programa, razen funkcije **potuj**, ki jo napiši ti!



### Povezava do naloge

### Ideja reševanja

Navodilo naloge pravi, da je potrebno napisati oz. dopolniti zgolj funkcijo **potuj**. Vendar predno se lotimo pisanje rešitve za funkcijo **potuj**, si pogledajmo preostali dve funkciji.

Funkcija `obrnj_se` določa, ali se mora Pišek obrniti v levo ali desno, glede na vrednost parametra `tsmer`.

Funkcija `zamenjaj_smer` pa je funkcija, ki se izvaja na robovih oz. zavojih. Npr. če Pišek prihaja z desne smeri (se premika v desno), se mora na koncu obrniti v levo, iti dva koraka naprej in nato še enkrat v levo (zavoj v levo); če pa prihaja z leve smeri (se premika v levo) pa ravno obratno oz. mora napraviti zavoj v desno. Na koncu pa tudi opazimo, da funkcija vrne »obratno« vrednost trenutne smeri, kar je super, saj se mora Pišek premikati v drugi smeri kot se je sedaj.

Lotimo se sedaj zapisa rešitve.

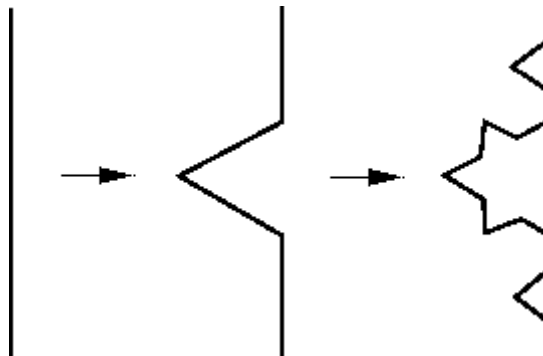
V funkciji `potuj` je potrebno sprva določiti za koliko korakov se mora Pišek premakniti naprej; nato pa mora napraviti zavoj, kar lahko storimo s funkcijo `zamenjaj_smer` (vrednost, ki jo funkcija `zamenjaj_smer` vrne je potrebno shraniti v spremenljivko `prava_smer`). Ko pa to storimo je pa potrebno znova klicati funkcijo `potuj`, le da mora Pišek, tokrat, napraviti en korak manj, itn. Torej funkcija `potuj` je pravzaprav rekurzivna funkcija (oz. uporabljamo »rekurzivni postopek«); pri rekurzivni funkciji pa je potrebno vselej določiti »robni pogoj«, ta je v našem primeru takrat, ko mora Pišek napraviti manj kot štiri korake. »Juhu«, pa smo na vrhu – Triglava!

[Povezava do rešitev](#)

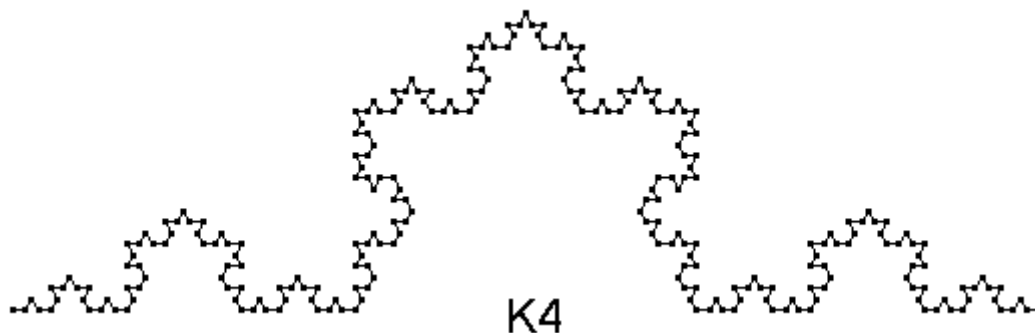
## KOCHOVA ČRTA

SŠ NAPREDNI

Kochovo črto stopnje 2 dobimo z naslednjim postopkom: Ravno črto (Kochovo črto stopnje 0) dolžine 81 razdelimo na tretjine in srednji del nadomestimo s stranicama enakostraničnega trikotnika (dobimo Kochovo črto stopnje 1). Nato enak postopek izvedemo na vseh 4 ravnih črtah, ki sestavljajo Kochovo črto stopnje 1.



Kochova črta stopnje 4 je torej:



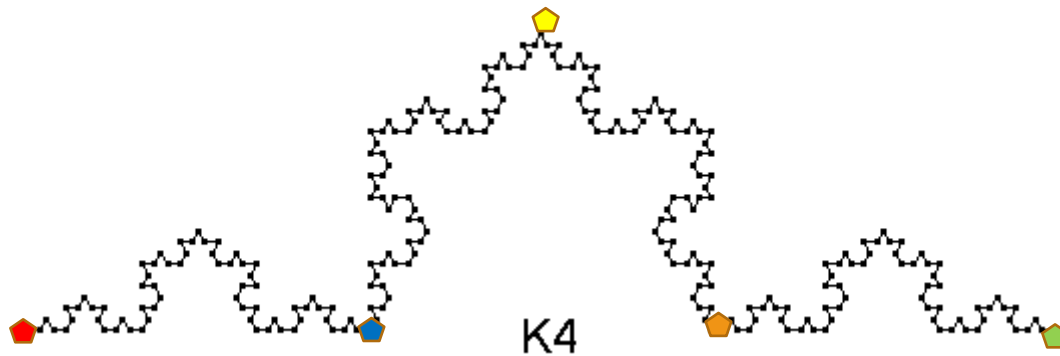
S pomočjo želje grafike nariši Kochovo črto na predvideno mesto. Ustrezen program je že napisan, ti moraš sestaviti le funkcijo **koch**. Program iz vsakega testa prebere, za katero stopnjo Kochove črte gre.

[Povezava do naloge](#)

### Ideja reševanja

Če pogledamo K4, lahko opazimo, da je sestavljena iz 4 Kochovih črt, ki pa so krajše (tretjino dolžine) in enkrat manj »zgubane« (torej stopnje 3).

Na sliki spodaj so z barvnimi petkotniki označeni začetki oziroma konci teh črt:



Enako velja v splošnem. Kochova črta stopnje  $n$  in dolžine  $d$ , je sestavljena iz štirih Kochovih črt stopnje  $n - 1$  in dolžine  $d/3$ . Odločimo se, da bo želva, s katero rišemo, na koncu risanja Kochove črte gledala v isto smer kot na začetku, le da bo za  $d$  enot naprej od začetka. Želvo moramo torej iz rdeče točke prestaviti v zeleno, vmes pa se mora gibati po Kochovi črti:

- Iz rdeče točke gre do modre.
- Obrne se v levo za 60 stopinj.
- Iz modre točke gre do rumene (smer je prava!).
- Obrne se v desno za 120 stopinj.
- Iz rumene točke gre do oranžne,
- kjer se spet obrne v levo za 60 stopinj in
- gre iz oranžne do zelene.
- Tam ostane in je že obrnjena tako, kot smo napovedali (gleda v isto smer kot na začetku)

Kako pa se želva giblje med, denimo, rumeno in oranžno točko? Seveda po Kochovi črti, ki je ene stopnje manj in pri tem se mora premakniti za  $d/3$ .

Za opis Kochove črte stopnje  $n$  smo torej spet uporabili Kochove črte, le z drugimi podatki (nižjo stopnjo in krajšo dolžino). Uporabili smo torej rekurzivni postopek.

`koch(n, d):`

- `koch(n - 1, d/3)`
- `zavij za 60 stopinj levo`
- `koch(n - 1, d/3)`

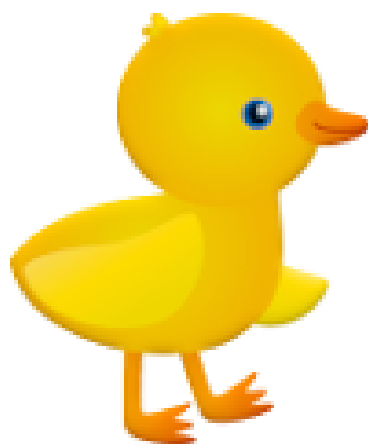
- zavij za 120 stopinj desno
- $\text{koch}(n - 1, d/3)$
- zavij za 60 stopinj levo
- $\text{koch}(n - 1, d/3)$

In to velja za poljubno Kochovo črto, razen za? Razen za Kochovo črto stopnje 0, ki še ni nič nagubana. Je lepa ravna črta.

Funkcija v nalogi ima torej že na ustreznem mestu pogojni stavek, ti moraš le še dopolniti del pri sicer, kjer vstaviš ukaze, ki storijo to, kar smo opisali zgoraj.

[Povezava do rešitev](#)



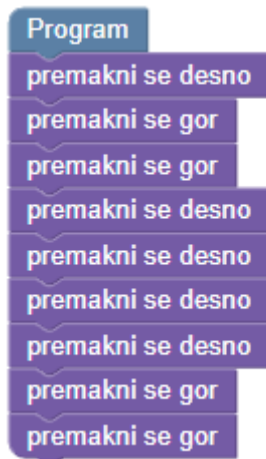


**Rešitve**

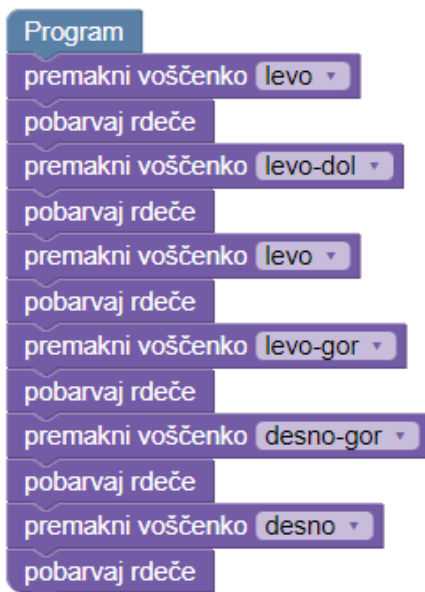
## Rešitve

OŠ 4. – 6. r. ZAČETNIKI

### Gusarji iščejo zaklad



### Neža Barva



## Čebelica Rozi oprahuje

### Rešitev 1

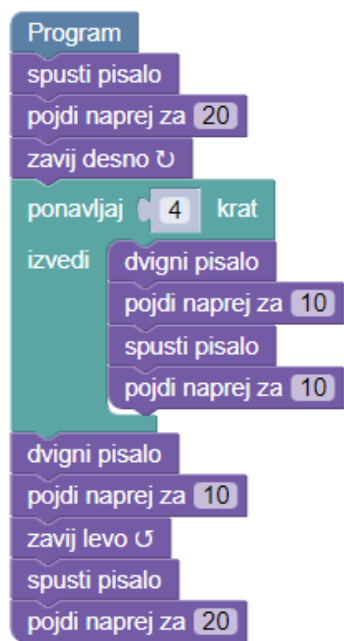


### Rešitev 2

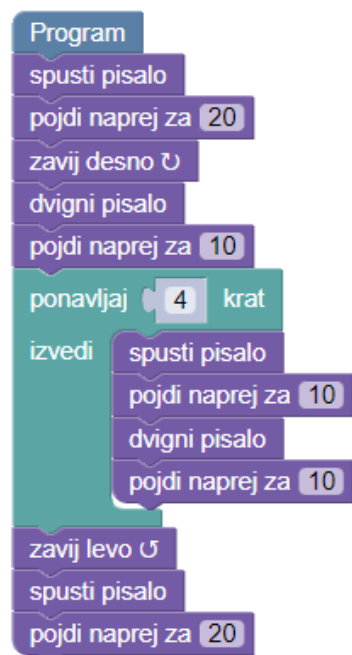


## Pišek riše

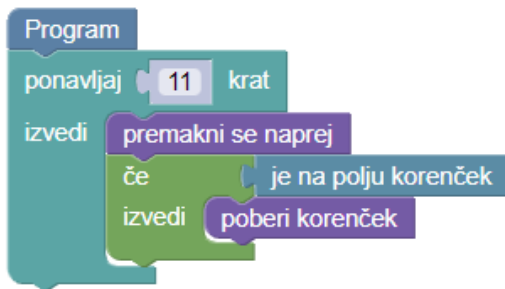
### Rešitev 1



### Rešitev 2



### Zajček pobira korenčke



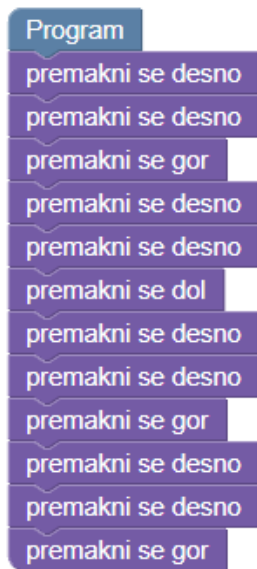
### Vrt sončnic



## Rešitve

### OŠ 4. – 6. r. NAPREDNI

#### Peščeni planet



#### Štirje zakladi



### Avto na daljinca



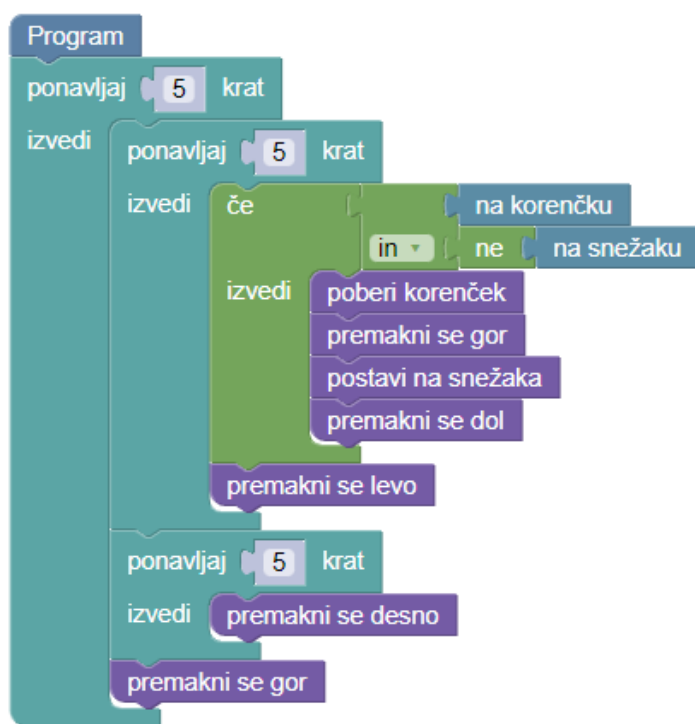
### Ježek se pripravlja na zimo



### Regratova lučka



### Snežaki in korenčki

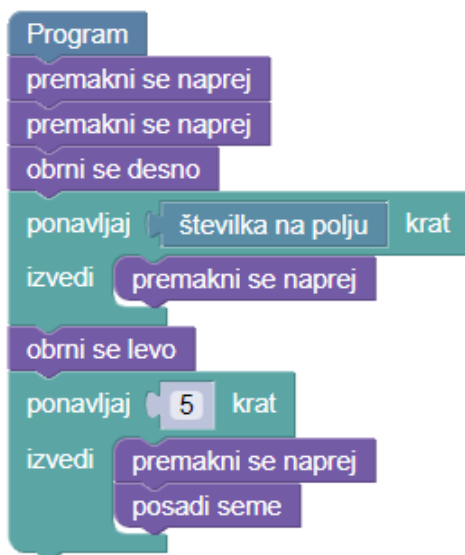




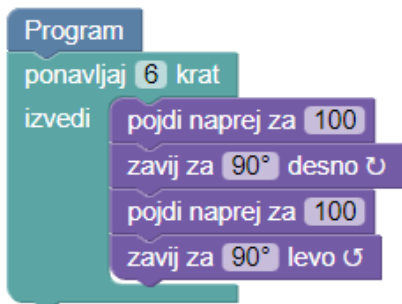
## Rešitve

OŠ 7. – 9. r. ZAČETNIKI

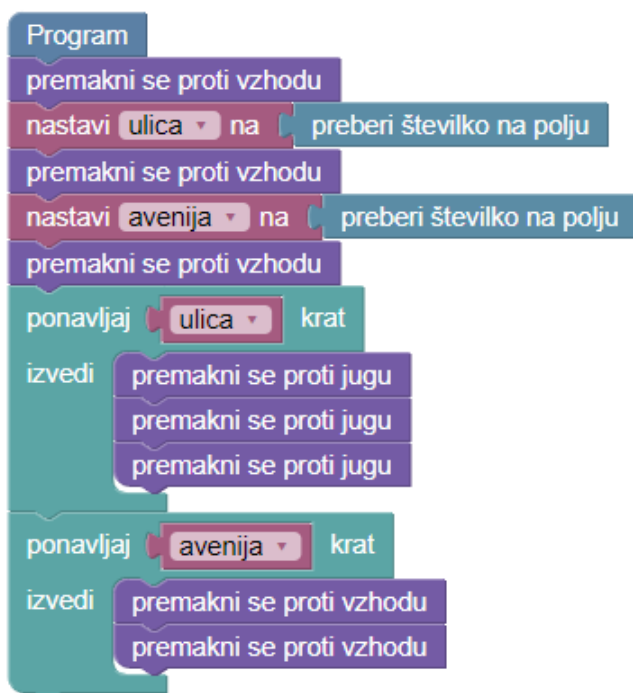
### Sejalni robot



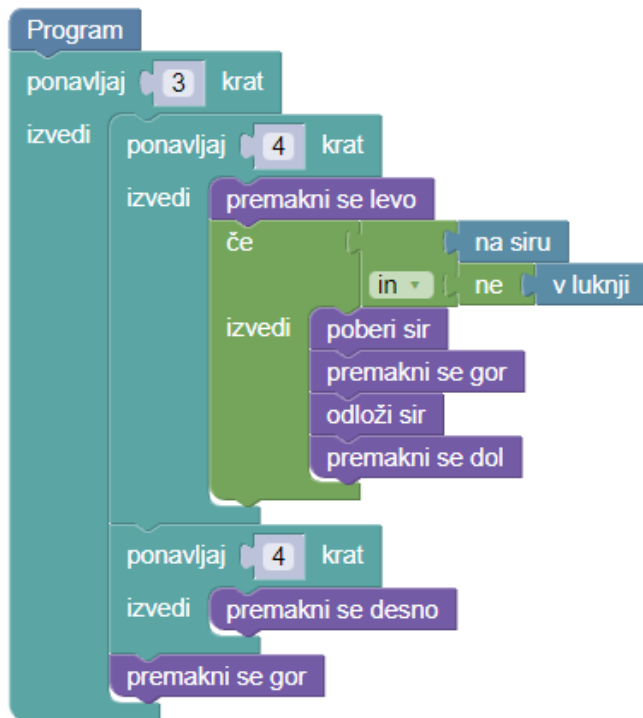
### Pišek gleda tekmo



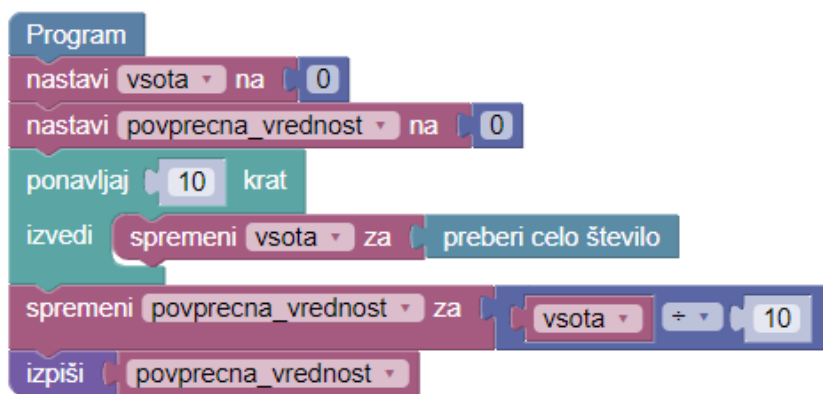
### Jure v New Yorku



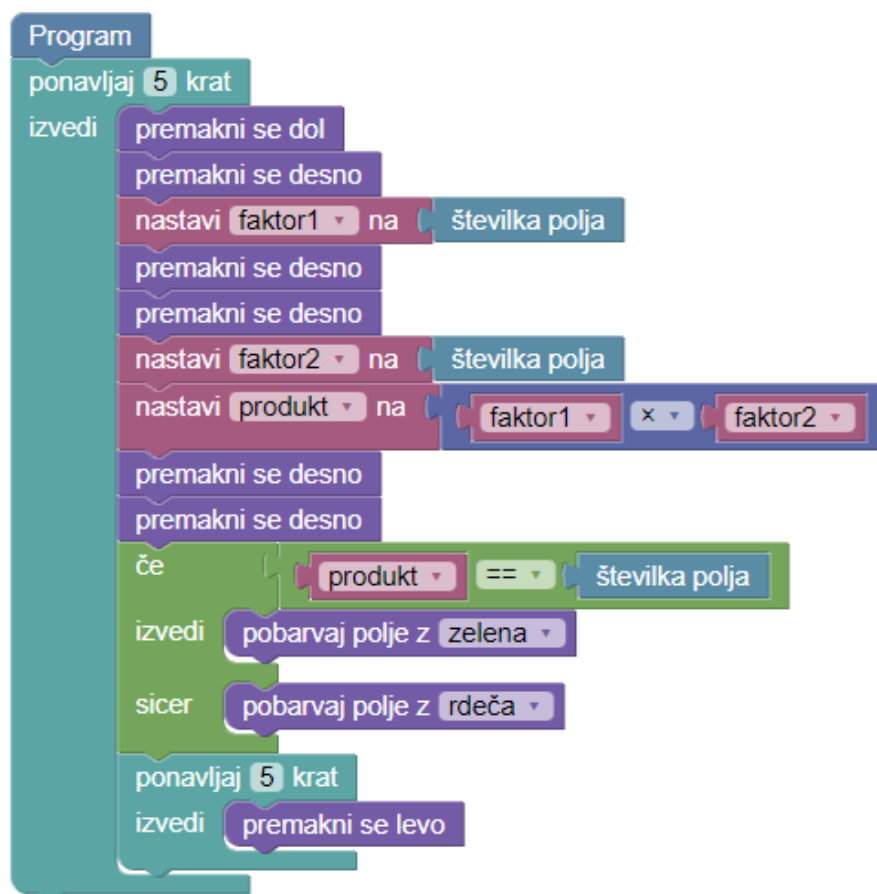
### Miška zbira sir



### Povprečna dolžina poti v šolo



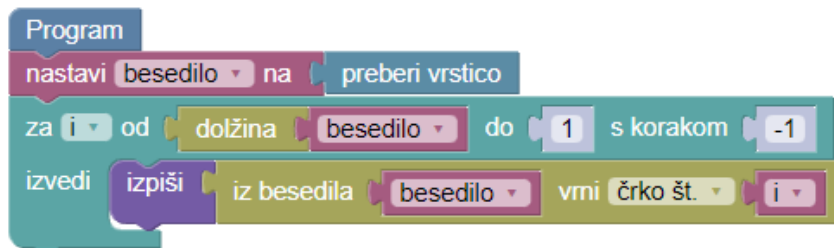
### Pišek vadi poštevanke



## Rešitve

OŠ 7. – 9. r. NAPREDNI

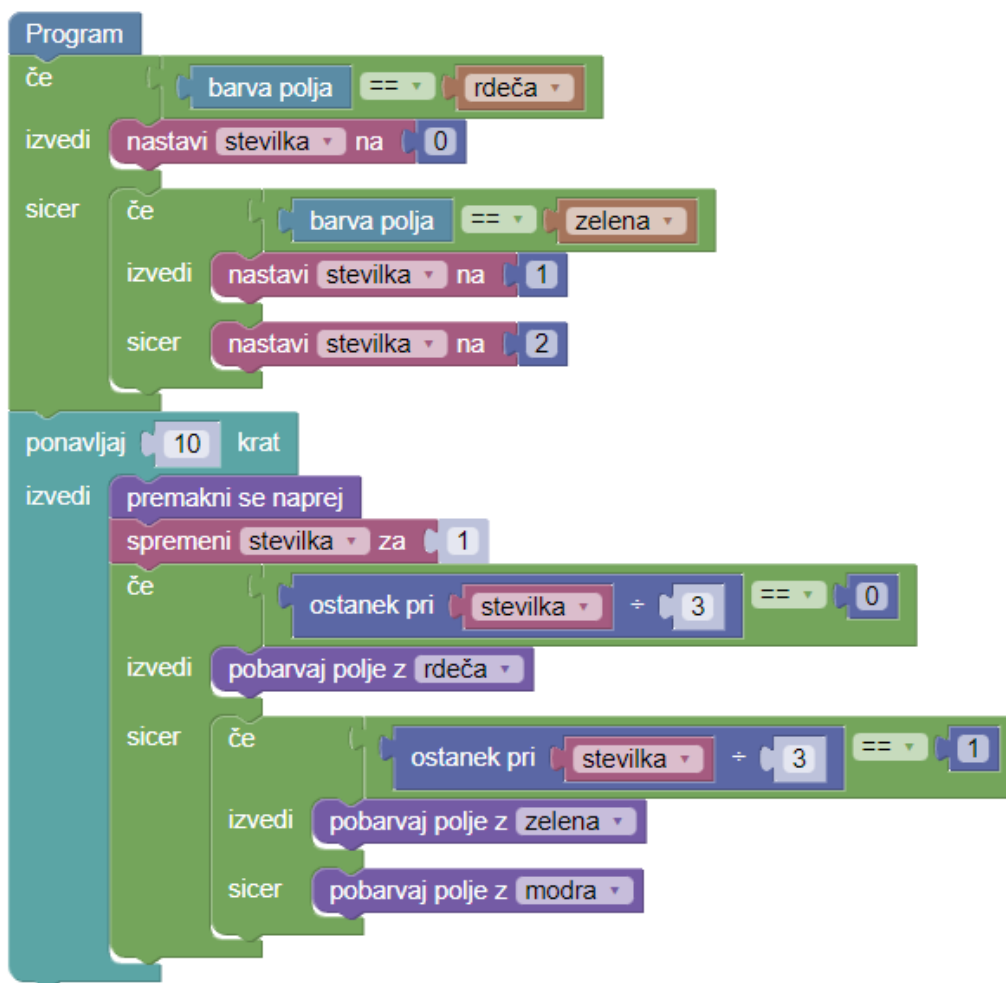
### Šifriranje sporočil



### Veverička in ozimnica

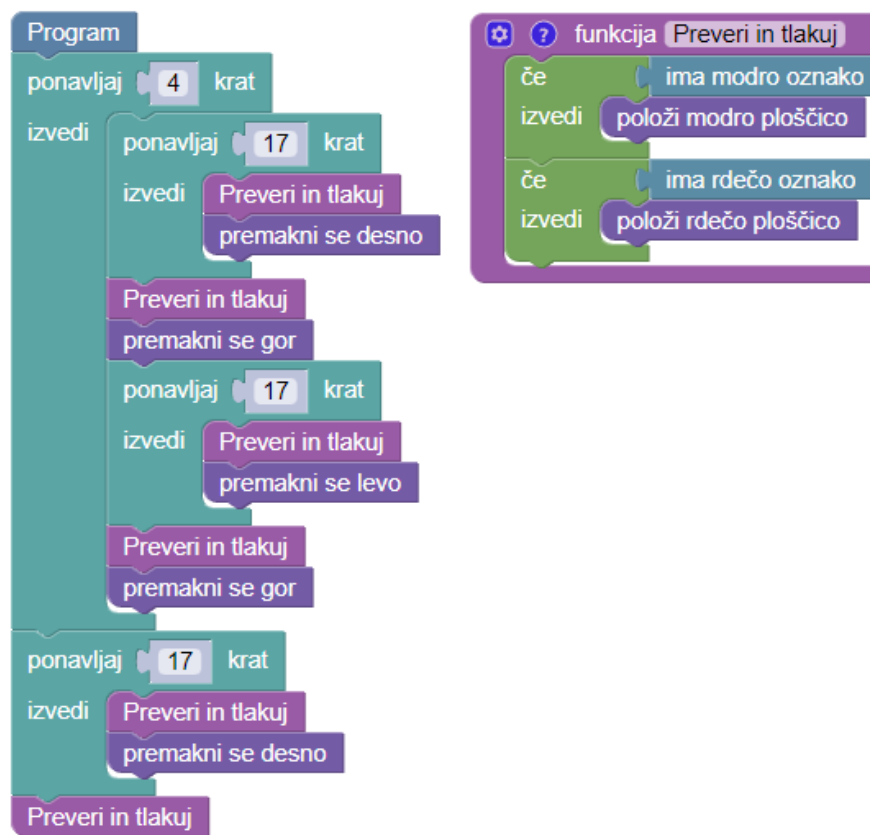


### Barvanje ograje

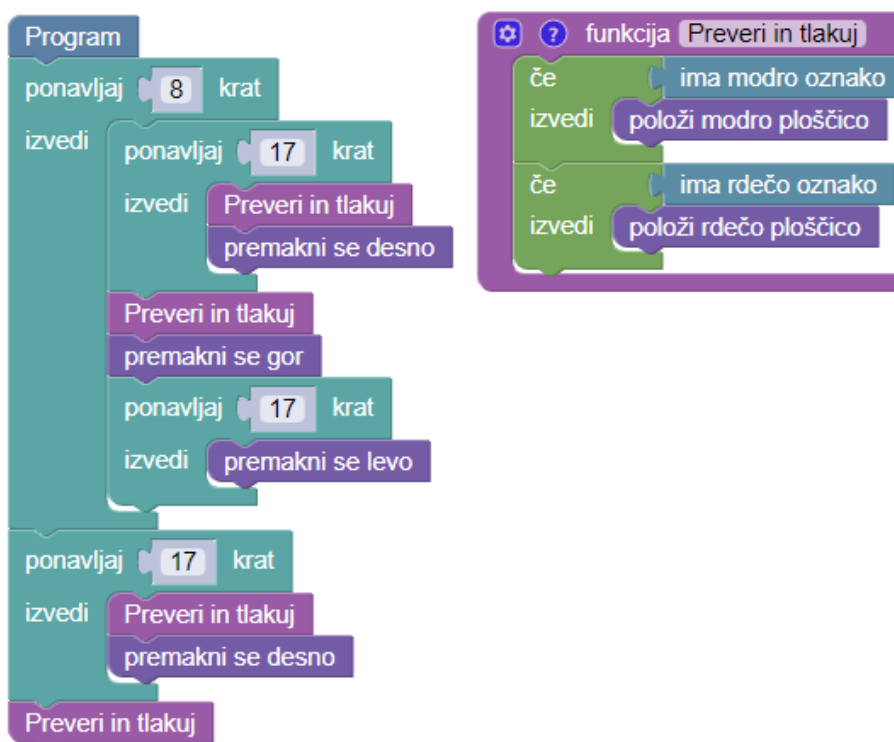


## Zmajček in tlakovanje

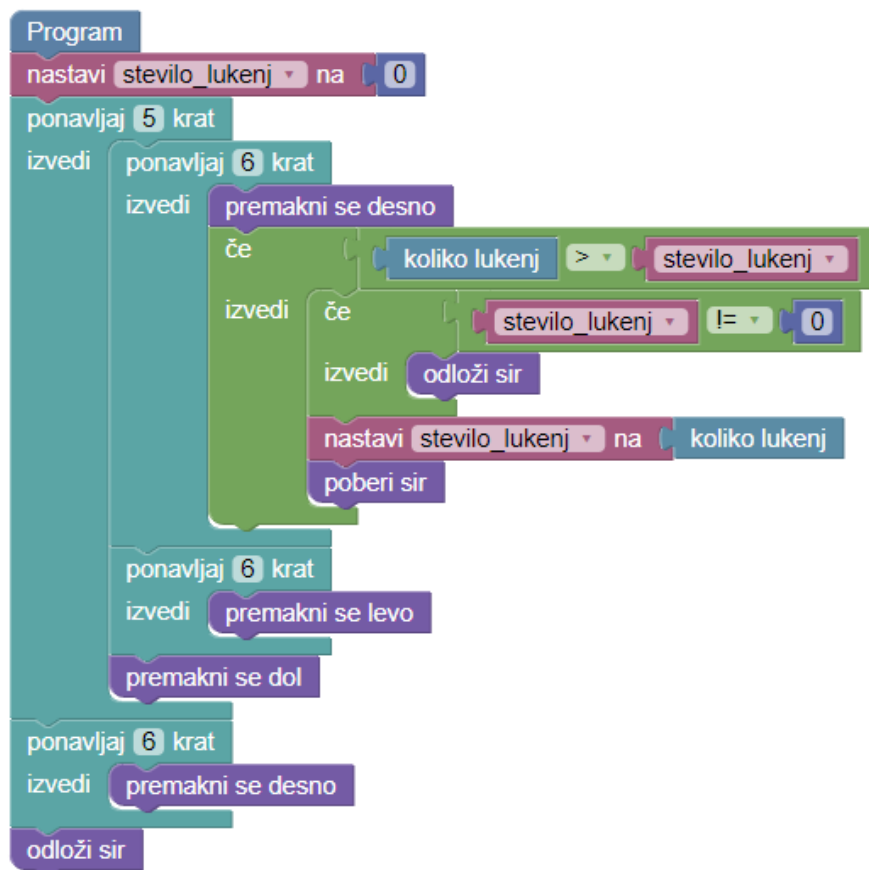
### Rešitev 1



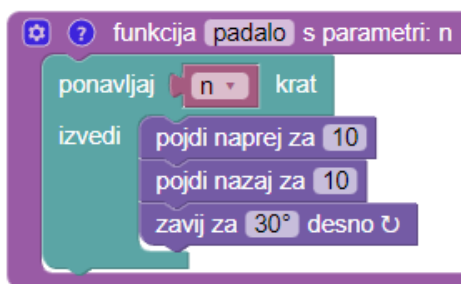
### Rešitev 2



### Miška išče sir



## Regratova lučka





## Rešitve

## SŠ ZAČETNIKI

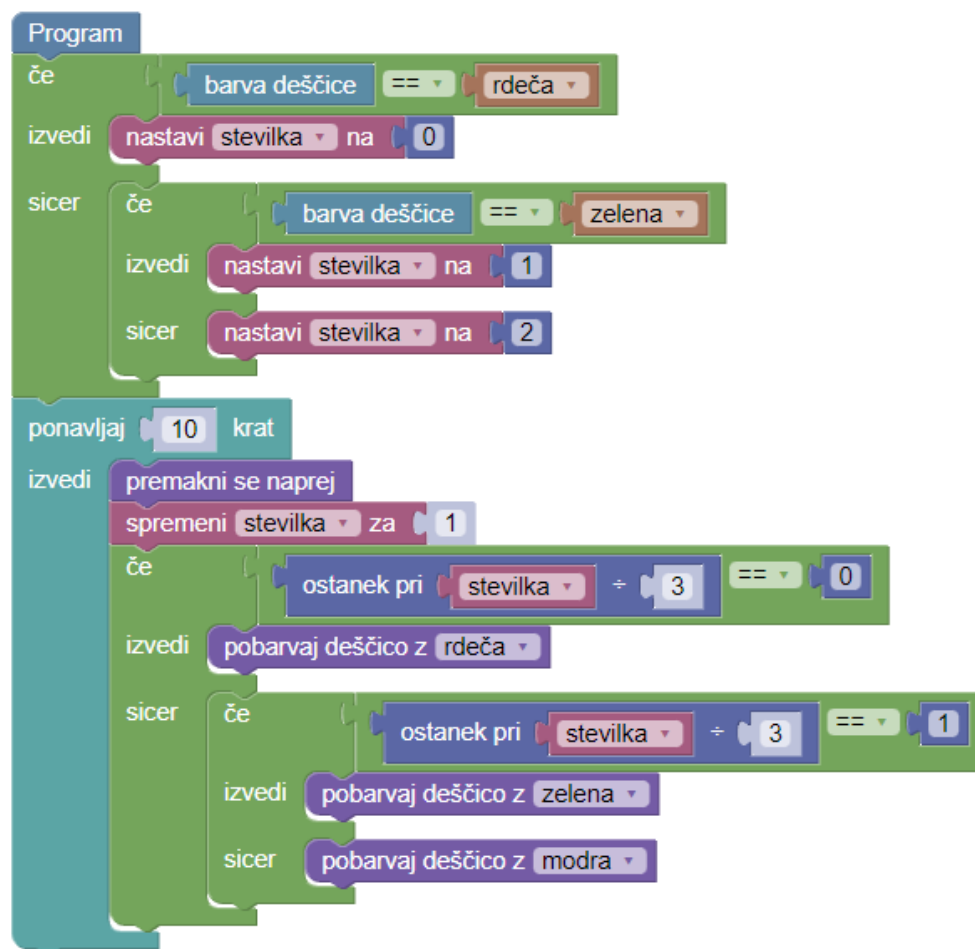
### Robotek Bojan riše



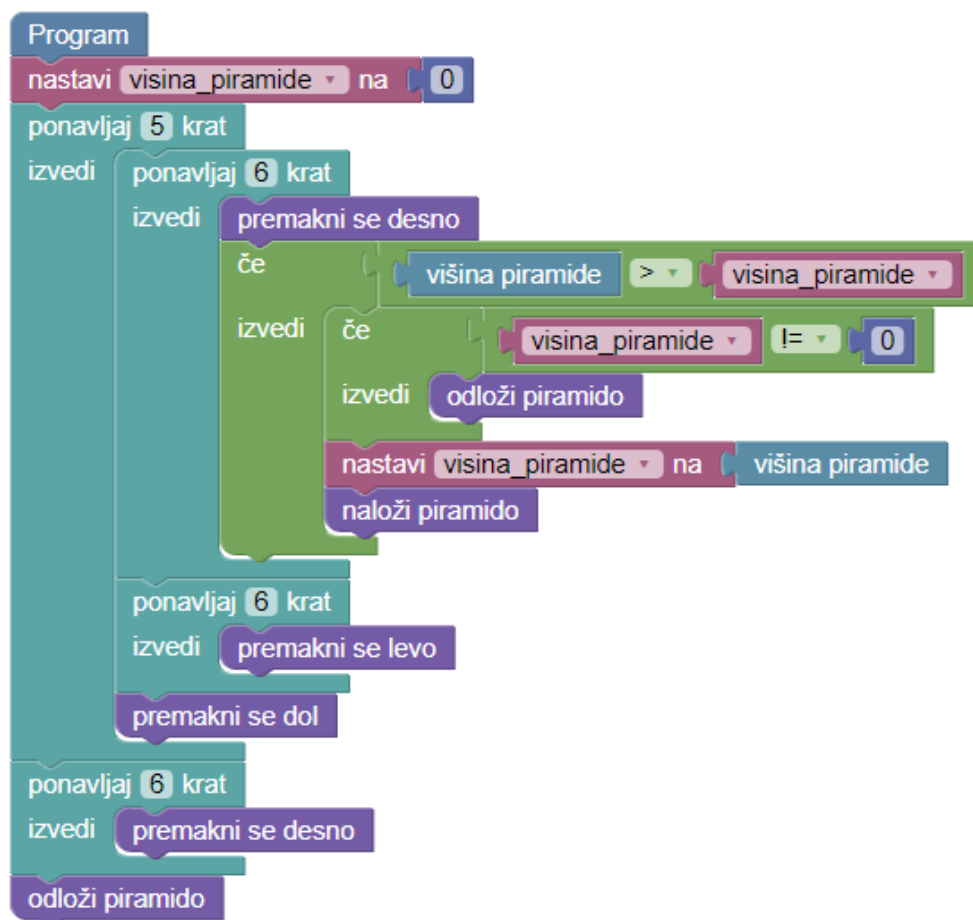
### Iskanje pogrešanega



### Barvanje ograje

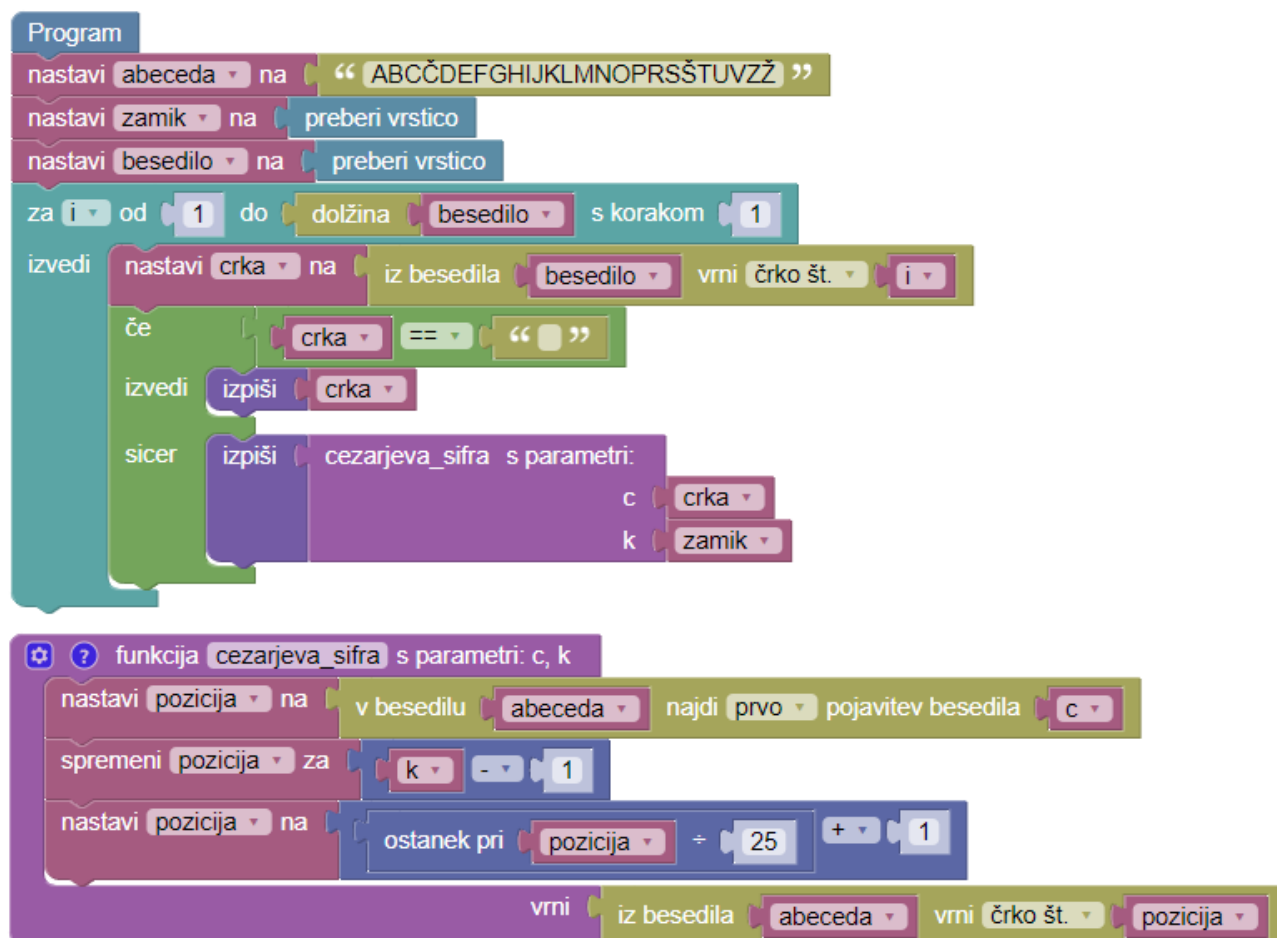


### Slaščičar Stane

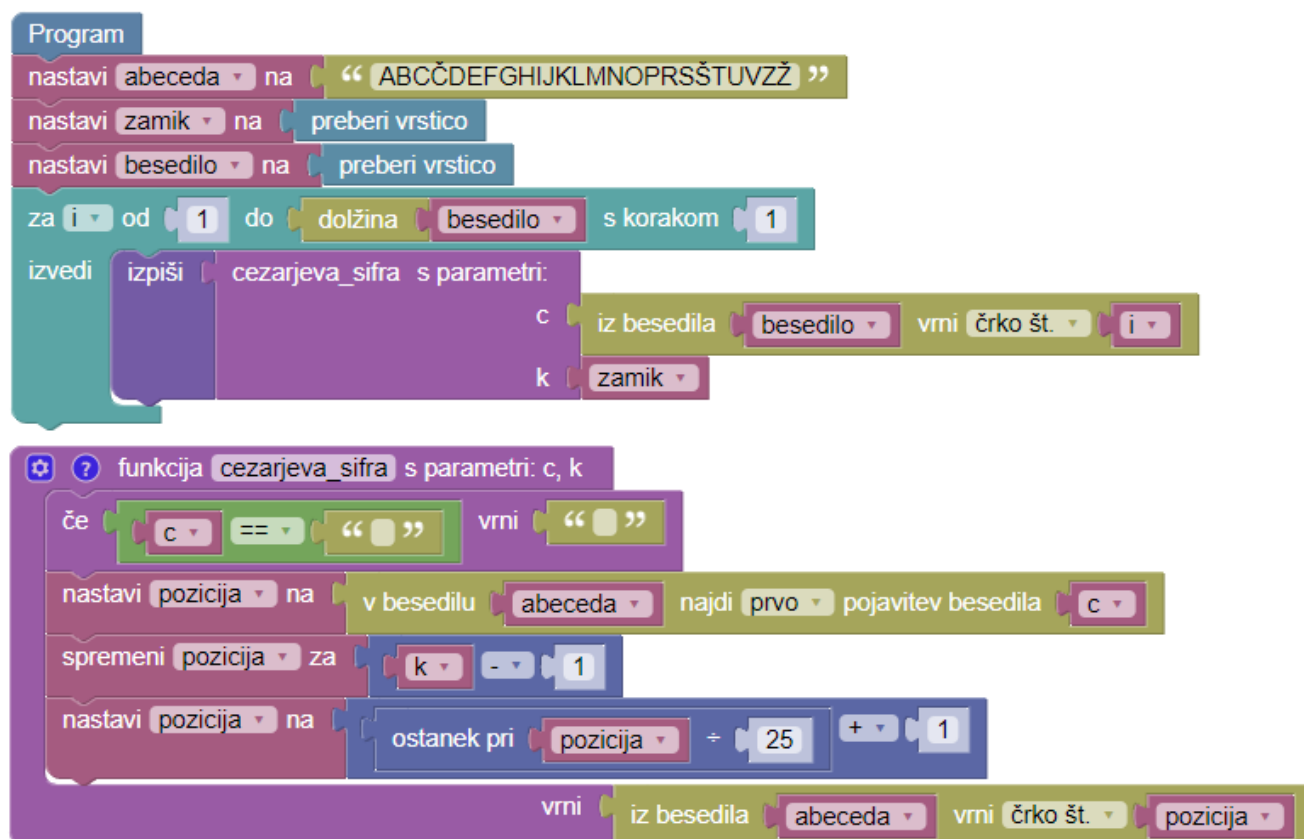


## Cezarjeva šifra

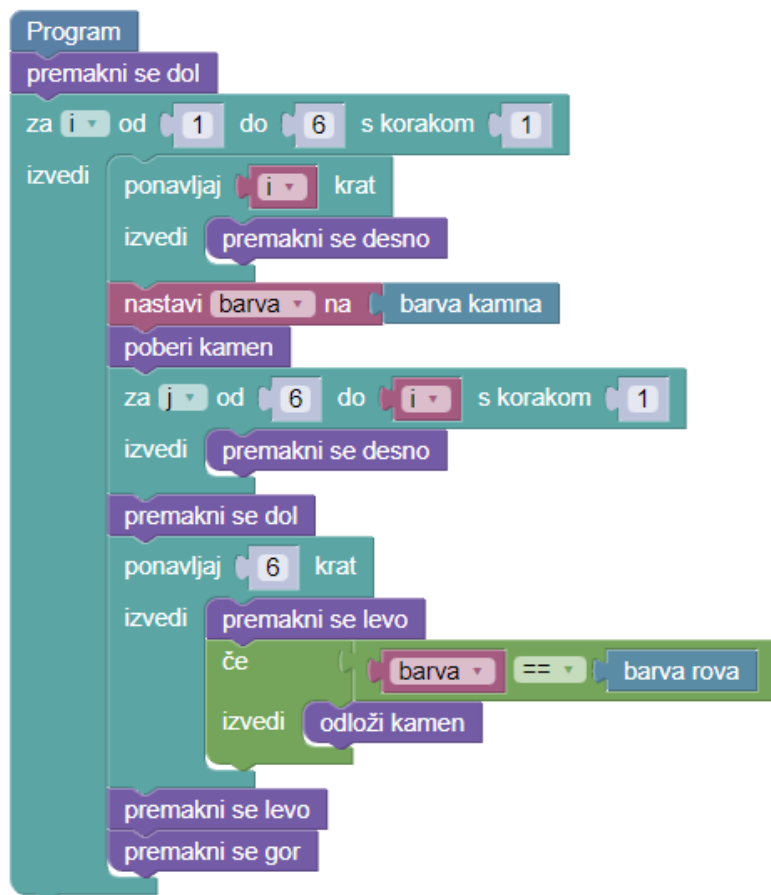
### Rešitev 1



Rešitev 2



### Krtkova družina



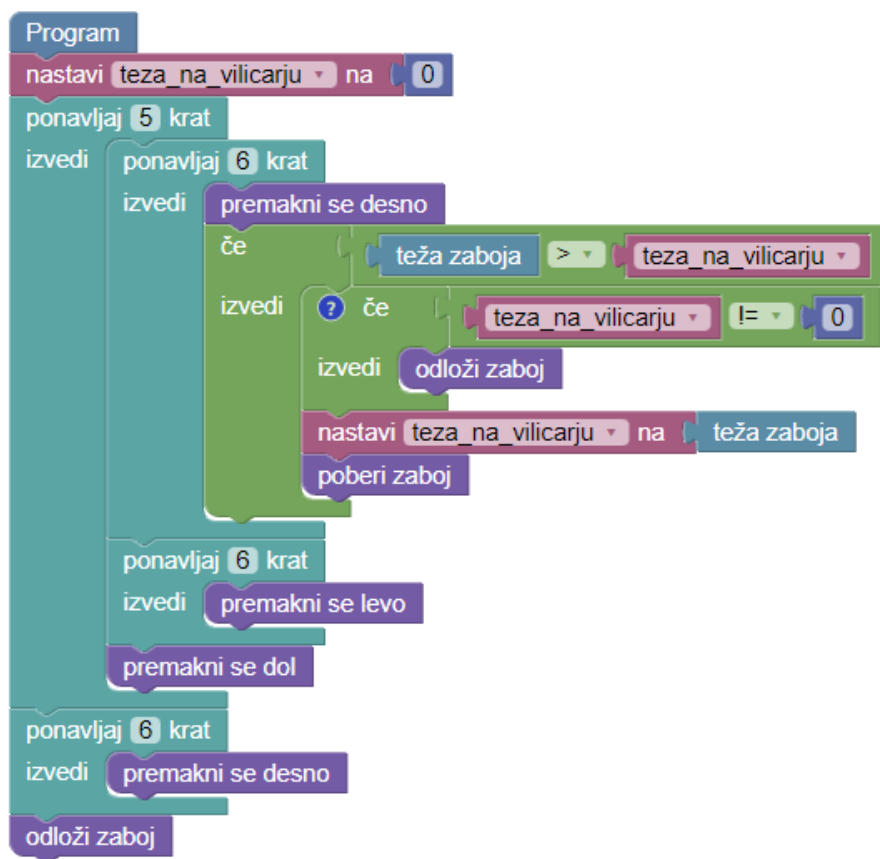
## Rešitve

## SŠ NAPREDNI

### Eksperiment s sodimi števili



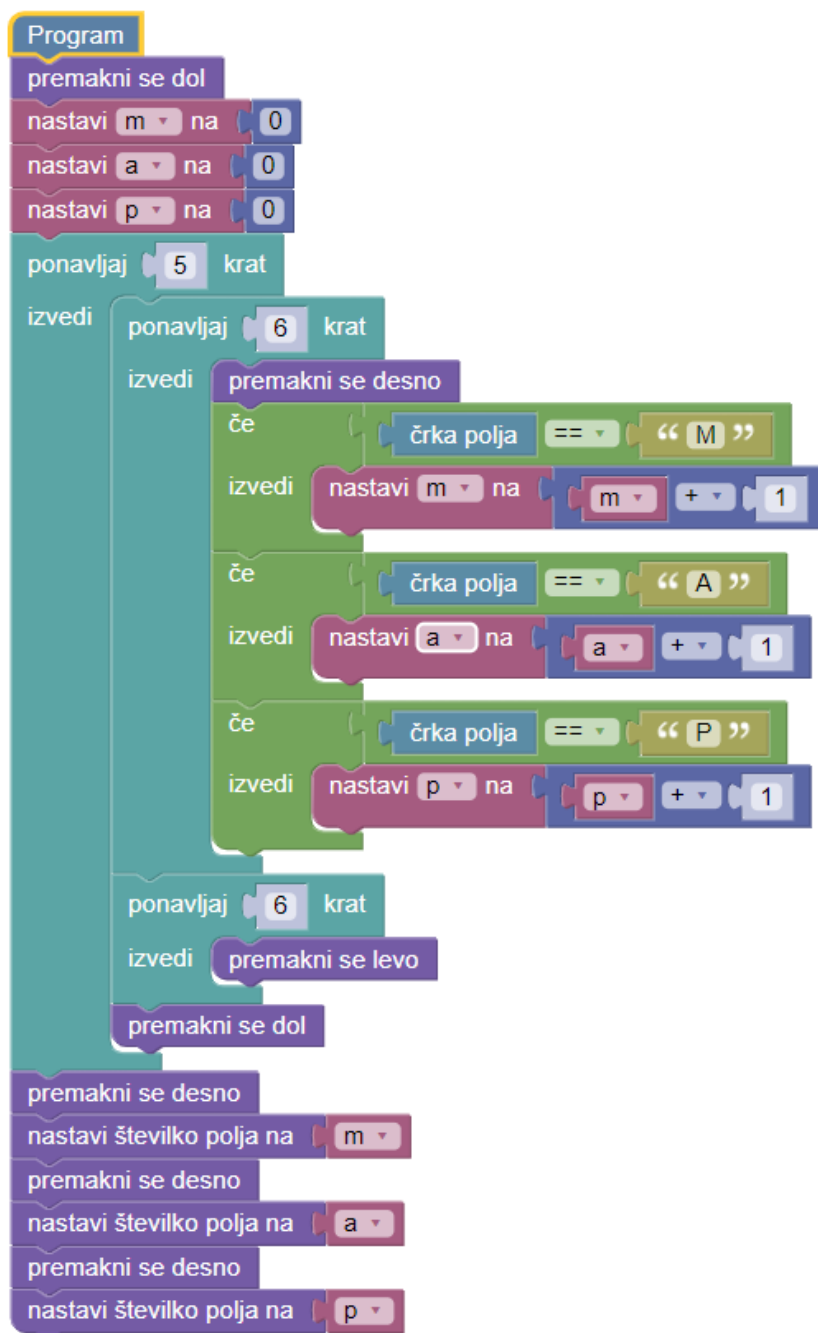
### Najtežji zaboj



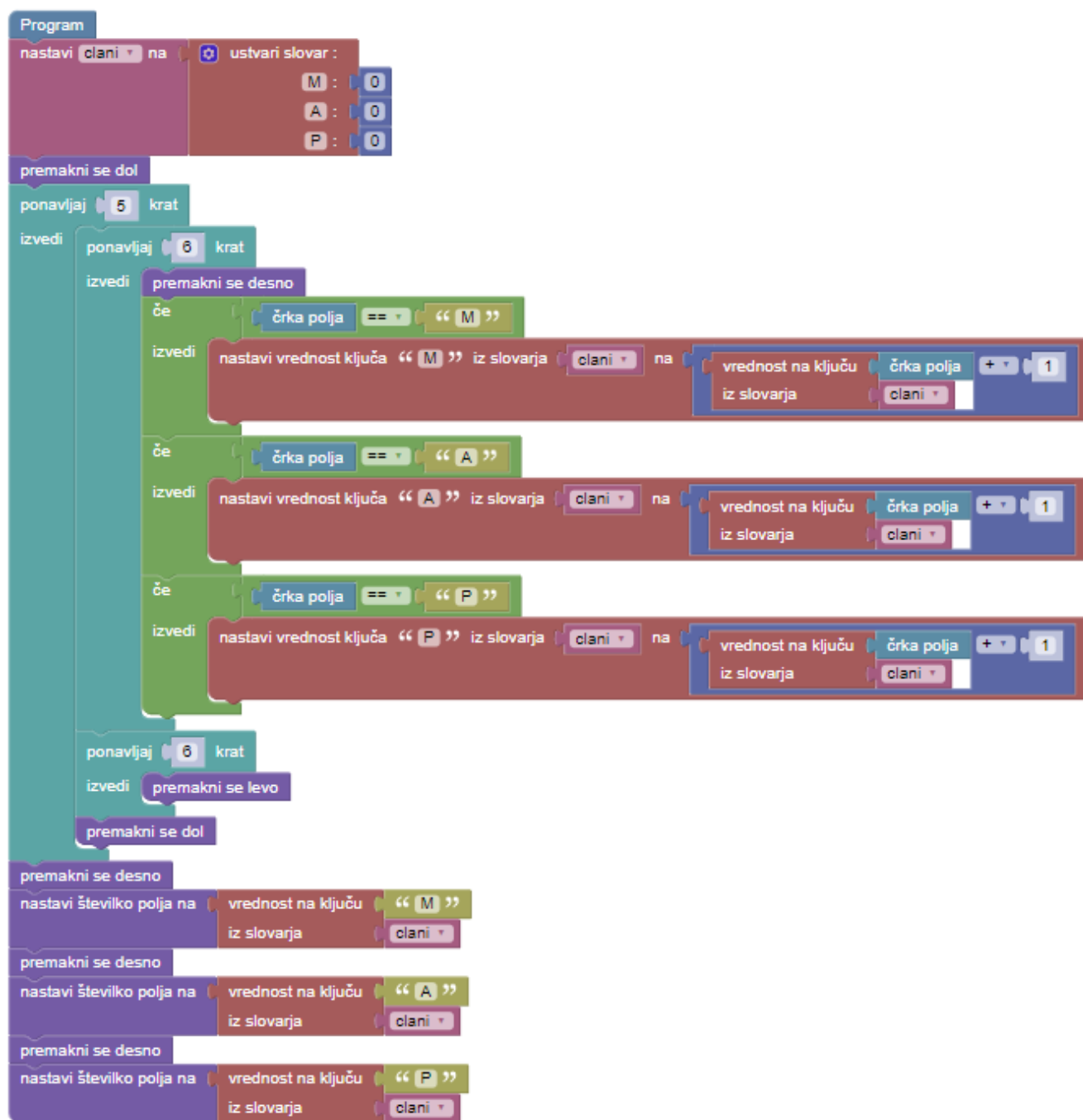


## Odnašanje smeti

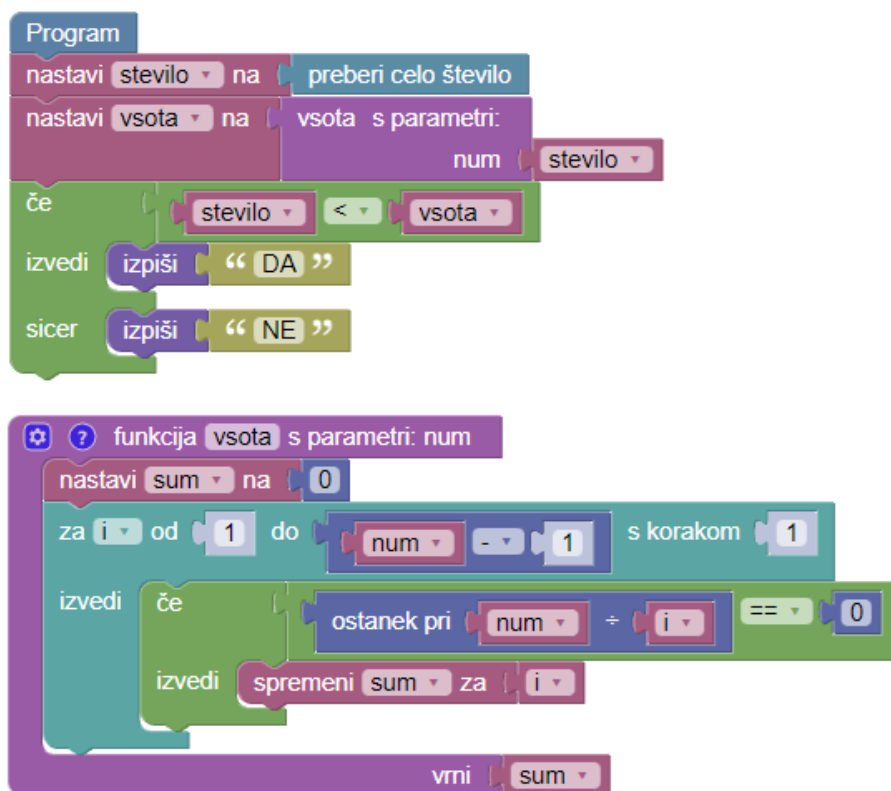
Rešitev (s spremenljivkami)



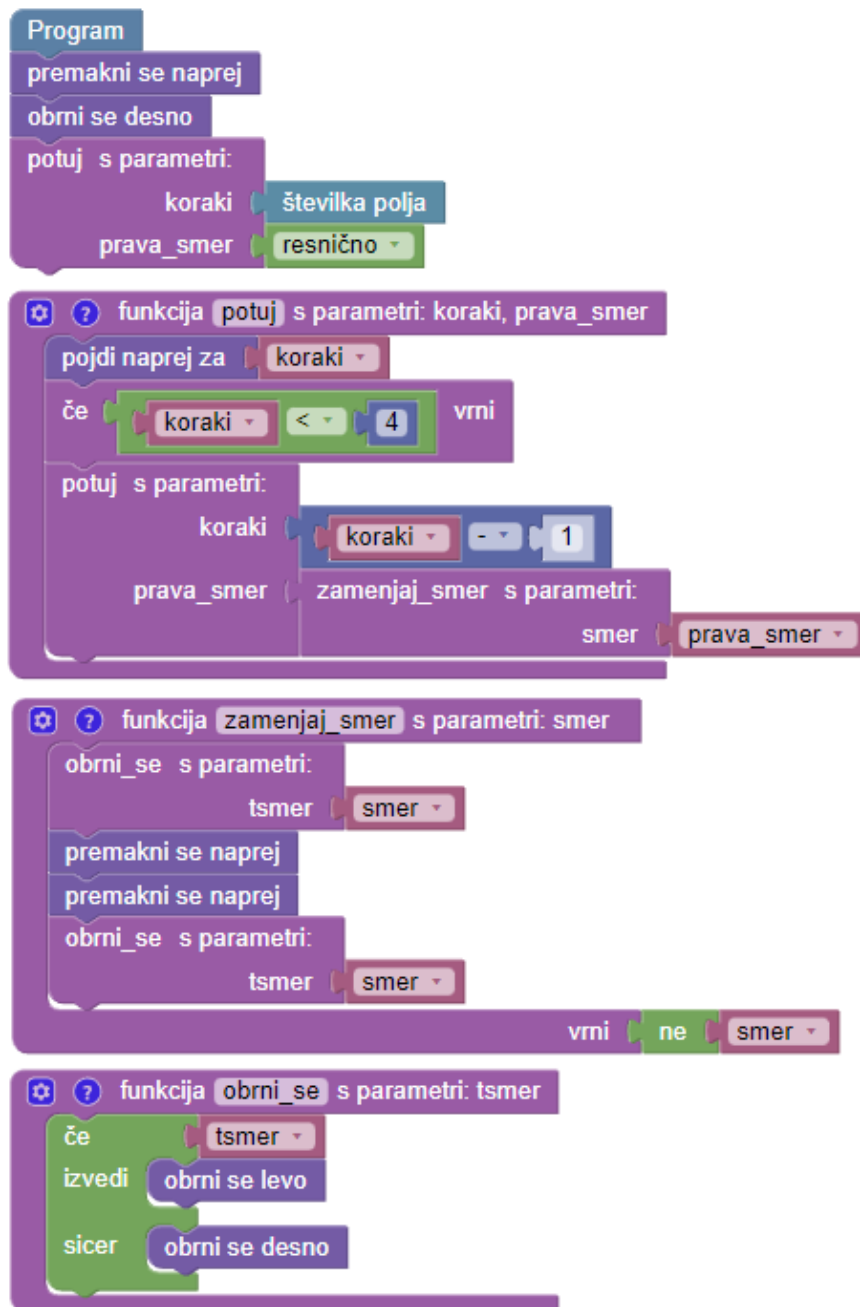
Rešitev (s slovarji)



### Družabna igra



## Triglav



### Kochova črta

