TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta přírodovědně-humanitní a pedagogická

DIDAKTIKA PROGRAMOVÁNÍ

Jindra Drábková

2019

Tento vzdělávací materiál vznikl v rámci projektu
CZ.02.3.68/0.0/0.0/16_036/0005322 **Podpora rozvíjení informatického myšlení**.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Předmluva

Tato skripta jsou určena studentům, kteří se připravují na povolání učitele informatiky a v předchozím studiu již absolvovali několik předmětů týkajících se algoritmizace a programování. Skripta by měla pomoci studentům v posílení schopností předávat své znalosti žákům. Skripta obsahují úvod do předmětu Didaktika programování a jedenáct témat. U každého tématu je uveden cíl a požadavky, dále je vysvětlen odborný pojem, podrobně rozebrán průběh vyučovací jednotky a na závěr jsou uvedeny otázky k tématu a příkladům do případné diskuse. Součástí všech témat jsou ukázkové příklady, na kterých jsou vysvětleny jednotlivé programové (řídicí) struktury. Příklady jsou voleny tak, aby na nich bylo ukázáno postupné nabalování problémů. Pro jednoduchost je použit jazyk Scratch, který je podle našeho názoru pro žáky základních škol nejvhodnější. Scratch je použit ve výuce jen jako prostředek, je možné použít jakýkoli jiný programovací jazyk s tím, že je potřeba některé příklady přizpůsobit danému programovacímu jazyku. Programovací jazyk Scratch je k dispozici online stejně jako všechny programy uvedené v těchto skriptech[1]. V poslední kapitole jsou pro inspiraci další příklady a časté chyby žáků.

Při výuce programování jsou v našem případě nejčastěji použity dvě metody: metoda problémového výkladu a výzkumná metoda. U metody problémového výkladu je vhodné na začátku definovat společně s žáky problém a pak ho společně řešit. Je dobré s žáky diskutovat o možných řešeních, o jejich vhodnosti, ověřit jednotlivá řešení, popř. je upravit. U výzkumné metody žáci samostatně hledají řešení, součástí metody je zkoumání. U této metody záleží na intelektu žáků a je potřeba ji používat přiměřeně v závislosti na intelektuálním rozložení žáků ve třídě. Při výuce je vhodné použít dialog mezi žáky a učitelem nebo mezi žáky navzájem ve formě burzy dobrých nápadů (brainstormingová metoda). Tímto způsobem je možné hledat nejvhodnější (optimální) řešení daného problému.

Věřím, že se skripta stanou dobrou pomůckou nejen pro výuku předmětu Didaktika programování, ale i pro samotnou výuku programování na základní škole.

Jindra Drábková

[1] Scratch je dostupný na stránkách https://scratch.mit.edu. Programy jsou ke stažení v e-learningovém kurzu Didaktika programování, který je umístěn na stránkách https://elearning.fp.tul.cz/course/view.php?id=2506.

Přehled témat

Úvod do předmětu Didaktika programování

Sekvence příkazů

Cyklus se známým počtem kroků

Použití proměnných

Řídící proměnná cyklu v cyklu se známým počtem kroků

Podprogram, podprogram s parametry

Podmíněný příkaz neúplný, relační a logické operátory

Podmíněný příkaz úplný

Rozpracování dvou komplexních aplikací

Cyklus s podmínkou

Seznamy

Rekurze

Další inspirace

Úvod do předmětu Didaktika programování

Cílem předmětu Didaktika programování není naučit studenty programovat na profesionální úrovni nebo je naučit v širokém rozsahu určitý programovací jazyk (v našem případě Scratch). Cílem je seznámit je se základními programovými konstrukcemi v určitém programovacím jazyce tak, aby byli schopni v budoucnu programování vyučovat, a to nezávisle na programovacím jazyku. Dále je důležité, aby (jako učitelé informatiky) uměli vysvětlit základy programování tak, aby žáci mohli na těchto základech stavět a dále je rozvíjet (např. při odborném studiu).

V prvním bloku si studenti zopakují pojmy, se kterými se již setkali v předmětech týkajících se algoritmizace a programování. Cílem těchto skript není tyto základní pojmy podrobně vysvětlovat, studenti by je měli znát z předchozího studia. Na začátku každého tématu je stručný popis (definice) pojmů, které s daným tématem souvisí.

- Algoritmus
- Základní pojmy: identifikátor, proměnná, konstanta
- Datové typy
    - jednoduché (celá čísla, logické hodnoty, znaky, reálná čísla)
    - strukturované (pole, řetězec)
- Operace, operátory, priorita operátorů
    - aritmetické
    - relační
    - logické
- Základní příkazy
    - příkaz vstupu
    - příkaz výstupu
    - přiřazovací příkaz
- Programové konstrukce
    - složený příkaz
    - podmíněný příkaz (úplný, neúplný)
    - přepínač (switch)
    - cyklus (se známým počtem kroků, s podmínkou na začátku, s podmínkou na konci)
- Podprogramy
    - proměnné lokální a globální
    - parametry
- Základní pojmy objektově-orientovaného programování (třída, objekt, vlastnosti OOP)

Ve výuce programování je možné postupovat různými způsoby:

- Žáci pracují převážně podle pracovních listů, učitel zasahuje do výuky velmi málo; když mají žáci problém, který nedokážou vyřešit, tak jim učitel pomůže.
- Učitel seznámí žáky s prostředím programovacího jazyka a postupně s jednotlivými příkazy, které žáci následně použijí při vytváření programů; programy vytvářejí žáci

samostatně, popř. s učitelovou pomocí, na závěr probíhá diskuse o možných postupech.

- Žáci se seznamují s prostředím programovacího jazyka a vytvářejí programy s pomocí nápovědy, popř. návodů, které jsou u daného programovacího jazyka k dispozici.

Sekvence příkazů

Cíl, požadavky

Cíl: ukázat žákům sekvenci příkazů na příkladech vykreslování jednoduchých geometrických tvarů. Žáci se seznámí se základním prostředím dětského programovacího jazyka Scratch.

Vstupní požadavky: základní znalosti z oblasti geometrie (základní geometrické tvary, úhly).

Vymezení pojmu

Algoritmus je postup, návod, jak danou úlohu řešit. Je to popis řešení sestávající z elementárních kroků. Nejjednodušší struktura, která se při vytváření algoritmu (programu) používá, je sekvence příkazů. Sekvence příkazů je posloupnost jednotlivých kroků (příkazů) v algoritmu (programu), ve kterém nedochází k větvení programu nebo k opakování nějaké části programu.

Struktura vyučovací jednotky

Na začátku učitel diskutuje s žáky o algoritmech (postupech) v reálném životě. Žáci by měli vymýšlet příklady, učitel by je měl korigovat v tom, aby uváděli příklady jen za sebou jdoucích příkazů (bez rozhodování, opakování). Příklady: návody na sestavování modelů, návod pro placení parkovného, jízdenek, automatu na kávu, čokoládu apod. Je dobré s žáky probrat, co rozumí pod pojmem elementární kroky.

V další části budou žáci pracovat v dětském programovacím jazyce Scratch. Seznámí se s principem vkládání jednotlivých příkazů pomocí přetahování, konkrétně s některými příkazy v nabídce **POHYB** (*DOPŘEDU*, *OTOČ SE*), v nabídce **VZHLED** (*UKAŽ SE*, *SKRYJ SE*) a s některými příkazy v nabídce **PERO** (*SMAŽ, PERO ZAPNI*, *PERO VYPNI*)[2]. Rychlejší žáci mohou změnit postavě kostým, který ukazuje směr, jak je postava natočena (např. šipku). Kostým se mění v záložce **KOSTÝMY**, po kliknutí na ikonu *Vyber kostým*, která je umístěna vlevo dole (šipka je kostým nazvaný Arrow). Ostatní kostýmy je možné odstranit ze seznamu kostýmů kliknutím na kostým a na křížek vpravo nahoře.

Před samotným kreslením je vhodné žáky seznámit s rozměry plátna a s tím, kde je počátek (souřadnice 0, 0). Žáci mohou rozměry i počátek zjistit sami, a to pohybem myši po plátně (počátek je uprostřed plátna, pr. hor. roh: 240, 180, pr. dol. roh 240, –180, levý dol. roh –240, –180, levý hor. roh –240, 180).

Nejdříve žáci mohou experimentovat (klikat na příkazy, a tak zjistit, co který příkaz vykonává). Posléze by měli vytvářet posloupnosti příkazů (vykreslení čtverce, trojúhelníku, popř. šestiúhelníku, osmiúhelníku). Je dobré naučit žáky, že po kliknutí na zelený praporek je

[2] Ve verzi Scratch 3.0 je potřeba vybrat nabídku PERO v rozšíření (vlevo dole klik na ikonu Přidej rozšíření).

možné celý program spustit. V tomto případě se na začátek posloupnosti příkazů vkládá příkaz *PO KLIKNUTÍ NA* , který je umístěn v nabídce **UDÁLOSTI**.

Možná posloupnost příkazů pro vykreslení čtverce:

Možná posloupnost pro vykreslení trojúhelníku:

Rychlejší žáky je možné nechat experimentovat i s dalšími příkazy z nabídky **POHYB**, **VZHLED**, **OVLÁDÁNÍ**.

Diskuse k tématu a příkladům

1. Je nutné poslední otočení u čtverce? Proč není u trojúhelníku?
2. Co se stane, když na začátku nebude příkaz smaž, natočení směrem nahoru nebo skok na pozici [0, 0]?
3. Je možné programy zjednodušit? Jak?
4. Diskuse nad přidáním příkazu pro opakování.

Cyklus se známým počtem kroků

Cíl, požadavky

Cíl: použití cyklu se známým počtem kroků (opakování) při vykreslování základních geometrických tvarů.

Vstupní požadavky: základní práce v prostředí dětského programovacího jazyka Scratch.

Vymezení pojmu

Cyklus se známým počtem kroků (opakování) se používá v případě, když je potřeba nějakou část programu opakovat, přičemž se zadává počet opakování. Tento cyklus se často nazývá FOR cyklus. Ve většině programovacích jazyků se v cyklu tohoto typu používá řídící proměnná cyklu, jejíž hodnota se mění v závislosti na tom, po kolikáté cyklus probíhá. V programovacím jazyce Scratch řídící proměnná v cyklu tohoto typu není.

Struktura vyučovací jednotky

Na začátku učitel s žáky probírá možné algoritmy (postupy), kde dochází k opakování (opakování cviků při tréninku, repetice v notovém zápise, opakování činnosti na výrobní lince).

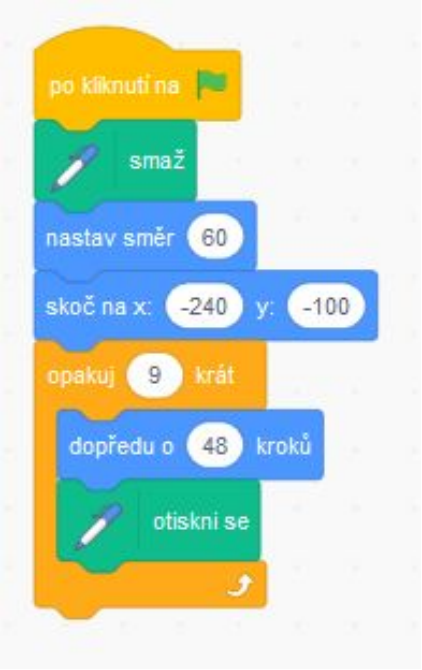
Žáci si otevřou program na vykreslení čtverce a učitel vede diskusi o tom, jak je možné použít v tomto případě cyklus. Diskusí se dojde k závěru, že v případě, že se nějaká část programu opakuje několikrát, je vhodné použít cyklus. Z nabídky **OVLÁDÁNÍ** žáci použijí příkaz *OPAKUJ* (4krát). Dále žáci zkusí nakreslit trojúhelník, rychlejší žáci mohou nakreslit víceúhelník (vždy s využitím cyklu).

Možné řešení:

Další příklad na použití cyklu s předem daným počtem opakování může být vykreslení hvězd do řádku. Výsledek může vypadat takto:

V tomto případě je třeba seznámit žáky s příkazem *OTISKNI SE* (nabídka **PERO**) a příkazem *SKOČ NA POZICI* (nabídka **POHYB**). Je možné nakreslit i šikmou řadu hvězd. V tomto případě se používá příkaz *NASTAV SMĚR* z nabídky **POHYB**. Rychlejší žáci mohou při vykreslování hvězd použít čekání (nabídka **OVLÁDÁNÍ**) nebo zvukový signál (nabídka **ZVUK**). Je dobré žákům nechat prostor, aby prozkoumali, jak se jednotlivé příkazy chovají, zda je nutné zachovat pořadí příkazů, co se stane při prohození příkazů v cyklu apod.

Jedno z možných řešení pro šikmou řadu:

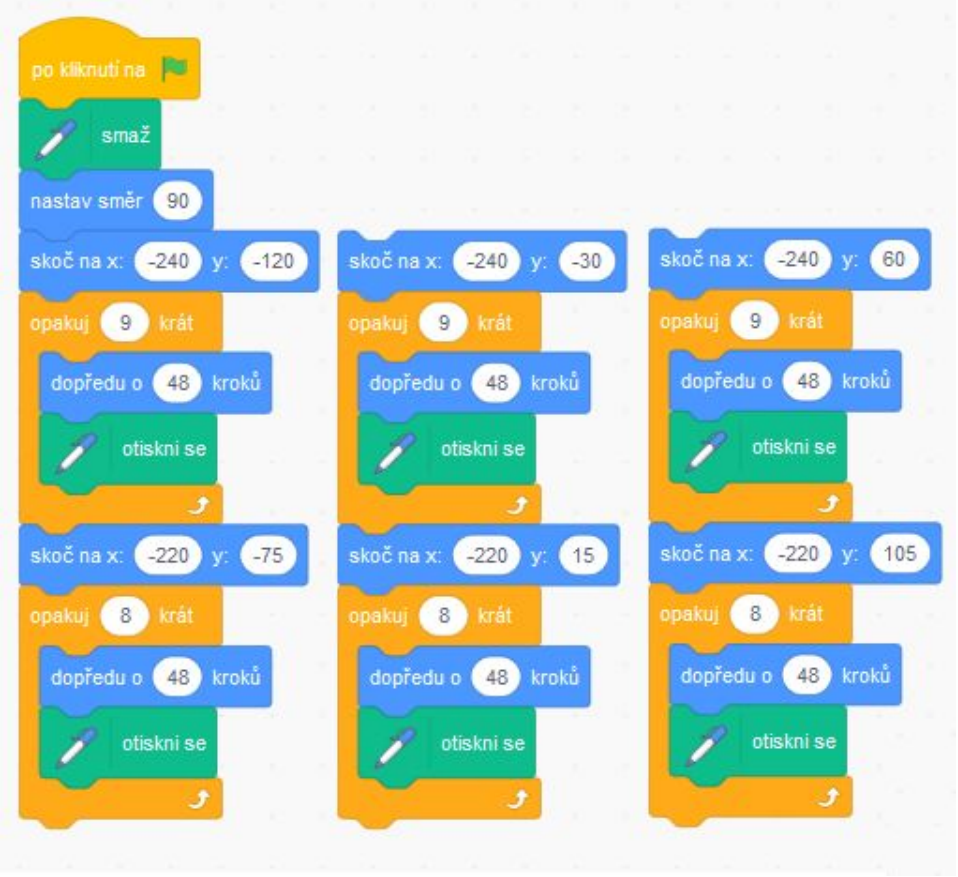


V případě, že se prohodí v cyklu příkazy *DOPŘEDU* a *OTISKNI SE*, dojde k tomu, že se vykreslí o jednu hvězdu více.

V dalším kroku mohou žáci vytvořit několik řad nad sebou, počet v lichých řadách může být jiný než v sudých, řady mohou být posunuté, v řadách se mohou postupně snižovat počty hvězd, žáci mohou vykreslit pyramidu apod. Je dobré nechat žákům prostor, každý může použít jiný postup, případně žákům rozdat obrázky, co by mohli udělat.

Řady hvězd s jiným počtem v sudých a lichých řádcích mohou vypadat takto:

Jedno z možných řešení, na kterém jsou vidět opakující se části:



Pyramida se čtyřmi patry může vypadat takto:

Jedno z možných řešení, na kterém jsou vidět opakující se části:

Ve všech výše uvedených případech by žáci měli přijít na to, že pokud budou vkládat cykly pro vykreslení jednotlivých řad za sebou, program bude dlouhý a části programu se budou opakovat (jen např. s jiným posunutím). Diskusí by je učitel měl navést na použití cyklu v cyklu a použití proměnných x, y, ve kterých jsou uloženy informace o poloze a které je možné měnit pomocí příkazů *ZMĚŇ X O*, *ZMĚŇ Y O, NASTAV X NA*, *NASTAV Y NA* (nabídka **POHYB**).

Diskuse k tématu a příkladům

1. Co je třeba upravit, když chceme nakreslit čtverec s jinou velikostí strany?
2. Co je třeba upravit, když chceme natočit trojúhelník jiným směrem?
3. Jak program upravit, kdybychom chtěli nakreslit objektů (čtverců, trojúhelníků) více.
4. Diskuse nad různými řešeními vykreslení řady hvězd, popř. pyramidy.
5. Bylo by možné použít cyklus v cyklu? Jak? Co by bylo potřeba?

Použití proměnných

Cíl, požadavky

Cíl: použití proměnných a náhodných čísel, zadávání hodnot z klávesnice, použití cyklu v cyklu v příkladech řady hvězd a pyramida.

Vstupní požadavky: používání základních příkazů a příkazu cyklu s pevným počtem opakování v prostředí Scratch.

Vymezení pojmu

Proměnná označuje místo v paměti, při běhu programu mění svoji hodnotu. K označení proměnné se používá identifikátor, což je název proměnné, a je dobré ho volit tak, aby co nejvýstižněji proměnnou charakterizoval. V některých programovacích jazycích je potřeba zadat, jakého typu proměnná je. Datové typy mohou být jednoduché nebo strukturované. Jednoduché datové typy jsou ordinální (pozici lze očíslovat, mají předchůdce a následníka) nebo neordinální. Mezi ordinální typy patří celá čísla, znaky, logické proměnné true a false. Neordinální datový typ jsou např. reálná čísla. Do strukturovaných datových typů se řadí pole a řetězec. Pokud je proměnná použita v celém programu, nazývá se globální, pokud je použita jen v podprogramu, jde o lokální proměnnou.

Existují proměnné, které jsou v daných programovacích jazycích předpřipravené a které je možné v průběhu programu měnit (tzv. systémové proměnné). V programovacím jazyce Scratch k takovým proměnným patří již zmiňované proměnné X a Y, ve kterých jsou hodnoty souřadnic, dále proměnná směr, ve které je uložen směr natočení. Tyto proměnné se nacházejí v nabídce **POHYB**.

Je možné zadat svou vlastní proměnnou. V programovacím jazyce Scratch se datový typ proměnné nezadává. Zadává se jen jednoduchá proměnná (*VYTVOŘ PROMĚNNOU*) nebo strukturovaná proměnná (*VYTVOŘ SEZNAM*) v nabídce **PROMĚNNÉ**. Dále se zadává, jestli se má vytvořit proměnná pro všechny postavy (globální proměnná), nebo jen pro tuto postavu (lokální proměnná).

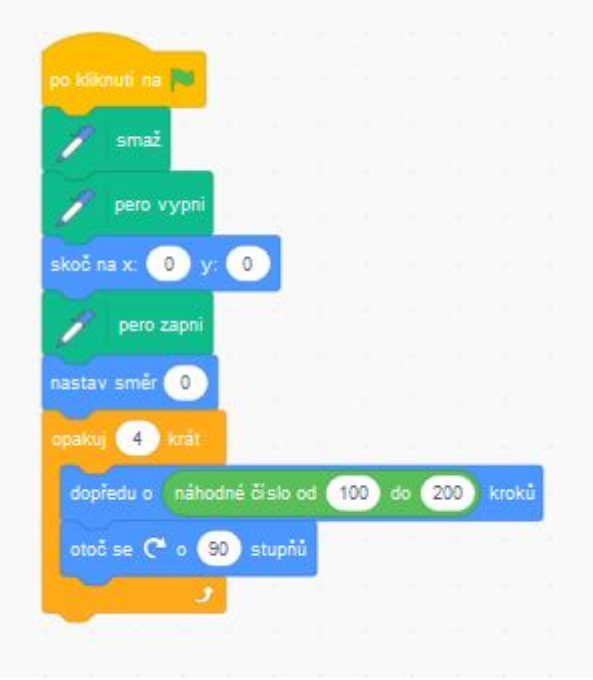
Struktura vyučovací jednotky

Na začátku je možné navázat na diskusi a příklady z vyučovací jednotky Cyklus se známým počtem kroků a vysvětlit pojem proměnná, a to konkrétně na proměnných X a Y, které v programovacím jazyce Scratch nesou hodnoty souřadnic.

Poté učitel s žáky diskutuje o tom, co je potřeba v programu pro vykreslení čtverce upravit, když chceme, aby strana čtverce měla při každém spuštění programu jinou (náhodnou) délku.

Pro generování náhodného čísla se používá v programovacím jazyce Scratch příkaz *NÁHODNÉ ČÍSLO OD ... DO ...* z nabídky **OPERÁTORY**, přičemž je potřeba zadat meze náhodného čísla. Žáci program pro vykreslení čtverce z vyučovací jednotky Cyklus se známým počtem kroků upraví tak, aby velikost strany čtverce bylo náhodné číslo mezi 100 a 200.

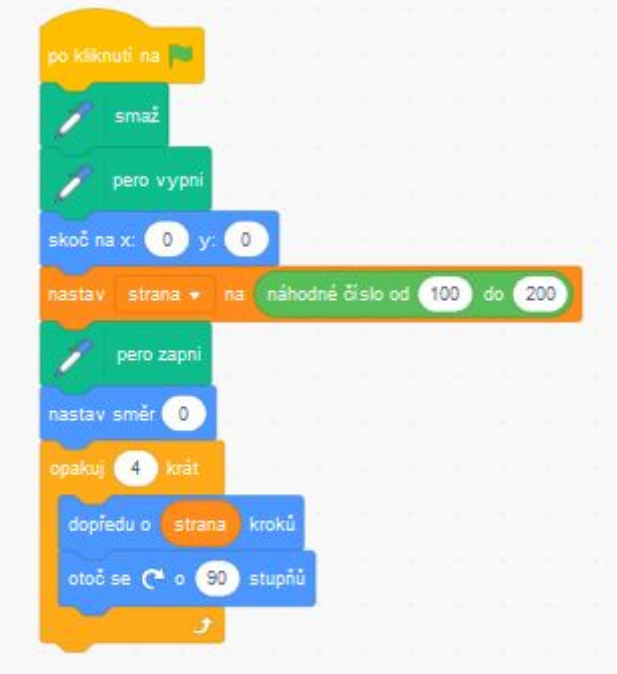
Nechte žáky, aby sami přišli na to, že je potřeba náhodné číslo přiřadit do proměnné, že není možné vložit náhodné číslo přímo do cyklu, kde se čtverec kreslí. Proberte s žáky, co se stane v následujícím programu, kdy se velikost strany čtverce mění v cyklu.



Na základě výše uvedeného příkladu s žáky rozebereme, že je potřeba vygenerované náhodné číslo někam uložit. Místu v paměti, kde se uchovává určitá hodnota, se říká proměnná. Je vhodné žákům říci, že v některých programovacích jazycích je potřeba zadat, jakého typu daná proměnná je. Obvykle se používají proměnné typu celé číslo, reálné číslo, písmeno, logická proměnná (true, false). Těmto typům proměnných se říká jednoduché datové typy, protože v sobě uchovávají jednu hodnotu.

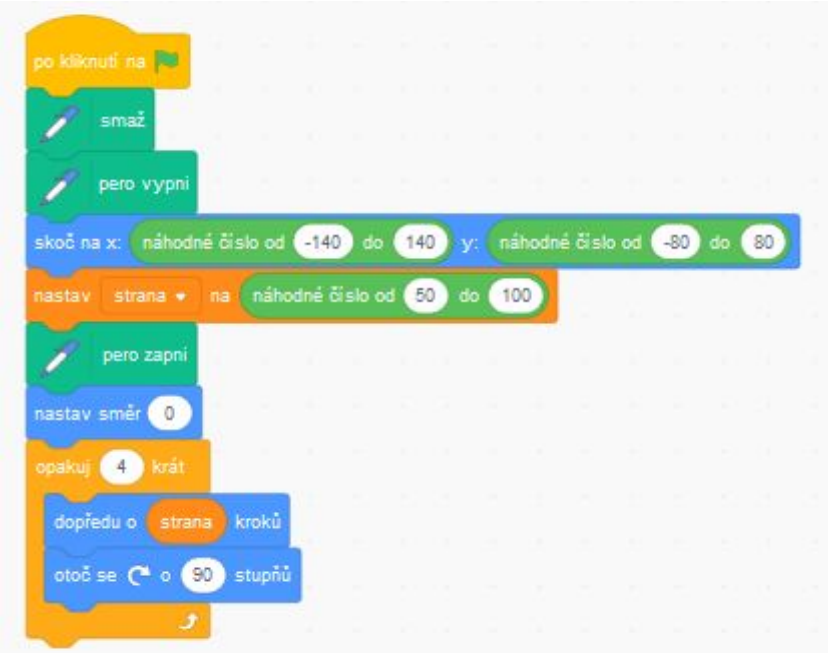
V případě, že chceme používat proměnné v programovacím jazyce Scratch, použijeme v nabídce **PROMĚNNÉ** tlačítko *VYTVOŘ PROMĚNNOU*. Zadává se jméno proměnné a zda chceme proměnnou používat pro všechny postavy (globální) nebo jen pro tuto postavu (lokální proměnná). V této chvíli je dobré s žáky rozebrat, že je vhodné používat pro proměnné názvy, které nějak souvisí s proměnnou (strana, pocet apod.), nejlépe bez diakritiky, popř. zkrácený název (str, poc). O lokálních a globálních proměnných je v této chvíli předčasné se zmiňovat. Přiřazení do proměnné se nachází v nabídce **PROMĚNNÉ**, příkaz *NASTAV*. Scratch umožňuje po celou dobu běhu programu zobrazit hodnotu proměnných. Z počátku je dobré nechat proměnné zobrazené, aby bylo vidět, jak a kdy se proměnné mění. Proměnné stejně jako čísla je možné vložit na ta místa v příkazech, která mají kruhový tvar.

Možné řešení vykreslení čtverce s náhodnou velikostí:



Rychlejší žáci mohou program upravit tak, aby se čtverec kreslil s náhodnou délkou strany na náhodná místa, případně aby se čtverců nakreslilo více. Náhodná čísla pro pozice x a y není potřeba ukládat do proměnných, protože se nikde jinde v programu nepoužívají.

Možné řešení vykreslení čtverce s náhodnou délkou na náhodnou pozici:



Velikost strany čtverce je možné zadávat také z klávesnice, a to v nabídce **VNÍMÁNÍ**, příkaz *OTÁZKA*. To, co zadá uživatel na klávesnici, se uloží do *ODPOVĚĎ*.

Možné řešení:

Rychlejší žáci mohou upravit (zjednodušit) program pro vykreslení hvězd v řadách, kde lze použít systémovou proměnnou y a cyklus v cyklu. Možné řešení:

Možné řešení pyramidy s použitím cyklu v cyklu a třemi proměnnými (pozice x, pozice y a počet hvězd v základně pyramidy) včetně nakreslené pyramidy:

Diskuse k tématu a příkladům

1. Jak upravit program pro zobrazení čtverce o náhodné velikosti tak, aby se zobrazovalo několik čtverců (s různými náhodnými velikostmi) najednou?
2. Je možné v příkladu vykreslení čtverce se zadáním strany z klávesnice použít proměnnou? Jak by program vypadal?
3. Diskuse o použití podprogramu (v jazyce Scratch se používá pojem blok).

Řídící proměnná cyklu v cyklu se známým počtem kroků

Cíl, požadavky

Cíl: použití řídící proměnné cyklu v cyklu se známým počtem kroků.

Vstupní požadavky: použití základních příkazů, příkazu cyklu a proměnných v programovacím jazyce Scratch.

Vymezení pojmu

Jak již bylo uvedeno v kapitole Cyklus se známým počtem kroků, ve většině programovacích jazyků se v cyklu tohoto typu používá řídící proměnná cyklu, jejíž hodnota se mění v závislosti na tom, po kolikáté cyklus probíhá. V programovacím jazyce Scratch řídící proměnná v cyklu tohoto typu není.

Struktura vyučovací jednotky

Na začátku učitel vysvětlí žákům pojem řídící proměnná cyklu. Ve většině programovacích jazyků se řídící proměnná cyklu při každém průchodu mění (zvětšuje nebo zmenšuje, je možné zadat i krok o kolik). V programovacím jazyce Scratch je potřeba proměnnou zavést a v cyklu měnit (obvykle zvětšovat o jedničku). Řídící proměnná cyklu je použita v příkladu pyramida (vyučovací jednotka Použití proměnných, proměnná nazvaná POČET), kdy se proměnná v cyklu zmenšovala o 2.

S žáky se snažíme na začátku vymyslet příklady, ve kterých by se řídící proměnná cyklu mohla využít, např.:

- tisk čísel od jedničky do desítky
- vypsání násobilky zadaného čísla
- zabubnování jednou, dvakrát, třikrát, …, n-krát
- zablikání jednou, dvakrát, třikrát, …
- vykreslení jedné, dvou, tří, … teček

Je možné nechat žáky naprogramovat příklady samostatně a pak s nimi diskutovat o jejich řešeních nebo jim nastínit postup a programy dělat společně.

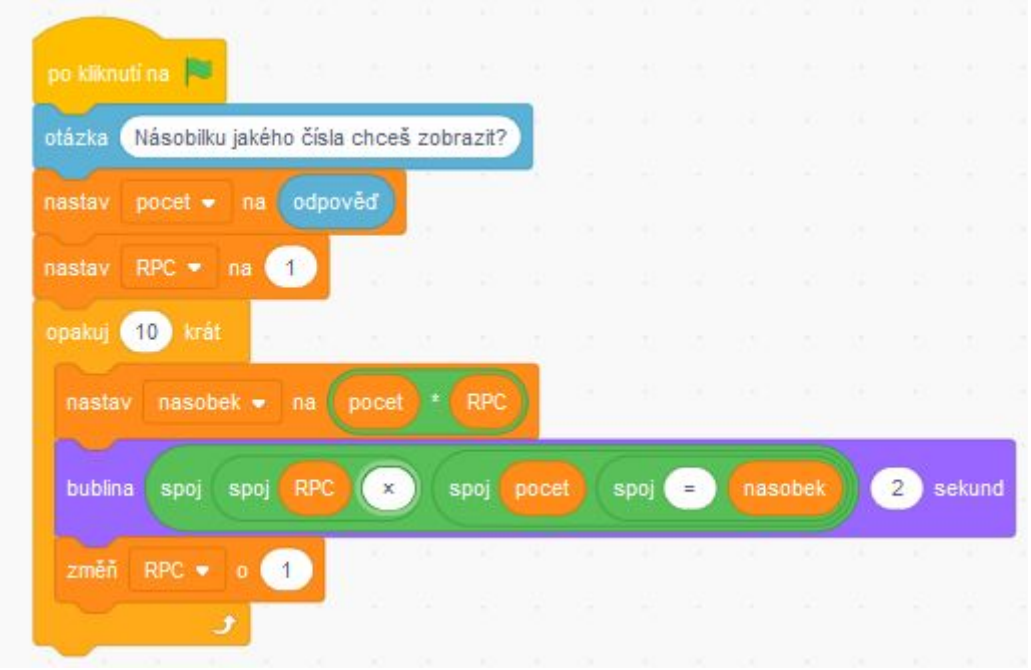
U všech výše uvedených příkladů je potřeba zavést řídící proměnnou cyklu, kterou v cyklu zvětšujeme o jedničku. Podstatné je, že je potřeba řídící proměnné cyklu před samotným cyklem přiřadit jedničku, případně nulu. S žáky může proběhnout diskuse o tom, jaký je rozdíl mezi tím, když se proměnné přiřadí nula, nebo jednička. Další téma do diskusi může být o počtu proměnných, které je v programu potřeba použít.

Tisk čísel od jedničky do desítky – možné řešení:

Vypsání násobilky – postup:

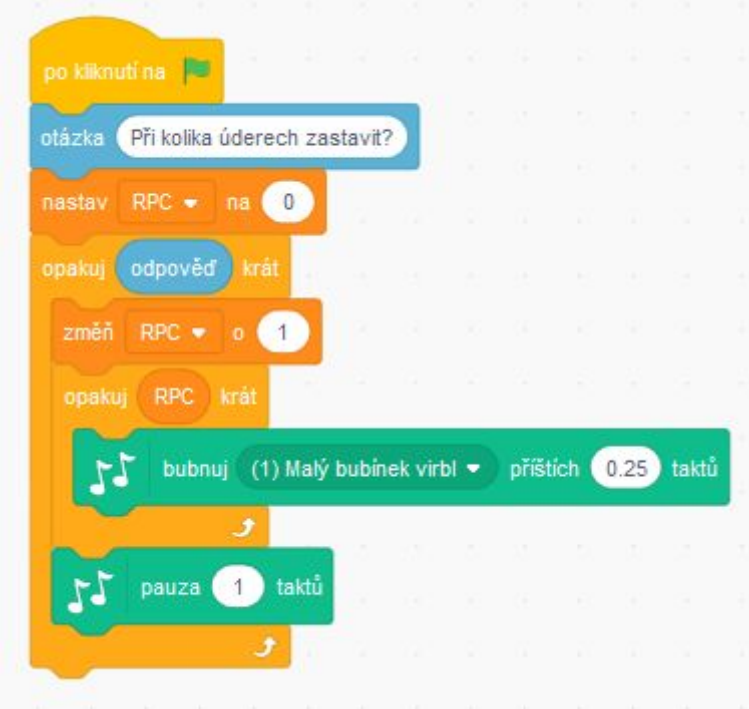
1. Dotaz na číslo, jehož násobilku chceme zobrazit.
2. Nastavení řídící proměnné cyklu na jedničku (případně nulu).
3. Cyklus se známým počtem kroků se provede desetkrát (tiskne se malá násobilka).
    a. Uložení násobku do proměnné.
    b. Tisk násobku, případně tisk celého příkladu s využitím příkazu *SPOJ* z nabídky **OPERÁTORY**.
    c. Zvětšení řídící proměnné cyklu o jedničku.

Možné řešení:



Zabubnování – postup:

1. Dotaz na konečný počet zabubnování.
2. Nastavení řídící proměnné cyklu na jedničku (případně nulu).
3. V tomto příkladu je potřeba použít cyklus v cyklu podobně jako u pyramidy.
    a. Ve vnitřním cyklu se provádí příkaz *BUBNUJ* z nabídky **HUDBA**[3]**.** Cyklus se opakuje tolikrát, jak je velká řídící proměnná vnějšího cyklu.
    b. Ve vnějším cyklu se zvětšuje hodnota řídící proměnné o jedničku a ještě je vhodné sem umístit příkaz pro pauzu (*PAUZA* v nabídce **HUDBA**) nebo příkaz čekání (*ČEKEJ* v nabídce **OVLÁDÁNÍ**).



Diskuse k tématu a příkladům

1. Jakou hodnotu má řídící proměnná cyklu na konci cyklu v případě, že je na začátku přiřazena jednička, a jakou, když je přiřazena nula.
2. Je třeba použít proměnnou POCET v příkladu vypsání násobilky?

[3] Tato nabídka není ve verzi Scratch 3.0 v základní sadě, je potřeba ji stejně jako nabídku PERO přidat (na obrazovce vlevo dole klik na ikonu Přidej rozšíření a vybrat HUDBA).

Podprogram, podprogram s parametry

Cíl, požadavky

Cíl: pochopit používání podprogramů a situací, kdy je vhodné podprogram použít; vytvořit program s použitím podprogramu (bloku).

Vstupní požadavky: používání základních příkazů, znalost cyklu s pevným počtem kroků, znalost pojmu proměnná a jejich používání, vše v programovacím jazyce Scratch.

Vymezení pojmu

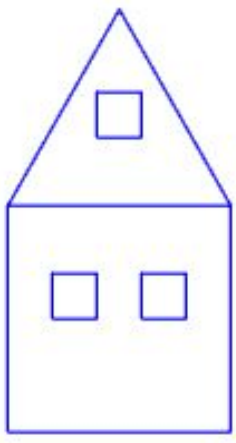
Podprogramy se používají při opakování posloupnosti příkazů v programu a pro zjednodušení čitelnosti složitého a rozsáhlého programu, kdy se program rozděluje na dílčí celky. Pro jednoduché úlohy je podprogramy zbytečné používat. Proměnné, které se používají pouze v podprogramu, se nazývají lokální. Proměnné, které se používají v celém programu, se nazývají globální. Činnost podprogramů je možné ovlivnit parametry.

Podprogramy (procedury, funkce, metody) mohou být bez parametrů nebo s parametry. Je třeba rozlišit formální a skutečné parametry. Formální parametry jsou použité v podprogramu, skutečné parametry se za formální dosadí ve chvíli volání podprogramu. Existují dva druhy parametrů: parametry volané hodnotou a parametry volané odkazem. Pomocí parametrů volaných hodnotou reprezentujeme vstupní hodnoty, předávají formálním parametrům svoji hodnotu. Jako skutečný parametr může být použita konkrétní hodnota nebo proměnná, jejíž hodnota zůstává po ukončení podprogramu stejná. Pomocí parametrů volaných odkazem reprezentujeme výstupní hodnoty. Skutečné parametry však mohou být jenom proměnné, které se změní podle změn provedených v podprogramu. Stejně jako parametry volané hodnotou předávají formálním parametrům svoji hodnotu.

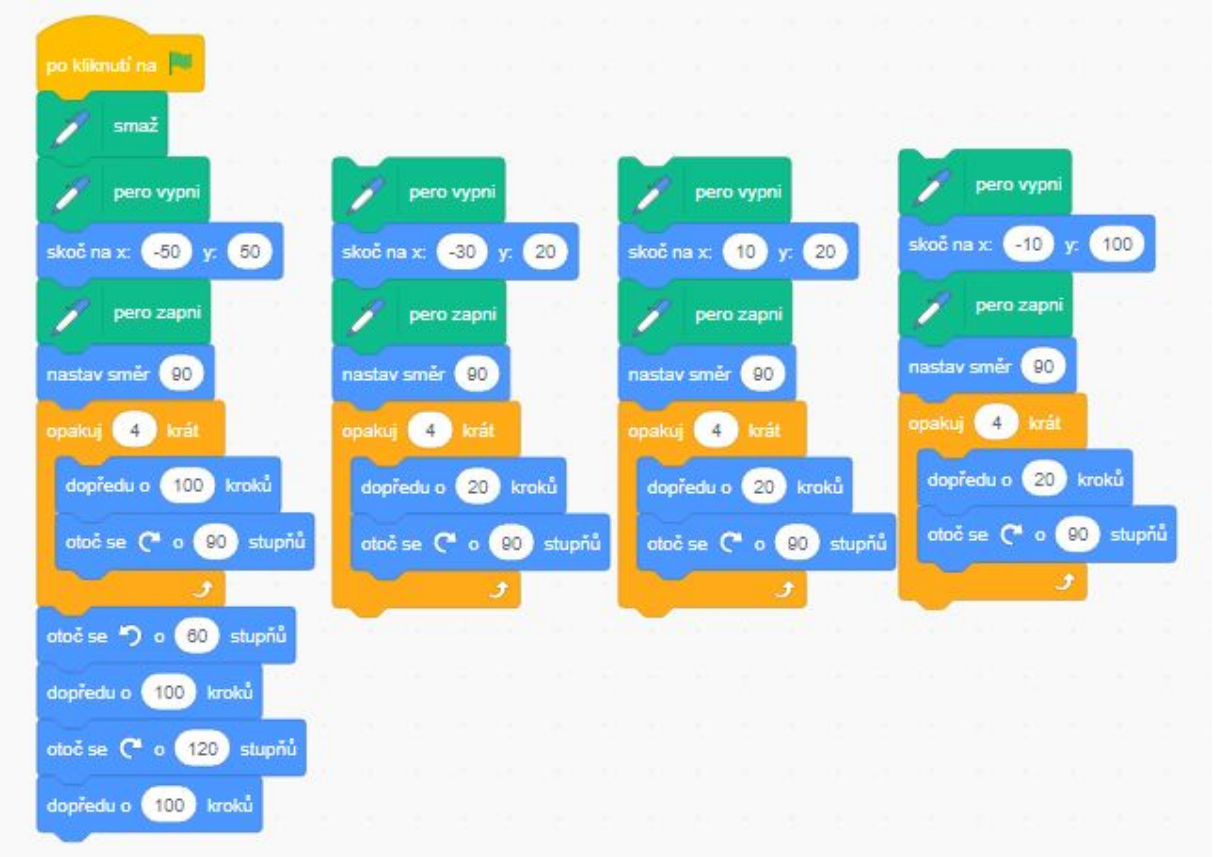
V programovacím jazyce Scratch je možné vytvářet podprogramy (bloky) bez parametrů a s parametry volanými hodnotou.

Struktura vyučovací jednotky

Na začátku žáci vytvoří program, který vykreslí domek (čtverec se střechou a dvěma až třemi čtvercovými okénky).



Možné řešení je na obrázku:

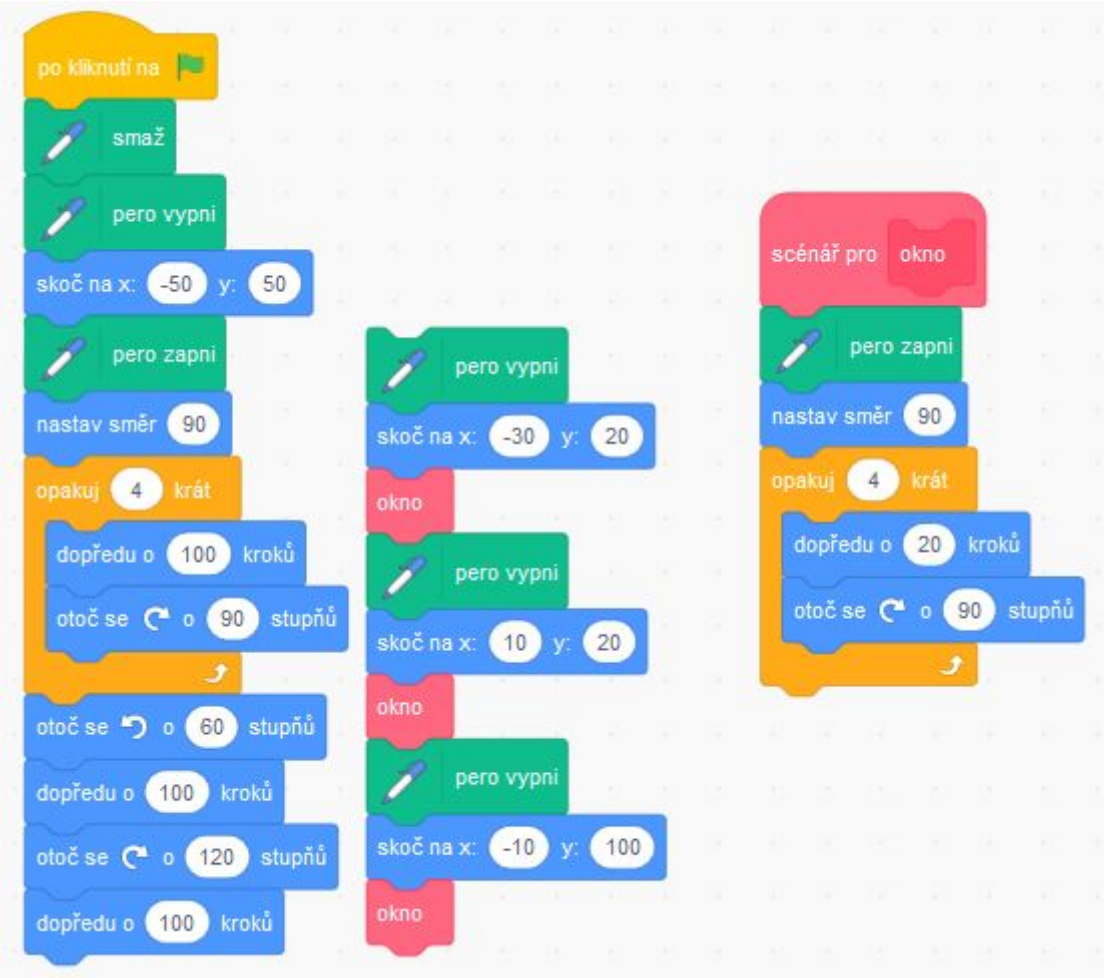


Otázka: Vidíte v programu část, která se tam několikrát opakuje?

Odpověď: Jde o část programu, kde se dává pero dolů a kreslí se čtverec. V případě, že používáme v programu na několika místech úplně stejnou posloupnost příkazů, můžeme ji nahradit podprogramem (blokem), který v programu voláme na místech, kde se daná posloupnost vyskytuje. Podprogramy jsou v jazyce Scratch nazvány bloky a vytvářejí se v nabídce **MOJE BLOKY**. Při vytváření bloků je potřeba zadat jméno bloku. Ve volbách se přidávají parametry, ke kterým se dostaneme později. Při vytvoření bloku se mimo hlavní program vytvoří blok, do kterého je možné přetáhnout část programu, která se opakuje:

Ze samotného programu se tyto části vymažou a nahradí se voláním bloku. Tak se stane program přehlednější.

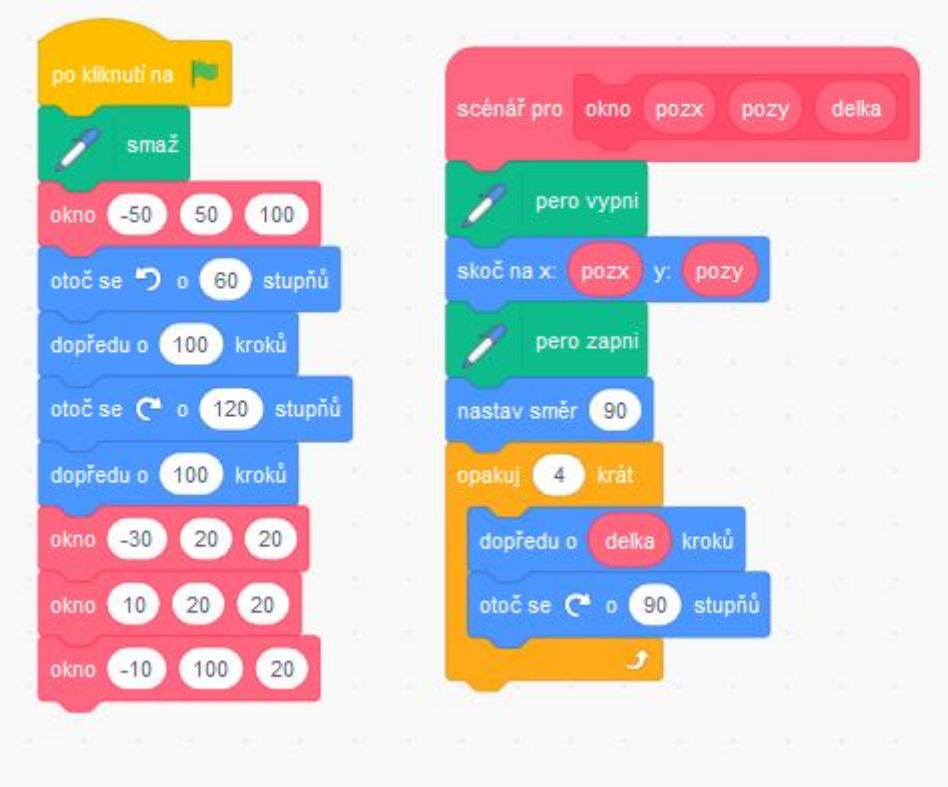
Možné řešení:



Otázka: Je možné program ještě více zjednodušit a zpřehlednit? Najdete část programu, která je podobná té části použité v podprogramu?

Odpověď: Jde o část programu, kde se vykresluje samotný domek (včetně přesunu). V těchto částech se ale používají u přesunu i u délky strany čtverce různé hodnoty. V tomto případě je vhodné použít podprogram s parametry. Při vytváření bloku v programovacím jazyce Scratch se objeví možnost *Přidej parametr (číslo nebo text)*. V již vytvořeném bloku se přidají parametry po kliknutí na blok pravým tlačítkem a výběru možnosti *Upravit*. V našem případě přidáme tři číselné parametry: pozx, pozy a delka. Při volání podprogramu se tyto parametry (formální) nahradí konkrétními hodnotami (skutečnými parametry), které se zadávají v hlavním programu při volání samotného podprogramu.

Možné řešení:



Diskuse k tématu a příkladům

1. Kdy je vhodné použít podprogram?
2. Jaký je rozdíl mezi proměnnou a parametrem?
3. Je možné použít při volání podprogramu s parametrem jako skutečný parametr proměnnou místo konkrétní hodnoty (čísla)?

Podmíněný příkaz neúplný, relační a logické operátory

Cíl, požadavky

Cíl: použití neúplného podmíněného příkazu, porozumění použití relačních a logických operátorů.

Vstupní požadavky: použití základních příkazů a příkazu cyklu, popř. podprogramů v programovacím jazyce Scratch.

Vymezení pojmu

Podmíněný příkaz se používá v případě, kdy je potřeba program rozvětvit. Základním prvkem podmíněného příkazu je podmínka. Výsledkem podmínky je pravda nebo nepravda; to znamená, že podmínka musí obsahovat relační nebo logickou operaci.

Mezi relační operátory patří: =, >, <, ≠, ≦, ≧. V různých programovacích jazycích se pro poslední tři uvedené operátory používají různé symboly, resp. dva symboly za sebou (např. <=). Mezi logické operátory patří logický součin, logický součet a negace. U logického součinu je logický výraz pravdivý v případě, že jsou oba (všechny) operandy pravdivé. U logického součtu je logický výraz pravdivý, když alespoň jeden z operandů je pravdivý. Negace je pravdivá, když operand je nepravdivý.

Neúplný podmíněný příkaz se používá v případě, kdy je třeba vykonat nějaký příkaz (příkazy) jen v tom případě, když je splněna určitá podmínka. V případě, že podmínka splněna není, program pokračuje dalším příkazem.

V programovacím jazyce Scratch jsou k dispozici jen některé z relačních operátorů (<, >, =), logické operátory jsou nazvány *a* (logický součin), *nebo* (logický součet), *ne* (negace).

Struktura vyučovací jednotky

Na začátku učitel s žáky diskutuje o tom, co si představují pod pojmem podmíněný příkaz. Podmíněný příkaz obsahuje podmínku (dotaz), na kterou je možno odpovědět ano nebo ne (např. Je číslo kladné? Je stisknuta klávesa šipka vpravo? Rovnají se dvě čísla?). U neúplného podmíněného příkazu se provede příkaz (příkazy) v případě, že je podmínka splněna; v případě, že podmínka splněna není, neprovede se nic. Podmínka nemůže být typu: U maturity jsi prospěl, nebo ne? Dále žáci vymýšlejí reálné situace v životě, ve kterých se podmíněný příkaz dá použít. Příkladem podmínky může být např.:

- Mám dost peněz na zmrzlinu? V případě, že je podmínka splněna, koupím si zmrzlinu.
- Mám hlad? V případě, že je podmínka splněna, najím se.
- Uplaval jsi 100 metrů? V případě, že ano, získal jsi bobříka plavce.

V další části učitel s žáky pracuje v programovacím jazyce Scratch. Neúplný podmíněný příkaz použijeme pro pohyb postavy doprava a doleva včetně otáčení postavy. Je vhodné použít jeden kostým, jehož postava je natočena vpravo (může se použít výchozí Cat-a). Výhodou programovacího jazyka Scratch je, že místa, kam se vkládají podmínky, proměnné, číselné nebo textové hodnoty, mají určitý tvar. Pro podmínky to jsou šestiúhelníky. To znamená, že do podmínky mohu vložit jen některé prvky z nabídky **OPERÁTORY** (relační a logické výrazy) a některé prvky z nabídky **VNÍMÁNÍ**.

Postup:

- Vytvoříme nekonečný cyklus: nabídka **OVLÁDÁNÍ**, příkaz *OPAKUJ DOKOLA*.
- Do cyklu vložíme dva podmíněné příkazy (příkaz *KDYŽ ... TAK* z nabídky **OVLÁDÁNÍ**).
- Do prázdného šestiúhelníku se vloží podmínka *KLÁVESA ŠIPKA VPRAVO/VLEVO STISKNUTA?* z nabídky **VNÍMÁNÍ**.

Program může vypadat takto:

V tomto případě je postava natočena pořád jedním směrem. Bylo by dobré, kdyby se po stisku klávesy šipka vlevo postava otočila vlevo a po stisku klávesy šipka vpravo zase doprava. V tomto případě je potřeba nastavil při pohybu i směr natočení.

Program pak může vypadat takto:

Rychlejší žáci mohou přidat dva podmíněné příkazy pro pohyb nahoru a dolů. U postavy je v tomto případě nutné změnit **styl otáčení** ve vlastnostech postavy (klik na směr pod oknem s postavou a u stylu otáčení nastavit šipku dokola).

Do programu je možné přidat ještě další podmíněné příkazy, které způsobí, že se po dosažení okraje postava přesune do počátku (popř. zleva doprava, zprava doleva, shora dolů, zdola nahoru).

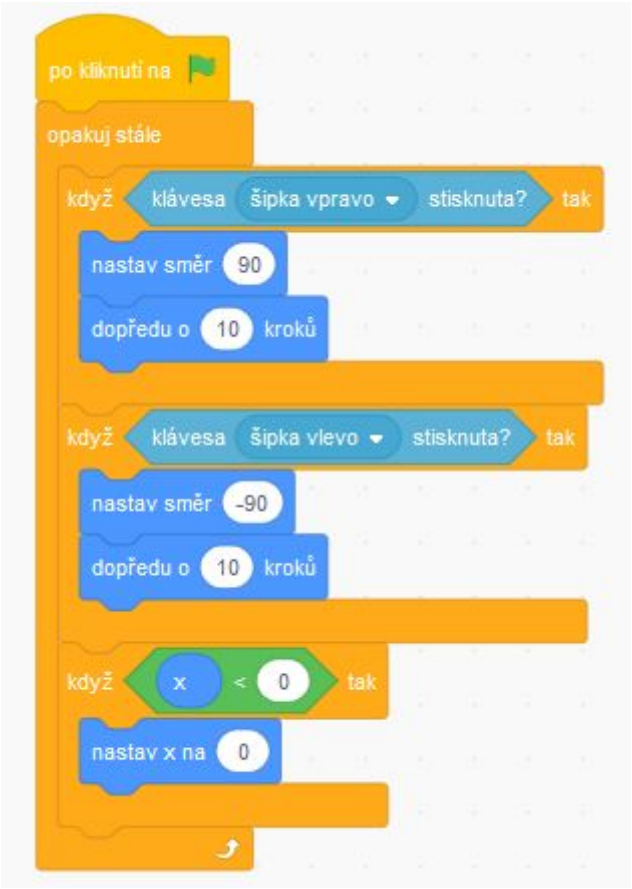
Před tím, než žáci začnou řešit výše uvedenou úlohu, je vhodné jim ukázat a vysvětlit použití relačních (>, =, <) a logických (*a*, *nebo* a *ne*) operátorů. Tyto operátory jsou umístěny v nabídce **OPERÁTORY**. Je důležité si zopakovat pojmy, co je to operátor a operand (např. v podmínce: **pozicex > 0**; operandy jsou: **pozicex**, **0**; operátor je: **>**).

Relační operátory by žáci měli znát z matematiky. Je důležité žáky upozornit na to, že všechny tyto operátory se používají při porovnávání (nejčastěji čísel), symbol „=“ slouží tedy pro porovnání, zda se nějaká hodnota rovná jiné hodnotě. Výsledkem relačního výrazu je pravda nebo nepravda. Relační výrazy se nejčastěji používají v podmínkách.

Logické výrazy se také nejčastěji používají v podmínkách. Logický operátor *a* se nazývá logický součin, nabývá pravdivé hodnoty v případě, že jsou oba (všechny) operandy pravdivé. Logický operátor *nebo* se nazývá logický součet a nabývá pravdivé hodnoty v případě, že aspoň jeden z operandů je pravdivý. Logický operátor *ne* je negace a má jen jeden operand. V případě, že operand je pravda, negace je nepravda a naopak.

S žáky je dobré vyzkoušet relační a logické operátory přímo na konkrétních příkladech.

- Postavou je možné pohybovat jen v pravé části kreslicí plochy:



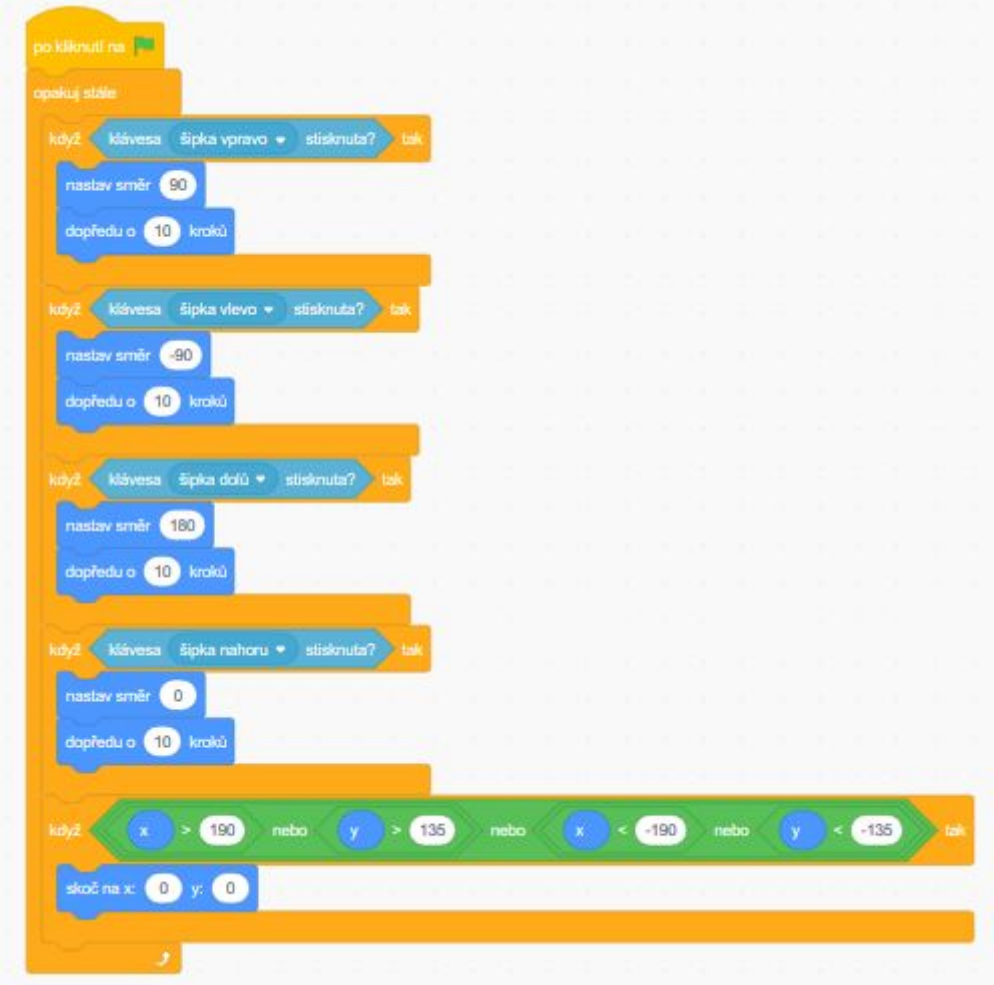
- Postavou je možné pohybovat jen v horní části kreslicí plochy:

- Postavou je možné pohybovat jen v pravé horní části kreslicí plochy.
    - Výše uvedené dva podmíněné příkazy se vloží do programu za sebe.
    - Je možné použít jednu podmínku? Jak by potom vypadala? Co bude v těle podmínky?

- Co se stane po provedení následující podmínky?

Nechte žáky přidat do programu další podmíněné příkazy, které způsobí, že se po dosažení okraje postava přesune do počátku (popř. zleva doprava, zprava doleva, shora dolů, zdola nahoru).

Program, u kterého se po dosažení jakéhokoli okraje postava přesune do počátku (do středu plátna), může vypadat takto:



Diskuse k tématu a příkladům

1. Kolik podmíněných příkazů žáci použili v příkladu, kdy se postava z okraje přesune do počátku (na souřadnice 0, 0)? Pokud někdo řešil přesun zleva doprava a naopak a shora dolů a naopak, kolik podmíněných příkazů použil?
2. Proč jsou v podmínce pro dosažení okrajů uvedeny relační operátory je větší (>) a je menší (<) a není tam uveden operátor rovná se (=)?
3. Je možné v tomto příkladu použít podprogram?
4. Jak je možné udělat zrychlení (zpomalení) pohybu?

Podmíněný příkaz úplný

Cíl, požadavky

Cíl: použití úplného podmíněného příkazu.

Vstupní požadavky: použití základních příkazů, neúplného podmíněného příkazu a příkazu cyklu, popř. podprogramů v programovacím jazyce Scratch.

Vymezení pojmu

Úplný podmíněný příkaz se používá v případě, když je třeba vykonat nějaký příkaz (posloupnost příkazů) v případě, když je splněna určitá podmínka, a jiný příkaz (posloupnost příkazů), když tato podmínka splněná není.

Struktura vyučovací jednotky

Na začátku učitel s žáky diskutuje o tom, co si představují pod pojmem úplný podmíněný příkaz. Je vhodné navázat na diskusi a příklady z vyučovací jednotky Podmíněný příkaz neúplný. Příklady ze života:

- Uplaval jsi 100 m? V případě, že ano, získal jsi bobříka plavce, v případě, že ne, navštiv kurz plavání.
- Je více než 25 °C? V případě, že ano, jdu se koupat, v případě, že ne, jdu domů.

Je důležité, aby žáci pochopili, jak úplný podmíněný příkaz funguje. Na jednoduchých příkladech ukážeme rozdíl mezi úplným a neúplným podmíněným příkazem. Je dobré žákům ukázat, jak udělat rozdělení do tří a více možností.

Jaký je rozdíl mezi následujícími dvěma částmi programu?

V těchto dvou příkladech je použit nový příkaz *NASTAV SMĚR*, který nastavuje směr postavy a v podmínce je použita další ze systémových proměnných *SMĚR*, která vrací aktuální směr

natočení postavy. Pro tyto příklady je vhodné použít jako postavu např. šipku (Arrow). S žáky je dobré rozebrat, co která část programu dělá.

V prvním příkladu, kde jsou dva za sebou jdoucí neúplné podmíněné příkazy se stejnou podmínkou, se při průchodu první podmínkou (je-li na začátku postava natočena vpravo, tedy směr = 90) postava otočí na druhou stranu (směr = –90), druhá podmínka tedy splněna není a nic se nestane. V případě, že je na začátku postava natočena vlevo (směr = –90), první podmínka splněna není, takže se nic nestane, druhá podmínka také splněna není, takže se také nic nestane.

V druhém příkladu, kdy je použit úplný podmíněný příkaz, se postava (je-li na začátku natočena vpravo, směr = 90) po provedení podmíněného příkazu otočí (směr = –90) a po dalším provedení podmíněného příkazu se otočí zpět (směr = 90). Takže ve druhém případě dochází při opakování podmíněného příkazu k otáčení z jedné strany na druhou.

Co se stane po provedení této části programu?

V případě, že žáci nepřijdou na to, co část programu dělá, je možné vložit mezi podmíněné příkazy příkaz *ČEKEJ* z nabídky **OVLÁDÁNÍ**. V případě, že žáci vloží mezi neúplné podmíněné příkazy čekání, dojde k tomu, že se postava (je-li natočena vpravo, tedy směr = 90) otočí, otočená počká a při provedení druhého podmíněného příkazu se otočí zpět. S žáky je možné projít i případ, kdy na začátku bude postava otočena vlevo (směr = –90).

Sestavte jednoduchý program, ve kterém se po kliknutí na postavu postava otočí opačným směrem. Pro tuto úlohu je vhodné použít postavu, u které je vidět, když mění směr.

Možné řešení:

Žáci mohou vymýšlet aplikace (hry), ve kterých by se dal tento jednoduchý příkaz použít.

Diskuse k tématu a příkladům

1. Diskutujte s žáky o možných příkladech rozdělení do tří nebo více větví.
2. Jakým způsobem by se řešilo rozdělení programu na tři větve?

Rozpracování dvou komplexních aplikací

Cíl, požadavky

Cíl: rozpracování dvou aplikací: zkoušení malé násobilky, hra hádání čísla z určitého intervalu.

Vstupní požadavky: použití základních příkazů, neúplného a úplného podmíněného příkazu a příkazu cyklu, popř. podprogramů v programovacím jazyce Scratch.

Struktura vyučovací jednotky

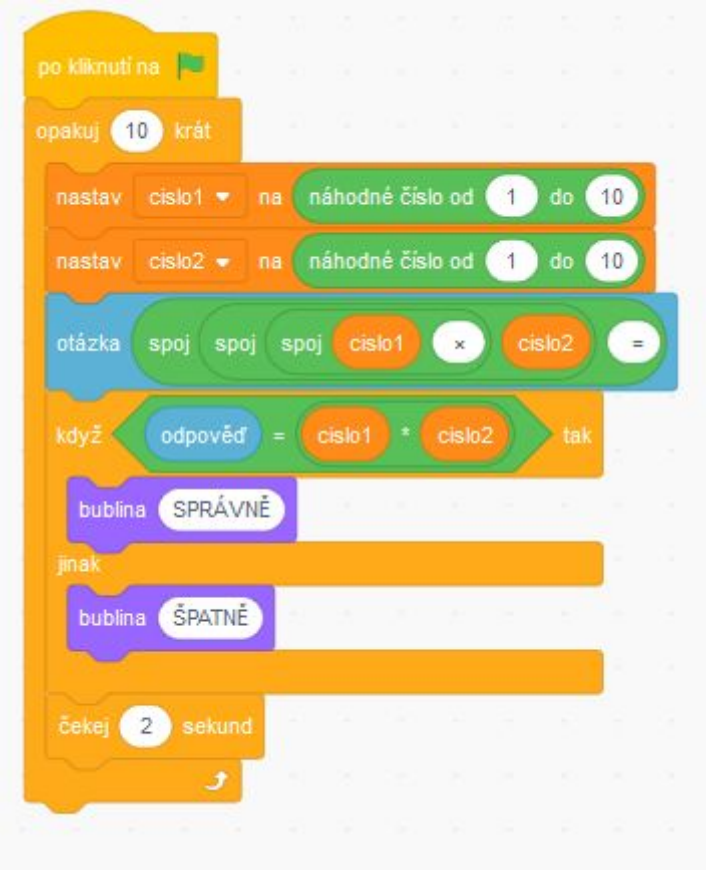
S žáky rozpracujeme dvě aplikace, na kterých budou pracovat více vyučovacích jednotek. Jedna z aplikací bude zkoušení malé násobilky, druhá hádání čísla. U obou aplikací se budou postupně nabalovat další problémy a aplikace se tak budou vylepšovat.

Postup u příkladu zkoušení násobilky:

1. Generování dvou náhodných čísel v rozmezí od 1 do 10 a jejich přiřazení do dvou proměnných:
    - V nabídce **PROMĚNNÉ** vytvoříme dvě proměnné (cislo1, cislo2).
    - Do programu přidáme přiřazení náhodných čísel do proměnných (nastavení čísel je v nabídce **PROMĚNNÉ**, generování náhodného čísla je v nabídce **OPERÁTORY**).
2. Zobrazení příkladu:
    - Pro zobrazení textu a čekání na zadání se použije v nabídce **VNÍMÁNÍ** příkaz *OTÁZKA*. Pro zobrazení celého příkladu je potřeba spojit několik řetězců (čísel, symbolů), k tomu slouží příkaz *SPOJ*, který je umístěn v nabídce **OPERÁTORY**. Pro zobrazení příkladu je potřeba použít několik vnořených příkazů.
3. Zadání výsledku příkladu uživatelem.
    - Číslo zadané uživatelem se uloží do systémové proměnné *ODPOVĚĎ*.
4. Vložení úplného podmíněného příkazu:
    - V nabídce **OVLÁDÁNÍ** vybrat příkaz *KDYŽ TAK JINAK*.
    - Do podmínky vložit relační operátor = (nabídka **OPERÁTORY**) a vložit jeden operand *ODPOVĚĎ* (nabídka **VNÍMÁNÍ**) a druhý operand součin vygenerovaných náhodných čísel.
    - Když je podmínka splněna (čísla jsou stejná), pak se zobrazí text správně, jinak se zobrazí text špatně (nabídka **VZHLED**, příkaz *BUBLINA*).
5. V případě zkoušení 10 příkladů celý program vložíme do cyklu (nabídka **OVLÁDÁNÍ**, příkaz *OPAKUJ*).

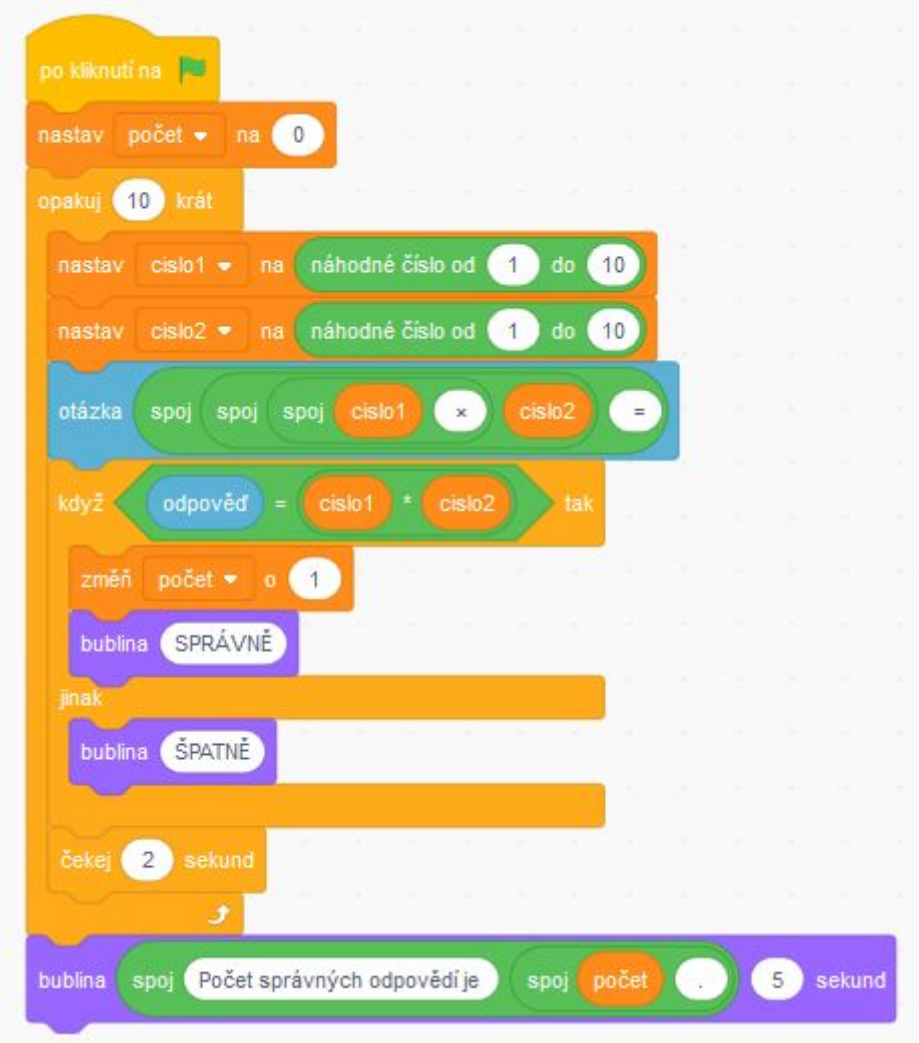
6. Po vyzkoušení programu je dobré přidat po zobrazení textu příkaz pro čekání (nabídka **OVLÁDÁNÍ**), upravit meze generování náhodného čísla (aby tam nebylo násobení 1 a 10), skrýt proměnné (nabídka **PROMĚNNÉ**) apod.

Program může vypadat takto:



Rychlejší žáci mohou do programu přidat počítání správných pokusů. V tomto případě je potřeba zavést další proměnnou, kterou je nutné na začátku vynulovat a potom k ní přičítat jedničku v případě, že je odpověď správná. S žáky je vhodné rozebrat, proč je potřeba proměnnou, která počítá správné odpovědi, na začátku vynulovat (nastavit ji na nulu).

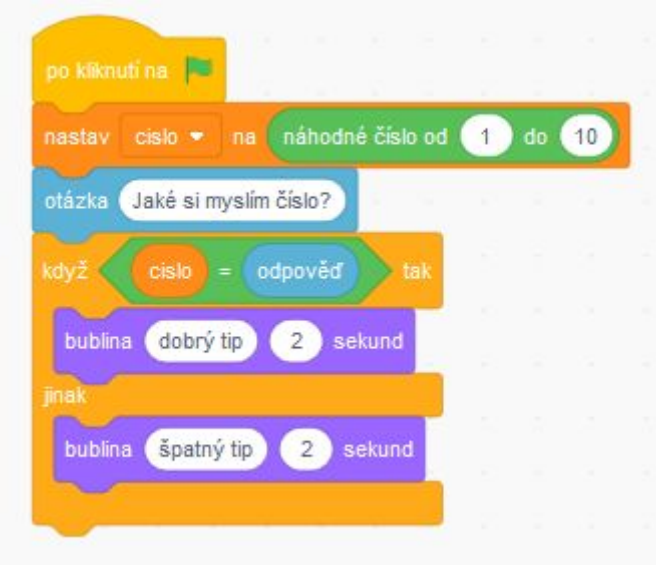
Možné řešení:



Postup u příkladu hádání čísla:

1. Vygenerování náhodného čísla v určitém rozmezí (rozmezí je možné i zadat), náhodné číslo uložíme do proměnné.
2. Zadání tipu uživatelem.
3. Vložení úplného podmíněného příkazu, v podmínce je porovnání zadaného a vygenerovaného čísla; když je podmínka splněna (čísla jsou stejná), zobrazí se text dobrý tip; v případě, že ne, zobrazí se špatný tip.

Možné řešení:



Diskuse k tématu a příkladům

1. Jak příklad Hádání čísla upravit tak, aby uživatel mohl hádat tak dlouho, až vygenerované číslo uhodne. Je možné použít příkaz cyklu *OPAKUJ*?
2. Nechte žáky napsat jednotlivé kroky za sebou a diskutujte o možných řešeních. Jaké všechny cykly Scratch nabízí?
3. Diskusí by se mělo dojít k použití jiného typu cyklu, a to cyklu s podmínkou na začátku, v programovacím jazyku Scratch *OPAKUJ DOKUD NENASTANE*.

Cyklus s podmínkou

Cíl, požadavky

Cíl: použití cyklu s podmínkou a jeho aplikace v příkladech Hádání čísla a Zkoušení malé násobilky.

Vstupní požadavky: použití základních příkazů, podmíněných příkazů a příkazu cyklu, popř. podprogramů v programovacím jazyce Scratch.

Vymezení pojmu

Kromě cyklu s pevným počtem opakování se v programování používá cyklus s podmínkou, a to s podmínkou na začátku nebo na konci.

Cyklus s podmínkou na začátku (nazývaný též cyklus WHILE) funguje tak, že se nejdřív vyhodnotí podmínka. V případě, že je podmínka splněna, dojde k provedení cyklu. V případě, že podmínka splněná není, příkazy, které jsou v cyklu, se neprovedou. To znamená, že je možné, že se příkazy v cyklu vůbec neprovedou (v případě nesplnění podmínky).

U cyklu s podmínkou na konci (nazývaný cyklus DO-WHILE nebo REPEAT-UNTIL) se příkazy umístěné v cyklu provedou minimálně jednou. Podmínka se vyhodnocuje na konci cyklu. V případě, že podmínka je splněna, dochází k opakování příkazů v cyklu; v případě, že splněna není, dojde k ukončení cyklu.

V programovacím jazyce Sratch existuje cyklus s podmínkou nazvaný *OPAKUJ DOKUD NENASTANE*.

Struktura vyučovací jednotky

Na začátku žákům vysvětlíme, jak si mají představit cyklus s podmínkou v reálném světě na konkrétním příkladu. Žáci se snaží vymyslet také nějaké příklady z reálného světa.

Příklady:

- opakování nějakého cviku až do té doby, než se povede,
- vybírání kuliček z pytlíku s různobarevnými kuličkami až do té doby, než se vytáhne určitá barva (např. žlutá).

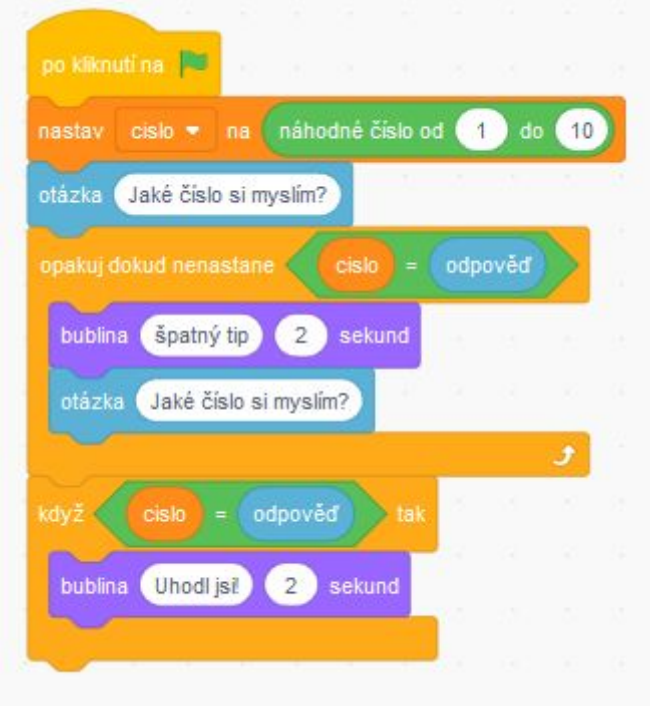
Je dobré s žáky probrat příklady jednotlivých cyklů v reálném životě.

- Příklady na cyklus FOR: Jdu na tři zmrzliny. Cvik opakuji desetkrát.
- Příklady na cyklus DO-WHILE: Opakuji cvik, dokud ho neudělám správně. Plavu 100 m, dokud čas nebude pod dvě minuty.
- Příklad na cyklus WHILE: Dokud mám peníze, kupuji si zmrzlinu.

Poté s žáky navážeme na diskusi týkající se úpravy programu pro hádání čísla. Program upravíme tak, aby uživatel hádal tak dlouho, až číslo uhodne. Na místo podmíněného příkazu se vloží cyklus s podmínkou (nabídka **OVLÁDÁNÍ**, příkaz *OPAKUJ DOKUD NENASTANE*). Do těla cyklu se vloží zobrazení textu špatný tip a zadání nového čísla. Za cyklus je vhodné vložit text uhodl jsi (nebo něco podobného).

Program může vypadat takto:

Častá chyba, kterou žáci často dělají, je umístění podmíněného příkazu za cyklus:



Rychlejší žáci mohou program zdokonalit:

- Při zadání čísla z klávesnice se kromě špatný tip zobrazí, zda je číslo menší, nebo větší než vygenerované náhodné číslo.

- Počítá se počet pokusů.
- Na začátku je možné zadat hranice generování náhodného čísla.
- Hádání čísla je možné opakovat až do stisknutí konkrétní klávesy: v tomto případě se použije cyklus v cyklu a do podmínky vnějšího cyklu se může vložit např. porovnání proměnné konec (do které se vloží uživatelova odpověď) s řetězcem „ano“ (proměnná může obsahovat kromě čísel také např. písmena nebo řetězce, které je možné také porovnávat).

Program může vypadat takto:

Podobně mohou rychlejší žáci upravit i program pro zkoušení z násobilky.

Diskuse k tématu a příkladům

1. Kolik proměnných žáci potřebovali v příkladu hádání čísla? Je možné v programu použít méně proměnných?
2. Je možné a vhodné v příkladech hádání čísla nebo zkoušení násobilky použít podprogram (blok)?

Seznamy

Cíl, požadavky

Cíl: použití seznamů (proměnných typu pole).

Vstupní požadavky: použití základních příkazů, podmíněných příkazů, příkazů cyklů a podprogramů v programovacím jazyce Scratch.

Vymezení pojmu

V programovacích jazycích se často používají strukturované datové typy. Mezi nejznámější strukturované datové typy patří pole a řetězec. Do proměnné strukturovaného datového typu je možné uložit více hodnot než jednu. Tyto hodnoty jsou u typu pole a řetězec stejného datového typu a k jednotlivým hodnotám se přistupuje pomocí indexů (většinou číselných). Jednotlivé prvky (hodnoty) proměnné typu pole mohou obsahovat proměnnou jednoduchého i strukturovaného datového typu. Řetězec je speciální typ pole, jehož prvky jsou proměnné typu char (znak). Pro proměnné typu řetězec jsou obvykle definovány speciální funkce (např. délka, výběr podřetězce z řetězce).

V programovacím jazyce Scratch je možné použít jen strukturovaný datový typ pole (SEZNAM) a některé funkce pro práci s řetězci (spojení řetězců, výběr písmena na určité pozici v řetězci, délka řetězce).

Struktura vyučovací jednotky

Na začátku diskutujeme s žáky o tom, jak naprogramovat aplikaci pro zkoušení slovíček.

- Jakým způsobem zadávat slovíčka, ze kterých se bude zkoušet?
- Jakým způsobem vybírat jednotlivá slovíčka ze seznamu?
- Jak zjistit, zda odpovězené slovíčko odpovídá českému/cizojazyčnému protějšku?

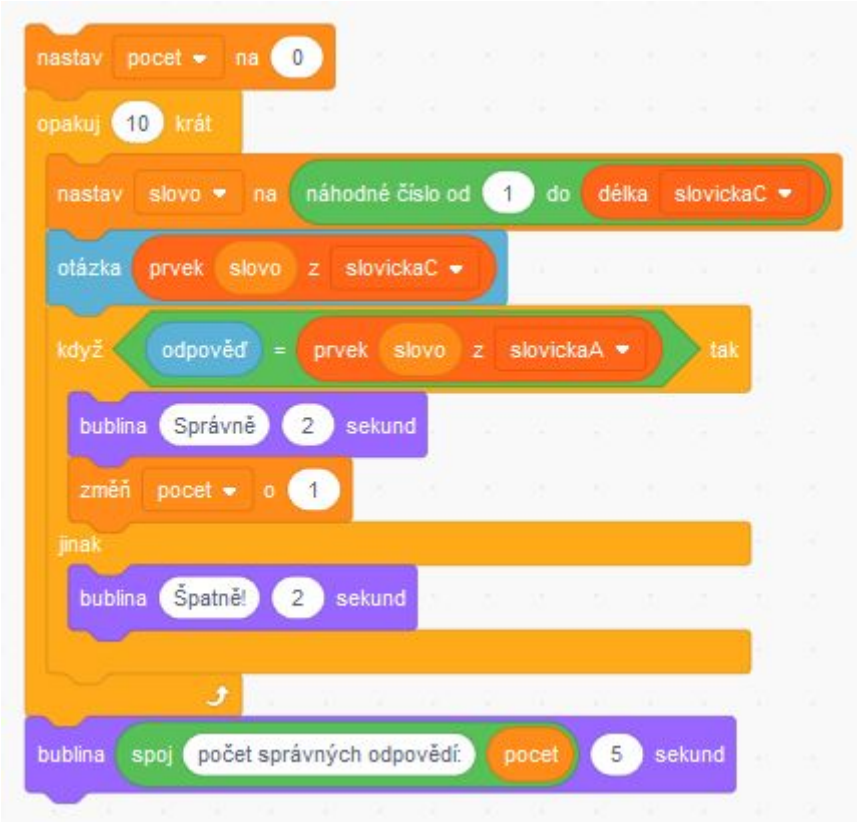
Z diskuse by mělo vyplynout, že nejlepší pro uložení slovíček je proměnná typu pole (v programovacím jazyce Scratch nazvaná SEZNAM). Pro česká slovíčka použijeme jeden seznam, pro cizojazyčná druhý seznam. České slovíčko a k tomu cizojazyčný protějšek jsou umístěny v polích na stejné pozici. V tomto případě také ukážeme žákům, že je možné do proměnných kromě čísel ukládat také slova (řetězec znaků).

V některých programovacích jazycích (např. Java, Python) je přímo strukturovaný datový typ slovník (dictionary), kam se ukládají dvojice prvků, kde jeden je označen jako klíč a druhý jako hodnota.

Postup:

1. Cyklus pro zadávání slovíček. V případě, že po každém spuštění chceme zadávat nová slovíčka, je potřeba seznamy (pole) vyprázdnit.

2. Cyklus pro zkoušení slovíček deseti slovíček:
    - výběr náhodného slovíčka v českém/cizím jazyce a jeho zobrazení,
    - zadání odpovídajícího slovíčka v cizím/českém jazyce,
    - porovnání odpovědi se slovíčkem umístěným na stejné pozici v poli s cizími/českými slovíčky,
    - započítání správných odpovědí.



Celý program je možné vylepšovat podle úrovně jednotlivých žáků:

- zadává se počet zkoušených slovíček,
- celé zkoušení se ukončí podle odpovědi ano/ne: použití cyklu s určitým počtem kroků v cyklu s podmínkou na začátku.

Při tvorbě programu je dobré mít proměnné zobrazené a postupně je skrývat.

Diskuse k tématu a příkladům

1. Vymyslete další příklady, kde je vhodné použít proměnné typu seznam.
2. Vymaže se po ukončení programu v programovacím jazyce Sratch obsah proměnných?
3. Jak je to s obsahem proměnných v jiných programovacích jazycích?

Rekurze

Cíl, požadavky

Cíl: použití rekurzivních podprogramů.

Vstupní požadavky: použití základních příkazů, podmíněných příkazů, příkazů cyklů a podprogramů v programovacím jazyce Scratch.

Vymezení pojmu

Rekurzivní podprogramy (procedury nebo funkce) využívají volání sebe sama. Při používání rekurze je potřeba dodržet dvě pravidla: v každém kroku musí dojít ke zjednodušení problému a v podprogramu musí být podmínka pro ukončení. Pokud tyto podmínky dodrženy nejsou, dojde k zacyklení. Typickým příkladem pro použití rekurze je výpočet faktoriálu, jehož rekurzivní definice je:

- $f(n) = n \cdot f(n-1)$ pro $n > 0$
- $f(0) = 1$

Struktura vyučovací jednotky

Na začátku by měla probíhat s žáky řízená diskuse o pojmu rekurze. Žáci mohou pojem rekurze vyhledat na webových stránkách. Je vhodné žákům ukázat rekurzivní (využívající volání procedury v proceduře samotné) a iterativní (využívající cyklus) definici např. n-té mocniny čísla x.

iterativní definice: $x^n = x \cdot x \cdot x \cdot x \cdot \ldots \cdot x$

rekurzivní definice: $x^n = 1$ pro $n = 0$

$$x^n = x \cdot x^{n-1} \text{ pro } n > 0$$

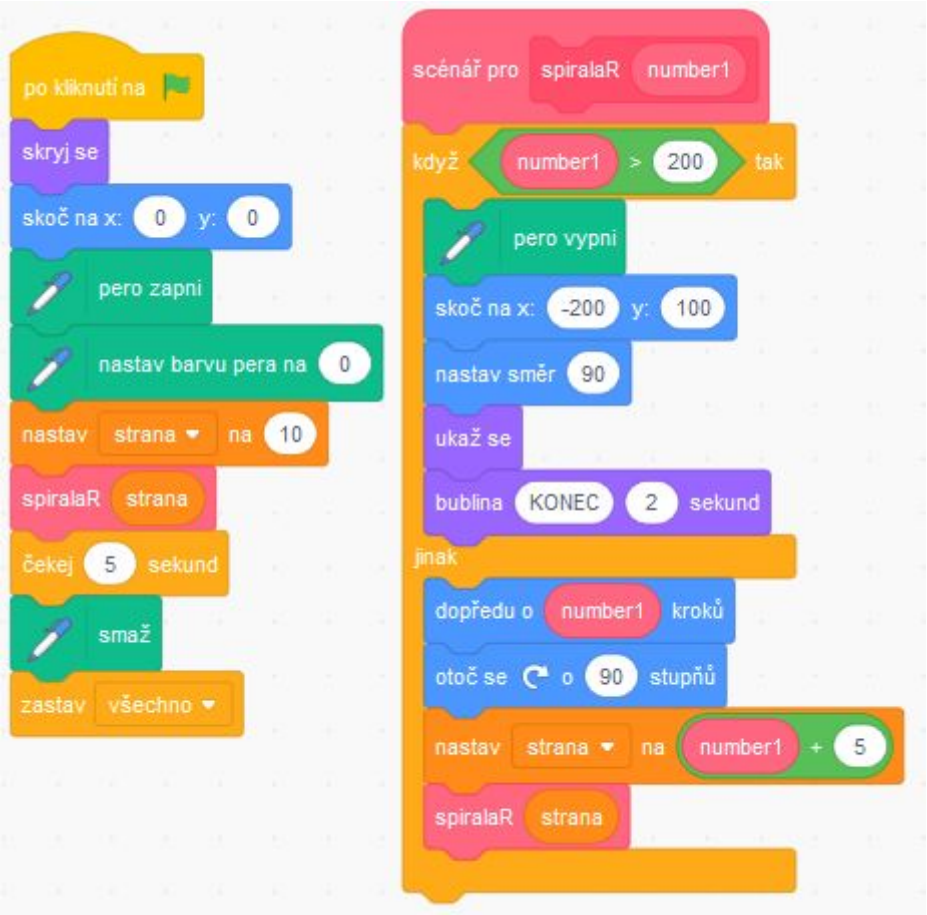
Pravidla pro rekurzivní podprogramy

- Musí být definován konec, rekurze už dál nepokračuje (např. u mocniny pro $x^n = 1$ pro $n = 0$).
- V každém kroku rekurze musí dojít ke zjednodušení problému (volá se procedura s parametrem n – 1).
- Jako první příkaz je podmínka, jestli nastala koncová situace; když ne, provádí se rekurzivní krok.

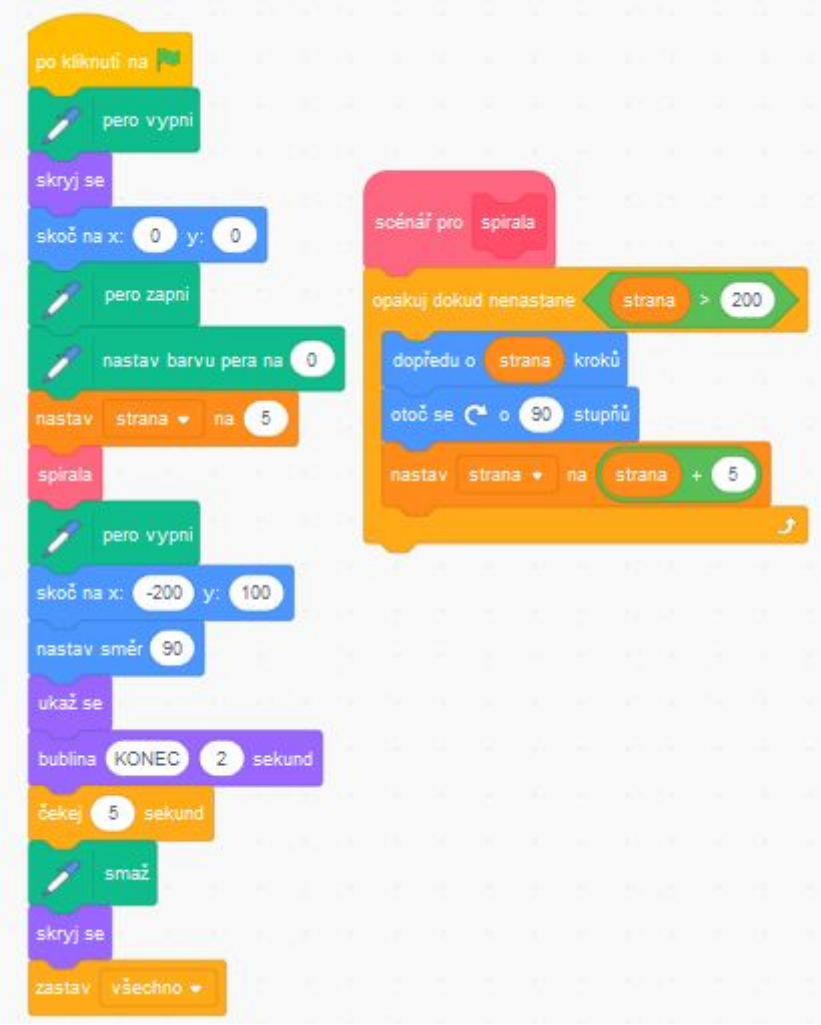
Rekurzivní algoritmy je možné využít i při kreslení grafických objektů („hranatá“ spirála, trojúhelníky v trojúhelníku, čtverce ve čtverci, zvětšující se čtverce se společným rohem apod.).

Je možné společně nakreslit spirálu pomocí rekurze, rychlejší žáci ji mohou vykreslit i pomocí cyklu nebo mohou vykreslit jiné obrazce pomocí rekurze.

Vykreslení „hranaté spirály“ pomocí rekurze:



Vykreslení „hranaté spirály“ pomocí cyklu:



Diskuse k tématu a příkladům

1. Co se stane, když z podprogramu pro vykreslení „hranaté spirály“ odstraníme podmínku? Žáci to mohou vyzkoušet přímo v programu:

   Žáci by měli dojít k závěru, že se program zacyklí.
2. Kdy je vhodné rekurzi použít a kdy ne?

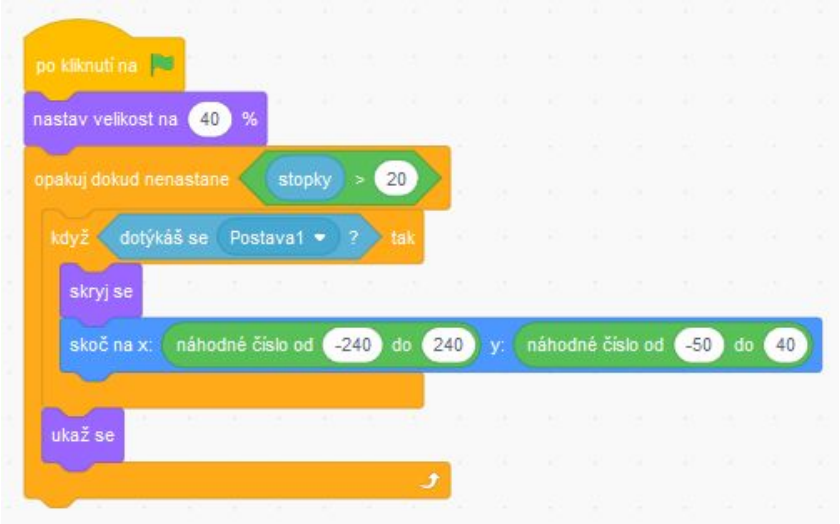
Další inspirace

V této části najdete další příklady v programovacím jazyce Scratch, které by měly pomoci k lepšímu pochopení některých programových konstrukcí a problémů při programování složitějších úloh. Dále jsou zde uvedeny chyby, které se při vytváření programů u žáků obvykle vyskytují.

Možné příklady s použitím více postav v jazyce Scratch

- Jedna postava: kočka, druhá postava: myš. Kočka se pohybuje doprava a doleva pomocí kláves šipka doprava a doleva, myš se objeví na náhodné pozici v části obrazovky, kde se pohybuje kočka, když kočka myš chytí, myš zmizí a po krátké chvíli se přesune na jinou náhodnou pozici.
    - scénář kočka:

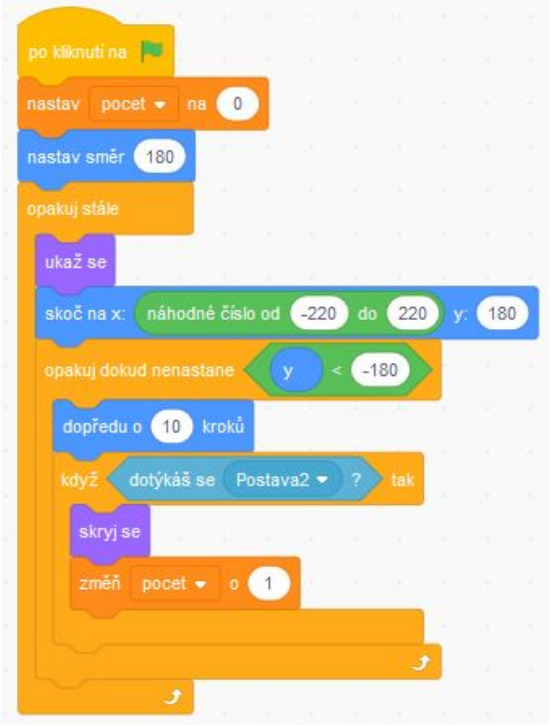
- scénář myš:



  - Rychlejší žáci mohou zobrazit počet chycení (vytvoření proměnné) za určitý čas (žáky je třeba seznámit se stopkami v kartě **VNÍMÁNÍ** – čas se zadává v sekundách, v tomto případě se použije cyklus s podmínkou na začátku místo nekonečného cyklu).

- Jedna postava: hvězda, druhá postava: kočka. Hvězda padá shora dolů vždy z náhodné pozice nahoře, kočka se pohybuje pomocí šipek doprava a doleva a padající hvězdu chytá, počítá se počet chycených hvězd.
  - scénář kočka:

- scénář hvězda:



  - Rychlejší žáci mohou omezit počet padajících hvězd, nastavit čas chytání hvězd, nastavit více úrovní hry (zrychlující se pohyb hvězd i postavy zvětšením kroků) apod.
- Zvíře a potrava, která po snědení zmizí a zvíře řekne MŇAM.
- Jedna postava: pálka, druhá: míček. Pálka se pohybuje zleva doprava při dolní straně a chytá míček, který se odráží od všech stěn. Při dopadu na dolní stranu (místo pálky) hra končí.

Diskuse k příkladům

1. Diskuse o ukončení programu z hlediska uživatele. Je vhodný nekonečný cyklus (**OPAKUJ STÁLE**)? Jak bývají ukončeny aplikace? Jakým způsobem by bylo dobré vytvořené aplikace ukončit?
2. Jak se chovají proměnné v cyklu, který je umístěn uvnitř jiného cyklu? Je vynulování proměnné POCET v příkladu s padající hvězdou umístěno na správném místě?
3. Je třeba si uvědomit, že použití času není přesné, že může dojít ke zpomalování v důsledku toho, že samotný program se spouští v prostředí programovacího jazyku Scratch, který běží v prohlížeči, takže může docházet ke zpoždění.

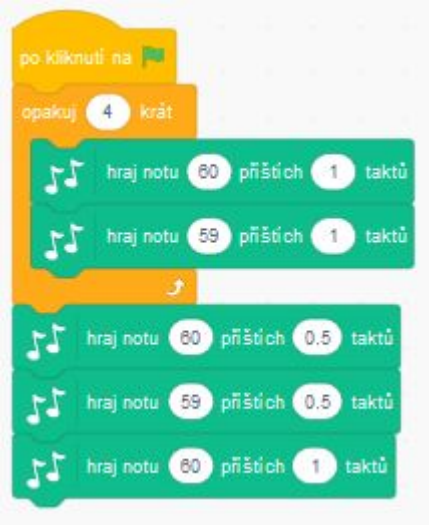
Problém synchronizace

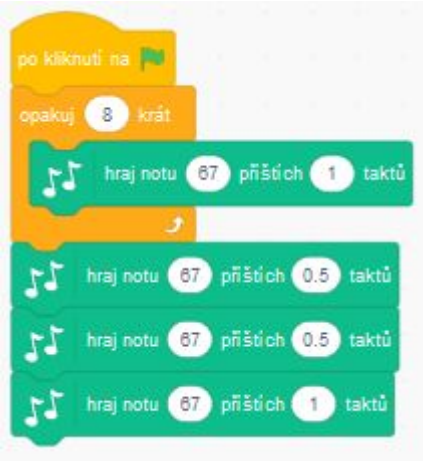
Při programování paralelních procesů je třeba si uvědomit, že ne jejich synchronizace není úplně přesná. Nejlépe je tento jev slyšet u jednoduché písničky naprogramované v jazyce Scratch, ve které by současně měly hrát čtyři tóny a buben (tři tóny v doprovodném akordu,

jeden tón jako melodie a buben). Je dobré si písničku poslechnout několikrát a poslouchat, jak jsou v různých místech nepřesnosti v synchronizaci.

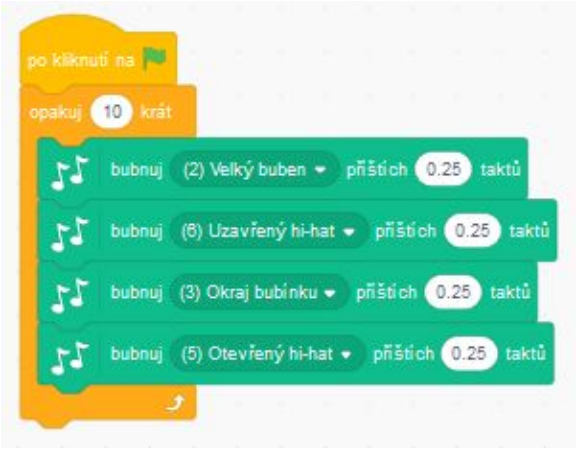
Melodie

Akord (tři tóny)





Bicí



Tyto nepřesnosti jsou způsobeny tím, že programovací jazyk Scratch běží v prohlížeči, takže dochází ke zpoždění a tak i k časovým posunům.

Časté chyby žáků

Seznam častých chyb žáků je možné rozšiřovat podle vlastních zkušeností. Některé chyby způsobí nefunkčnost programu nebo jeho nesprávné fungování. Některé chyby nejsou chybami v pravém slova smyslu, protože program funguje, jen nevyužívá všech možností, které programovací jazyk nabízí (např. používání podprogramů).

- Nastavení proměnné (přiřazení určité hodnoty) na nesprávném místě v programu
- Použití podmínky místo cyklu s podmínkou
- Nepoužití podprogramu místo opakujících se částí v programu

Použitá literatura

DRÁBKOVÁ, J. Možné přístupy k výuce dětských programovacích jazyků. In: *DidactIG*, sborník příspěvků. Liberec: Technická univerzita v Liberci, 2016. ISBN 978-80-7494-189-4.

Klasifikace výukových metod podle I. J. Lernera [online], [vid. 5. 6. 2019]. Dostupné na: http://dielektrika.kvalitne.cz/klasiflerner.html

Scratch [online], [vid. 23. 8. 2019]. Dostupné na: https://scratch.mit.edu/

Výukové metody [online], [vid. 5. 6. 2019]. Dostupné na: http://www.pf.ujep.cz/obecna-didaktika/pdf/Vyukove_metody.pdf

Název	Didaktika programování
Autor	Ing. Jindra Drábková, Ph.D.
Recenzent	Ing. Jana Kolaja Ehlerová, Ph.D.
Vydavatel	Technická univerzita v Liberci
Povoleno	Rektorátem TU v Liberci dne 1. 10. 2019 čj. RE 44/19
Číslo publikace	55-044-19
Počet stran	53
Vydání	první

Tato publikace neprošla redakční ani jazykovou úpravou.

ISBN 978-80-7494-497-0