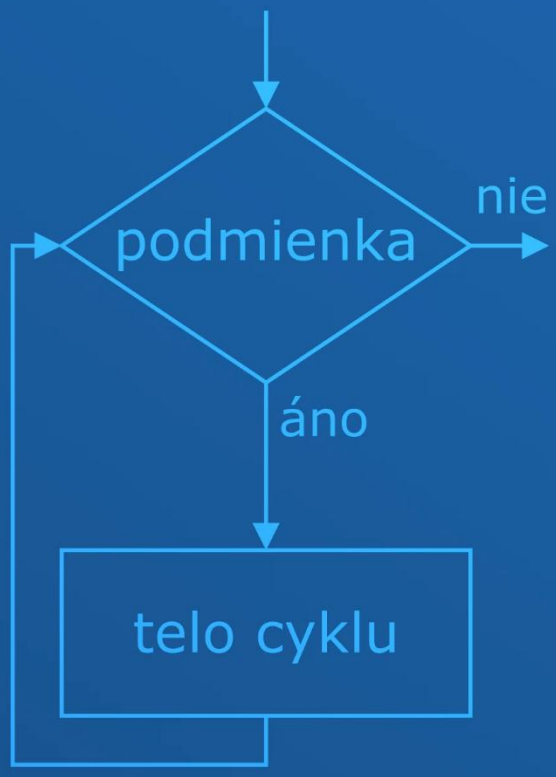


Stoffová
Horváth
Gabaľová

Detské programovacie jazyky

ako úvod do programovania

2025

Detské programovacie jazyky ako úvod do programovania

Autori:

prof. Ing. Veronika Stoffová, CSc.

Mgr. Ing. Roman Horváth, PhD.

PaedDr. Veronika Gabaľová, PhD.

Vydavateľ:

Pedagogická fakulta Trnavskej univerzity v Trnave

Editor:

prof. Ing. Veronika Stoffová, CSc.

Recenzenti:

prof. PaedDr. Alena Hašková, CSc.

doc. PhDr. Miroslav Chráska, Ph.D.

Typografia a zalamovanie:

Mgr. Ing. Roman Horváth, PhD.

© 2025, Autori, © 2025 Pedagogická fakulta Trnavskej univerzity v Trnave

Žiadna časť tejto učebnice nesmie byť reprodukovaná, uložená v systéme vyhľadávania alebo prenášaná v akejkoľvek forme akýmikoľvek prostriedkami, elektronicky, mechanicky, kopírovaním, nahrávaním alebo inak bez predchádzajúceho písomného súhlasu autorov a vydavateľa.

ISBN 978-80-568-0210-6

https://doi.org/10.31262/978-80-568-0210-6/2025

Obsah

Predslov **5**

1 Úvod do algoritmizácie a programovania **7**

1.1 Slovné vyjadrenie algoritmu 9

1.2 Zápis algoritmov vývojovými diagramami 11

1.3 Zápis algoritmov štruktúrogramami – Nassi-Schneidermannovými diagramami 14

1.4 Zápis algoritmov v algoritmickom jazyku 17

1.5 Tvorba algoritmov 20

1.6 Otázky a úlohy 21

2 Detský programovací jazyk Scratch **23**

2.1 Ako používať Scratch on-line 23

2.2 Ako používať Scratch off-line 26

2.3 Prostredie a príkazy Scratchu 29

2.3.1 Hlavná ponuka *29*

2.3.2 Panel príkazov *30*

2.3.3 Skripty, kostýmy a zvuky *33*

2.3.4 Okno výstupu *34*

2.3.5 Časť objektov *35*

2.4 Otázky a úlohy 35

3 Riešenie jednoduchých úloh **37**

3.1 Ovládanie postavičky na obrazovke s použitím vstupu z klávesnice 37

3.2 Cyklus a podmienka v programe 39

3.3 Používanie premenných v programe 42

3.4 Upevňovanie vedomostí a zopakovanie učiva 46

3.5 Otázky a úlohy 47

4 Praktické príklady v programovacom prostredí Scratch **49**

4.1 Poletujúci netopier 49

4.2 Kresliace akvárium s rybičkou 51

4.3 Štyri kvadranty 53

4.4 Praslička 55

4.5 Sierpinského trojuholník 57

4.6 Skákajúca loptička 59

4.7 Ovládacie tlačidlá 62
4.8 Pochytaj hviezdičky! 65
4.9 Otázky a úlohy 67
5 Kapusta, koza, vlk a dievča v prostredí Baltík 68
5.1 Analýza úlohy 68
5.2 Riešenie 69
5.3 Hlavný blok programu 70
5.4 Definícia pomocníkov 71
6 Námety na riešenie 75
6.1 Fraktály a opakujúce sa vzory 75
6.2 Didaktické hry v JavaScripte 76
Záver 78
Literatúra 79

Predslov

Príprava učiteľov a odborníkov v oblasti informatiky nie je jednoduchá. Vzhľadom na to, že v oblasti informatiky sa priebežne skracuje časový interval, počas ktorého sa množstvo informácií zdvojnásobí, je dôležité vhodne a správne stanoviť obsah študijných programov a jednotlivých predmetov tohto odboru. Je potrebné do výučby konkrétnych predmetov vybrať tie poznatky, ktoré budú tvoriť pevný základ na ich permanentné samovzdelávacie na ceste profesijným životom. V oblasti programovania je potrebné, aby študenti – budúci učitelia informatiky získali počas učiteľskej prípravy pevné základy algoritmizácie, aby zvládli konkrétne algoritmy riešenia úloh a aby dokázali aplikovať teoretické vedomosti na riešenie neštandardných problémov.

Algoritmizácia a programovanie má v informatikom vzdelávaní špecifické postavenie. Teoretické základy a univerzálne, často empirické poznatky z tejto oblasti sa dajú aplikovať pri používaní rôznych programovacích jazykov a prostredí a na tvorbu aplikácií v nich. Aj programovacie jazyky sa neustále vyvíjajú a programovacie prostredia sa stávajú čoraz inteligentnejšími. Teda ani v tomto prípade nie je dobré sa pevne orientovať na jeden prostriedok. Hlavná autorka učebnice často zdôrazňuje svojim študentom, adeptom učiteľského štúdia informatiky, že sa neučia (len) programovací jazyk a tvorbu aplikácií v programovacom prostredí, ale predovšetkým algoritmizáciu a programovanie – treba rozvíjať ich algoritmické, programátorské a analytické myslenie. Pritom nepodceňuje ani znalosť prostredia, v ktorom pracujú, čo je veľmi dôležité na efektívnu tvorbu, optimálne riešenie a implementáciu. Takto sa stretne návrh zhora nadol a zdola nahor.

Predpokladáme, že adepti učiteľstva informatiky budú sami v praxi pri vyučovaní takto postupovať a že svojich žiakov a študentov pripravia na plnohodnotný život v informačnej spoločnosti, ktorá kladie nemalé nároky na informačnú gramotnosť svojich členov. Hlavným cieľom učebnice je zefektívniť a zatraktívniť vyučovanie programovania na základných školách a poskytnúť vhodné námety na riešenie úloh. Chceme tiež ukázať, že programovanie je jednoduché a môže byť aj zábavné. Vhodné programové aplikácie a zavedenie nových technológií do vyučovania prispieva k budovaniu pozitívneho vzťahu žiaka k digitálnym technológiám a aj k vyučovaniu a k škole.

Inovovaný štátny vzdelávací program pre základné školy zdôrazňuje dôležitosť rozvoja digitálnej gramotnosti a kultúry, algoritmického, logického a kritického myslenia. Na to je dôležitým nástrojom a prostriedkom predmet informatika. V tomto predmete je vytvorený priestor aj na programovanie, edukačnú robotiku, uplatnenie metód umelej inteligencie a na využívanie moderných vzdelávacích technológií.

K osvedčeným mikrosvetom reprezentovaným prostrediami ako Comenius Logo, Imagine, Baltík a Baltazár pribudli ďalšie prostredia na tvorbu programových aplikácií interaktívnym spôsobom – poskladaním riešenia úlohy zo stavebných elementov. Konštruovanie riešenia sa uplatňuje aj v programovaní robotov. Univerzálnym nástrojom na tvorbu programov na základnej škole sa využíva programovacie prostredie Scratch. Dá sa

využívať na programovanie robotických hračiek, robotov zostavených z robotických stavebníc alebo na tvorbu rôznych aplikácií.

Autori budú vďační za každý dobrý námet, konštruktívny návrh a za každú kritickú pripomienku, ktoré by mohli kvalitu predkladaného učebného textu vylepšiť a skvalitniť.

Prajeme každému čitateľovi veľa úspechov a chuť do práce pri tvorbe programov, na riešenie jednoduchých úloh, ako aj pri realizácii zložitejších programových projektov.

V Trnave 5. 7. 2025 Autori

1 Úvod do algoritmizácie a programovania

V každodennom živote nás obklopuje nekonečný počet algoritmov. Mnohé z nich sú veľmi jednoduché, ale niektoré sú náročné a veľmi zložité. Podobne, ako je zložitý sám ľudský život.

Mnohé aktivity ľudského života je možné vyjadriť vo forme algoritmu. Pri mnohých činnostiach je potrebné postupovať podľa prísnych a presných pravidiel a dodržiavať presné pracovné prípadne výrobné postupy. Napríklad každý chirurgický zákrok je realizovaný podľa prísnych pravidiel. Príprava každého lieku si vyžaduje presný technologický postup. Príprava každého jedla, výroba ľubovoľného výrobku, vybavenie hocakej maličkosti sa deje podľa určitého algoritmu.

Algoritmy v jednoduchej písomnej forme sú napríklad v kuchárskych knihách, v návodoch na obsluhu telefónnych automatov, automatov na cestovné lístky či na rýchle občerstvenie.

V mnohých vedných disciplínach, ako napríklad matematika, sa používajú špeciálne algoritmy ako postupy riešenia určitého okruhu problémov. Niektoré z nich sú aj označené ako algoritmy, napríklad Euklidov algoritmus (na určenie najväčšieho spoločného deliteľa). Mnohé z nich sú známe ako určitá metóda alebo postup riešenia konkrétneho problému.

Algoritmy v oblasti programovania sa líšia len v tom, že sú zostavované s určitým cieľom: cieľom automatizovať riešenie určitých vybraných problémov pomocou počítača.

Na to, aby riešenie určitých úloh mohlo byť ponechané počítaču, je potrebné napísať program, v ktorom bude zachytený presný postup riešenia. Dobrý algoritmus je základom dobrého programu. Dobrý program sa nepodarí napísať bez optimálneho a efektívneho algoritmu.

I keď program sa dá považovať za jednu formu zápisu algoritmu, na zápis algoritmov sa používajú aj iné prostriedky, ktoré nie sú poznačené zvláštnosťami toho ktorého programovacieho jazyka.

Algoritmus sa dá vyjadriť slovne (použiť na jeho vyjadrenie slovnú formuláciu), dá sa zapísať v grafickej forme pomocou vývojového diagramu alebo štruktúrogramu (tzv. Nassi-Schneidermannovho diagramu). Na vyjadrenie algoritmu sa dajú použiť aj rozhodovacie tabuľky. V niektorých literárnych prameňoch, ale aj v učebniciach sa na zápis algoritmu používa špeciálne zavedený algoritmický jazyk alebo metajazyk. Celkovo, v literatúre sa dajú nájsť rôzne verzie definície algoritmu. Napríklad:

Algoritmus je jednoznačný zrozumiteľný postup, umožňujúci zo zadaných vstupných údajov získať v konečnom čase po vykonaní konečného počtu krokov správe výsledky – riešenie zadanej úlohy.

Algoritmizáciou sa potom označuje proces, počas ktorého je vytváraný algoritmus.

Algoritmus musí spĺňať nasledujúce základné požiadavky:

1. **Elementárnosť** – postup musí pozostávať z čiastkových činností, ktoré sú na danej úrovni (na úrovni realizačného prostriedku) elementárne.
2. **Determinovanosť** – v každom momente realizácie postupu musí byť jednoznačne určené, ktorá činnosť má byť realizovaná ako nasledujúca (za činnosť sa považuje aj ukončenie procesu).
3. **Konečnosť** – postup sa musí skončiť v konečnom čase, po vykonaní konečného počtu elementárnych krokov (činností).
4. **Rezultatívnosť** a **korektnosť** – po konečnom počte krokov sú získané správne výsledky.
5. **Hromadnosť** – postup musí byť použiteľný pre celú triedu prípustných vstupných údajov, teda musí zadaný problém riešiť všeobecne a nesmie problém zúžiť len na riešenie niektorých špeciálnych prípadov.

Okrem uvedených základných vlastností na algoritmus sú kladené aj ďalšie požiadavky, napríklad:

Efektívnosť – postup musí predkladaný problém riešiť efektívne, za čo najkratší čas a s využitím minimálnych prostriedkov.

Pri tvorbe programov sa efektivita zabezpečuje minimalizáciou výpočtovej zložitosti a pamäťovej náročnosti programu. Pri riešení ľubovoľného problému je snahou nájsť čo najefektívnejší algoritmus.

Čitateľnosť a zrozumiteľnosť – každý algoritmus musí byť prehľadne napísaný, aby bol ľahko a rýchlo pochopiteľný.

Pri zápise istého postupu sú dôležité dve kvalitatívne odlišné stránky postupu:

Riadiaca zložka – určuje, v akom poradí majú byť vykonané jednotlivé čiastkové činnosti (kroky) postupu. (Krok, ktorý sa práve realizuje má riadenie a po skončení svojej činnosti ho odovzdá ďalšiemu kroku. Tok riadenia vyjadruje logickú časovú následnosť vykonávania jednotlivých krokov algoritmu).

Operačná zložka – určuje charakter (náplň) činností v danom okamihu realizácie procesu.

Prevažná časť úloh, ktoré treba riešiť pomocou výpočtovej techniky, nie sú jednoduché. Len výnimočne vedú k jednoduchým, jednokrokovým algoritmom. Väčšinou je potrebné pri riešení problémov použiť rozklad problému na podproblémy. Používajú sa tri základné typy rozkladov.

Konjukntívny rozklad – je rozdelením riešenia problému na sekvenciu krokov. Takto konštruovaný algoritmus sa nazýva lineárny alebo tiež sekvenčný – teda **sekvencia.**

Pri konjunktívnom rozklade je problém P rozkladaný na podproblémy P1, P2 … Pn, pričom konjukciou ich riešení v prislúchajúcom poradí je získané riešenie daného problému.

Disjunktívny rozklad – je rozdelením riešenia problému na niekoľko selektívnych riešení, pričom riešením konkrétneho prípadu je vždy jedno jediné riešenie z alternatívnych možností. Algoritmus, ktorý vznikne s využitím disjunktívneho rozkladu, sa nazýva vetvený algoritmus alebo **vetvenie.**

Pri disjunktívnom rozklade teda je problém P rozkladaný na podproblémy P1, P2 … Pn tak, že riešením konkrétneho problému je riešenie iba jedného vybraného problému Pi.

Cyklický (repetičný) rozklad – je vytvorenie riešenia problému (alebo podproblému) opakovaním určitej postupnosti krokov, opakovaním riešenia určitého jednoduchšieho problému. Algoritmus, ktorý sa získa opakovaním jednoduchšieho algoritmu, sa nazýva cyklický algoritmus alebo **cyklus.**

Vzájomná väzba podproblémov môže mať rôzny charakter. Charakter tejto väzby vyjadruje vzájomný vzťah stavebných elementov (operačných prvkov a riadiacich štruktúr) vyjadrujúcich ich riešenia. Z bázových elementov sa dajú vytvárať riadiace konštrukcie na vyššej úrovni. A tiež každá konštrukcia môže byť bázovým stavebným elementom inej konštrukcie.

Pri vytváraní algoritmov budeme používať nasledujúce pravidlá:

- Zložitý problém bude vždy rozdelený na určité podproblémy – podľa možností samostatné celky. Pri rozkladaní problému sa bude postupovať **zhora nadol.**
- Budú vždy vytvárané ucelené algoritmické konštrukcie. (Ucelená algoritmická konštrukcia je taká riadiaca štruktúra, ktorá má jeden vstup a jeden výstup vzhľadom na tok riadenia.)

1.1 Slovné vyjadrenie algoritmu

Algoritmus môže byť vyjadrený aj slovne bez použitia špeciálnych prostriedkov. Slovný zápis algoritmu je najjednoduchší a používateľa nijakým spôsobom neobmedzuje. I napriek tomu, ak má byť tento spôsob zápisu algoritmu použitý, je vhodné tento postup činností vyjadriť tak, aby jednotlivé body vyjadrovali určité elementárne kroky (operácie, príkazy, inštrukcie, stavebné elementy) na danej úrovni riešenia.

Pri riešení ľubovoľnej úlohy pomocou výpočtovej techniky sa najčastejšie používa metóda **zhora nadol.** To znamená, že úloha sa postupne rozkladá na jednoduchšie podúlohy. Na jednotlivých úrovniach riešenia sú používané len štandardné (štruktúrované) elementy algoritmizácie a programovania. Mnohé z nich majú priamy implementačný ekvivalent vo vyšších programovacích jazykoch. Medzi základné riadiace konštrukcie patria **sekvencia, vetvenie** a **cyklus.**

Na riešenie mnohých problémov jestvujú známe postupy, známe algoritmy. Známe sú mnohé algoritmy v oblasti exaktných vied. Na druhej strane sú aj také problémy, riešenie ktorých je ťažko algoritmizovateľné alebo jednoducho riešenie nejestvuje, prípadne nie je

deterministické. Na riešenie mnohých problémov jestvuje viacero algoritmov, ale sú aj také problémy, na riešenie ktorých zatiaľ žiadny algoritmus nie je známy.

Z matematiky je dobre známy už spomínaný Euklidov algoritmus na určenie najväčšieho spoločného deliteľa (NSD) nezáporných čísel.

Pred pokračovaním treba vysvetliť jednu vec: Keď programátor chce, aby si počítač niečo zapamätal – napríklad skóre v hre alebo meno hráča – potrebuje na to miesto, kde si počítač túto informáciu uloží. Takéto miesto sa v programovaní nazýva *premenná.* V detských programovacích jazykoch, ako je Scratch, vyzerá premenná ako „nálepka," na ktorej je napísané meno a do nej sa dajú vložiť rôzne hodnoty.

Premenná je pomenované miesto, do ktorého sa dajú ukladať rôzne informácie – číslo, text alebo inú hodnota – a táto hodnota sa dá neskôr meniť alebo čítať.

V algoritme nižšie sú použité premenné **a, b** a **c.**

Euklidov algoritmus:

Nech **a, b** sú celé nezáporné čísla, nech sa (pre potreby tejto ukážky) aspoň jedno z **a** a **b** nerovná nule a **a ≥ b.** Potom ak **b ← 0,** najväčším spoločným deliteľom čísel **a, b** (NSD(**a, b**)) je **a.** Ak **b ≠ 0,** potom pre čísla **a, b** a **c,** kde **c** je zvyšok celočíselného delenia **a / b** platí rovnosť NSD(**a, b**) = NSD(**b, c**).

Napríklad NSD(15, 6) = NSD(6, 3) = NSD(3, 0) = 3.

Na základe tejto matematickej definície sa dá relatívne jednoducho zapísať slovný algoritmus na určenie najväčšieho spoločného deliteľa dvoch nezáporných celých čísel.

1. Urči čísla **a** a **b**.
2. Testuj čísla **a** a **b** {**a**, **b** sú nezáporné, aspoň jedno sa nerovná 0, **a** ≥ **b**}.
3. Ak **b** = 0 potom NSD ← **a**; koniec.
4. Inak **c** ← zvyšok po delení **a** / **b**; **a** ← **b**; **b** ← **c**; vráť sa na bod 3.

Sledujme realizáciu algoritmu pre dve konkrétne čísla: **a** ← 32 a **b** ← 12:

1. **a** ← 32 a **b** ← 12.
2. Čísla vyhovujú predpokladom.
3. **b** ≠ 0.
4. **c** ← zvyšok(32 / 12 = 8); **a** ← 12; **b** ← 8.
3. **b** ≠ 0.
4. **c** ← zvyšok(12 / 8 = 4); **a** ← 8; **b** ← 4.
3. **b** ≠ 0.
4. **c** ← zvyšok(8 / 4 = 0); **a** ← 4; **b** ← 0.
3. **b** ← 0; NSD ← 4.

Poznámka: Na jednoznačnosť algoritmu ešte treba určiť činnosť v prípade nevyhovujúceho výsledku testu v bode 2. Pre **a < b** treba vzájomne vymeniť hodnoty **a** a **b** a pokračovať bodom 3.

Poznámka: K pojmu *premenná* jestvujú v programovaní aj príbuzné pojmy, ktoré v detských programovacích jazykoch buď nejestvujú, alebo sú zamlčané. Napríklad **konštanta** – vyzerá programovacích jazykoch presne ako *premenná,* ale je to pomenovaná hodnota, ktorá sa nemení (podobne ako napríklad vo fyzike je konštantou **c** označovaná rýchlosť svetla). Na odlíšenie od *„konštantných hodnôt“* zapísaných priamo do zdrojového kódu (napríklad číslo **10** alebo reťazec (text) *„Ahoj, zdravím Ťa!“*) bol zavedený pojem **literál,** čiže niečo *doslovne uvedené,* hodnota priamo zapísaná v kóde.

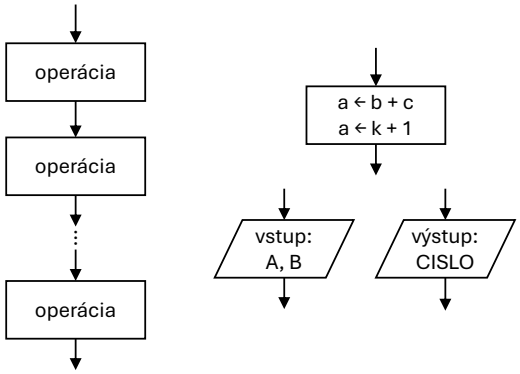
1.2 Zápis algoritmov vývojovými diagramami

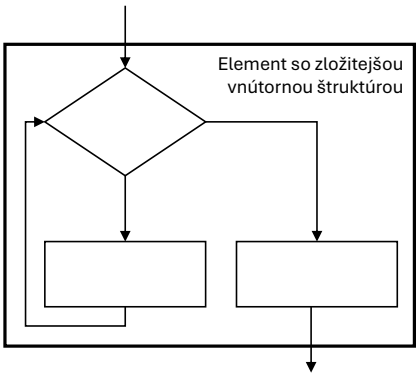
Algoritmus je možné zapísať vývojovým diagramom. Pri zostavovaní vývojových diagramov je algoritmus vyjadrovaný pomocou definovaných značiek. Jednotlivé bloky vývojových diagramov umožňujú vyjadriť rôzne riadiace konštrukcie, ktoré sú štandardné z hľadiska algoritmizácie a programovania. Pri zostavovaní algoritmov je snahou nájsť taký postup riešenia, ktorý je možné vyjadriť prostredníctvom týchto štandardných riadiacich konštrukcií.

Základné riadiace štruktúry používané v algoritmizácii majú v rôznych odborných textoch zaoberajúcich sa programovaním rôzne označenia, napríklad **podmienené spracovanie** sa môže inak volať **vetvenie bez alternatívy, neúplné vetvenie, neúplné binárne vetvenie, „podmienka“** (veľmi nepresné – podmienka je iba malá súčasť tejto štruktúry a samotné podmienky sú prítomné aj v iných štruktúrach), **if** (podľa anglického kľúčového slova programovacieho jazyka), prípadne inak. Nasledujúci text zhŕňa základné riadiace štruktúry a ich najpoužívanejšie označenia.

Sekvencia – je sled za sebou nasledujúcich štandardných operačných prvkov a algoritmických konštrukcií.

Ako jednoduché operácie môžu vystupovať napríklad priradenie alebo vstup/výstup.



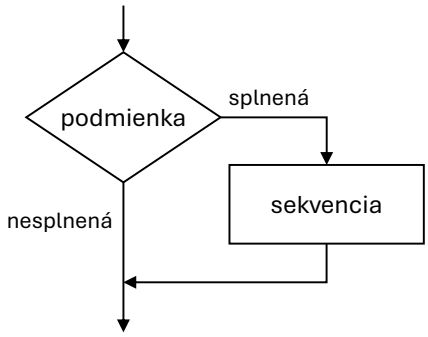


Vnútorná štruktúra elementu môže byť aj zložitejšia, môže byť zložitou, štruktúrovaným spôsobom vybudovanou, algoritmickou konštrukciou.

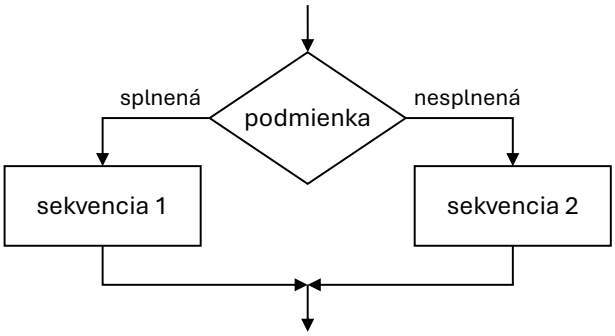
Jedinou požiadavkou sekvenčného radenia blokov je, že musia mať jeden vstup a jeden výstup.

Vetvenie – je výber jednej možnosti z viacerých na základe určitej podmienky. Sú rôzne druhy vetvenia:

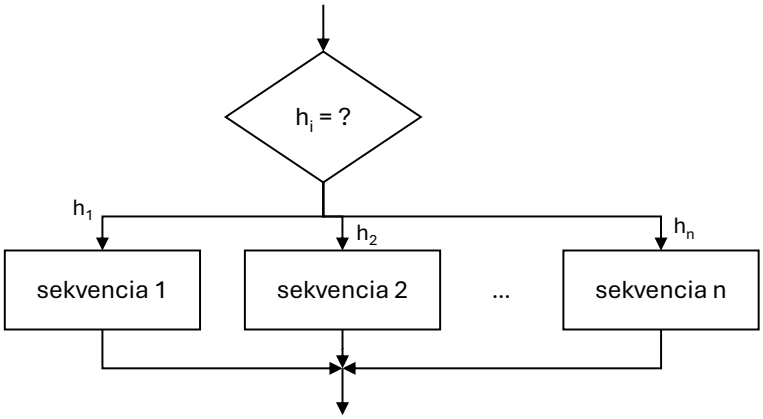
a) **Podmienené spracovanie; neúplné vetvenie** – uvedená sekvencia sa realizuje len v prípade splnenia podmienky. Ak podmienka nie je splnená, sekvencia sa vynechá (preskočí).



b) **Jednoduché vetvenie; binárne vetvenie** – realizuje sa jedna z dvoch možností (sekvencia 1 alebo sekvencia 2) podľa splnenia alebo nesplnenia podmienky.

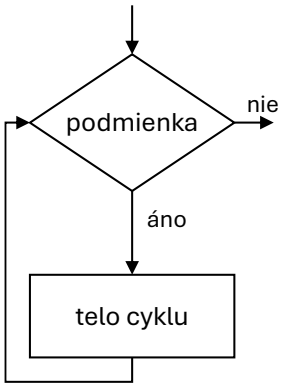


c) **Viacnásobné vetvenie; selekcia** – podľa hodnoty selektora sa realizuje jedna z viacerých možností. Napríklad ak má selektor h_i hodnotu h_2, tak sa vykoná sekvencia 2.

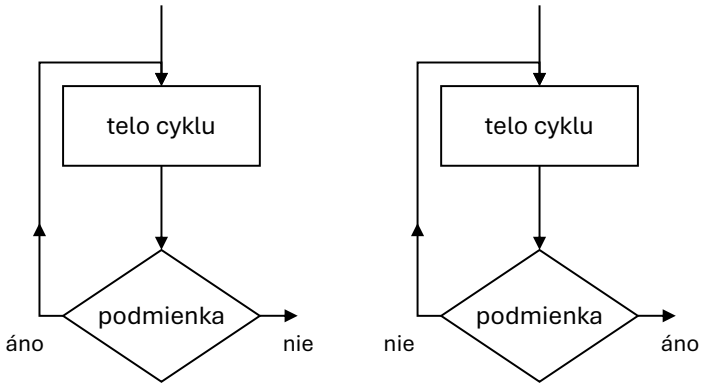


Cyklus – je viacnásobné opakovanie určitej aktivity (sekvencie) tzv. tela cyklu.

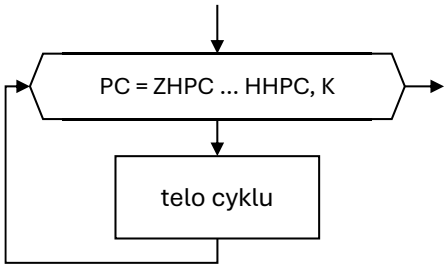
a) **Cyklus s podmienkou na začiatku; cyklus s testovaním na začiatku** (pred akciou) – najprv je testovaná podmienka cyklu. Ak je podmienka splnená, tak je vykonané telo cyklu. V opačnom prípade sa pokračuje realizáciou nasledujúceho bloku, do ktorého vedie vetva označená **nie.**



b) **Cyklus s podmienkou na konci; cyklus s testovaním na konci** (po akcii) – najprv je vykonané telo cyklu a potom je testovaná podmienka. Tento cyklus má dve alternatívy. Rôzne programovacie jazyky k nemu pristupujú inak. V niektorých sa telo cyklu opakuje, ak je splnená podmienka (**do-while;** diagram vľavo), v iných je to presne naopak (**repeat-until;** diagram vpravo).

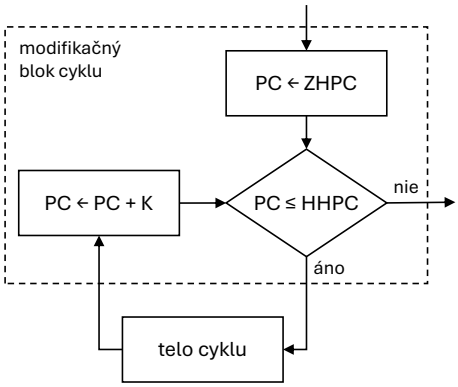


c) **Cyklus so známym počtom opakovaní; cyklus riadený parametrom** (niekedy aj: *cyklus s určitým počtom iterácií*) – na začiatku je nastavená hodnota parametra cyklu (PC) na začiatočnú hodnotu parametra cyklu (ZHPC). Potom je testovaná platnosť koncovej podmienky cyklu (napríklad PC ≤ HHPC). Väčšina programovacích jazykov súčasnosti umožňuje túto podmienku zadať explicitne a upraviť ju podľa potrieb. Ak je táto podmienka splnená, realizuje sa telo cyklu a parameter cyklu sa modifikuje (často pripočítaním/odpočítaním hodnoty kroku K: PC ← PC + K, resp. PC ← PC − K). Aj spôsob modifikácie parametra je v moderných programovacích jazykoch explicitne modifikovateľný. Musí byť zabezpečené najmä to, aby modifikácia parametra cyklu smerovala ku splneniu koncovej podmienky. Potom nasleduje návrat na testovanie podmienky.



PC – parameter cyklu;
ZHPC – začiatočná hodnota parametra cyklu;
HHPC – hraničná (koncová) hodnota parametra cyklu;
K – krok

Na lepšie pochopenie funkcie a významu modifikačného bloku poslúži príklad podrobnejšieho rozkreslenia aktivít prostredníctvom skôr vysvetlených blokov vývojových diagramov:



Z obrázka vidieť, že v modifikačnom bloku sú združené tri aktivity:

- nastavenie parametra cyklu na začiatočnú hodnotu,
- testovanie parametra cyklu,
- modifikácia parametra cyklu.

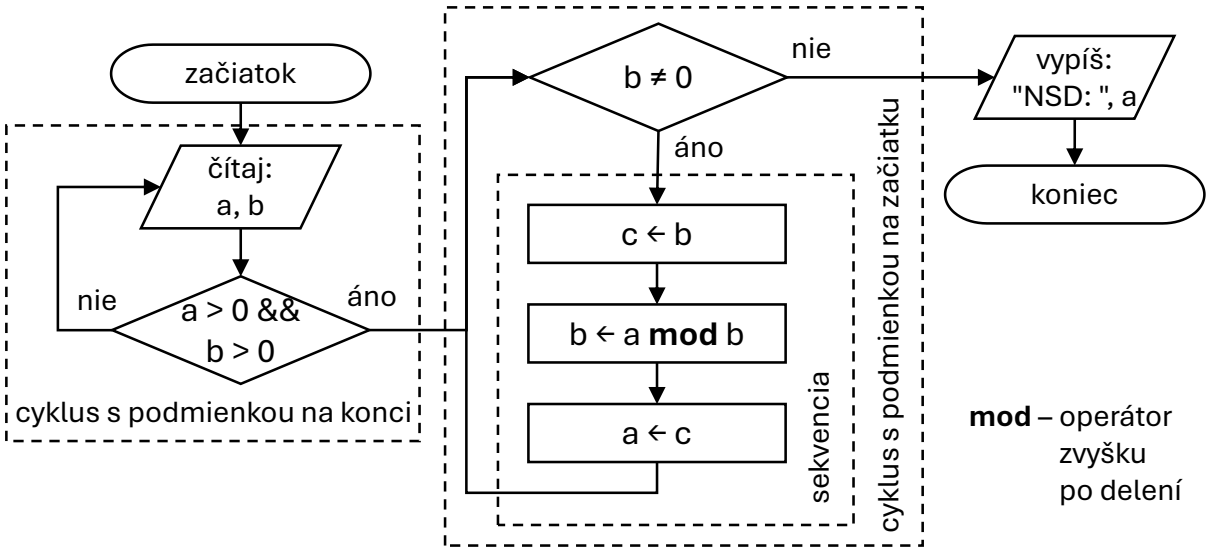
(Ide o príklad cyklu s rastúcim parametrom.)

Pri používaní vývojových diagramov treba dbať na to, aby sa používali len uvedené štandardné riadiace štruktúry a tak bola zabezpečená štruktúrovaná tvorba algoritmov. Každý

blok na vyššej úrovni má jeden vstup a jeden výstup. Vnútorná štruktúra každého bloku je iná, ale vždy je to jedna zo základných riadiacich konštrukcií.

Algoritmus vyjadrený vývojovým diagramom môže byť ilustrovaný známym Euklidovým algoritmom. (Na zjednodušenie testovania bude teraz požadované, aby ani jedna zo vstupných hodnôt nemohla byť iná, než kladná.)

Euklidov algoritmus:

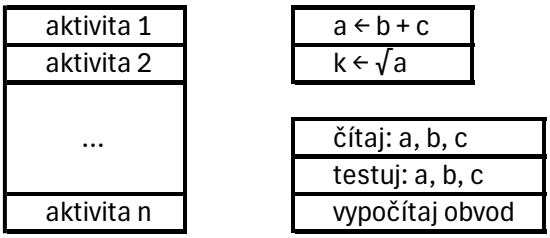


1.3 Zápis algoritmov štruktúrogramami – Nassi-Schneidermannovými diagramami

Štruktúrogramy alebo Nassi-Shneidermanove diagramy sa začali používať v 70. rokoch ako nástroj podporujúci štruktúrovaný spôsob tvorby algoritmov. Diagramy pozostávajú z elementov zodpovedajúcich základným riadiacim konštrukciám a preto sa často využívajú na vizuálne znázornenie štruktúry algoritmov alebo jednoduchých programov. Na opis syntaxe programovacích jazykov sa používajú napríklad syntaktické diagramy alebo Backusova-Naurova forma (BNF). Na znázornenie štruktúry algoritmov sa používajú vývojové diagramy alebo štruktúrogramy.

Nasledujúca pasáž uvádza stručný prehľad jednotlivých riadiacich konštrukcií. Ich význam je rovnaký ako v prípade vývojových diagramov.

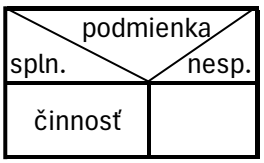
Sekvencia

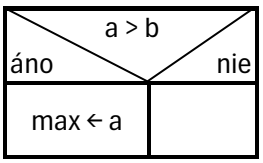


Uvedené sekvencie je možné použiť na ľubovoľnej úrovni rozpracovania postupu riešenia. Napríklad telo cyklu môže obsahovať jediný príkaz, jednoduchú sekvenciu príkazov, ale aj zložitejšie riadiace konštrukcie, vrátane ďalších cyklov alebo vetvení. (V najjednoduchšom prípade však ide len o jednu alebo viacero priamych inštrukcií bez ďalších vnorených štruktúr.)

Vetvenie

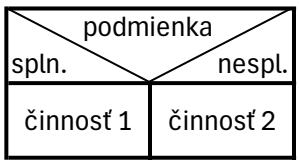
a) **Podmienené spracovanie; neúplné vetvenie:**

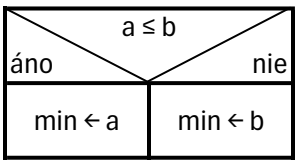




Príklad ukazuje situáciu, kedy sa do premennej **MAX** priradí hodnota inej premennej **a** len vtedy, ak platí podmienka **a > b.**

b) **Jednoduché vetvenie, binárne vetvenie:**

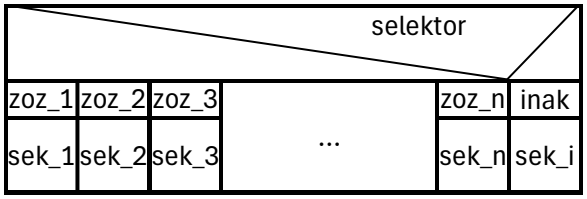




Príklad ukazuje situáciu, kedy ak platí podmienka **a ≤ c,** tak do premennej **MIN** sa priradí hodnota premennej **a.** Ak uvedená podmienka neplatí, tak sa do premennej **MIN** priradí hodnota premennej **c.**

c) **Viacnásobné vetvenie; selekcia** (prepínač):

Zakreslenie tejto štruktúry sa v rôznych zdrojoch líši. V pôvodnom návrhu Nassiho a Schneidermanna je táto štruktúra zakreslená tak, že hodnoty selektora nie sú zoznamami a sú nakreslené schodovito tesne pod ľavou šikmou čiarou. Takéto zakreslenie je však technicky ťažšie realizovateľné, tak sa prirodzene stretávame s rôznymi zjednodušeniami, ktoré sa v rôznej miere odchyľujú od originálu. Najväčší rozdiel, s ktorým je možné sa v rôznych zdrojoch stretnúť sa skôr podobá na akúsi inverznú verziu hornej časti štruktúry paralelného spracovania pôvodného návrhu Nassiho a Schneidermanna (tou sa v tomto materiáli nezaoberáme).



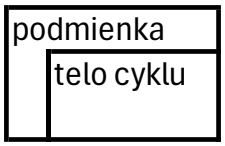
Pri tejto štruktúre má algoritmus na výber z niekoľkých vetiev. To, ktorá bude vykonaná sa vyberá na základe aktuálnej hodnoty premennej (*selektora*), ktorá sa porovnáva so zoznamami hodnôt (**zoz_1 ... zoz_n;** každý zoznam môže byť aj jednoprvkový) uvedených pri jednotlivých vetvách. Jedna hodnota smie byť uvedená len v jednom zozname. Keď je

aktuálna hodnota premennej nájdená v niektorom so zoznamov, tak je vykonaná práve táto vetva príkazov (**sek_1 ... sek_n**). Ak nie je hodnota premennej nájdená v žiadnom so zoznamov, tak je vykonaná vetva alternatívnych príkazov (**inak: sek_i**), ak je prítomná.

Cyklus

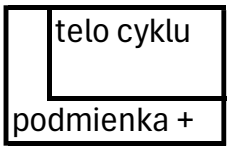
Pri cykloch sa v dostupných zdrojoch je možné stretnúť s rozmanitými spôsobmi zakreslenia, ktoré sa odchyľujú od pôvodného návrhu Nassiho a Schneidermanna. Rozdiely sa dotýkajú najmä cyklov s podmienkou na konci a cyklu so známym počtom opakovaní. Pri podmienke na konci sa dajú nájsť zdroje s blokom tela cyklu napravo aj naľavo, zrejme podľa toho, či má podmienka na konci platiť, alebo nie. Pri známom opakovaní sa dajú nájsť zdroje, v ktorých je tento cyklus totožný s cyklom s podmienkou na začiatku, len s odlišným zápisom „podmienky" a zdroje, kde je tento cyklus vyjadrený tak, ako bol v pôvodnom návrhu vyjadrovaný vnorený blok príkazov. Dokonca sú zdroje, v ktorých sú všetky tri cykly v úplnom nesúlade s pôvodným návrhom a sú vyjadrené s použitím neúplného vetvenia.

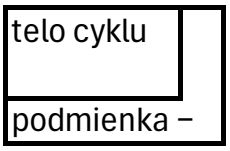
a) **Cyklus s podmienkou na začiatku; cyklus s testovaním na začiatku** (pred akciou):



b) **Cyklus s podmienkou na konci; cyklus s testovaním na konci** (po akcii):

Pri tejto štruktúre sa v rôznych zdrojoch opäť dá stretnúť s rozdielmi. Tie plynú z toho, že vo väčšine jazykov musí podmienka na konci cyklu platiť, aby sa telo cyklu zopakovalo (**do-while**), ale stále jestvujú jazyky, pri ktorých je kritérium opačné a podmienka musí „začať platiť," aby sa telo cyklu „prestalo opakovať," čiže na zopakovanie tela cyklu podmienka nesmie platiť (**repeat-until**).





Grafický rozdiel medzi štruktúrou vyžadujúcou pravdivú (**do-while;** vľavo) a nepravdivú (**repeat-until;** vpravo) podmienku na zopakovanie tela cyklu nie je nikde zdokumentovaný (nepodarilo sa to nikde nájsť explicitne vyjadrené). Naopak, originálny návrh Nassiho a Schneidermanna obsahuje iba jeden variant a ten je považovaný za „univerzálny" –vyzerá ako **do-while** a v texte sa na neho odvolávajú zároveň ako na **repeat-until** a zároveň ako na **repeat-while** (čo je unikátne označenie, ktoré sa dnes vôbec nepoužíva). Je však pravdepodobné, že s rozmachom jazykov používajúcich štruktúru **do-while** vznikla potreba rozlíšenia štruktúry **repeat-until** a vtedy ju (pravdepodobne) začali niektorí autori „preklápať" doľava. Je to jediné logické vysvetlenie a takéto grafické „preklopenie" zároveň dáva zmysel.

c) **Cyklus so známym počtom opakovaní; cyklus riadený parametrom** (varianty):

```
pre ⟨premenná⟩ od ⟨výraz 1⟩ po ⟨výraz 2⟩ krokom ⟨výraz 3⟩
    ⟨sekvencia⟩
    ...
```

```
pre ⟨premenná⟩ ← ⟨počiatočný výraz⟩; ⟨podmienka ukončenia⟩; ⟨iteračný výraz⟩
    ⟨sekvencia⟩
    ...
```

Pri zakresľovaní tejto štruktúry sa opäť dá naraziť na rozdiely dotýkajúce sa spôsobu zakreslenia tela cyklu. Okrem vyššie uvedeného spôsobu, ktorý je v súlade s pôvodným návrhom Nassiho a Schneidermanna, je možné sa stretnúť so spôsobom, ktorý je v pôvodnom návrhu uvedený ako vnorený blok. Buď ide o omyl, keďže tieto dve štruktúry sú v pôvodnom článku Nassiho a Schneidermanna nakreslené vedľa seba a sú vysvetľované jedna po druhej, alebo ide o úsilie o vizuálne odlíšenie od cyklu s podmienkou na začiatku, keďže vnorený (anonymný) blok sa pri programovaní používa iba výnimočne.

Pri použití Nassi-Schneidermannových diagramov samotný spôsob zápisu neumožňuje iný než štruktúrovaný spôsob tvorby algoritmu.

1.4 Zápis algoritmov v algoritmickom jazyku

V literárnych prameňoch sa dajú nájsť rôzne zavedené a definované algoritmické jazyky, ktoré umožňujú vyjadrenie algoritmov. Sú to formálne jazyky, ktoré sa spôsobom formulácie svojich príkazov približujú základným myšlienkovým operáciám človeka pri riešení problémov. Väčšinou nie sú implementované na počítačoch, ale formálne sú veľmi blízke k programovacím jazykom.

Nižšie opísaný algoritmický jazyk je vybudovaný štruktúrovane. Používa kľúčové slová odvodené zo slovenčiny, ktoré vyjadrujú charakter činností, ktoré reprezentujú. Obsahuje všetky štandardné riadiace konštrukcie moderných programovacích jazykov.

Jednotlivé príkazy jazyka sú písané lineárne v tvare viet (štruktúr) algoritmického jazyka. Všeobecné, na mieste bližšie neurčené prvky, ktoré musí programátor dodefinovať, budú uvedené v lomených zátvorkách: ⟨⟩. Je tu uvedený len stručný prehľad jednotlivých algoritmických konštrukcií známych z predchádzajúcich častí tejto kapitoly:

Sekvencia (zložený príkaz)

```
⟨príkaz P1⟩;
⟨príkaz P2⟩;
  .
```

.

⟨príkaz Pn⟩;

Príkazy P1, P2 … Pn sa vykonajú v poradí, v akom sú uvedené. Môžu sa pri tom používať príkazové zátvorky, napríklad: {⟨príkaz P1⟩; ⟨príkaz P2⟩; … ⟨príkaz Pn⟩;}

Vetvenie

a) **Podmienené spracovanie; neúplné vetvenie:**

ak ⟨podmienka⟩ **tak** ⟨príkaz P⟩

Príkaz P sa vykoná len v prípade splnenia podmienky.

b) **Jednoduché vetvenie, binárne vetvenie:**

ak ⟨podmienka⟩ **tak** ⟨príkaz P1⟩

inak ⟨príkaz P2⟩

Ak je podmienka splnená, tak sa realizuje príkaz P1. Ak podmienka splnená nie je, tak sa vykoná príkaz P2.

c) **Viacnásobné vetvenie; selekcia:**

v prípade, že ⟨premenná⟩ **je v** ⟨zoznam hodnôt 1⟩, **tak** ⟨príkaz/sekvencia 1⟩

je v ⟨zoznam hodnôt 2⟩, **tak** ⟨príkaz/sekvencia 2⟩

.

.

je v ⟨zoznam hodnôt n⟩, **tak** ⟨príkaz/sekvencia n⟩

inak ⟨príkaz/sekvencia k⟩

Vykoná sa ten príkaz (prípadne sekvencia), resp. tá vetva, ktorá má v zozname hodnôt uvedenú aktuálnu hodnotu selektujúcej premennej. Ak sa hodnota, ktorú nadobudol výraz, nenachádza ani v jednom zozname hodnôt štruktúry, tak sa vykoná príkaz (sekvencia) k. Zoznamy hodnôt by mali byť zoznamy literálov.

Cyklus

a) **Cyklus s podmienkou na začiatku; cyklus s testovaním na začiatku** (pred akciou):

kým platí ⟨podmienka⟩ **opakuj** ⟨sekvencia⟩

Sekvencia (telo cyklu) sa vykoná len vtedy, keď je splnená podmienka. Prvé nesplnenie podmienky spôsobí ukončenie cyklu. Ak nie je podmienka splnená ani pri prvom testovaní, tak sa sekvencia nerealizuje ani raz.

b) **Cyklus s podmienkou na konci; cyklus s testovaním na konci** (po akcii):

opakuj ⟨sekvencia⟩, **kým platí** ⟨podmienka⟩

opakuj ⟨sekvencia⟩, **kým neplatí** ⟨podmienka⟩

Najprv sa vykoná sekvencia a až potom je testovaná podmienka. Tu je možné sa stretnúť s dvomi variantmi. Častejší je v dnešných jazykoch variant, kedy podmienka musí platiť, aby sa cyklus zopakoval, ale stále jestvujú jazyky, v ktorých je platnosť opačná. V prípade, že je v oboch prípadoch splnené kritérium opakovania, cyklus sa zopakuje, v opačnom prípade skončí. Sekvencia (telo cyklu) sa pri tomto cykle **vždy** vykoná **aspoň raz.**

c) **Cyklus so známym počtom opakovaní; cyklus riadený parametrom:**

pre ⟨premenná⟩ **od** ⟨výraz 1⟩ **po** ⟨výraz 2⟩ **krokom** ⟨výraz 3⟩ **opakuj** ⟨sekvencia⟩

pre ⟨premenná⟩ **od** ⟨výraz 1⟩ **dole po** ⟨výraz 2⟩ **krokom** ⟨výraz 3⟩ **opakuj** ⟨sekvencia⟩

Ak programovací jazyk nevyjadruje kritériá vykonávania tohto cyklu explicitne, tak predpokladáme, že na začiatku ⟨premenná⟩ získa hodnotu ⟨výraz 1⟩ a potom sa uvedená ⟨sekvencia⟩ vykoná toľkokrát, kým platí, že hodnota riadiacej premennej pri stúpajúcom cykle vyhovuje podmienke ⟨premenná⟩ ≤ ⟨výraz 2⟩ a pri klesajúcom podmienke ⟨premenná⟩ ≥ ⟨výraz 2⟩. Zároveň sa po každej iterácii hodnota riadiacej premennej cyklu automaticky modifikuje podľa hodnoty výrazu ⟨výraz 3⟩ – pri stúpajúcom cykle:

⟨premenná⟩ ← ⟨premenná⟩ + ⟨výraz 3⟩

a pri klesajúcom:

⟨premenná⟩ ← ⟨premenná⟩ − ⟨výraz 3⟩

(po čom sa a realizuje návrat na testovanie podmienky opakovania).

pre ⟨premenná⟩ ← ⟨počiatočný výraz⟩; ⟨podmienka ukončenia⟩; ⟨iteračný výraz⟩ **opakuj** ⟨sekvencia⟩

V moderných jazykoch býva podmienka ukončenia cyklu vyjadrená explicitne, podobne aj príkaz na zmenu hodnoty riadiacej premennej (vykonávaný po každej iterácii) býva uvedený explicitne. Všetko funguje rovnako, ibaže podmienku ukončenia cyklu a spôsob zmeny hodnoty riadiacej premennej je vidno priamo v hlavičke zápisu cyklu.

Stúpajúci implicitne zapísaný cyklus riadený parametrom (úplne prvý variant vyššie) sa dá vyjadriť pomocou cyklu s testovaním na začiatku takto:

⟨premenná⟩ ← ⟨výraz 1⟩

kým platí ⟨premenná⟩ ≤ ⟨výraz 2⟩ **opakuj**

⟨sekvencia⟩

⟨premenná⟩ ← ⟨premenná⟩ + ⟨výraz 3⟩

Okrem uvedených riadiacich konštrukcií sa používajú príkazy priradenia (napríklad a ← k + 1), vstupu (napríklad čítaj(a, b)) a výstupu (napríklad píš(x, y)) a podobne.

Je možné tiež definovať ďalšie zápisy. Napríklad algoritmus riešenia problému, ktorý je samostatnou sekvenciou (podprogram), bude mať nasledujúcu štruktúru:

algoritmus ⟨meno⟩

⟨sekvencia⟩

1.5 Tvorba algoritmov

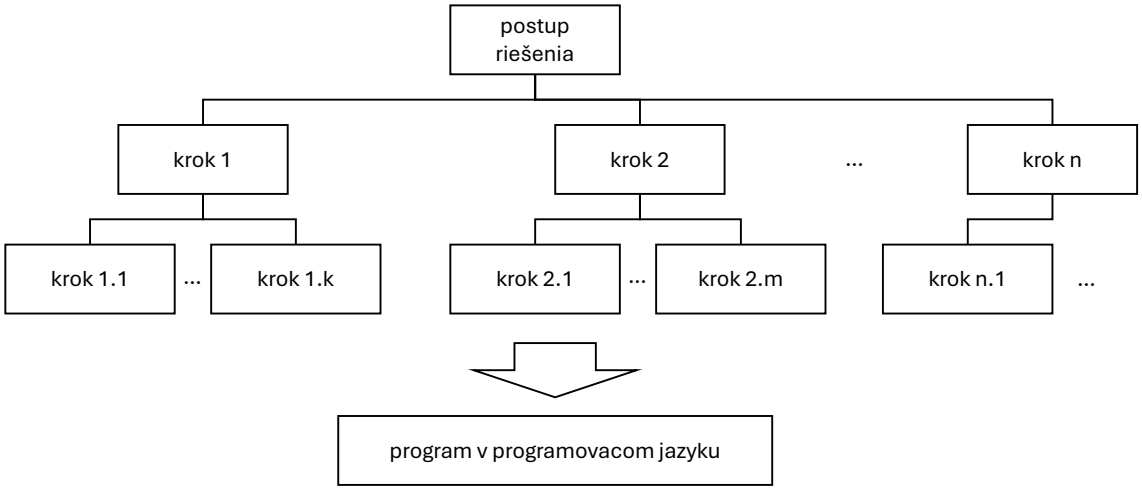
Na napísanie dobrého programu je potrebné vopred vykonať určitú postupnosť činností. Je potrebné dokonale zvládnuť (pochopiť) problém (úlohu), zvládnuť potrebné metódy riešenia, rozpracovať pravidlá riešenia – t. j. zostaviť algoritmus a až potom sa dá pristúpiť k písaniu programu. Na to, aby bol výsledný programový produkt kvalitný, je okrem toho nevyhnutná dokonalá znalosť programovacieho jazyka, všetkých jeho prostriedkov a ich zručné používanie.

V prvom kroku je nevyhnutné problém v dostatočnej miere pochopiť. V prípade jednoduchých úloh na to môže stačiť niekoľko sekúnd. Na pochopenie zložitých problémov treba viac času. Správne porozumenie problému je kľúčovým krokom k správne zostavenému algoritmu. Veď načo by bol aj ten najdokonalejší program, ktorý nerieši zadanú úlohu?

Po úplnom a správnom pochopení problému sa dá pristúpiť k zostaveniu plánu riešenia vo forme časovej následnosti bodov. Každý bod môže obsahovať jednoduchú činnosť alebo celý zložitý komplex činností.

Jednotlivé činnosti sú rozkladané na jednoduchšie dovtedy, kým nie sú dosiahnuté také elementárne činnosti, ktoré je možné realizovať pomocou príkazov programovacieho jazyka. Pri projektovaní algoritmu sa postupuje podľa metódy **zhora nadol,** teda zložité úlohy (problémy) sa postupne „rozbíjajú“ na jednoduchšie dovtedy, kým nezostanú také jednoduché problémy, ktorých riešenie sa už dá jednoducho vyjadriť pomocou príkazov používaného programovacieho jazyka.

Nasledujúci obrázok ukazuje graf projektovania algoritmu riešenia problému.



I keď tvorba algoritmu je zdanlivo nezávislá od použitého programovacieho jazyka, každý programátor rozvinie riešenie tak, aby používal také algoritmické konštrukcie, ktoré sú pre použitý programovací jazyk vlastné, ktoré majú priamu implementáciu v danom programovacom jazyku.

Na zápis algoritmov jednotlivých úrovní riešenia sa dá použiť ľubovoľná predtým uvedená forma zápisu algoritmov. Ktorý spôsob zápisu je lepší? To pravdepodobne závisí od individuálneho spôsobu myslenia riešiteľa. Často práve ten spôsob, ktorý edukant zvládne ako prvý, sa mu bude zdať najlepším, najnázornejším a najvýstižnejším. Aj v tomto prípade platí to, čo platí v prípade programovacích jazykov (a tiež v prípade prirodzených jazykov). Rovnako každého poznačí jeho materinský jazyk (jazyk, ktorý požíva ako prvý, ktorý dokonale pozná) a rovnako poznačí každého programátora jeho prvý programovací jazyk a tiež jeho prvý prostriedok na vyjadrenie algoritmov.

1.6 Otázky a úlohy

Otázky na zopakovanie

1. Čo je to *algoritmus* a aké základné vlastnosti musí spĺňať?
2. Vymenuj aspoň tri *príklady* z bežného života, ktoré sa dajú vyjadriť ako algoritmus.
3. Vysvetli rozdiel medzi *riadiacou* a *operačnou* zložkou algoritmu.
4. Aký je rozdiel medzi *konjunktívnym, disjunktívnym* a *cyklickým* rozkladom problému?
5. V čom sa líši *slovné vyjadrenie* algoritmu od *vývojového diagramu?*
6. Aké sú *výhody* a *nevýhody* zápisu algoritmu v algoritmickom jazyku?
7. Čo je to *štruktúrogram* a v čom je jeho výhoda oproti vývojovému diagramu?
8. Aké *základné riadiace konštrukcie* poznáte a čo vyjadrujú?
9. Prečo je *efektívnosť* algoritmu dôležitá? Ako je možné ju dosiahnuť?
10. Vysvetli metódu *„zhora nadol“* pri tvorbe algoritmu.

Úlohy na precvičenie

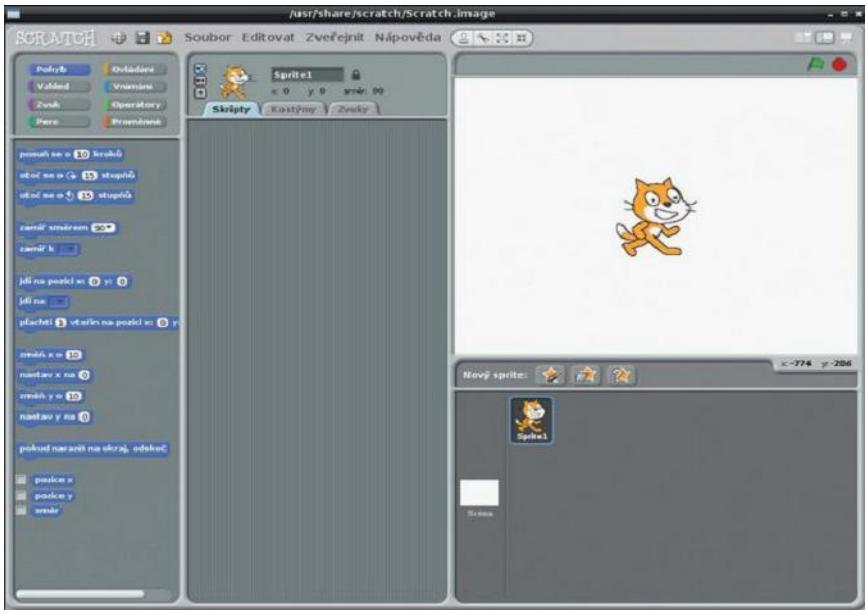
1. **Slovný algoritmus:** Vyjadri slovne algoritmus na uvarenie vajíčka na mäkko. Rozdeľ ho na elementárne kroky.
2. **Rozklad problému:** Rozlož problém „nakúpiť potraviny na obed“ na podproblémy pomocou konjunktívneho rozkladu.
3. **Euklidov algoritmus:** Vypočítaj najväčší spoločný deliteľ čísel 60 a 36 pomocou slovného Euklidovho algoritmu.
4. **Vývojový diagram:** Navrhni vývojový diagram, ktorý zistí, či je zadané číslo párne alebo nepárne.
5. **Štruktúrogram:** Nakresli štruktúrogram, ktorý vypočíta súčet čísel od 1 do 10 pomocou cyklu so známym počtom opakovaní.
6. **Algoritmický jazyk:** Zapíš algoritmus, ktorý načíta dve čísla a vypíše väčšie z nich, v algoritmickom jazyku.
7. **Typy cyklov:** Navrhni príklad pre každý z troch typov cyklov (s podmienkou na začiatku, s podmienkou na konci, so známym počtom opakovaní).

8. **Sekvencia, vetvenie, cyklus:** Vytvor jednoduchý algoritmus (v ľubovoľnej forme), ktorý vypočíta faktoriál zadaného čísla (napr. 5!) a označ v ňom, kde sú aké programové štruktúry (sekvencia, vetvenie, cyklus).
9. **Alternatívne zápisy:** Zober jeden zo svojich predchádzajúcich algoritmov a zapíš ho v inej forme (napr. ak si ho napísal ako vývojový diagram, skús ho zapísať slovne alebo v algoritmickom jazyku).
10. **Porovnanie:** Porovnaj výhody a nevýhody vývojových diagramov a štruktúrogramov z hľadiska čitateľnosti a jednoduchosti zápisu.

2 Detský programovací jazyk Scratch

Scratch je programovací jazyk s integrovaným programovacím prostredím vhodným pre začínajúceho programátora. Jeho prostredie je jednoduché, ľahko a intuitívne ovládateľné. Vytváranie programu je interaktívne a realizuje sa ukladaním jednotlivých prvkov programu do hlavného okna. Paleta príkazov obsahuje všetky príkazy (základné riadiace štruktúry), z ktorých sa dajú vytvárať programy.

Umožňuje jednoduchým spôsobom vytvárať interaktívne aplikácie, komiksy, hry, animácie a pod. Obsahuje rôzne možnosti, napríklad zdieľania vytvorených programov na internetovej stránke, inšpirácie rôznych programov aj z iných častí sveta a pod. Scratch je dostupný v mnohých jazykoch. Je možné použiť ho aj bez nutnosti inštalácie na vlastnom počítači, a to priamo na stránke Scratchu – on-line.



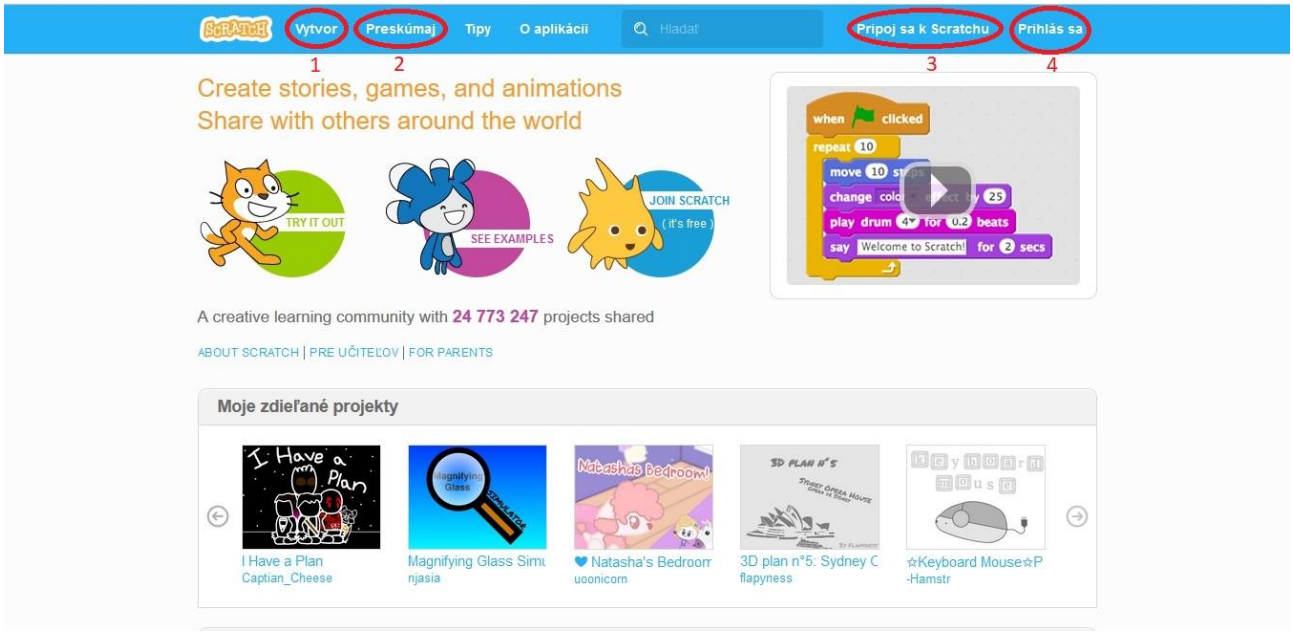
Obrázok 2.1. Prostredie Scratchu.

Táto časť učebnice nadväzuje na informácie opísané v predchádzajúcich častiach, na základný pojmový aparát súvisiaci s programovaním a tiež na opísané programovacie jazyky a programovacie prostredia z kategórie detské programovacie jazyky a na programovanie v nich.

2.1 Ako používať Scratch on-line

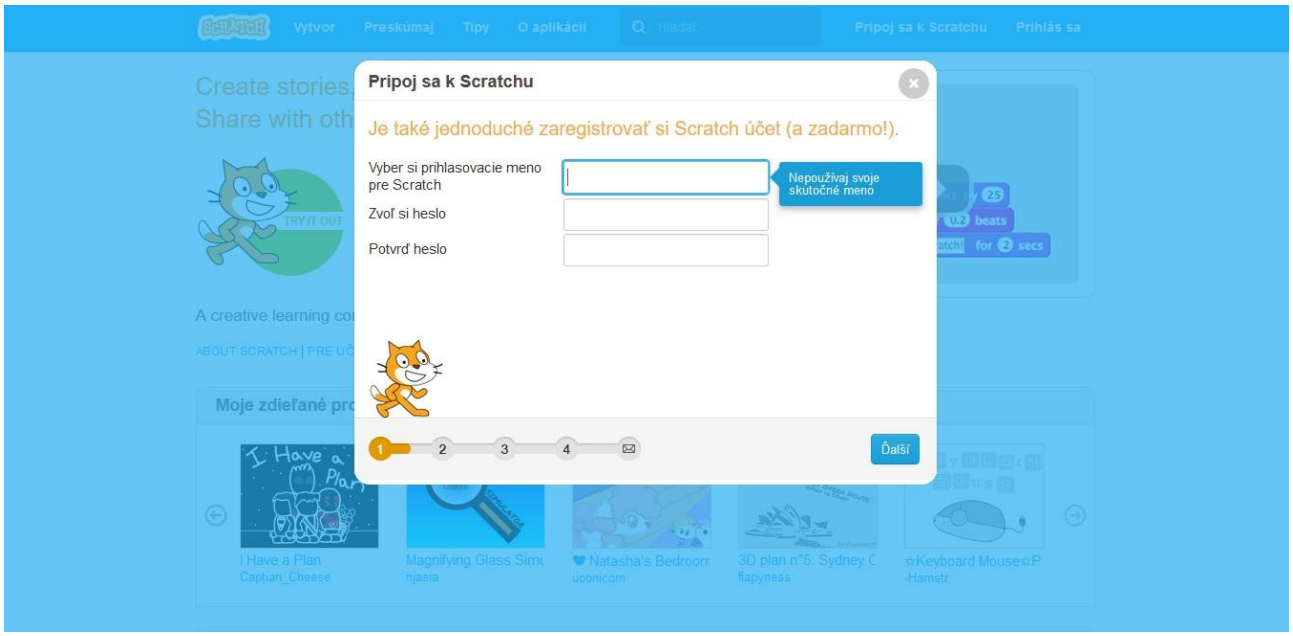
Na používanie Scratchu on-line je v prvom rade potrebné otvoriť stránku https://scratch.mit.edu/ vo webovom prehliadači. Väčšina webových prehliadačov umožňuje bezproblémovú prácu v tomto prostredí.

Na obrázku 2.2 je úvodná stránka Scratchu. Mala by byť preložená do slovenčiny alebo češtiny hneď po spustení, ale ak nebude, tak jazyk sa dá zmeniť aj neskôr. Teraz vysvetlíme, čo sa nachádza na úvodnej stránke.



Obrázok 2.2. Úvodná stránka Scratchu.

1. Voľba *Vytvor* slúži na prepnutie do vývojárskeho prostredia, kde sa dajú tvoriť aplikácie. Dá sa zapnúť aj bez registrácie.
2. *Preskúmaj* umožňuje prezeranie všetkých projektov, ktoré boli vytvorené v prostredí Scratchu.
3. *Pripoj sa k Scratchu* slúži na zaregistrovanie – na vytvorenie nového účtu.
4. *Prihlás sa* slúži na prihlásenie jestvujúceho účtu, v ktorom môže byť napríklad uložený celý projekt.



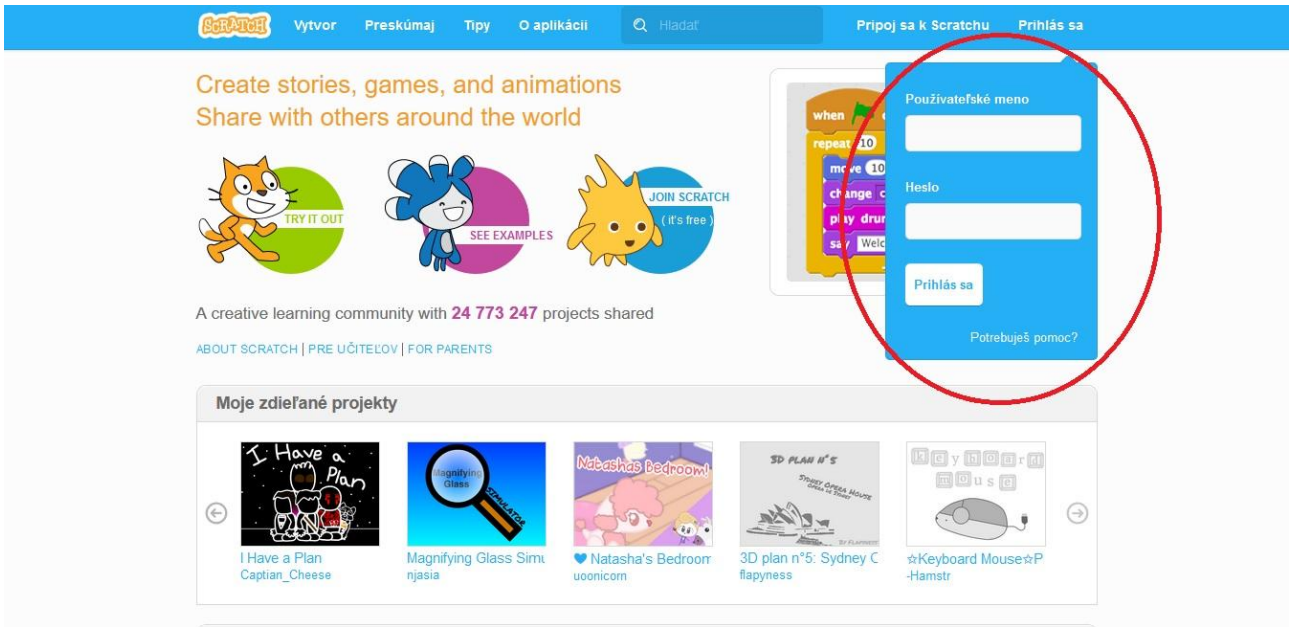
Obrázok 2.3. Registrácia v Scratchi.

Registrácia

Na využitie jednoduchého ukladania projektov, ich zdieľania a využitie ďalších možností je potrebné zaregistrovať sa prostredníctvom ponuky *Pripoj sa k Scratchu.* Pri registrácií je nutné postupovať podľa pokynov, ktoré sa zobrazia v okne. Na úspešné dokončenie registrácie stačí vyplniť údaje, ktoré sa od nás požadujú. Pre možnosť zdieľania projektov treba zadať platnú e-mailovú adresu, ktorú bude potrebné potom potvrdiť.

Prihlásenie sa

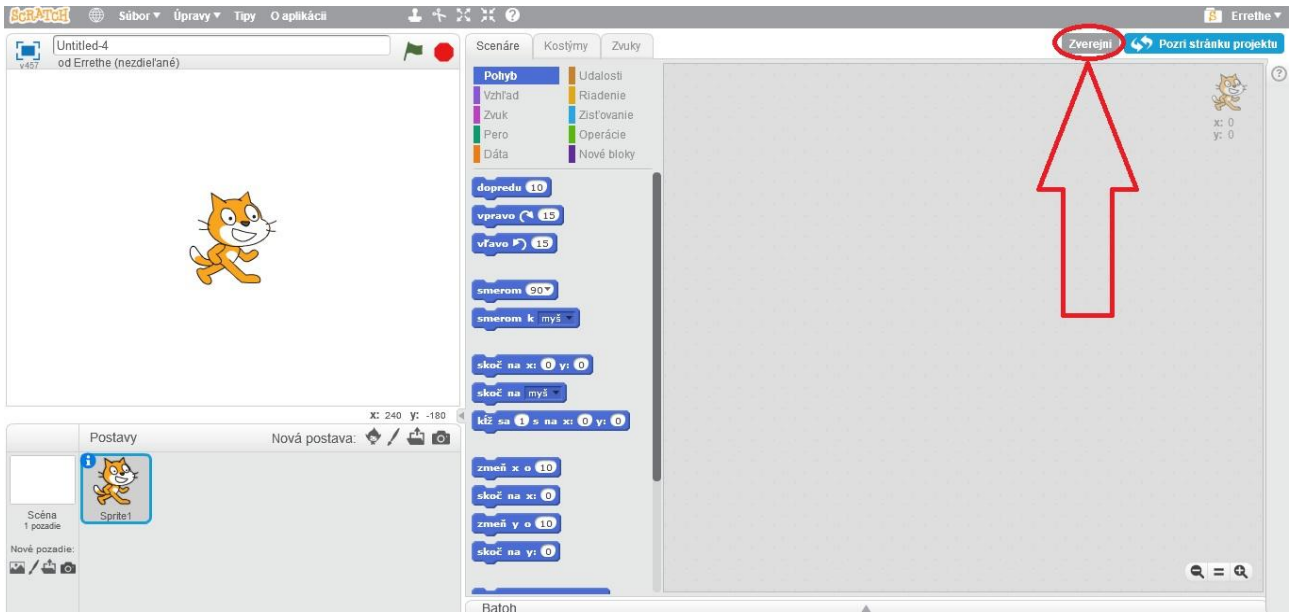
Prihlásenie sa pre registrovaných používateľov je veľmi jednoduché. Stačí kliknúť na *Prihlás sa* a vyplniť prihlasovacie údaje, ktoré sme zvolili pri registrácii. Keď sme prihlásení, nemusíme sa starať ani o ukladanie projektu, celý tento proces je automatický. Projekt sa ukladá do účtu, pod ktorým sme prihlásení.



Obrázok 2.4. Prihlásenie sa v Scratchi.

Zdieľanie projektu

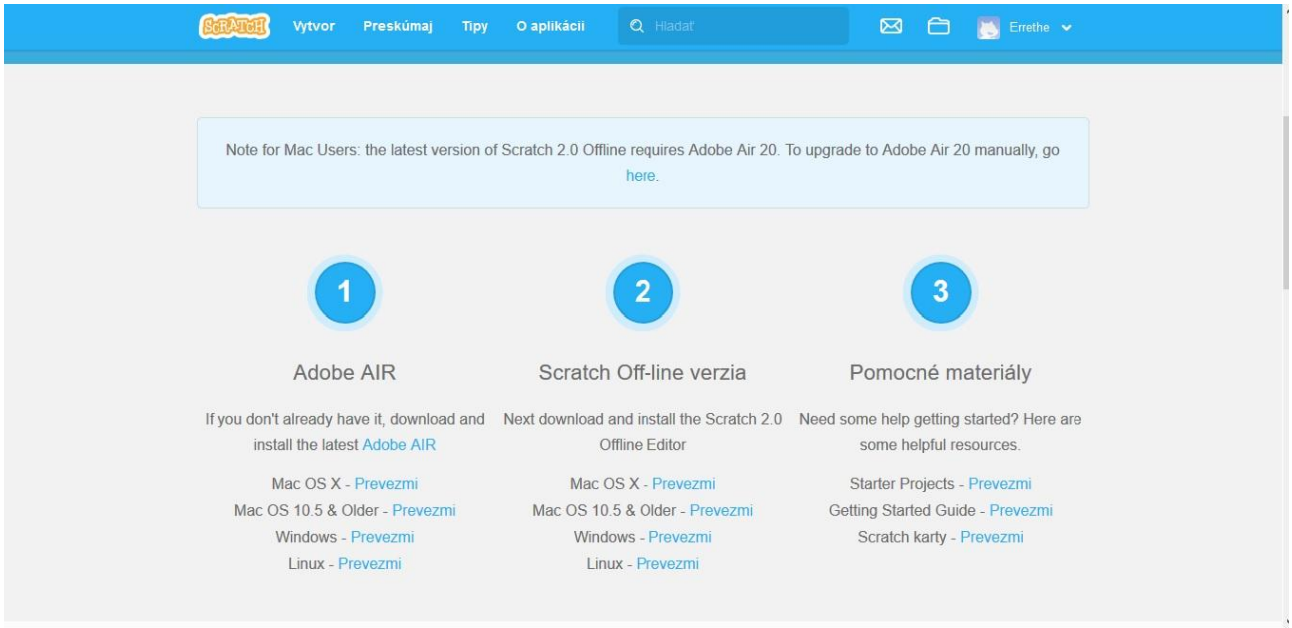
Zdieľanie projektu s ostatnými je po prihlásení jednoduché. Pri vytváraní aplikácie stačí kliknúť na tlačidlo *Zverejni* a každý bude môcť nájsť náš objekt, pozrieť si ho, prípadne ho spustiť a vyskúšať.



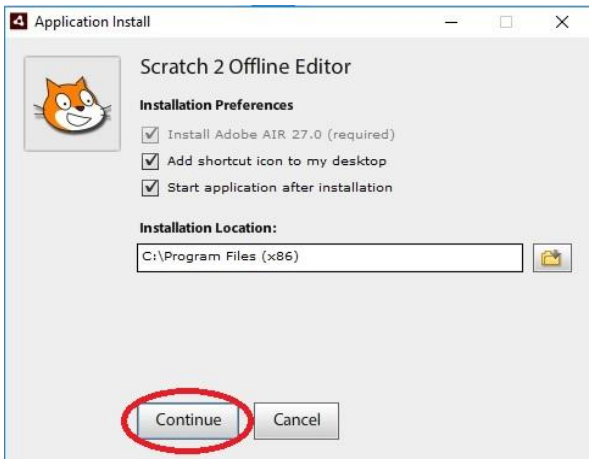
Obrázok 2.5. Zdieľanie projektu.

2.2 Ako používať Scratch off-line

Na používanie Scratchu off-line je potrebné vo webovom prehliadači prejsť na stránku ps://scratch.mit.edu/download a stiahnuť si Scratch a Adobe AIR (pozri obrázok 2.6). Väčšina webových prehliadačov je podporovaná.



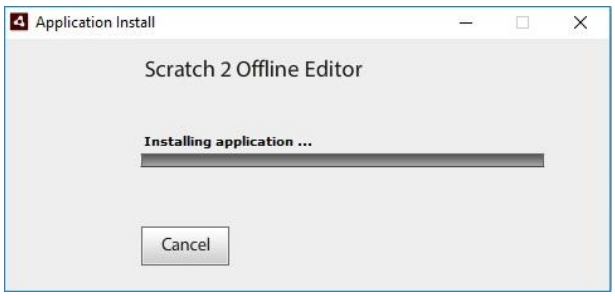
Obrázok 2.6. Používanie Scratchu off-line.



Obrázok 2.7. Inštalácia Scratchu na vlastnom počítači.

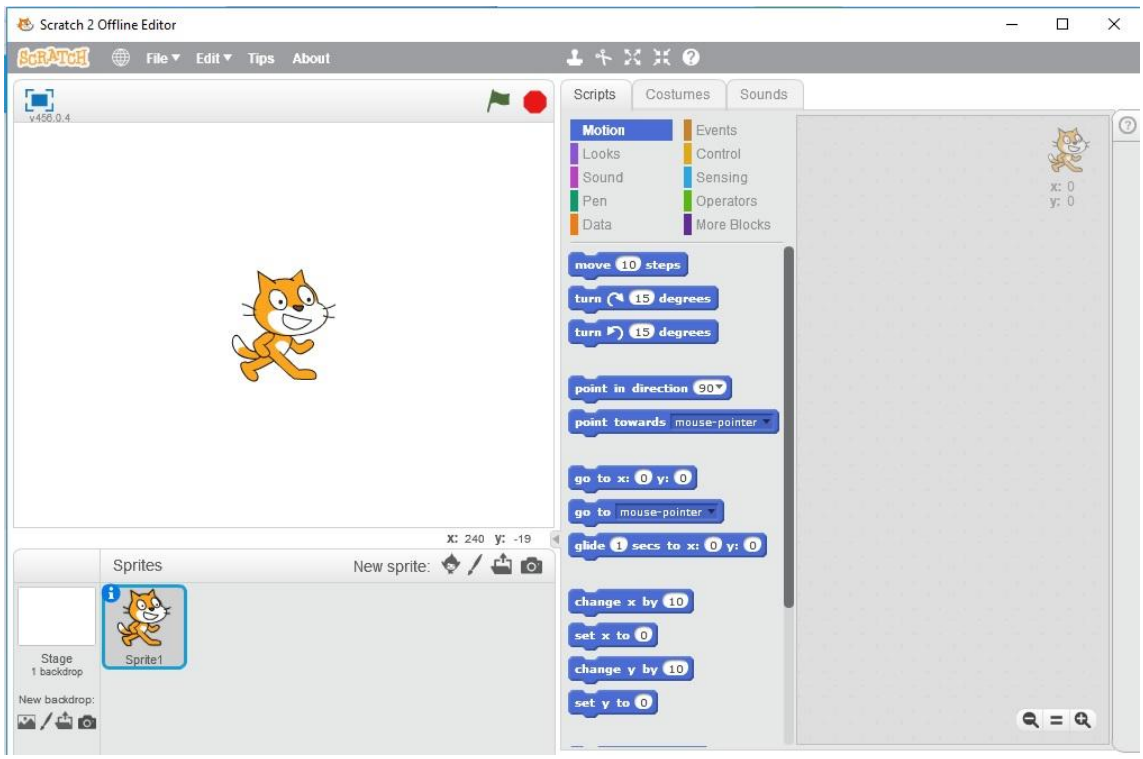
Obrázok 2.8. Licenčné podmienky pre Adobe AIR.

Po spustení stiahnutého súboru inštalácie sa načíta okno napravo. Používateľ si môže vybrať, kam sa má Scratch nainštalovať. Tiež sa dá vybrať z možností ako pridanie ikonky na plochu a možnosť spustenia Scratchu hneď po nainštalovaní. Na pokračovanie v inštalácii treba kliknúť na *Continue.* Ako ďalšie okno sa zobrazia licenčné podmienky pre Adobe AIR, s ktorými je potrebné súhlasiť kliknutím na *I Agree.*



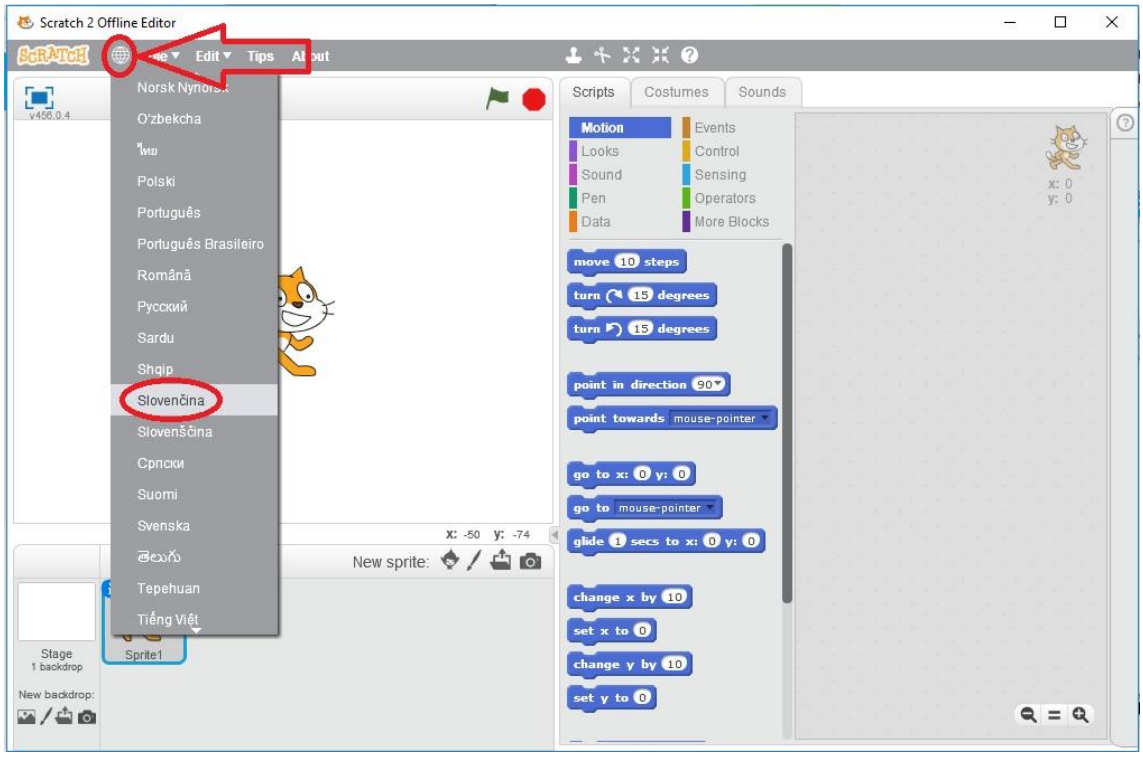
Obrázok 2.9. Dokončenie inštalácie Scratchu.

Počkáme, kým sa inštalácia nedokončí. Po dokončení inštalácie sa Scratchu automaticky spustí a používateľ v ňom môže začať pracovať.



Obrázok 2.10. Editor Scratchu.

Ak je to potrebné, dá sa zmeniť jazyk prostredia na slovenčinu/češtinu tak, že používateľ klikne na „planétu," na ktorú ukazuje šípka, nájde slovenčinu/češtinu a klikne na ňu. Pozri obrázok 2.11.

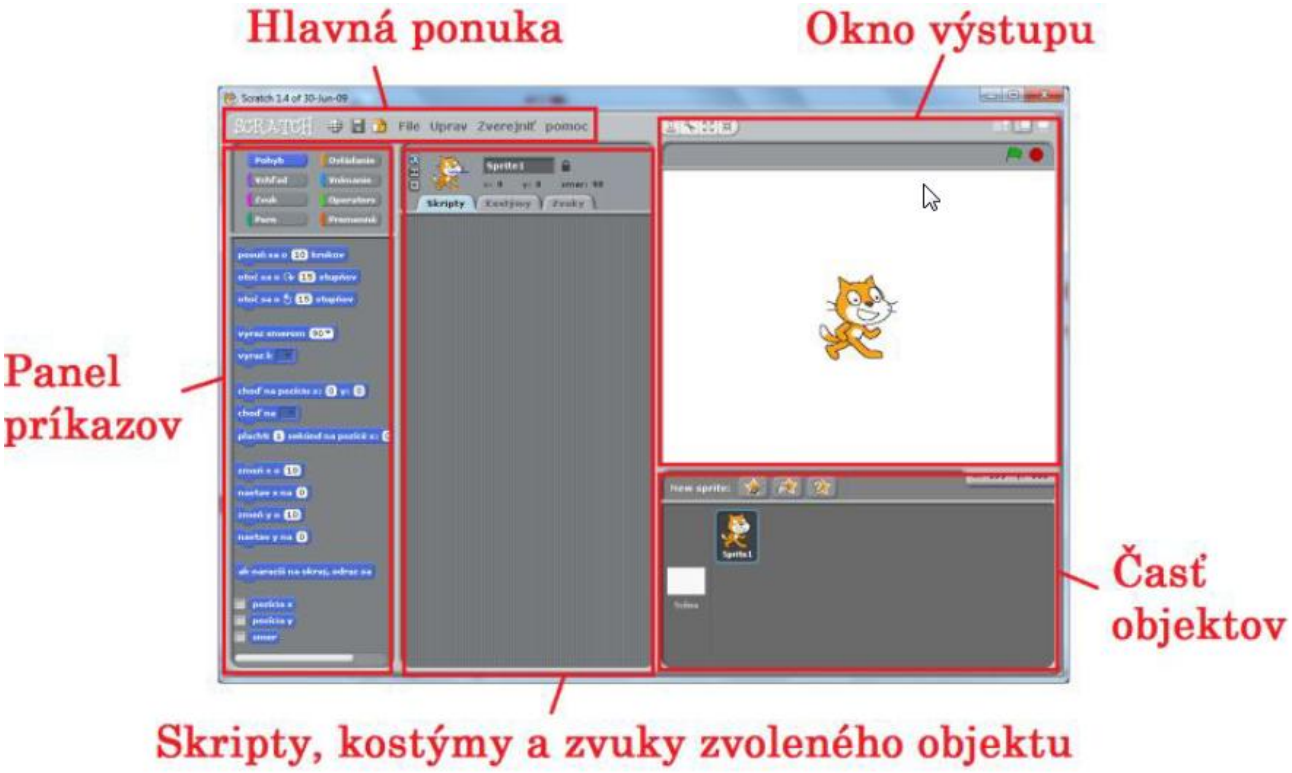


Obrázok 2.11. Zmena jazyka komunikácie.

2.3 Prostredie a príkazy Scratchu

Pre budúceho používateľa – tvorcu aplikácií – je dôležité oboznámiť sa s podrobnou charakteristikou prostredia tohto programovacieho jazyka. Preto v aktuálnej podkapitole sa opisuje prostredie tohto programovacieho jazyka, ako aj príkazy, ktoré môžu byť v ňom interaktívne používané.

Prostredie Scratchu je rozdelené na niekoľko základných častí: *hlavná ponuka, časť príkazov, okno vlastností zvoleného objektu, časť s programom zvoleného objektu, okno výstupu* a *časť objektov.* Toto základné rozdelenie je znázornené na obrázku 2.12.



Obrázok 2.12. Prostredie Scratchu.

2.3.1 Hlavná ponuka

Hlavná ponuka obsahuje ikonu planéty, ktorou je možné meniť jazyk programu. Program je preložený do viacerých svetových jazykov, do češtiny a slovenčiny. Ďalej sa v tejto časti nachádza ikona na uloženie projektu a ikona na zverejnenie projektu na serveri Scratchu. Táto funkcia vyžaduje mať vytvorený bezplatný účet na stránke ps://scratch.mit.edu/.

Nasleduje položka menu *File.* V ponuke *File* sa nachádzajú základné funkcie programu: *otvoriť, uložiť projekt, pridať poznámky k projektu a ukončiť program.* Prostredie Scratch ukladá projekty do súborov vo formáte *.sb. Obrazovka ukladania a otvárania projektu je o to prehľadnejšia, že sa v nej nachádzajú veľké tlačidlá uľahčujúce navigáciu v súborovom systéme počítača, napríklad odkaz na pracovnú plochu, na dokumenty používateľa či na špeciálny priečinok v dokumentoch, ktorý si program vytvorí pri inštalácii.

V položke menu *Uprav* je funkcia späť na vrátenie poslednej zmeny, ďalej sú tu funkcie krokovania programu, možnosť komprimovať obrázky a zvuk a nakoniec je tu voľba na zobrazenie *Motor blocks* v paneli príkazov.

Položka *Zverejniť* umožňuje publikovať vytvorený projekt na stránkach Scratchu a prejsť na ich domovskú stránku. Položka *pomoc* (taký je preklad položky – s malým písmenom) umožňuje prejsť na webové stránky s pomocníkom k programu a zobraziť základné informácie o Scratchi.

2.3.2 Panel príkazov

Panel príkazov obsahuje všetky príkazy, ktoré programovací jazyk Scratch umožňuje používať. Príkazy sú vo výslednom programe pre väčšiu prehľadnosť farebne odlíšené a rozdelené do ôsmich kategórií: *pohyb, vzhľad, zvuk, pero, ovládanie, vnímanie, operátory* a *premenné.* Každá kategória obsahuje príkazy z danej oblasti.

Časť *pohyb* obsahuje príkazy pre otáčanie, posun a určovanie súradníc *x* a *y* zvoleného objektu. Časť vzhľad obsahuje príkazy pre zmenu kostýmu (objektu/obrázka), farby, veľkosti objektu. Ďalej sú tu príkazy na zobrazenie a skrývanie objektu a príkaz *hovor,* ktorý zobrazí nad objekt bublinu s prislúchajúcou správou.

Časť *zvuk* obsahuje príkazy, ktoré umožňujú pridávať zvuky a meniť ich hlasitosť. Scratch obsahuje vlastnú databázu zvukov rozdelenú do kategórií ako zvieratá, efekty, ľudia. Umožňuje nahrať vlastnú zvukovú sekvenciu mikrofónom či importovať do aplikácie už uložený zvuk.

Časť *pero* obsahuje príkazy vymazať plochu, zapnúť a vypnúť pero, meniť hrúbku a farbu pera a opečiatkovať objekt.

Časť *ovládanie* obsahuje iniciačné príkazy (udalosti), príkazy cyklov a podmienok, umožňuje posielať správy medzi objektmi a zastaviť program.

Časť *vnímanie* obsahuje príkazy, ktoré dokážu zistiť, či sa daný objekt dotýka iného objektu alebo farby, ďalej príkazy, ktoré zisťujú farbu podkladu alebo stlačenie klávesy. Okrem nich obsahuje príkazy, ktoré reprezentujú nejakú hodnotu, napríklad súradnicu objektu, hlasitosť zvuku alebo vzdialenosť od iného objektu.

Časť *operátory* (operators) obsahuje matematické operátory, ako je súčet, rozdiel, súčin, podiel, väčší, menší, rovná sa, ďalej sa tu nachádzajú logické operátory *a* a *alebo.* Je tu príkaz na generovanie náhodného čísla zo zvoleného rozsahu, príkazy na spájanie slov a zaokrúhľovanie.

Časť *premenná* obsahuje dve tlačidlá: *vytvoriť premennú* a *vytvoriť zoznam.* Pri ich vytváraní si Scratch vypýta ich názov a objavia sa príkazy pre základné operácie s premennými ako *nastav premennú na, zmeň premennú o* a ikona so samotnou premennou.

Používateľ (učiteľ/žiak) programuje vybraný objekt tak, že z tejto časti príkazov presúva ľavým tlačidlom myši jednotlivé *príkazy* do časti *Skripty.* Odstraňuje ich tak, že ich z časti skriptov presunie späť do časti príkazov.

Príkazy sa odlišujú tvarom. V hornej časti vysunuté príkazy, ktoré sa nachádzajú v časti ovládanie. Sú to takzvané inicializačné príkazy a príkazy vstupu, ktoré môžu stáť na začiatku bloku príkazov. Ide napríklad o príkazy *po stlačení klávesy* (v prostredí je chyba, malo by byť *klávesu*) alebo *po kliknutí na.* Tieto príkazy majú v spodnej časti výstupok, ktorý naznačuje, že zaň môžu byť umiestňované ďalšie príkazy.

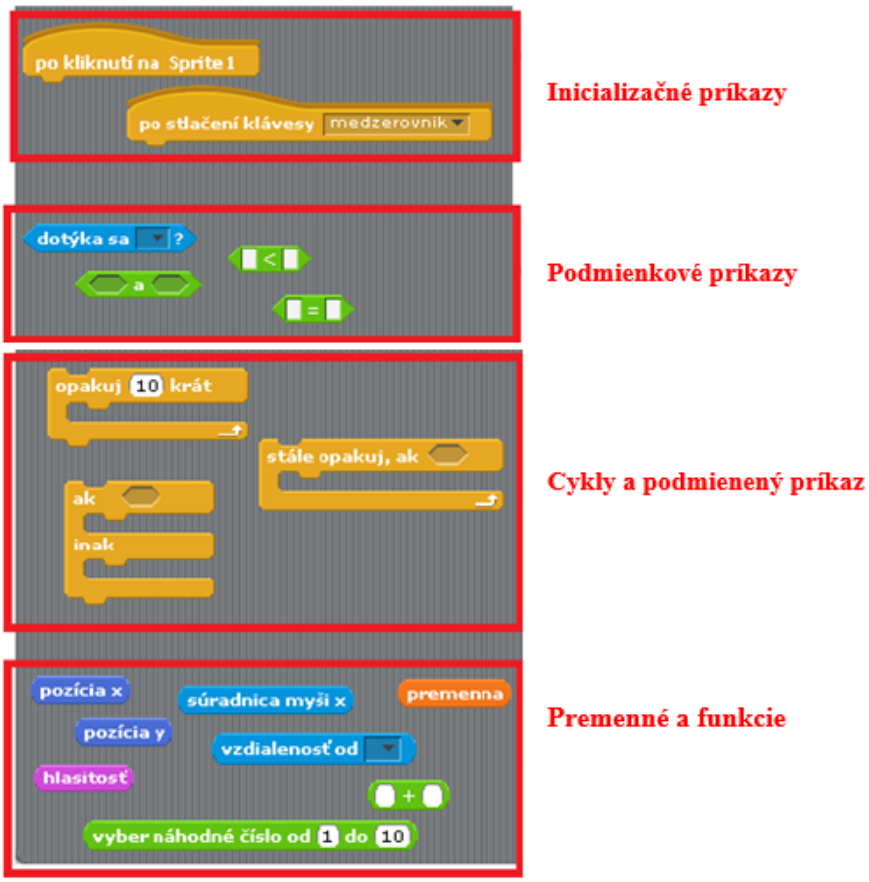
Bežné príkazy majú obdĺžnikový tvar so zaoblenými okrajmi s preliačinou v hornej časti a výstupkom v dolnej časti. Tieto príkazy sa dajú pomocou týchto výstupkov spájať a zoraďovať do programu.

Ďalším typom príkazov sú *podmienky.* Ide vlastne o zjednodušené operátory typu *boolean,* ktoré vracajú hodnotu *pravda* alebo *nepravda.* Tieto príkazy majú ikony na koncoch zašpicatené do tvaru lomených zátvoriek. Tieto príkazy je možné vkladať do iných príkazov, v ktorých je podmienka. Otvor v takýchto príkazoch má rovnako tvarované obrysy ako tieto podmienky, čo uľahčuje skladanie potrebnej programovacej konštrukcie a zároveň potvrdzuje správnosť ich používania.

V Scratchi sa nachádzajú štandardné matematické operátory ako väčší (>), menší (<), rovná sa (=). Ďalej sú tu logické operátory *a, alebo* a *nie je* (ekvivalenty k *and, or* a *not*) a špeciálne podmienky ako *dotýka sa,* pri ktorej treba nastaviť objekt, a *dotýka sa farby,* kde treba nastaviť prislúchajúcu farbu.

Ďalším typom príkazov sú *cykly a podmienka (if).* Tieto príkazy sa od ostatných odlišujú tým, že majú tvar akýchsi „klieští," do ktorých je možné vkladať príkazy a tak vytvárať bloky príkazov. Tieto „kliešte" môžu obsahovať len jeden blok príkazu, ako napríklad príkazy *opakuj, opakuj n-krát* (cyklus *for*), *stále opakuj, ak* (cyklus *while*), *ak* (podmienka *if*) alebo *opakuj, pokiaľ nenastane* (cyklus *repeat-until*). V posledných troch treba do príkazu vložiť podmienkové operátory (podmienku opakovania, či zastavenia opakovania). Zložitejší je úplný príkaz *if,* kde pribúda vetva *else,* čiže obsahuje dva bloky príkazov.

Posledným typom príkazov v tejto časti sú *premenné alebo príkazy,* ktoré reprezentujú určitú hodnotu získanú napríklad výpočtom alebo vzdialenosťou objektov. Ikony týchto príkazov tvoria obdĺžniky so zaoblenými okrajmi bez akýchkoľvek výstupkov či preliačin. Okrem premenných sú tu špeciálne príkazy, ktoré reprezentujú súradnicu *x* alebo *y* objektu alebo myši, hlasitosť, vzdialenosť zvoleného objektu od iného či výber náhodného čísla z určeného rozsahu. Špeciálnym typom príkazov sú matematické funkcie (operátory), ktoré vracajú súčet, rozdiel, súčin alebo podiel dvoch čísel. Do tejto časti patria tiež príkazy *mod* na výpočet zvyšku po delení dvoch celých čísel, príkaz na zaokrúhlenie čísla a príkaz na výpočet hodnoty funkcie, ktorú sa nachádzajú v rolovacom menu. Ponuka obsahuje základné goniometrické a logaritmické funkcie.



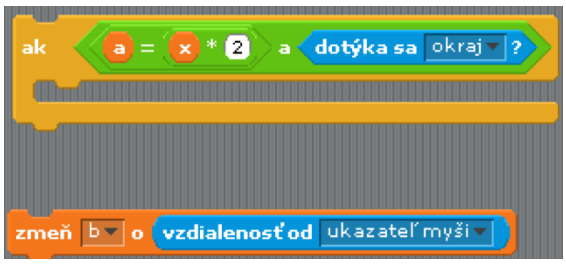
Obrázok 2.13. Základné príkazy prostredia Scratch.

Obrázok 2.13 demonštruje vyššie opísané príkazy v programovacom jazyku Scratch. Viaceré príkazy sa dajú upravovať, doladiť, spresniť určením hodnôt parametra príkazu. V príkaze *posuň sa o 10 krokov* sa dá kliknutím vstúpiť do časti s číslom a zadať inú hodnotu – zmeniť hodnotu desať na inú. V príkaze *hovor* sa takýmto spôsobom dá vpísať text, ktorý má postava povedať, resp. ktorý sa má objaviť v bubline nad postavou.

Niektoré príkazy obsahujú rolovacie menu, kde je možné vybrať si z určitého počtu objektov. Napríklad inicializačný príkaz *po stlačení klávesy* (v prostredí je chyba, malo by byť *klávesu*), kde treba vybrať prislúchajúci kláves, alebo podmienka *dotýka sa,* kde treba vybrať daný objekt, kurzor myši či okraj.

Špeciálnym prípadom úpravy príkazu je skladanie, čo znamená, že do jedného príkazu sa dá vložiť ďalšia podmienka alebo príkaz, ktorý sa opäť môže skladať z viacerých príkazov (v zmysle štruktúrovaného programovania). Napríklad na obrázku 2.14 je podmienka, ktorá platí, ak sa *a* rovná dvojnásobku *x* a súčasne sa daný objekt dotýka okraja. Možností je skutočne veľa a je len na fantázii a šikovnosti používateľa, ako jednotlivé „stavebné elementy“ využije na dosiahnutie stanovaného cieľa.

Interaktívne skladanie príkazov a tvary jednotlivých ikoniek uľahčujú skladanie a tiež čitateľnosť programov. Nie je náročné pochopiť ani programátorovi začiatočníkovi, čo to je vlastne programovanie.



Obrázok 2.14. Zložitejšie príkazy.

2.3.3 Skripty, kostýmy a zvuky

Strednú časť obrazovky prostredia Scratch tvorí časť skriptov, kostýmov a zvukov zvoleného objektu. Táto časť sa teda mení podľa toho, ktorý objekt máme označený v časti objektov. Žiakom to treba opakovane zdôrazňovať a pripomínať, lebo sa často stáva, že si nevšimnú, ktorý objekt majú označený a „programujú“ (upravujú) iný, než chceli.

Táto časť programu má v hornej časti základné vlastnosti zvoleného objektu: jeho meno, súradnice x a y a smer natočenia. Meno objektu je možné meniť. Poloha $x \leftarrow 0$ a $y \leftarrow 0$ sa nachádza v strede okna výstupov. Smer natočenia je definovaný stupňami, kde 0° je smerom nahor, 90° je vpravo 180° je dole a 270° (alebo −90°) je vľavo. Vľavo od názvu objektu je jeho obrázok s modrou čiarou, ktorá naznačuje, ktorým smerom je objekt otočený. Vľavo od obrázka sú tri tlačidlá, ktoré určujú, akým spôsobom sa bude správať obrázok objektu pri jeho otáčaní: *možné otočiť, only face left-right* a *neotáčať.* Voľba *only face left-right* znamená, že sa obrázok objektu bude otáčať len zrkadlovo vľavo a vpravo (90° a −90°).

Prvá záložka, *skripty,* predstavuje priestor, do ktorého ľavým tlačidlom myši presúvame jednotlivé príkazy z panela príkazov, a tak vytvárame program/riadenie zvoleného objektu. Pri spájaní príkazov do postupnosti treba mať na pamäti, že pripájame spodný príkaz k hornému. Ak chceme takéto príkazy oddeliť, tak jedine odpojením spodného od horného. Ak uchopíme horný príkaz, tak sa budú presúvať oba príkazy naraz ako jeden blok a neoddelia sa. Ako už bolo spomínané, príkazy z tejto časti odstránime tak, že ich presunieme naspäť do panela príkazov.

Záložka *kostýmy* obsahuje zoznam obrázkov, do ktorých môže byť daný objekt „prezlečený.“ Je to vhodné napríklad vtedy, keď je žiadúce postavičku pri pohybe animovať. Vtedy jej v tejto časti treba prideliť dva alebo viacero na seba nadväzujúcich obrázkov a objektu nastaviť, aby ich pri pohybe striedal.

V hornej časti sa nachádzajú tri tlačidlá, ktorými sa kostýmy dajú pridávať: *Maľba, Import* a *Camera.* Tlačidlo *Maľba* spustí vstavaný editor obrázkov, kde je možné nový obrázok nakresliť. Tento editor obsahuje nástroje bežne dostupné v podobných softvéroch: štetec, guma, výplň plochy, čiary, vkladanie textu, možnosť obrázok zväčšovať a zmenšovať, otáčať a prevracať. Pomocou tlačidla *Set costume center* je možné nastaviť stred kostýmu, okolo ktorého sa bude otáčať. Prostredie vstavaného editora obrázkov je na obrázku 2.15.

Obrázok 2.15. Vstavaný/vnútorný editor obrázkov.

Tlačidlo *Import* umožňuje načítať obrázok zo súboru na pevnom disku počítača. Prostredie Scratch obsahuje databázu kostýmov rozdelených do kategórií: *Animals, Fantasy, Letters, People, Things* a *Transportation,* čiže Zvieratá, fantasy, písmená, ľudia, veci a dopravné prostriedky.

Je možné importovať akýkoľvek obrázok uložený v niektorom z bežných grafických formátov, ako je napríklad BMP alebo JPEG. Tretím tlačidlom v tejto časti je Camera, ktoré umožňuje vyfotografovať obrázok pomocou webovej kamery.

Pri každom z kostýmov je jeho názov, tlačidlo uprav, ktoré otvára editor obrázkov, tlačidlo kopíruj, ktoré kostým zduplikuje, a krížik, ktorý daný kostým odstráni. Pri každom je tiež uvedené jeho rozlíšenie v pixeloch a koľko pamäte zaberá.

Záložka *zvuky* obsahuje zoznam zvukov priradených vybranému objektu. Každý zvuk má svoje meno, tlačidlo na prehratie, zastavenie a odstránenie. Pri každom zvuku sú uvedené informácie o jeho trvaní a pamäti, ktorú zaberá. Ďalej sa tu nachádza tlačidlo záznam, pomocou ktorého je možné nahrať zvuk prostredníctvom mikrofónu, a tlačidlo import, ktoré otvorí už spomínanú databázu zvukov alebo umožňuje načítať zvuk z pevného disku počítača.

2.3.4 Okno výstupu

V tejto časti prostredia Scratch sa realizuje výstup na obrazovku. Skladá sa z pozadia, samotných objektov a niekoľkých tlačidiel. Nachádza sa tu tlačidlo na spustenie programu (zelená vlajočka) a tlačidlo na zastavenie programu (červený osemuholník). Ďalej sú tu tlačidlá na zmenu zobrazenia: *Switch to small stage* – zmenší okno výstupu a časť objektov, *Switch to full stage* – štandardné zobrazenie a *Switch to presentation mode* – okno výstupu sa zobrazí na celú obrazovku.

Pri celoobrazovkovom zobrazení zostanú v hornej časti viditeľné len tlačidlá na spustenie, zastavenie programu a tlačidlo na návrat do predchádzajúceho zobrazenia. Nad oknom výstupu sa nachádzajú štyri tlačidlá na prácu s objektami: Duplicate, Vymaž, Grow sprite a Shrink sprite. Ide o nástroje na duplikáciu, vymazanie, zväčšovanie a zmenšovanie objektu. Duplikovať, vymazať, meniť veľkosť a otáčať objekt je možné aj pomocou ponuky,

ktorá sa zobrazí po kliknutí pravým tlačidlom na objekt. V pravej dolnej časti sa nachádzajú aktuálne súradnice polohy kurzoru myši vzhľadom na stred scény.

2.3.5 Časť objektov

Poslednou funkčnou časťou programu je časť objektov. Je tu vyobrazená ikona scény a všetkých ostatných objektov daného projektu. Podľa toho, ktorý objekt máme označený, sa mení panel skriptov, kostýmov a zvukov v strednej časti obrazovky. To znamená, že každý objekt má priradené vlastné skripty, kostýmy a zvuky.

Ďalej sa tu nachádza tlačidlo na nakreslenie nového objektu, ktoré otvorí už spomínaný vstavaný editor obrázkov. Vedľa je tlačidlo na výber kostýmu z počítača. Toto tlačidlo je prednastavené tak, aby otvorilo priečinok s databázou obrázkov prostredia Scratch, ale je možné otvoriť akýkoľvek obrázok uložený na pevnom disku počítača. Posledné tlačidlo v tejto časti nesie popis *Get surprise sprite,* ktoré po kliknutí vyberie náhodný obrázok z databázy.

2.4 Otázky a úlohy

Otázky na zopakovanie

1. Čo je to Scratch a pre koho je určený?
2. Aké typy projektov je možné vytvárať v Scratchi?
3. Aký je rozdiel medzi on-line a off-line verziou Scratchu?
4. Na čo slúži tlačidlo **Vytvor** na stránke Scratchu?
5. Vymenuj aspoň tri výhody registrácie na stránke Scratchu.
6. Ako je rozdelené prostredie Scratchu? Vymenuj jeho hlavné časti.
7. V akých kategóriách sú rozdelené príkazy v Scratchi?
8. Čo znamená to, že niektoré príkazy majú „kliešte“?
9. Aký je rozdiel medzi príkazmi **po stlačení, opakuj** a **ak...inak?**
10. Čo sú **skripty, kostýmy** a **zvuky** objektu v Scratchi?
11. Akú funkciu má **zelená vlajočka** a **červený osemuholník** v okne výstupu?
12. Čo sa stane, keď označíš iný objekt v časti objektov?
13. Ako môžeš zmeniť jazyk prostredia v Scratchi?
14. Ako sa zmení správanie objektu, ak vyberieš voľbu **only face left-right?**

Úlohy na precvičenie

1. **On-line verzia Scratchu:** Otvor stránku https://scratch.mit.edu/, klikni na **Vytvor** a zisti, v akom jazyku sa prostredie otvorilo. Ak nie je v slovenčine alebo češtine, prepni jazyk.
2. **Registrácia (simulácia):** Prejdi si krok za krokom, čo je potrebné na vytvorenie účtu na stránke Scratchu. (Nemusíš sa registrovať, len opíš kroky.)

3. **Prieskum prostredia:** Otvor editor Scratchu a nájdi: *panel príkazov, okno výstupu, skripty, časť objektov.*
4. **Farby príkazov:** Ktorá kategória príkazov je modrá? Ktorá oranžová? A ktorá zelená?
5. **Programátorský tréning:** Vytvor jednoduchý program, ktorý po kliknutí na vlajku povie *„Ahoj, svet!“* a posunie sa o 50 krokov dopredu.
6. **Cyklus a podmienka:** Vytvor program, v ktorom postavička bude stále opakovať pohyb, kým sa nedotkne okraja.
7. **Práca s kostýmami:** Vyber jednu postavičku, pridaj jej aspoň dva kostýmy a vytvor program, ktorý ich bude striedať pri pohybe.
8. **Zmena pozadia:** Pridaj pozadie z databázy Scratchu a vytvor jednoduchú scénku s dvoma objektmi, ktoré sa pohybujú rôznymi smermi.
9. **Zvukový sprievod:** Pridaj k postavičke zvuk z databázy a vytvor program, ktorý ho prehrá pri stlačení klávesu.
10. **Zložitejšie zadanie:** Vytvor program, ktorý spraví toto: Postava sa pohne na súradnice $x = -100$, $y = 0$; povie *„Poďme na to!“;* spraví tri skoky dopredu (50 krokov každý); zmení kostým; prehrá zvuk; a ak sa dotýka okraja, tak povie *„Koniec!“*

3 Riešenie jednoduchých úloh

V tejto časti uvedieme riešenie niekoľkých jednoduchých úloh, ktoré môžu poslúžiť ako motivácia ako začať pracovať, tvoriť jednoduché aplikácie – teda programovať v prostredí Scratch. Úlohy, ktoré riešime sú takéto: ako realizovať vstup z klávesnice, ako fungujú príkazy cyklus a podmienka a ako používať premennú v programe. Jednotlivé úlohy môžu tvoriť jednu vyučovacie jednotku. Záleží len na fantázii učiteľa, prípadne mladých tvorcoch, ako budú realizovať vlastné riešenia.

3.1 Ovládanie postavičky na obrazovke s použitím vstupu z klávesnice

Cieľom je vytvoriť jednoduchý program s použitím vstupu z klávesnice na ovládanie objektu na obrazovke. Motiváciou môže byť vytvoriť vlastnú hru, v ktorej chcú vyviesť postavičku z bludiska.

Pohybovať objektom v prostredí Scratch nie je náročné. Deti riešenie takejto úlohu bez problémov zvládajú. Treba im len na príklade pripomenúť zásady tvorby programu a ukázať/pripomenúť im, ako funguje príkaz *po kliknutí na* a program spúšťajú kliknutím na zelenú vlajočku. Tiež je vhodné na úvod zopakovať základné príkazy na ovládanie postavy.

Napríklad jednoduchý program na nakreslenie jednoduchého obrázka – napríklad štvorca. Program môže vyzerať tak ako na obrázku 3.1.

Žiakov treba neustále upozorňovať, aby mali v časti objektov kliknuté na panáčika, ktorého programujú.

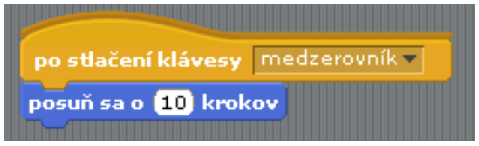
Teraz už len poukážeme na to, v čom sa líši naša úloha od predchádzajúcej, že potrebujeme panáčika ovládať pomocou klávesnice.

Obrázok 3.1. Program na nakreslenie štvorca.

Je potrebné pripomenúť žiakom, čo je to vstup a čo je to výstup. Vieme, že výstupom počítačového programu je to, čo sa objaví na obrazovke, v našom prípade výstupom bol nakreslený štvorec. Jediným vstupom bolo spustenie programu kliknutím na zelenú vlajočku. Bludiskové hry, ktoré žiaci poznajú, napríklad Pac-Man, Dyna Blaster a pod., sa ovládajú pomocou kurzorových klávesov na klávesnici.

Hry, o ktorých bolo hovorené na začiatku, nestačí len spustiť. Hráč musí stále stláčať nejaké „tlačidlá“ (klávesy) na ovládanie postavičky. Čiže počas hry na ovládanie postavičky sú používané niektoré klávesy (najčastejšie kurzorové) z klávesnice. Teraz bude ukázané, ako môže byť naprogramovaný vstup z klávesnice. Na ovládanie postavičky pomocou klávesnice bude používaný príkaz *po stlačení klávesy* (v prostredí je chyba, malo by byť *klávesu*), v ktorom musí byť nastavený prislúchajúci kláves. Pri stlačení tohto klávesu sa vykonajú príkazy, ktoré budú k danému príkazu priradené.

Nižšie je jednoduchý ukážkový program na pohyb pri stlačení medzerníka (v prostredí je chyba – je tam „medzerovník“). Na obrázku 3.2 sú bloky, ktoré spôsobia, že *po stlačení medzerníka* sa panáčik pohne o 10 bodov tým smerom, ktorým je práve orientovaný.



Obrázok 3.2. Príkaz: po stlačení klávesu medzerník.

Teraz už vieme, aký príkaz použiť a ako naprogramovať vstup z klávesnice. Pomocou tohto príkazu potrebujeme naprogramovať ovládanie postavičky šípkami. Predstavme si, že chceme panáčika, ktorý sa bude pohybovať štyrmi smermi: vľavo, vpravo, hore a dole. Aké príkazy na to budeme potrebovať? Využijeme na vstup kurzorové klávesy:

Zvolíme v príkaze *po stlačení klávesy* (v prostredí je chyba, malo by byť *klávesu*) s voľbu *šípka hore.*

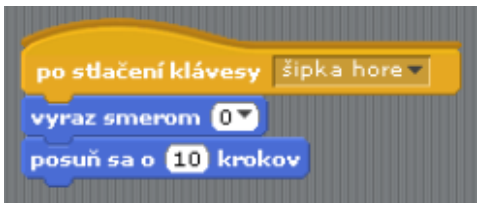
Aké príkazy treba mať v udalosti pri stlačení šípky hore?

1. Treba natočiť panáčika správnym smerom.
2. Treba sa posunúť vpred o určitý počet bodov.

Častým problémom býva to, že si žiaci neuvedomujú, že nejestvuje žiadny príkaz „hore.“ Postavička sa dá len *otáčať okolo svojho stredu* alebo *posúvať vpred či vzad.* To sú štandardné príkazy, ktoré máme k dispozícii a z ktorých treba poskladať/naprogramovať pohyb postavičky.

Obrázok 3.3. Ukážky kurzorových klávesov – „šípok.“

Učiteľ oboznámi žiakov s príkazom *vyraz smerom.* Tento príkaz natočí postavičku určitým smerom, kde 0° je hore, 90° je vpravo, 180° je dole a 270° (alebo −90°) je vľavo. Teraz už máme všetko k dispozícii a vieme dokončiť príkaz pre stlačenie šípky hore:



Obrázok 3.4. Príkaz: po stlačení klávesu šípka hore (otoč sa do smeru „hore“ a posuň sa o 10 krokov).

Ostatné príkazy na ovládanie postavičky už žiaci naprogramujú bez problémov. Uvedomia si, ktorá časť programu sa mení (a ako sa mení) pre jednotlivé klávesy. Je to príkaz *vyraz smerom.*

Poznámka: Žiakov treba upozorniť na to, že v Scratchi je *šípka vľavo* zle preložená a prostredie uvádza jej označenie ako *zarovnanie vľavo.*

Výsledný program bude vyzerať ako na obrázku 3.5.

Obrázok 3.5. Výsledný program: ovládanie postavičky z klávesnice.

Žiaci na konci hodiny svoje projekty nahrajú, aby mohli v nich na ďalšej hodine pokračovať.

3.2 Cyklus a podmienka v programe

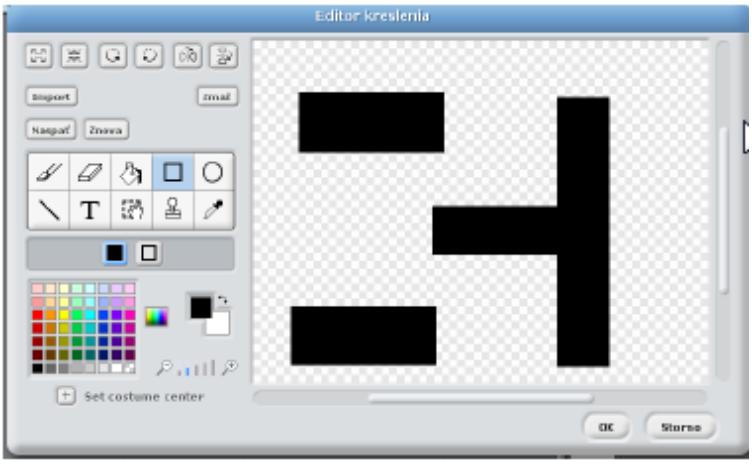
Cieľom tejto časti bude pokračovať tvorbe našej hry. Máme už program na ovládanie postavičky. Aktuálnou úlohou bude vytvoriť jednoduchý program na nakreslenie bludiska na obrazovke (pre našu hru). Motiváciou môže byť vytvoriť vlastnú hru, v ktorej chceme vyviesť postavičku z bludiska. Teraz toto bludisko vytvoríme. Edukačným cieľom je pochopiť, ako fungujú v programe vetvenie a cyklus. Žiak má porozumieť využívaniu cyklu (teda opakovaniu činnosti) a podmienkam na zabránenie prechodu stenou bludiska.

Na začiatku práce žiaci s učiteľom zrekapitulujú, čo sa naučili na minulej hodine pri programovaní ovládania postavičky z klávesnice. Spoločne sa dohodnú, čím by mali pokračovať v tvorbe hry *bludisko.*

Žiaci si zapnú počítače, prihlásia sa a otvoria si projekt z minulej hodiny s ovládaním postavičky pomocou kurzorových klávesov. Ovládanie postavičky máme k dispozícii a teraz musíme nakresliť bludisko. Bludisko nakreslíme ako nový objekt pomocou tlačidla Kresliť novú animáciu v časti New sprite v pravej dolnej časti obrazovky.

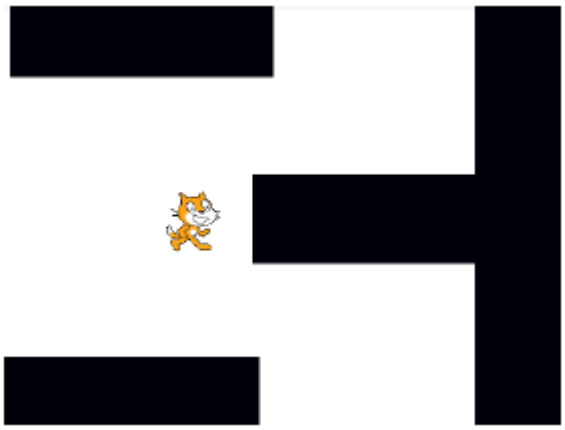
Po voľbe sa otvorí vstavaný editor obrázkov. Žiakov treba viesť k tomu, aby bludisko kreslili čo najjednoduchšie. Bludisko nakreslíme pomocou nástroja obdĺžnik. Pre naše potreby bude stačiť niekoľko jednoduchých stien. Napríklad ako je to na obrázku 3.6.

Učiteľ dozrie na to, aby žiaci v bludisku nechávali dostatočné priestor na pohyb postavičky a súčasne aby neboli steny bludiska príliš tenké, pretože by ich postavička mohla „preskočiť." Po dokončení bludiska stlačíme tlačidlo OK.



Obrázok 3.6. Príklad bludiska vytvoreného z obdĺžnikov.

Veľkosť bludiska sa dá upraviť tak, že naň klikneme pravým tlačidlom myši a zvolíme *resize sprite.* Objaví sa ikonka s dvojšípkou a pomocou nej roztiahneme bludisko na celú plochu.



Obrázok 3.7. Výsledok zväčšenia bludiska a zmenšenia postavičky.

Môže sa stať, že postavička je pre bludisko príliš veľká. Úlohu teraz je zmenšiť ju rovnakým spôsobom, akým sme bludisko zväčšovali. Výsledok môže vyzerať podobne ako na obrázku 3.7.

Teraz je na čase skúsiť ako sa pohybovať v bludisku. V programe už sú dva objekty: panáčik a bludisko. Aby sa neplietli, budú premenované na *Hrdina* a *Bludisko.*

Postup:

Klikneme na prvý objekt (Sprite1) a premenujeme ho v hornej časti na Hrdina. Podobne premenujeme bludisko (Sprite2) na Bludisko.

Žiaci sa zoznamujú s konceptom objektov. Na daný objekt treba najprv kliknúť v pravej dolnej časti a až potom sa nám zobrazia jeho vlastnosti a program. Na daný objekt teda musia najprv kliknúť a až potom ho môžu premenovať. Učiteľ dôsledne kontroluje splnenie úlohy, pritom spoločne rozmýšľajú, čo v programe chýba, ako musíme program vylepšiť atď.

Žiaci napríklad zistia, že postavička prechádza stenami bludiska. Musíme teda naprogramovať, aby sme postavičke zabránili v prechádzaní stenami bludiska. Na to použijeme nekonečný cyklus a rozhodovanie. Program musíme vždy spúšťať zelenou vlajočkou. Nezabúdajme pri tom, že keď programujeme Hrdinu, tak na neho musíme kliknúť v pravej dolnej časti.

Do programu vložíme príkaz *po kliknutí na* so zelenou vlajočkou z časti *Ovládanie.*

Príkaz na nekonečný cyklus sa nachádza v tej istej časti. Ide o príkaz *stále opakuj, ak.*

K inicializačnému príkazu bude pripojený príkaz *stále opakuj, ak.* Teraz je ešte potrebné rozhodovanie. Upriamujeme pozornosť na to, že príkaz *stále opakuj, ak* obsahuje prázdny otvor. Sem treba vložiť podmienku *dotýka sa* z časti *Vnímanie.*

V príkaze *dotýka sa* máme rolovacie menu, kde zvolíme *Bludisko.*

Teraz vyskúšajme, ako bude program fungovať po spustení zelenou vlajočkou.

Učiteľ vysvetlí žiakom, že počítač/program stále kontroluje, či sa náš Hrdina dotýka Bludiska. Teraz žiaci potrebujú naprogramovať, čo sa stane, keď sa Hrdina dotkne Bludiska. Čo vlastne má urobiť Hrdina, keď sa dotkne Bludiska?

Má sa zastaviť – znie správna odpoveď. Učiteľ musí vysvetliť žiakom, že pri dotyku bludiska musíme hrdinu vrátiť na predchádzajúcu polohu. Scratch neobsahuje príkaz na cúvanie, preto použijeme pohyb dopredu o záporný počet krokov. Zapracovaním tohto príkazu máme vytvorenú výslednú podobu programu, ako je to na obrázku 3.8.

Obrázok 3.8. Výsledný program.

Po dokončení programu žiaci spustia program pomocou zelenej vlajočky a pohybujú Hrdinom v Bludisku, pričom sa uisťujú, že Hrdina neprejde cez steny. Učiteľ opakuje

žiakom, ako program funguje: počítač neustále kontroluje, či sa Hrdina nedotýka Bludiska (steny Bludiska) a ak áno, cúvne. Zdôrazňuje, že bol použitý cyklus na opakovanie a rozhodovanie (podmienka) na zisťovanie dotyku.

Žiaci ukončia program (projekt) červeným osemuholníkom a uložia projekty, aby s k nim mohli vrátiť pri najbližšej príležitosti a pokračovať v práci s nimi.

3.3 Používanie premenných v programe

Cieľom je ukázať žiakom, ako sa v programe dajú používať premenné. Po absolvovaní vyučovacej jednotky žiak rozumie použitiu premennej na uloženie hodnoty. Motiváciou je vylepšiť funkčnosť vlastnej hry o ďalšiu možnosť – o chytanie myši.

Učiteľ na zvýšenie motivácie premietne obrázok z hry Pac-Man (prípadne inej známej hry na podobnom princípe). Diskutuje so žiakmi o tom, ako sa táto hra hrá, aké má vlastnosti, čo je výsledkom hry, čo je cieľom hry, ako je hra ukončená, ako je hodnotený výkon hráča a podobne.

Žiaci odpovedajú, že hlavná postavička sa pohybuje v bludisku, vyhýba sa nepriateľom a zbiera body. V prípade, že žiaci hru nepoznajú, môže učiteľ v krátkosti predviesť hranie hry na niektorej z on-line verzií tejto hry. V tomto prípade učiteľ vysvetlí, kedy je splnený cieľ hry. Cieľom hry v tomto prípade je pozbierať všetky body v bludisku. Hra sa teda končí vtedy, keď sú všetky body vyzbierané. Kedy sa končí hra, ktorú sme naprogramovali?

Na naprogramovanie konca našej hry použijeme premennú, čiže hodnotu uloženú pod nejakým názvom/menom.

Postup:

Učiteľ vyvoláva žiakov, ktorí s pomocou celej triedy vyrobia na učiteľskom počítači ovládanie postavičky a cyklus s podmienkou na zabránenie prechodu stenami.

Učiteľ zhrnie, čo žiaci vedia, a demonštruje to na príklade z programu:

Vieme používať vstup z klávesnice na ovládanie Hrdinu. Ďalej vieme používať nekonečný cyklus, ktorý neustále kontroluje pomocou podmienky, či sa Hrdina dotýka objektu Bludisko. Ak sa dotkne bludiska, náš Hrdina cúvne.

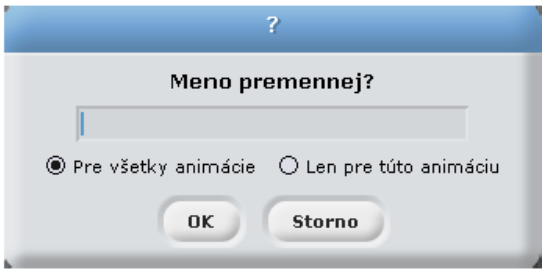
Teraz potrebujeme naprogramovať koniec hry. Čo bude koniec našej hry?

Diskusia: Čo keby sme do bludiska rozmiestnili potravu pre nášho Hrdinu? Hrdina bude musieť potravu zjesť a tým splní cieľ hry. Aby sme mohli rátať, koľko potravy Hrdina – kocúr zjedol, potrebujeme premennú. Premenná hodnota musí byť uložená pod nejakým menom. Pomenujme ju **Skóre.**

Žiaci si zapnú počítače a otvoria uložené projekty. Premennú vytvoríme/zadefinujeme v časti príkazov Premenné. Tu klikneme na tlačidlo *vytvoriť premennú.* Počítač si vyžiada

označenie premennej (obrázok 3.9). Do okienka Meno premennej treba napísať meno premennej (**Skóre**) a potvrdiť pomocou tlačidla OK. Do tejto premennej sa bude ukladať skóre, ktoré získa náš Hrdina pri zbieraní potravy.

Poznámka: Žiakov treba upozorniť, aby si všimli, že hodnota **Skóre** sa zobrazuje aj v okne výstupu. V okne výstupu sa zobrazí obdĺžnik s názvom premennej a číslom nula, čo je začiatočná hodnota premennej **Skóre.**



Obrázok 3.9. Tvorba premennej.

Teda hodnota premennej **Skóre** na začiatku hry je nula.

To musíme ale naprogramovať. V časti príkazov sa nachádza príkaz *nastav Skóre na 0.* Tento príkaz musí byť na začiatku programu, hneď po príkaze *po kliknutí na.*

Učiteľ kontroluje, či majú žiaci kliknuté na Hrdinu v pravej dolnej časti obrazovky a či príkaz *nastav Skóre na 0* vkladajú na správne miesto medzi inicializačný príkaz a cyklus.

Obrázok 3.10. Nastavenie začiatočnej hodnoty premennej **Skóre.**

Teraz potrebujeme naprogramovať potravu. Potrava (v podobe *myš* bude nový objekt. Nový objekt vložíme prostredníctvom tlačidla *Vyber animáciu zo súboru* v pravej dolnej časti. Otvoríme priečinok *Animals* (Zvieratá) a pozrieme sa, či vo fonde Zvieratá nájdeme vhodnú potravu pre mačku. Ktoré zviera by mohlo byť potravou pre mačku? Žiaci jednoznačne odpovedajú: myš. Zvolíme teda kostým *Mouse1.* Myš je však pre naše bludisko príliš veľká – treba ju zmenšiť. Tu pripomenieme žiakom, ako sme zmenšovali nášho Hrdinu. Po kliknutí pravým tlačidlom myši a zvolení *rezise this sprite* sa objaví dvojšípka a pomocou nej je možné meniť veľkosť objektu. Žiaci zmenšia myš na potrebnú veľkosť. Aby sa objekty neplietli, premenujeme našu myš zo *Sprite1* na *Myš 1.*

Učiteľ kontroluje, či žiaci klikli na objekt (a tým ho označili), ktorý majú premenovať. Prípadne zopakuje, že názov objektu sa mení v hornej časti nad záložkami *Skripty, Kostýmy* a *Zvuky.*

Teraz musíme Myš 1 naprogramovať. Upozorňujeme deti, aby skontrolovali v pravej dolnej časti, že majú objekt Myš 1 aktívny.

Časť s programom – stredná časť obrazovky je prázdna, lebo sme objektu Myš 1 nepriradili žiadne príkazy. Zopakujeme pre žiakov, že každý program sa začína zelenou vlajočkou alebo príkazom *po kliknutí na* so zelenou vlajočkou. Žiaci presunú inicializačný príkaz *po kliknutí na* do priestoru pre program.

Myš 1 naprogramujeme tak, aby zmizla, keď sa jej Hrdina dotkne, čiže aby to vyzeralo tak, že ju Hrdina zjedol. Na to musíme najprv zistiť, či sa jej dotkol. Zopakujeme si, ako sme zisťovali dotyk (kolíziu) objektov. Bolo to pomocou príkazov *stále opakuj, ak* a *dotýka sa.* Vytvoríme konštrukciu podmienky *dotýka sa* s nekonečným cyklom *stále opakuj, ak.* Prediskutujeme, ktorý objekt musíme zvoliť v príkaze *dotýka sa,* aby sme postupovali správne. Je to objekt Hrdina.

Teraz Myš 1 vie zistiť, že sme sa jej dotkli a čo má urobiť, keď sa jej náš Hrdina dotkne? Správnou odpoveďou je: zmiznúť.

Učiteľ vyzve žiakov, aby skúsili nájsť príkaz na zmiznutie objektu, aby podporil rozvoj nielen ich fantázie, ale aj intuície pri používaní takýchto programátorských prostredí. Tento príkaz sa nazýva *schovaj sa* a nachádza sa v časti Vzhľad. Žiaci spustia program zelenou vlajočkou a vyskúšajú, či Myš 1 zmizne.

A čo sa má stať s naším **Skóre,** keď „zjeme“ *Myš?* Žiaci prediskutujú, na čo vlastne má slúžiť premenná **Skóre,** že skóre (akýsi výkon hráča) sa dá vyjadriť počtom zjedených myší. Teda učiteľ dovedie žiakov k odpovedi, že musíme premennú zvýšiť o číslo jeden. Príkaz na zvýšenie premennej sa nazýva *zmeň Skóre o 1* a nachádza sa v paneli príkazov v časti *Premenné.*

Ďalšou otázkou je, kam treba umiestniť príkaz na zvýšenie premennej.

Správnou odpoveďou je, že dovnútra cyklu, k príkazu *schovaj sa.*

Žiaci spustia program a skúšajú, čo sa stane, keď Hrdina „zje“ *Myš 1.* Tá pri dotyku zmizne a premenná sa zvýši o 1. Žiaci experimentujú s programom a prídu na to, že keď raz zjedia Myš 1, tak už sa im neobjaví, aj keď program zastavia a znova spustia.

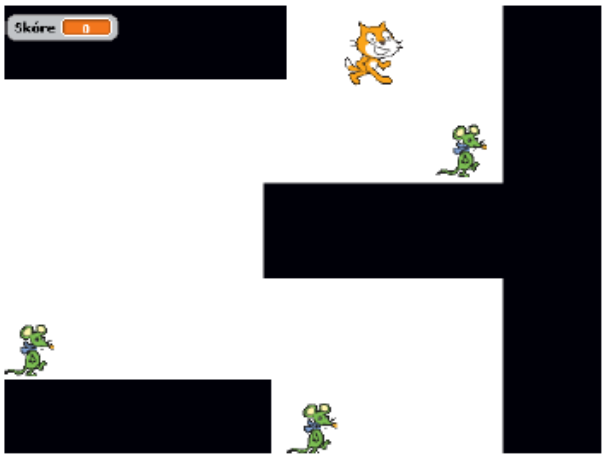
Musíme doplniť príkaz, aby sa Myš 1 pri spustení programu zobrazila. Na to použijeme príkaz *zobraz* v časti *Vzhľad.* Tento príkaz umiestnime hneď za príkazom *po kliknutí na* so zelenou vlajočkou. Výsledný program pre Myš 1 vyzerá tak, ako je to na obrázku na 3.11.

Obrázok 3.11. Program pre Myš 1.

Obrázok 3.12. Program pre Hrdinu.

V pravej dolnej časti klikneme na Hrdinu. Pridáme mu program, ktorý bude zisťovať, či sa dosiahol **Skóre** 3, ak áno, Hrdina povie *„Vyhral si.“* Program môže tvoriť celá skupina spolu. Program môže vyzerať ako na obrázku 3.12.

Cieľové skóre je nastavené na tri, ale my máme len jednu Myš 1. Ako umiestniť do bludiska tri myši? Stačí skopírovať už jestvujúcu a naprogramovanú a vyrobiť z nej kópiu (Duplicate).



Obrázok 3.13. Príklad rozmiestnenia myší v bludisku s ukazovateľom hodnoty premennej **Skóre.**

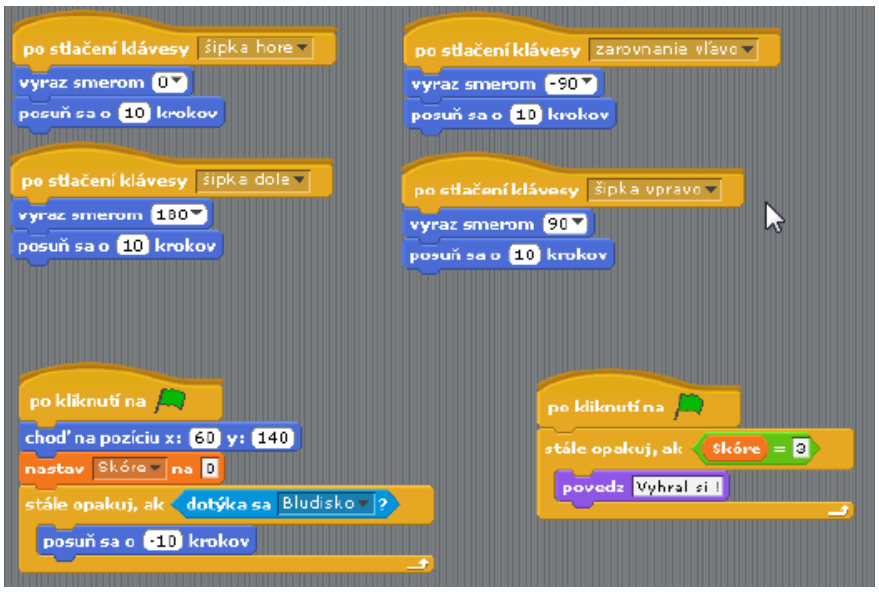
Postup:

Klikneme pravým tlačidlom myši na Myš 1 a zvolíme možnosť *Duplicate.* Novú myš premenujeme v hornej časti na Myš 2. Premiestnime ju niekam v bludisku. Druhýkrát skopírujeme Myš 1. Ďalšiu novú myš premenujeme na Myš 3 a takisto ju umiestnime do bludiska.

Príklad rozmiestnenia myší v bludisku s ukazovateľom hodnoty premennej **Skóre** vidíme na obrázku 3.13.

Ešte chýba štartovná (štartovacia) pozícia pre Hrdinu. Žiaci uchopia Hrdinu, umiestnia ho v bludisku na miesto, kde chcú, aby sa hra začínala, a zapamätajú si súradnice x a y v pravej dolnej časti okna výstupu. Tieto súradnice vložia do príkazu *choď na pozíciu,* ktorý umiestnia za inicializačný príkaz *po kliknutí na* so zelenou vlajočkou.

Výsledný program objektu *Hrdina* potom vyzerá ako na obrázku 3.14.



Obrázok 3.14. Príklad kompletnej hry Mačka a myši v bludisku.

Žiaci spustia program a pohybujú Hrdinom v Bludisku tak, aby sa dotkli všetkých troch myší. Myši postupne miznú a žiaci sledujú, ako sa mení premenná **Skóre.** Pri dosiahnutí hodnoty tri Hrdina povie: Vyhral si.

Učiteľ opakuje žiakom, že použili premennú na uchovanie počtu zjedených myší. Keď táto hodnota dosiahne 3, je splnený cieľ hry a hra sa ukončí.

Žiaci opakovane hru spúšťajú, učiteľ otázkami kontroluje, či rozumejú použitým programátorským princípom.

3.4 Upevňovanie vedomostí a zopakovanie učiva

Žiaci uložia svoje projekty a učiteľ ich upozorní na možnosť zverejniť ich na stránke https://scratch.mit.edu/. Ukáže im, ako sa to dá urobiť prostredníctvom položky menu *ZVEREJNIŤ* a voľby *Zdieľať tento projekt online…* Vyučujúci vyzve žiakov, aby vyjadrili svoj názor na vyučovanie a postoj k programovaniu. Žiaci voľne diskutujú o tom, čo sa im páčilo a čo sa im nepáčilo, či sú spokojní so svojím výkonom, aké nové pojmy (príkazy) sa naučili priebeh hodiny, o nových námetoch na riešenie, prípadne na vylepšenie svojich aplikácií a pod. Na záver učiteľ zhodnotí aktivitu žiakov a žiakov povzbudí do ďalšej práci nielen na hodinách, ale aj doma.

3.5 Otázky a úlohy

Otázky na zopakovanie

1. Aký je rozdiel medzi vstupom a výstupom v programe?
2. Čo zabezpečuje príkaz **po stlačení klávesy** (klávesu) v prostredí Scratch?
3. Prečo v Scratchi neexistuje príkaz *choď hore* a aký príkaz sa namiesto toho používa?
4. Ako zabezpečíme, aby postavička neprechádzala stenami bludiska?
5. Na čo slúži nekonečný cyklus v programe?
6. Ako funguje podmienka **dotýka sa [objektu]**?
7. Čo je to premenná a na čo sa používa v Scratchi?
8. Ako zistíme, že sme „zjedli“ objekt, napríklad myš?
9. Ktorý príkaz v Scratchi slúži na zvýšenie hodnoty premennej?
10. Čo zabezpečí príkaz **schovaj sa** a kedy ho používame?
11. Ako môžeme zabezpečiť, aby objekt po spustení hry znova „ožil“?
12. Ktorým príkazom umiestnime postavičku na konkrétne miesto na scéne?
13. Ako skopírujeme objekt v Scratchi?
14. Čo je cieľom jednoduchej hry *Mačka a myši?*

Úlohy na precvičenie

1. **Ovládanie postavičky:** Vytvor program, v ktorom sa postava pohybuje štyrmi smermi pomocou klávesových šípok.
2. **Kreslenie štvorca:** Použi príkazy pohybu na nakreslenie štvorca pomocou postavičky.
3. **Dotyková kontrola:** Pridaj do projektu bludisko a over, či vieš naprogramovať to, aby postavička pri dotyku so stenou cúvla.
4. **Zmenšenie objektov:** Zmenši objekt (napr. myš) tak, aby sa pohodlne zmestil do bludiska.
5. **Premenná Skóre:** Vytvor premennú **Skóre** a zobraz ju v okne výstupu.
6. **Zbieranie objektov:** Vytvor objekt (napr. myš), ktorý zmizne, keď sa ho dotkne postavička a zároveň zariaď, aby sa vtedy skóre sa zvýšilo o 1.
7. **Viac objektov:** Skopíruj objekt (napr. myš), premenuj ho a umiestni na iné miesto v bludisku. Over, či všetky objekty fungujú rovnako.
8. **Vyhodnotenie výsledku:** Vytvor program, ktorý zobrazí hlásenie *„Vyhral si,“* keď postavička pozbiera tri objekty.
9. **Spustenie hry zo štartovnej pozície:** Naprogramuj, aby sa postavička po kliknutí na vlajočku vždy vrátila na východiskové súradnice.
10. **Zobrazovanie objektov:** Uprav program tak, aby sa po opätovnom spustení hry všetky objekty, ktoré sa počas hrania skryli (napr. myši) opäť zobrazili.

11. **Vylepšenie bludiska:** Navrhni a nakresli vlastné bludisko s viacerými vetvami a slepými uličkami.
12. **Hra na čas:** Vytvor novú premennú **Čas** a naprogramuj odpočítavanie od 30 do 0. Nech sa po dosiahnutí nuly hra zastaví.
13. **Zmena kostýmu:** Pri každom zjedení objektu zmeň postavičke kostým – ako znak rastúcej sily alebo transformácie.
14. **Doplnkový objekt:** Pridaj do bludiska nepriateľa, ktorý sa pohybuje náhodne. Ak sa ho postavička dotkne, hra sa skončí.
15. **Zverejnenie projektu:** Ulož svoj projekt a zverejni ho on-line. Pošli ho spolužiakom a požiadaj ich, aby tvoju hru vyskúšali.

4 Praktické príklady v programovacom prostredí Scratch

V predchádzajúcej časti bolo predstavené prostredie Scratch. Nasledujúce príklady predstavené základy rozšíria, oboznámia čitateľa s konceptmi, z ktorých niektoré sú typické hlavne pre prostredie Scratch, iné majú v rámci programovania širšiu platnosť. Sú to podmienené spracovanie, jednoduché vetvenie, konečný a nekonečný cyklus, premenná, pohyb a otáčanie sprajtu (deterministický pohyb, kĺzanie na náhodnú polohu, automatický odraz sprajtu), úprava vizuálnych vlastností sprajtu (o. i. zmena kostýmov), definícia vlastného bloku, rozšírenie funkcie **Pen** (Pero), zoznam, časovač a rekurziu.

Tip: Vynikajúcim spôsobom na zdokonaľovanie sa v programovaní je dodatočné experimentovanie s príkladom, ktorý edukant samostatne vytvoril (hoci s pomocou návodu, ale s vložením vlastného úsilia, nie prevzatím hotového projektu z úložiska). Treba vyskúšať každý projekt – pozmeniť ho, rozvinúť do väčšej šírky, ale aj hĺbky – posunúť ho ďalej. Je nevyhnutné si klásť vlastné postupne rastúce ciele!

Poznámka: Sprajt (angl. sprite) doslovne znamená škriatok. V programovaní je to dynamický objekt prostredia, ktorý môže meniť svoj tvar, farbu a pod. V príkladoch v nasledujúcej časti je použité anglické komunikačné prostredie.

Táto kapitola bola inšpirovaná materiálom autora Romana Horvátha, ktorý vznikol v rámci projektu *Informatika ako nástroj rozvoja znalostnej ekonomiky (ITMS: 312011G208)* a je dostupný na adrese: https://pdfweb.truni.sk/horvath/materialy?scratch-examples. Je jeho modifikáciou a miernym rozšírením, pričom táto učebnica už nie je výstupom uvedeného projektu.

4.1 Poletujúci netopier

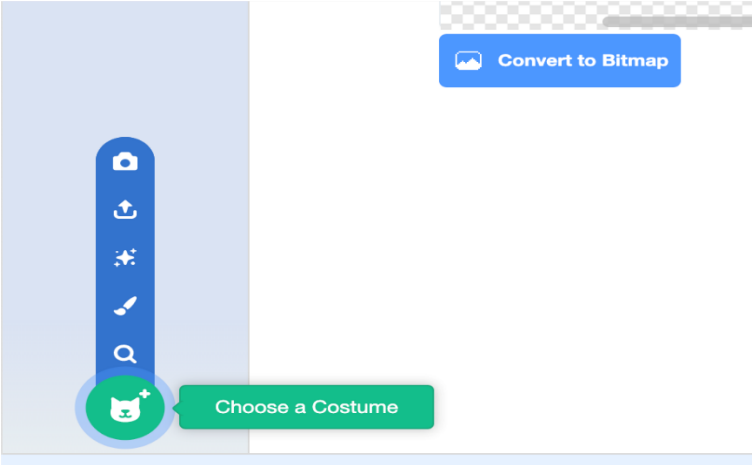
Úloha

Vytvorte program na imitáciu náhodne poletujúceho netopiera.

Tento príklad je veľmi jednoduchý. Využíva nekonečný cyklus, kĺzanie na náhodnú polohu a zmenu kostýmov.

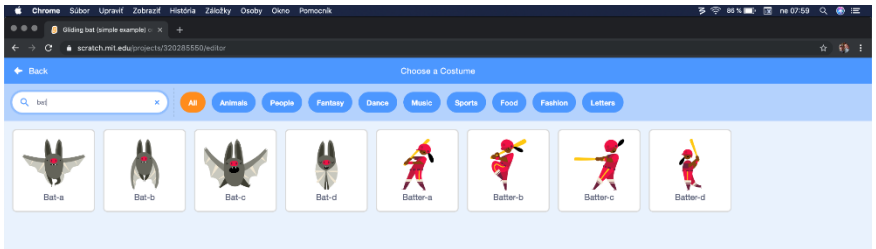
Postup riešenia

Najskôr treba zmeniť kostým sprajtu. Bude zvolený preddefinovaný kostým netopiera, ktorý má definované viaceré fázy animácie využiteľné na simuláciu lietania.



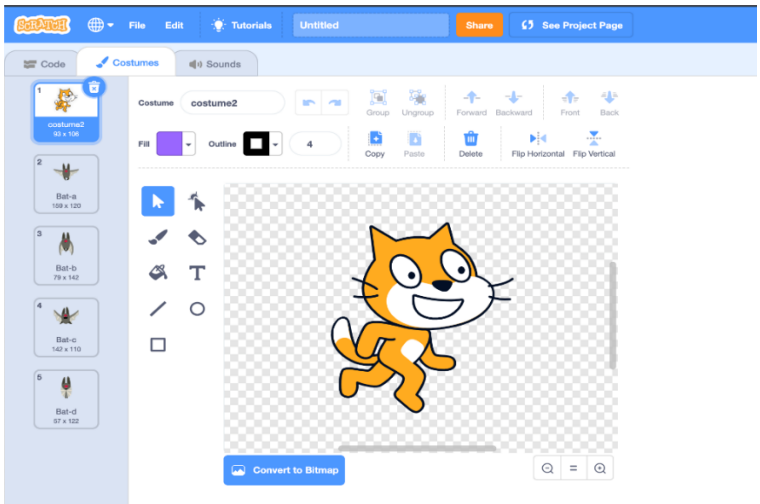
Treba zvoliť tlačidlo *Choose a Costume* vľavo dole a vybrať z ponuky lupu.

Do vyhľadávania je v tomto prípade potrebné zadať **bat** a zvoliť prvého netopiera. Postup treba opakovať pre všetky kostýmy netopiera.

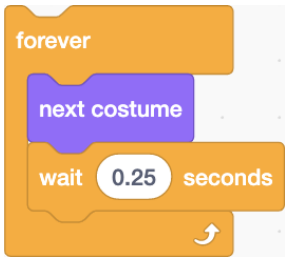


Nadbytočné kostýmy scratchového kocúra je treba vymazať.

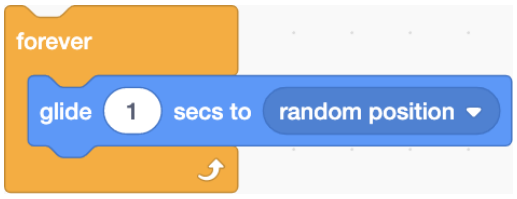
Potom treba z ponuky blokov vybrať nekonečný cyklus, do ktorého treba vložiť blok zmeny na nasledujúci kostým a čakanie 0,25 s. Po kliknutí na blok cyklu začne netopier mávať krídlami.



V ďalšej fáze bude pridaný ešte jeden nekonečný cyklus, do ktorého bude vložený blok kĺzania sprajtu na náhodnú pozíciu s trvaním jednej sekundy. Po kliknutí na cyklus sa netopier začne náhodne pohybovať po scéne.



Cyklus s blokom zmeny na nasledujúci kostým a čakaním 0,25 s.



Blok kĺzania sprajtu na náhodnú pozíciu s trvaním jednej sekundy.

4.2 Kresliace akvárium s rybičkou

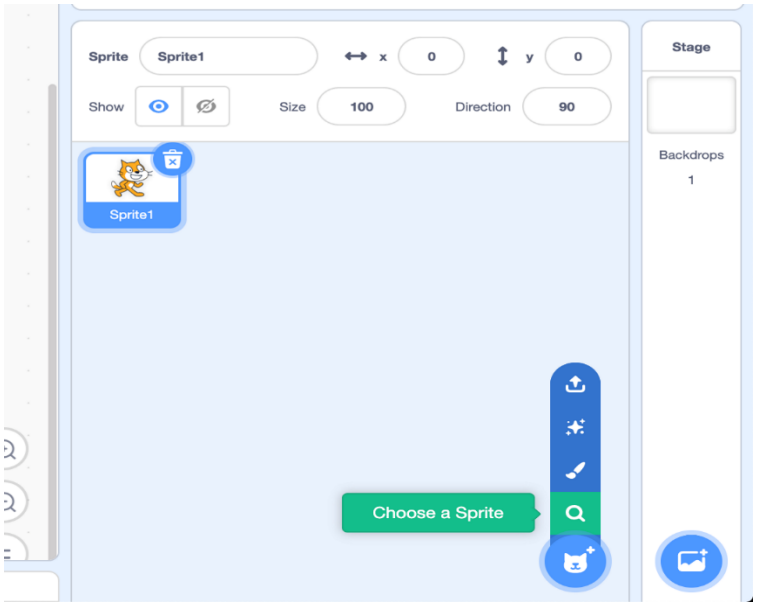
Úloha

Naprogramujte akvárium s rybičkou, v ktorom bola nedávno vymenená voda, preto je ešte mokré a preto keď ho niekto posúva po stole, zanecháva za sebou mokrú stopu.

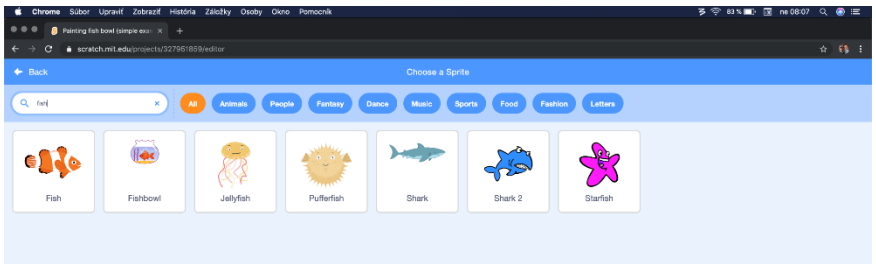
Tento príklad ukáže nakreslenie mnohouholníka s postupne rastúcou hrúbkou čiary. Počas kreslenia bude stopa kresliaceho sprajtu postupne miznúť a zmenšovať sa. Je využité rozšírenie **Pen** (pero), úprava vizuálnych vlastností sprajtu, pohyb sprajtu, otáčanie sprajtu a cyklus.

Postup riešenia

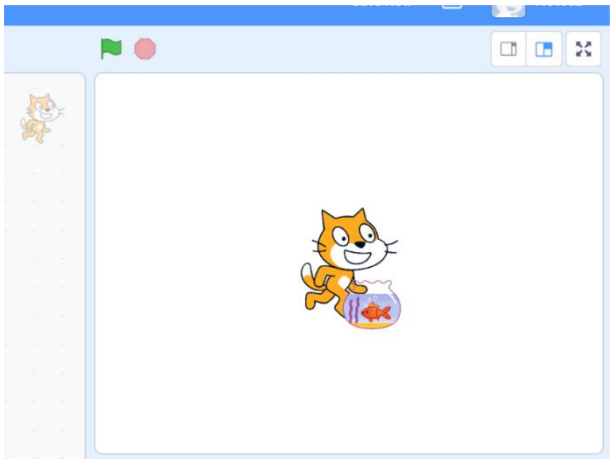
Najskôr treba zmeniť tvar sprajtu. V tejto ukážke je využité akvárium s rybkou.

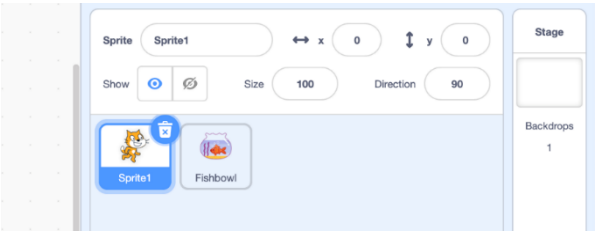


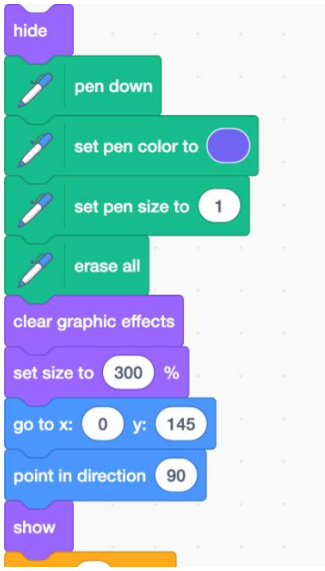
Treba zvoliť tlačidlo *Choose a Sprite* vpravo dole a vyberať z ponuky lupu.

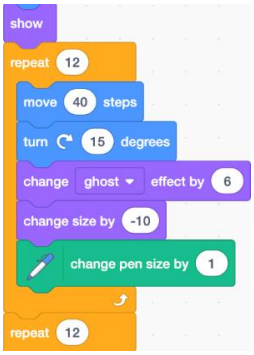


Do vyhľadávania treba v tomto prípade zadať **fish** a zvoliť sprajt **Fishbowl.**





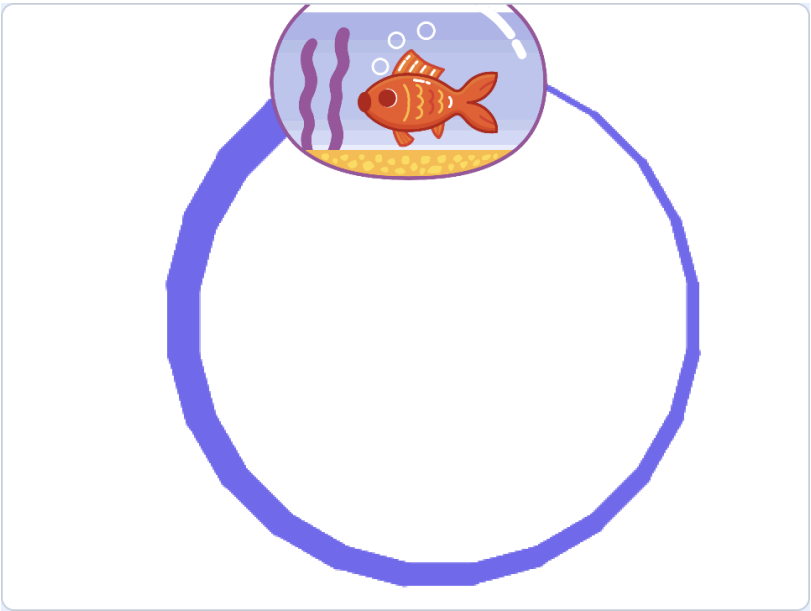




Prvé štyri zelené bloky nastavujú vlastnosti pera, za nimi nasleduje krátka sekvencia upravujúca vizuálne vlastnosti, polohu a orientáciu sprajtu a obidve tieto sekvencie sú uzavreté medzi príkazy *hide* a *show,* ktoré zabezpečia to, aby sa pred úpravami sprajt skryl a po ich vykonaní spätne zobrazil (na cieľovej pozícii s nastavenými požadovanými vlastnosťami).

Po tejto úvodnej sekvencii nasledujú dva cykly. Otočenie o 360° po 15° vyžaduje 24 opakovaní, ktoré budú rozdelené na dve časti. V prvej časti sa bude sprajt (počas otáčania a pohybu; o 40 krokov) zmenšovať a miznúť, v druhej časti spätne zobrazovať a zväčšovať (na poradí akcií veľmi nezáleží, dá sa si vyskúšať, že aj po ich vymenení bude správanie prakticky rovnaké). Počas obidvoch opakovaní sa bude meniť hrúbka pera sprajtu.

Výsledok po kliknutí na sekvenciu to bude vyzerať takto:



Ak by bolo žiaduce lepšie vidieť, čo sa počas spustenia deje, dal by sa do obidvoch cyklov zaradiť blok krátkeho čakania (napríklad 0,1 s).

4.3 Štyri kvadranty

Úloha

Rozdeľte kresliace plátno na štyri rovnaké časti (kvadranty) pomocou neviditeľnej zvislej a vodorovnej čiary (osi). Úlohou je tieto štyri kvadranty vyplniť útvarmi, ktoré budú v rámci jedného kvadrantu rovnaké.

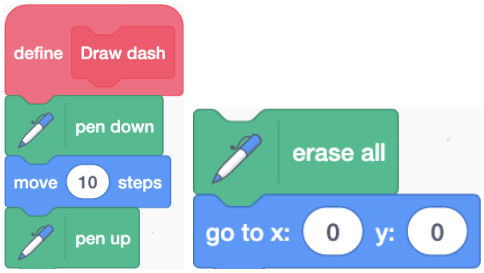
Príklad: Nech sú kvadranty vyplnené krátkymi čiarkami, ktoré by mali rovnaký sklon v rámci hornej polovice plátna a rovnaký v rámci dolnej, pričom by boli navzájom viditeľne odlišné. Napríklad nech v hornej polovici zvierajú uhol s vodorovnou osou 135° a v dolnej 45°. Podobne nech sa odlišujú objekty v ľavej a pravej časti, ale iným spôsobom. Napríklad nech majú čiarky v ľavej časti čiernu farbu a v pravej oranžovú.

Cieľ

V tomto príklade je využité rozšírenie funkcií pera, opakovanie s vopred známym počtom iterácií (opakovaní), jednoduché vetvenie a definícia vlastného bloku.

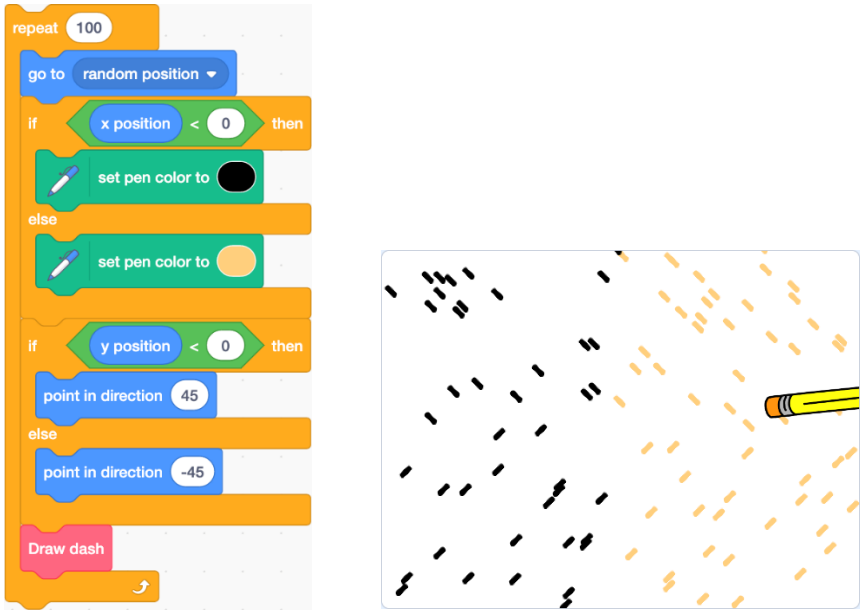
Postup riešenia

Na to, aby sa dali používať bloky pera, treba pridať rozšírenie pero (**Pen**). Po pridaní tohto rozšírenia bude k dispozícii ďalšia množina blokov. Najskôr budú využité v rámci vlastného definovaného bloku, ktorý sa bude volať *Draw dash.* Do tohto bloku budú vložené tri bloky: položenie pera, presunutie pera o 10 krokov a zdvihnutie pera. Tým bude na scénu nakreslená krátka čiarka.



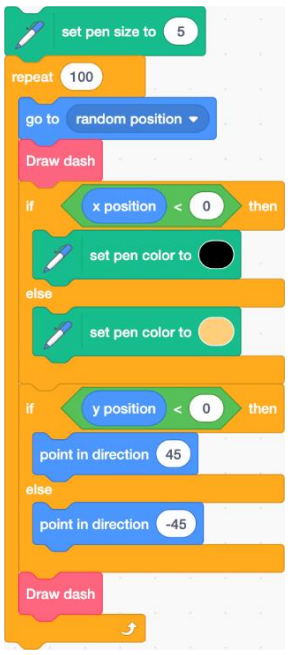
Aby bolo možné rýchlo zresetovať scénu, bude vytvorená jednoduchá sekvenciu pozostávajúca z dvojice blokov: vymazanie scény a presun pera na súradnice [0; 0]. Vždy po kliknutú na túto sekvenciu sa vymaže obsah plátna a pero sa presunie do stredu plátna.

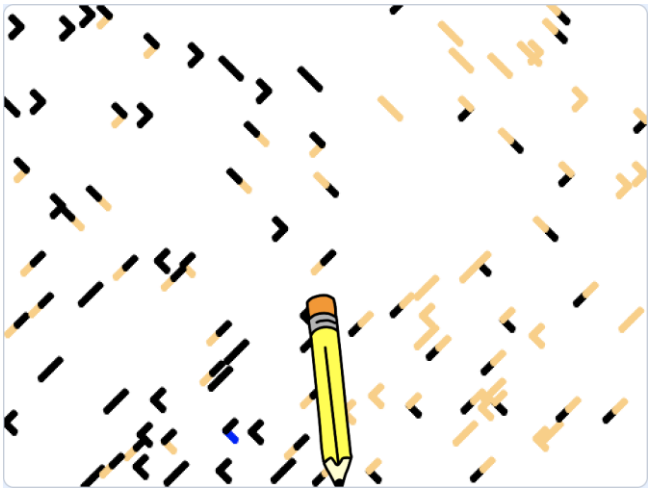
Potom bude vytvorená jedna dlhšia sekvencia blokov, ktorej cieľom bude nakresliť sto čiarok, pričom každý kvadrant bude obsahovať čiarku s mierne odlišnou vlastnosťou. Budú definované dve pravidlá: 1. čiarky v pravej polovici plátna budú mať inú farbu než čiarky v ľavej polovici a 2. čiarky v hornej polovici budú mať odlišný sklon oproti čiarkam v dolnej polovici.



Na vytvorenie pravidiel bude použitý blok jednoduchého vetvenia, do ktorého budú vložené príkazy pre jednotlivé pravidlá. Pravá a ľavá časť scény sa dá odlíšiť porovnaním x-ovej súradnice a spodná časť od hornej zasa s pomocou porovnania y-ovej súradnice.

Obidve vetvenia budú zreťazené do bloku opakovania s počtom iterácií 100. Pred vetvenia bude vložený blok na náhodnú zmenu pozície a za opakovania blok spustenia nášho bloku *Draw dash.* Na zvýraznenie je možné upraviť hrúbku čiary. Po kliknutí na túto sekvenciu bude sprajt skákať a kresliť čiarky.





Keď bude do blokov zaradený ešte jeden blok na spustenie *Draw dash* za náhodnú zmenu súradnice pred vetvenia (obrázok nižšie), tak sa výsledok zmení…

4.4 Prasličkа

Úloha

Nakreslite vetvičku – prasličku s použitím čo najmenšieho počtu príkazov.

Cieľ

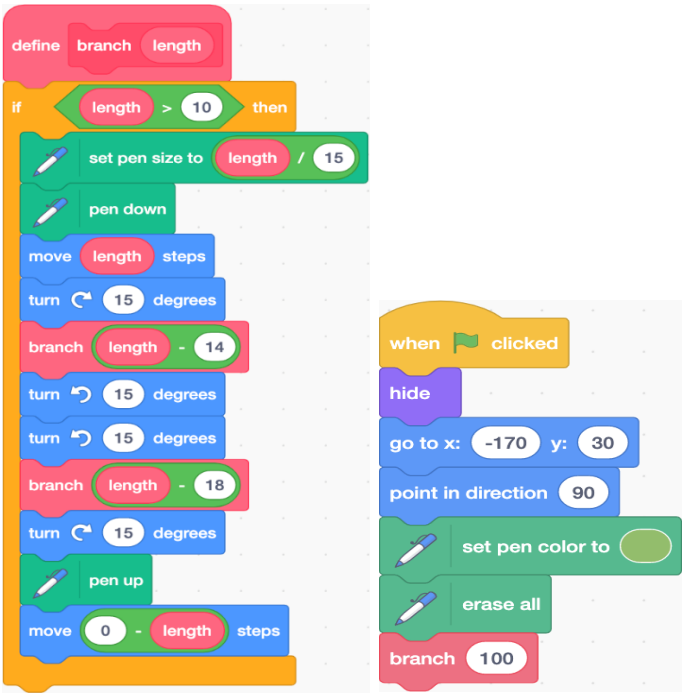
Tento príklad ukazuje, ako sa dajú pomocou Scratchu kresliť jednoduché fraktálne útvary. Umožňuje edukantovi pochopiť, ako sa dá nakresliť jednoduchá praslička s použitím minimálneho počtu príkazov. V príklade je použité rozšírenie pero (**Pen**), pohyb sprajtu a rekurzia.

Postup riešenia

Prvý bude vytvorený rekurzívny vlastný blok *branch.* S pomocou bloku podmieneného spracovania bude zabezpečené ukončenie rekurzie pri dĺžke 10 krokov. Vnútri podmieneného spracovania bude upravené kreslenie perom tak, aby bolo docielené vytvorenie požadovaného prírodného útvaru (prasličky). Nezaobíde sa to bez rekurzie.

Prvé dva bloky vnorenej sekvencie (zelené) upravujú vlastnosti pera tak, aby sa dala rastlinka postupne stenčovať. V ďalšej sekvencii sa sprajt postupne pohybuje a otáča tak, aby bolo zabezpečené, že pred a po každom rekurzívnom volaní bude mať sprajt rovnakú orientáciu. Je to dôležité. Ak by bol tento aspekt zanedbaný, kreslenie by nebolo konzistentné a výsledkom kreslenia by bol úplne odlišný útvar.

Aby počas rekurzívneho pohybu nebola čiara rastlinky kreslená dvakrát, je výhodné zdvihnúť pero vždy, keď sa sprajt pohybuje smerom dozadu a položiť vtedy, keď sa pohybuje dopredu. Celý rekurzívny blok bude vyzerať tak, ako je na ľavej strane nasledujúceho obrázka.



Nakoniec bude vytvorená reakcia na kliknutie na zástavku (*Go*), v ktorej budú nastavené vlastnosti sprajtu a zabezpečené prvé spustenie rekurzívneho bloku.

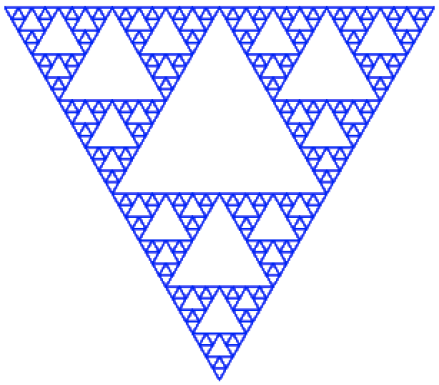
Nakreslená rastlinka bude vyzerať takto:

Tip: Na rýchlejšie nakreslenie rastlinky je výhodné zapnúť rýchly režim: Edit – Turn on Turbo Mode.

4.5 Sierpinského trojuholník

Úloha

Nakreslite grafický útvar, tzv. Sierpinského trojuholník, ktorý vyzerá takto:



Cieľ

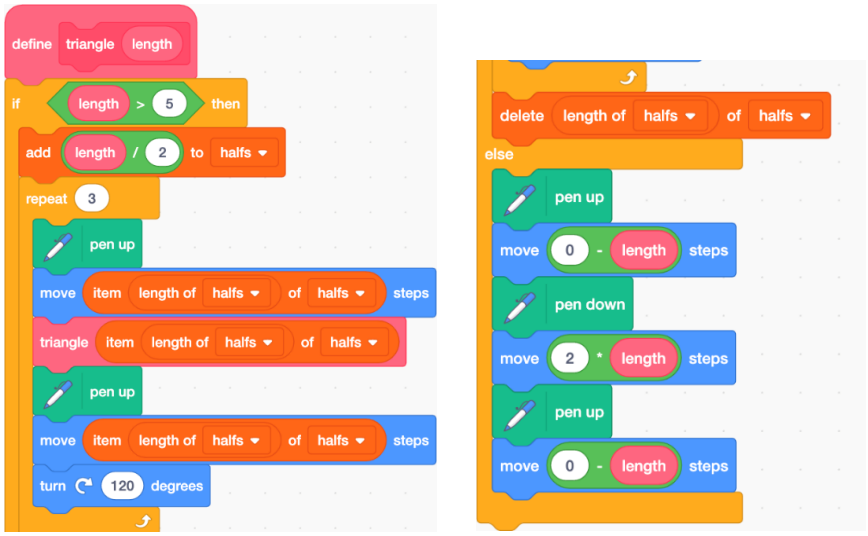
V tomto príklade je ukázané to, ako sa dá nakresliť iný fraktálny útvar. Ide o takzvaný Sierpinského trojuholník. Je to relatívne známy tvar, ktorý sa dá vytvoriť rôznymi spôsobmi. Vzniká rekurzívnym vyberaním strednej časti ľubovoľného rovnostranného trojuholníka. Toto riešenie kreslí tento tvar pomocou čiar. Bude potrebné rozšírenie pero (**Pen**), bude využité opakovanie, zoznam, pohyb sprajtu a rekurzia.

Postup riešenia

Na správne nakreslenie celého útvaru je dôležité si uvedomiť základný princíp celého správania. Rekurziu treba vykonať vždy v strede každej strany kresleného rovnostranného trojuholníka. Skutočný fraktálny útvar siaha do nekonečnej hĺbky (jeho kreslenie by sa nikdy neskončilo), človek však nemá k dispozícii nekonečne veľa času, preto je potrebné sa uspokojiť s nakreslením útvaru s určitou „dostatočnou presnosťou."

Na tieto účely postačí, keď bude rekurzia ukončená pri dĺžke strany päť krokov. Počas vykonávania rekurzívneho vlastného bloku bude využitý zoznam **halfs** (ktorý treba vopred vytvoriť), do ktorého budú priebežne ukladané (a z neho mazané) aktuálne dĺžky kreslenia v krokoch. Bez týchto akcií by nebolo možné mať prehľad o správnych dĺžkach kreslenia počas dlhej reťaze rekurzií.

Vlastný definovaný blok *triangle* na nasledujúcom obrázku od edukanta vyžaduje podrobnú a pozornú analýzu:



Pozostáva z jednoduchého vetvenia. Podmienka určuje ukončenie rekurzie pri dĺžke päť krokov. Prvým príkazom v hlavnej vetve je uloženie ďalšieho prvku do nášho zoznamu **halfs.** Za ním nasleduje cyklus s tromi opakovaniami. Ten bude zabezpečovať kreslenie trojuholníkov. Uhol otáčania sprajtu je 120°. Zdôvodniť sa to dá dvojako:

1. tri krát stodvadsať je tristošesťdesiat (čiže jedno úplné otočenie okolo svojej osi);
2. sprajtom musíme otáčať o kvázi „vonkajší“ (v skutočnosti doplnkový) uhol rovnostranného trojuholníka, nie o vnútorný (ktorý je 60°).

Vnútri cyklu je okrem otáčania vykonávaný pohyb o polovicu aktuálnej dĺžky, ktorá je čítaná z aktuálneho prvku zoznamu **halfs,** rekurzívne volanie tohto bloku a dokončenie pohybu o polovicu dĺžky. Zdvihnutie pera, ktoré je možné v cykle vidieť, je vykonané v podstate „pre istotu.“ Malo by byť zabezpečené príkazom v alternatívnej vetve a potom je možné a odporúčané si samostatne skúsiť, čo sa stane, keď bude odtiaľto vymazané.

Dokončenie celého algoritmu je v alternatívnej vetve, v ktorej sa nachádza nakreslenie čiarky na poslednej úrovni rekurzie. Kreslenie je vykonané tak, aby sa čiarka nakreslila s dvojnásobnou dĺžkou, a to presne so stredom na aktuálnej pozícii sprajtu – sprajt najprv cúvne o aktuálnu dĺžku so zdvihnutým perom, potom nakreslí čiarku s dvojnásobnou dĺžkou a nakoniec sa vráti späť do východiskového bodu.

V reakcii na kliknutie na štartovaciu zástavku *Go* sú vykonané úvodné nastavenia sprajtu a prvé volanie rekurzívneho bloku *triangle.* Výsledok kreslenia vyzerá presne tak, ako na obrázku v zadaní (hore).

Tip: Na rýchlejšie nakreslenie fraktálu Sierpinského trojuholníka je výhodné zapnúť rýchly režim: Edit – Turn on Turbo Mode.

Poznámka: Žiakom treba na názorných príkladoch vysvetliť význam základných kľúčových pojmov, ako je iterácia, rekurzia, fraktál, klon, klonovanie a podobne, ktoré sa vyskytujú v použitých príkladoch. Nie je odporúčané používať tieto pojmy na prvom stupni alebo v prvom cykle základnej školy. Na druhom stupni alebo v ďalších dvoch cykloch základnej školy ich treba používať len ojedinele a citlivo – nikdy nie bez názorného vysvetlenia ich významu. Je odporúčané radšej používať slovenský ekvivalent (ak jestvuje), prípadne používať slovenský, hoci viacslovný opis.

4.6 Skákajúca loptička

Úloha

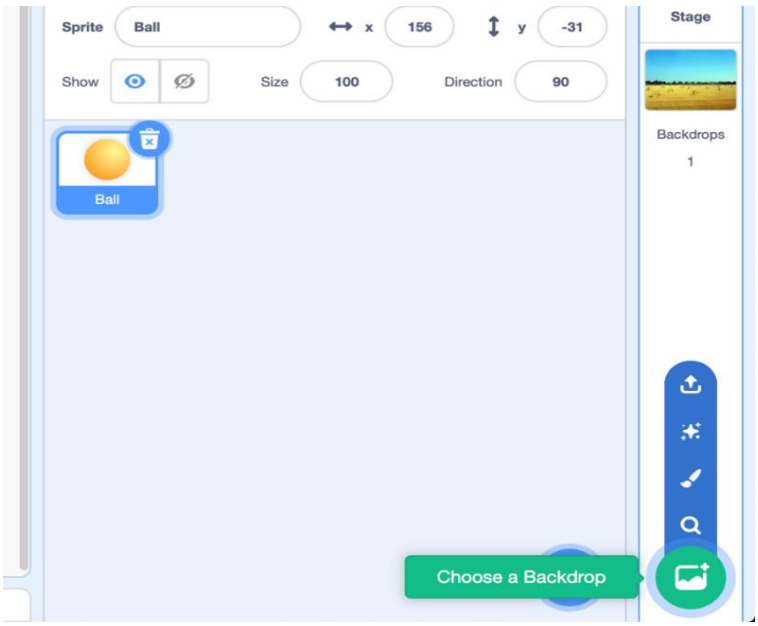
Naprogramuj skákajúcu loptičku.

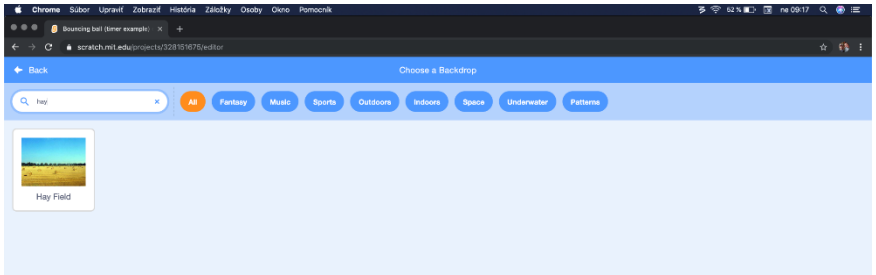
Cieľ

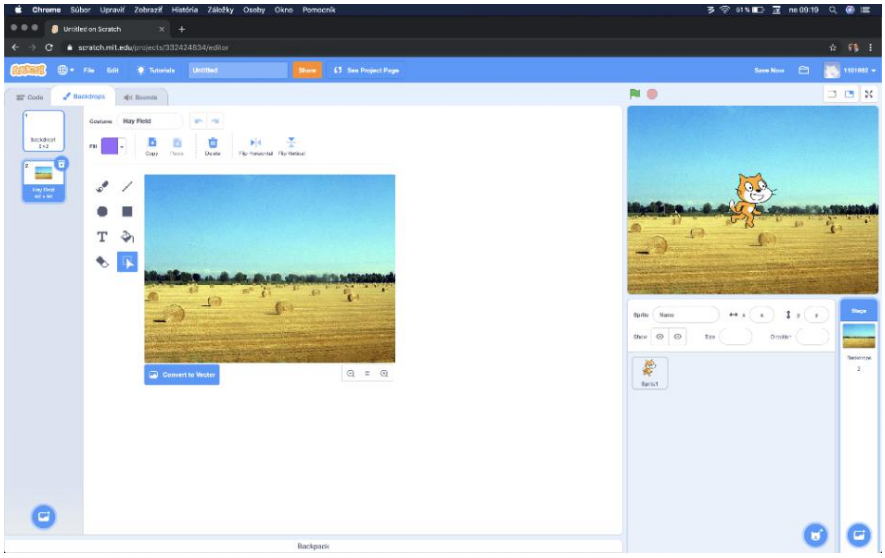
V tomto príklade bude ukázaná implementácia jednoduchej skákajúcej loptičky podľa zákonov fyziky. Na zobrazenie pohybu loptičky bude využitá zmena pozadia, zmena sprajtu, pohyb sprajtu, premenná, podmienené spracovanie, automatický odraz sprajtu a časovač.

Postup riešenia

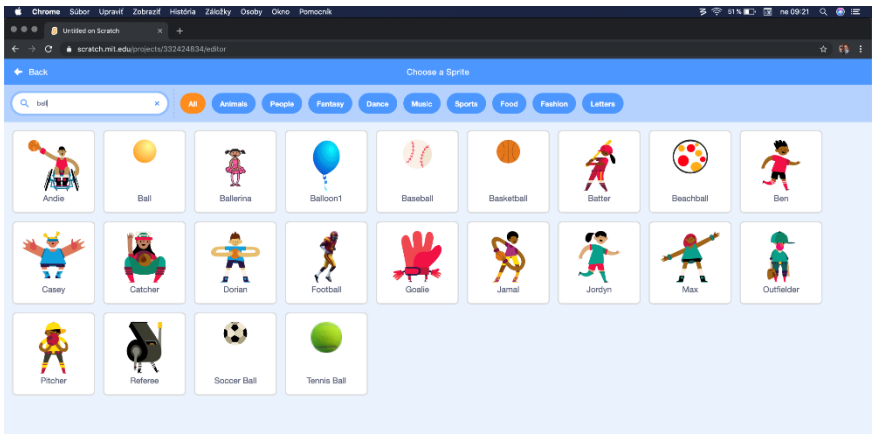
Na úpravu pozadia treba kliknúť na *Choose a Backdrop* vpravo dole a zvoliť lupu. Do vyhľadávania treba v tomto prípade zadať **hay** a zvoliť ponúknuté pozadie (**Hay Field**).



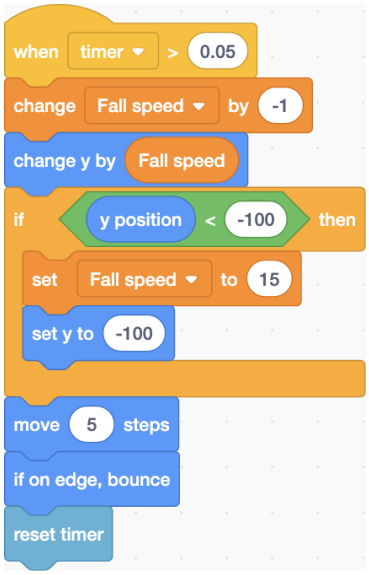




Nadbytočné prázdne pozadie (vľavo hore) môže byť vymazané. Potom treba zmeniť tvar sprajtu. Postup je podobný ako v predchádzajúcich príkladoch. Zvolený sprajt v tomto príklade sa volá **Ball.** Na správne fungovanie tohto príkladu treba nastaviť hodnotu smeru sprajtu na 90° – treba zmeniť hodnotu **Direction.**



Obsah nasledujúcej reakcie na časovač si zaslúži bližšiu pozornosť:



Spúšťa sa každých 50 milisekúnd a vykoná tieto akcie:

- Upraví hodnotu definovanej premennej **Fall speed** o jednotku smerom nadol (odčíta jednotku).
- Upraví aktuálnu vertikálnu polohu loptičky (y-ovú súradnicu) o aktuálnu hodnotu premennej **Fall speed.**
- Overí, či vertikálna poloha klesla pod zápornú stovku. Ak áno, nastaví hodnotu premennej **Fall speed** na kladných 15 krokov a „zresetuje“ hodnotu y-ovej súradnice presne na hodnotu zápornej stovky (tým je simulovaný náraz na pevnú zem; zároveň sa tým zabraňuje vzniku prípadných nezrovnalostí v rámci opakovaného vykonávania tejto reakcie).
- Potom posunie loptičku v aktuálnom smere o päť krokov. Táto akcia bude posúvať loptičku sprava doľava a naopak.

Nakoniec treba vykonať dve akcie. Prvou bude test toho, či loptička narazila na okraj pracovného plátna. Ak áno, treba zabezpečiť automatické odrazenie sa opačným smerom a reset časovača (inak by nefungoval správne).

Príklad sa dá spustiť buď kliknutím na zástavku *Go,* alebo kliknutím na sekvenciu reakcie na časovač. V obidvoch prípadoch je efekt rovnaký. Loptička bude donekonečna skákať po ploche (a toto správanie sa v aktuálnej verzii Scratchu nedá vypnúť ani po kliknutí na červené tlačidlo *Stop*).

4.7 Ovládacie tlačidlá

Úloha

Vytvorte vlastné ovládacie tlačidlá, ktoré budete môcť využívať na ovládanie postavičiek alebo iných objektov v rôznych aplikáciách, napríklad v počítačových hrách vytvorených v prostredí Scratch.

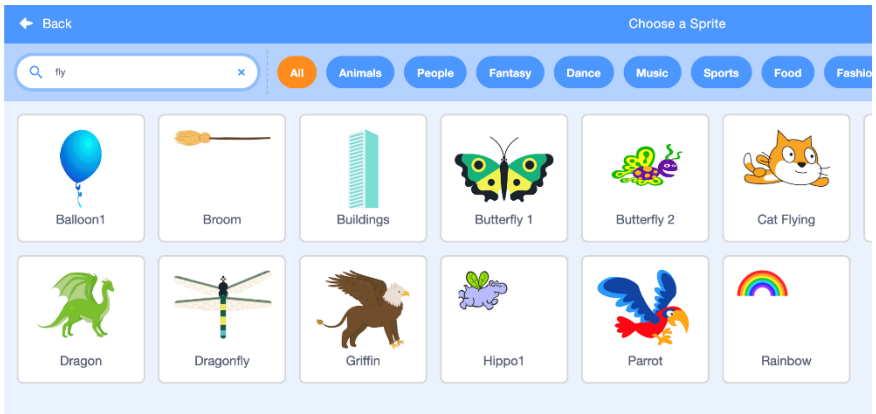
Cieľ

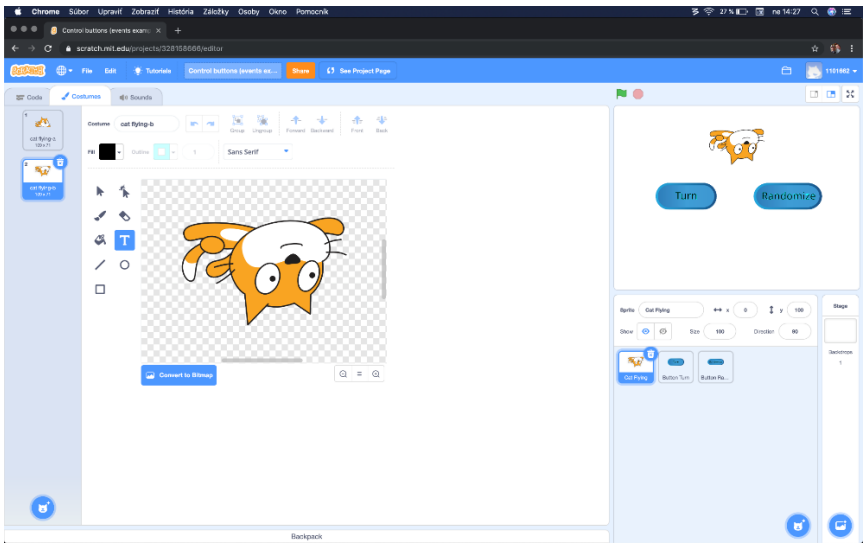
Tento príklad demonštruje jednu z možností, ako si pripraviť a použiť vlastné ovládacie prvky na hru alebo miniaplikáciu. Budú vytvorené dve tlačidlá, ktorými bude ovládaná postavička scratchovej mačky pripravenej na let (prípadne lietajúcej).

Postup riešenia

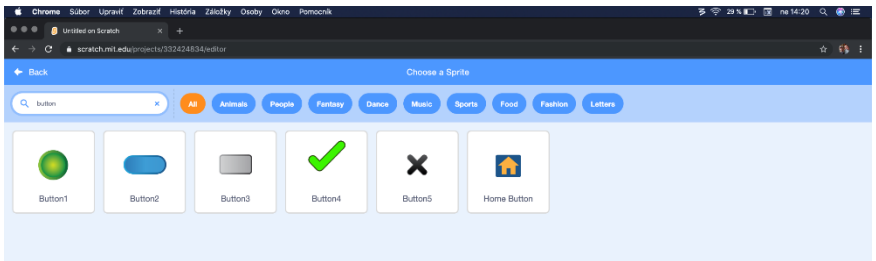
Prvým krokom tohto návodu je vloženie letiacej mačky.

Podobne ako v predchádzajúcich návodoch teba vložiť nový tvar. Na účely tohto riešenia bol zvolený tvar s názvom **Cat Flying** (tvar na kartičke úplne vpravo hore na obrázku nižšie). Kostým mačky bol naklonovaný, jeho klon bol premenovaný na **cat flying-b** a prevrátený vertikálne (čiže podľa horizontálnej osi).

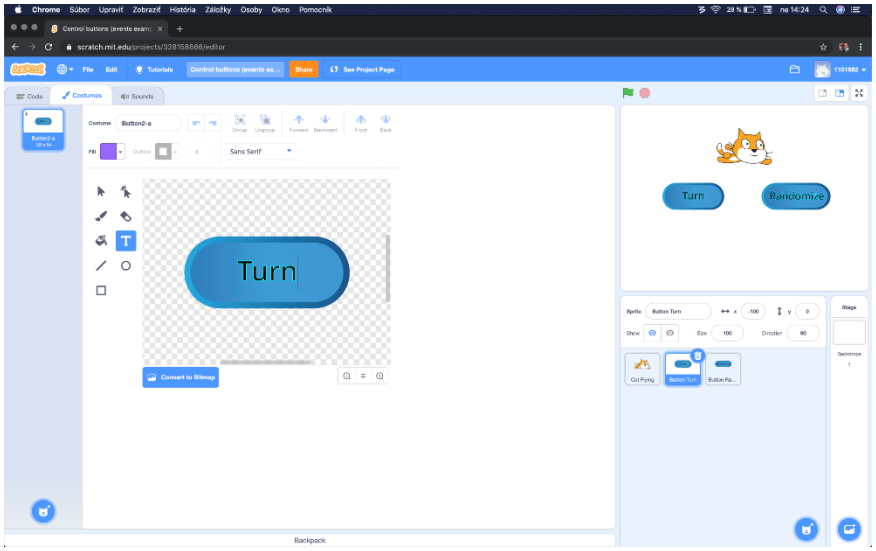


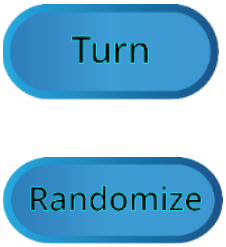


Klon kostýmu mačky je prevrátený dole hlavou úmyselne, pretože pri otočení sprajtu na mínus deväťdesiat stupňov bude jeho tvar smerovať správne. (Dá sa to v Scratchi zabezpečiť aj inak, ale na tomto mieste je použité toto riešenie.) Ďalším krokom návodu je pridanie dvoch tlačidiel. Pre ne bol z ponuky zvolený tvar **Button2.**



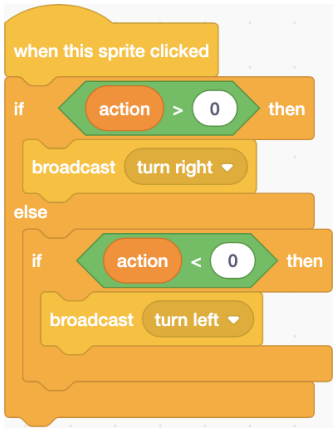
V editore kostýmov boli obidve upravené tak, aby obsahovali požadovaný text. Prvému tlačidlu, ktoré bolo náležite pomenované, bol pridaný text **Turn,** podobne bolo upravené druhé s textom **Randomize.**

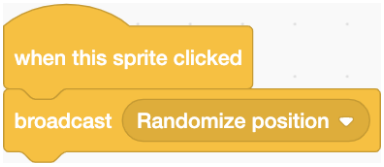




V ďalšom kroku budú do jednotlivých sprajtov vložené sekvencie blokov príkazov, ktoré zariadia, aby po kliknutí na jednotlivé tlačidlá mačka vykonávala želané akcie. Bude na to

využité posielanie správ. Na tento účel budú potrebné tri správy pomenované: **turn left, turn right** a **Randomize position.** Správy budú odosielané v sprajtoch tlačidiel:



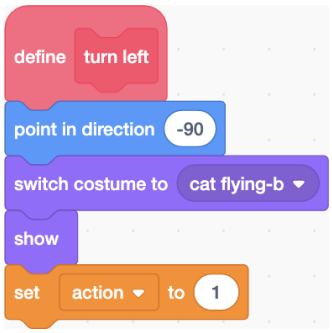


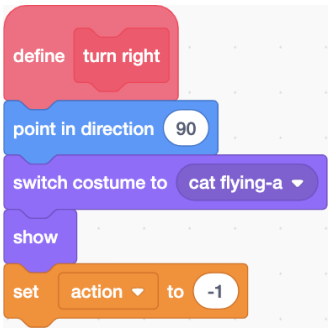
Otáčanie (tlačidlo **Turn**) využije premennú **action** (ako vždy – premennú treba vytvoriť vopred). Hodnota premennej bude alternovať medzi kladnou a zápornou jednotkou, podľa toho bude alternovať poslanie správy **turn left** alebo **turn right.**

Nula je rezervovaná na zablokovanie tlačidla, takže na to, aby tlačidlo fungovalo, treba hodnotu premennej predvoliť na kladnú alebo zápornú jednotku. Najjednoduchšie sa to dá zabezpečiť definovaním hodnoty v nejakej akcii a potom vykonaním tejto akcie (v tomto príklade je nastavovanie hodnôt vložené do vlastných blokov, ktoré budú vytvorené o malú chvíľu).

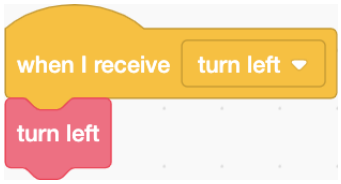
Presun mačky na náhodnú polohu bude spustený ako reakcia na stlačenie tlačidla **Randomize,** a to odoslaním správy **Randomize position.**

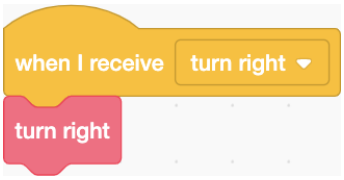
V sprajte mačky budú správy prijímané a spracúvané. Pred vytvorením samotných reakcií na spracovanie správ treba vytvoriť dva vlastné bloky, ktoré budú potom použité v uvedených reakciách. To poskytne možnosť využitia týchto blokov aj iným spôsobom než iba pri spracovaní správ, čiže tento postup je odporúčaný použiť aj pri iných projektoch – prináša benefit „recyklovateľnosti“ (odborne povedané znovupoužiteľnosti kódu). Vlastné bloky budú pomenované rovnako ako správy (v Scratchi to nie je technický konflikt a ich vizuálne rozlíšenie zjednodušuje farebné označenie blokov Scratchu – vlastné bloky majú ružovočervenú farbu a správy žltú) **turn left** a **turn right:**



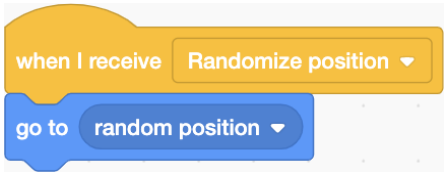


Rovnomenné reakcie na spracovanie správ (**turn left** a **turn right**) sa tým stávajú veľmi priamočiarymi:





Rovnako jednoduchým bude aj spracovanie správy **Randomize position:**



Ak by bolo želané mačku počas presunu animovať, treba nahradiť blok *go to random position* blokom *glide (1) secs to random position,* pričom hodnota 1 môže byť upravená napríklad na **0.1** (poznámka: v Scratchi sa používa „desatinná bodka“).

4.8 Pochytaj hviezdičky!

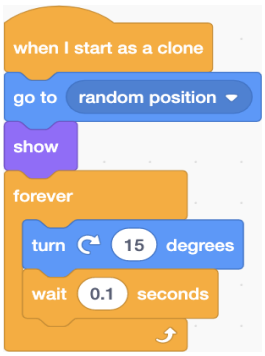
Cieľ

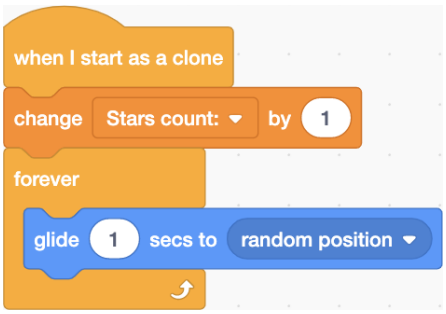
Tento príklad je komplexnejší. Ukazuje vytvorenie jednoduchej hry. Hráč musí „pochytať“ všetky hviezdičky rýchlo poletujúce po ploche. Stav je signalizovaný zobrazovanou hodnotou premennej a úspech je „odmenený“ zobrazením kývajúceho sa zvončeka.

Postup riešenia

Budú použité dva sprajty (resp. dva druhy sprajtov) a bude využité klonovanie objektov a posielanie správ.

Prvý sprajt bude hviezdička upravená štandardným spôsobom (pozri predchádzajúce príklady). Do neho budú vložené štyri odozvy (reakcie) na udalosti opísané v ďalšom texte.

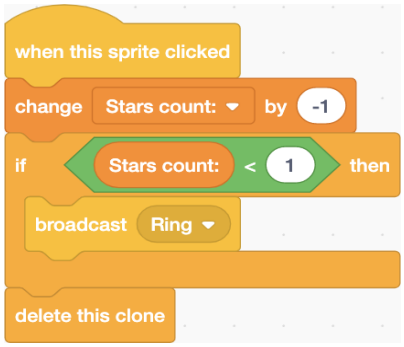




1. reakcia 2. reakcia

Prvé dve sú ukončené nekonečným cyklom a zabezpečujú zobrazenie a pohyb klonov hviezdičky. Každý nový klon hviezdičky bude zobrazený na náhodnej pozícii, bude sa do-

nekonečna otáčať o pätnásť stupňov doprava (za jednotku času určenou časovačom Scratchu) a bude sa náhodným spôsobom kĺzať po ploche (porovnaj aj s príklad o kĺzajúcom sa netopierovi).



3. reakcia 4. reakcia

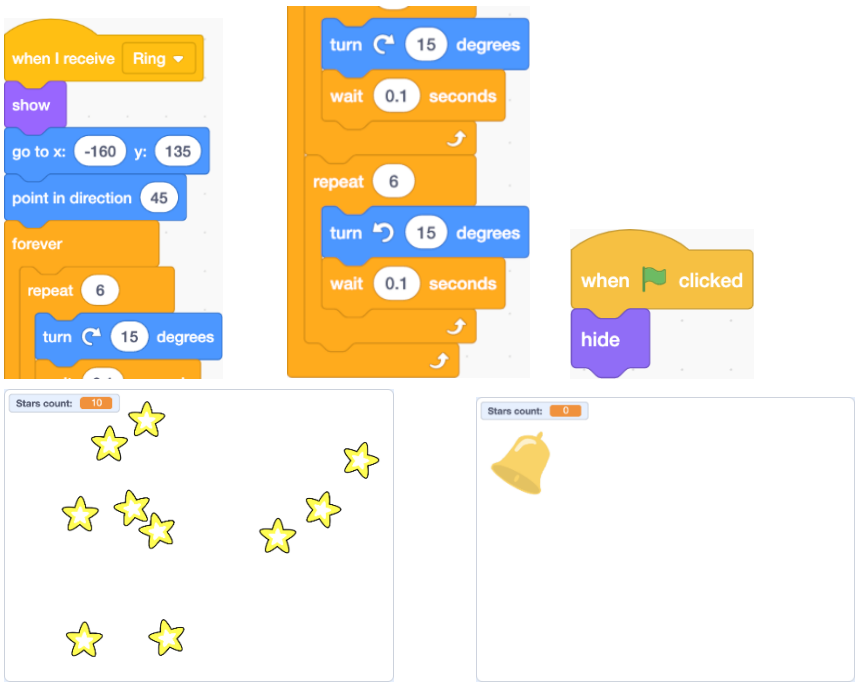
Tretia reakcia je odozvou na klikanie na hviezdičky. Každé úspešné kliknutie na klon hviezdičky ho vymaže. Keď sa počet klonov zníži na nulu, tak je odvysielaná správa **Ring** (ktorá bude spracovaná sprajtom zvončeka, čo bude zabezpečené neskôr).

Posledná reakcia je odozvou na kliknutie na zelenú zástavku (*Go*). V reakcii je vynulovaná hodnota premennej sledujúcej počet klonov hviezdičiek a je vytvorených desať klonov.

Druhý sprajt je zvonček. Jeho tvar treba v editore kostýmov mierne pozmeniť. Cieľom je presunúť stred otáčania sprajtu do hornej časti zvončeka, do miesta, kde bývajú zvončeky obvykle prichytené. Inak by sa zvonček otáčal „zvláštnym spôsobom.“ (Pre skúsenejších: dalo by sa to vyriešiť afinnými transformáciami, ale toto riešenie je zbytočne komplikované.)

Do príkazovej časti budú vložené dve reakcie:

Prvá z nich len skryje zvonček po kliknutí na zástavku *Go* a druhá bude čakať na odvysielanie správy **Ring,** ktorá bola spomenutá vyššie. Po prijatí tejto správy sa zvonček zobrazí na určenom mieste a s použitím dvoch vnorených cyklov umiestnených v nekonečnom cykle sa bude kývať doprava a doľava.



Po skončení hry ju stačí opätovne spustiť zástavkou *Go*. Veľa zábavy!

4.9 Otázky a úlohy

Otázky na zopakovanie

1. Aký je rozdiel medzi *cyklom s podmienkou* a *cyklom s vopred známym počtom opakovaní?*
2. Na čo slúžia *kostýmy* v prostredí Scratch a ako je ich možné meniť?
3. Vysvetli pojem *rekurzia* a uveď jeden príklad jej využitia v tejto kapitole.
4. Čo je to *klon* v Scratchi a ako sa používa?
5. Ktorý príklad v tejto kapitole ukazuje využitie *zoznamu* a na čo je tam použitý?
6. Čo znamená to, že objekt sa *odráža od okraja scény* a kde je v tejto kapitole takýto jav využitý?

Úlohy na precvičenie

1. **Využitie premennej:** Vytvor program, ktorý využíva *premennú* na sledovanie počtu skokov loptičky. Zobrazuj túto hodnotu na scéne.
2. **Veľkosť objektu:** Uprav program s netopierom tak, aby menil svoju veľkosť pri každom pohybe.
3. **Pridanie časového limitu:** Rozšír program *Pochytaj hviezdičky!* o časový limit – ak hráč nezachytí všetky hviezdičky do 15 sekúnd, tak sa hra skončí.
4. **Zmena fraktálu:** Navrhni novú verziu programu Sierpinského trojuholníka, ktorá používa iný uhol alebo pravidlo rekurzie.

5 Kapusta, koza, vlk a dievča v prostredí Baltík

Zadanie tejto kapitoly je jednoduché a priamočiare: Implementujte známu didaktickú hru Kapusta, koza, vlk a dievča alebo Ako dostať bezpečne na druhú stranu rieky kapustu, kozu a vlka? Na osvieženie príbehu – zápletka je nasledujúca:

Dievča sa chystá preplaviť rieku, aby dopravilo na druhý breh tri dôležité veci: svojho domáceho miláčika (kozu), vzácnu kapustu zo školského záhona a túlavého vlka, ktorého našla zraneného v lese a chce ho odviesť k ochranárom. Má k dispozícii malú loďku, do ktorej sa zmestí vždy len ona a jeden z pasažierov. Musí však riešiť dilemu: keď nechá bez dozoru kapustu s kozou, tak ju koza zje a podobne keď nechá bez dozoru vlka s kozou, vlk kozu zraní alebo zabije. Dokáže všetko prepraviť bezpečne bez toho, aby niečo alebo niekoho stratila?

5.1 Analýza úlohy

V ďalšom texte bude uvedený opis postupu riešenia tejto úlohy bývalej študentky trnavskej pedagogickej fakulty, Soni Horváthovej. Implementácia bola súčasťou jej semestrálnej práce z predmetu tvorba didaktických aplikácií.

Študentka si vybrala na riešenie známu logickú úlohu o prievozníkovi, vlkovi, koze a kapuste, čo je vlastne zadaním tejto úlohy. Na implementáciu didaktickej hry použila ikonické programovacie prostredie Baltík. Študentka v používaní tohto prostredia bola úplnou začiatočníčkou, nikdy toto prostredie na tvorbu aplikácií nepoužívala.

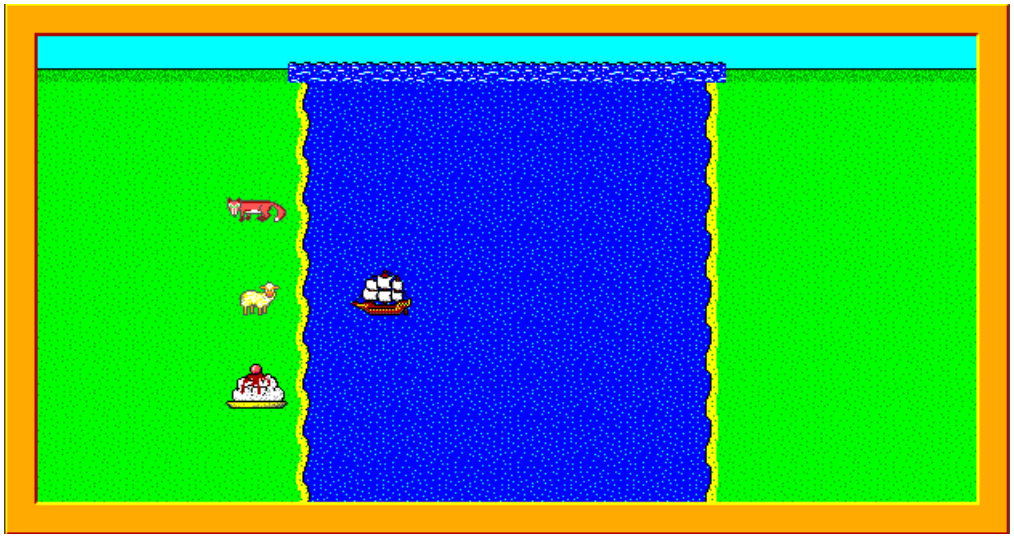
Úlohu sa snažila vyriešiť „cestou najmenšieho odporu“ s použitím nástrojov a stavebných elementov, ktoré priamo ponúka toto prostredie.

Keďže Baltík nemá medzi svojimi grafickými objektmi kozu ani kapustu, koza bola nahradená ovcou a kapusta koláčikom na tanieri. Prievozníka nahradila plachetnica.

Pravidlá hry sú takéto:

1. Scéna pozostáva zo širokej rieky, po ktorej pláva loď.
2. Na ľavom brehu sa nachádzajú traja pasažieri: vlk, ovca a koláčik.
3. Hlavným cieľom hráča je dostať (previezť) všetkých troch pasažierov na pravý breh (vtedy sa hra skončí úspechom), musí však dbať na dve (nasledujúce) jednoduché pravidlá.
4. Ak zostane osamote vlk s ovcou, zje ju.
5. Ak zostane osamote ovca s koláčikom, zje ho.

Obrazovka vyzerá tesne po spustení hry takto:



5.2 Riešenie

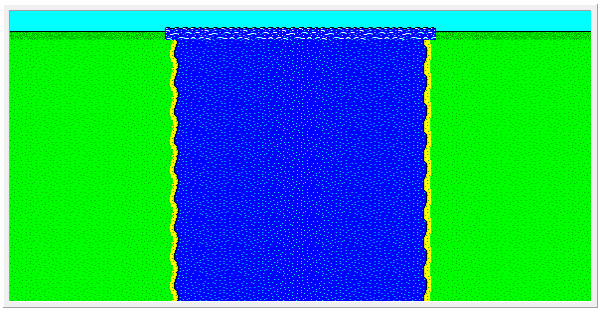
Pred začatím programovania hry bolo potrebné pripraviť scény, ktoré budú používané v programe:

S00 – je statická časť scény (prostredie).

S01 – je hlásenie situácie, keď vlk zožral ovcu.

S02 – je hlásenie situácie, keď ovca zožrala koláčik,

S03 – je úspešné ukončenie hry, splnenie úlohy.



vlk-koza-kapusta.S00

vlk-koza-kapusta.S01

vlk-koza-kapusta.S02

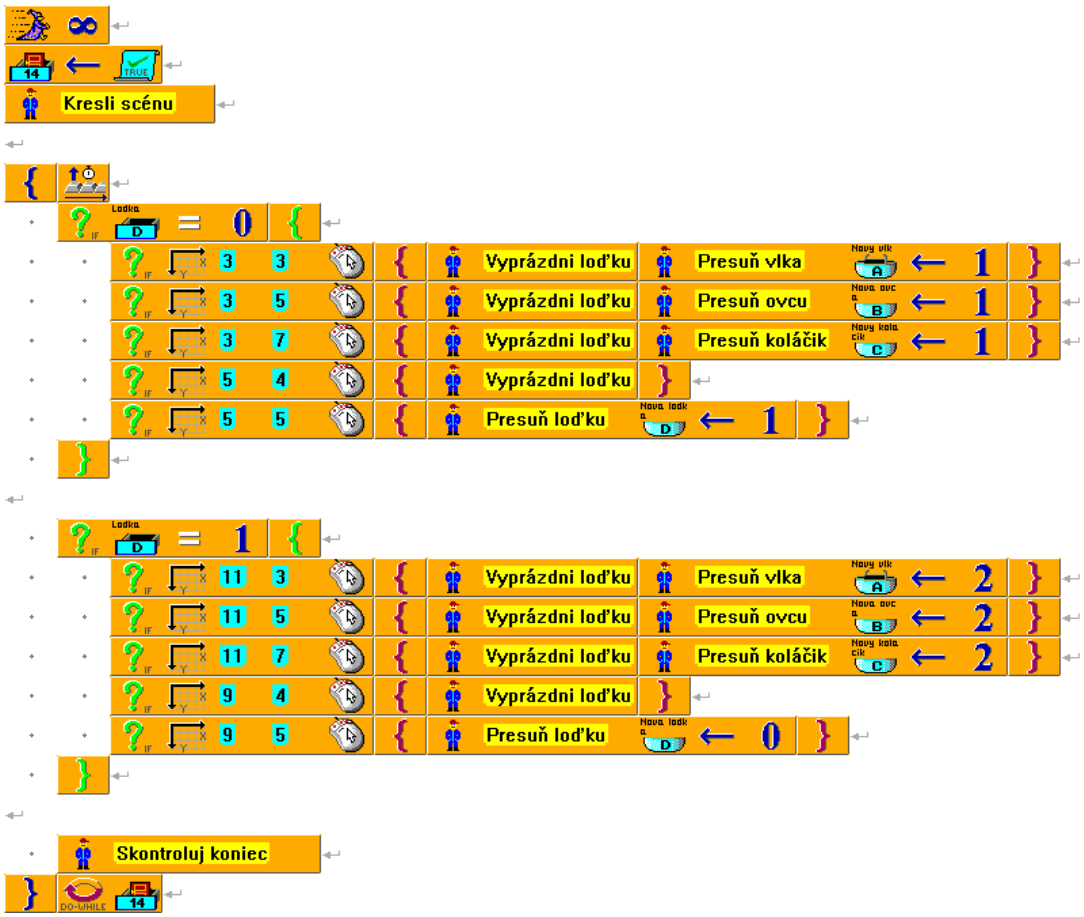
vlk-koza-kapusta.S03

5.3 Hlavný blok programu

V hlavnom bloku je nastavená rýchlosť Baltíka na nekonečno a sú vykonané ďalšie počiatočné nastavenia. Tu je vytvorený hlavný cyklus programu.

V hlavnom cykle sa iba kontroluje, kam hráč klikol, a v prípade, že ide o niektorú kľúčovú pozíciu, Baltík požiada o pomoc ďalších prislúchajúcich pomocníkov. (Tí sú definovaní nižšie.)

Hra sa skončí vtedy, keď je hodnota globálnej premennej **14** rovná **false.**



5.4 Definícia pomocníkov

Kresli scénu a Vyprázdni loďku

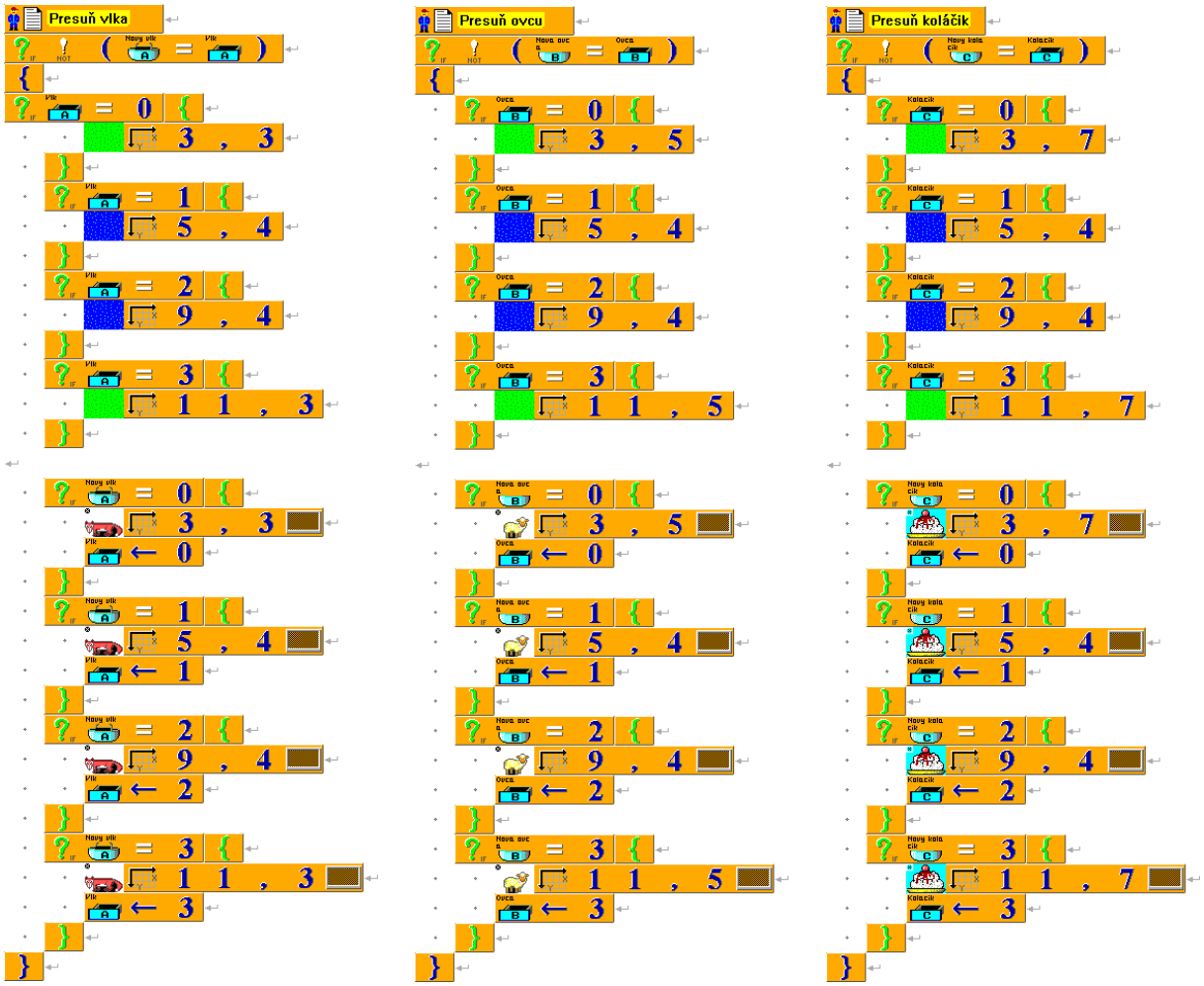
Títo dvaja pomocníci sú veľmi jednoduchí. Pomocník *Kresli scénu* okrem nakreslenia predvolenej scény (č. 0) a pasažierov nastaví globálne premenné **A** až **D** na nuly, čo bude znamenať ľavý breh. Pre premenné **A** až **C** sú povolené ďalšie hodnoty 1, 2, 3 s významom: 1. na loďke na ľavom brehu, 2. na loďke na pravom brehu a 3. na pravom brehu (na súši). Pre premennú **D** sú povolené len hodnoty 0 a 1 – ľavý a pravý breh.

Pomocník *Vyprázdni loďku* iba skontroluje, či sa niektorý pasažier nachádza na loďke (pri ľubovoľnom brehu) a ak áno, tak ho vysadí na najbližší breh.

Presuň vlka, Presuň ovcu a Presuň koláčik

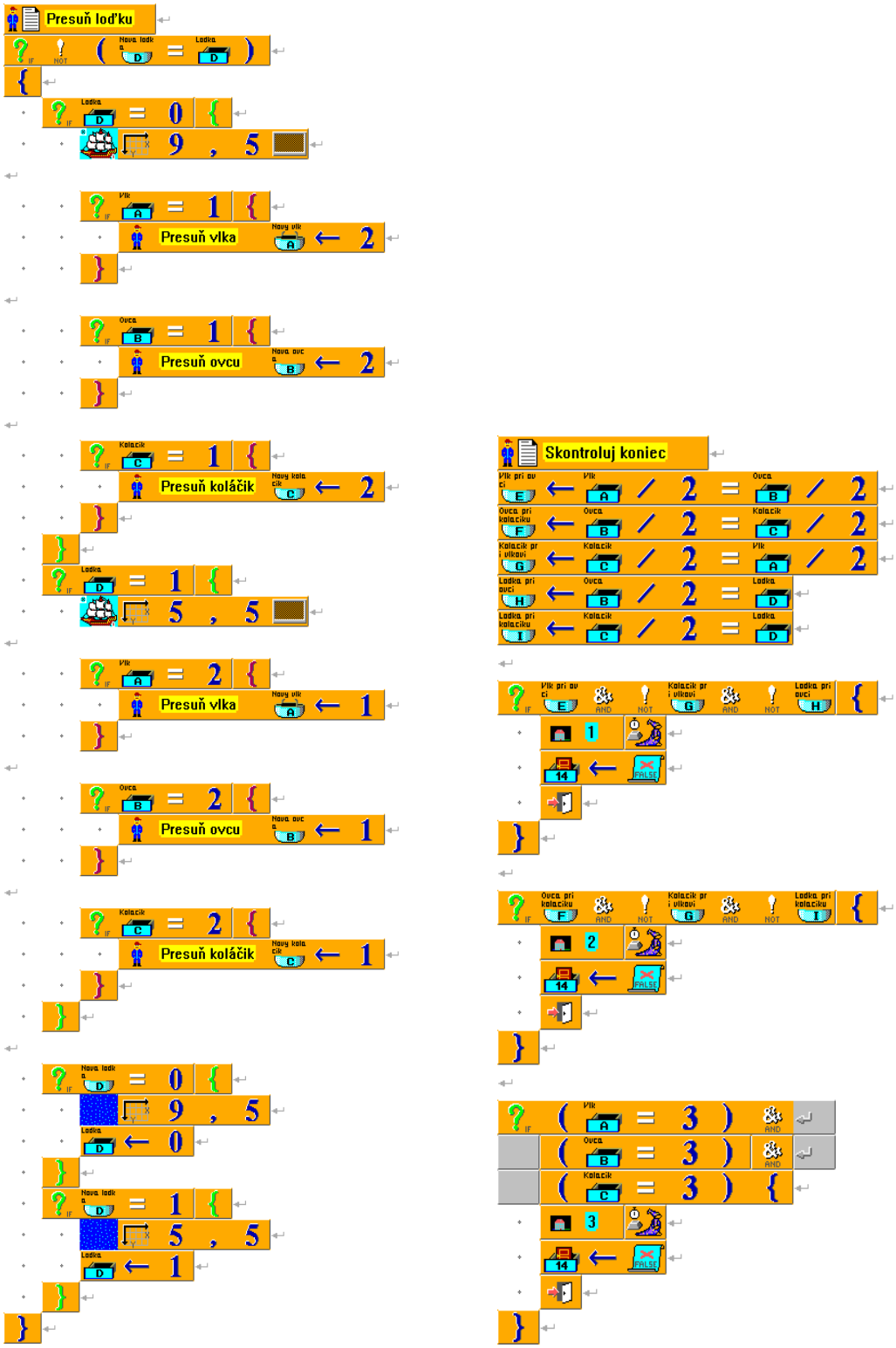
Títo traja pomocníci sú takmer navlas rovnakí. Líšia sa len v tom, o ktorého pasažiera sa starajú. Principiálne robia to isté. Overia, či sa požadovaná pozícia líši od aktuálnej, a potom postupujú takto:

1. vymažú postavičku z jej aktuálnej pozície,
2. nakreslia postavičku na jej novej pozícii,
3. zapamätajú si novú pozíciu.



Poznámka: Toto bolo možné riešiť aj tak, že priradenie do premenných **A, B, C** sa dalo urobiť len raz (po posledných štyroch *podmienených spracovaniach*), ale pravda je, že tým by vzniklo určité nepravdepodobné riziko, že sa v premennej ocitne iná hodnota, než je reálna poloha postavičky. (Čo by by však tiež nemuselo spôsobovať problémy. Je však vhodné implementovať také riešenia, pri ktorých takáto neistota nevzniká.)

Presuň loďku a Skontroluj koniec



Pomocník *Presuň loďku* sa od predchádzajúcej trojice líši tým, že 1. sa menia len dva stavy (namiesto štyroch) – ľavý a pravý breh – a ešte tým, že 2. loďka zároveň sama so sebou presúva aj prípadného pasažiera, ktorého má na palube. Na to s výhodou používa predchádzajúcich troch pomocníkov.

Posledný pomocník *Skontroluj koniec* postupne overí všetky situácie, keď môže nastať úspešný alebo neúspešný koniec hry:

1. Keď zostane sám vlk s ovcou, zje ju – zobrazí sa scéna vlk-koza-kapusta.S01 a hra sa končí.
2. Keď zostane ovca sama s koláčikom, zje ho – zobrazí sa scéna vlk-koza-kapusta.S02 a hra sa tiež končí.
3. Keď sú všetci traja pasažieri úspešne na pravom brehu, zobrazí sa posledná definovaná scéna vlk-koza-kapusta.S03 a hra sa tento raz končí úspechom.

Úloha na záver

Prepracujte hru z tejto kapitoly do programovacieho prostredia Scratch.

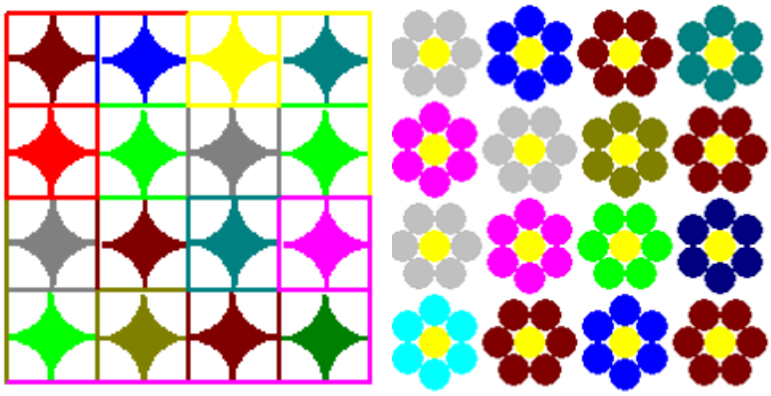
6 Námety na riešenie

Ako je uvedené v úvode 4. kapitoly v časti **tip,** autori predkladanej učebnice považujú za vhodný spôsob zdokonaľovania sa v programovaní analýzu a modifikáciu jestvujúcich zadaní a ich riešení – buď úpravu predloženého vzorového riešenia, alebo úplnú reprodukciu funkčného hotového riešenia bez možnosti náhľadu do zdrojového kódu. Táto kapitola poskytuje druhú možnosť – predstavuje niekoľko nápadov na vlastnú implementáciu.

6.1 Fraktály a opakujúce sa vzory

Zaujímavými oblasťami príkladov na riešenie je tvorivé programovanie kreslenia rôznorodých farebných motívov alebo mozaikových šablón s využitím náhodnosti a fraktály – geometrické útvary založené na opakovaní kreslenia podľa jednoduchého pravidla (pozri aj kapitoly 4.4 a 4.5).

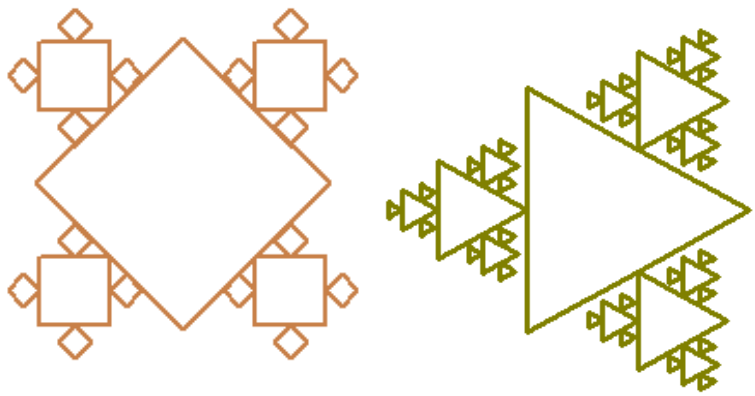
Na obrázkoch 6.1 a 6.2 je uvedených niekoľko rôznych námetov na riešenie vo forme jednoduchých rekurzívnych grafických vzorov, ktoré sa dajú využiť aj v detskom programovacom jazyku Scratch. S týmito vzormi sa používateľ môže stretnúť aj v programovacom prostredí Imagine.



Obrázok 6.1. Príklady grafických výstupov programového riešenia.

Učiteľ má k dispozícii množstvo námetov na vyučovanie programovania v detských programovacích jazykoch, ktoré sú voľne dostupné na internete. Učiteľ si môže vyselektovať vhodné príklady v závislosti od vyučovaného jazyka, prípadne aj od vedomostnej úrovne žiakov, ktorých aktuálne vyučuje. Tvorivý učiteľ obyčajne používa vlastné námety a často čerpá námety na riešenie z iných predmetov.

Vhodnou motiváciou učenia sa základov programovania môže byť vyučovanie programovania prostredníctvom programovania hry. Práve táto oblasť zatraktívňuje vyučovanie programovania už najmladším žiakom. Je nesmierne dôležité, aby vyučovanie programovania bolo pre žiaka zaujímavé. Žiaci si v daných úlohách definujú vlastné nové bloky, ktoré následne využívajú.



Obrázok 6.2. Grafický výstup z rekurzívneho programu.

Ďalšími námetmi môžu byť návrhy na vykreslenie rôznych šetričov obrazovky napríklad v tvare náhodne vykresľujúcich sa krúžkov alebo iných objektov, prípadne môže byť zmenené/upravené pozadie jestvujúcich alebo úplne nových návrhov.

6.2 Didaktické hry v JavaScripte

V tejto podkapitole je predstavená implementácia množiny úloh – didaktických hier Ladislava Végha, ktoré sú naprogramované v JavaScripte. Pri riešení týchto úloh je potrebné využiť poznatky hlavne z logiky a kombinatoriky. Majú príjemný grafický dizajn, jednoduché interaktívne ovládanie a tri stupne náročnosti. Sú to vlastne tri rôzne verzie jednej úlohy – ako previesť *objekty* hry člnom z jednej strany (brehu) rieky na druhú. Na úspešné vyriešenie každej úrovne úlohy treba dodržiavať presne stanovené pravidlá. Množina je dostupná v troch jazykoch na stránke: https://anim.ide.sk/prekrocenie.php s názvom **Interaktívne animácie na vyučovanie algoritmov a programovania.**

Úloha: Kapusta, koza, vlk a dievča – 1. stupeň náročnosti

Dievča ide domov z trhu, kde kúpilo jednu kapustu, jednu kozu a jedného vlka. Cestou domov má prejsť cez rieku, na ktorej je jeden čln. Pádlovať vie iba dievča a navyše, čln je taký malý, že okrem dievčaťa sa doň zmestí iba jeden ďalší objekt. Dievča pri prechode riekou nemôže nechať ani kozu s kapustou (inak by koza zjedla kapustu), ani vlka s kozou (inak by vlk zjedol kozu). Ako má dievča previezť vlka, kozu a kapustu cez rieku? *Pomôžte – poraďte im!*

Obrázok 6.4. Začiatočná situácia pri rieke.

Obrázok 6.5. Posledná „jazda“ loďkou cez rieku.

Úloha: Vojaci a deti –2. stupeň náročnosti

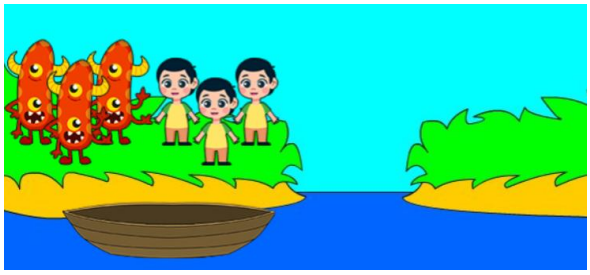
Traja vojaci a dvaja chlapci chcú prejsť cez rieku člnom. Čln je taký malý, že doň sa zmestí iba jeden vojak alebo maximálne dvaja chlapci. Ako majú vojaci a chlapci prejsť cez rieku? *Pomôžte a poraďte im, ako to majú urobiť s najmenším počtom presunov člna cez rieku!*

Obrázok 6.6. Začiatočná situácia pri rieke (druhý stupeň náročnosti úlohy).

Obrázok 6.7. Koniec hry – vyriešená úloha. Všetci vojaci aj deti sa dostali na druhú stranu rieky.

Úloha: Príšery a deti – 3. stupeň náročnosti

Tri príšery a traja chlapci chcú prejsť cez rieku člnom. Do člna sa zmestia maximálne dva objekty. Aby sa predišlo tragédii, nemôže byť ani na jednej strane rieky viac príšer ako detí (inak by príšery zjedli deti). Ako majú príšery a chlapci prejsť cez rieku?



Obrázok 6.8. Začiatočná situácia pri rieke (tretí stupeň náročnosti).

Obrázok 6.9. Chybný krok.

Záver

Programovať sa nikto nenaučí tak, že si prečíta knihu venovanú programovaniu alebo vypočuje prednášky z programovania. Programátorom sa človek stáva predovšetkým programovaním. Ak chce byť niekto dobrým programátorom, musí vyriešiť mnoho úloh a riešenia doviesť až do funkčnej programovej realizácie.

Mnohí aj v súčasnosti programovanie považujú za umenie. Treba ale povedať, že toto „umenie“ má svoje pravidlá, ktoré dokáže výhodne a rutinne uplatňovať len ten, kto sa ich naučil a dobre natrénoval častým používaním. Rozumný programátor sa neuspokojí s hocijakým (i keď z hľadiska získaných výsledkov správnym) riešením predloženej úlohy, ale hľadá čo najlepšie, „optimálne“ riešenie problému. To optimálne riešenie sa dotýka rovnako algoritmickej stránky aj programovej realizácie riešenia problému.

V súčasnosti, keď je už využívanie počítačov pri vyučovaní považované za samozrejmé, je dôležité, aby učitelia vedeli deti motivovať nielen k využívaniu už jestvujúcich aplikácií, ale aj k tvorbe ich vlastných. V predkladanej učebnici sú námety na to, akým spôsobom zasvätiť žiakov – začínajúcich programátorov – do tajov programovania v detskom programovacom jazyku Scratch. Sú v nej, okrem opisu programovacích prostredí, vysvetlenia jednotlivých príkazov, základných riadiacich štruktúr, skriptov, okien a objektov. Cennou časťou učebnice je zbierka riešených úloh v samostatnej kapitole. V ďalších kapitolách sa nachádzajú riešenia komplexnejších problémov. Opis riešenia jednotlivých úloh má rovnakú štruktúru: zadanie, analýza problému, opis postupu a metodiky riešenia a prislúchajúcim programovým kódom, ktorý je doplnený o vizualizáciu jednotlivých riešení. Publikácia je doplnená o námety na tvorbu vlastných miniprojektov.

Prajeme vám veľa úspechov a chuť do práce pri tvorbe programov, na riešenie jednoduchých úloh, ako aj pri realizácii zložitejších programových projektov.

Literatúra

1. BLAHO, A.: *Comenius Logo.* 2002. Dostupné na: https://dai.fmph.uniba.sk/~blaho/comlogo/index.html. Naposledy pristúpené: 7. 7. 2025.
2. BLAHO, A.: *Imagine.* 2005. Dostupné na: https://imagine.input.sk/. Naposledy pristúpené: 7. 7. 2025.
3. CZAKÓOVÁ, K. – STOFFOVÁ, V.: Animačné modely v didaktických aplikáciách vytvorených v LogoMotion. In Hájková, E. – Vémolová, R. (ed.): *XXX. International Colloquium on the Management of Educational Process. Proceeding of electronic version of reviewed contributions (CD-ROM).* Brno : Univerzita obrany, 2012, s. 66–69. ISBN 978-80-7231-865-0.
4. GABAĽOVÁ, V.: *Problémové vyučovanie programovania* [dizertačná práca]. Univerzita Konštantína Filozofa v Nitre, Fakulta prírodných vied. Školiteľ: prof. Ing. Veronika Stoffová, CSc. Stupeň odbornej kvalifikácie: PhD. Odbor: teória vyučovania informatiky, Nitra : FPV, 2010. 152 s.
5. HORVÁTH, R.: *Príklady v prostredí Scratch.* Trnava : Pedagogická fakulta Trnavskej univerzity v Trnave, 2019. Dostupné na: https://pdfweb.truni.sk/horvath/materialy?scratch-examples. Naposledy pristúpené: 7. 7. 2025.
6. KALAŠ, I., WINCZER, M.: *Tvorivá informatika.* Bratislava : SPN, 2007. 48 s. ISBN 978-80-10-00887-2.
7. NASSI, I., SHNEIDERMAN, B.: *Flowchart techniques for structured programming.* SIGPLAN Notices XII, August 1973.
8. STOFFA, V.: *Algoritmizáció és programozás I.* Komárom : Selye János Egyetem, 2005. 174 s. ISBN 80-969251-7-2.
9. STOFFOVÁ, V.: Algoritmy v počítačovom testovaní vedomostí I. In *Technológia vzdelávania (Slovenský učiteľ),* ročník VIII, 2000, č. 9, s. 15–18.
10. STOFFOVÁ, V.: Computer-aided learning of programming. In *Proceedings of the International Conference of Computer Systems and Technologies (e-Learning).* Editors B. Rachev, A. Smrikarov, Sofia : Bulgarian Union of Automation and Informatics, 2003, s. IV.33-1–IV.33-6. ISBN 954-9641-33-3.
11. STOFFOVÁ, V. – CZAKÓOVÁ, K.: Propedeutika programovania a nová školská reforma. In *Trendy ve vzdělávání 2012* : Díl II. 1. vyd. Editori Martin Havelka, Miroslav Chráska, Milan Klement a Čestmír Serafín. Olomouc : Agentura Gevak, 2012, s. 588–594. ISBN 978-80-86768-36-6.
12. STOFFOVÁ, V.: Didaktika informatiky v príprave budúcich učiteľov informatiky a informačnej výchovy (Didactics of computer science in informatics and information technologies education teacher training) In *Education – Technology – Computer Science (Edukacja – Technika – Informatyka)* : Quarterly journal No. 1/15/2016, Kwartalnik naukowy NR 1/15/2016.
13. STOFFOVÁ, V.: Didaktické hry. In *Zborník z vedeckého seminára Quo vadis výchova…? na tému „Hra a hračka.“* Bratislava : IUVENTA Bratislava, 1994, s. 61–65.

14. STOFFOVÁ, V. et al.: *Informatika, informačné technológie a výpočtová technika : Terminologický a výkladov slovník.* 1. vyd. Nitra : Fakulta prírodných vied UKF v Nitre, 2001. 230 s. ISBN 80-8050-450-4.

15. STOFFOVÁ, V. – LOVÁSZOVÁ, G.: Testovanie vedomostí z informatiky. In *Zborník 2 z vedeckej konferencie MEDACTA '97 – Vzdelávanie v meniacom sa svete.* 1. vyd. Nitra : Slovdidac, 1997, s. 487–493.

16. STOFFOVÁ, V.: O potrebe zavedenia predmetu Tvorba elektronických učebných pomôcok do učiteľskej prípravy. In Jiří Dostál (Ed.): *INFOTECH 2007 Moderní informační a komunikační technologie ve vzdělávání* : Díl 1. Olomouc : Votobia Olomouc, 2007, s. 34–37. ISBN 978-80-7220-301-7.

17. STOFFOVÁ, V.: *Počítač – univerzálny didaktický prostriedok.* 1. vyd. Nitra : Fakulta prírodných vied UKF v Nitre, 2004. 172 s. ISBN 80-8050-450-4.

18. STOFFOVÁ, V. – SKALKA, J. – MORVAI, M.: Didaktické počítačové hry. In *Informatika v škole,* 1995, č. 12, s. 19–24.

19. STOFFOVÁ, V.: Súčasné problémy prípravy učiteľov (z) informatiky. In *DidInfo 2016 : 19. ročník národnej konferencie o vyučovaní informatiky.* Ed. Ľ. Trajteľ. Banská Bystrica : Univerzita Mateja Bela, Fakulta prírodných vied, 2016, s. 222–228. ISBN 978-80-557-0527-9.

20. STOFFOVÁ, V. – TÓTH, K.: Integrácia IKT a mikrosvetov do vyučovania na ZŠ (Integration of ICT and micro-worlds into the education at elementary schol). In Stoffová, V. (ed.): *XXI. DIDMATTECH 2008 1st part.* Eger – Komárno : Eszterházy Károly Főiskola, Univerzita J. Selyeho, 2009. 203–208 s. ISBN 978-963-9894-17-4.

21. STOFFOVÁ, V.: *Zbierka úloh z programovania I.* Nitra : Vysoká škola pedagogická v Nitre, 1993. 152 s.

22. ŠTOFOVÁ, V. – CINGEL, P. – PANČÍK, D.: *Zbierka úloh z algoritmizácie a programovania I.* Liptovský Mikuláš : VVTŠ ČSSP, 1985. 132 s.

23. ŠTOFOVÁ, V. – CINGEL, P. – PANČÍK, D.: *Zbierka úloh z algoritmizácie a programovania II.* Liptovský Mikuláš : VVTŠ ČSSP, 1985. 130 s.

24. VARGA, M. – BLAHO, A. – ZIMANOVÁ, R.: *Informatika pre gymnáziá. Algoritmy s Logom : Tematický zošit.* Slovenské pedagogické nakladateľstvo, 1999. 48 s. ISBN 80-08-02965-X.

25. VÉGH, L.: *Interaktívne animácie na vyučovanie algoritmov a programovania.* Komárno : 2012 – 2020. Dostupné na: https://anim.ide.sk/prekrocenie.php. Naposledy pristúpené: 7. 7. 2025.

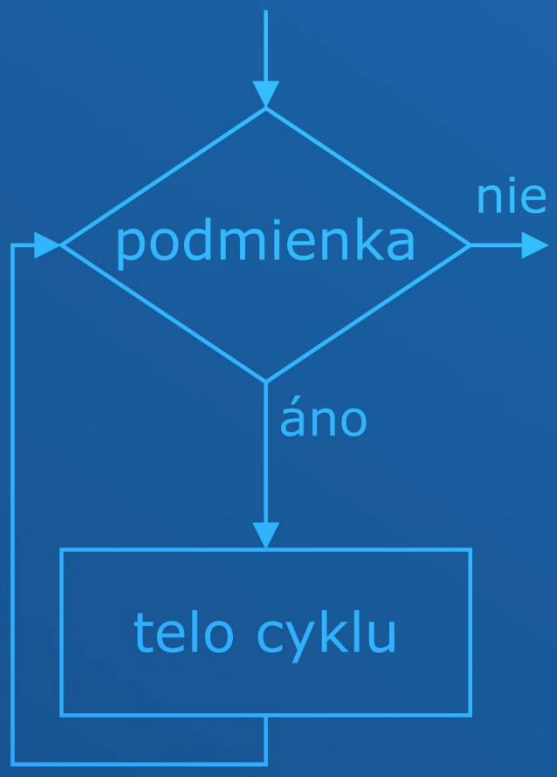
Obálka

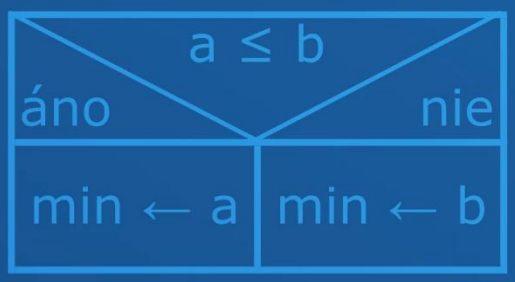
Grafika obálky bola vytvorená s použitím vlastných obrázkov a obrázkov voľne dostupných na nasledujúcich adresách:

- https://freesvg.org/vector-scratch-cat
- https://commons.wikimedia.org/wiki/File:Sierpinski_triangle_2nd_iteration.svg

Názov: Detské programovacie jazyky ako úvod do programovania
Autori: prof. Ing. Veronika Stoffová, CSc., Mgr. Ing. Roman Horváth, PhD., PaedDr. Veronika Gabaľová, PhD.
Recenzenti: prof. PaedDr. Alena Hašková, CSc., doc. PhDr. Miroslav Chráska, Ph.D.
Vydavateľ: Pedagogická fakulta Trnavskej univerzity v Trnave
Vydanie: prvé
Rok vydania: 2025
Počet strán: 80

ISBN 978-80-568-0210-6
https://doi.org/10.31262/978-80-568-0210-6/2025





9 788056 802106