TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta přírodovědně-humanitní
a pedagogická

Práce s žákovskými chybami v kroužku Programování

Bakalářská práce

Studijní program: B0114A300065 Informatika se zaměřením na vzdělávání
Studijní obory: Informatika se zaměřením na vzdělávání
Anglický jazyk se zaměřením na vzdělávání

Autor práce: **Kateřina Vašíčková**
Vedoucí práce: Mgr. Daniel Lessner, Ph.D.
Katedra aplikované matematiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta přírodovědně-humanitní
a pedagogická

Zadání bakalářské práce

Práce s žákovskými chybami v kroužku Programování

Jméno a příjmení: **Kateřina Vašíčková**
Osobní číslo: P19000016
Studijní program: B0114A300065 Informatika se zaměřením na vzdělávání
Specializace: Informatika se zaměřením na vzdělávání
Anglický jazyk se zaměřením na vzdělávání
Zadávající katedra: Katedra aplikované matematiky
Akademický rok: **2020/2021**

Zásady pro vypracování:

Cílem bakalářské práce je shromáždit a popsat chyby, které dělají žáci při programování v prostředí Scratch, a popsat, jak s nimi začínající učitelka může pracovat.

- Studentka provede rešerši na téma miskonceptů v informatice, chyb při programování a jejich roli ve výuce informatiky. Vymezí pojem chyby v kontextu výuky programování.
- Pozorováním ve vlastní výuce shromáždí vzorky žákovských chyb.
- Pozorované chyby porovná s tím, co očekávala na základě rešerše. Popíše, které chyby se neobjevují a které naopak neočekávala.
- Určí typické nebo jinak významné chyby a popíše, jak s nimi ve výuce pracovala a jaký jim přisuzuje význam pro učení žáka.

Rozsah grafických prací:	dle potřeby
Rozsah pracovní zprávy:	cca 45 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

- ČERNOCHOVÁ, M., P. VAŇKOVÁ a J. ŠTÍPEK. *Programování ve Scratch pro pokročilé –Projekty pro 2. stupeň základní školy.* Praha: Univerzita Karlova, Pedagogická fakulta, 2020. ISBN 978-80-7603-085-5.
- SPIŠÁKOVÁ, M. a Ľ. SALANCI. *Chyby ako súčasť motivácie programovania.* DidInfo and DidactIG 2017. Banská Bystrica : Univerzita Mateja Bela, 2017. ISBN 978-80-557-1216-1. s. 136–140.
- SWIDAN, A., F. HERMANS a M. SMIT. *Programming Misconceptions for School Students.* In ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research. ACM New York, NY, USA, 2018. s 151–159.
- GUZDIAL, M. *Learner-Centered Design of Computing Education: Research on Computing for Everyone.* Morgan & Claypool, synthesis edition. ISBN 978-1-62705-351-8. kap. 2–3.

Vedoucí práce:	Mgr. Daniel Lessner, Ph.D. Katedra aplikované matematiky
Datum zadání práce:	4. června 2021
Předpokládaný termín odevzdání:	30. července 2021

L.S.

prof. RNDr. Jan Picek, CSc.
děkan

doc. RNDr. Miroslav Koucký, CSc.
vedoucí katedry

V Liberci dne 4. června 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracovala samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědoma toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědoma následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

28. dubna 2022 Kateřina Vašíčková

Poděkování

Chtěla bych poděkovat všem, díky kterým byla tvorba této práce možná. Jmenovitě svému vedoucímu práce, Mgr. Danielu Lessnerovi, Ph.D., za věcné rady a pomoc při tvorbě práce, Mgr. Janu Berkimu, Ph.D. za pomoc při výběru vhodného tématu, doc. Mgr. Cyrilu Bromovi, Ph.D za doporučení dalších zdrojů a své sestře Bc. Zuzaně Hartmanové za pomoc s korekturou. V neposlední řadě bych ráda poděkovala také své rodině, přátelům a partnerovi, bez jejichž podpory bych se neobešla.

Abstrakt

Cílem této práce bylo shromáždit a popsat chyby, které dělají žáci při programování v prostředí Scratch, a popsat, jaký vliv mají na žáka. Kromě toho práce obsahuje kapitolu zabývající se přímo programovacím jazykem Scratch.

V práci se chybám nejdřív věnuji teoreticky a poté empiricky. Na začátku empirické části je popis pozorování i vzorku pozorovaných žáků. Dalším krokem práce bylo zhodnocení přínosu chyb k procesu učení a nastínění, kterým chybám se můžeme pokusit zcela předcházet a které naopak prospívají žákovi více, když je sám udělá, objeví a s případnou pomocí opraví.

Klíčová slova: programování, Scratch, chyby, práce s chybou

Abstract

The aim of this thesis was to gather and describe mistakes made by pupils while programming with Scratch and describe the mistakes' effect on the pupil. In addition the thesis includes a chapter dealing with Scratch programming language itself.

I start with the theory of the mistakes then discuss them from the empirical side. There is also a description of the study and studied sample of pupils in the theoretical part of the work. The next step of this thesis was evaluation of the mistakes' contribution to the learning process and suggest which mistakes it might be desirable to prevent altogether and which help the pupil more when they can make, discover and with possible help correct them.

Keywords: programming, Scratch, mistakes, mistake handling

Obsah

Seznam obrázků

Seznam grafů

Úvod

Říká se, že chybami se člověk učí. Metcalfe ve své studii zkoumá, zda tomu tak je a jestli pokusy o prevenci chyb žákům naopak neškodí. Uvádí, že chyby opravdu pomáhají při učení, ovšem jen pokud jsou vhodně opraveny a správné řešení je dostatečně vysvětleno [1]. Spišáková a Salanci ve svém článku také uvádí, že chyby rozvíjejí mozek a jsou nedílnou součástí procesu učení [2]. Tento názor sdílejí i autoři metodické příručky učebnice Programování ve Scratch pro pokročilé a doporučují, aby vyučující dal žákům příležitost dělat chyby a poučit se z nich. Žáky je podle nich vhodné povzbuzovat aby se nebáli chybovat a aby se po první chybě nevzdávali [3 s. 8–9].

Práce se dělí na dvě hlavní části – teoretickou a empirickou.

Teoretická část sestává z kapitol 1 a 2. Kapitola 1 se zabývá prostředím Scratch, popisuje pro práci relevantní možnosti tohoto programovacího jazyka. Také dává základ pro podrobnější popis chyb, který je v následující kapitole, respektive její podkapitole 2.2. Kapitola 2 se nejprve zabývá obecně popisy chyb, které mohou vzniknout při programování, poté se zaměřuje na chyby v programovacím jazyce Scratch.

V průběhu vedení kroužku programování jsem pozorovala a zaznamenávala žákovské chyby. Těmito údaji se poté zabývá empirická část sestávající z kapitol 3 a 4. Kapitola 3 se zabývá způsobem a specifikacemi pozorování. Obsahuje bližší informace jak o výuce samotné, tak o vzorku pozorovaných žáků. Kapitola 4 obsahuje výsledky pozorování – od typů chyb a jejich výskytů, přes konkrétní příklady až po postoj žáků k chybám. Zmiňuje také které chyby žáci opakovali. V závěru této kapitoly jsou i ukázky zajímavých a specifických chyb, které žáci během pozorování udělali.

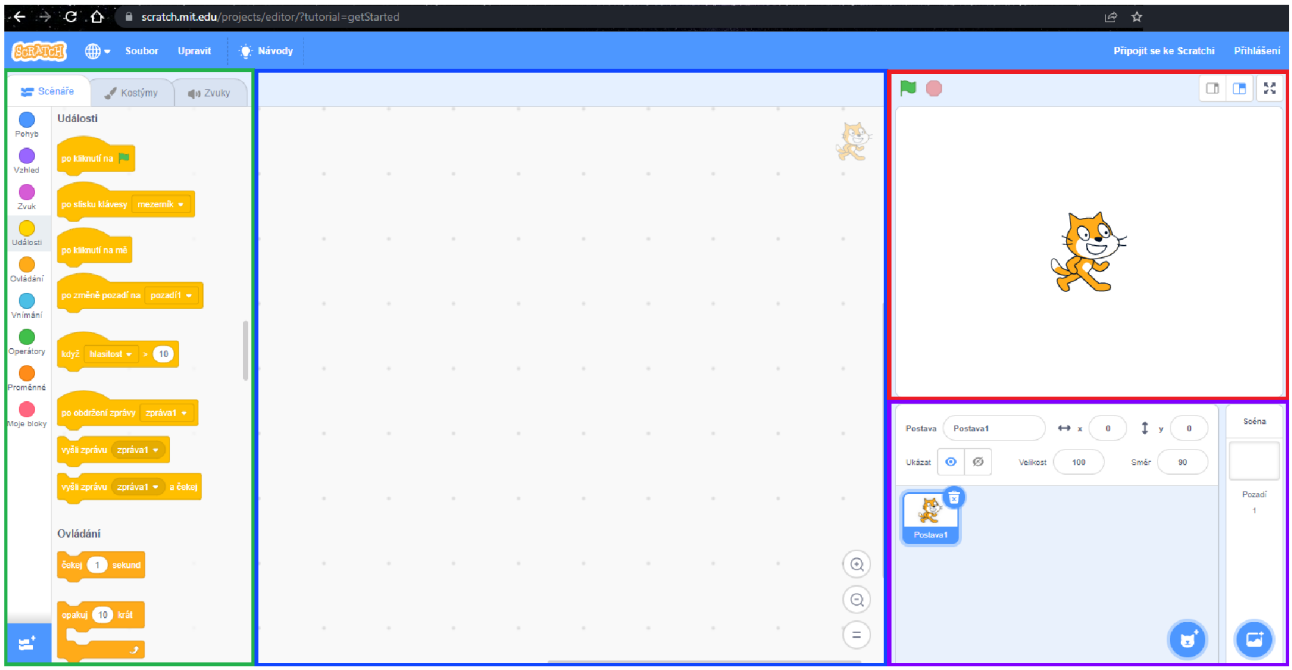
Závěr pak shrnuje celou práci a vyzdvihuje její výsledky a důležité informace.

1 Scratch

Scratch 3.0 (dále jen Scratch) je vizuální blokový programovací jazyk. Vytvořila jej skupina Scratch z Lifelong Kindergarten group na MIT Media Lab. Dostupný je zdarma na https://scratch.mit.edu. [4] Pro začátek je dobré pochopit, v jakém programovacím prostředí byly chyby pozorovány, proto se v této kapitole zaměřím na jeho představení.

1.1 O blocích a prostředí Scratche

Samotné prostředí Scratche v jeho webové verzi má čtyři základní části (viz Obrázek 1). Offline aplikace vypadá podobně, tato práce se ovšem zabývá chybami a ne Scratchem samotným, proto by popis a specifikace obou variant byly nadbytečné.



Obrázek 1: Rozložení Scratche

Zleva doprava, odshora dolů: zeleně označená část je především **zásobník bloků kódu**, které lze vzít myší a přetáhnout do modré části – to je **pracovní plocha** pro skládání programu. Každá postava a pozadí má svou vlastní pracovní plochu a tím i scénář[1]. Červená část je **plátno projektu**, zde se ukazuje projekt a jeho neskryté části. Pod ní, fialově označené, je místo pro **postavy a pozadí**, také jsou v něm vidět některé informace o vybrané postavě (jako název, souřadnice umístění na plátně nebo třeba zda je viditelná či nikoliv).

Scratch je tedy vizuální blokový programovací jazyk, jehož bloky do sebe zapadají podle svého typu. Výjimkou jsou logické bloky, které lze vložit do všech prostorů pro zadání parametru, nejen těch pro tento typ bloku určených, a stejně tak do oválných oblastí rolovací nabídky.

1 Scénářem nazýváme kompletní spustitelný kód každé postavy a pozadí.

Bloky Scratche lze rozdělit na šest základních kategorií podle jejich tvaru, respektive toho, jak na sebe jdou „stavět“.

Bloky události – tento typ bloku může být pouze na začátku sekvence příkazů a lze na ně napojit pouze bloky příkazové, řídící bloky nebo bloky ukončovací (viz Obrázek 2). Tyto bloky umožňují spouštění scénářů, které jsou na ně napojeny. Některé (například „po stisku klávesy __“ nebo „po kliknutí na mě“) umožňují uživateli spouštět scénáře napřímo. Jiné (například „po změně pozadí na __“ nebo „po obdržení zprávy __“) musí proběhnout uvnitř programu a uživatel do nich nemůže přímo zasahovat. O konkrétních možnostech zásahu do průběhu programu ovšem samozřejmě rozhoduje naprogramování konkrétního projektu.

Bloky příkazové – tento typ bloku lze napojit za blok události, řídící blok nebo další blok příkazový, také jej lze vnořit do řídících bloků. Lze za něj napojit všechny tyto zmíněné bloky (s výjimkou bloku události) a bloky ukončovací (viz Obrázek 3). Tyto bloky jsou nejpočetnější kategorií a obsahují všechny bloky vyvolávající nějakou akci, případně pokus o ni. Pokusem je zde myšleno například pokus o změnu kostýmu na aktuální kostým nebo pohyb na současné souřadnice. Nejčastěji se jedná právě o zmiňované skupiny pohybu a změny vzhledu (jak postavy, tak pozadí), méně často poté o práci se zvukem či proměnnými, v jednom případě i o blok skupiny vnímání.

Bloky alfabetické či numerické hodnoty – tyto oválné bloky nejsou samostojné a je třeba je vložit do příslušného prostoru pro parametr (oválná místa, viz Obrázek 4), kde je jejich hodnota zpracována. Tato místa se vyskytují u alespoň jednoho bloku všech kategorií a to včetně této kategorie samotné. Mezery pro parametry mohou být vyplněny buď těmito bloky, nebo napřímo po kliknutí do daného prostoru a zápisem z klávesnice. Ovšem ne všechny přijímají jak numerické tak alfabetické hodnoty. Například bloky aritmetických operací za běžného používání[2] nepřijímají alfabetické znaky. Kromě míst pro parametry lze tyto bloky vložit také do rolovací nabídky stejného tvaru (viz Obrázek 5 blok „vyšli zprávu (zpráva1)“). Více o tomto jevu se dočtete v podkapitole 1.2.

Bloky logické hodnoty (true nebo false[3]) – tyto bloky (viz Obrázek 6) opět nejsou samostojné a je třeba je vložit do jejich příslušného místa pro parametr (nezaoblená místa,viz Obrázek 8), tam je jejich hodnota zpracována. Tato místa se vyskytují především u bloků řídících a bloků vracejících logickou hodnotu, mohou se ovšem vyskytnou i u bloku příkazového. Do těchto míst pro parametry je nutné vložit příslušný blok a nelze do nich napsat logickou hodnotu napřímo. Mimo míst pro parametry je možné tyto bloky také vložit do oválných rolovacích nabídek. Více v následující podkapitole 1.2.

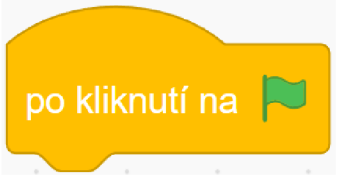
2 Při použití klávesové zkratky Ctrl + V lze vložit alfabetické znaky, zadáním z klávesnice nikoliv.
3 Ve Scratchi jsou opravdu hodnoty true a false zobrazovány s malými písmeny.

Bloky řídicí – obvykle mají tvar písmene C (viz Obrázek 7). Díky jejich tvaru je lze napojovat stejně jako bloky akce a zároveň do nich lze vnořovat bloky akce či další řídicí bloky. Výjimkou je blok „opakuj stále“, který funguje také jako blok ukončovací a nelze za něj nic napojit. Tyto bloky mají řídicí funkce, lze jejich pomocí vytvářet například cykly nebo větvit program pomocí podmínek.

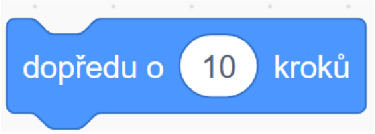
Bloky ukončovací – tyto bloky musí být na konci sekvence, neboť za ně už nelze nic napojit (viz Obrázek 9). Drobnou výjimku představuje jejich vnoření do některého z řídících bloků – například v podmínce. V takovém případě tímto blokem celá část kódu nemusí končit, ale daná větev ano. Respektive pod tímto blokem se mohou vyskytovat další bloky, žádné z nich však nebudou přímo napojeny na blok ukončovací (viz Obrázek 10).

Bloky jsou rozděleny do skupin, viz Obrázek 11, dále jsou k dispozici rozšíření, jako je pero nebo třeba napojení na LEGO BOOST.

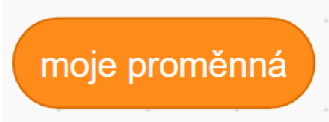
Většina kategorií bloků (až na bloky obsahující alfabetickou či numerickou hodnotu) obsahují rolovací nabídku, vybrat lze v závislosti na konkrétním bloku například klávesa, postava nebo barva.



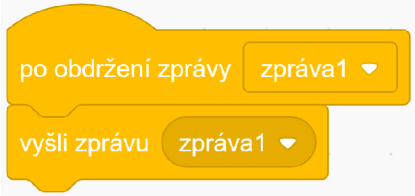
Obrázek 2: Blok události



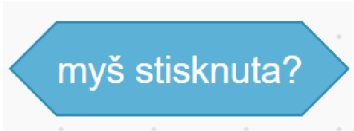
Obrázek 3: Příkazový blok



Obrázek 4: Blok alfanumerické hodnoty

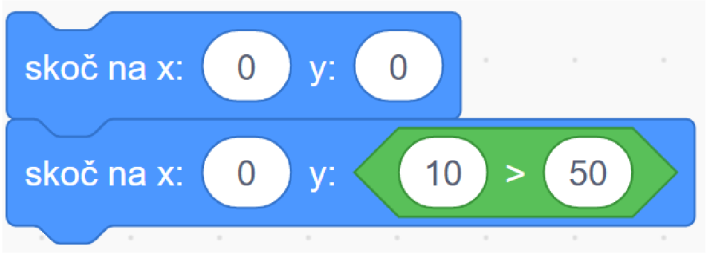


Obrázek 5: Rolovací nabídka, nezaoblená a zaoblená



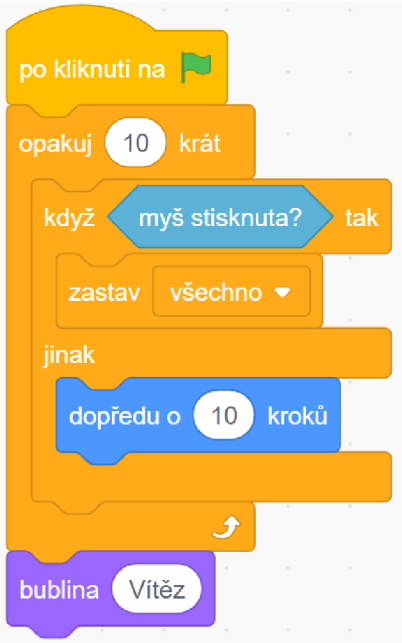
Obrázek 6: Blok logické hodnoty

Obrázek 7: Řídící blok



Obrázek 8: Boolean hodnota v oválné mezeře

Obrázek 9: Ukončovací blok



Obrázek 10: Ukončovací blok uvnitř podmínky

Obrázek 11: Skupiny bloků

1.2 Limity a specifika ve Scratchi, jejich klady a zápory

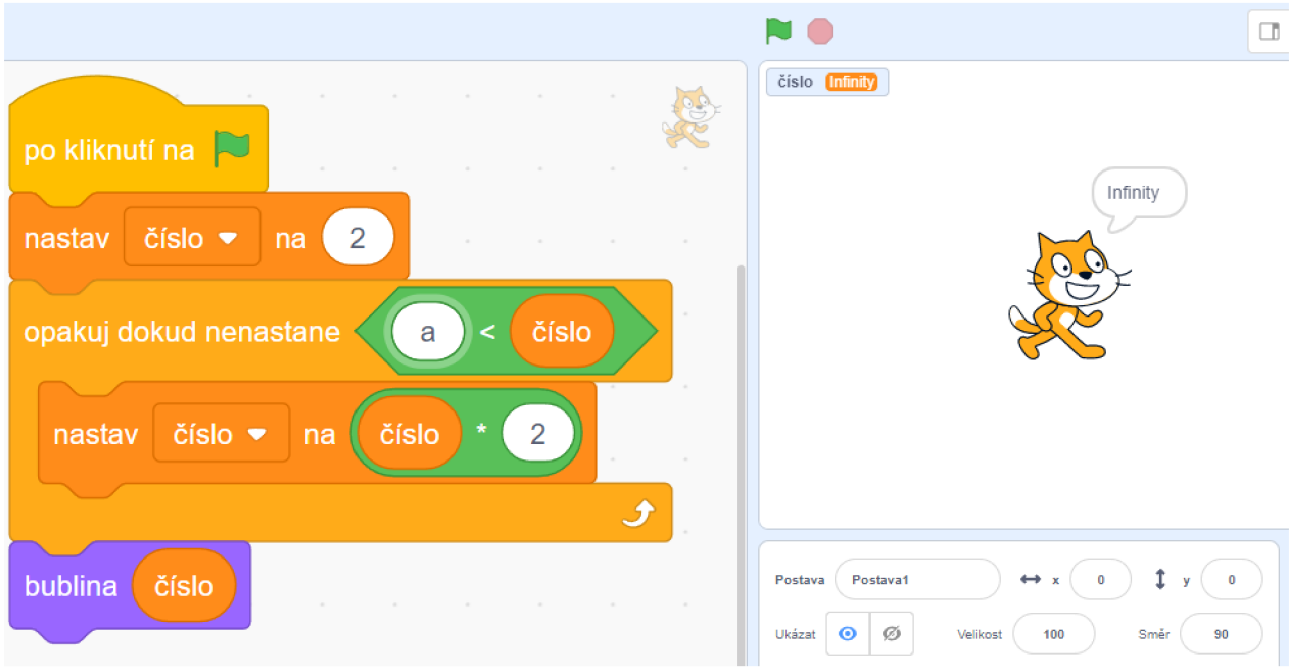
Jako každý programovací jazyk, i Scratch má své limity, s nimiž přichází jejich klady a zápory. Je nutno podotknout, že při hodnocení některých z nich nemusí být lehké dojít k závěru, se kterým bude souhlasit každý.

Jedním z hlavních specifik oproti jiným programovacím jazykům je **umístění souřadnicových os x a y** na plátně projektu. Scratch má průsečík os, tedy bod na souřadnicích [0,0], ve středu plátna, nikoli v jeho rohu. Kopíruje tím obvyklé rozložení i orientaci[4] os, se kterým se žáci setkají například v matematice. Pro žáky je pak toto rozložení jednodušší právě díky návaznosti na matematiku.

Jak již bylo uvedeno v předešlé části kapitoly, bloky do sebe ve většině případů zapadají podle svého typu a chybovat tak použitím nesprávného datovému typu příliš nelze. Není to ovšem nemožné. Pravděpodobně právě z důvodu vysoké nepravděpodobnosti, že k této chybě dojde, nejsou tyto možnosti vhodně ošetřeny.

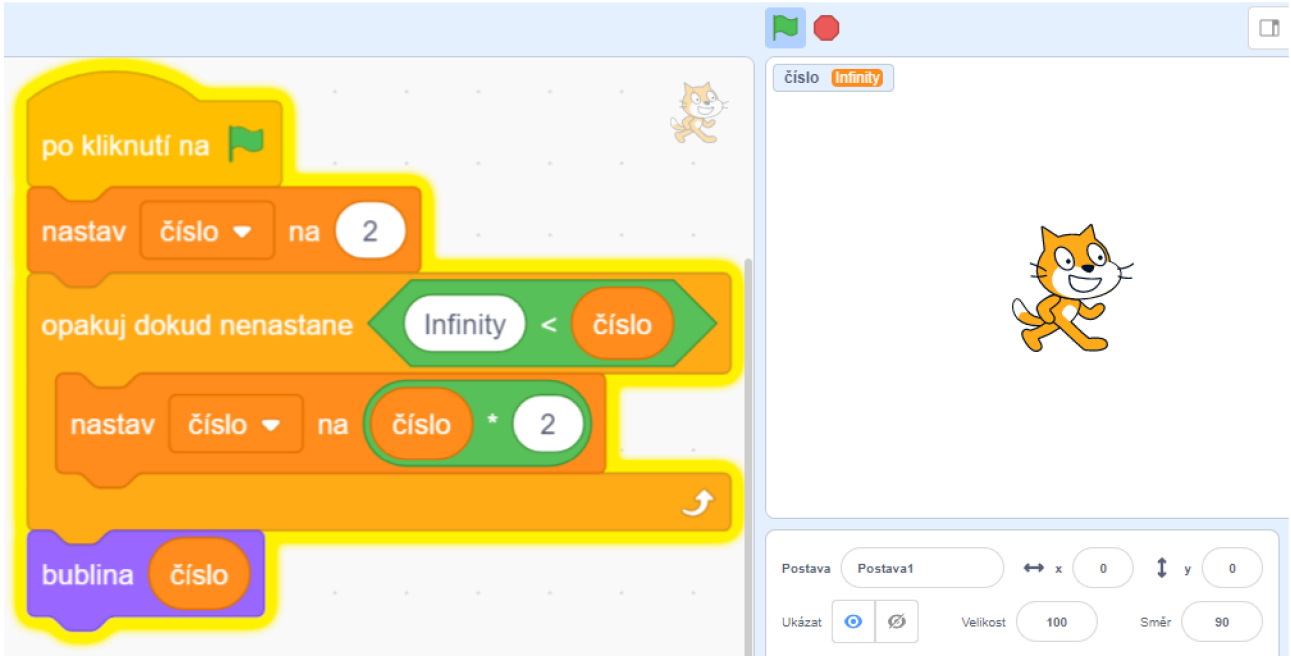
Jako nevýhodu v tomto případě vnímám možnost **zápisu nenumerických znaků do operátorů porovnání** "<" a ">", přestože zřejmě nemají žádnou numerickou hodnotu. Při pokusu o jejich násobení – to musí být buď pomocí proměnných nebo vložením pomocí klávesové zkratky Ctrl+V, totiž vždy vyjde 0. Avšak zadání $a > 1$ do operátoru porovnání vrací true, zároveň porovnání $aa > a$ vrací true. Zároveň ale $-a > 1$ vrací false. Násobení i zbylé aritmetické operace jsou tedy ošetřeny, porovnávání nikoli.

4 Znaménka souřadnic jsou v kvadrantu I (+, +) a kvadrantu III (-, -).



Obrázek 12: Porovnávání nenumerické hodnoty s numerickou č. 1

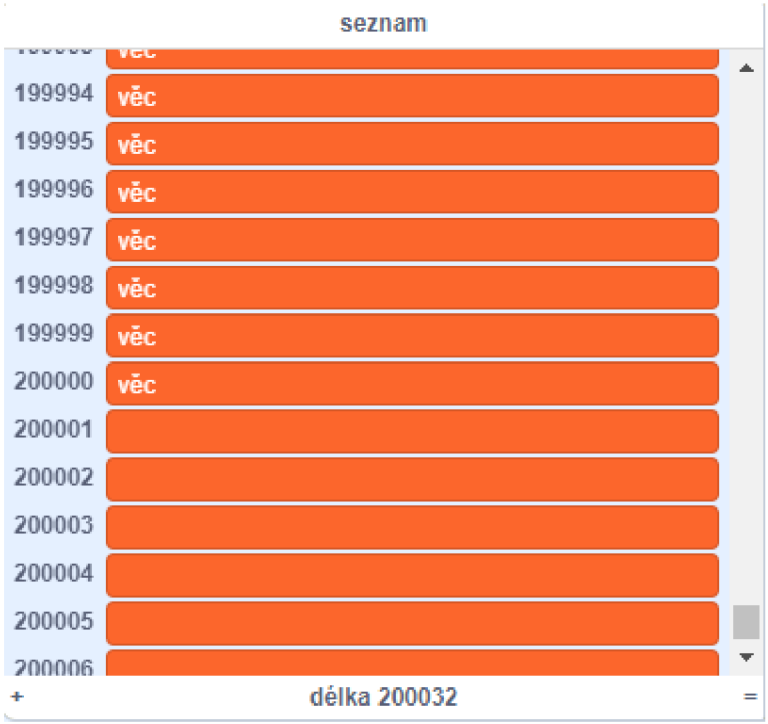
Z první ukázky (viz Obrázek 12) se zdá, že numerická hodnota písmene *a* je menší ve chvíli, kdy hodnota proměnné číslo je nekonečno. Ovšem následující ukázka (viz Obrázek 13) odhaluje, že ve skutečnosti ani zde neporovnáváme alfabetickou hodnotu s numerickou, ale dvě alfabetické. Hodnota proměnné číslo totiž není při tomto porovnávání nekonečno, jak by si žáci mohli myslet, ale text Infinity. Tato zjištění otevřela dveře dalším poznatkům problematiky datových typů ve Scratchi, nejsou ovšem hlavní částí práce. Především kvůli jejich specifickému charakteru a rozsahu potřebného k jejich objasnění nejsou v práci dále rozváděny.



Obrázek 13: Porovnávání nenumerické hodnoty s numerickou č. 2

Jako klad vnímám limit **300 aktivních klonů**[5], žák se tím může přiučit něco o výpočetní kapacitě stroje. Je ovšem vhodné na toto téma udělat přímo aktivitu, protože z běžného používání tato skutečnost nemusí vyplynout. Scratch na dosažení daného limitu nijak neupozorňuje, program pouze přestane klony vytvářet.

Podobně má limit i **délka seznamu**. Pomocí opakování lze vložit do seznamu až 200.000 položek. Zajímavá část přichází ovšem v okamžiku, kdy i při dosažení 200.000 položek v seznamu, lze přidat další položky. Nelze tak již učinit blokem „přidej __ k [název seznamu]“, ale malým znaménkem plus v dolní části zobrazení seznamu (viz Obrázek 14). Zobrazí se prázdná položka, kterou lze, stejně jako blokem vytvořenou, přepsat na něco jiného.



Obrázek 14: Seznam

Limit zde byl vytvořen protože množství zhruba 300.000 položek v seznamu bylo pro chod programu kritické [6]. „Vada“ toho, že lze tento až absurdní limit 200.000 položek obejít pomocí tlačítka na přidání položky je tedy nepodstatná, neboť rozhodně v běžném uživatelském použití nedojdeme k číslu blížícímu se počtu 300.000 položek.

V tomto případě považuji limit za spíše technický a nijak neovlivňující běžnou práci žáka. Může být využit na zajímavou aktivitu a zkoumání, v jakých případech by nám tento limit mohl způsobit nějaký problém a jak by jej bylo možné řešit.

U **funkce bloku „opakuj __ krát“** (obecně bloků „opakuj“), je zajímavé, že následující dvě ukázky kódu (viz Obrázek 15) ve skutečnosti na plátně projektu proběhnou jinak.

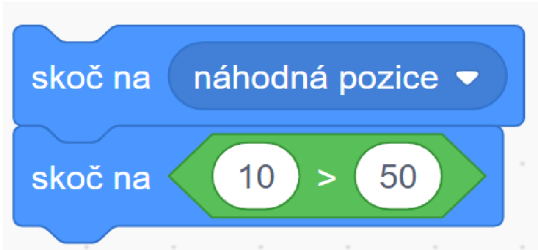
Zatímco kód vlevo posune postavu čtyřikrát o 10 kroků a vytvoří tím iluzi více či méně plynulého pohybu, kód napravo posune postavu jedenkrát o 4×10 kroků, tedy najednou změní pozici o 40 bodů ve směru, kam je postava natočená.

Tento fakt je ovšem opět spíše zajímavý, než že by výrazně narušoval práci žáka. V ukázkovém případě níže (viz Obrázek 15) bychom pro využití možnosti kódu napravo a zároveň použití co nejmenšího počtu bloků použili blok „dopředu o 40 kroků“. Drobného zpoždění lze naopak vyu-

5 V importovaném prostředí Scratch na platformě Algorithmics i v článku o klonování na Scratch wiki [5] je toto maximum 300 klonů, při reprodukci stejného kódu v prostředí prohlížečového Scratche se dostaneme na množství 301.

žít pro již zmiňovanou imitaci plynulého pohybu (obecně čím menší počet kroků v jednom opakování, tím plynulejší pohyb).

Obrázek 15: Porovnání opakuj vs. blok vícekrát

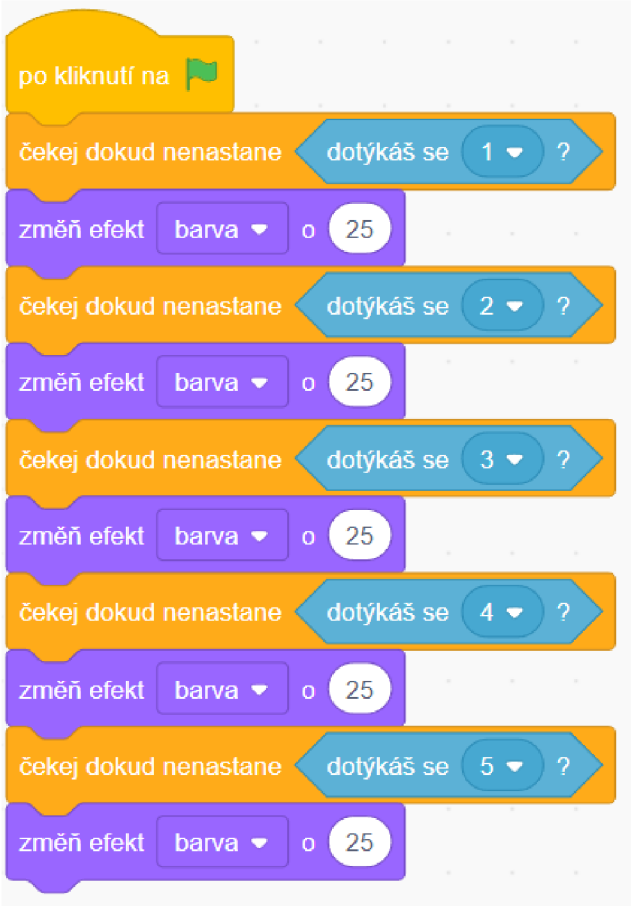


Obrázek 16: Boolean hodnota v rolovací nabídce

Jak již bylo v předešlé podkapitole uvedeno, bloky vracející hodnotu (ať už alfabetickou, numerickou či logickou) lze vložit do zaoblených oblastí rolovací nabídky (viz Obrázek 16).

Zprvu jsem tuto skutečnost považovala za vadu jazyka, ovšem později jsem zjistila, že se jedná o využitelnou funkci. Musím ale nejprve popsat další zajímavou funkci Scratche. Tou je možnost uložení jména postavy jako hodnoty proměnné a blok obsahující hodnotu proměnné poté používat pro odkazování se na danou postavu. V tuto chvíli tedy víme, že lze uložit de facto odkaz na postavu do proměnné. Tato proměnná lze následně vložit do oválné nabídky libovolného bloku. V tuto chvíli není příliš zajímavá možnost využití této funkce, neboť relevantní oblasti nabídky obsahují mimo jiného i názvy jednotlivých postav. Ovšem postavy lze pojmenovat i pomocí čísel. V takovém případě začne být zajímavá možnost jednoduché úpravy argumentu předávaného bloku s nabídkou.

Představte si zadání projektu, kde je cílem naprogramovat hru, ve které se hráč musí dotknout věcí (jiných postav) v určitém pořadí. Dejme tomu, že postav k posbírání je 5, pokud bychom použili kombinaci bloků „čekej dokud nenastane“, „dotýkáš se [název postavy]“ a bloků definující akci/akce, ke kterým v takovou chvíli má dojít. V takovou chvíli se můžeme jednoduše dostat k počtu bloků minimálně 16 (nesmíme zapomínat na událost spouštějící danou část programu). Ovšem v případě, že bychom si vhodně pojmenovali postavy od 1 do 5, šlo by stejný program naprogramovat s menším množstvím bloků a to s 8. S vyšším množstvím postav by se tento rozdíl pouze prohluboval. Pro jasnější porovnání i lepší představu ukázky přikládám Obrázek 17 a Obrázek 18.



Obrázek 17: Porovnání 1: bez bloku hodnoty v rolovací nabídce

Obrázek 18: Porovnání 2: blok hodnoty v rolovací nabídce

2 Chyby při programování

Chyba, neboli omyl, má mnoho různých podob a každý si pod tímto pojmem může představit něco naprosto odlišného. Proto je důležité tento pojem definovat, aby se předešlo nedorozumění z nesprávného pochopení.

Chybou při programování je v této práci myšlena chyba, která lze objevit z kódu či z výsledného programu (co dělá, jak reaguje na vstup, apod.). Chyby v rámci fyzické tvorby programu[6] by šly zaznamenávat jedině v pozorování jednoho žáka jedním pozorovatelem. Případně také zcela upřímnou zpětnou vazbou od žáka samotného, ovšem v takovém případě by žák nemusel chybu sám objevit.

Chyby při programování je pak pro potřeby této práce dobré rozdělit na chyby, které platí obecně pro většinu programovacích jazyků, a ty, které lze pozorovat ve Scratchi.

2.1 Typy chyb při programování a vymezení souvisejících pojmů

Chyby při programování obvykle dělíme na chyby syntaktické a sémantické (někdy také označované jako logické).

Chyby **syntaktické** lze rozdělit ještě na přímo závislé na konkrétním programovacím jazyce (špatně napsané klíčové slovo, chybějící středník, …) a nezávislé (překlepy). Tyto chyby je obvykle snazší odhalit, překladač často vyšle uživateli chybové hlášení nebo v závažnějších případech program vůbec nespustí. Chybové hlášky nám mohou dokonce ukázat i na kterém řádku se chyba pravděpodobně nachází a díky typu hlášky může být snazší ji najít.

Chyby **sémantické** potom způsobují, že program funguje jinak, než by měl, ale funguje, a tak se často zdá, že je vše v pořádku. To znamená, že může být složité vůbec zjistit, že v programu chyba je, natož ji poté najít a opravit. Tyto chyby nemusí být přímo závislé na programovacím jazyce (například použití nesprávného porovnávání < namísto <=, inicializaci proměnné uvnitř podmínky, která nemusí projít, …).

Chyby mohou být způsobeny miskonceptem, úplnou neznalostí nebo třeba nepozorností. Podle Swidana mezi obvyklé miskoncepty patří nesprávné pochopení následujících konceptů: sekvenční návaznosti jednotlivých příkazů, způsobu fungování proměnné (ukládání a pamatování si hodnoty – jedna hodnota v jeden okamžik) – a interaktivity programu ve chvíli, kdy má přijmout něco na vstupu (input) [7].

6 Například použití mezerníku namísto tabulátoru pro odsazení nebo opakované přetahování stejných bloků namísto jejich kopírování.

Guzdial upozorňuje na fakt, že některé chyby jsou způsobeny kognitivními preferencemi žáků [8]. Pokud žák vymyslí strategii řešení problému, ale jazyk ji nepodporuje, vznikne buď další problém, který je třeba vyřešit, nebo chyba. Je ovšem důležité si uvědomit, že i když programovací jazyk obsahuje všechny nezbytné prvky pro použití dané strategie, může žák udělat stejné chyby jako v opačném případě, a to v důsledku výše zmiňovaného miskonceptu či úplné neznalosti. V takovou chvíli je nutné se zamyslet, zda učíme programovací jazyk, nebo algoritmické myšlení.

Chyby při programování je možné rozdělit také podle jejich původu na chyby na úrovni **algoritmu** a chyby na úrovni **kódu**. Rozdíl v nich je především v tom, kde se nachází neporozumění žáka. V případě algoritmu se může jednat o obecný miskoncept zatímco v případě chyby na úrovni kódu může jít o (částečné) nepochopení daného programovacího jazyka.

2.2 Specifika chyb při programování v jazyce Scratch

Jak již bylo uvedeno v předešlé kapitole, Scratch je vizuální blokový jazyk. Snižuje se tak riziko udělání některých chyb (např.: použití špatného datového typu, špatné odsazení) a dalším se zcela předchází (např.: překlepy v klíčových výrazech, chybějící závorka či středník). I tak je ve Scratchi prostor pro vytvoření poměrně velkého množství chyb.

Práce pracuje i s pojmem miskoncept, který, jak už bylo uvedeno v předešlé podkapitole, je jedním z původců chyby. Chybou potom souhrnně nazývá vše, co negativně ovlivňuje chod žákovského programu.

V tomto případě nebudeme chyby dělit na syntaktické a sémantické, neboť syntaktickým chybám se Scratch snaží svou podstatou předcházet. Rozhodla jsem se chyby rozdělit podle jejich závažnosti na chyby funkční a odchylky od kanonického postupu.

Funkční chyba zabraňuje správné funkci – něco kvůli ní proběhne jinak, než by mělo, nebo vůbec. Ve Scratchi se obvykle projeví vizuálně, akusticky nebo oběma způsoby.

Funkční chyby lze rozdělit do několika skupin:

- nesprávný blok kódu pro konkrétní situaci
- nesprávné pořadí bloků kódu či jiné narušení sekvence
- chybějící blok kódu či jeho definice[7]
- nesprávné zacházení s proměnnou

Odchylky od kanonického postupu – program funguje tak, jak má, ale kód kvůli nim může být nepřehledný či jeho tvorba zabírat delší dobu, než by bylo nutné.

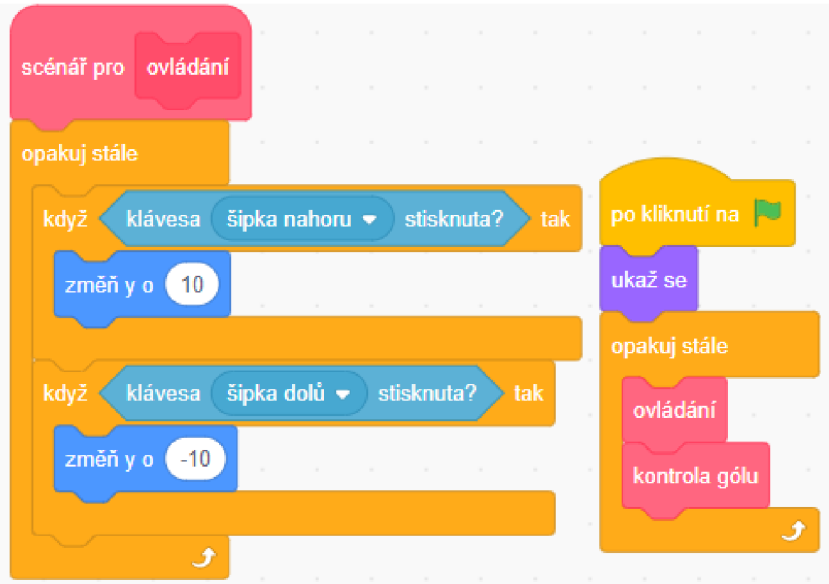
Tyto odchylky zahrnují:

- jakýchkoli duplicitních kódů,

7 Definicí je zde myšlena především inicializace proměnné či jiného párového bloku (Zprávy, Otázka, Můj blok).

• použití složitějších konstrukcí namísto využití kanonického postupu.

Stejné chyby mohou být u různých žáků způsobeny různým důvodem. Ilustrativní příklad na autentické chybě:



Obrázek 19: Nesprávně umístěný nekonečný cyklus

Žák 1, Žák 2 i Žák 3 udělají totožnou chybu[8] ve svém kódu (viz Obrázek 19). Skupina Žáka 1 ještě neprobírala téma Moje bloky a tak neví, jak fungují. Skupina Žáka 2 již toto téma probrala, ovšem Žák 2 jej zcela nepochopil. Skupina Žáka 3 téma také probrala a Žák 3 mu dostatečně rozumí, předělával ovšem svůj kód tak, aby odpovídal zadání a zapomněl ho zkontrolovat.

Negativní dopad chyb **funkčních** je pravděpodobně jasný, program **funguje nesprávně**. S tímto typem chyby se běžně setkáváme například i v matematice. Pokud uděláme chybu ve výpočtu, je nám jasné, že ji musíme opravit, abychom dosáhli správného výsledku.

U **odchylek** od kanonického postupu tento jasný závěr nemusí hned vyplývat, mohou se totiž zdát v porovnání s chybami funkčními jako nepodstatné. Je to přeci žákova práce navíc a ke správnému výstupu se nakonec dostal. Ovšem nejen u programování je třeba si uvědomit, že co se sotva projeví v drobném projektu (nebo výpočtu), může mít významný dopad na chod programu rozsáhlejšího (výpočet delšího příkladu). U programování se jedná především o možnost bezdůvodného **navýšení výpočetní složitosti** a tím **nechtěnému zpomalení běhu programu** (ve Scratchi méně časté), ale také o zdržování při samotné tvorbě. To se projeví ještě víc v případě skupinové práce na jednom projektu. Kód je pak často **nepřehledný** a v důsledku toho se v něm mohou udělat další chyby. To může nastat nejen ve skupinové práci, ale i ve chvíli, kdy se k němu žák po čase vrátí, aby jej opravil či rozšířil.

8 K provedení scénáře bloku kontrola gólu nikdy nedojde, protože scénář pro blok ovládání obsahuje nekonečný cyklus. Událost „Scénář pro [vlastní blok]“ je spouštěn odpovídajícím hranatým blokem.

Ve všech zkoumáních je třeba zvážit i znalosti žáka – nelze očekávat, že bude používat určitou funkci programovacího jazyka, pokud ji ještě nezná. V takovém případě použití složitější konstrukce nelze považovat za chybu jako takovou, naopak by měla být tato vynalézavost oceněna a, pokud to situace dovoluje, měla by být spolu s kanonickou konstrukcí žákům náležitě vysvětlena a to včetně důvodu, proč je jedno řešení vhodnější než druhé.

Teď, když jsme si představili programovací prostředí a typy chyb, které budou pozorovány, můžeme se přesunout na druhou část práce – část empirickou.

3 Pozorování chyb

V této kapitole se podíváme na podmínky a průběh pozorování žáků programujících v prostředí Scratch i na podmínky výuky samotné. Tyto údaje pomohou nejen k lepší představě o vzorku žáků, které jsem pozorovala, ale také jakým způsobem a v jakém prostředí probíhalo samotné pozorování.

Považuji za podstatné čtenáře upozornit, že všechny pozorované chyby jsou vázány na **konkrétní zadání** a pro řešení jiného problému mohou být správným řešením. Nebudu uvádět u každé chyby i zadání úlohy, na závěr této kapitoly ovšem uvedu ilustrativní zadání. To by mělo stačit na to, aby si čtenář udělal představu o podstatě zadání, která žáci dostávali. Zároveň u zmiňovaných chyb budu uvádět, z jakého důvodu se jedná o chybu.

Dále bych ráda upozornila, že v žádném případě **nebylo možné objevit všechny chyby**, které žáci během kurzu udělali. V několika případech probíhala skupinová práce, kde se žáci společně podíleli na skupinovém projektu. V takovém případě je téměř jisté, že chybu mohl najít někdo ze spolužáků. Teprve následně, v případě neúspěchu ve skupině, jsem měla šanci objevit chybu při pokusu žáka o odevzdání projektu s chybou.

3.1 Průběh pozorování

Pozorování probíhalo vlastním zúčastněným pozorováním žáků (sdílených obrazovek) a při žádosti žáka o radu. Další chyby byly zkoumány z opravování úloh žáků. Chyby jsem buď ihned vyfotila nebo jinak zaznamenala a následně zrekonstruovala.

Pozorování chyb probíhalo u vzorku žáků ve věku **7 – 12 let** v celkovém množství **29 žáků** při zájmové výuce programování v období **od 20. 09. 2021 do 27. 03. 2022**. Ne všichni žáci docházeli do výuky po celou dobu pozorování, jelikož byly z celkem 4 skupin, které začaly, a tedy i skončily, různě. Celkem proběhlo **49 řádných lekcí** a **10 náhradních/soukromých lekcí**.

Součástí pozorování byly i dotazy na vztah žáka k chybám doplněné o vlastní pozorování reakcí žáka. Toto bylo provozováno především během rozsáhlejších projektů a méně u projektů procvičujících novou látku. Mezi takové dotazy patřily zejména otázky:

- „Baví tě složitější úkoly, nebo pak nechceš pracovat?“,
- „Když uděláš chybu, chceš si ji opravit, nebo jen nechceš, aby tam byla?“,
- „Když ti poradím, chceš vysvětlit proč to tak je, nebo ti stačí správné řešení?“.

Správná řešení byla vysvětlována i těm žákům, kteří na poslední otázku odpovídali, že jim stačí správné řešení. Otázky byly v průběhu pozorování přeformulovávány podle potřeb a chápání žáků. Odpovědi žáků jsem si zapisovala pro budoucí použití.

3.2 Podmínky výuky a její cíle

Nejprve se pokusím nastínit technické podmínky výuky. Následuje stručný popis žáků, jejich motivace a obecně přístupu k výuce. Poté uvedu obecné informace o časové organizaci kurzu a hodin. Dále se pokusím převést cíle výuky na didaktické cíle podle RVP ZV[9] a na závěr uvedu typický příklad zadání, která žáci dostávali.

3.2.1 Technické podmínky výuky

Veškerá výuka probíhala online v soukromém prostředí firmy Algorithmics. To má dvě relevantní funkce pro žáky a dvě pro učitele. Žák po přihlášení vidí svůj kurz (v tomto případě programování ve Scratchi), kde má **všechny úlohy**, které už byly probrány – může se tedy jednoduše vracet ke starším úlohám. Druhá část je společná pro žáka i učitele a tou je připojení do **online hodiny**. Podobně jako například Google Meet má následující základní funkce: sdílení obrazu z kamery či obrazovky a zvuku z mikrofonu, chat se všemi nebo jen s učitelem a možnost přihlásit se o slovo. Učitel kromě místnosti vidí v kurzu **úlohy všech svých žáků** v dané skupině, může tak jednoduše kontrolovat, zda žák úlohu alespoň začal (v případě automaticky opravovaných úloh rovnou zda je úloha správně, či nikoliv) a po jejím rozkliknutí může úlohu zkontrolovat a označit, které úkoly v ní jsou dobře a které špatně. Žák s učitelem může u jednotlivých úloh komunikovat pomocí komentářů.

3.2.2 Popis typického žáka, respektive pozorovaného vzorku žáků

V práci se o žácích mluví v mužském rodě. V pozorovaném vzorku se nacházelo i několik dívek, ovšem vzhledem k poměru množství dívek a chlapců (5:24) nevidím smysl je jakkoli rozdělovat. Naopak, nedostačující vzorek by mohl vést k vyvozování nepodložených či dokonce zavádějících závěrů při pokusu o jejich rozdělování.

Jelikož se jednalo o pozorování při zájmové výuce, naprostá většina žáků měla vysokou motivaci k učení. Obvykle se jednalo o mladé nadšence, někteří se následně rozhodli pokračovat ve vzdělávání se podobným směrem a pokračovali kurzem Python nebo Game design.

Většina žáků projevovala vlastní iniciativu a zájem o učivo během hodiny, ovšem méně než polovina se sama od sebe vracela k programování mimo přímou výuku.

Více než polovina žáků reagovala kladně na své chyby – k chybám se stavěli pozitivně a obvykle je přijímali bez znaků ostychu či studu. Ještě kladnější přístup měli k chybám, které udělali jejich spolužáci – navzájem se povzbuzovali k lepším výkonům a podporovali tak růst svůj i svých spolužáků. Více o jejich vztahu a postoji k chybám se dozvíte v podkapitole 4.2 Postoj žáků k chybě.

9 RVP ZV z roku 2021, ve kterém došlo k aktualizaci didaktických cílů pro vzdělávací oblast Informatika.

3.2.3 Časová organizace výuky

Kurz byl členěn do modulů podle témat, která se na sebe nabalovala. Moduly byly následně členěny na jednotlivé hodiny. Hodiny každé skupiny byly jednou týdně po dobu 90 minut. Obvyklý průběh hodiny:

Čas v minutách	Část hodiny	Popis části hodiny
5–10	Začátek	Povídání si, sbližování se s kolektivem, opakování, plán hodiny
5–10	Nové téma	Představení nového tématu, ukázky, teorie
5	Popis a rozebrání úlohy	Diskuze o tom, co budeme muset vymyslet, co na to bude potřeba
20	Samostatná práce	Žáci samostatně pracují na zadaných úlohách, učitel průběžně odpovídá na dotazy
5–10	Pauza	Fyzická aktivita
20–25	Pokračování samostatné práce	Pokračování v práci na úloze, případně popis a řešení další úlohy na podobné téma
5–10	Ukončení hodiny	Opakování, povídání si, téma další hodiny

Žáci obvykle nedostávali žádné domácí úkoly. Když ano, tak se jednalo o dokončení rozpracovaných úloh nebo splnění úloh Otestuj se – ty sloužily k testování znalostí žáka na zrovna probírané téma. Žáci mimo hodiny měli také možnost kontaktovat svého lektora a požádat buď o konkrétní radu, nebo o soukromou lekci mimo obvyklou dobu. Stejnou možnost měli v případě zameškání některé lekce – v takovém případě to obvykle domlouvali rodiče žáka. Žáci nebyli během kurzu hodnoceni známkou, pouze slovně a to zejména splněno/nesplněno a případným konstruktivním komentářem k plnění úlohy či pochválením způsobu plnění.

3.2.4 Didaktické cíle výuky

Následují cíle výuky přeformulovány podle RVP ZV. Jelikož se jednalo o kurz vizuálního programování, jedná se o cíle z části Algoritmizace a programování. Kurz z nich alespoň částečně naplňuje všechny očekávané výstupy na úrovni pro 1. stupeň ZŠ, jmenovitě:

a) čtení a porozumění algoritmu,
b) algoritmus pro řešení problému,
c) optimalizace algoritmu,
d) testování algoritmu a programu,
e) tvorba programu,

a některé i na úrovni pro 2. stupeň ZŠ, konkrétně a), b) a e).

Očekávané výstupy těchto kompetencí, které by průměrný žák na konci kurzu měl splňovat jsou následující:

- *žák sestavuje a testuje symbolické zápisy postupů*
- *žák po přečtení jednotlivých kroků programu vysvětlí celý postup*
- *žák popíše jednoduchý problém, navrhne a popíše jednotlivé kroky jeho řešení*
- *žák rozdělí problém na jednotlivě řešitelné části a navrhne a popíše jednotlivé kroky k jejich řešení*
- *žák ověří správnost jím navrženého postupu či programu, najde a opraví v něm případnou chybu*
- *žák v blokově orientovaném programovacím jazyce sestaví program; rozpozná opakující se vzory, používá opakování a připravené podprogramy*
- *žák používá opakování, větvení programu a proměnné* [9]

Můžete si povšimnout, že ne všechny zmíněné očekávané výstupy jsou kompletní. Snažila jsem se vybrat pouze ty části, které většina žáků reálně zvládala, respektive které na konci kurzu vykazovala.

Vzhledem ke své podstatě kurz rozvíjel také digitální kompetence žáků. Konkrétně bych vyzdvihla následující:

- *žák ovládá a využívá určené výukové aplikace při svém učení*
- *žák komunikuje se svými blízkými a učiteli pomocí zadané digitální technologie*
- *žák zachycuje skutečnosti ze svého okolí a vyjadřuje své představy i za pomoci digitálních technologií*
- *žák vytváří jednoduchý digitální obsah (texty, tabulky, obrázky, audio, video), vyjadřuje se za pomoci digitálních prostředků ke splnění stanovených cílů*
- *žák provádí základní změny obsahu, který vytvořil někdo jiný s cílem přizpůsobit ho novým účelům* [9]

3.3 Typický příklad zadání

Jedná se o typický způsob zadání, které žáci dostávali, konkretizovaný příklady na jednom mírně upraveném autentickém zadání. Žáci obvykle na začátku mohli vidět **cílový výstup**, který jim měl pomoci pochopit, jak by jejich projekt měl zhruba fungovat. Následovalo slovní zadání, které v jedné či dvou větách shrnovalo hlavní body. Například: „Naprogramuj sněžení“.

Jednotlivé úkoly poté vedly žáky při tvorbě. Například: „Naprogramuj klonování postavy vločka“, „Přidej rozšíření pero“, „Vločka dopadne a otiskne se“, „Sněží neustále“. Při úlohách zamě-

řených více na kreativitu byly úkoly více obecné nebo zaměřené spíše na počty postav nebo levelů či prvky, které se musí v projektu vyskytnout.

Abych tuto kapitolu shrnula, pozorováno bylo 29 žáků po dobu 6 měsíců v různých fázích kurzu. Výuka jednotlivých skupin probíhala online jednou týdně s lekcemi dlouhými 90 minut. Didaktické cíle se zaměřovaly především na Algoritmizaci a programování a z digitálních kompetencí byly rozvíjeny především ty z částí Informace a komunikace a Tvorba a vyjádření.

Následující kapitola se bude věnovat samotným výsledkům pozorování. Přiblížím v ní chyby, kterých se žáci dopouštěli, a porovnám rozdělení chyb z teoretické části práce se skutečně pozorovanými chybami.

4 Výsledky pozorování

Teď, když už jsme se dozvěděli teorii i praktické informace o žácích, pozorování i výuce, můžeme se podívat na samotné výsledky pozorování.

Je důležité uvědomit si, že případná pochybení v procesu tvorby programu v tomto pozorování nebyla chápána jako chyby a nebyla tedy zaznamenávána. Tyto – v některých případech pouze dočasné – nedostatky jsou součástí procesu tvoření a je jedině dobře, když si na ně žáci zvyknou již v průběhu učení.

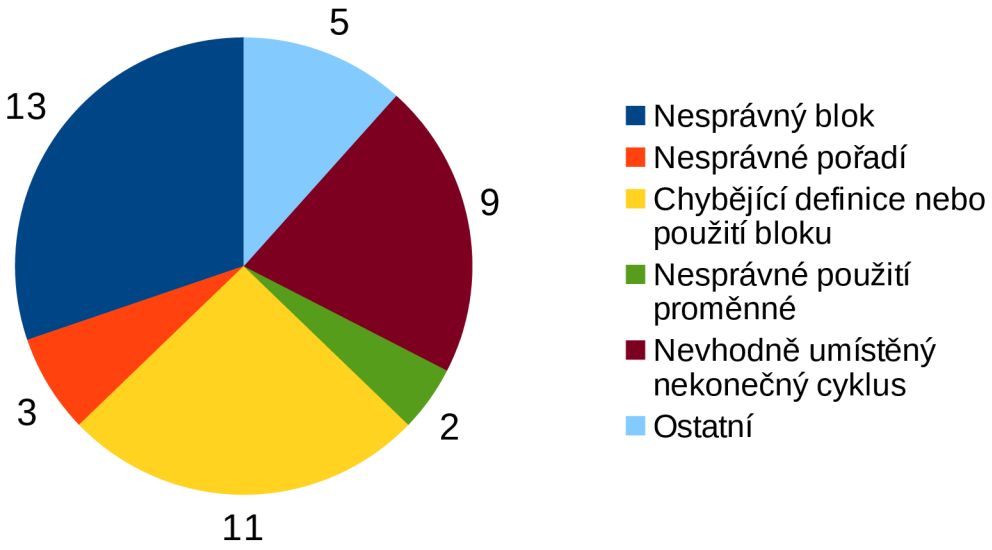
4.1 Typy chyb, jejich výskyt a příklady

V této podkapitole se dozvíte více o jednotlivých skupinách pozorovaných chyb včetně konkrétních příkladů a jejich četnosti. Tato podkapitola je členěna na dvě části. V první se věnuji funkčním chybám, ve druhé potom odchylkám od kanonického postupu.

4.1.1 Chyby funkční

Jak bylo uvedeno v kapitole 2.1, funkční chyby zabraňují správnému chodu žákovského programu. Těchto chyb si žáci obvykle všímali, především tedy jejich následků. Právě vizuální či akustické projevy těchto chyb napomáhají k jejich odhalení. Ovšem ne vždy tyto stopy vedou k chybě přímo, v některých případech totiž i jediná chyba může spustit lavinu chyb, které na sebe více či méně očividně navazují. U těchto chyb není reálné očekávat, že by je Scratch ani jiný programovací jazyk sám odhalil (viz kapitola 2.2, Obrázek 19: Nesprávně umístěný nekonečný cyklus).

Ovšem některé chyby (viz kapitola 2.2, Obrázek 16: Boolean hodnota v rolovací nabídce) Scratch odhalí a v jiném programovacím jazyce by byly i nahlášeny. Například při spouštění programu psaném v jazyce Python by tato chyba byla dohledatelná díky chybové hlášce, kterou pokus o spuštění vrátí. V tomto případě by se jednalo pravděpodobně o nesprávný datový typ. Scratch ovšem chybu uživateli neoznámí, sám se s ní vypořádá a to obvykle přeskočením dané části kódu. Podobný případ, kdy nastane chyba, kterou Scratch odhalí, sám se s ní vypořádá, ale uživateli tuto skutečnost nikterak neoznámí, je překročení limitu počtu aktivních klonů.



Graf 1: Funkční chyby

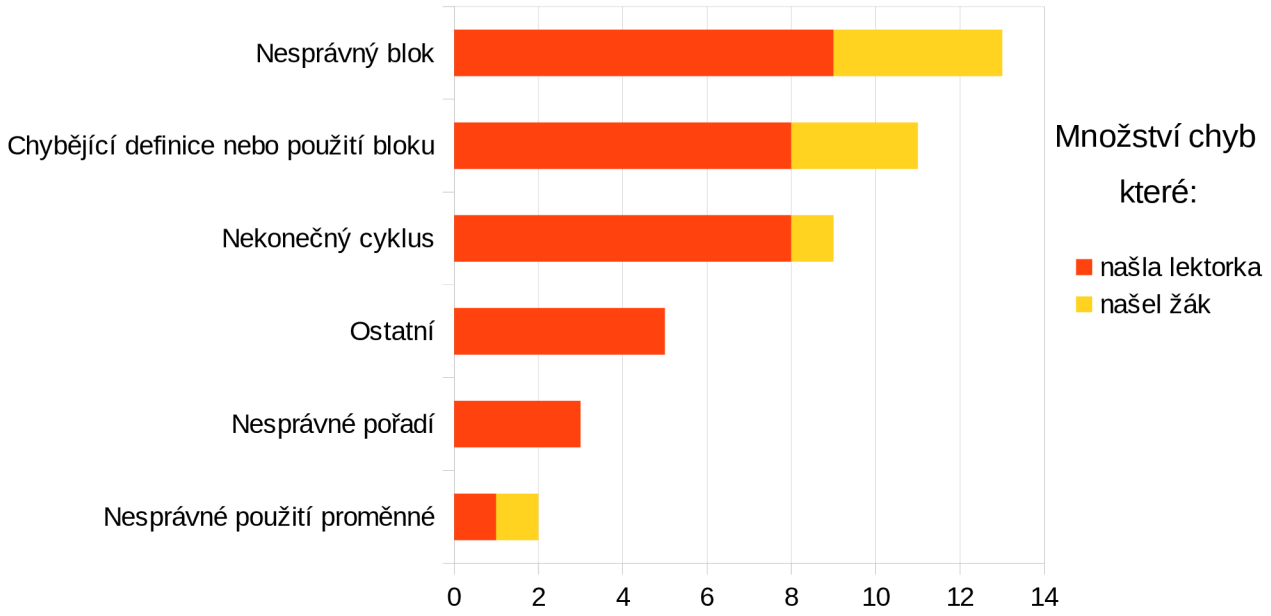
Z uvedeného grafu (viz Graf 1) můžete vyčíst, že nejčastější pozorovanou funkční chybou byl **nesprávný blok**, konkrétně v množství 13 výskytů. **Chybějící definice nebo použití bloku** bylo druhou nejčastější chybou, žáci se jí dopustili 11krát. **Nesprávné pořadí** se vyskytlo 3krát a **nesprávné použití proměnné** 2krát.

V grafu jsou navíc dvě kategorie, Nevhodně umístěný nekonečný cyklus a Ostatní. **Nevhodně umístěný nekonečný cyklus** jsem se rozhodla oddělit, protože plně nezapadal do žádné z kategorií. Šlo by jej zařadit do kategorie Blok(y) navíc, která je v následující podkapitole 4.1.2 Odchylky od kanonického postupu. To by však mohlo být matoucí, jelikož v tomto případě se rozhodně jedná o chybu funkční, jelikož zabraňuje správné funkci programu. Zároveň částečně zapadá do kategorie Nesprávné pořadí, ani zde ovšem není zcela shoda, neboť Nesprávné pořadí evokuje pocit toho, že stačí dát chybný blok na správné místo, zde ovšem tento blok neměl být vůbec.

Kategorie **Ostatní** zahrnuje zbývající chyby, které nebyly v takovém počtu, abych jim vytvořila vlastní kategorii, ale zároveň nezapadaly do žádných již stanovených kategorií.

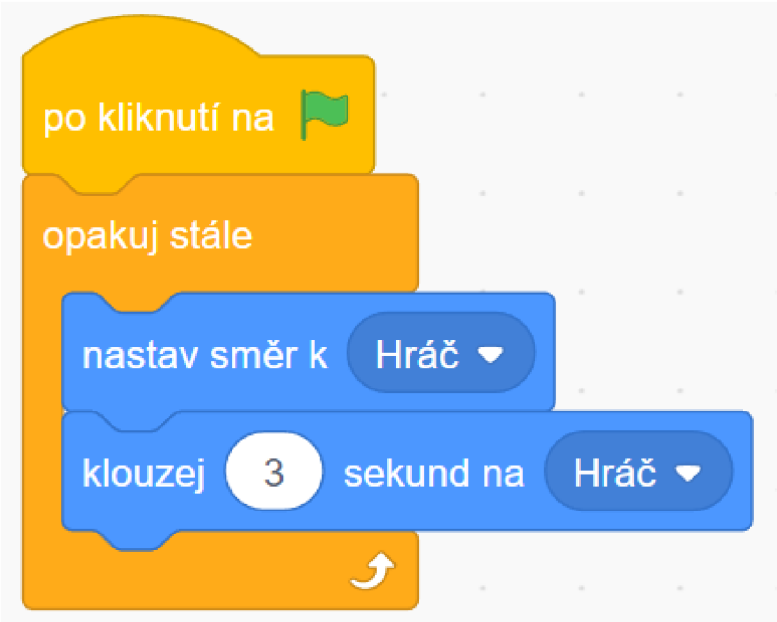
S ohledem na upozornění ze začátku této kapitoly uvádím i Graf 2 znázorňující, v kolika případech na chybu přišel sám žák (respektive kdy sám buď během mého hledání chyby nebo při drobné nápovědě v podobě dobře mířené otázky chybu objevil) a v kolika jsem na ni musela upozornit přímo.

Jak již bylo naznačeno v úvodu této kapitoly, všechny pozorované chyby jsem odhalila já, tento graf ukazuje pouze, zda při upozornění žáka na to, že program není správně, žák chybu sám našel či nikoli.

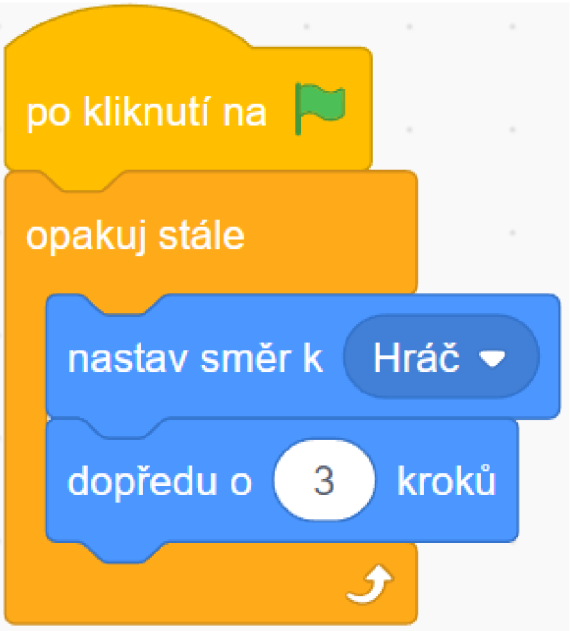


Graf 2: Funkční chyby 2

Pod označením **nesprávný blok** se skrývá několik typů chyby. Někdy se jednalo o použití nesprávného bloku z důvodu miskonceptu.



Obrázek 20: Funkční chyba 1

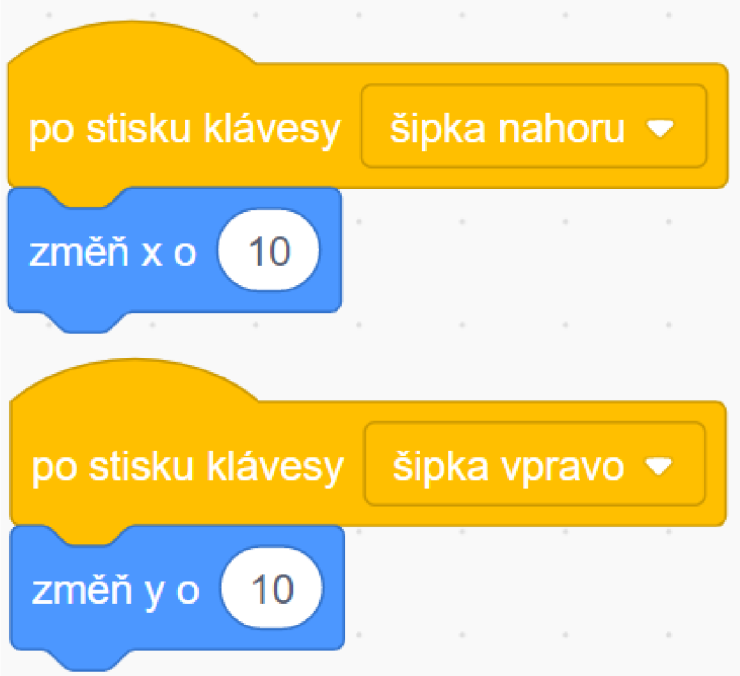


Obrázek 21: Funkční chyba 1 (řešení)

Zde se jedná o chybu pouze v konkrétní situaci, respektive záleží, co žák (případně učitel) chce, aby se při běhu programu dělo. Rozdíl je ten, že zatímco „chybný“ kód (viz Obrázek 20) nastaví současné souřadnice postavy Hráč jako cíl a na ten následně 3 sekundy klouže, ať už se Hráč pohne nebo ne, správné řešení (viz Obrázek 21) zajistí, že se postava téměř ihned otáčí a následuje Hráče.

Žák v tomto případě věřil, že blok „klouzej“ vytváří plynulý pohyb na místo, „kam ho pošle“. Nezamyslel se ovšem nad otázkou, zda se jedná skutečně o klouzání na postavu jako takovou, nebo na cíl v podobě aktuálních souřadnic postavy.

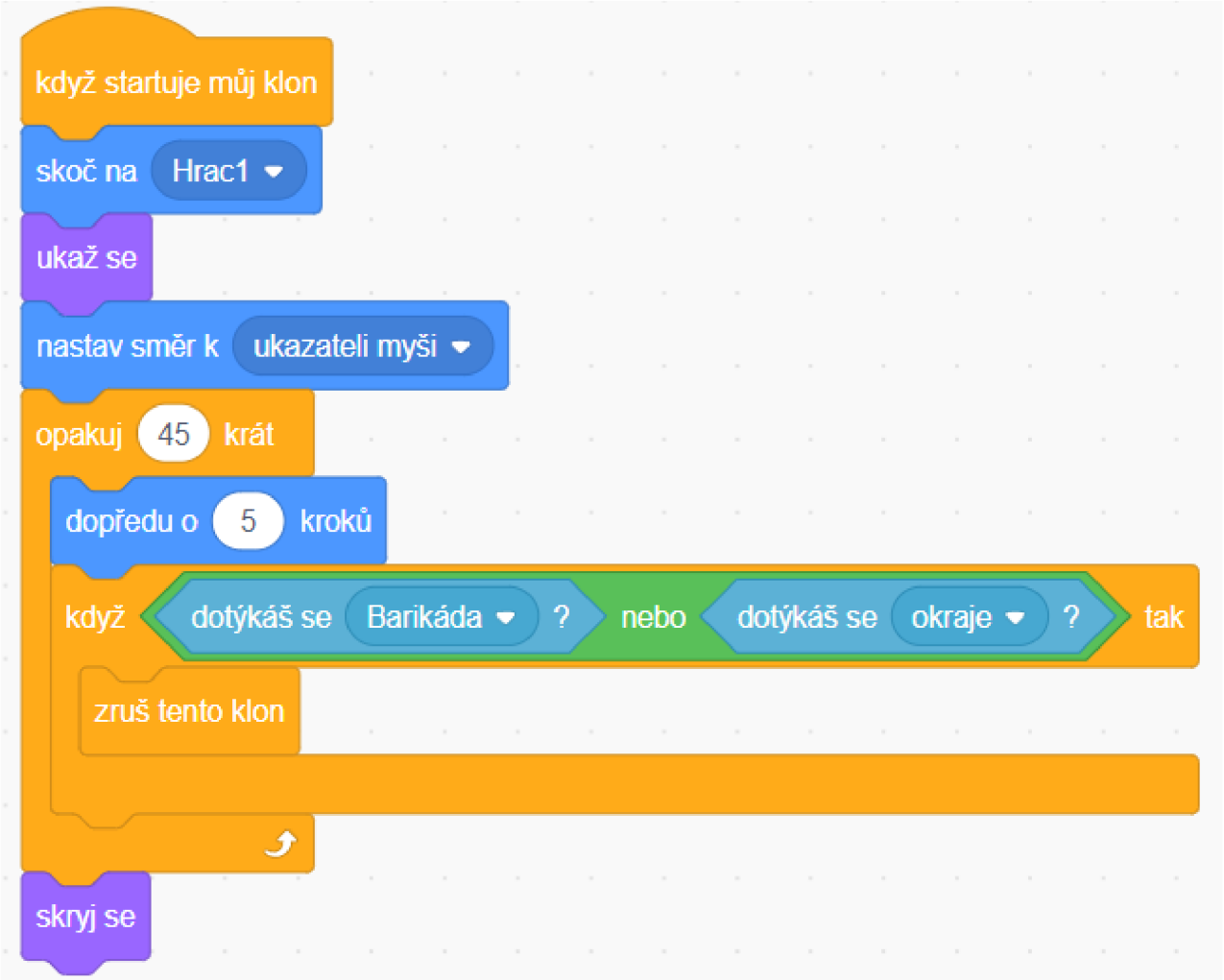
Jindy se jednalo spíše o nepozornost či mechanickou záměnu dvou bloků, nikoli funkční.



Obrázek 22: Funkční chyba 2

V tomto případě (viz Obrázek 22) samozřejmě došlo k záměně os. To byl také nejčastější případ záměny dvou bloků. Zda se jednalo o pouhou záměnu nebo o neznalost není jisté, neboť při dotazu, která osa je směr nahoru a dolů a která je doprava a doleva, dotazovaní žáci odpovídali správně. Ovšem ve chvíli, kdy zjistili, že tak, jak to mají naprogramované, nefunguje program tak, jak chtěli, zbývá pouze druhá varianta, která osa je která. Nelze tedy vyvozovat, zda došlo k záměně os jako takových, nebo pouze bloků. To, že doleva a dolů jsou záporné hodnoty a naopak směry nahoru a doprava používají kladné hodnoty ovšem obvykle nepletli.

Druhou skupinou byly chyby v podobě **chybějícího bloku kódu nebo jeho definice**. Jedním takovým příkladem může být chybějící **rušení klonu** v následující ukázce (viz Obrázek 23). Povšimněte si faktu, že klon se ruší pouze pokud se dotkne postavy Barikáda nebo okraje. Dochází tak potenciálně k nezrušení klonu. Plátno projektu má rozměry přibližně 480 × 360, 45 × 5 je 225. Je tedy možné, že se nedotkne ani okraje ani postavy Barikáda, v takovém případě může po čase dojít k dosažení maxima (300 pro Scratch 3.0) aktivních klonů (jejich skrytí je nedeaktivuje) a tím k zastavení tvorby dalších klonů.

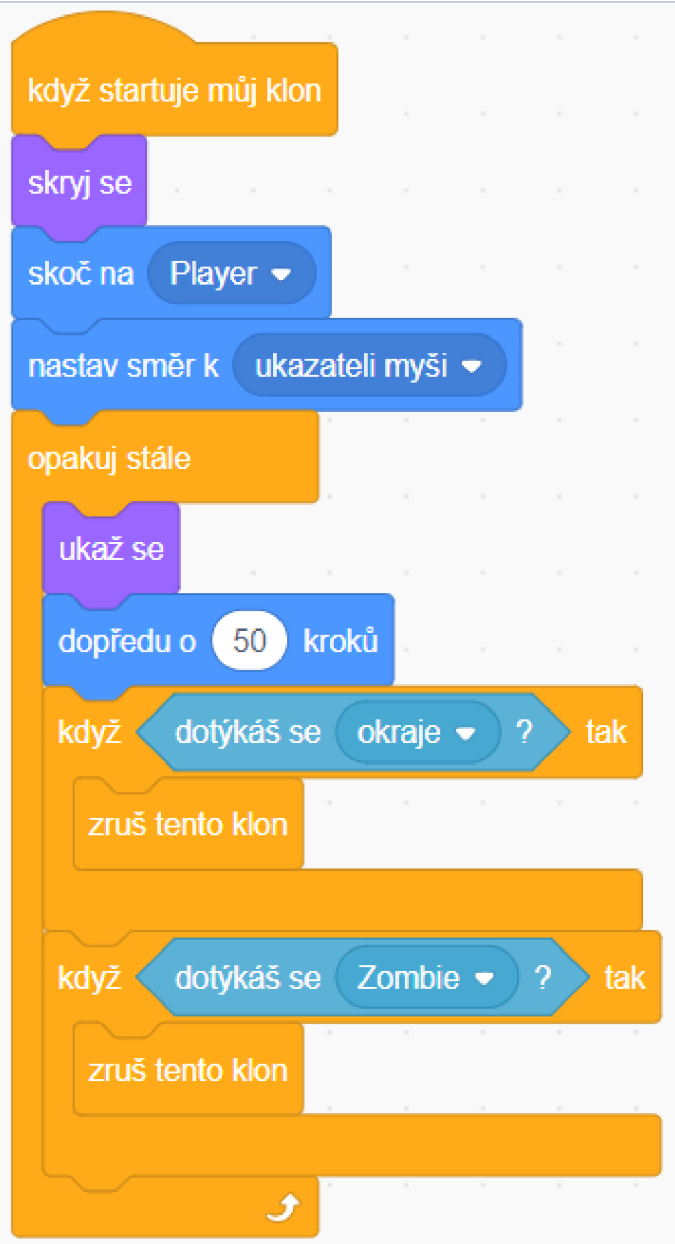


Obrázek 23: Funkční chyba 3

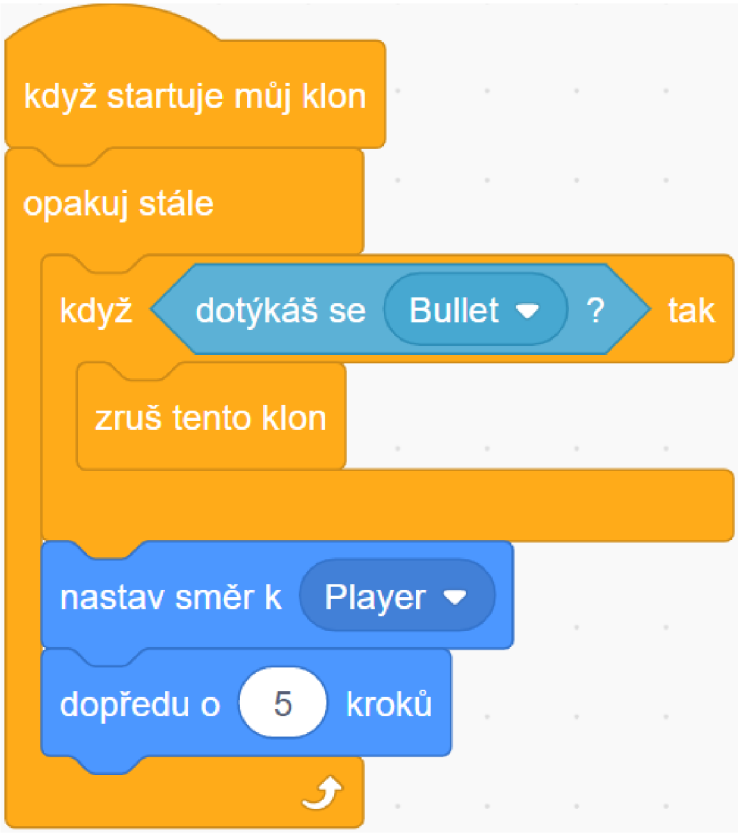
I další ukázka (viz Obrázek 24) je na první pohled v pořádku, ovšem při spuštění program nedělá zcela to, co by uživatel očekával. Součástí tohoto programu je totiž i následná část kódu (viz Obrázek 25), která patří postavě Zombie. Postava Bullet má dotykem s postavou Zombie zrušit svůj klon a zároveň klon Zombie. To se ovšem nestane, jelikož klon Bullet stihne zmizet dřív, než klon Zombie, který zůstane aktivní. Mé teorie k této zvláštní funkčnosti byly a) postava Zombie je ve chvíli dotyku v části „nastav směr k __“ a „dopředu o __“ a než se dostane k podmínce, klon Bullet se zruší. Při otestování (vyjmutí této části nebo použití jiné konstrukce – „Opakuj dokud nenastane __“) se ovšem ukázalo, že ani tak není klon Zombie zrušen. Druhá teorie, b), se tedy zdá pravděpodobnější. Jelikož kódy neprobíhají simultánně, v momentě, ve kterém je klon zrušen, nestihne ke stejné kontrole dotyku dojít i u druhé postavy. Řešení v tomto případě bylo přidání bloku „čekej __ sekund“ před blok „Zruš tento klon“. Zvláštní ovšem je, že stačí samotný blok „Čekej“ (lze mu zadat hodnotu 0 nebo dokonce žádnou).

Tyto kódy (viz Obrázek 24 a 25) jsou ukázkou také odchylek od kanonického postupu – **nevhodné konstrukce**. Pro tento případ existuje vhodnější, respektive přehlednější, konstrukce –

„Opakuj dokud nenastane __“. Také jsou v Obrázek 24 **nevhodně umístěny bloky** „skryj se“ a „ukaž se“ – původní postava byla již skryta, skryj se je tedy duplicitní blok, ukaž se je poté zbytečně v opakování, stačilo by jej použít pouze jednou, před opakováním.

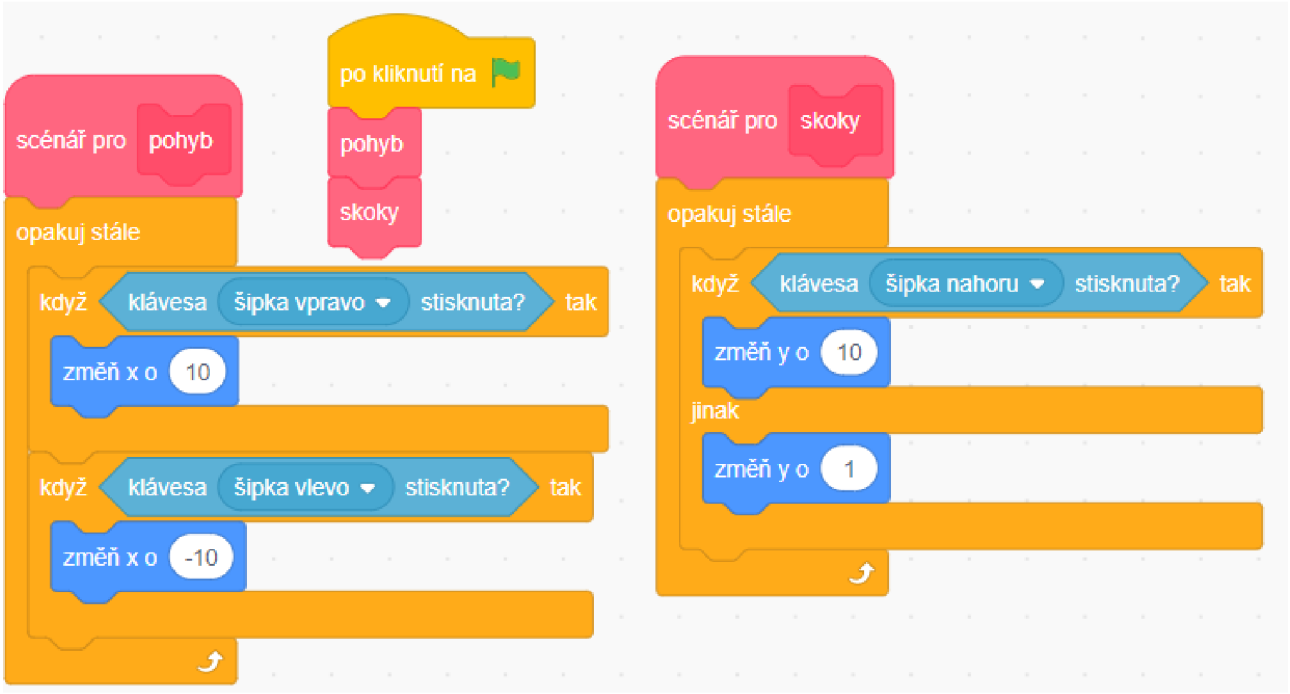


Obrázek 24: Funkční chyba 4.1



Obrázek 25: Funkční chyba 4.2

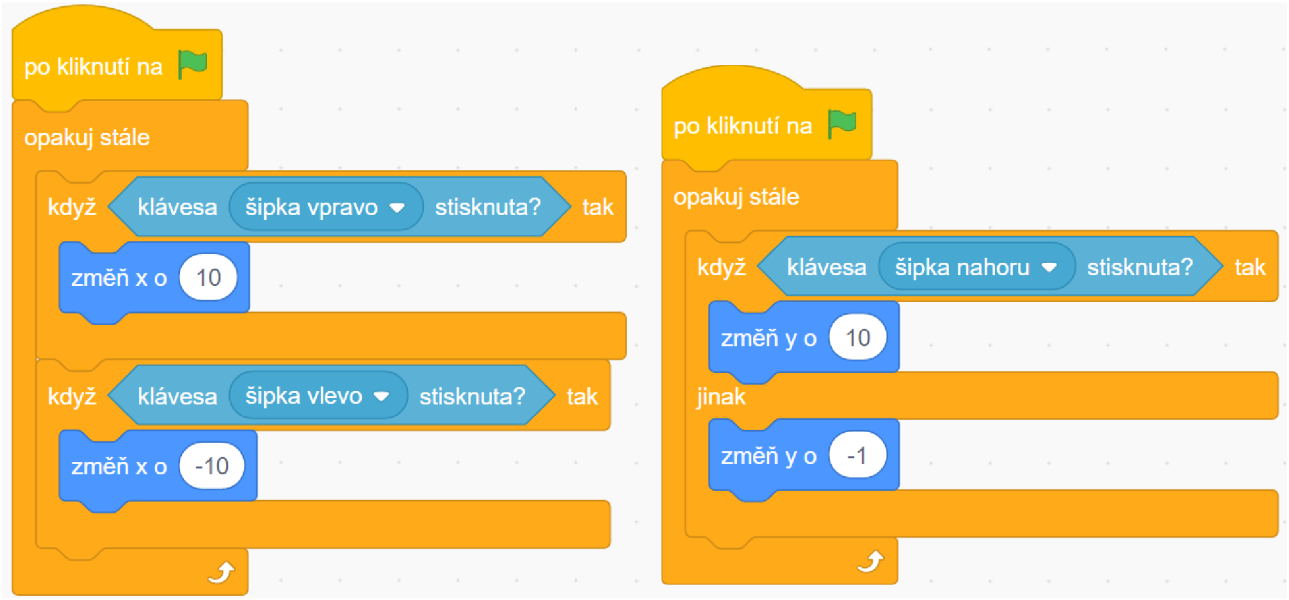
Další poměrně častou chybou byl nevhodně umístěný nekonečný cyklus. Jednalo se o jeho umístění ve scénáři pro Můj blok. Tuto chybu už jsme si představili v kapitole 2 (viz Obrázek 19), další ukázkou této chyby je velmi podobný případ (viz Obrázek 26).



Obrázek 26: Funkční chyba 5

Pro rychlé připomenutí, tento program se nedostane ke spuštění scénáře pro skoky, neboť se zasekne v nekonečné smyčce, která je ve scénáři pro pohyb.

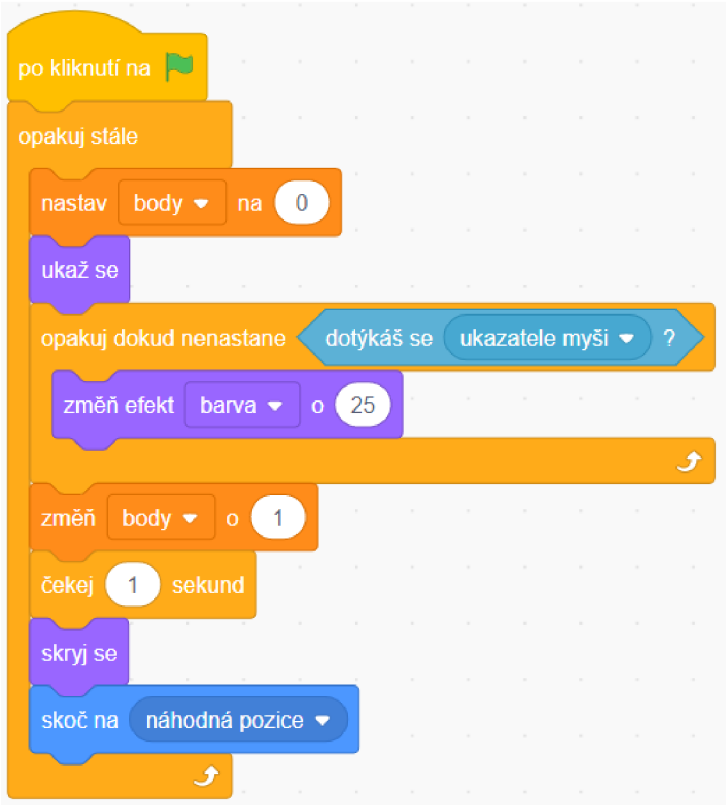
V případě těchto chyb se obvykle jednalo o pokus napodobení již zaběhnutého způsobu programování ovládání pomocí „opakuj stále“ a vnořených jednotlivých „když __ tak“ pro každou klávesu a naprogramování chování postavy ve chvílích, kdy je daná klávesa stisknuta (viz Obrázek 27).



Obrázek 27: Funkční chyba 5 (obvyklý způsob řešení žáky)

Čtvrtou kategorií bylo **nesprávné pořadí**. Tam se jednalo o špatně umístěné bloky nastavení hodnoty proměnné. Tento typ chyby jsem nezařadila pod kategorii nesprávné zacházení s proměnnou z toho důvodu, že tato chyba není důsledkem neznalosti práce s proměnnou, ale opravdu chybného pořadí.

Zde měl být blok „nastav [body] na 0“ nad opakováním, aby proběhl pouze jednou, při spuštění této sekvence kódu.



Obrázek 28: Funkční chyba 6

Obrázek 29: Funkční chyba 7

Překvapivě málo chyb žáci dělali při **používání proměnných**, ačkoliv se podle Swidana proměnných týká jeden z nejčastějších miskonceptů [7]. Je pravdou, že někteří žáci měli s proměnnými ze začátku velké obtíže, a proto se jim snažili za každou cenu vyhnout. To, společně s faktem, že když už proměnné využívali, bylo to nejčastěji pro záznam bodů, bude pravděpodobně důvod, proč v nich příliš nechybovali. Nelze tedy přímo tvrdit, že by obecně žáci neměli problémy s fungováním proměnných, spíš že jsou často schopni upravit program i svá očekávání od programu tak, aby jich nemuseli využít. Ve chvílích, kdy se nejednalo o aktivitu na procvičování proměnných, jsem žáky nechávala vypořádat se se situací, jak uznali za vhodné. Vždy jsem jim ovšem připomněla bloky proměnných, obvykle však bez kladné odezvy v podobě jejich použití.

Obrázek 35 je jasným příkladem špatného zacházení s proměnnou. Žák postupoval tak, jako kdyby diktoval pokyny člověku. Nejprve mu tedy řekl, že si bude muset uložit následující odpověď, a poté se zeptal na otázku. Bloky „otázka“ a „odpověď“ ovšem fungují přesně naopak, nejdřív

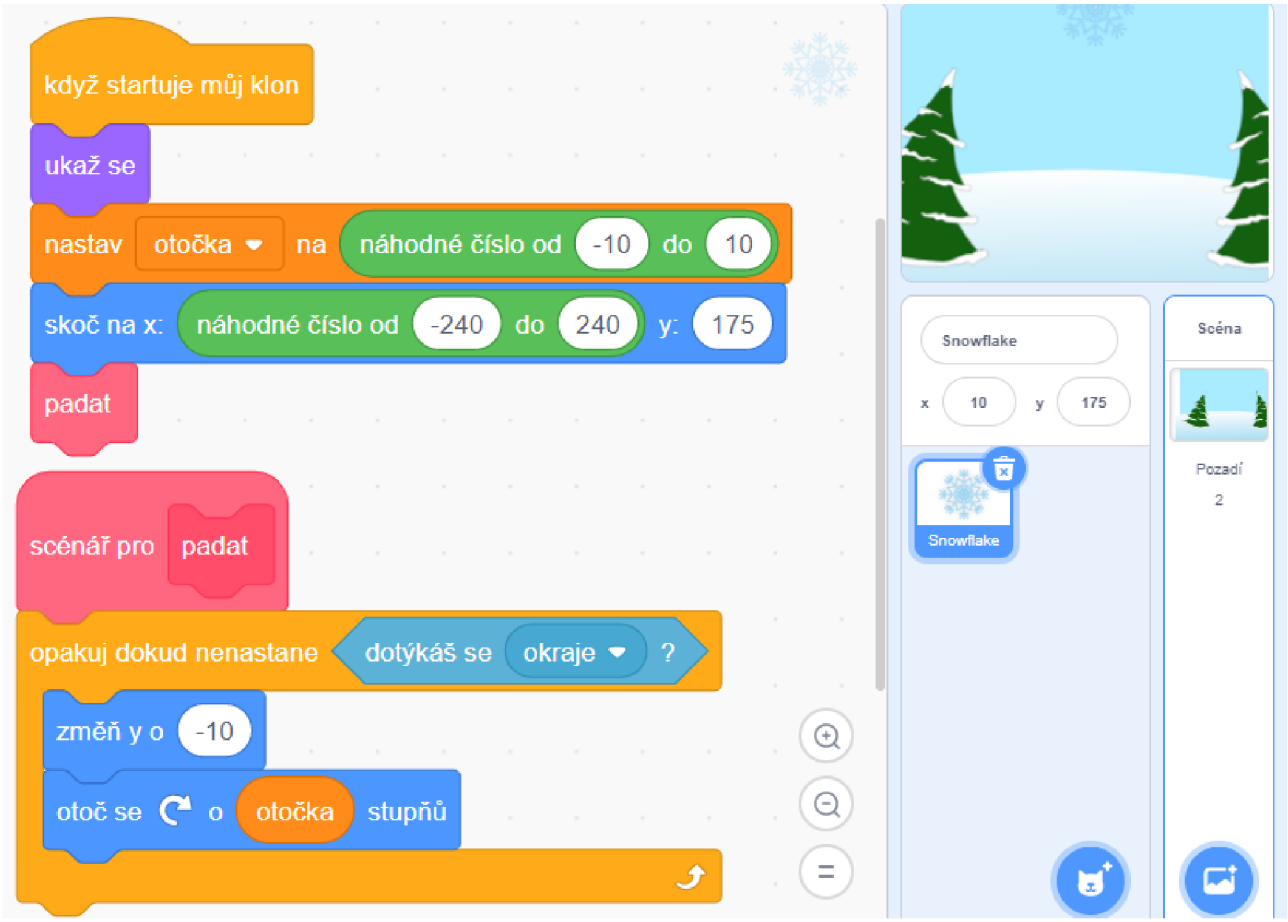
položíme otázku a až poté můžeme pracovat s odpovědí. Žák si zde dokonce při dalším spouštění projektu myslel, že vše funguje tak, jak má, jelikož při opakovaném spouštění projektu se hodnoty uložené v proměnných, a tedy ani v odpovědi, neresetují. Při druhém spuštění se tedy do proměnné *meno* uložila odpověď z předešlého průběhu programu.

Zbylé chyby (kategorie Ostatní) byly chyby v podmínkách a pak velmi specifické chyby.

Obrázek 30 ukazuje nevhodně zvolenou podmínku. Postava vločky totiž nikdy nezačne padat, neboť se už od začátku kvůli svému umístění dotýká vždy alespoň jednoho z okrajů. V tomto případě bylo tedy lepší zvolit podmínku, která se týkala umístění postavy, respektive jejího klonu, na ose y.

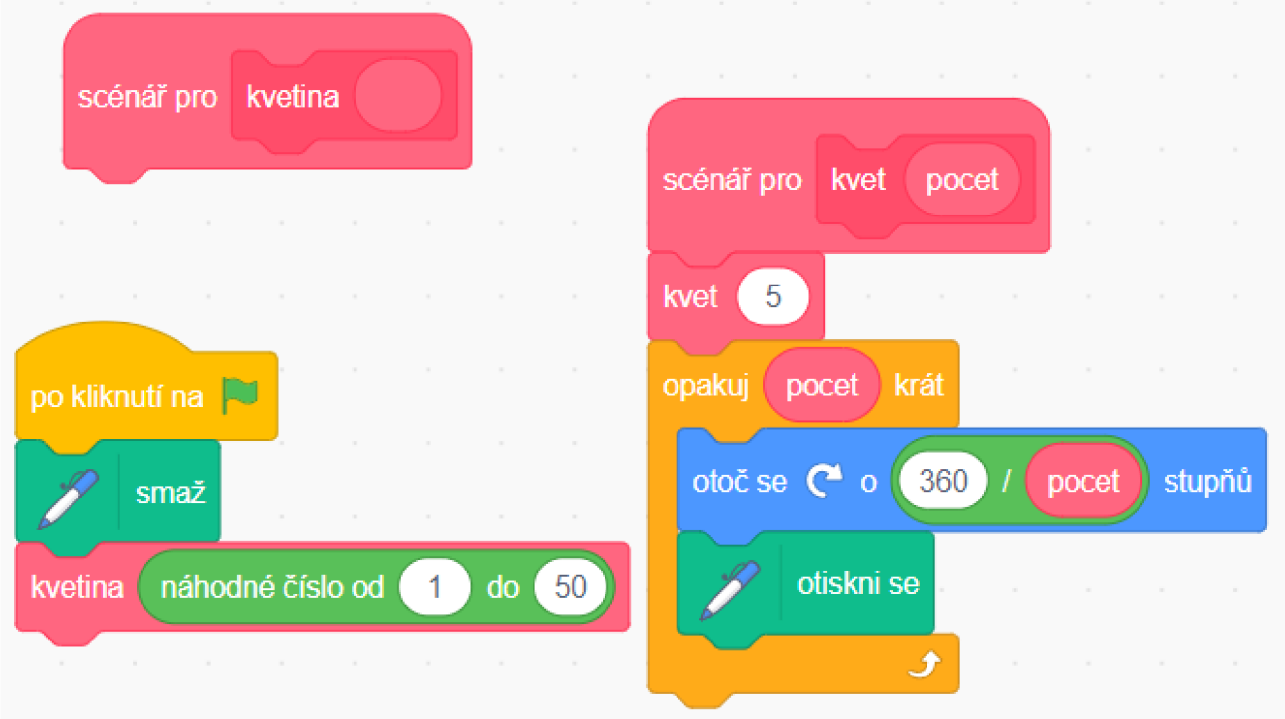
Další z chyb v této kategorii bylo nesprávné zacházení s bloky skupiny Můj blok (viz Obrázek 31). Nejen že zde žák pracoval se dvěma různými scénáři, ale ještě používal blok, který vyvolává spouštění scénáře v daném scénáři.

Nejspecifičtější chybou bylo vložení bloku vnímání do místa pro výběr barvy v dalším bloku vnímání (viz Obrázek 32[10]). Tato chyba mohla vzniknout kvůli možnosti vkládání boolean bloků do všech mezer pro parametr zmiňované v kapitole 1.2.

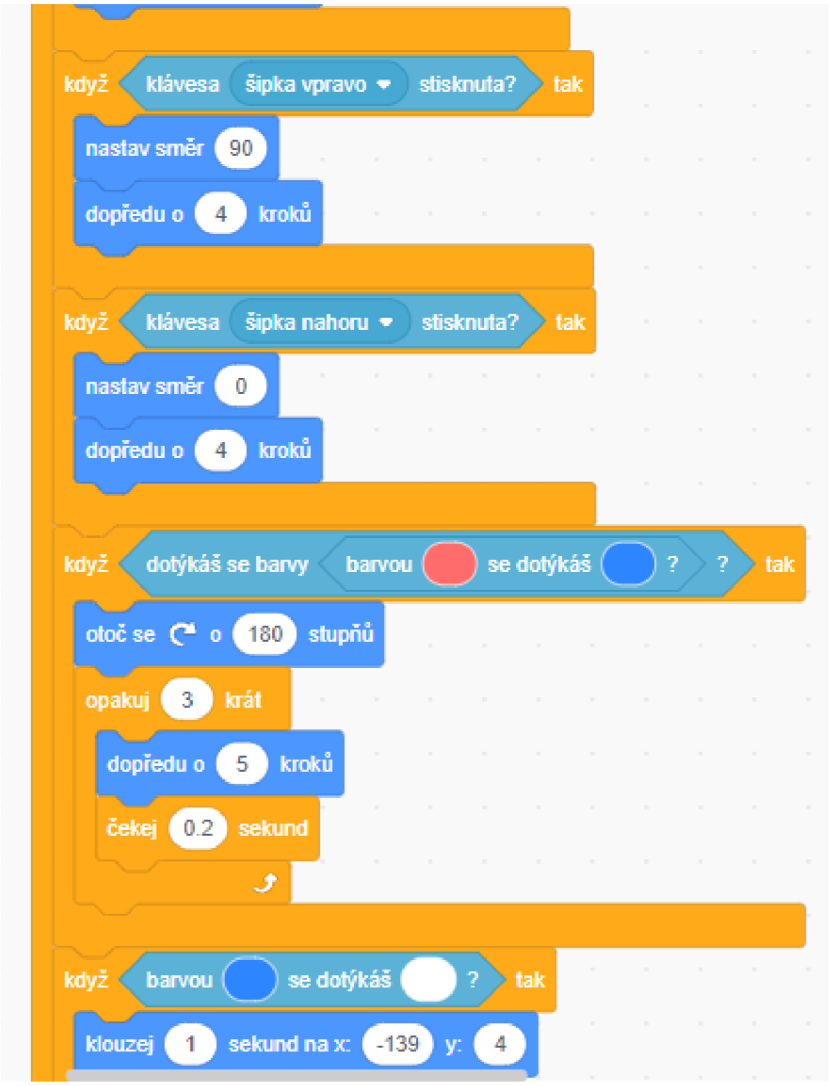


Obrázek 30: Funkční chyba 8

10 Kód zde není celý, vzhledem k jeho reálné délce by to bylo nejen velmi obtížné, ale také nepřehledné.



Obrázek 31: Funkční chyba 9



Obrázek 32: Funkční chyba 10

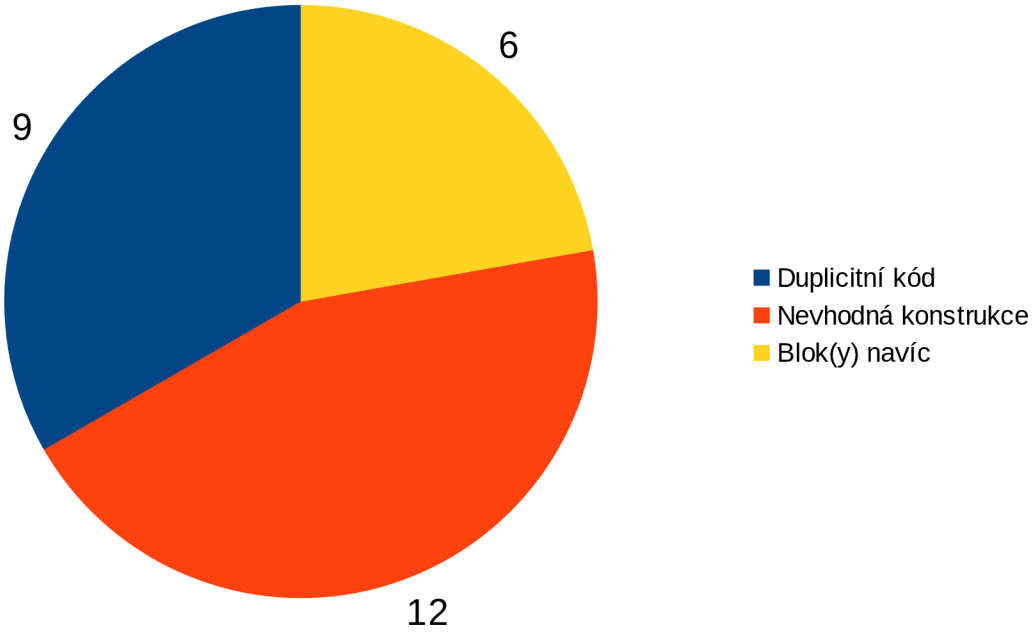
4.1.2 Odchylky od kanonického postupu

Na rozdíl od chyb funkčních, tyto odchylky jsou do jisté míry subjektivní. I tak se na některých shodne většina. Oproti kapitole 2 zde ve výčtu k **Duplicitnímu kódu** a **Nevhodné konstrukci** přibyla další kategorie a tou je **Blok(y) navíc**. Při zkoumání chyb jsem zjistila, že tyto bloky není vhodné spojovat se skupinou Duplicitní kód, neboť tyto bloky nejsou duplicitní, ale naprosto přebývají.

Z grafu níže (Graf 3) vyplývá, že nejčastěji zaznamenanou odchylkou byly Nevhodné konstrukce. Jsou zde níže zmiňované případy, kdy žák nevyužil funkce můj blok, kterou již v době tvorby ovládali, ale také nepoužití vhodného opakování.

Druhou nejčastější kategorií byly duplicitní kódy. Nejen že žáci někdy zapomínali využívat konstrukce Můj blok (tyto případy jsou zařazeny v kategorii Nevhodná konstrukce), ale také často používali duplicitních částí kódu bez jakéhokoli efektu.

Bloky navíc byly pak nejméně časté. Jak bylo již řečeno, tyto bloky se od duplicitních liší tím, že tyto kusy kódu neměly být v projektu vůbec. Zároveň byly tyto odchylky počítány jen jako jeden výskyt ve chvílích, když zbylé bloky navíc byly přítomny kvůli jinému bloku navíc (například „opakuj stále“ a „zastav tento scénář“ na konci smyčky)



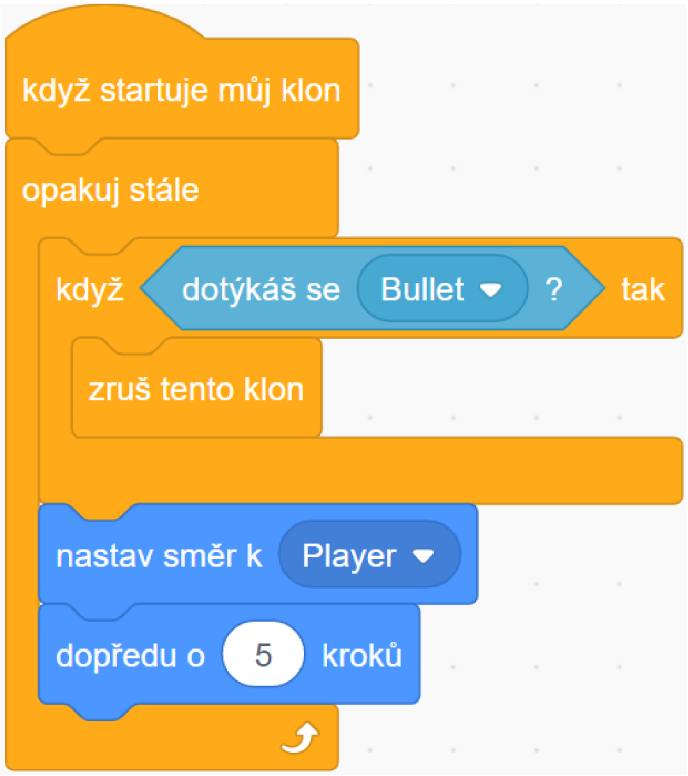
Graf 3: Odchylky od kanonického postupu

Zde nemá smysl ukazovat graf s přehledem, kdo odchylku odhalil, nalezené odchylky jsem vždy našla já. Hlavním důvodem, kromě již zmiňovaného faktu, že pokud na odchylku přišel sám žák, často jsem neměla šanci ji zaregistrovat, je pravděpodobně to, že se příliš neprojevují. Ve chvíli, kdy program dělá podle žáka to, co má, tak obvykle žák nehledá, co by mohl v programu změnit,

a přesouvá se na další aktivitu. Proto byla většina těchto odchylek od běžných postupů odhalena až zpětně při důkladné kontrole úloh, případně při hledání chyby funkční. Žáci na tyto odchylky byli upozorňováni buď prostřednictvím komentáře u úlohy nebo na začátku další hodiny.

Mezi **nevhodné konstrukce** byly tedy zařazeny i některé výskyty duplicitních kódů, ale pouze v případě, že byly způsobeny nepoužitím bloků kategorie „Můj blok“ nebo „opakuj“. Hlavní rozdíl mezi těmito dvěma variantami je ten, že pokud je kód duplicitní z důvodu špatné konstrukce, opravdu má proběhnout vícekrát, pokud ne, jedná se čistě o duplicitní kód. Příkladem takové nevhodné konstrukce je i Obrázek 33.

Obrázek 33: Odchylka od kanonického postupu 1



Obrázek 34: Odchylka od kanonického postupu 2

Zbylé odchylky v této kategorii způsobovalo použití nevhodného opakování, především pak kombinace bloků „opakuj stále“ a „když __ tak“ namísto „opakuj dokud nenastane __“ (viz Obrázek 34).

Duplicitní kódy byly často výsledkem nepozornosti. Žáci si po sobě příliš nekontrolovali své výtvory, především pak ve chvílích, kdy se na plátně projektu dělo to, co chtěli či očekávali, že se dít bude.

Zde (viz Obrázek 35) je ukázka nejjednodušší formy duplicitního kódu. Jedná se o naprosto totožný úsek programu. V tomto případě se jedná pouze o odchylku od kanonického postupu,

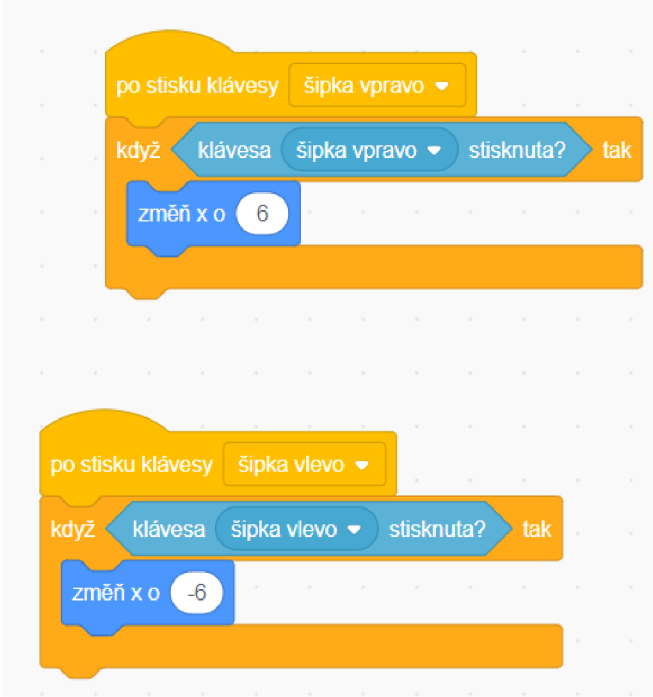
ovšem pokud namísto „změň na __“ typu bloků byly použity některé z bloků „změň o __“ nebo „další __“, mohlo by dojít k nesprávnému průběhu kódu. V takovou chvíli by bylo nutné tento kód označit za funkční chybu.

Obrázek 35: Odchylka od kanonického postupu 3

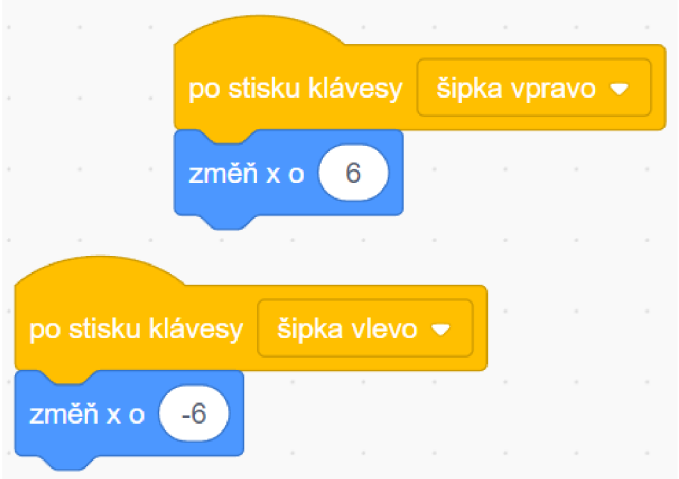
Další ukázkou duplicitního kódu je Obrázek 36. Zde se jedná o použití bloku a konstrukce se stejnou funkcí v jedné sekvenci kódu. Jednodušší a přehlednější verzí se stejným výsledkem je Obrázek 37.

Bloky navíc zahrnují veškeré nadbytečné bloky, které ale nenarušují chod programu jako chyby funkční. Běžně se vyskytovaly bloky „skryj se“ a „ukaž se“ ve chvílích, kdy nebyly zapotřebí („skryj se“ ve chvíli, kdy byla postava skryta a naopak „ukaž se“, když byla viditelná). Pak také blok opakování a na jeho konci zastavení scénáře pro vytvoření sekvence, která se nebude opakovat (viz Obrázek 38[11]), v tomto případě se jednalo o následující dvě hodiny po představení cyklů, v pozdějších lekcích už pak k těmto konkrétním chybám nedocházelo. Zbylé případy byly nadbytečné události „po obdržení zprávy __“ (viz Obrázek 39).

11 Obsah některých bublin v ukázce byl smazán kvůli anonymizaci žáka, který daný kód vypracovával.

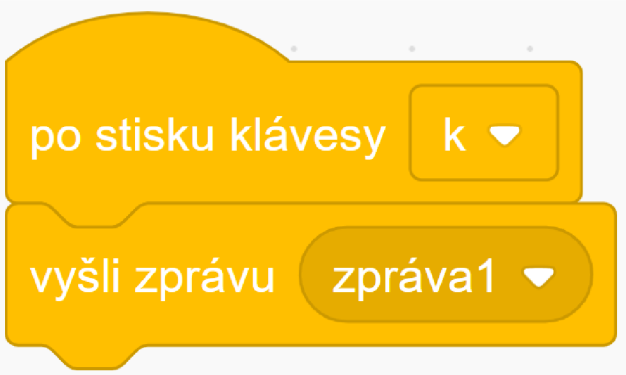


Obrázek 36: Odchylka od kanonického postupu 4



Obrázek 37: Odchylka od kanonického postupu 4 (řešení)

Obrázek 38: Odchylka od kanonického postupu 5

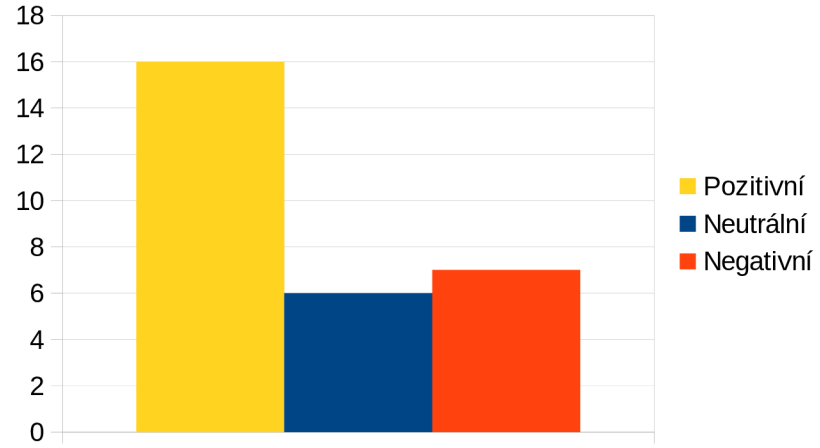


Obrázek 39: Odchylka od kanonického postupu 6

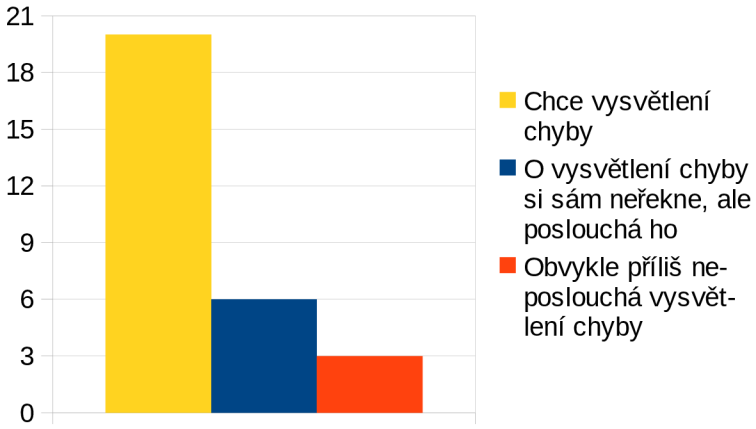
4.2 Postoj žáků k chybě

V rámci výuky jsem žákům často opakovala, že chyby jsou součástí učení. V tomto ohledu jsem šla žákům příkladem a svých případných chyb se neostýchala a přiznávala je. Domnívám se, že to alespoň z části žáky podpořilo v přijímání vlastních, ale i cizích chyb – během žádné z hodin se mi nestalo, že by došlo k posměškům nebo nějakým jiným negativním komentářům mezi žáky.

Následuje grafem znázorněný přehled, kolik žáků se z celkového počtu 29 žáků staví k chybám pozitivně (bere je jako výzvu), negativně (odrazuje je od dalšího snažení) nebo neutrálně, viz Graf 4.



Graf 4: Postoj žáka k chybě



Graf 5: Postoj žáka k vysvětlení chyby

Valná většina přistupovala k chybám pozitivně, často v chybách viděli šanci zlepšit se a dokázat něco víc než minule.

Ještě kladnější přístup měli žáci k vysvětlování chyby a správného řešení, viz Graf 5. Někteří žáci vyžadovali vysvětlení i těch chyb, které si sami opravili, aby si příště mohli být jistí svým řešením. Naopak se v pozorovaném vzorku vyskytlo i několik žáků, kteří vysvětlení sotva poslouchali.

Na první pohled by se mohlo zdát, že bude jasná korelace mezi žáky, kteří si vysvětlení nevyslechli, a žáky, kteří chyby opakovali. Pozorování však odhalilo, že je to pravda jen částečně. Zatímco žáci, kteří vysvětlení chtěli, opakovali chyby nejméně často, ti, kteří vysvětlení příliš neposlouchali, neopakovali své chyby tak často, jako ti s neutrálním přístupem.

Z toho usuzuji, že žáci, kteří neměli o vysvětlení zájem, měli tento postoj z důvodu, že chybu (a tím i řešení) často pochopili už ze správného řešení. Tento typ žáků byl ovšem v hodinách často velmi pasivní a obvykle pouze plnil zadání bez touhy ke svému řešení přidat něco navíc.

Naopak žáci toužící po vysvětlení a opravení i drobných chyb byli obvykle ti, kteří do svých projektů vkládali originální části a části navíc, také často plnili úlohy, které byly navíc.

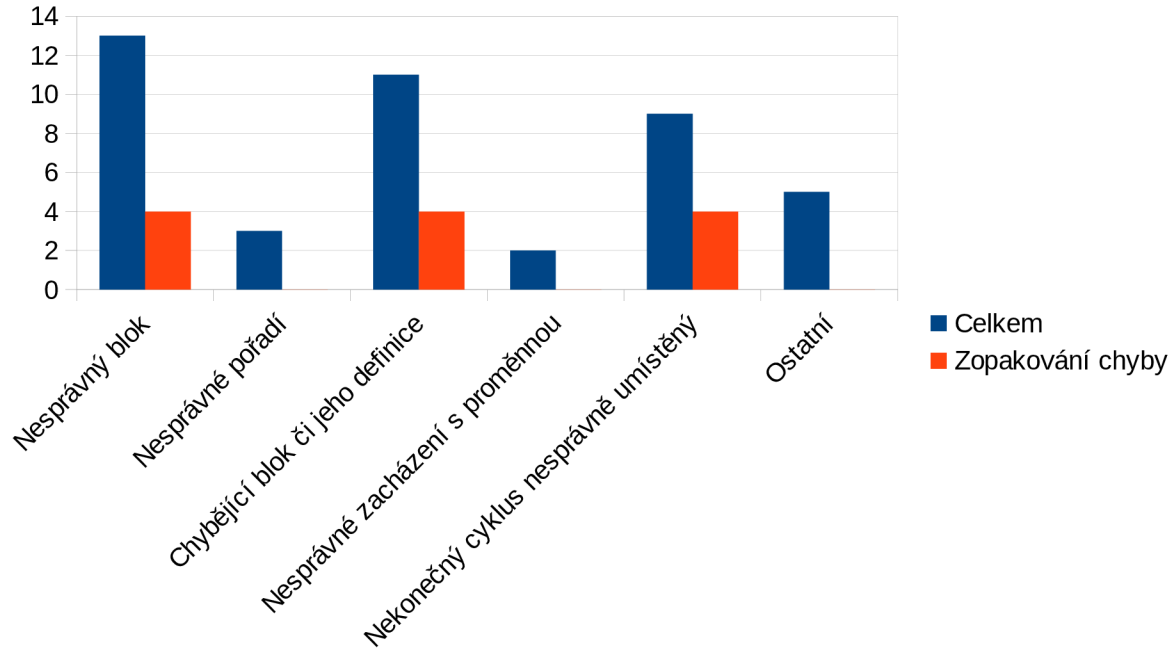
Nejčastěji opakovali chyby žáci s neutrálním postojem k vysvětlení i k chybě samotné. Zde usuzuji, že je to nejspíš z toho důvodu, že chyba jim nepřinášela ani pozitivní ale ani negativní motivaci, a proto se z ní příliš neponaučili.

Tato pozorování by ovšem měla být brána s nadhledem vzhledem k velikosti vzorku jak pozorovaných žáků, tak vypozorovaných chyb.

4.3 Nejčastěji opakované chyby a práce s nimi

Zde je nutno opět podotknout, že vzhledem k celkovému množství pozorovaných chyb mohou být tyto údaje zavádějící a bylo by vhodné v budoucnu toto pozorování rozšířit.

Následující graf (Graf 6) vyobrazuje, kolik žáků určitou chybu alespoň jednou zopakovalo.



Graf 6: Opakování chyb

Nejhorší poměr celkového výskytu chyby a jejího zopakování žáky je u kategorie **nekonečný cyklus**. Chyba byla v tomto případě poprvé vysvětlena pomocí heuristického rozhovoru. Přibližný přepis autentického rozhovoru, kde L značí lektorku a Ž žáka:

L: Jak jsi zjistil, že program nefunguje, jak má?

Ž: Nepočítá mi to góly.

L: A kde počítáš góly?

Ž: [najde část kódu, která započítává body] tady.

L: Řekni mi řádek po řádku, co tato část dělá.

Ž: [po chvíli přemýšlení žák postupně vysvětlí, co dělá daná část kódu]

L: Super, takže tohle by mělo fungovat správně. Čím tuto část spouštíme?

Ž: Vlaječkou.

L: Je na začátku tohoto kousku kódu žlutá událost Po kliknutí na vlaječku?

Ž: Není.

L: Čím tuto část tedy spouštíme? Co je na začátku?

Ž: Je to scénář.

L: Správně. A čím spouštíme scénář?

Ž: Tím malým blokem.

L: Ano, spouštíme ho pomocí malého hranatého bloku se stejným názvem. Tak se podíváme, jestli ho někde používáme.

Ž: [po troše napovídání, jak blok vypadá, žák blok najde] Tady je.

L: Tak mi tady zase hezky postupně řekni, co se tu děje.

Ž: [postupně odříká, co který blok dělá]

L: Takže tu máme v opakování dva růžové bloky, co jsme říkali, že tedy dělají?

Ž: Spouští scénář.

L: Přesně tak, před růžovým blokem Kontrola gólů máme ještě blok Ovládání, tak se podíváme, co spouští.

Ž: [najde blok a sám začne říkat, co se v něm děje]

L: Přesně tak. A kdy přestane tenhle scénář probíhat?

Ž: Nikdy, je tam opakuj stále.

L: Přesně, takže kdy se dostane počítač k tomu druhému růžovému bloku?

Ž: Nikdy!

L: Přesně, takže co je potřeba opravit?

Ž: Nesmí se to opakovat!

L: Super, tak to oprav a vyzkoušej znovu.

Při opakování chyby už obvykle stačilo připomenout tento rozhovor pomocí otázky „A kde [problémová část] spouštíš?“. Napotřetí tuto chybu už žádný z žáků neopakoval.

Je také důležité podotknout, že při výkladu tématu jsem žákům vysvětlila, jak fungují bloky typu „Můj blok“, i tak je ovšem žáci využívali spíš jako bloky „vyšli zprávu __“ a „po obdržení zprávy __“. Největší rozdíl mezi těmito konstrukcemi je ten, že blok „vyšli zprávu __“ aktivuje událost „po obdržení zprávy __“, ale nečeká na dokončení sekvence pod příslušnou událostí. Hranatý blok typu Můj blok v tomto ohledu ovšem funguje spíš jako blok „vyšli zprávu __ a čekej“, kdy program čeká na dokončení sekvence pod příslušnou událostí.

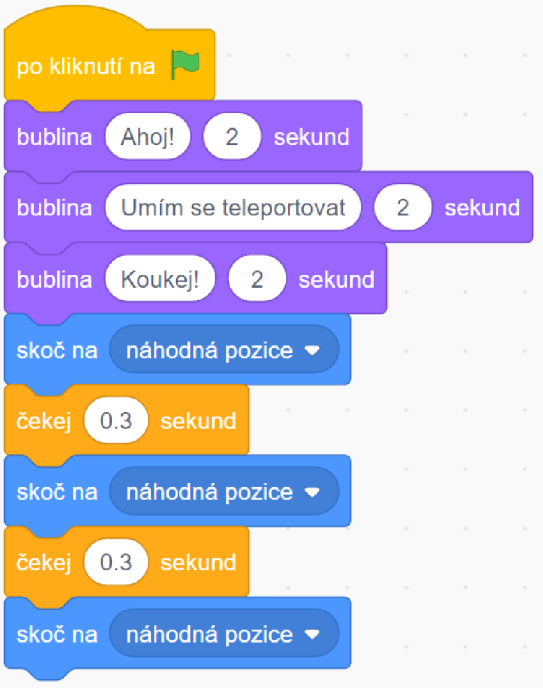
V případě této poněkud specifické chyby, kterou lze vztáhnout i na některé bloky Zpráv, jsem zjistila, že žák si touto chybou nakonec musí projít sám, především pokud se o této funkci dozvídá až v pozdější fázi učení a je navyklý na jiný způsob řešení stejného problému. Tím se sice nevyvarujeme případného opakování chyby, ale žák je poté schopen jednodušeji tuto chybu odhalit a opravit. Zároveň se domnívám, že tento typ chyby, týkající se de facto podprogramu a jeho volání, je dobré prožít si sám, protože jeho řešení podporuje obecnou kontrolu kódu a přináší návod, jak co nejefektivněji hledat chybu.

Druhý nejhorší poměr měla kategorie **Chybějící blok nebo jeho definice**. Nedá se to však s jistotou označit za přímé opakování chyby, neboť se často jednalo o různé bloky. Nejčastější – a to i v opakování – však byly chybějící bloky „ukaž se“, obecněji poté správné počáteční nastavení. Ačkoliv ale možná i právě proto, že je to jedna z prvních věcí, kterou se žáci v kurzu učí, dělalo některým žákům problémy správné počáteční nastavení scény. Žáci chybějící blok „ukaž se“ objevovali poměrně často sami.

Chybějící blok „ukaž se“ se projevuje velmi zřetelně vizuálně, větší problém je pak nalezení správného místa, kde by měl být umístěn. Častou chybou, která na tyto opravy navazovala byla nevhodné umístění bloku, obvykle uvnitř opakování.

U zbytku těchto chyb byl často větší problém najít či uvědomit si, v čem chyba spočívá. Žáci věděli, že program dělá něco jiného, než chtěli, ale neuměli určit, co v něm přesně nefunguje, na rozdíl od předešlého typu chyby, kde obvykle věděli, která část nefunguje.

V těchto případech jsem se dotazovala nejprve na to, jak žák zjistil, že program nedělá to, co by měl. Žáci obvykle poukazovali na rozdíly mezi jejich projektem a projektem ukázkovým. Poté jsme šli společně od konce, respektive od místa, kde se projekty odlišovaly, přes propojené části kódu až k nalezení chyby.



Obrázek 40: Ilustrativní příklad – chybějící blok

Ilustrativní (neautentický) příklad:

Zde chybí blok Skoč na x: 0 y: 0, aby postava začínala vždy tam, kde má. To samozřejmě nemusí být vždy chyba, ale v tomto příkladu to chyba je.

L: Co dělá tvůj projekt jinak, než ten ukázkový?

Ž: Ta kočka je jinde.

L: Tvá postava je jinde než jejich? Čím si myslíš, že to je?

Ž: Jim se teleportuje ještě jednou.

L: Aha, a teleportuje se vždycky jinam?

Ž: Ne, vždycky doprostřed.

L: A nepřipomíná ti to něco?

Ž: Nevím.

L: A kdy se na to místo teleportuje? Co jsme si říkali, že je dobré nastavovat?

Ž: [po chvíli přemýšlení] na začátku. Začátek!

L: Přesně tak, takže co tam musíš mít?

Ž: Skoč na to místo.

L: Super, tak to tam přidej.

V případě těchto chyb jsem vypozorovala, že je vhodné při vysvětlování nejdřív žáky nechat tuto chybu udělat, aby pochopili její význam a především význam počátečních nastavení. Poté jim ze začátku připomínat otázkami na téma, čím by měl náš projekt začínat (událostí, počátečním na-

stavením), že něco takového je dobré nastavovat. Žáci si pak poměrně rychle navykli tato nastavení sami vkládat. Ani tak se těmto chybám však ne vždy vyvarovali, ale domnívám se, že tomu bylo v nižším množství.

Poslední opakovanou skupinou chyb byla kategorie **Nesprávný blok**. Opakování těchto chyb bylo výhradně u bloků pohybu a to zejména prohazování os x a y. Poté ještě blok „dopředu o __“ namísto „změň x/y o __“ nebo kombinace „dopředu o __“ a „nastav směr na __“ či „otoč se o __“.

U těchto chyb jsem nebyla schopna identifikovat, zda se jedná o chyby v důsledku miskonceptu nebo nepozornosti. Žáci tento typ chyby byli obvykle schopni opravit sami (prohozené osy) nebo s drobnou radou (Dopředu namísto Změn x/y).

Obecně této kategorii chyb dle mé zkušenosti nelze příliš dobře předcházet, zejména kvůli jejímu rozsahu. Nelze během výuky dopodrobna vysvětlit, ukázat a procvičit funkci každého bloku (přibližně 100 bloků, kde některé mají možnost upravení), proto je v tomto případě úspornějším řešením nechat žáky takové chyby objevit. Když je zvládnou sami opravit, můžou je pak demonstrovat, opravit a vysvětlit ostatním. Pokud ne, vysvětlí je učitel. Z mé zkušenosti žáci rádi ukazují, co zvláštního se jim podařilo při práci vytvořit, ať už je to způsobené chybou či nikoliv.

4.4 Zbývající chyby a práce s nimi

U zbývajících chyb jsem nezaznamenala jejich opakování. Netvrdím, že je žáci neudělali znovu, ale už je zvládli najít a vypořádat se s nimi sami.

První kategorií, o které budu mluvit, je kategorie **Nesprávné pořadí**. Tuto chybu jsem zaznamenala celkem 3krát. V těchto případech jsem postupovala zhruba následovně:

Nejprve jsem žákovi ukázala, že program nedělá, co by měl, a poté jsem mu dala chvíli čas, aby chybu mohl najít. To se ani jednou nepodařilo, následně jsem se žáka pokoušela na chybu navést. Když ani tento přístup nevyšel, žákovi jsem sdělila, v čem spočívá chyba. Následně jsme spolu debatovali o tom, jak tuto chybu opravit. V případě pořadí jsem pomáhala otázkami „A co se musí stát teď?“ a „Co musíme udělat, aby se to stalo v [pořadí]?“.

Žákům ovšem pořadí nedělalo příliš problémy, což mě překvapilo, neboť dle Swidana se jeden z nejčastějších miskonceptů týká právě sekvenční návaznosti příkazů [7].

Další kategorií je **Nesprávné použití proměnné**. O mém překvapení týkajícím se nízkého množství výskytu chyb této kategorie jsem se zmiňovala již při jejím rozebíráním v podkapitole 4.1.1 Chyby funkční.

Již jsem se zmiňovala i o tom, že žáci s nimi ze začátku mívali problémy, postupně si na ně však buď zvykli, nebo se jim vyhýbali. Zároveň Scratch velkou část práce s proměnnými dělá

za uživatele, například práci se souřadnicemi nebo přijímání vstupu od uživatele pomocí bloku „otázka __“.

V tomto případě vždy stačilo připomenout žákům obecnou práci s proměnnými a souvisejícími bloky. Především připomenout rozdíl mezi blokem „nastav [název proměnné] na __“ a „změň [název proměnné] o __“.

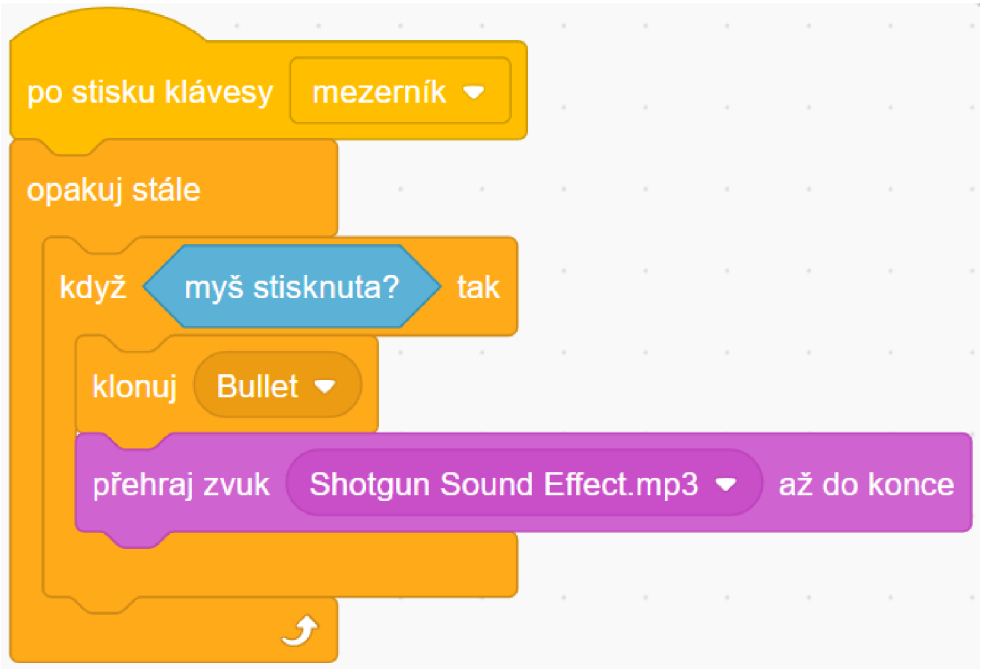
Další kategorií jsou **Ostatní chyby** z chyb funkčních. Vzhledem k různorodosti této kategorie nemohu přesně říci, jak jsem s nimi pracovala. Obecně jsem však vždy šla postupně od projevů přes projití celého kódu s žákem, dokud jsme chybu nenašli.

S odchylkami od kanonického postupu jsem příliš nepracovala, respektive jsem na ně žáky upozornila a nechávala je, aby se rozhodli, co a jak s nimi chtějí udělat. Společně s tím jsem jim vysvětlila, proč je někdy považujeme za chyby a co mohou způsobit (nejčastěji nepřehlednost). V některých případech (především u nevhodných konstrukcí) žáci chtěli poradit. V takovou chvíli jsem se je snažila navést na vhodnější variantu, případně jim ji rovnou řekla.

4.5 Zajímavé a specifické chyby

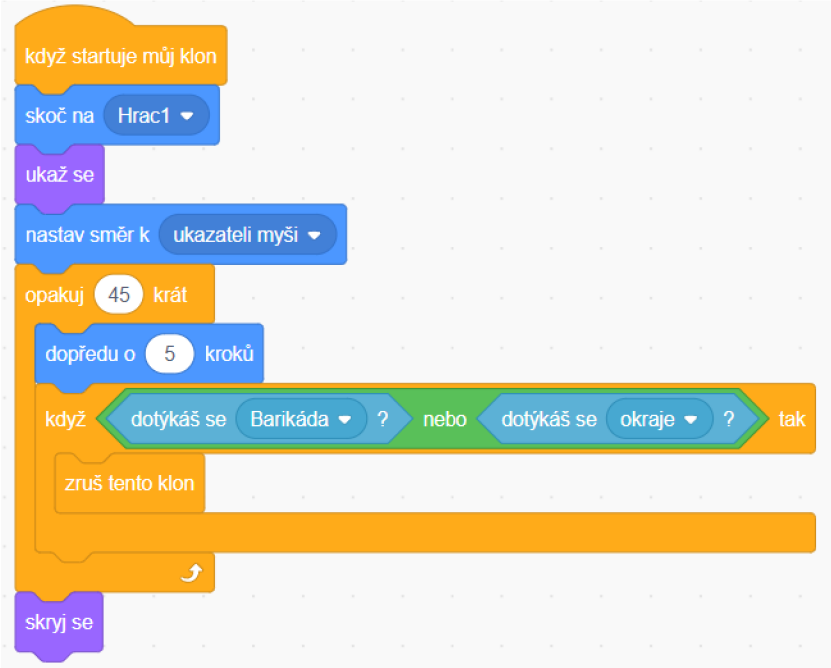
V této podkapitole bych se ráda věnovala několika specifickým a pro mou výuku významným chybám. Některé z nich už jsme si představili v předchozích částech práce, například umístění bloku vnímání na nevhodné místo (viz Obrázek 32 – zde je umístěn ve výběru barvy).

Celou skupinu by mohla vytvořit nevhodná použití bloků podle toho, zda vyžadují dokončení nějaké akce před přesunem na další blok či nikoliv. Jako hlavní příklady bych ráda vyzdvihla již zmíněné bloky ze skupiny Můj blok, bloky „vyšli zprávu __“ / „vyšli zprávu __ a čekej“, ale také blok ze skupiny Zvuk – „přehraj zvuk __ až dokonce“ / „začni hrát zvuk __“.



Obrázek 41: Nesprávný blok zvuku

Tento poslední příklad byl důvodem jedné z chyb kategorie Nesprávný blok (viz Obrázek 41). Zde měl být použit blok „začni přehrávat zvuk ___“, tento konkrétní zvuk má totiž 1.52 sekundy, které musí uplynout, než může hráč znovu vystřelit – nahradili jsme jej tedy zmíněným blokem společně s blokem „čekej 0.5 sekundy“.



Obrázek 42: Obhájená chyba

Další ukázku jsme nakonec se žákem vyhodnotili jako v pořádku a neoznačila jsem mu ji za chybu. Tomuto rozhodnutí předcházela podnětná debata a uvádím ji tu právě kvůli tomu, že to chyba skutečně byla, ale žák si ji dostatečně obhájil. Ve skutečnosti byl tento případ již v ukázce Obrázek 23, tam byla ovšem kvůli funkční chybě, proto pro jednodušší orientaci přikládám totožný obrázek i sem.

Hrubý přepis autentického rozhovoru s žákem:

L: Tady máš, že se to bude opakovat 45krát.

Ž: Jo.

L: A kolik je 45 krát 5? Klidně použij operátor násobení.

Ž: [za použití kalkulačky] 225.

L: A pamatuješ si, jak je velké plátno projektu?

Ž: Ne.

L: Tak si zkus vzít postavu a dát ji do jednoho rohu plátna.

Ž: Dobře [dá postavu do jednoho z rohů].

L: Super, tak se podívej, na jakých je souřadnicích.

Ž: x je -214 a y je -141.

L: Super, říkali jsme si, že taková použitelná část plátna je pro x od -240 a pro y -160, co plusové hodnoty?

Ž: Opak, takže 240 a 160.

L: Přesně, myslíš, že když se ta kulka posune o 225, že opravdu doletí až na konec?

Ž: Ne.

L: Tak jak to opravíme?

Ž: [po chvíli přemýšlení] zvýším opakování.

L: To by určitě šlo, tak to zkus.

Ž: [pár minut pracuje, ale příliš se mu nedaří a pak se začne hlásit].

L: Ano, [jméno žáka]?

Ž: A paní učitelko, když je to pistole, tak nemusí dostřelit tak daleko, ne?

L: Jak to myslíš?

Ž: No pistole mají dostřel.

L: To je pravda.

Ž: Takže kdybych chtěl, tak může mít menší dostřel.

L: To máš pravdu, chceš aby měla menší dostřel, nebo chceš poradit, jak to opravit?

Ž: [po chvíli přemýšlení] já jí nechám dostřel.

L: Dobře.

Ž: A přidám další zbraně a ty budou mít jiný dostřel.

L: Super nápad.

V tomto případě bylo jasné, že žák toto nezamýšlel, ale vzhledem ke správné argumentaci nebylo možné, a ani k tomu nebyl důvod, označit to za chybu.

5 Závěr

Cílem práce bylo pozorovat chyby, které žáci dělali v kroužku programování v jazyce Scratch, a zhodnotit jejich vliv na učení žáků. Práce obsahuje jak teoretickou část (rešerše chyb v programování obecně i těch pozorovatelných v jazyce Scratch) a empirickou (samotné pozorování chyb, popis výuky i žáků a výsledky pozorování a jejich vyhodnocení). Pozorování probíhalo po dobu 6 měsíců na vzorku 29 žáků ve věku 7 – 12 let.

Zaprvé bych ráda znovu upozornila na fakt, že nebylo možné postřehnout všechny chyby, které žáci v průběhu pozorování udělali. Zadruhé je nutné zdůraznit, že se jednalo o zájmovou výuku a vzorek žáků je tím pádem zkreslený a není reprezentativní. V budoucnu by bylo vhodné podobné pozorování opakovat na vhodnějším vzorku, nejlépe v několika oblastech.

Žáci dělali přibližně stejné chyby, které jsem našla při rešerši této problematiky. Nejvíce se vzdálili v kategorii špatná práce s proměnnou, kde pozorovaný vzorek žáků příliš nechyboval. Jak jsem již ovšem uváděla, toto bude pravděpodobně způsobeno zejména tím, že žáci, kterým toto téma dělalo problémy, se práci s proměnnými snažili vyhýbat. Zároveň ve většině případů byly proměnné používány pouze pro záznam bodů. Obecně používání proměnných je zřejmě pro žáky (především ty mladší) ve Scratchi intuitivnější, než by bylo například v jazyce Python.

Kromě práce s proměnnou žáci příliš nechybovali ani při rozhodování o pořadí bloků. I to byl jeden z častých miskonceptů, který se objevoval při rešerši, ovšem pozorovaní žáci s tím nemívali problémy. V tomto případě usuzuji, že podíl na tom má samotný Scratch. Některé bloky přímo zabraňují špatnému pořadí (například bloky události nebo bloky zakončovací. To samozřejmě není zárukou správného pořadí, už jen z toho důvodu, že největší částí bloků jsou bloky akce, které lze napojovat téměř jakkoli. Vizuálnost Scratche ovšem žákům jasněji ukazuje návaznost jednotlivých bloků (příkazů) na sebe.

Naopak žáci dělali chyby v přidávání částí navíc do svých kódů. Tyto části nic nedělaly (především neaktivní sekvence, tzv. dead code), nebo dělaly to stejné, co již jiná část kódu. Tato skutečnost byla překvapivá především z toho důvodu, že jsem nečekala, že by si dobrovolně přidělávali práci.

Nejčastějšími typy chyb byly použití nesprávného bloku pro daný účel, chybějící blok či jeho definice a nevhodně umístěný nekonečný cyklus. Nejhorší poměr opakování a celkového výskytu chyby byl u kategorie nevhodně umístěný nekonečný cyklus. Konkrétně při použití bloků ze skupiny „Moje bloky“, tuto chybu lze však vztáhnout na kterékoli bloky spouštějící vlastní sekvenci s čekáním na její dokončení.

Pozorovaní žáci se obvykle za své chyby nestyděli a měli zájem si je opravit. Celkově příjemné klima jednotlivých skupin jedině přispívalo k využití potenciálu chyb při učení. Žáci se nebáli experimentovat a svá díla neměli problém ukazovat ani v případech, kdy program nefungoval tak, jak zprvu plánovali.

Nevypozorovala jsem, že by žáci opakovali stejný typ chyby, chyb bylo celkově poměrně malé množství a žáci je příliš neopakovali. To považuji za indikátor toho, že žáci se ze svých chyb poučili. Z toho usuzuji, že u většiny chyb je pro žáka lepší (co se délky uchování informace týče) chybu sám udělat a opravit, než aby mu byla vysvětlována předem. Co se práce s chybou týče, nejvíc se mi osvědčil způsob heuristického rozhovoru, ve kterém se snažím žáka navést k nalezení chyby. Žák se je po čase schopen ptát na podobné otázky sám sebe při vlastním hledání chyb.

Seznam literatury

[1] METCALFE J. *Annual Review of Psychology Vol. 68:* 465-489 https://doi.org/10.1146/annurev-psych-010416-044022

[2] SPIŠÁKOVÁ, M. a Ľ. SALANCI. *Chyby ako súčasť motivácie programovania. DidInfo and DidactIG 2017*. Banská Bystrica: Univerzita Mateja Bela, 2017. ISBN 978-80-557-1216-1. s. 136–140.

[3] ČERNOCHOVÁ, M., P. VAŇKOVÁ a J. ŠTÍPEK. *Programování ve Scratch pro pokročilé – Projekty pro 2. stupeň základní školy*. Praha: Univerzita Karlova, Pedagogická fakulta, 2020. ISBN 978-80-7603-085-5.

[4] Scratch - FAQ. Scratch - Imagine, Program, Share [online]. [vid. 11. 4. 2022]. Dostupné z: https://scratch.mit.edu/faq

[5] *Cloning - Scratch Wiki. Scratch Wiki* [online]. [vid. 10. 4. 2021]. Dostupné z: https://en.scratch-wiki.info/wiki/Cloning

[6] *List - Scratch Wiki. Scratch Wiki* [online]. [vid. 10. 4. 2022]. Dostupné z: https://en.scratch-wiki.info/wiki/List#Limits_on_List_Size

[7] SWIDAN, A., F. HERMANS a M. SMIT. *Programming Misconceptions for School Students. In ICER '18: Proceedings of the 2018 ACM Conference on International Computing Education Research*. ACM New York, NY, USA, 2018. s. 151–159.

[8] GUZDIAL, M. *Learner-Centered Design of Computing Education: Research on Computing for Everyone*. Morgan & Claypool, synthesis edition. ISBN 978-1-62705-351-8. kap. 2–3.

[9] Rámcový vzdělávací program pro základní vzdělávání. [online]. Praha: MŠMT, 2021 [vid. 14. 4. 2022]. Dostupné z: https://www.msmt.cz/file/56005/