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

Didaktika programování

Tomáš Hornik
Michal Musílek
Eva Milková

Hradec Králové 2019

Obsah

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

Milé studentky a milí studenti učitelství informatiky. Význam výuky algoritmizace a programování pro rozvoj samostatného a tvůrčího myšlení je nezpochybnitelný a neustále stoupá. Přesto se často jedná o téma nepříliš oblíbené, považované za velmi obtížné, a to nejen žáky základních a středních škol, ale bohužel, v některých případech také samotnými učiteli informatiky. Někdy je tento vlažný vztah k programování posilován nevhodnou volbou programovacího jazyka. Vynikající nástroj pro profesionálního programátora nemusí být vhodný pro začátečníka, protože jeho výkonnost a komplexnost jde ruku v ruce se složitostí a nutností ovládnout mnoho prvků jazyka předtím, než je možné v něm realizovat efektivní algoritmy.

Zcela jiné vlastnosti mají dětské programovací jazyky a s nimi spojená vývojová prostředí. Jsou určeny primárně pro výuku algoritmizace a pro rozvoj tvůrčího myšlení při uplatnění konstruktivistických přístupů. Přitom se ovšem někdy jedná o velmi kvalitní a univerzální programátorské nástroje, využitelné pro realizaci širokého spektra úloh. v rámci této učebnice vás seznámíme se širokou řadou těchto vzdělávacích programovacích jazyků, které jsme osobně vyzkoušeli a otestovali v rámci reálné výuky na školách a pokusíme se vám předat naše zkušenosti a poznatky z praxe.

Možnost výuky algoritmizace byla až doposud na základních a středních školách volitelná[1], ale s novým aktualizovaným kurikulem informatiky tomu tak již pro vás nebude[2]. Pokud tento učební text přispěje ke snadnějšímu, intenzivnějšímu a zábavnějšímu zařazení dětských programovacích jazyků do výuky informatiky, ať už v rámci povinné výuky informatiky, v rámci volitelných a nepovinných předmětů, nebo zájmových útvarů zaměřených na oblast algoritmizace a programování, pak splní svůj účel.

---

1 Národní ústav pro vzdělávání: Metodický portál RVP. (2017). *Rámcový vzdělávací program pro základní vzdělávání*. [online]. Praha: Výzkumný ústav pedagogický v Praze, 166 s. [cit. 2019-07-12]. Dostupné z: http://www.msmt.cz/file/43792/

2 Národní ústav pro vzdělávání: Metodický portál RVP. (2018). *RVP v oblasti informatiky a ICT*. [online]. Praha: Výzkumný ústav pedagogický v Praze, 17 s. [cit. 2019-05-12]. Dostupné z: http://www.nuv.cz/t/revize-rvp-ict

Úvod a organizace skript

V dobách (ne)dávno minulých byla výuka algoritmizace a programování takřka výhradně doménou specializovaných vysokých škol. Ačkoliv si komplexnost celé problematiky vyžádala tvorbu tzv. vzdělávacích programovacích jazyků (ALGORITMY na FIM, Pascal, různé nadstavby k dalším jazykům), tyto veskrze textové jazyky a prostředí nebylo možné využít na základních školách a ani na většině škol středních. v posledních letech však učitelé mladších žáků a studentů již nejsou omezeni jen na Karla, LOGO a Alici, ale mohou si vybrat z nepřeberného množství různých jazyků a prostředí. Ne všechny jsou ale ideální pro použití v běžné praxi a začínající učitel tak může být zahlcen už samotným procesem výběru.

Tento text vás postupně provede od obecného přehledu vybraných a osobně otestovaných programovacích jazyků a prostředí k tutoriálovému vizuálnímu programovacímu prostředí Hour of Code, a dále přes takřka neomezený ale stále ještě vizuální Scratch až k textovým jazykům LOGO a jeho pravděpodobně nejlepší implementaci v rámci profesionálního programovacího jazyka Python.

Ačkoliv byste již měli být blíže seznámení s problematikou algoritmizace, základů tvorby počítačových programů a s obecnými principy počítačů, jsou tato skripta psána tak, aby byla pochopitelná pro každého, kdo dokáže alespoň zapnout počítač a otevřít internetový prohlížeč.

Skripta vycházejí a čerpají ze starší verze skript doktora Musílka,[3] a jsou aktualizována a rozšířena o nové poznatky, nové programovací jazyky a postupy. Skripta dále vycházejí z diplomové práce magistra Hornika,[4] která se zabývala možnostmi využití projektu Hour of Code v běžných hodinách informatiky a představovala další vzdělávací projekty pro výuku programování na základní škole. Část úvodní kapitoly vychází z aktuální verze skript profesorky Milkové, která jsou zaměřena specificky na problematiku algoritmů a jejich znázornění.[5] Ze všech těchto tří prací je se svolením autorů čerpáno.

3 MUSÍLEK, Michal. (2012). *Dětské programovací jazyky.* Univerzita Hradec Králové, 97 s. Scriptum. Dostupné z: http://www.musilek.eu/michal/pdf/deproja0.pdf

4 HORNIK, Tomáš. (2016). *Možnosti rozvíjení algoritmického myšlení s využitím projektů Hour of Code a Scratch*. Hradec Králové: Pedagogická fakulta Univerzity Hradec Králové. 135 s. Diplomová práce.

5 MILKOVÁ, Eva, HAVIGER, J., RUBÁČEK, F. & VOBORNÍK, P. (2010). *Algoritmy - základní konstrukce v příkladech a jejich vizualizace*. Univerzita Hradec Králové, Gaudeamus: 2010, 80 s. Scriptum. ISBN 978-80-7435-064-1.

Očekávané výstupy, osvojené znalosti a dovednosti

Po přečtení těchto skript byste měli být schopni:

- ✔ odlišovat pojmy programovací jazyk a programovací prostředí (anglicky IDE neboli Integrated Development Environment);
- ✔ popsat rozdíly mezi textovými a vizuálními programovacími prostředími a kategorizovat vybraný programovací jazyk a prostředí;
- ✔ vyjmenovat alespoň základní klady a zápory vybraných jazyků a prostředí;
- ✔ vybrat a odůvodnit výběr vhodného jazyka a prostředí pro vaše žáky;
- ✔ nalézt (či vytvořit vlastní) vzdělávací materiály a úlohy pro své žáky;
- ✔ individuálně upravit výuku pro rychlé i pomalé žáky;
- ✔ identifikovat možné problémy, které mohou ve vaší hodině nastat;
- ✔ vytvářet jednoduché i komplexnější scénky, hry i klasické algoritmy;
- ✔ ukládat a otevírat své výtvory, jakož i výtvory vašich budoucích žáků;
- ✔ své programy veřejně publikovat (obsahuje-li IDE danou funkci);
- ✔ vést nejen hodiny informatiky zaměřené na programování, ale také volitelné předměty a zájmové útvary;
- ✔ pracovat s robotickými "hračkami" pro výuku algoritmizace a programování, jmenovitě jsou demonstrovány možnosti OzoBot Evo;
- ✔ a mnoho dalšího. ☺

Navíc se seznámíte s následujícími postavičkami, se kterými zažijete mnohá dobrodružství v hravém světě dětských programovacích jazyků. a to už se vyplatí.

Význam ikonek použitých v těchto skriptech

Aby nedošlo k vašemu úplnému ztracení v tomto relativně obsáhlém textu, můžete v levé části některých stran narazit na podivné obrázky. Jejich účelem není vyplnění prázdného prostoru ani nebyly vybrány pro svou krásu. Jedná se o **ikonky popisující druh textu u kterého jsou přiřazeny**. Setkáte se s těmito obrázky[6]:

Každá kapitola začíná **seznamem otázek, na které byste měli být schopni po přečtení této kapitoly odpovědět** a své odpovědi podložit solidním vysvětlením.

Důležité informace, které je nutné si zapamatovat, jsou vyznačeny panáčkem s vykřičníkem. Čtěte dvakrát! Zejména v **úvodní části a všude, kde je něco definováno**, se s tímto panáčkem setkáte velice často. Občas i několikrát na stránce.

Na konci kapitoly se vždy vyskytuje **shrnutí** těch nejdůležitějších bodů. Přečíst ke zkoušce jen tato shrnutí nestačí, ale měla by Vám pomoci se orientovat v probrané látce.

Shrnutím však lekce nekončí, protože se zde nachází **sekce pro samostatné práce a úkoly**. Důrazně doporučujeme "hands-on" přístup - nepřeskakovat a poctivě procvičovat.

Některá témata jsou označena mozkem. Nejedná se mezipředmětové vztahy s biologií, ale o **rozšiřující látku**, která může pomoci zájemcům prohloubit své znalosti. v seminářích je takováto látka (až na výjimky) jen zmíněna.

Postřehy autora z jeho praxe (práce na základní škole) jsou označeny takto a zkušenosti a názory jiných učitelů se mohou lišit! v každé výuce záleží ve velké míře na osobnosti a přístupu učitele, **neberte tedy prosím tyto postřehy jako tvrdá fakta**.

Všechny kapitoly by měly sdílet strukturu a posloupnost informací naznačenou ikonkami výše. Tato skripta jsou však zamýšlena jako podpůrný materiál k jednosemestrovému předmětu s časovou dotací tři hodiny týdně. Větší kapitoly (typu Scratch) ale potřebují větší prostor, takže shrnutí a procvičování je v takovém případě vloženo za podkapitolu, kterou by měl daný seminář končit.

6 Icons made by Prosymbols from www.flaticon.com

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

1 Dětské programovací jazyky

Jaké dokumenty určují závazný obsah výuky informatiky a jak je v nich reflektována problematika programování a algoritmizace?

Jaké jsou dostupné vzdělávací materiály pro výuku algoritmizace a programování a kde se tyto materiály vzaly?

Jak byste definovali programování, algoritmu, programovacího jazyka a programovacího prostředí?

Jaké jsou klady a zápory výuky programování na školách?

Jakými způsoby lze dělit dětské programovací jazyky?

Jaké jsou výhody a nevýhody tutoriálů typu Hour of Code?

1.1 Aktuální kurikulární stav v ČR a ve světě

Neustále se zvyšující technologická úroveň celého světa vede také ke zvýšeným nárokům na jeho obyvatele. Lidé jsou každý den v interakci s mobilními telefony, notebooky a počítači, ledničkami, televizemi, auty, atp. Každý člověk by tak dnes měl mít alespoň obecnou představu, jak svět kolem nás funguje. Cílem zavedení výuky algoritmizace a programování do běžného kurikula není vytvořit z každého programátora (stejně jako ne každý, kdo absolvoval předměty matematika a fyzika, skončil jako teoretický fyzik ve výzkumném vědeckém ústavu), ale právě pomoci chápat svět kolem nás a rozvíjet nové způsoby myšlení.

Poptávka po odborných pracovnících v oblasti informatiky a programování také neustále stoupá[7, 8] a firmy po celém světě hlásí nedostatek těchto specializovaných pracovních sil. Reakcí na všechny tyto výše popsané problémy jsou nutné změny v kurikulu základního vzdělávání po celém světě. Slovenské ŠVP[9] již problematiku algoritmizace a programování dávno obsahují. Její výuka je detailně popsána

7 POSKOČILOVÁ, Martina (2018). "Ajťáci" chybějí dvěma třetinám firem. In: *Statistika & My: Měsíčník Českého statistického úřadu* [online]. Praha: Český statistický úřad, 10/2018 [cit. 2019-06-07]. Available at: http://www.statistikaamy.cz/2018/10/ajtaci-chybeji-dvema-tretinam-firem/

8 DU, Jie, Wimmer, H., & Rada, R. (2017). "Hour of Code": a Case Study. In: *Proceedings of the EDSIGCON*: 2017, Austin, Texas USA, Vol. 3, p. 1-10. Available at: http://proc.iscap.info/2017/pdf/4343.pdf

a začíná již v 5. ročníku. Obdobná reforma s důrazem na téma programování v hodinách informatiky proběhla i v Británii[10], kde byly samotné kurikulární změny doplněny i webem https://www.computingatschool.org.uk obsahujícím vzdělávací materiály pro britské učitele. v USA je situace komplikovanější, protože každý z 50 států má vlastní kurikulum, do kterého centrální vláda prakticky nezasahuje. Jednoznačný posun směrem k nasazení výuky algoritmizace a programování již na základních školách lze však podložit i vyjádřením bývalého prezidenta Baracka Obamy[11], který svou podporu této oblasti vzdělávání vyjádřil například i natočením motivačního videa pro projekt Hour of Code (viz dále).

V rámci RVP (v aktuálním znění z roku 2017) je obsah informatiky velice všeobecně a povrchně definovaný, což bylo v praxi dvojsečnou zbraní. Minulý čas je zde použit proto, že toto stále ještě platné znění bude z hlediska informatiky kompletně přepracováno a nové znění má vejít v platnost již letos (2019).

Stará verze RVP měla z hlediska informatiky celkový rozsah tři stránky a čtyři řádky, takže kreativní učitelé měli na základě této verze možnost zařadit si do předmětu v podstatě cokoliv a v libovolném rozsahu. Líní učitelé či učitelé bez aprobace Informatika však zpravidla neměli dostatečné znalosti a mnohdy ani chuť vymknout se z tradičního pojetí připomínajícího povinný kurz sady Microsoft Office (což je také špatně, protože se jedná o komerční software ke kterému samozřejmě existují free verze typu Apache OpenOffice nebo Libre Office).

Doposud také existovala jedna třídílná učebnice informatiky[12] a jedna jednodílná[13] obsahující i algoritmizaci a programování, kromě toho však **kvalitních metodických materiálů bylo minimum**. Tyto dvě sady učebnic byly jedinými učebnicemi se **schvalovací doložkou ministerstva školství[14]** a v nejnovější verzi již i výše zmiňovaná jednodílná učebnice chybí. **Dle školského zákona** můžete používat i zdroje jiné, ale schvalovací doložka zajišťuje soulad s kurikulem.

---

9 Štátny pedagogický ústav (2014). *Inovovaný ŠVP pre 2. stupeň ZŠ - Informatika.* [online]. Bratislava: ŠPÚ, 2017 [cit. 2018-06-19]. Available at: http://www.statpedu.sk/files/articles/dokumenty/ inovovany-statny-vzdelavaci-program/informatika_nsv_2014.pdf

10 ANDREWS, Stuart. (2014). Inspire kids to code. In: *PC Pro*. London: July 2014, 237, p. 24-36.

11 The White House, Office of the Press Secretary (2016). FACT SHEET: President Obama Announces Computer Science For All Initiative [online]. Washington, DC: January 30, 2016. Available at: https://www.whitehouse.gov/ sites/whitehouse.gov/files/images/FACT%20SHEET%2BPresident%20Obama%20Announces%20Computer%20Science%20For%20All%20Initiative_0.pdf

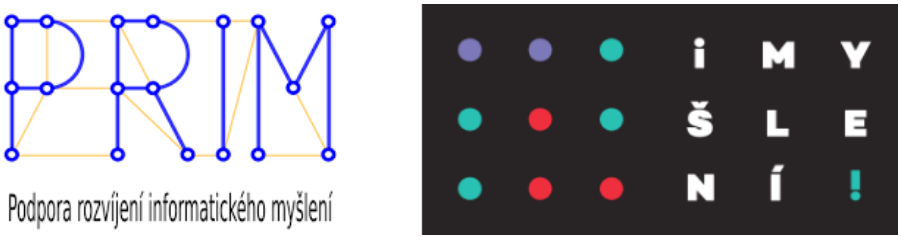
12 KOVÁŘOVÁ, Ludmila, Vladimír NĚMEC, Michal JIŘÍČEK a Pavel NAVRÁTIL. *Informatika pro základní školy (1. až 3. díl)*. 2. vydání. Computer Media, 2012. ISBN 978-80-7402-015-5.

13 VANÍČEK, Jiří. (2012). *Informatika pro 1. stupeň základní školy: informační a komunikační technologie*. v Brně: Computer Press, 2012. ISBN 9788025137499.

14 MŠMT - Ministerstvo školství, mládeže a tělovýchovy. (2019). *Schvalovací doložky učebnic*. [online]. Publikoval Pohořelý Svatolupk, 2019-08-13. Dostupné z: http://www.msmt.cz/vzdelavani/skolstvi-v-cr/schvalovaci-dolozky-ucebnic-2013

Výše jmenované nedostatky vedly k nutnosti hledání řešení celkově nešťastné situace, a to zejména proto, že schopnost algoritmizace (neboli vytváření postupů řešení různých problémů) není vrozená a nerozvíjí se samovolně s věkem daného žáka či žákyně. Tato dovednost by tedy měla být rozvíjena v rámci povinné školní docházky jako jedna ze základních kompetencí a to právě v rámci předmětu informatika a výpočetní technika[15].

Aktualizace zastaralých RVP a tvorba nových vzdělávacích materiálů se staly cílem projektu PRIM (Podpora Rozvoje Informatického Myšlení), v rámci kterého vznikla také aktualizace těchto skript. Jako součást tohoto projektu ale hlavně vznikl **web imysleni.cz**, kde jsou publikovány veškeré vzdělávací materiály pro výuku informatiky vytvořené v projektu PRIM (těmto materiálům je věnována samostatná podkapitola).



Obrázek 1: Oficiální logo projektu PRIM (vlevo) a webu Informatické myšlení (vpravo)

Pod vedením Jihočeské univerzity v Českých Budějovicích probíhá tento projekt již od 1.10.2017 a jeho plánovaný konec je 30.9.2020. PRIM spadá pod Strategii digitálního vzdělávání do roku 2020,[16] která má za cíl kromě aktualizace RVP i poskytnutí veřejně dostupných vzdělávacích materiálů zajišťovaných právě projektem PRIM.

Stejně jako jsou dostupné betaverze vzdělávacích materiálů na webu imysleni.cz, je dostupný i návrh revize RVP v oblasti informatiky a ICT ve vzdělávání (viz Národní ústav pro vzdělávání 2018) a tvorba vzdělávacích materiálů probíhá ve shodě s tímto návrhem. Přestože je v těchto skriptech neustále opakováno téma programování a algoritmizace a je zdůrazňován neustále vzrůstající význam této problematiky na světovém měřítku, neznamená to, že by nově měl být předmět informatika zaměřen jen tímto směrem. Návrh RVP obsahuje rozvoj žáků ve všech směrech oblastech - hardware, práce s daty a informacemi, informační systémy, multimédia, prezentace, slušné chování a bezpečnost na internetu, atd. (viz strany 10-20 plného znění návrhu revize). z hlediska algoritmizace a programování nás potom zajímají konkrétně strany 11 a 12.

15 VANÍČEK, Jiří. (2016). *Výuka algoritmizace patří především do informatiky*. České Budějovice: Jihočeská univerzita, 5 s. [cit. 2018-06-27]. Příspěvek na konferenci Počítač ve škole 2016 v Novém Městě na Moravě. Dostupné z: http://www.pocitacveskole.cz/system/files/soubory/sbornik/2016/vanicek1.pdf

16 RŮŽIČKOVÁ, Daniela. (2017). *Strategie digitálního vzdělávání do roku 2020: Pracovní jednání k revizím SDV.* Praha: MŠMT, 23.10.2017. Dostupné z: http://www.msmt.cz/uploads/SDV2/Ruzickova_DigEdStrat_171023.pdf

1.2 Vymezení základních pojmů

Programování a algoritmizace jsou ve své podstatě dva navzájem neoddělitelné pojmy, protože programování (ve smyslu psaní kódu) musí vždy bezpodmínečně následovat po algoritmickém zpracování daného problému. **Zjednodušeně lze algoritmus definovat** jako postup řešení nějakého problému v podobě sekvence na sebe navazujících dílčích kroků či instrukcí.

Takto chápaný algoritmus je velice **jednoduché ilustrovat na každodenních činnostech** člověka (*uvaření jídla, přecházení přes silnici, oprava židle, atd.*), přičemž člověk je chápán jako **autonomní bytost** se schopností rozhodovat se dle vlastního uvážení. Tuto schopnost počítač jakožto stroj prozatím postrádá a je schopen jen toho, čemu ho člověk-programátor naučí. Hlavní výhodou (a zároveň původním účelem) počítače je **astronomická rychlost výpočtů a zpracovávání dat** obecně. Veškeré postupy výpočtů ale musí počítači zpracovat programátor, který je při programování omezován mimo jiné právě i absencí jakékoliv flexibility počítače v oblasti autonomního rozhodování.

Program lze definovat jako *„algoritmus zapsaný v některém programovacím jazyce.“* [17] Programování je tudíž přepis daného algoritmu do daného programovacího jazyka, ať už textového nebo vizuálního.

Každý programovací jazyk je ale jiný a ideální pouze pro řešení určité množiny úloh, nelze tedy obecně tvrdit, že nějaký konkrétní programovací jazyk je „nejlepší“. Právě zde se celá situace ještě více komplikuje, protože přestože lze tvrdit, že některé algoritmy jsou efektivnější než jiné (stejný problém vyřeší v menším počtu kroků), různé programovací jazyky potřebují pro vykonání stejné instrukce různě dlouhou dobu. Tyto faktory ale z hlediska vzdělávacích programovacích jazyků není potřeba vůbec brát v úvahu.

Dětské programovací jazyky a jejich vývojová prostředí lze v podstatě rozdělit do dvou skupin. Na vývojová prostředí **restriktivní** (tedy ta, která jsou řízená jednotlivými krátkými a vysoce specifickými úlohami zaměřenými na konkrétní problémy z oblasti programování) a **jazyky volné** (tedy naopak ty, které umožňují uživatelům vytvářet cokoliv dle libosti, ale zároveň zpravidla „nevedou uživatele za ručičku“). Doc. PaedDr. Jiří Vaníček, Ph.D. tyto dvě kategorie programovacích prostředí označuje jako **„uzavřená“ a „otevřená“** (Vaníček 2016).

17 ĎURÁKOVÁ, Daniela, Jiří DVORSKÝ a Eliška OCHODKOVÁ. (2002). Základy algoritmizace. Ostrava, 289 s. Scriptum. VŠB Technická univerzita Ostrava, Katedra informatiky.

Mezi první jmenované patří právě Hour of Code, zatímco Scratch je prostředí volné. v případě tohoto způsobu dělení se nelze odkazovat na programovací jazyk jako takový, rozhodující je totiž vždy konkrétní programovací prostředí. Například obecně **jazyk LOGO je volný, ale TurtleAcademy** je již prostředí restriktivní (pro bližší popis viz kapitolu 5).

Rozdělit si programovací jazyky tímto způsobem je pro učitele výhodné proto, že pro prvotní seznámení s novým programovacím jazykem je nejvhodnější využití restriktivního vývojového prostředí. Žáci si tak mohou všechny funkce systematicky a řízeně vyzkoušet a naučí se také základní programovací konstrukce. Učitel při takové hodině funguje jen jako podpora samostatně pracujících žáků.

Poslední zásadní dělení je na **programovací jazyky textové a vizuální/grafické**. v textovém jazyce musí žáci sami psát kód, což vede k obrovskému množství překlepů a neustálému hledání bugů. Ve vizuálním jazyce žáci skládají kód z již předem připravených bloků a problémy spojené se syntaxí zcela odpadají.

Přestože je vizuální programování v posledních letech na vzestupu a většina nových vzdělávacích jazyků je založena na obdobném principu jako Scratch, nelze textové programování úplně zatracovat. Žáci mají často problémy z hlediska čtenářské gramotnosti, systematické práce a soustředění pozornosti. Jelikož je počítač absolutně nemilosrdný, práce s textovým programovacím jazykem tak žáky nutí se soustředit na každé písmenko a učí je tak i pečlivosti. Jedná se také o ideální předstupeň pro programování v plnohodnotném jazyce jako C#, Java, Python, atd.

1.3 Stručné zopakování základů algoritmizace

Co to je algoritmus a jak je definován jste se dočetli již na začátku minulé kapitoly (a také dozvěděli na nějakém předcházejícím předmětu zaměřeném přímo na problematiku algoritmizace, jako je například předmět *Algoritmy a datové struktury* vyučovaný na Univerzitě Hradec Králové). Ačkoliv takovéto vysvětlení je pro žáky zpočátku dostačující, vy jako učitelé musíte znát i základní vlastnosti algoritmů, jejichž dodržování při tvorbě programů musíte kontrolovat.

1.3.1 Vlastnosti algoritmů

Každý algoritmus musí mít následující vlastnosti:

- ✔ **jednoznačnost neboli determinovanost** – v rámci algoritmu musí být jednoznačně určeno, který konkrétní krok/instrukce následuje, přičemž v případě rozhodování mezi vícero možnostmi musí na základě aktuálních dat opět existovat jen jediná správná možnost (počítač se jinak není schopen rozhodnout podle sebe);

- ✔ **resultativnost** – algoritmus vede ke správnému výsledku, cíli či řešení; **konečnost** – algoritmus se nesmí dostat do nekonečného zacyklení a výsledek musí být poskytnut ve smysluplném časovém intervalu (kdyby výpočet trval milion let, algoritmus by neměl smysl);
- ✔ **opakovatelnost** – při opakování algoritmu se zadáním stejných vstupních dat musí být výsledek vždy stejný;
- ✔ **hromadnost neboli obecnost** – vstupní data nejsou definována konkrétními hodnotami, ale jsou označována symbolickými jmény (proměnnými značícími množiny, ze kterých lze data vybrat), díky čemuž lze algoritmus aplikovat na celou skupinu podobných úloh, protože v závislosti na zadaných vstupních hodnotách jsou zohledněny možné alternativy a odchylky v postupu řešení;[18]
- ✔ **srozumitelnost neboli přehlednost** – doc. Ing. Arnošt Motyčka, CSc. ještě dále uvádí jako jednu z nutných vlastností algoritmu i jeho srozumitelnost či přehlednost, která je logickým předpokladem modifikovatelnosti umožňující další rozšiřování či vylepšování stávajícího algoritmu.[19]

1.3.2 Možnosti zápisu algoritmů

Důležitým tématem jsou z hlediska výuky programování také různé možnosti „vyobrazení“ našeho řešení. **Vytvořený algoritmus lze vyjádřit velkým množstvím různých prostředků či zápisů**, přičemž neexistuje jednotná norma, která by byla jednoznačně „správná“ a používaná všemi. Nejjednodušší a nejméně formalizované je **vyjádření algoritmu obyčejnými popisnými větami** v daném jazyce (čeština, angličtina, atd.).

Složitějším a zpravidla formalizovaným způsobem jsou různá grafická znázornění. Mezi tato **grafická znázornění** algoritmů patří například obdélníkové strukturogramy či pravděpodobně nejznámější **vývojové diagramy**. Ty jsou založeny na několika základních tvarech (jako je elipsa, kosočtverec, lichoběžník, obdélník, apod.) z nichž každý má svůj jednoduchý význam a definovaný požadovaný obsah, tedy přiřazenou řídící strukturu. Ovál znázorňuje začátek a konec algoritmu, obdélník jednotlivé výkonné příkazy, nebo jejich bloky, kosočtverec větvení algoritmu pomocí podmíněného příkazu, nepravidelný šestiúhelník řídící příkaz cyklu s předem daným počtem opakování. Chceme-li zdůraznit, že některý výkonný příkaz bude popsán samostatným algoritmem, zdvojíme čáry na levém a pravém okraji obdélníku. Pro znázornění příkazů vstupu a výstupu za běhu programu se používá rovnoběžník. Přechod od jedné části ke druhé je značen šipkami a postup diagramem je zpravidla ve směru svrchu dolů,

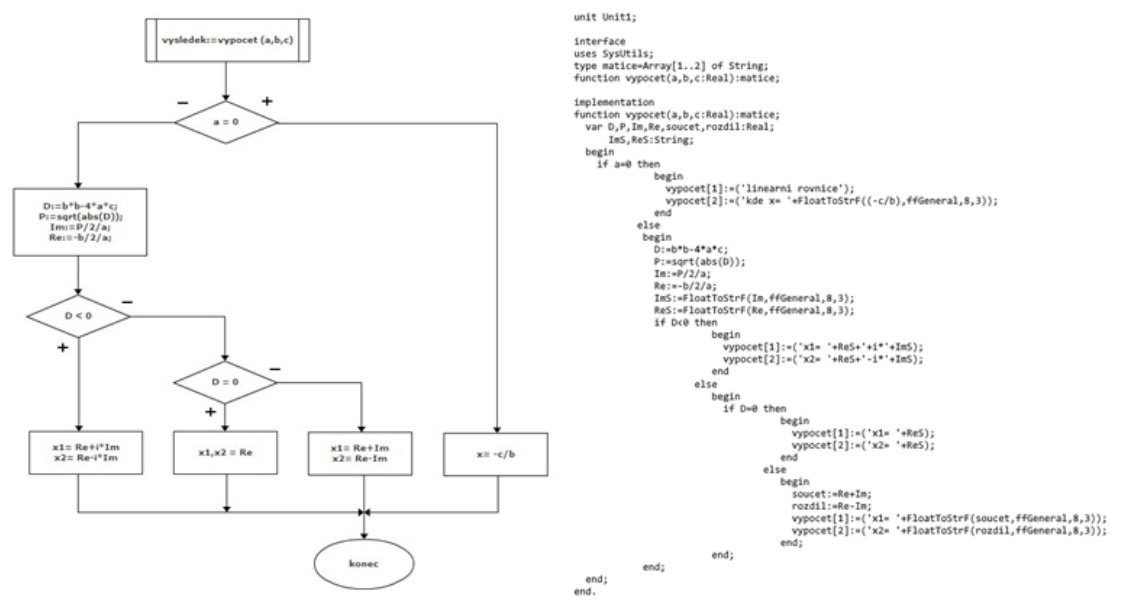
18 VIRIUS, Miroslav. (1995). Základy algoritmizace. Praha: ČVUT, 179 s. Scriptum. ISBN 80-01-01346-4.

19 MOTYČKA, Arnošt. (1999). *Algoritmizace*. Brno: Konvoj, 75 s. Scriptum. ISBN 80-85615-80-0.

zleva doprava, či kombinací těchto dvou možností. Výsledný algoritmus je v tomto zápisu relativně srozumitelný i běžnému člověku ne-programátorovi.

Oproti tomu **kopenogramy rozlišují jednotlivé typy řídících struktur barevně**. Záhlaví příkazu se vybarvuje žlutě, výkonné příkazy růžově, podmíněný příkaz světle modře a začátek a konec cyklu světle zeleně. Konkrétní ukázky práce s kopenogramy naleznete v kapitole o programovacím jazyce Karel. v té době již také bude znát programovací jazyk Scratch a nápadná podoba zápisu barevného kopenogramu a programu ve Scratchi by vám tak neměla uniknout.

Právě zmiňovaný Scratch (a další jazyky založené na původním Google Blockly) jsou takovým mezistupněm mezi grafickým zápisem algoritmu a programovacími jazyky. Nejsložitějším a pro ne-programátora nejméně srozumitelným zápisem algoritmu je jeho zpracování ve formě **programu v konkrétním programovacím jazyce**. Typickým příkladem může být programovací jazyk ALGOL 60 (ze kterého vyšel i novější Pascalu, což býval také dedikovaný vzdělávací jazyk), který přestože nebyl masově rozšířený v praxi, byl po několik desetiletí považován za standard pro publikace algoritmů ve vědeckých časopisech.[20]



Obrázek 2: Vývojový diagram (vlevo) a program v PASCALu (vpravo) pro výpočet kvadratické rovnice

Na obrázku výše můžete vidět, že i relativně jednoduchý problém může ve standardních stylech zápisů působit odstrašujícím a komplikovaným dojmem. Výuka pomocí vývojových diagramů do běžných hodin informatiky na základní škole nepatří a může být takřka plnohodnotně nahrazena některým z dětských programovacích jazyků představeným v těchto skriptech. To však neznamená, že by takováto výuka měla být opomíjena na specializovaných středních školách.

20 MUSÍLEK, Michal. (2011). Kapitoly z dějin informatiky. Vyd. 1. Hradec Králové: Gaudeamus, 193 s. Scriptum. ISBN 978-80-7435-129-7.

1.3.3 Základní algoritmické konstrukce

Ve všech následujících kapitolách se budou velice často opakovat určité klíčové pojmy, jejichž zjednodušené vysvětlení najdete v této kapitole. **Příkazy** jsou zpravidla jednoduché jednořádkové instrukce, které má počítač v danou chvíli konkrétně provést. Příkazy lze dělit na jednoduché nebo strukturované, můžeme je slučovat do bloků, atp. **Cykly** nám umožňují řízeně opakovat určité vybrané úseky našeho programu. Cykly jsou různého druhu, například cyklus s podmínkou na začátku nebo cyklus s podmínkou na konci.

Aby program mohl reagovat na měnící se situaci (vstupní data, nastavení vnitřních proměnných), je nutné využívat **rozhodovací podmínky** jako například "když... tak..." (občas překládáno jako „jestliže… pak…“). Díky nim můžeme program větvit (představíte-li si nějaký složitější vývojový diagram, často bude vypadat jako takový obrácený strom). Stejně jako cyklů existuje i více druhů podmínek a stejně jako s cykly s nimi budete seznámeni na konkrétních příkladech dále v textu.

Datové typy a struktury jsou velice variabilní jazyk od jazyka. Zpravidla rozlišujeme jednoduché datové typy jako je třeba integer (kladné nebo záporné celé číslo) nebo char (jeden znak), případně komplexnější struktury typu string (řetezec znaků) nebo list (seznam).

V dětských programovacích jazycích se zpravidla datový typ neřeší. Většinou rozlišujeme jednoduchou **proměnnou**, jejíž datový typ blíže nespecifikujeme a **seznam**, nic víc. Stejně funguje například JavaScript nebo Python, u jiných programovacích jazyků typu Java nebo C# či C++ je však nutné datové typy specifikovat.

Veškeré výše jmenované operace a struktury v podstatě existují v každém programovacím jazyce a odlišují se jen minimálně, **největším rozdílem je vždy přesná syntaxe** daného programovacího jazyka. To je jeden z důvodů, proč jsou dětské programovací jazyky tak důležité a v rámci výuky informatiky skutečně významné. Přestože **nikdo nevytvoří reálně použitelný bankovní systém ve Scratchi**, veškeré výše jmenované abstraktní konstrukce se lze naučit již v dětských programovacích jazycích a pochopí-li žáci princip práce se jmenovanými strukturami, potom jim skutečně zbývá jen naučit se novou syntaxi a specifika vybraného programovacího jazyka.

Pro připomenutí problematiky dále uvedeme výtah ze skript profesorky Milkové, citovaných v úvodu těchto skript. Porovnáte-li probírané algoritmické konstrukce se zápisy v dětských programovacích jazycích, rychle zjistíte, že všechny konstrukce jsou ve vzdělávacích jazycích zastoupeny a většina úloh z vysokoškolských skript je bez problémů řešitelná v prostředí primárně určeném pro výuku mladších žáků a studentů.

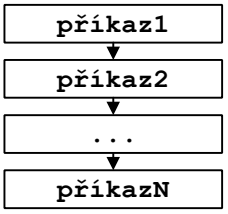
1.3.4 Posloupnost příkazů

Posloupnost příkazů je algoritmická konstrukce, která obsahuje mezi klíčovými slovy **začátek** a **konec** jednoduché nebo strukturované příkazy, navzájem od sebe oddělené středníkem, které jsou vykonány v uvedeném pořadí.

Pseudokód

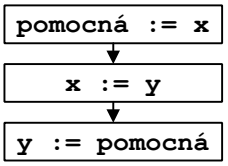
```
začátek
  příkaz1;
  příkaz2;
  ...
  příkazN;
konec
```

Grafický zápis



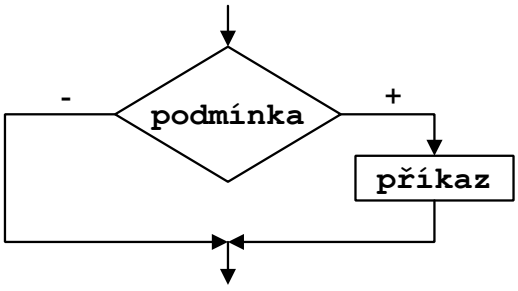
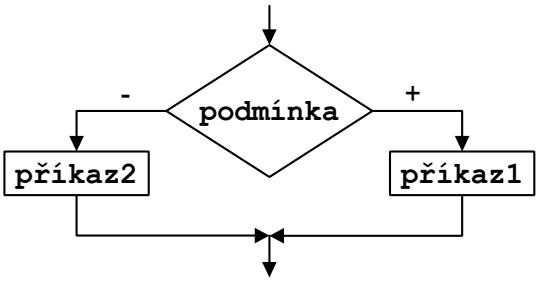
Je-li zapotřebí provést záměnu hodnot proměnných `x` a `y` (tj. původní hodnotu proměnné `x` uložit do proměnné `y` a naopak, původní hodnotu proměnné `y` uložit do proměnné `x`), použijeme následující posloupnost tří přiřazovacích příkazů.

```
začátek
  pomocná := x;
  x := y;
  y := pomocná;
konec
```



1.3.5 Příkaz větvení

Příkaz větvení je algoritmická konstrukce, která obsahuje za klíčovým slovem **jestliže** podmínku, na jejímž splnění nebo nesplnění závisí vykonání příkazu (buď jednoduchého nebo strukturovaného) uvedeného za příslušným klíčovým slovem (**pak**, **jinak**). Rozlišujeme dva základní typy, a to větvení neúplné a úplné.

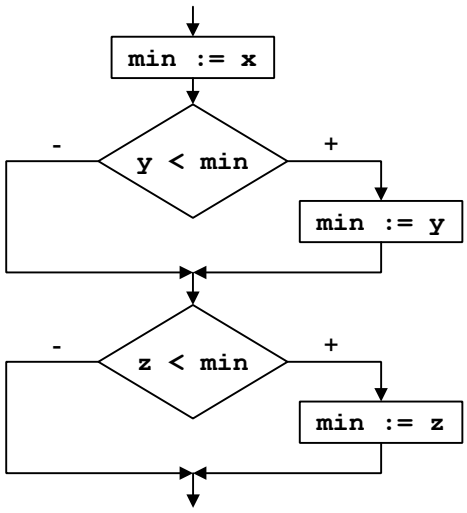
Neúplné větvení	Úplné větvení
	

jestliže *podmínka* **pak**
  příkaz
	jestliže *podmínka* **pak**
	  příkaz1
	jinak
	  příkaz2

Příkaz uvedený za klíčovým slovem **pak** se vykoná pouze v případě, že podmínka je splněna. v grafickém zápise značíme splnění podmínky znaménkem + a její nesplnění znaménkem -.	Příkaz uvedený za klíčovým slovem **pak** se vykoná v případě, že podmínka je splněna a příkaz uvedený za klíčovým slovem **jinak** se vykoná v případě, že podmínka splněna není. V grafickém zápise značíme splnění podmínky znaménkem + a její nesplnění znaménkem – (případně slovy ano/ne).

Příklad neúplného větvení

Napišme posloupnost příkazů, kterými určíme minimální hodnotu ze tří hodnot uložených v proměnných x, y a z.



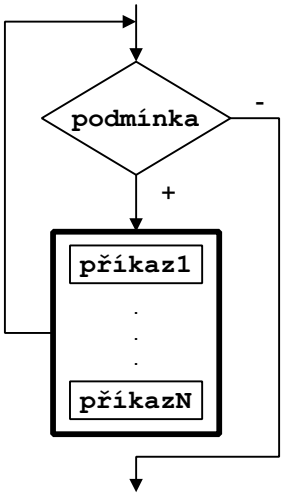

```
začátek
  min := x;
  jestliže y < min pak
    min := y;
  jestliže z < min pak
    min := z;
konec
```

Příklad úplného větvení

Sestavme algoritmus, který vypočítá a standardním způsobem vypíše čtvrtý člen posloupnosti určené následovně. První dva členy jsou zadány, a to tak, že první člen je menší než druhý, a další členy se získají vynásobením předchozích dvou členů, tj. $a_n = a_{n-2} \cdot a_{n-1}$, n = 3,4,…. (výsledek je pak např.: 2, 5, 10, **50**, 500, 25000, ...)

```
začátek
  čti(x);
  čti(y);
  jestliže x < y pak
  začátek
    další := x * y;
    x := y;
    y := další;
    další := x * y;
    napiš("čtvrtý člen posloupnosti = ", další);
  konec
  jinak
    napiš("Špatně zadány první dva členy posloupnosti.");
konec.
```

1.3.6 Příkaz cyklu

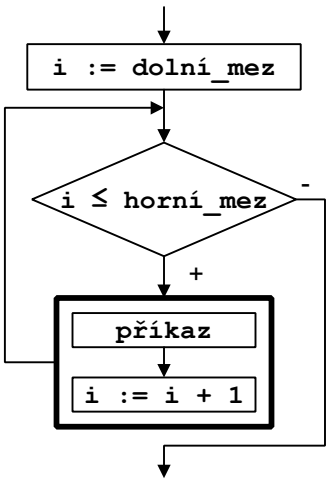


Příkaz cyklu je algoritmická konstrukce, která obsahuje za klíčovým slovem **dokud** podmínku, při jejímž splnění je opakovaně prováděn příkaz (buď jednoduchý nebo strukturovaný), který je uveden za klíčovým slovem **opakuj**, a který nazýváme tělem cyklu. Tj. dokud je podmínka splněna, opakovaně se vykonává tělo cyklu.

```
dokud podmínka opakuj
  příkaz
```

Např.: Sestavme algoritmus, který vypočítá pátý člen posloupnosti zadané v předchozím příkladu.

```
začátek
  čti(x);
  čti(y);
  jestliže x < y pak
  začátek
    i := 3;
    dokud i <= 5 opakuj
    začátek
      další:= x * y;
      x := y;
      y := další;
      i := i + 1;
    konec;
    napiš("pátý člen posloupnosti = ", další);
  konec
  jinak
    napiš("Špatně zadány první dva členy posloupnosti.");
konec.
```



V algoritmech se často vyskytuje následující algoritmická konstrukce příkazu cyklu:

```
i := dolníMez;
dokud i <= horníMez opakuj
začátek
  příkaz;
  i := i + 1;
konec;
```

Tělo cyklu je v ní opakováno přesně `(horníMez - dolníMez + 1)`-krát.

Pro zkrácení zápisu tohoto často používaného příkazu cyklu je možno použít zápis následujícího typu (dále uváděný jako *stručný zápis příkazu cyklu*):

```
pro i od dolníMez do horníMez opakuj
  příkaz;
```

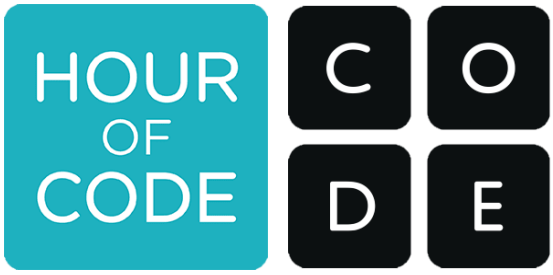
S použitím stručného zápisu příkazu cyklu zapíšeme předchozí algoritmus následovně:

```
začátek
  čti(x);
  čti(y);
  jestliže x < y pak
  začátek
    pro i od 3 do 5 opakuj
    začátek
      další:= x * y;
      x := y;
      y := další;
    konec;
    napiš("pátý člen posloupnosti = ", další);
  konec
  jinak
    napiš("Špatně zadány první dva členy posloupnosti.");
konec.
```

Tyto úlohy lze vytvořit ve většině otevřených vzdělávacích programovacích prostředích (co to jsou otevřená prostředí viz dále). Není tedy problém vyučovat základy algoritmizace za pomoci například jazyka a prostředí Scratch a využít tak jeho intuitivního ovládání a bezproblémové syntaxe. Po seznámení se vzdělávacími programovacími jazyky tak pro výuku zejména na střední škole teoreticky není problém vzít sbírku základních algoritmizačních úloh z téměř jakýchkoliv skript či učebnice a vypracovávat je se žáky tímto způsobem.

1.4 Hour of Code

Prvním z projektů, které si v této kapitole představíme, je jeden z nejvlivnějších projektů vůbec. Přestože má Hour of Code už v samotném názvu slovo hodina, a původní myšlenkou bylo skutečně v rámci jediné hodiny žákům základních a středních škol ukázat, že programování není něco, čeho by se měli bát, celý **projekt se od svého založení v roce 2013 neziskovou organizací Code.org velice rozrostl**. Jejich hlavní kampaň je přesto stále zaměřena na tzv. **„týden kódu,“** v jehož průběhu se na celém světě masově realizují ukázkové hodiny v rámci informatiky, zájmových kroužků apod. Tento týden probíhá zpravidla na začátku prosince a organizace Code.org (které patří projekt HoC) udává, že si jejich ukázkové hodiny ke dni 16.8.2019 vyzkoušelo celkem 820,730,505 lidí (jedná se však o odhad, protože projekt nevyžaduje registraci a nezaznamenává unikátní ID - postup vedoucí k tomuto odhadu je popsán zde https://code.org/loc).



Obrázek 3: Oficiální logo projektu HoC (vlevo) a organizace Code.org (vpravo)

Zakladatelem projektu je Hadi Partovi, který v příspěvku *The „Secret Agenda“ of Code.org*[21] obhajuje myšlenky, na kterých je HoC založen. **Cílem není ze všech žáků vychovat softwarové inženýry, ale samotné pochopení, čeho všeho jsou počítače schopné**, může změnit svět k lepšímu. Hadi Partovi zde tvrdí, že:

> *„Většina dnešních právníků a politiků nemá ani tušení, jak webová stránka funguje, ale přesto řídí internet. Většina dnešních nemocnic stále používá papírové záznamy, které způsobují enormní náklady. Nesčetné množství světových problémů lze vyřešit technologiemi, ale ne v případě, kdy 90 % škol ani neučí, jak vlastně technologie fungují.“*

Podstatou projektu je tedy **přiblížit žákům principy programování** a počítačů obecně, **motivovat** žáky a upoutat pozornost těch, kteří mají potenciál zabývat se problematikou dále. Všichni žáci by dle Partovi měli mít možnost studovat jak počítače fungují, **bez ohledu na jejich pohlaví, věk, národnost či ekonomické a sociální pozadí**. Právě to je problém, který HoC adresuje nejvíce (a na kterém si i vybudoval značnou mediální kampaň), protože v roce 2013 platilo, že

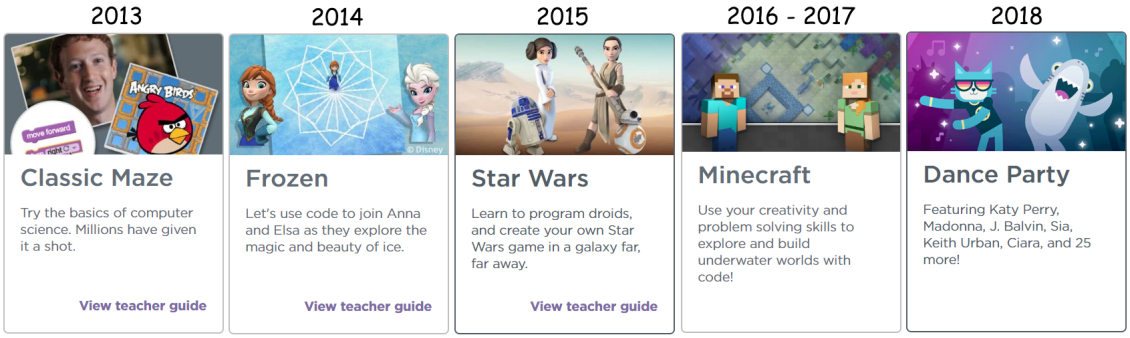
21 PARTOVI, Hadi. (2014). *The “Secret Agenda” of Code.org* [online]. Seattle (Washington) [cit. 2016-03-12]. Dostupné z: http://blog.code.org/post/73963049605/the-secret-agenda-of-codeorg

programování a informační gramotnost vyučovalo jen 10 % škol v USA, které navštěvovali jen privilegovaní jedinci z bohatých rodin (viz Partovi 2014).

Mezinárodní dopad Hour of Code není zanedbatelný. **Dle údajů uváděných společností Code.org bylo navázáno 102 mezinárodních partnerství, tutoriály HoC byly přeloženy do 63 světových jazyků a celý kurz je přeložen do 25 jazyků** (Code.org 2019). Šíření projektu je dále usnadněno optimalizací pro tablety jak s operačním systémem Google Android, tak Apple iOS. Není tak potřeba počítač pro každého žáka, postačí slušné WiFi připojení pokrývající příslušnou učebnu. v rámci překladu je možné narazit na nedodělky či podivné formulace, avšak tyto nepřesnosti zpravidla dokončení hodiny nebrání a žáci si jich většinou nevšimnou.

1.4.1 Šest verzí oficiálního tutoriálu Hour of Code

Tvůrci HoC ve vývoji nestagnovali a každý rok ve svých ukázkových hodinách něco změní a vylepší. Restriktivnost jasně specifikovaných úkolů však ze vzdělávacích důvodů zůstává v každé verzi zachována.



Obrázek 4: Jednotlivé verze Hour of Code podle roku vydání (zdroj: https://code.org/hourofcode/overview)

První verze Hour of Code byla vizuálně založena na enormně populárních hrách **Angry Birds a Plants vs Zombies**, což tvůrcům HoC povolili majitelé zmíněných licencovaných her, firmy **Rovio a PopCap**. Tato verze byla ještě v témže roce modifikována a několik posledních úrovní **změnilo vizuál na Ice Age**.

Po úspěchu oblíbeného animovaného filmu **Frozen** (česky: Ledové království) přibyla v roce 2014 verze HoC **Kóduj s Annou a Elsou**. Zatímco se ve většině tutoriálů postavičky pohybují v mřížce ve stylu programovacího jazyku Karel (viz samostatná kapitola dále), je tato Frozen verze založena na takzvané *"želví grafice,"* se kterou přišel vzdělávací programovací jazyk LOGO (kterým se budeme zabývat v úplném závěru skript).

Spolu s premiérou sedmé epizody **Star Wars: The Force Awakens** byla v roce 2015 vytvořena verze HoC zasazená do tohoto univerza. v této variantě žáci ovládají oblíbené roboty **BB-8, R2-D2 a C-3PO**, jimž pomáhají plnit dílčí úkoly. v roce 2015 například přidali do svého tutoriálu s motivem Star Wars (viz dále) možnost vytvářet nové instance objektů v podobě přidávání nových postaviček a nově i možnosti manipulace s celým prostředím.

Po Star Wars v roce 2016 následovala verze založená na populární hře **Minecraft**, která byla rozšířena ve spolupráci s Microsoftem i o tutoriál pro rok 2017. Verze z roku 2017, zvaná *Hero's Journey,* přidala do jednotlivých úkolů prvek přímého ovládání postavy, přičemž žákem vytvořený kód ovládá postavičku Agenta, která umožňuje manipulaci s herním světem a cílem je umožnit postavičce hráče dostat se na určité místo. Verzi z roku 2016 šlo považovat za určité zklamání, protože výuku neposouvala dále (například k objektově orientovanému programování). Není zde vidět žádná evoluce ve způsobu výuky a stejně tak zde nelze nalézt ani sebemenší náznaky OOP. Naopak se jedná o drastický návrat k samotným základům procedurálního psaní kódu, které bylo k vidění již v roce 2014. Verze z roku 2017 je tedy podstatně zajímavější, ačkoliv je velice krátká.

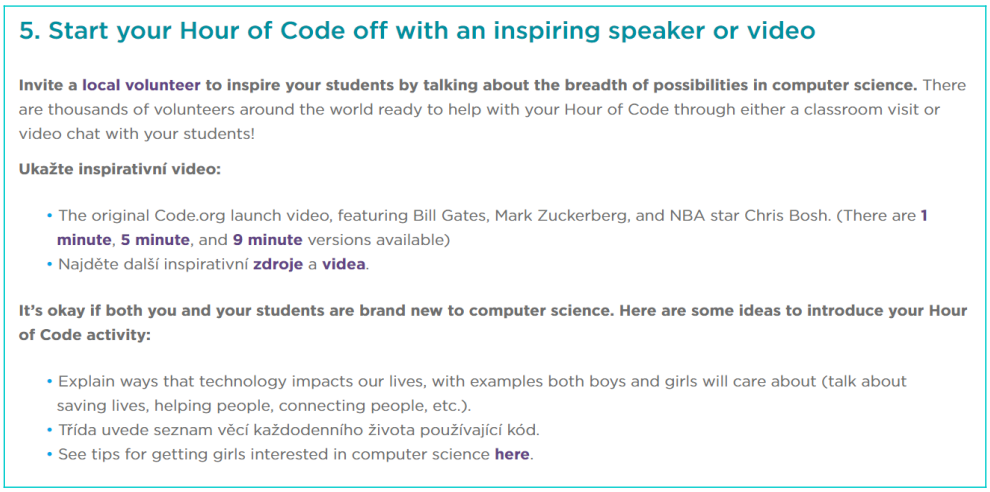
Poslední verze z roku 2018 se jmenuje **Taneční párty** a obsahuje slavné aktuální hudební hity, na které žáci roztančí své postavičky. Výtka určená tutoriálu z roku 2016 je zde anulována, protože každý tanečník je v podstatě instancí objektu třídy tanečníka a stejně tak se s ním i pracuje. Jedná se tak o velice nenásilnou ukázku jednoho ze základních principů objektově orientovaného programování, kterým je vícenásobné využití jednoho kódu. Programátor v tomto případě vytváří takzvané *třídy*, které obsahují nadefinované základní vlastnosti (proměnné) a schopnosti (metody), které by měly všechny objekty daného typu mít. Běžným příkladem třídy je Pes (čtyři nohy, barva srsti, rasa,…) a instancí objektu je pes Alík.

Obrázek 5: Prostředí aktuální verze Hour of Code z roku 2018

Závěrem lze tedy tvrdit, že tento projekt můžete využívat každý rok a žáci se nikdy nudit nebudou, jen je nutné počítat s tím, že zkušenější žáci tímto hodinovým tutoriálem projdou třeba již během dvaceti minut. Tento problém se však dá vyřešit kombinací s nějakým dalším tutoriálem, třeba i od jiného tvůrce.

1.4.2 Ukázková hodina Hour of Code

Pro každého, kdo by chtěl hostovat hodinu kódu (ať již v rámci hodin informatiky, nějakého zájmového útvaru či třeba jen propagační akce pro rodiče, úředníky či další učitele) jsou připraveny rozsáhlé doprovodné materiály (viz dále) poskytující rady stran co nejhladšího průběhu takové hodiny. Bohužel se však setkáte s drobnou jazykovou bariérou - velké části textu jsou jakousi kombinací češtiny a angličtiny (viz obrázek níže), ale většina textu je pouze v angličtině. Tento problém se v samotných tutoriálech až na výjimky nevyskytuje. Tyto výjimky jsou způsobené aktualizací nebo opravou části textu v originální anglické verzi a následně časovou prodlevou k jejímu dopřeložení do češtiny.



5. Start your Hour of Code off with an inspiring speaker or video

Invite a local volunteer to inspire your students by talking about the breadth of possibilities in computer science. There are thousands of volunteers around the world ready to help with your Hour of Code through either a classroom visit or video chat with your students!

Ukažte inspirativní video:

- The original Code.org launch video, featuring Bill Gates, Mark Zuckerberg, and NBA star Chris Bosh. (There are **1 minute**, **5 minute**, and **9 minute** versions available)
- Najděte další inspirativní **zdroje** a **videa**.

It's okay if both you and your students are brand new to computer science. Here are some ideas to introduce your Hour of Code activity:

- Explain ways that technology impacts our lives, with examples both boys and girls will care about (talk about saving lives, helping people, connecting people, etc.).
- Třída uvede seznam věcí každodenního života používající kód.
- See tips for getting girls interested in computer science **here**.

Obrázek 6: Kombinace češtiny s angličtinou na webu Hour of Code (https://hourofcode.com/cz/how-to)

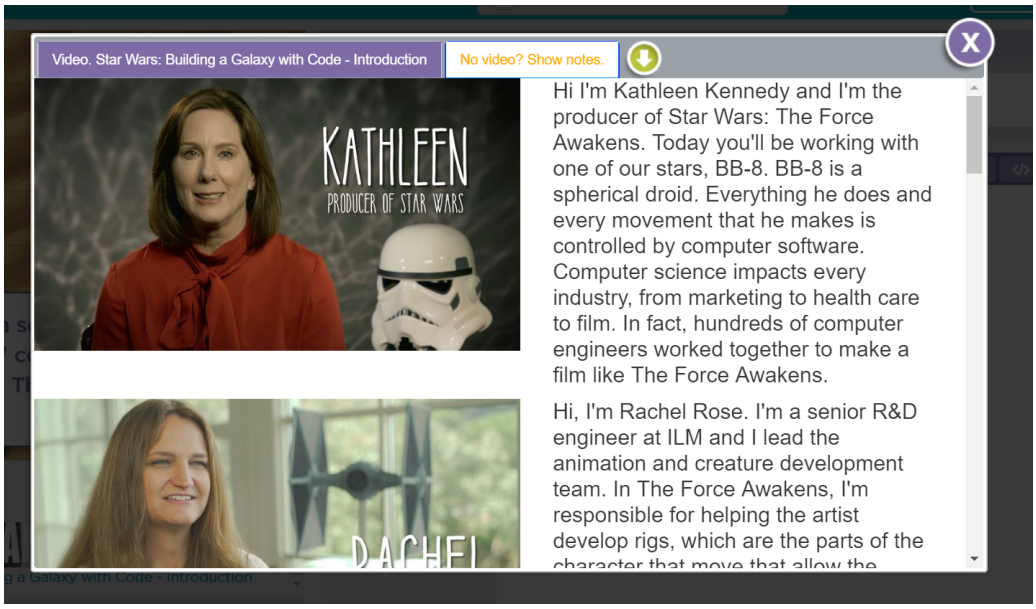
Z těchto rad zde budou vypsány jen některé (a přidány některé extra), které měly na průběh hodin signifikantní dopad a budou popsány nejčastější problémy, na které žáci opakovaně narážejí. Jako úplně první setkání s tímto projektem a tématem programování obecně je původní tutoriál z roku 2014 stále jeden z nejvhodnějších a některé rady se budou týkat právě této verze:

- ✔ **Příprava a kontrola počítačů** - Před samotnou hodinou byste se měli ujistit, že všechny počítače jdou zapnout, fungují a mají připojení k internetu. Nefunkční počítač buď opravte nebo viditelně označte jako nepoužitelný. Jakékoliv technické komplikace vás budou v hodině stát ohromné množství času, který potřebujete na jiné koordinační činnosti. Tato rada ostatně neplatí jen pro HoC, ale pro všechny hodiny informatiky.
- ✔ **Sdílení odkazu a .bat soubor** - Odkaz na vámi vybraný tutoriál by měl být přinejmenším napsaný na tabuli (dotazy *"a jak se tam dostanu?"* jsou neskutečně časté a stejně tak iritující). Ještě lepším řešením je tvorba tzv.

dávkového BAT souboru. Samotná tvorba dávkových BAT souborů je velmi jednoduchá, jedná se totiž o textový soubor s koncovkou .bat (namísto .txt), jehož obsahem je pouze klíčové slovo **start** následované adresou stránky - například "start https://studio.code.org/hoc/1" (bez uvozovek). Takovýto soubor stačí spustit klepnutím myši a on otevře defaultní prohlížeč přímo na požadované stránce.

✔ **Přepnutí jazyka na češtinu** - v závislosti na nastavení konkrétního prohlížeče se může stát, že bude celá stránka HoC defaultně v angličtině. Jazyk se přepíná vlevo dole a po kliknutí na aktuální jazyk se vám ukáže drop-down seznam obsahující právě i češtinu.

✔ **Společně úvodní video a možnost textů s obrázky** - Chcete-li využít velice povedená motivační videa, musíte je pustit celé třídě na projektoru ještě než žáci začnou sami pracovat. Více jak polovina žáků videa automaticky okamžitě vypne, z čehož poté pramení velké množství problémů, například už jen s ovládáním prostředí (které je ve videu krásně ukázáno). Nestíhají-li žáci číst titulky videa, všechna videa lze přepnout na obrázkovo-textovou verzi (viz obrázek níže). Obsah textové verze a videa je totožný, ale u textové verze se pomalejší žáci nestresují tím, že jim titulky utíkají. Tuto možnost však využívá minimum žáků. Počítejte jednoduše s tím, že čtení je problém a že videa budou přeskakována.



Obrázek 7: Ukázka textové verze k videu z tutoriálu Star Wars

✔ **Upozornění na pracovní postup a mazání bloků** - Pustíte-li si společně úvodní motivační video, toto by vůbec neměl být problém. Necháte-li vše na žácích, takřka vždy se najde někdo, kdo video přeskočí a poté neví, co a jak má dělat. Nechcete-li video pouštět, můžete vy či někdo z žáků ukázat ovládání prostředí na první úrovni bludiště na projektoru.

✔ **Upozornění na doporučený limit a světle/tmavě zelené puntíky** - Ne všechna řešení, která fungují, jsou ta správná. Zejména od složitejších úloh, ve kterých je potřeba do cyklu vnořit více příkazů, žáci často využívají výpis příkazů BEZ použití cyklu. Takový program sice funguje, ale není správně. Tutoriál sice žáky pustí do další úrovně, ale puntík dané úlohy bude vybarven jen světle zeleně, namísto tmavě zelené barvy pro správné splnění. Užitečným indikátorem je doporučený počet kostiček nad pracovní plochou. Světle zelené puntíky jsou výborným indikátorem toho, kde se daný žák "ztratil" a co byste mu mohli pomoci pochopit.

Obrázek 8: Tmavě zelená perfektní řešení a světle zelená řešení s příliš velkým množstvím bloků

✔ **Šedé nesmazatelné bloky** - Žákům pomůže zdůraznění, že v úlohách s bloky, které nejdou smazat, jsou tyto bloky již v ideálním složení. Šedé bloky by se tedy neměly nijak přehazovat a upravovat, ale pouze doplnit o chybějící příkazy. Jestliže se do takovéhoto problému zamotají, měli by využít možnost *Začít znovu* vpravo nahoře, která úroveň zrestartuje.

✔ **Přečíst si šeptem příkazy** - Velice často žáci nevědí, kdy se má příkaz vykonat. v takovém případě pomáhá doporučit žákům, aby si sestavený program (zpravidla právě před-připravené šedé bloky) pomalu přečetli nahlas (ale šeptem), případně si příkaz přeformulovali. *"když je cesta vpřed... takže když je přede mnou cestička volná, tak..."*

✔ **Co s dotazy žáků** - v jednom z návodů tvůrci HoC doporučují metodu *"zeptej se tří,"* kdy se má žák s problémem nejprve obrátit na alespoň tři své spolužáky a teprve poté se zeptat vás. v tomto případě však záleží na vašem konkrétním přístupu k třídnímu managementu, kdy tyto dotazy na ostatní žáky můžete považovat za rušení dotazovaných spolužáků či výuky obecně. Může se také stát, že dotaz bude teoretičtějšího rázu (ale stále souviset s tématem) či z nějakého úhlu pohledu, který vás zaskočí. Můžete udělat odhad založený na vašich odborných znalostech, ale nevíte-li ani kde začít, odpověď "já nevím" je naprosto v pořádku a v každém případě lepší než vymýšlení nesmyslů a lží, abyste se z vašeho pohledu neshodili před třídou.

✔ **Pomocník z řad žáků -** Máte-li ve třídě nějaké šikovné a rychlé žáky, můžete je využít jako své pomocníky. Poté, co dokončí celý tutoriál, vám mohou pomáhat radit ostatním žákům, kteří se někde zaseknou. Dobré pravidlo však je, že radící žák nikdy nesmí sahat ani na myš, ani na klávesnici a radu musí dát výhradně verbálně.

✔ **Vnořování více příkazů do cyklu a úprava počtu opakování cyklu** - Opět problém vycházející z přeskakování videí a nečtení instrukcí. Žáka, který na tento problém narazí, většinou bezpečně poznáte podle dlouhé řady světle zelených puntíků.

✔ **Rada k používání cyklů** - Častým problémem také je, že žáci neví, jaké příkazy se mají v cyklu opakovat. v takovém případě pomáhá nechat žáka vyřešit úlohu po jednotlivých příkazech bez využití cyklů a v takto vytvořeném dlouhém seznamu potom najít, jaké příkazy se pořád dokola opakují, dlouhý seznam příkazů "roztrhat" právě na tyto sekce a nakonec spočítat, kolik těchto opakujících se částí je a na řešení tak přijdou sami.

Obrázek 9: Dvanáctá úroveň původního Hour of Code a ukázka práce s cykly

✔ **Vytištění certifikátu** – Na závěr ukázkové hodiny můžete žákům rozdat vytištěné certifikáty s jejich jmény přímo od HoC. Dávkové generování certifikátů je dostupné na adrese https://code.org/certificates a máte-li možnost certifikáty vytisknout, jedná se o pozitivně motivační prvek.

Obrázek 10: Oficiální Hour of Code certifikát pro žáky

✔ **Možnost párové výuky** - Tvůrci HoC tvrdí, že nemáte-li dostatek počítačů, mohou žáci pracovat ve dvojicích. To je sice pravda, ale takovou spolupráci v párech musí žáci umět. v opačném případě se "spolupráce" zpravidla zvrhne v práci jednoho z žáků a tupé zírání druhého žáka. Dobrým pravidlem je například nechat žáky střídat se v ovládání počítače po úlohách, případně je naučit vysvětlovat si navzájem proč se má něco udělat. Ve spíše výjimečných případech je však párová výuka velice efektivním nástrojem. Nejvíce se nám osvědčila v celkově slabších skupinách, kdy dvojice slabších žáků společnými silami na řešení přišla.

V rámci diplomové práce (viz Hornik 2016) bylo na vzorku 123 žáků druhého stupně základní školy zjištěno, že úspěšný průchod všemi dvaceti úrovněmi Hour of Code (tedy tmavě zelené puntíky) zabral nejrychlejšímu z žáků 32 minut, průměrný čas byl 44 minut a celkem tři žáci nestihli projít celý Hour of Code pod 60 minut (výzkum byl prováděn v rámci dvouhodinových lekcí, bylo tedy možné pokračovat přes 45 minutový limit jedné vyučovací hodiny).

Je tedy rozumné **počítat s nějakým plánem pro rychlejší žáky**. Ideálním řešením se ukázalo doplnění HoC o nějaký další krátký ukázkový tutoriál, přičemž v rámci této práce byl k naprosté spokojenosti využit konkrétně Lightbot (viz podkapitola 1.5.1).

Během výzkumu v rámci diplomové práce byla ukázková lekce Hour of Code zopakována celkem jedenáctkrát a autor od té doby tento projekt využil na mnoha ukázkových akcích a v rámci své výuky na základní škole. Veškeré poznatky výše jsou založeny na takto získaných zkušenostech a popisované problémy se opakovaly téměř ve všech skupinách a ve všech věkových kategoriích.

Nespornou výhodou projektu Hour of Code je fakt, že učitel figuruje jen jako koordinátor práce a případný poradce při potížích. To plně odpovídá nutnosti individuálního přístupu k jednotlivým žákům, který je v této podobě skutečně realizovatelný. Další výhodou je možnost zviditelnit vaši školu a pomoci tak například ke sponzorským darům. Podmínkou je ovšem správná propagace vaší realizace projektu.

Naopak určitou nevýhodou je fakt, že samotný Hour of Code sice jednoznačně zvýší motivaci žáků a pozitivně je naladí na téma programování, ale žáci si takto programování za jednu hodinu rozhodně neosvojí. [22, 23]

22 HORNIK, Tomáš. (2016). *Možnosti rozvíjení algoritmického myšlení s využitím projektů Hour of Code a Scratch*. Hradec Králové: Pedagogická fakulta Univerzity Hradec Králové. 135 s. Diplomová práce.

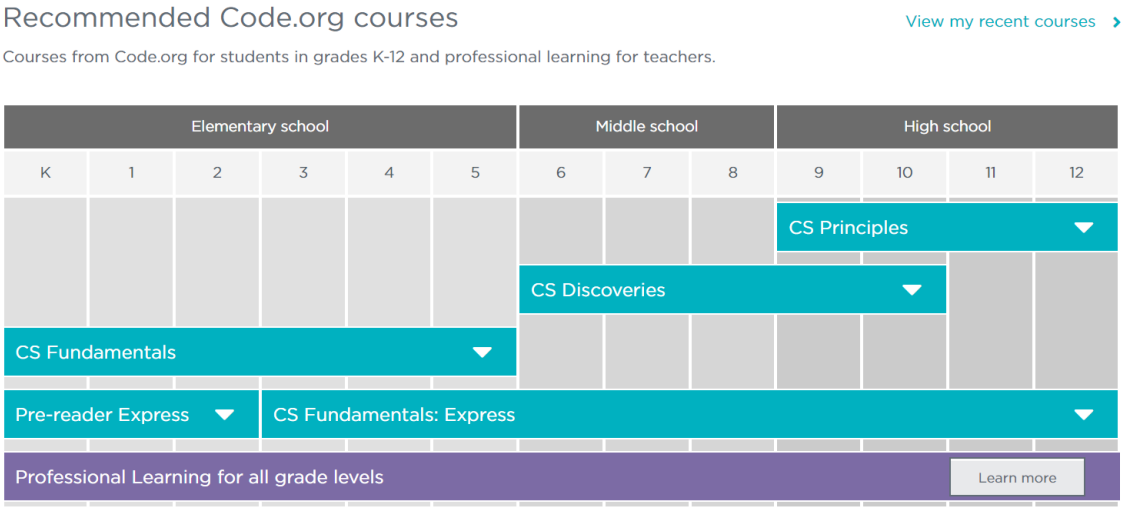
23 DU, Jie, Wimmer, H., & Rada, R. (2016). "Hour of Code": Can it Change Students' Attitudes toward Programming? In: *Journal of Information Technology Education: Innovations in Practice*: 2016, Vol. 15, p. 52-73. Available at: http://www.jite.org/documents/Vol15/JITEv15IIPp053-073Du1950.pdf

1.4.3 Celý kurz a tvorba vlastní třídy

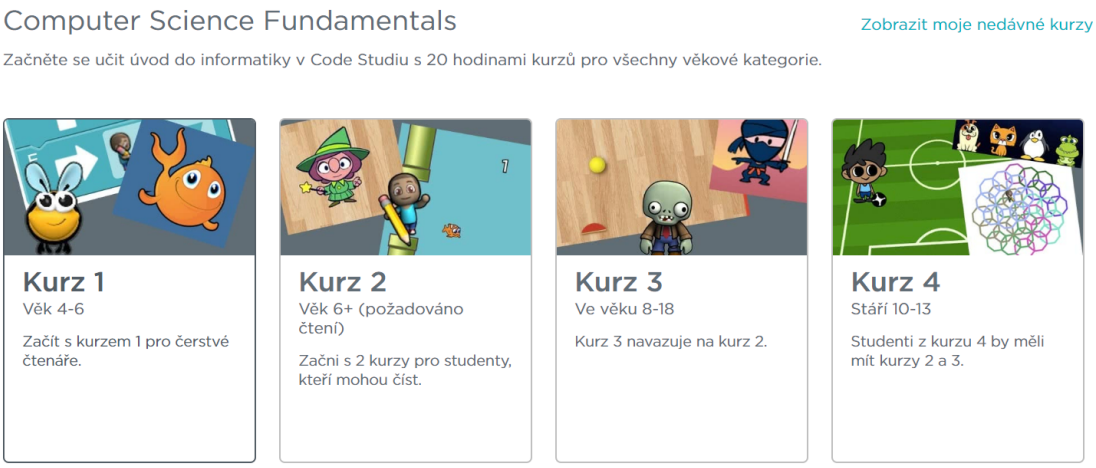
Ačkoliv se může zdát, že realizací jednohodinových tutoriálů projekt Hour of Code končí, v žádném případě tomu tak není. Organizace Code.org s HoC získala masivní momentum, které nasměrovala na komplexní výuku programování a computer science.

Na webu https://studio.code.org/ se nachází katalog kurzů, které je možné vyzkoušet a používat zadarmo a bez registrace. Zde je však nutné upozornit na **omezení české jazykové mutace**. Nabídka kurzů v angličtině je nesrovnatelně rozsáhlejší.

Na obrázcích níže vidíte pro ilustraci jak vypadají nabídky kurzů v závislosti na volbě jazyka (přepnutí na jiný jazyk se provádí na stránce úplně dole uprostřed). Zatímco v **češtině jsou k dispozici čtyři kurzy, každý v rozsahu cca 20 hodin**, v anglické verzi se nachází tři komplexnější kurzy v rozsahu zhruba 50 až 180 hodin, které jsou určené pro celoroční nebo celo-semestrální výuku. Navíc jsou k dispozici dva zrychlené Express kurzy - jeden v rozsahu 14 hodin pro děti, které ještě neumí číst a druhý v rozsahu 30 hodin, který zkráceně shrnuje veškeré významnější učivo až po konec střední školy (v americkém systému označovaný jako K-12). v češtině dále býval k dispozici "Zrychlený úvodní kurz CS" v rozsahu 20 hodin pro žáky až do 8. tříd. Tento kurz stále existuje a je plně česky lokalizovaný, ale není možné se k němu nijak proklikat, proto zde máte k dispozici přímý odkaz https://studio.code.org/s/20-hour (funguje i pro anglickou jazykovou verzi).



Obrázek 11: Základní nabídka kurzů Code.org v ANGLIČTINĚ (zdroj: studio.code.org)



Obrázek 12: Základní nabídka kurzů Code.org v ČEŠTINĚ (zdroj: studio.code.org)

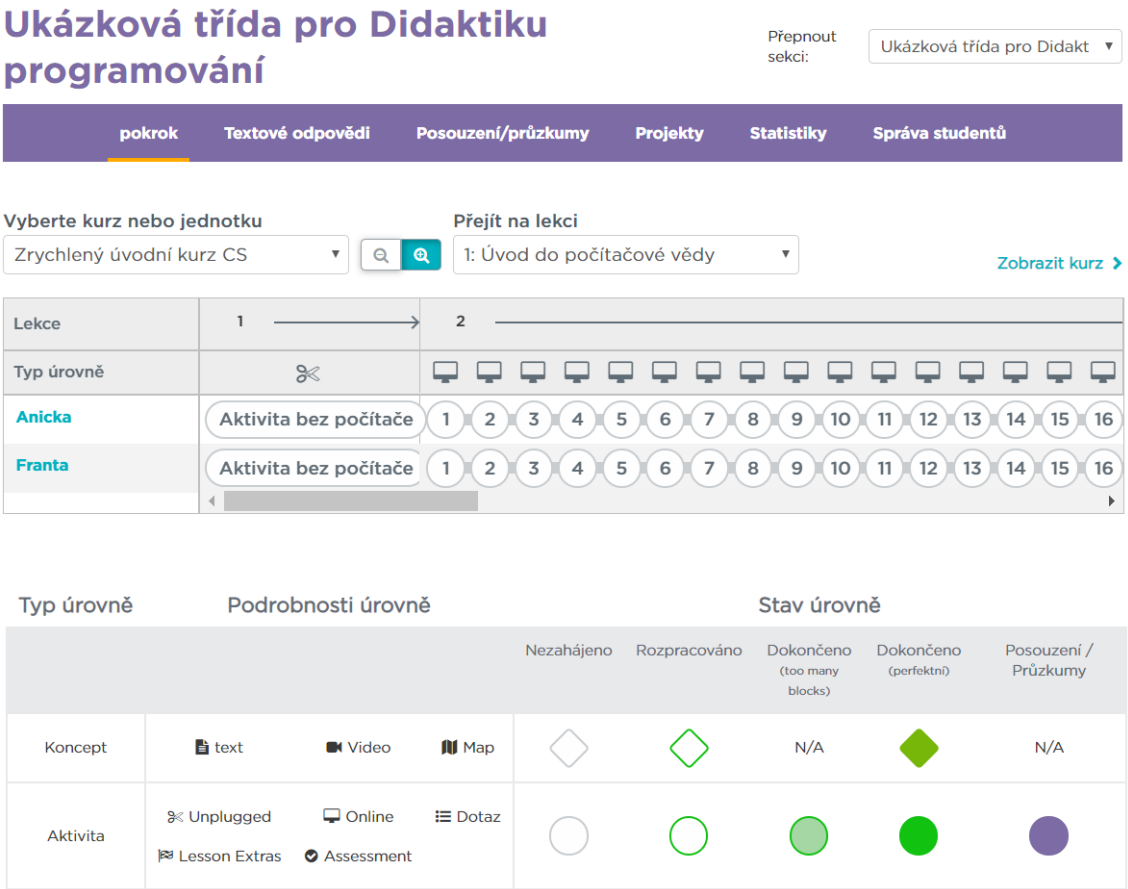
Nevýhodou práce bez registrace je absence ukládání průběžného postupu práce. Na konci každé hodiny se tak žákům smaže postup (získané zelené puntíky) a ačkoliv se dá začít pracovat z jakékoliv části/úlohy v kurzu, není to efektivní způsob. Žáci poté mají pocit, že začínají vždy od nuly a práce nemá smysl, když stačí vypnout prohlížeč a vše je pryč.

Registrace je zdarma pro každého a jednou z výhod pro žáky je právě **ukládání postupu práce**. Pro učitele je zde však podstatně významnější výhoda - **možnost vytvářet si třídy a sledovat postup jednotlivých žáků** v každé třídě.

V záložce *Můj přehled* si můžete **pomocí několika kliknutí založit novou třídu** (*vytvořit sekci*), kterou buď můžete jedním kliknutím propojit s vaší existující Google Classroom (používáte-li tento e-learningový systém), nebo můžete žákům vytvořit nové účty sami či jim umožnit vlastní registraci a připojení do vaší Code.org třídy pomocí šestipísmenného kódu. Poslední varianta je pravděpodobně nejlepší, avšak vyžaduje, aby žáci měli vlastní emailové adresy. Následně se po vás chce zadat název vaší sekce (třídy), který by měl být dostatečně jasný pro snadnou orientaci vaši i vašich žáků, například "*Informatika 8.B (2019/2020)*" a ano, školní rok je pro přehlednost důležitá položka názvu sekce. Následující položka *stupeň* odpovídá americkému vzdělávacímu systému, kde například možnost K-8 odpovídá žákům ve věku 14 let, čili zhruba 8. třídě základní školy v ČR. Poslední důležitou částí je výběr kurzu, kde jsou k dispozici všechny kurzy, které Code.org nabízí. Užitečná je i možnost vytvořit si kurz jen pro ukázkovou lekci Hour of Code, protože tvorba sekce trvá jen pár vteřin a možnost sledovat pokrok žáků i jen v HoC je velice užitečná. Tvorba kurzu navíc rovnou umožňuje práci ve dvojicích, kdy se splněné úkoly zapisují oběma spolupracujícím žákům do obou jejich účtů.

Všechny nabízené kurzy obsahují kromě úloh na počítači **také aktivity bez počítače**, které umožňují vysvětlit a procvičit algoritmizační principy a principy práce s počítačem například i v případě, že nepůjde elektrický proud. Jedná se o aktivizační metody, které vám také pomohou zpestřit celkovou hodinu a zvednout alespoň na chvíli žáky od počítačů. Po rozkliknutí takovéto aktivity mátě většinou k dispozici krátké motivační video a kompletní plán lekce včetně doprovodných aktivit a rozšiřujících otázek, který je ale vždy výhradně anglicky.

Na následujícím obrázku vidíte, jak vypadá vytvořená třída se dvěma ukázkovými žáky, kteří ještě nesplnili ani jednu úlohu. Při splnění úlohy se bílé kolečko změní ve světle zelené (splněno s výhradami) nebo tmavě zelené (splněno perfektně). Zde platí stejná rada jako u HoC - začne-li nějaký žák akumulovat velké množství světle zelených koleček, pravděpodobně nepochopil nějaký významný koncept a je třeba mu pomoci. Jako učitel si navíc můžete na vybranou úlohu nějakého žáka kliknout a zobrazí se vám konkrétní řešení, které daný žák použil.

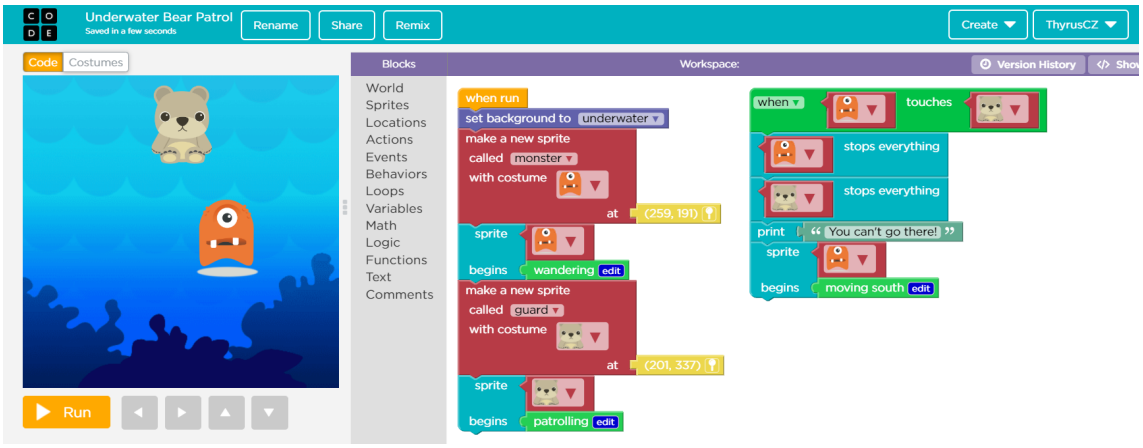


Obrázek 13: Zobrazení třídy z pohledu učitele a postup žáků v jednotlivých úlohách (zdroj: studio.code.org)

Celý kurz si jako učitel navíc můžete lehce upravovat, protože **máte možnost u každé lekce nastavit, jestli je v dané sekci (třídě) viditelná nebo ne**. Nevyhovuje-li vám například nějaká aktivita bez počítače nebo se vám nějaká lekce zdá repetitivní, můžete ji tímto způsobem jednoduše vyřadit.

Posledním velkým přínosem, který je pro učitele operujícího s českou verzí přinejmenším skrytý, je **kurz určený přímo pro učitele a budoucí učitele informatiky**. Tento kurz je totiž dostupný jen v anglické verzi a ještě se k němu musíte proklikat. Poté, co se přepnete do angličtiny, se vám nahoře na hlavní liště zobrazí nová záložka *Professional Learning*. Zde se přes odkaz *Not in the US? Learn more* dostanete na delší text, jehož součástí je i odkaz na *online workshop*. Nebo jednoduše zadáte https://studio.code.org/s/K5-OnlinePD?section_id=1713772, což je přímá URL adresa na zmiňovaný workshop. Veškeré informace zde jsou podány anglicky, ale vysvětlení jsou srozumitelná a obsahují mnoho příkladů. Necítíte-li se na výuku programování a algoritmizace a ovládáte-li anglický jazyk, jedná se o jednu z možností, jak se naučit více a připravit se na výuku vašich žáků.

Code.org navíc kromě tutoriálů a kurzů **umožňuje i práci hned v několika otevřených prostředích založených na Google Blockly**. Prostředí ***Artist*** je zaměřeno na želví grafiku (viz LOGO); ***AppLab*** je prostředí pro tvorbu vlastních aplikací v JavaScriptu (kód lze vytvářet pomocí bloků nebo psát ručně a je zde i možnost krokování a debuggování); ***SpriteLab*** je zjednodušené prostředí pro tvorbu kratších scének a her a ***GameLab*** je podstatně komplexnější verze SpriteLabu založená na práci s JavaScriptem. Na obrázku níže vidíte kratičkou ukázkovou scénku ve SpriteLabu s patrolujícím medvědem zahánějícím vandrující oranžovou slizku.



Obrázek 14: Ukázková scénka vytvořená ve SpriteLabu (zdroj: studio.code.org)

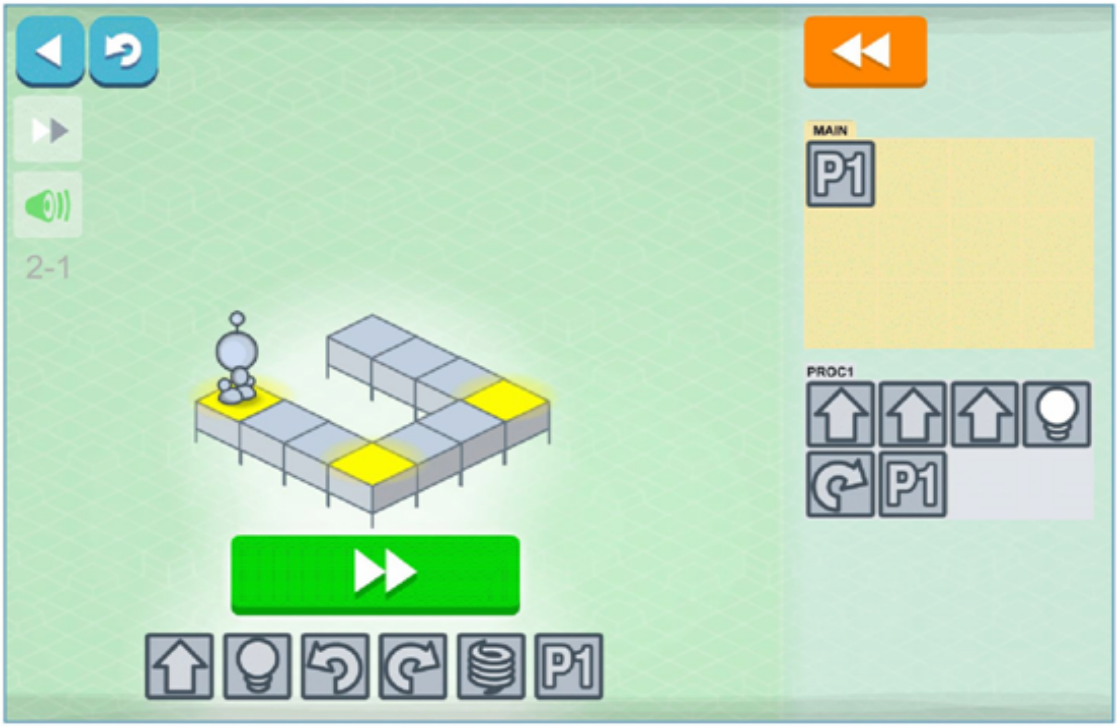
Využití v podstatě libovolného tutoriálu Hour of Code jako úplně prvotního setkání s problematikou programování a algoritmizace se v praxi osvědčilo v podstatě v jakékoliv věkové skupině od zhruba páté třídy až po vysokou školu. Nejenže jsou tyto lekce vysoce motivační, ale připraví také žáky pro další práci například v jazyce Scratch (nebo jakémkoliv jiném, založeném na Blockly).

1.5 Další projekty pro výuku programování

Ačkoliv jsou tutoriály projektu Hour of Code a kurzy na Code.org velice zdařilé, nejedná se o jedinou možnost. Situace je právě opačná - k dispozici máte desítky, možná až stovky různých projektů zaměřených na výuku programování, jejichž počet navíc každým rokem stoupá. Ve zbytku této kapitoly představíme několik vybraných projektů, které se ve výuce také osvědčily a jejichž zapojení je pro učitele zpravidla velice nenáročné.

1.5.1 Lightbot

Lightbot je logická hra vyvinutá v roce 2008, tedy pět let před samotným HoC. Cílem hráče je pomoci šedému robotkovi (který je přepnutelný na růžovou robotku) rozsvítit všechna modrá pole pomocí vytvoření sady instrukcí. Ačkoliv je základní premisa jednoduchá, se stoupajícími úrovněmi se obtížnost zvyšuje.



Obrázek 15: Vyřešená úroveň 2-1 v Hour of Code verzi logické hry Lightbot

Původně měla hra dvě placené verze, a to pro děti ve věku 4 až 8 let a pro děti starší osmi let. Obě tyto hry jsou standardizovány pro Apple iOS, Google Android a Kindle. Varianta pro starší děti má ještě verze pro Windows a Mac. Po nástupu Hour of Code byla vytvořena ještě třetí verze Lightbota, která funguje v podstatě jako demo, protože zahrnuje úvodní úrovně verze pro děti starší osmi let a je zakončená klasickým certifikátem HoC.[24] Tato hodinová „demo“ verze obsahující 20 úrovní je zdarma dostupná jako aplikace pro iOS a Android a také online pro webové prohlížeče podporující Flash. Plná zpoplatněná verze má celkem 50 úrovní.

24 LIGHTBOT INC. (2016). *Lightbot* [online]. Ontario (Kanada). Dostupné z: https://lightbot.com/flash.html

Lightbot byl do výzkumu v rámci diplomové práce zařazen (a od té doby používán i nadále) jako **rozšíření pro nejrychlejší žáky a pro demonstraci odlišnosti programovacích jazyků**. Žáci sami viděli, že přestože vše *vypadalo* jinak než HoC, podstata jejich práce byla takřka totožná. Této zkušenosti poté bylo využito k velice zjednodušenému vysvětlení rozdílů mezi například Javou a C# a dalšími jazyky.

V kombinaci s původní verzí Hour of Code z roku 2013 se žáci seznámí s relativně velkým množstvím různých programovacích prvků. Žáci by měli chápat princip programování ve vizuálním programovacím prostředí, znát základní cykly a podmínky a přidá-li se i Lightbot, znají i **princip podprogramů**. Jako ukázka na různé kratší kurzy a propagační akce je tato kombinace HoC s Lightbotem osvědčená, ale nelze počítat s tím, že by všichni žáci byli schopni využívat vše, s čím se v takovéto lekci setkali. Lepší je považovat takovouto úvodní hodinu za čistě motivační a navázat výukou od začátku.

1.5.2 CodeCombat

Zatímco doposud zmiňované projekty byly založeny na možnosti programovat pomocí skládání grafických bloků, projekt CodeCombat zastává diametrálně odlišný přístup. Tvůrci tohoto projektu v rámci příspěvku *3 důvody proč „Počítačová gramotnost" kazí výuku kódování*, zveřejněném na dedikovaném blogu, uznávají, že vizuální programování skutečně je vhodné pro malé děti, avšak tvrdí, že starší a dospělí už z něj nic nezískají. Ještě horší je podle nich fakt, že vydává-li se takováto výuka za programování, reálné programování potom vypadá nudně a komplikovaně a žáci nemají dostatečně osvojeny důležité programovací návyky.[25]

Obrázek 16: Rozložení herního okna v programovacím prostředí CodeCombat (mise na výuku komentářů)

25 SAINES, George. (2014). 3 Reasons Why „Computational Literacy" Is Ruining Coding Education. In: *CodeCombat Blog* [online]. San Francisco, California, 30 September 2014 [cit. 2019-08-22]. Dostupné z: https://blog.codecombat.com/3-reasons-why-computational-literacy-is-ruining-coding-education/

CodeCombat se zaměřuje na výuku skutečného programování za využití skutečných programovacích jazyků Python a JavaScript, včetně všech formálních pravidel syntaxe. Vše stále probíhá formou hry, přičemž se jedná o jedno z mála skutečně povedených plně gamifikovaných prostředí.

Na obrázku 16 je vidět rozložení herního okna, ve kterém hráč pomocí zápisu kódu na pravé straně obrazovky ovládá svého vybraného avatara (a potažmo jeho vojáky) na straně levé. Hra je rozdělena do pěti velkých oblastí a každá oblast do dílčích úkolů, jejichž splnění může trvat pouhou minutu, ale také hodinu i déle. Za splnění úkolu je hráč odměněn krystaly, za které si pořizuje nové vybavení pro svého avatara. Tyto krystaly se získávají i v rámci achievementů a lze je nakoupit v rámci mikrotransakčního modelu hry.

Obrázek 17: Inventář hrdiny CodeCombat na vyšší úrovni

Vybavení určuje klasické množství životů, obranu a útok, ale také co všechno avatar může provést, co všechno umí. v **podstatě se tedy jedná o knihovny s předdefinovanými metodami**. Uprostřed obrázku 17 je vidět seznam dostupných metod, u kterých se po najetí myši zobrazí plnohodnotná programová dokumentace. s novým vybavením se objevují nové metody a hráč se kromě klasického základu spočívajícího v cyklech, podmínkách atp. postupně učí například i plnohodnotnou tvorbu funkcí a jejich volání. Motivace hráčů je dále podněcována systémem achievementů a zvyšováním úrovně postavy.

Pro hraní je nutná registrace, která je sice zdarma, ale zároveň neobsahuje všechny úrovně a nabízí jen čtyři hrdiny (po dvou od každého pohlaví). Pro odemčení dalších úrovní je vyžadována měsíční platba. Ta ale není úplně nutná, protože v podstatě celá hra se dá „dohrát“ i zdarma. Placené úrovně jsou totiž namíchány s úrovněmi zdarma a lze na ně tak pohlížet jako na určité rozšíření či prohlubování učiva, jehož vynechání nijak nebrání dalšímu postupu hrou.

Z hlediska placené verze hry je největší problém v tom, že **hru nestačí jednou zakoupit, ale je nutné platit si relativně vysoký měsíční poplatek**. Na druhou stranu je v současné době již 330 placených úrovní, ke kterým nemá hráč hrající zdarma vůbec přístup. Pro nasazení v rámci výuky je však základní verze bez poplatků pro žáky naprosto postačující.

Co se týče jazykové verze, **základ je samozřejmě anglicky. Český překlad je implementován**, avšak nové úrovně neustále přibývají a jejich překlady se objevují nepravidelně. Nelze tedy počítat s tím, že žáci nenarazí na mnohdy velice složité úrovně celé pouze anglicky. Zadání je zpravidla jasné a velice jednoduché, objeví-li se ale úplně nový programovací koncept, jehož vysvětlení je pouze anglicky, žáci budou nejspíše potřebovat vaši pomoc.

Hra je logicky strukturována od nejjednodušších částečně před-připravených konstrukcí, do kterých stačí jen doplnit chybějící příkazy, až po naprosto volné úrovně, kdy hráč sám vytváří celý, zpravidla značně složitý zápis všech příkazů pro svého avatara. z didaktického hlediska **lze vytknout chybějící přehledné shrnutí toho, co bylo kde probráno a případně i ucelené teoretické shrnutí nové problematiky**, které by se mohlo nacházet například na konci každé z pěti velkých oblastí hry. i přes tyto značně citelné nedostatky je hra koncipována velice dobře.

Po průchodu všech volně dostupných úrovní jsem měl pocit, že jsem si vlastně jen hrál a nic moc se nenaučil, avšak při následném pročítání interaktivní učebnice *JavaScript Tutorial* (w3schools.com 2016) jsem si opakovaně uvědomoval: *„...to už umím, to taky, tohle taky...“* Mezi žáky má hra také velký ohlas, přičemž dostatečným důkazem je, že ji hrají i sami ve svém volném čase.

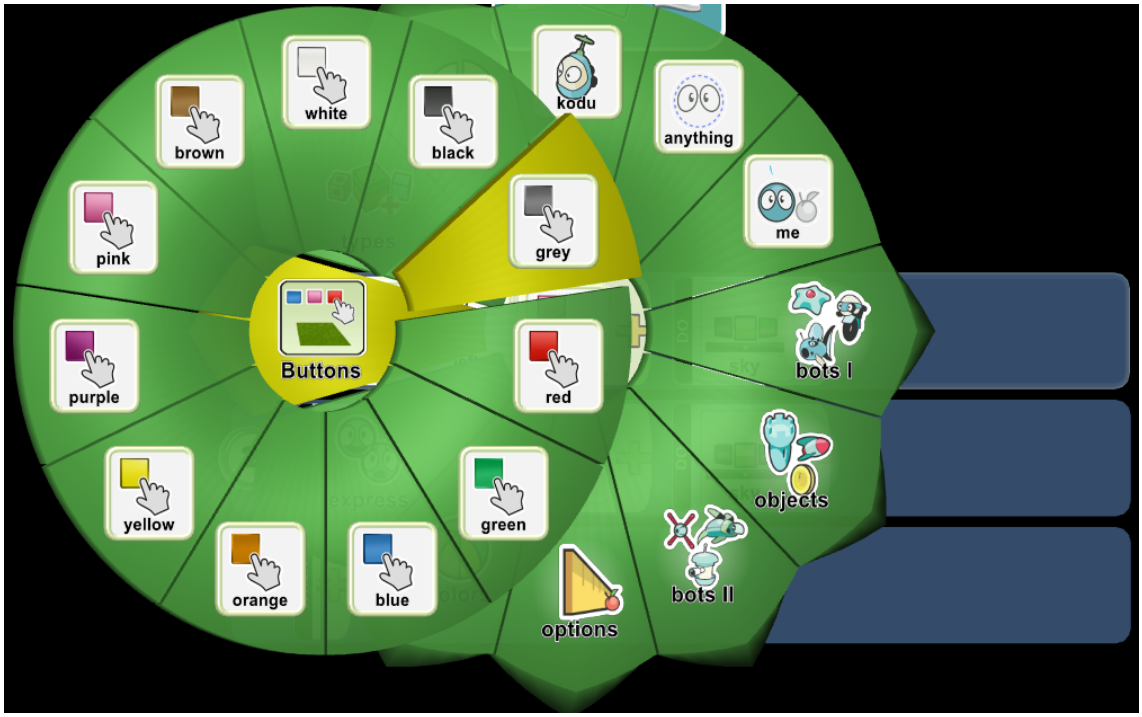
1.5.3 Kodu

Hlavním vzdělávacím projektem společnosti Microsoft je v současné době Minecraft: Education Edition.[26] Bohužel se ale jedná o komerční produkt a verze zdarma je omezená na 25 loginů pro učitele a 10 pro žáka. Placená verze se poté kupuje jako rozšíření existující licence Office 365 a stojí $5 USD na počítač/rok. Ačkoliv projekt vypadá velice zajímavě, nabídka vzdělávacích materiálů zdarma je dnes již dostatečně široká, abyste si jako budoucí učitelé nemuseli přidělávat práci se zakupováním licencí a s tím spojenou administrací. Všechny projekty a materiály představené v těchto skriptech jsou zdarma (s výjimkou Ozobotů a LEGO Mindstorms, kde se cena odvíjí od materiálních pomůcek) a výběr je velký.

Microsoft však již v minulosti vytvořil projekt pro výuku algoritmizace a programování, který je také kompletně zdarma a stále pro něj vycházejí nové aktualizace. Jedná se o **KODU Game Lab**, zajímavý počin původně určený pro Xbox, pro který byl také v roce 2009 vytvořen. První Windows verze vznikla v roce 2010 a v současné době je také jedinou dostupnou verzí, protože podpora Xbox skončila

26 Dostupné na https://education.minecraft.net/

v roce 2017. Nelze však popisovat KODU bez návratu k jeho kořenům, protože celé prostředí muselo být přizpůsobeno ovládání pomocí Xbox ovládače (viz obrázek 18) a pro využití na Windows nebylo nijak extra modifikováno. Výsledkem tak je prostředí, jehož ovládání je na Windows lehce neohrabané a méně intuitivní než ve většině ostatních vzdělávacích programovacích jazyků.



Obrázek 18: Ovládání KODU pomocí vějířovitého uživatelského rozhraní (zdroj: https://www.kodugamelab.com/resources/tips-and-tricks/)

Oproti většině projektů zmíněných v těchto skriptech je také třeba **počítat s komplikací spojenou s nutností instalace**.[27] Tato „komplikace“ se ale může ukázat být výhodou, nastane-li situace, že vám nepůjde internet, ale elektrický proud ano. Drtivá většina online projektů totiž nemá stažitelnou offline verzi.

KODU je plně otevřené programovací prostředí, ve kterém žáci mohou tvořit vlastní hry, hlavolamy, bludiště, scénky, apod. Těmto výtvorům se zde říká *světy*. **Pro začátečníky jsou hned po nainstalování k dispozici *tréninkové světy***, které fungují jako tutoriál pro práci s prostředím. Uživatelé si dle návodu mohou tvořit vlastní tutoriálové světy a jeden ukázkový kurz je zdarma k dispozici po vyplnění krátkého formuláře, například od českého projektu Akademie programování.[28]

Ve hře zpravidla hráč ovládá jednoduchého levitujícího robotka jménem Kodu, který se buď řídí naprogramovanými instrukcemi, nebo lze naprogramovat na přímé ovládání hráčem (viz obrázek 19). Herní svět vypadá hezky a v podstatě veškeré jeho mechaniky záleží přímo na žácích-programátorech, kteří jsou limitováni jen existujícími postavičkami.

27 Zdarma ke stažení zde: https://www.microsoft.com/en-us/download/details.aspx?id=10056

28 Kurz je dostupný v nabídce zde: https://www.akademieprogramovani.cz/kodovani-na-doma/

Obrázek 19: Jak vypadá jednoduchý herní svět KODU

(zdroj: http://blog.ikizoglu.com/2017/12/kodu-game-lab-nedir-nasil-kullanilir/)

Jedním z hlavních důvodů, proč KODU do výuky programování zařadit, je **možnost zapojení se do soutěže KODUCup**. Soutěže jsou pro žáky silným motivačním prvkem, přičemž v současné době je programovacích soutěží vhodných pro žáky základních škol velmi málo a KODUCup tak nelze přehlédnout. Ve slovenské verzi proběhl v roce 2018 již čtvrtý ročník této soutěže[29] a v **Čechách byl v roce 2018 ročník třetí**[30]

Jakmile si zvyknete na nezvyklé ovládání, prostředí je použitelné bez větších problémů, KODU si žáci přesto příliš neoblíbili. Autorem skript byl tento jazyk využit jako rozšíření pro rychlé žáky, kteří si po vypracování zadaných úkolů mohli volně pracovat v KODU. Ačkoliv si žáci díky příslibu možnosti zúčastnit se soutěže KODUCup tento jazyk vyzkoušeli, žádný z žáků v něm nechtěl pracovat dlouhodobě. Nejčastější výtkou bylo pomalé vytváření kódu a velká nepřehlednost při delších úsecích kódu.

1.5.4 Blockly

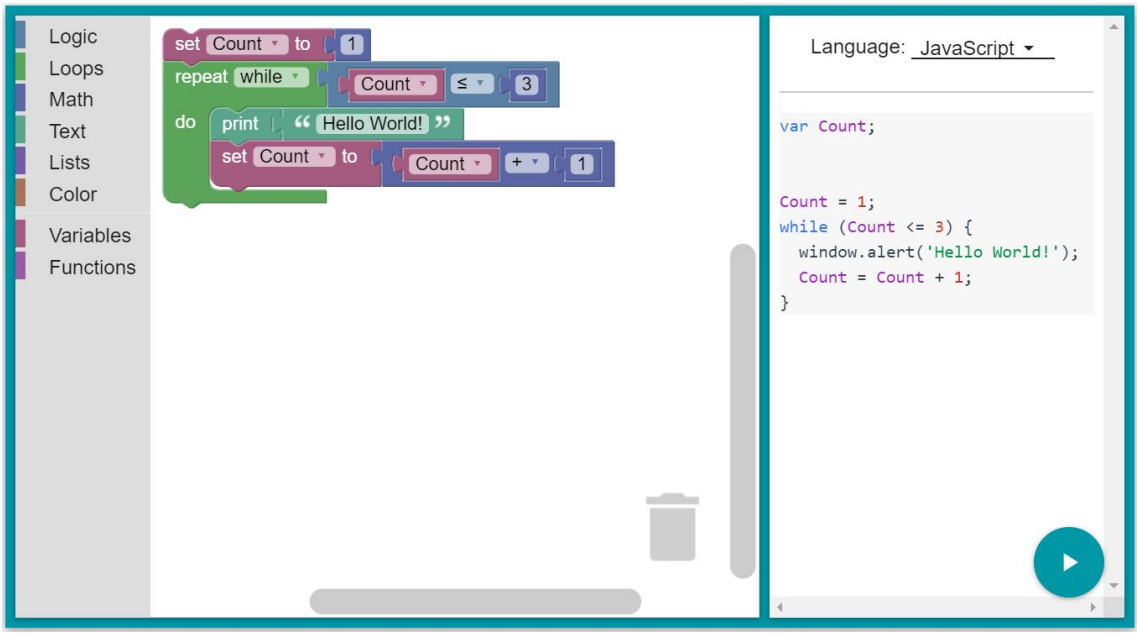
Posledním zde zmíněným projektem je Google Blockly (ale ve zbytku skript se setkáte ještě s projekty Scratch, Snap, Ozobot, LEGO Mindstorms, xLOGO, TurtleAcademy a Python). Blockly je však od ostatních vzdělávacích jazyků **diametrálně odlišný**. Google sice vytvořil obdobný tutoriál jménem Blockly Games[31] a mnoho lidí si pod názvem Blockly představí právě „něco jako Scratch," což ale není vůbec přesné. **Google definuje Blockly jako** *„knihovnu, která přidává*

29 MOLČANOVÁ, Zuzana. (2018). Kodu Cup 2017/2018 pozná svojich víťazov. In: *Blog inovatívnych učiteľov a študentov: Vzdelávame pre budúcnosť* [online]. Bratislava, ©2016, 11. júna 2018 [cit. 2019-08-22]. Dostupné z: http://blog.vzdelavameprebuducnost.sk/sutaz/kodu/kodu-cup-2017-2018-pozna-svojich-vitazov/

30 *Kodu Cup Česko.* [online]. ©2019 [cit. 2019-08-01]. Dostupné z: http://www.koducup.cz

vizuální editor kódu na web nebo do mobilních aplikací. […] z pohledu vývojáře je Blockly před-připravené uživatelské rozhraní pro tvorbu vizuálních jazyků, které generuje syntakticky správný, uživatelem vytvořený kód.“ [32]

Kód, který uživatel z Blockly kostiček postaví, potom Blockly vyexportuje do vybraného programovacího jazyka (Javascript, Python, PHP, LUA nebo Dart). **Samo o sobě je ale Blockly jen sada kostiček, pracovní plochy (kde se kostičky spojují) a generátoru. Smysl Blockly je toto pracovní prostředí přidat do nějaké aplikace nebo na web**, čímž rozšíříme tuto naši aplikaci či web o možnost tvorby kódu ve vizuálním stylu. Funkcionalita, která je v Blockly defaultně obsažená, je veškerá práce s podmínkami, cykly, matematickými a logickými operátory, proměnnými, seznamy a texty. Uživatel si také může vytvářet vlastní funkce z dostupných bloků.



Obrázek 20: Programovací prostředí Google Blockly (zdroj: https://developers.google.com/blockly/)

V praxi potom práce s Blockly vypadá následovně - chceme-li například vytvořit něco jako Scratch, **musíme připravit plnohodnotnou webovou stránku včetně kódu (třeba v Javascriptu)**, který bude ovládat veškerou grafiku v herním okně (vykreslování a pohyb postavičky, fyzikální engine, kontrolu kolizí, atp.). **Teprve poté si sami vytvoříme nové Blockly bloky**, kterým přidáme např. námi vytvořený kód pro posun postavičky po obrazovce nebo kód pro změnu vzhledu.

Takováto práce již vyžaduje zkušeného programátora a pro využití v rámci běžné výuky je tak naprosto nevhodná[33] (přičemž teď je řeč o Blockly jako

31 *Blockly Games: Games for tomorrow's programmers.* [online]. © Google. [cit. 2019-07-10]. Dostupné z: https://blockly-games.appspot.com/

32 Introduction to Blockly. *Google Developers: Build anything with Google.* [online]. © Google. [cit. 2019-08-18]. Dostupné z: https://developers.google.com/blockly/guides/overview

33 FRASER, Neil. *Blockly*. [online]. California [cit. 2019-07-12]. Dostupné z: https://neil.fraser.name/blockly/

takovém, ne o výše zmiňovaném tutoriálu Blockly Games, který je využitelný bez problémů). Blockly je ale vhodné například na zájmové útvary zaměřené přímo na programování nebo na informaticky zaměřené střední školy, kde lze Blockly využívat například při práci s Arduinem nebo Raspberry Pi.

Význam Blockly spočívá také v tom, že se jedná o základ mnoha velkých projektů, jako je například Scratch nebo Hour of Code.[34]

Zajímavým projektem je také **NOVA LABS – Cyber Security**[35], kde je část úkolů pro žáky plněná právě v prostředí Blockly a kde se žáci naučí nejen základy algoritmizace, ale zejména kyber bezpečnosti (jak vypadají útoky na internetu, co je phishing, jak má vypadat správné heslo, apod.) Samostatnou kapitolu tento projekt nedostal a dále se zde jím již nebudeme zabývat, protože je pouze v angličtině a obsahuje velké množství složitějšího textu.

Obecně lze tvrdit, že základní premisou Blockly je možnost vytvářet kód skládáním grafických bloků a tak se vyhnout syntaktickým problémům spojeným s psaním textu, což je také jeden z důvodů jeho častého využití ve vzdělávacích programovacích jazycích. Zkušení programátoři ale dokáží Blockly zakomponovat i do zajímavějších projektů úplně mimo oblast vzdělávání, jako je např. open source projekt Domoticz[36] pro automatizaci domácích systémů (tvorbu smart home).

Závěrečné shrnutí kapitoly 1

Po přečtení této kapitoly byste měli:

- ✔ mít zopakované základní termíny z oblasti algoritmizace a programování včetně různých druhů zápisů algoritmů;
- ✔ dokázat vysvětlit a obhájit proč a jak učit programování;
- ✔ vědět, kde hledat vhodné vzdělávací materiály a projekty;
- ✔ nalézt vhodný projekt pro výuku vašich žáků a analyzovat jeho klady, zápory a použitelnost v porovnání s jinými;
- ✔ pohodlně připravit a zrealizovat ukázkovou Hour of Code;
- ✔ vytvořit si vlastní třídu na Code.org a pokračovat ve výuce;
- ✔ vědět, že Google Blockly skutečně (většinou) není pro děti.

34 APONE, Katie. (2019). Why does Code.org use Blockly, a visual programming language, for its elementary-level courses?. In: *Code* [online]. Seattle, July 25, 2019 [cit. 2019-08-18]. Dostupné z: https://support.code.org/hc/en-us/articles/202518363-Why-does-Code-org-use-Blockly-a-visual-programming-language-for-its-elementary-level-courses-

35 Volně dostupné na adrese https://www.pbs.org/wgbh/nova/labs/lab/cyber/

36 Blockly. (2018). In: *Domoticz: Control at your finger tips.* [online]. ©2012-2018, 28 February 2018 [cit. 2019-08]. Dostupné z: https://www.domoticz.com/wiki/Blockly

Samostatná práce (část 1)

Úlohy pro samostatnou práci jsou v této lekci odlišné od všech následujících. Jejich cílem není procvičit si různé aspekty programovacího jazyka, ale zamyslet se nad celkovou problematikou. Vypracujete-li následující úlohy, může se vám podařit například zjistit dopad Hour of Code z vědecké stránky, či jak je vnímán z hlediska společnosti, připravit si argumentaci pro vaše budoucí kolegy či vedení stran výuky programování na vaší škole, nebo si ujasnit v čem je nástroj, s kterým již umíte pracovat, nějak výjimečný.

a) Nalezněte na internetu **delší článek či vědeckou studii na téma projektu Hour of Code**. Článek si přečtěte a napište **shrnutí o rozsahu 150 až 200 slov**. Článek může být česky, ale i v libovolném jiném světovém jazyce. Nezapomeňte do shrnutí přiložit odkaz na Vámi vybraný článek.

b) Napište **tři důvody pro plošné zařazení výuky programování** na nespecializovaných školách (základní školy, gymnázia a další netechnické školy) a **tři důvody proti**. Své názory rozveďte a podložte jasnou argumentací. Celkový rozsah textu by měl být cca 150 až 200 slov.

c) Znáte-li nějaký **projekt zaměřený na výuku programování, který v rámci hodiny nebyl zmíněn** a se kterým jste již pracovali, umíte pracovat a nebo jste s ním alespoň trochu obeznámeni, popište tento projekt (spolu s odkazem) v rozsahu cca 150 až 200 slov. Byl-li projekt zmíněn, ale vy s ním máte větší zkušenosti, můžete ho využít také a doplnit právě vaše zjištění a zajímavosti, které v hodině a těchto skriptech nezazněly.

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

2 Vizuální programování Scratch

V čem se liší Hour of Code a Scratch?

Na jaké technologii je založen Scratch a jaké jsou alternativy?

Jaký je princip práce v prostředí Scratch a jaké jsou jeho výhody?

Jakým vývojem a jakými změnami prošel Scratch v průběhu let?

Jaký je rozdíl mezi postavou a kostýmem?

Jaké projekty mohou žáci či studenti vytvářet v prostředí Scratch?

Jak rozšířit zadání scénky či hry pro rychlé žáky? Uveďte příklad.

Na jakých příkladech lze demonstrovat využití složitějších konstrukcí a datových typů (proměnné, seznamy, vlastní funkce)?

Co je to klonování a jaké programovací principy ilustruje?

2.1 Seznámení s prostředím a skriptované scénky

Scratch je programovací jazyk a prostředí **vyvinuté na Massachusetts Institute of Technology (MIT) v roce 2007 skupinou *LifeLong Kindergarten*** (volně přeloženo *„Celý život ve školce“*), pod vedením Mitchela Resnicka. Vychází z podstatně staršího vzdělávacího jazyka LOGO, který vznikl taktéž na MIT již v roce 1967 a se kterým se v jeho podobě xLOGO seznámíte v samém závěru těchto skript, jakožto jedním z možných nástrojů pro úvod do textového programování.

Scratch oproti svému předchůdci LOGO přináší celou řadu nových prvků, intuitivnost ovládání a tvorby programů, samozřejmou vícevláknovost, přirozenou objektovost atd. **Veškerá tvorba kódů je zde založena na grafickém prostředí Google Blockly**, které automaticky eliminuje syntaktické problémy, a které je využíváno ve velkém množství dalších vzdělávacích projektů, takže **přechod od jednoho prostředí založeného na Blockly k jinému je naprosto přirozený**. Tento fakt také nahrává kombinování různých projektů, jež je využito i v rámci těchto skript.

2.1.1 Technologie a změny v prostředí

Tvůrci Scratche však od jeho vytvoření nezaháleli a **jazyk i prostředí podléhá neustálému vývoji a modernizac**i. Ve své úplně původní verzi popisované ve třetí části skript doktora Musílka[37] byl Scratch značně zjednodušen a tím také poněkud omezen, protože neumožňoval např. práci s podprogramy a funkcemi (tedy vytváření vlastních nových bloků kódu) a tím i vyřazoval možnost použití rekurzivních algoritmů. v reakci na toto výrazné omezení tak vznikl **konkurenční projekt BYOB na univerzitě Berkeley**, přičemž slovo konkurenční zde není úplně přesné. Tvůrci projektu přiznávají Scratchi plné zásluhy, ale sami uvádějí, že *„Naše dřívější verze BYOB byla přímou modifikací zdrojového kódu Scratch. Snap! Je kompletně přepracovaný, ale struktura kódu a uživatelské rozhraní zůstávají hluboce dlužny Scratchi.“* [38] Tuto hluboce zakořeněnou podobu můžete vidět na obrázku 21 znázorňujícím jak vypadal původní Scratch ve stažitelné offline verzi 1.4 a jak stále vypadá aktuální BYOB (respektive nyní již Snap! Verze 5.1.0).

BYOB byl původní název tohoto projektu, jehož význam je *Build Your Own Blocks*, tj. vytvoř své vlastní bloky (procedury a funkce, včetně rekurzivních), takže už název prostředí vyzdvihoval hlavní výhodu tohoto programu nad původní verzí Scratche. Od verze 4.0 byl **BYOB přejmenován na Snap!**, přičemž jeden z důvodů bylo pravděpodobně ne úplně přesné využívání plného názvu mezi žáky a studenty, kteří projektu přezdívali *Bring/Buy Your Own Beer*. Je však nutno podotknout, že původnímu vývojáři Jensi Mönigovi tato změna názvu na Snap! již od samého počátku nevyhovovala.[39] v dokumentaci citované výše se můžete dočíst o tom, v čem se Snap! od Scratche liší. Většinou se jedná o rozšířenou funkcionalitu, z které můžeme jmenovat například komplexní práci se sprity (2D grafickými obrázky), kdy je možné jeden sprite využít jako kotvu na kterou se dají „nabalovat“ další části výsledného obrázku (např. oblečení na postavičku).

Scratch nezůstal dlouho pozadu a vývojáři z MIT vytvořili nejprve rozšíření (plug-in) přidávající možnost tvorby vlastních bloků i ve Scratchi a poté tuto funkcionalitu zabudovali přímo do jádra projektu Scratch. v současné době tak můžeme využívat tvorbu vlastních bloků bez nejmenších komplikací i ve Scratchi.

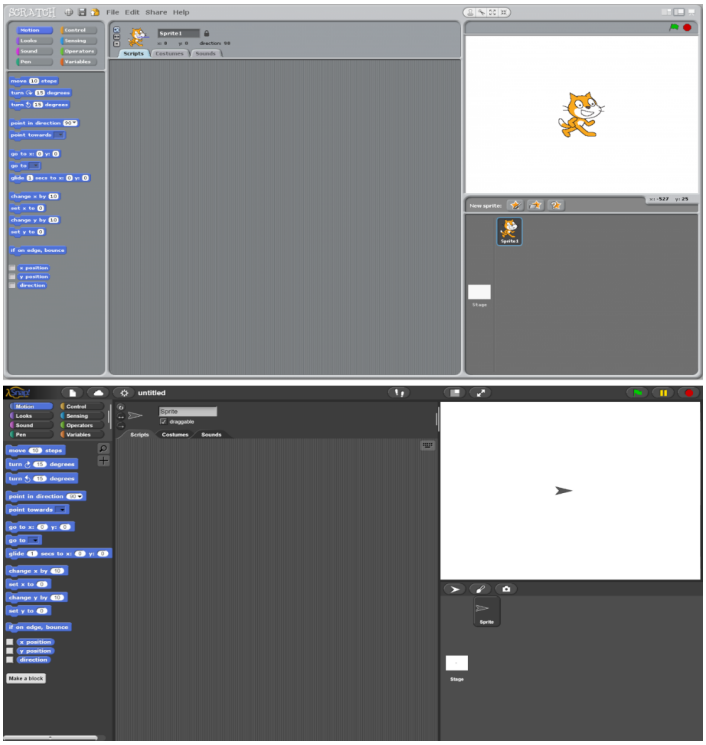
Druhým **významným limitujícím faktorem jazyku Scratch byla jeho závislost na nyní již zastaralé technologii Flash**, kvůli čemuž nebylo možné využívat online prostředí na mobilních dotykových zařízeních.

---

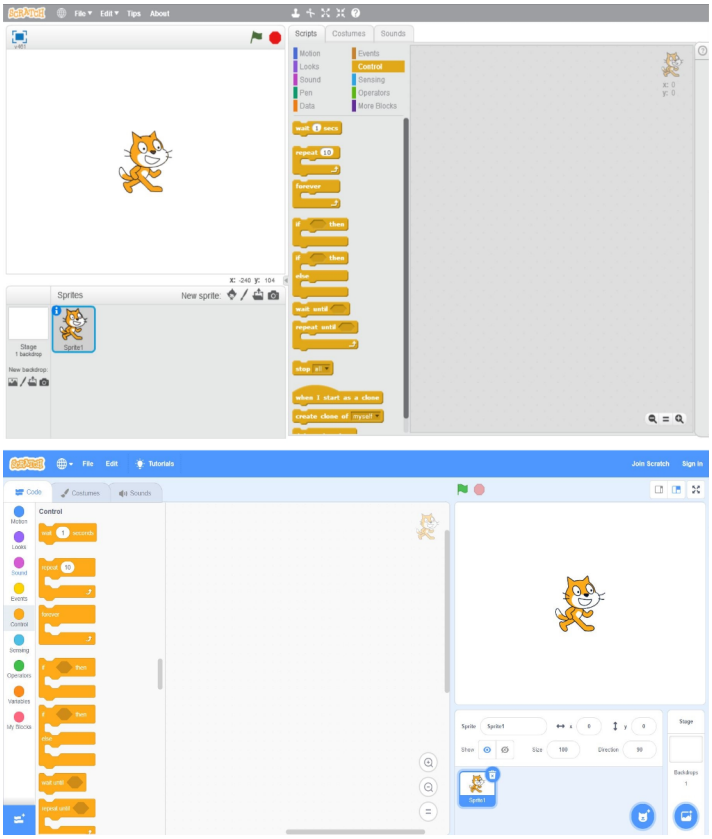
37 MUSÍLEK, Michal. (2012). *Dětské programovací jazyky – Programovací jazyk SCRATCH*. Univerzita Hradec Králové, 27 s. Scriptum. Dostupné z: http://www.musilek.eu/michal/pdf/deproja3.pdf

38 HARVEY, Brian & MÖNIG, Jens. (2017). *Snap! Reference Manual.* 124 s. Dostupné z: https://cloud.snap.berkeley.edu/ snapsource/help/SnapManual.pdf

39 NATHAN. (2014). Rename repository #300. In: *GitHub* [online]. ©2019, 11 Jan 2014 [cit. 2019-08-19]. Dostupné z: https://github.com/jmoenig/Snap/issues/300



Obrázek 21: Nahoře Scratch 1.4 v porovnání se Snap! 5.1.0 (nejaktuálnější verze BYOB)



Obrázek 22: Nahoře Scratch 2.0 v porovnání s nejnovějším Scratch 3.0

Tento nedostatek už však nadále neplatí, protože **nová verze Scratch 3.0 publikovaná v lednu 2019[40] je již plně založena na technologii HTML5**, která je podporovaná všemi moderními prohlížeči, včetně mobilních. Spolu s tím souvisí výrazná změna v rozměrech jednotlivých bloků, které jsou nyní vyšší právě z důvodu zjednodušení manipulace na dotykových zařízeních. Prohození pracovní plochy a herní scény lze pak okomentovat jako návrat ke kořenům.

Tvůrci Scratche sice vytvořili určité návody a postupy pro seznámení s jejich programovacím prostředím, ale česká lokalizace je často většinou špatná nebo úplně chybí, takže je jejich využití přímo pro výuku v běžných hodinách informatiky na základních školách omezené. Jelikož je Scratch založen na Blockly, je tedy pro učitele **výhodné pro první seznámení využít nějaké uzavřené tutoriálové prostředí typu Hour of Code nebo Blockly Games**. Tato prostředí si jsou všechna velice podobná a zkušenosti s jejich ovládáním jsou proto plně přenositelné. Ze stejného důvodu je za touto kapitolou zařazen minirobot jménem Ozobot, kterého lze (mimo jiné) ovládat v prostředí OzoBlockly. Takovéto řetězení prostředí umožňuje efektivnější a plynulejší práci v časově omezených hodinách IT.

2.1.2 Vývojové prostředí Scratch

Stránky projektu Scratch najdete na adrese http://scratch.mit.edu/ a kromě možnosti downloadu vývojového prostředí[41] zde najdete také obrovskou databázi hotových programů a možnost zapojení se do komunity uživatelů, nebo do komunity učitelů používajících Scratch při výuce programování.

Pro plnohodnotnou práci v tomto prostředí **není nutná registrace**, ale v takovém případě je potřeba si vytvořené či rozpracované programy stahovat do počítače. v případě registrace se všechny vaše programy ukládají na váš Scratch účet, kde je můžete i publikovat a nechat si je okomentovat ostatními uživateli.

Podíváme-li se na vývojové prostředí Scratch, zjistíme, že **lze celé přepnout na českou jazykovou verzi**. Původně bývaly postavičky označovány anglickým slovíčkem *sprite* (doslovný překlad by byl skřítek, či víla, ačkoliv se tento výraz do češtiny nepřekládá, protože se jedná o termín označující 2D obrázky ve hrách), v poslední verzi již je ale i tento název upraven na české „postava.“

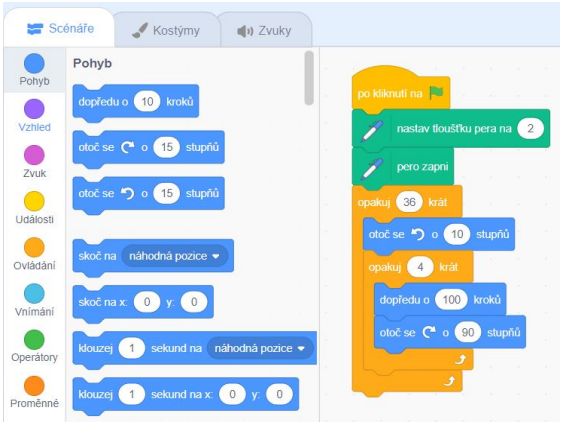
Program se sestaví podobně jako kostky dětské stavebnice, **jednotlivé příkazy a řídící struktury do sebe zapadají**. Program se vytváří jednoduchým přetahováním myší z nabídky příkazů vlevo do pole skriptů uprostřed. Velice široká paleta příkazů je zde již předem připravená a graficky znázorněná. Žáci se tedy nejen nemusí učit syntaxi programovacího jazyka, ale při takovéto tvorbě

40 CEEBEE. (2019). Scratch 3.0 is here!. In: *Scratch: Discuss Scratch* [online]. ©2019, Jan. 2, 2019 [cit. 2019-09-09]. Dostupné z: https://scratch.mit.edu/discuss/topic/326861/

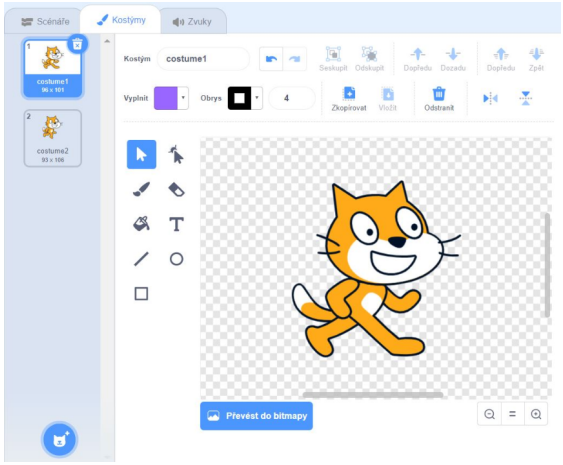
41 Scratch Desktop: Offline Editor. In: *Scratch* [online]. ©2019, Aug. 30, 2019 [cit. 2019-09-09]. Dostupné z: https://scratch.mit.edu/download

programů **zároveň vytvářejí i takzvané strukturogramy** (varianta k vývojovým diagramům zmiňovaným v první kapitole a ke kopenogramům, které budou blíže vysvětleny v kapitole o jazyce Karel).

Prostřední sloupec slouží kromě psaní *scénářů*, ovlivňujících chování určité postavy také ke změně jejího vzhledu, nebo jednoduché animaci na záložce *kostýmy*, případně k vytváření zvukových projevů postavy na záložce *zvuky*. Například maskot jazyka Scratch, oranžový kocour, má po přidání rovnou dva různé kostýmy (chůze, běh) a umí mňoukat.



Obrázek 23: Záložka "scénáře"



Obrázek 24: Záložka "kostýmy"

Sloupec vpravo má v horní části *scénu*, tj. oblast, ve které se zobrazují výsledky vašeho programu. **Nad scénou jsou základní ovládací prvky**: zelený startovní praporek a červené kulaté tlačítko stop. Ještě o kousek výš jsou tlačítka umožňující přepnout prostředí na malou scénu, na plnou scénu (výchozí velikost) a do tzv. modu prezentace, kdy scéna zabírá téměř celou plochu obrazovky. Zpět z modu prezentace se dostaneme zcela intuitivně – stiskem tlačítka ESC. Ve spodní části pravého sloupce (pod scénou) najdeme tlačítka pro přidávání nových postav (*sprit*ů). Každá postava pak „žije svým vlastním životem“, tj. programování je více-vláknové. Kliknutím na ikonku můžeme vybrat a upravovat jednotlivý *sprite* (postavu), nebo scénu a měnit její chování (naprogramování), vzhled a zvuky. Vzhled postavy nazýváme kostým, vzhled scény pak pozadí.

2.1.3 Pohyb, vzhled a ovládání spritu, želví geometrie

Spustíme-li kód znázorněný na obrázku 23 a 25, výsledkem bude náš kocour Scratch stojící před květem ze čtverců. Tento druh kreslení budeme využívat velice často, protože se jedná o takzvanou želví grafiku nebo želví geometrii z programovacího jazyka LOGO (jehož maskotem je právě želva). Se stejným programem se tak občas setkáte ve Scratchi i v LOGO. Algoritmy pro kreslení takovýchto obrazců zůstávají stejné, ale ve Scratchi takový program sestavíme ještě

snáze. Jednak díky českým příkazům, jednak díky intuitivnímu stavebnicovému skládání z „kostek“. To, že jednotlivé příkazy jazyka není potřeba zapisovat, zcela eliminuje možnost syntaktických chyb. Žák se díky tomu může plně soustředit na myšlenku algoritmu a jeho sémantickou správnost, což činí z prostředí Scratch ideální nástroj pro výuku programování už na prvním stupni základní školy.

```
po kliknutí na startovní praporek
    nastav tloušťku pera na 2
    pero dolů
    opakuj 36 krát
        otoč se vlevo o 10°
        opakuj 4 krát
            posuň se o 100 kroků
            otoč se vpravo o 90°
zastav scénář
```

Obrázek 25: Ukázka scénáře pro kreslení květu ze čtverců

Výsledný algoritmus můžeme zapsat pomocí nápisů na jednotlivých „kostkách“. Pro přehlednost odsadíme řádky, které představují příkazy vnořené dovnitř řídícího příkazu (v tomto případě do cyklu s předem daným počtem opakování). Pro srovnání je vedle textového přepisu algoritmu obrázek vzhledu skriptu ve vývojovém prostředí.

Příkazy jazyka Scratch vycházejí z jazyka LOGO. Porovnáme-li příkazy pro pohyb postavy s příkazy pro pohyb želvy, zjistíme, že jsou téměř shodné. Obohacením je např. příkaz pro plachtění na pozici [*x*, *y*], které znamená pomalejší plynulý pohyb, trvající nastavený čas t, na rozdíl od prostého „jdi na pozici [*x*, *y*]“, které se provede ihned. Další zajímavou novinkou je příkaz „když narazíš na okraj, odraz se“.

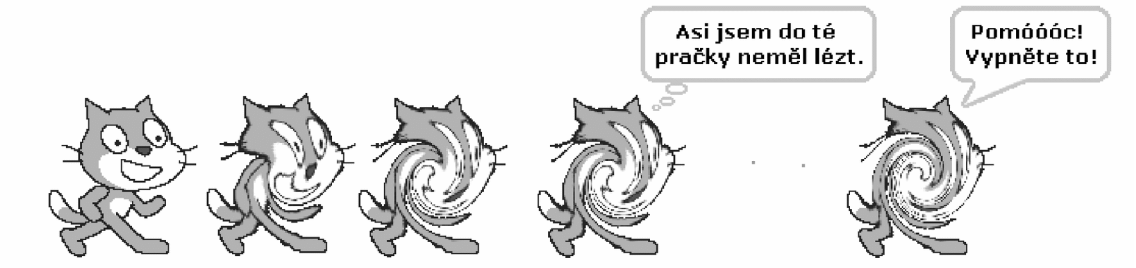
Hned v první skupině příkazů se setkáme s dvěma tvary „kostek“. Kromě „hranatých kostek s výstupky“, které představují výkonné příkazy, zde máme také „oválné kostky“ bez výstupků, které vracejí číselnou hodnotu (zde konkrétně polohu postavy, nebo její natočení). Scratch používá následující tvary kostek:

- ✔ hranaté kostky s výstupkem a zářezem pro **výkonné příkazy a řídící příkazy** podmínek a konečných cyklů;
- ✔ hranaté kostky s výstupkem a obloukovým horním okrajem pro **příkazy spouštějící jednotlivé skripty** (příkazy typu START);

- ✔ hranaté kostky se zářezem a rovnou spodní hranou pro **příkazy ukončující jednotlivé skripty** (přikazy typu STOP) a nekonečný cyklus;
- ✔ oválné kostky (tj. pruhy s kulatým levým a pravým zakončením) pro funkce vracející číselnou hodnotu, **proměnné a seznamy**;
- ✔ a šestiúhelníkové kostky (tj. pruhy se špičatým levým a pravým zakončením) pro **funkce vracející logickou hodnotu (ANO / NE)**.

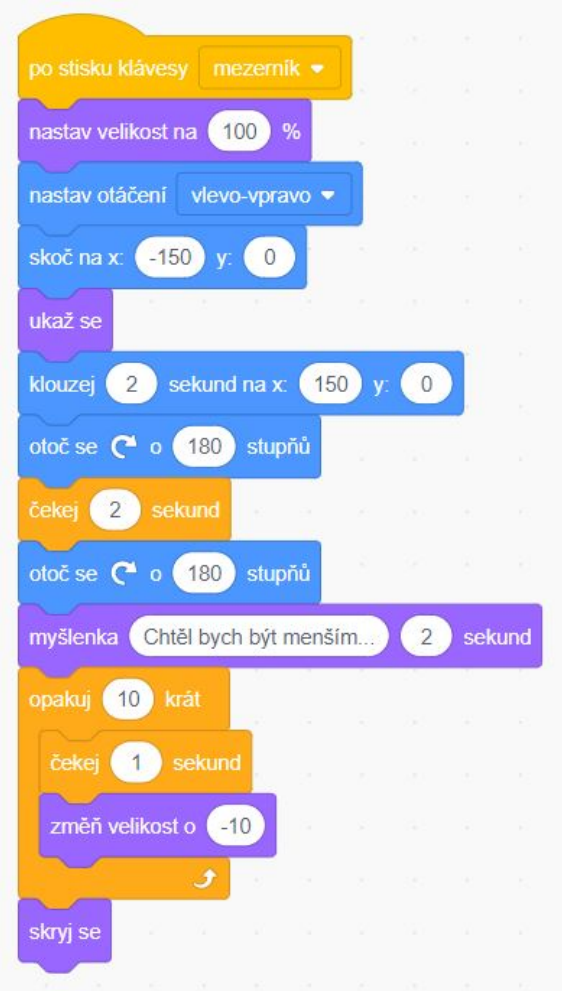
Všechny příkazy pro pohyb postavy (*sprite*) mají světle modrou barvu. Úzce s nimi souvisí další dvě skupiny příkazů, označené fialovou barvou – příkazy ovlivňující vzhled postavy – a petrolejovou modrou barvou – příkazy vnímání (smyslů, senzorů) postavy. z příkazů pro změnu vzhledu mají analogii v jazyce LOGO pouze dva: příkazy „ukaž se“ (odpovídá ShowTurtle) a „schovej se“ (odpovídá HideTurtle). Ostatní jsou zcela nové. Můžeme jimi ovládat převlékání kostýmů, změnu velikosti postavy, či různé grafické efekty včetně změny barvy. Postava může něco „říkat“ či něco „si myslet“ způsobem obvyklým u kreslených komiksových příběhů, tj. objeví se příslušná bublina s textem, který zadáme. Můžeme také ovlivnit, ve které vrstvě grafiky se postava nachází, přesunout ji do popředí, nebo naopak do pozadí scény.

Zkusme nyní vytvořit jednoduché scénáře (skripty) s příkazy ze skupin **Pohyb**, **Vzhled** a **Ovládání**. Skupina **Ovládání** má oranžovou barvu a obsahuje příkazy typu START („kostky“ s výstupkem a obloukovým horním okrajem), příkazy typu STOP („kostky“ bez výstupku s rovnou spodní hranou), řídící příkazy podmínek a cyklů, ale také příkazy pro rozesílání zpráv a příkaz „čekej, dokud nenastane“. Všechny příkazy se objeví, když vlevo nahoře vybereme (kliknutím myší) skupinu **Ovládání**, proto seznamy dostupných příkazů nikdy nebudeme uvádět formou přehledu či tabulky.

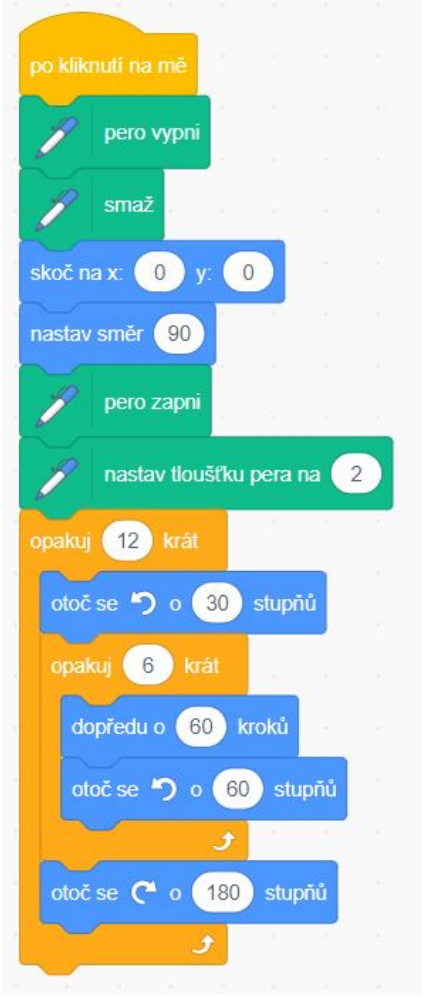


Obrázek 26: Efekt víření a využití textových bublin

Scénáře vytvářených skriptů můžeme pro žáky formulovat velmi volně. Např. kocour přejde sem tam přes scénu, zastaví se, pomyslí si: *„Chtěl bych být menším a ještě menším …“* a pak se opravdu zmenší. Nebo jiný scénář: kocour nakreslí návrh pravidelné vitráže pomocí kresby překrývajících se šestiúhelníků. Praktická realizace skriptů těchto dvou scénářů pak může být:



Obrázek 27: Scénka menšování postavičky



Obrázek 28: Vitráž z šestiúhelníků

Řadu dalších zajímavých nápadů najdeme na stránce projektu Scratch ve formě videotutorialů.[42] Nejlépe je tutoriál shlédnout, pokusit se jej nejprve zpaměti zopakovat a nakonec tvůrčím způsobem obměnit.

2.1.4 Programy ve Scratchi – tvorba strukturogramů

Jak již bylo v předchozí kapitole řečeno, programy vytvářené ve vývojovém prostředí Scratch představují **nejen slovní zápis algoritmů, ale zároveň grafické znázornění struktury algoritmu**. Jedná se tedy o specifický typ strukturogramů, ve kterých hraje určitou roli tvar jednotlivých „kostek“ stavebnice, ale také jejich barva. Tvary kostek jsme zmínili v minulé kapitole. z hlediska řídících struktur stojí za to připomenout, že podmíněné příkazy a příkazy cyklů graficky vymezují vnitřní prostor (s výstupkem a zářezem posunutým o stupínek doprava), do něhož vkládáme příkazy tvořící tělo cyklu. Podmíněný příkaz s větvením „pokud – jinak“ má dokonce dva vnitřní prostory.

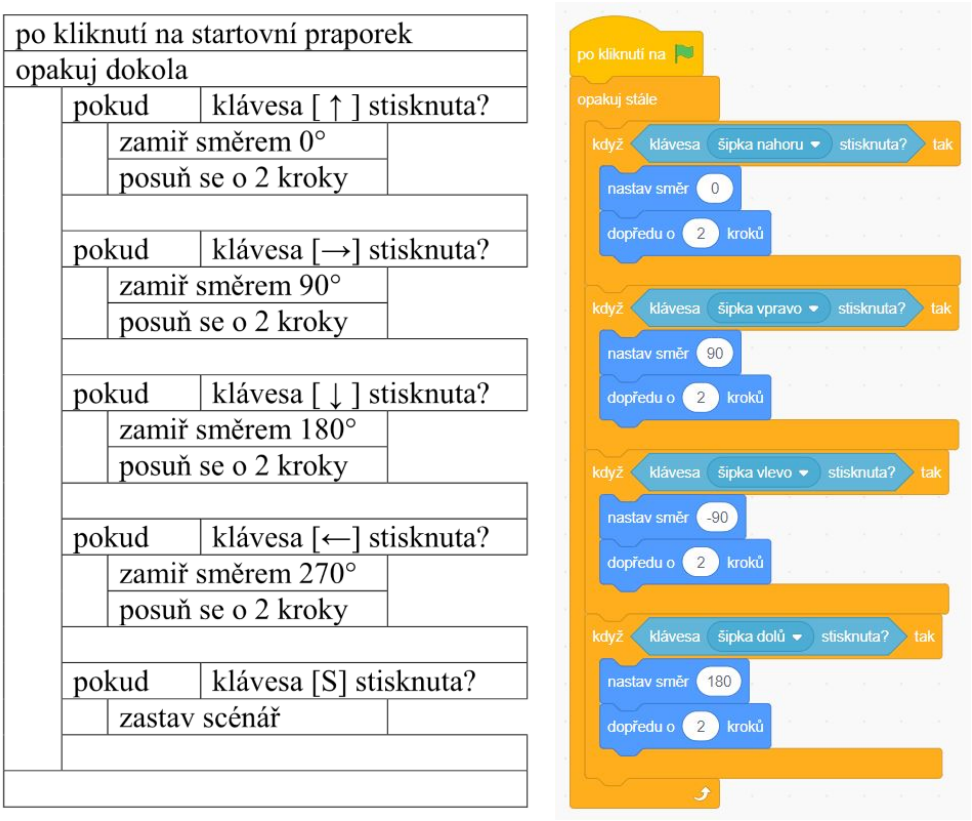
42 LIFELONG KINDERGARTEN. Video Tutorials: Introductory Tutorials and Paint Editor Tips. In: *Scratch* [online]. ©2019, July 8, 2016 [cit. 2019-09-10]. Dostupné z: https://scratch.mit.edu/help/videos/

Druhým faktorem posilujícím čitelnost strukturogramů je **barevné rozlišení skupin příkazů**. Pokud budeme se žáky promýšlet algoritmy mimo počítačovou učebnu, můžeme si pomoci pastelkami k vybarvení příslušné plochy strukturogramu.

Tabulka 1: Barvy skupin příkazů v prostředí Scratch

Skupina	barva	Skupina	barva
Pohyb	modrá	**Vnímání**	petrolejová
Vzhled	fialová	**Operátory**	zelená
Zvuk	malinová	**Proměnné**	cihlově oranžová
Události	světle oranžová	**Moje bloky**	růžová
Ovládání	tmavě oranžová	**Pero**	tmavě zelená

Ukážeme si to na scénáři, ve kterém postava (*sprite*) reaguje na povely uživatele z klávesnice. k tomu musíme využít jednu z možností skupiny **Vnímání**, a to logickou funkci „klávesa [*název klávesy*] stisknuta?“, pomocí níž zjistíme, zda je stisknuta příslušná klávesa pro pohyb kurzoru (šipka nahorů, dolů, vlevo, či vpravo). Tuto logickou podmínku jednoduše vsuneme do místa označeného šestiúhelníkovým výřezem v „kostce“ podmíněného příkazu.

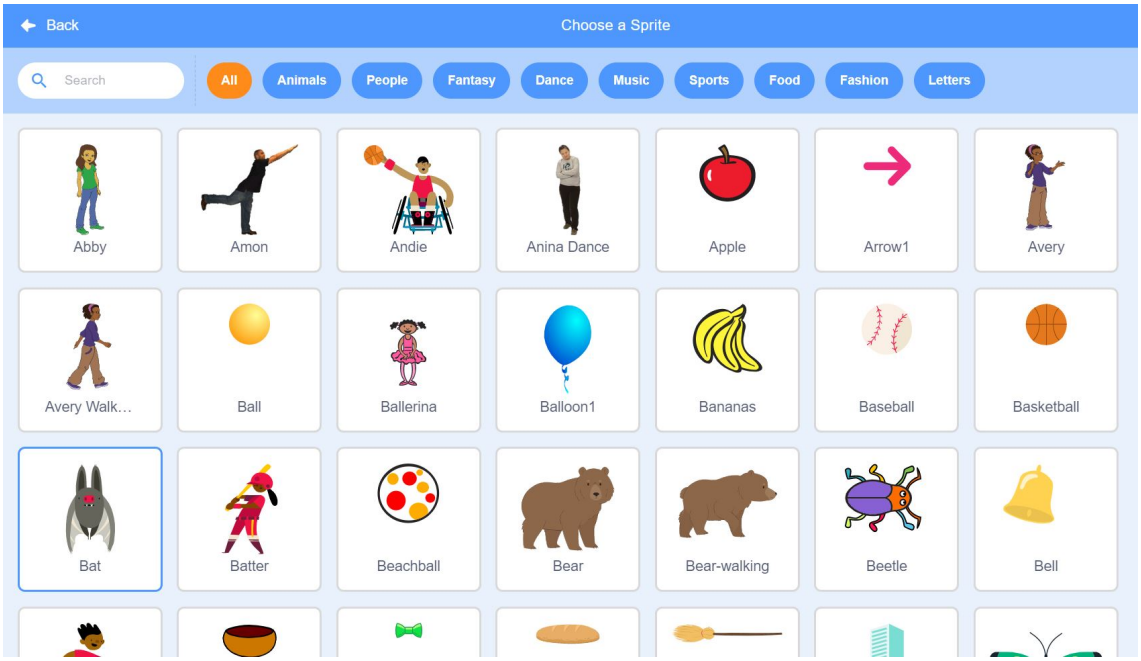


Obrázek 29: Strukturogram scénáře a jeho znázornění ve Scratchi

Porovnáme-li zjednodušený strukturogram, složený z obdélníkových polí, s mnohem názornějším zobrazením skriptu ve vývojovém prostředí Scratch, které využívá „kostky“ různých tvarů, uvědomíme si další obrovskou výhodu tohoto jazyka pro výuku programování. Stírá rozdíl mezi grafickým znázorněním algoritmu (vývojovým diagramem nebo strukturogramem) a jeho přepisem do programovacího jazyka. v **jazyce Scratch je znázornění algoritmu a psaní programu spojeno v jednu činnost**, která se navíc nápadně podobá práci s oblíbenou stavebnicí LEGO, a to nejen způsobem manipulace s „kostkami“, ale také svým tvořivým charakterem.

2.1.5 Práce s postavami a nastavení vlastností scény

Již jsme se zmínili, že Scratch umožňuje **vícevláknové programování**, kdy si každá postava (*sprite*) žije podle svého scénáře. Jak ale nové sprity na scénu dostaneme? Nejjednodušší je vybrat si už hotový sprite z nabídky. Pod scénou v sekci výběru postavy klikneme na tlačítko „Nový sprite“ a otevře se nám dialog nabízející široké spektrum již předem připravených spritů, které si dále můžeme vytřídit do kategorií na zvířata, lidi, jídlo, písmenka, apod. Užitečná je také možnost vyhledávání, kdy například klíčové slovo „car“ nabídne všechna auta (včetně autobusu). Mezi možnými dopravními prostředky nechybí například létající koště, ale oproti předchozí verzi ubyl létající koberec či helikoptéra.

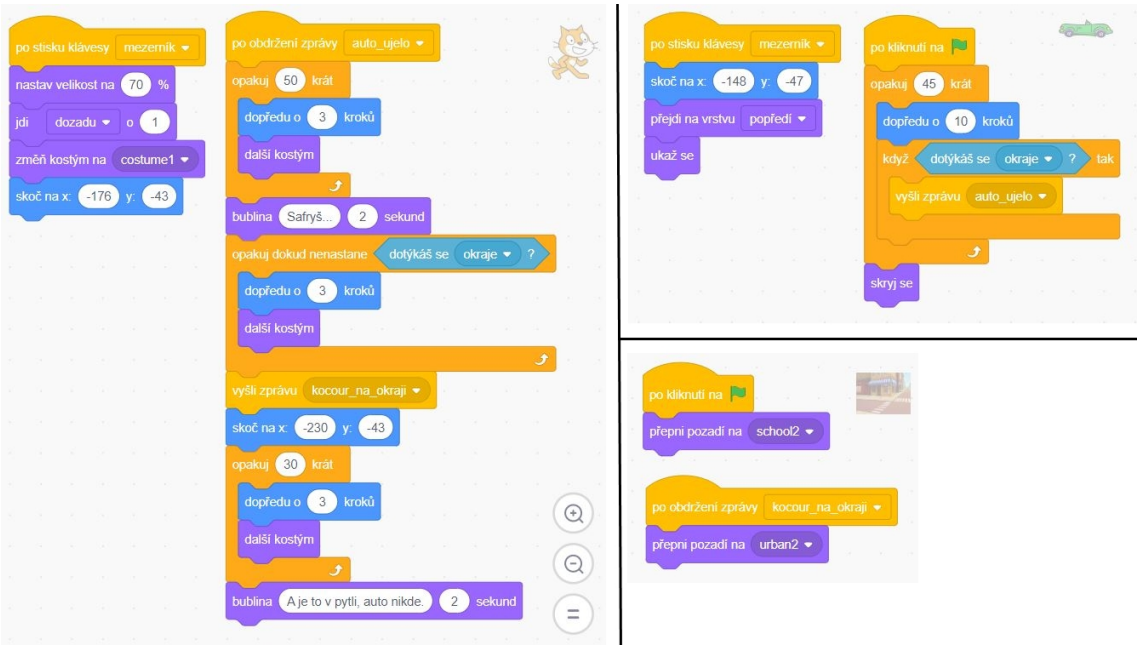


Obrázek 30: Otevřená nabídka předem připravených spritů v prostředí Scratch 3.0

Při vícevláknovém programování, má **každý sprite svůj scénář a mezi sebou mohou skripty komunikovat pomocí rozesílání zpráv**. Tento postup lze demonstrovat na následujícím jednoduchém scénáři. U levého okraje scény je auto

a za ním kocour, takže to vypadá, jako kdyby kocour seděl v autě a řídil ho. Po kliknutí na zelenou vlaječku se auto rozjede, přejede celou scénu a téměř zmizí za jejím okrajem. Kocour si toho všimne v okamžiku, kdy auto dorazí k okraji scény a rozběhne se za autem. Ve skutečnosti ovšem využijeme rozesílání zpráv. Auto v okamžiku, kdy se dotkne okraje scény rozešle zprávu „jedu“ (zpráva se rozešle všem spritům na scéně, ale jen některé na ni mají naprogramovanou reakci). Kocour se ihned po obdržení zprávy „jedu“ rozběhne za autem, ale když vidí, že už ho nedohoní, zastaví zhruba uprostřed scény.

Tato jednoduchá scénka lze rozšířit o široké spektrum navazujících úkolů. Kocour může například uprostřed scény zastavit jen na chvíli, něco říci a běžet dál. Jakmile se kocour dotkne okraje scény také, pozadí se přepne, změní se pozice kocoura na levý okraj scény a ujíždějící auto už nikde nebude (protože mělo náskok). Takovýchto možností je mnoho a lze tak bez problémů zabavit i rychlé žáky. Zároveň na této scéně ukážeme velké množství programovacích struktur, které je nutné s žáky probírat postupně a na kratších, specificky zaměřených příkladech. Na následujícím obrázku 31 lze vidět jednotlivé postavy a scénu komunikující prostřednictvím rozesílání zpráv.

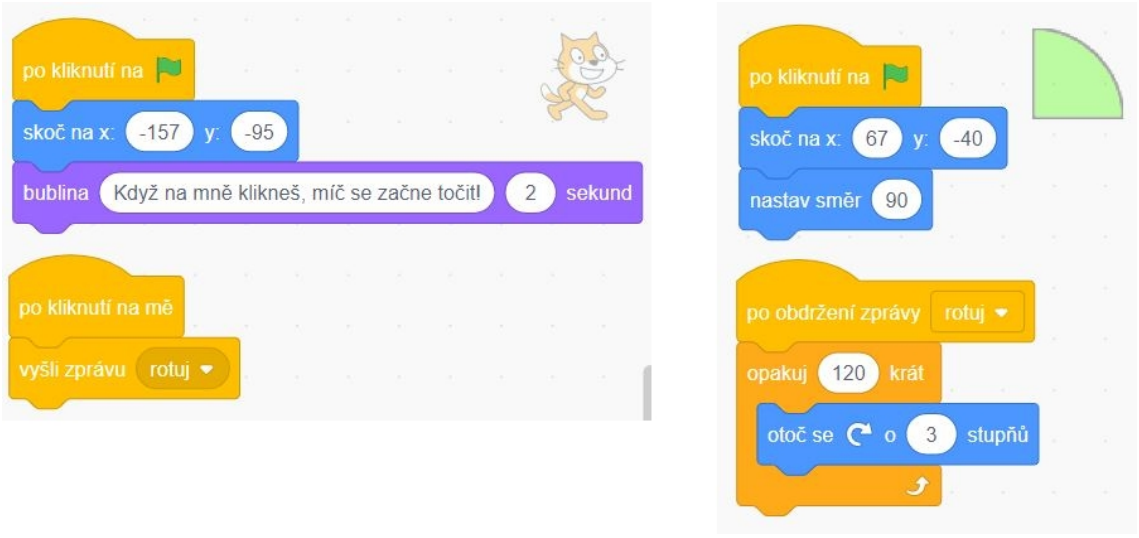


Obrázek 31: Všechny scénáře pro scénku s ujíždějícím autem

Nové sprity je možné kreslit v jednoduchém editoru obrázků přímo ve vývojovém prostředí Scratch, nebo je **možné je vytvořit v libovolném grafickém editoru**, který umožní ukládání ve formátu PNG, nebo GIF s průhlednou barvou. Přiměřenou komplexností se osvědčil například volně dostupný program Paint.NET. Pokud nemáme výtvarné nadání, můžeme pracovat třeba i jen s jednoduchými geometrickými tvary.

V následujícím rozšiřujícím příkladu využijeme kruhové výseče (přesněji řečeno čtvrtkruhy) různých barev (modrá, fialová, světle zelená a oranžová) a vyzkoušíme si, jak funguje rozesílání zpráv pro větší počet spritů.

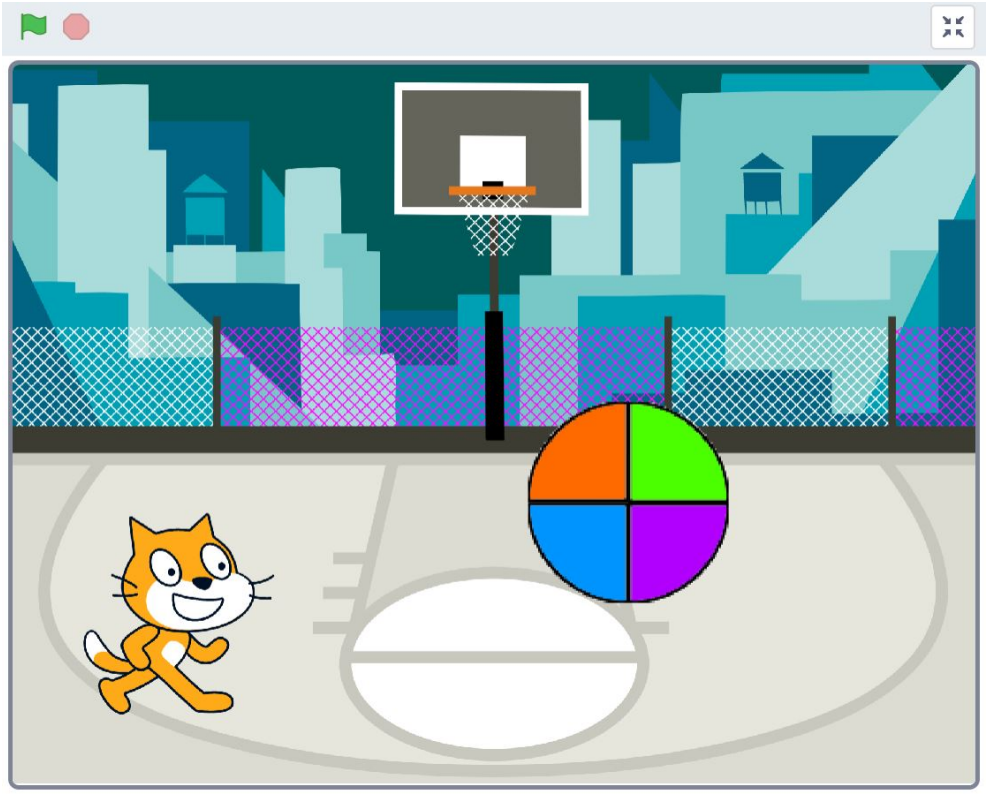
Tento jednoduchý scénář, v němž se po kliknutí na sprite kocoura všechny čtyři čtvrtkruhy otočí o 360° proti směru hodinových ručiček, a tak vytvoří dojem rotujícího míče, je možné různě obměňovat a doplňovat. Např. sprity 2 a 4 se mohou točit proti směru a sprity 3 a 5 ve směru hodinových ručiček, nebo lze rotaci spritů kombinovat s jejich posuvným pohybem, zapojit do „choreografie“ kocoura, přidat projíždějící zelené auto, nebo postavu psa.



Obrázek 32: Ukázka komunikace pomocí rozesílání zpráv a reakce více postav

Stejně jednoduše jako postavy můžeme vybírat z nabídky, kreslit, nebo vyfotografovat i **pozadí** scény. Záložka pozadí se objevuje na prostřední pozici místo záložky **kostýmy**. Obě krajní záložky, **skripty** a **zvuky**, zůstávají u scény stejné jako u spritů. Ukázku scény s pozadím v podobě hřiště na basketbal vidíme na obrázku 33 a stejně jako má Scratch širokou nabídku připravených kostýmů postav, můžeme si vybírat i z obdobně široké nabídky různých pozadí.

Pozadí samo o sobě příliš možností kódování nenabízí, z hlediska přehlednosti kódu se ale osvědčilo přiřazovat mu sekce kódu, které nelze jednoznačně zařadit k nějaké konkrétní postavě. Například typické proměnné SKÓRE nebo ŽIVOTY mohou být ovlivňovány velkým množstvím různých postav (hráč, nepřítel, prostředí, apod.) a nabízí se tedy otázka, která z těchto postav má na začátku hry skóre anulovat. Není-li odpověď jednoznačná, je ideální variantou odpovídající kód přiřadit právě pozadí.



Obrázek 33: Výsledná scéna s pozadím basketbalové hřiště

Samostatná práce (část 2)

K procvičení doposud probrané problematiky doporučujeme v první řadě vytvořit si vlastní účet ve Scratchi z důvodu automatického ukládání a možného publikování vašich projektů. Přestože je v této sekci jen jedna velice volně formulovaná úloha, dáte-li si na ni opravdu záležet, může se jednat o ukázku možností programovacího jazyku Scratch, kterou můžete žákům předvést jako motivační prvek na začátku první hodiny práce s tímto jazykem. Tuto vaši úlohu doporučujeme nasdílet s vašimi spolužáky, čímž rovnou získáte knihovnu ukázek a nápadů do vaší praxe:

a) Vytvořte **vlastní komplexnější skriptovanou scénku** (takže aby tam nebylo pět příkazů a nazdar). Scénka může být v **podstatě libovolná** - může se jednat o nějaký příběh, situaci, přepracovanou filmovou či seriálovou scénu, vtip, atp. Doporučuje se něco humorného (a má to být **použitelné i jako ukázka pro vaše žáky, takže také slušného**), ale můžete realizovat takřka jakýkoliv nápad máte (včetně scének využívajících zvuk). Ale pozor, má se jednat skutečně o nějakou scénku, tj. **nic interaktivního s uživatelem**, vše se má stát samo po stisknutí vlaječky.

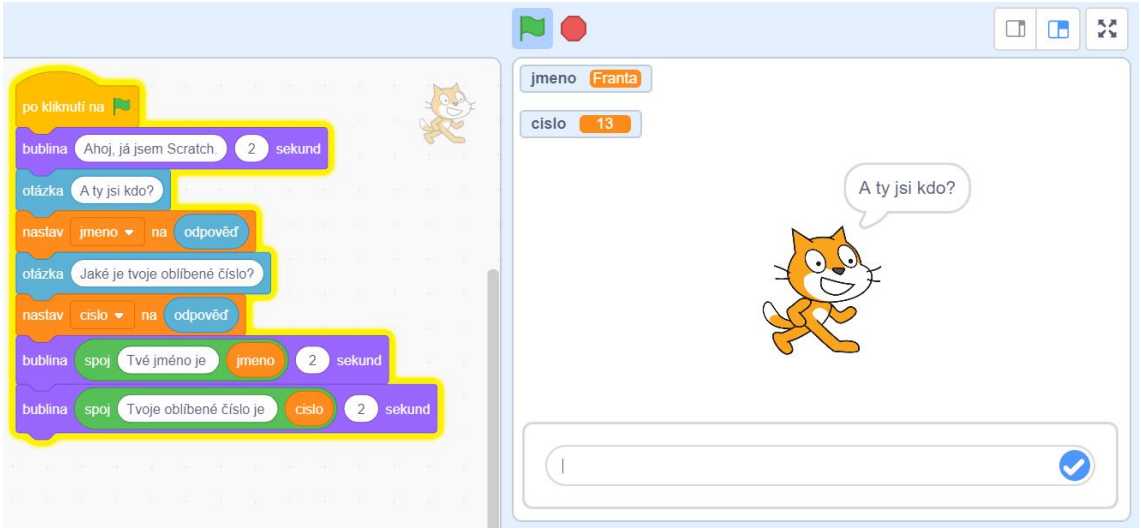
2.2 Vnímání, proměnné, seznamy a hlavolamy

O vnímání postav jsme se již stručně zmínili, nyní se na něj podíváme podrobněji. Když otevřeme skupinu vnímání, uvidíme řadu kostek, které mají stejnou petrolejově modrou barvu, ale liší se svým tvarem:

- ✔ **hranaté kostky** s výstupkem a zářezem jsou zde pouze tři a jedná se o procedury, které nevrací samy o sobě žádnou hodnotu (avšak otázka ukládá uživatelovu odpověď do vnitřní proměnné *Odpověď*);
- ✔ **oválné kostky** (tj. pruhy s kulatým levým a pravým zakončením) představují funkce (nebo, chcete-li, měřicí přístroje), jež vracejí číselnou hodnotu, jen u funkce **odpověď** může jít také o text;
- ✔ **šestiúhelníkové kostky** (tj. pruhy se špičatým levým a pravým zakončením) představují logické funkce, vracejí hodnotu pravda, či nepravda.

Libovolnou návratovou hodnotu funkce (číslo, text, logickou hodnotu) můžeme přiřadit k proměnné. Proměnnou je však nejprve nutné vytvořit. Vybereme skupinu proměnné, klikneme na tlačítko **Vytvoř proměnnou** a potom v rámci jednoduchého dialogu přidělíme proměnné jméno a určíme, zda se jedná o proměnnou globální (pro všechny sprity), či vázanou jen na jeden konkrétní sprite (potom se jedná o takzvanou lokální proměnnou, která je viditelná a přístupná jen z dané postavy, ke které je proměnná přiřazena). s vytvořením první proměnné se objeví nové oranžové kostky. Mezi nimi kostka **nastav [...] na[...]**. Do koncového okénka je možné nejen psát čísla (konstanty), či řetězce znaků, ale také vkládat oválné, či šestiúhelníkové kostky, tj. přiřazovat proměnné návratovou hodnotu funkce (a tuto hodnotu si můžeme uschovat po námi požadovanou dobu, zatímco petrolejové oválné proměnné jsou neustále aktualizovány a přepisovány).

Zajímavou dvojicí kostek je procedura **ptej se [...] a čekej** a s ní svázaná funkce **odpověď**. Naprogramujme tento jednoduchý scénář: Kocour se uživatele zeptá jak se jmenuje a jaké je uživatelovo oblíbené číslo. Získané odpovědi nakonec v rámci vlastní věty kocour uživateli zopakuje. Tato zdánlivě jednoduchá scénka obsahuje velké množství nových konstrukcí. Dvě otázky jsou použity z důvodu přepisování proměnné *Odpověď*. Proto je nutné vytvořit vlastní proměnné a odpovědi do nich přepsat. Pro zobrazení komplexnější odpovědi obsahující proměnnou je dále potřeba využít zeleného bloku *spoj*.



Obrázek 34: Textový input od uživatele pomocí příkazů dotaz

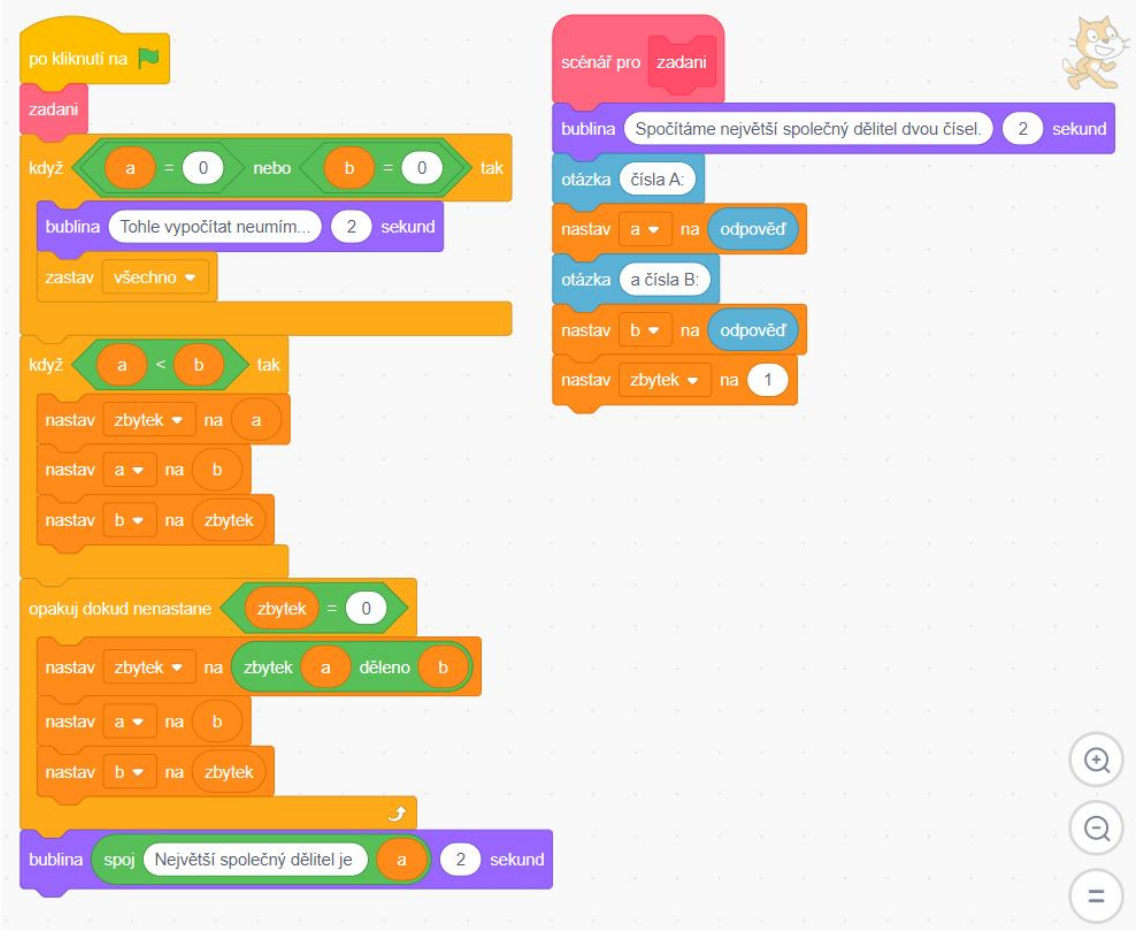
Funkce skupiny **vnímání** jsou vůbec jako stvořené pro interaktivitu skriptů a projektů, které vytváříme v jazyce Scratch. Umožňují práci s klávesnicí – **klávesa [...] stisknuta?**; myší – **souřadnice myši x, y, myš stisknuta?**; sledování polohy, vzhledu a hlasitosti jednotlivých spritů; nebo měření času stopkami.

V nejnovější verzi Scratch 3.0 v roce 2019 navíc byly **některé karty příkazů defaultně skryty a před jejich použitím je nutné je aktivovat** z menu vlevo dole. v této nabídce rozšiřujících karet příkazů se nachází mimo práce se zvuky a kreslením například také příkazy pro LEGO Mindstorms, Boost a WeDo 2.0, nebo příkazy pro micro:bit. Pomocí těchto dodatečných funkcí je možné propojení programů s vnějším světem. Senzory snímají hodnoty v reálném světě, se kterými počítač dále pracuje a pomocí Scratche lze ovládat i motory a pohyb robota.

Další možnosti nám dávají **skupiny operátory a proměnné**. Zkusme naprogramovat výpočet největšího společného dělitele dvou čísel. Při výpočtu využijeme tři proměnné *a, b a zbytek.* Celý finální kód je na obrázku 35.

Vraťme se ještě k práci s proměnnými, která je nepatrně složitější než práce s ostatními kostkami. Kromě jednoduchých proměnných umí Scratch pracovat také se seznamy. Stejně jako u jednoduchých proměnných je i **každý seznam potřeba nejprve vytvořit**, pojmenovat a rozhodnout, zda půjde o globální seznam, se kterým mohou pracovat všechny postavy, nebo o seznam pouze pro jednu postavu.

Seznam je možné nastavit na prázdný pomocí „zruš všechno z [*s*]“, připojit na jeho konec další prvek pomocí kostky „přidej … k [*s*]“, přečíst první, poslední, nebo libovolný jiný prvek seznamu pomocí funkce „prvek [*k*] z [*s*]“, rozhodnout, zda seznam obsahuje daný prvek pomocí logické funkce „[*s*] obsahuje …“ apod.

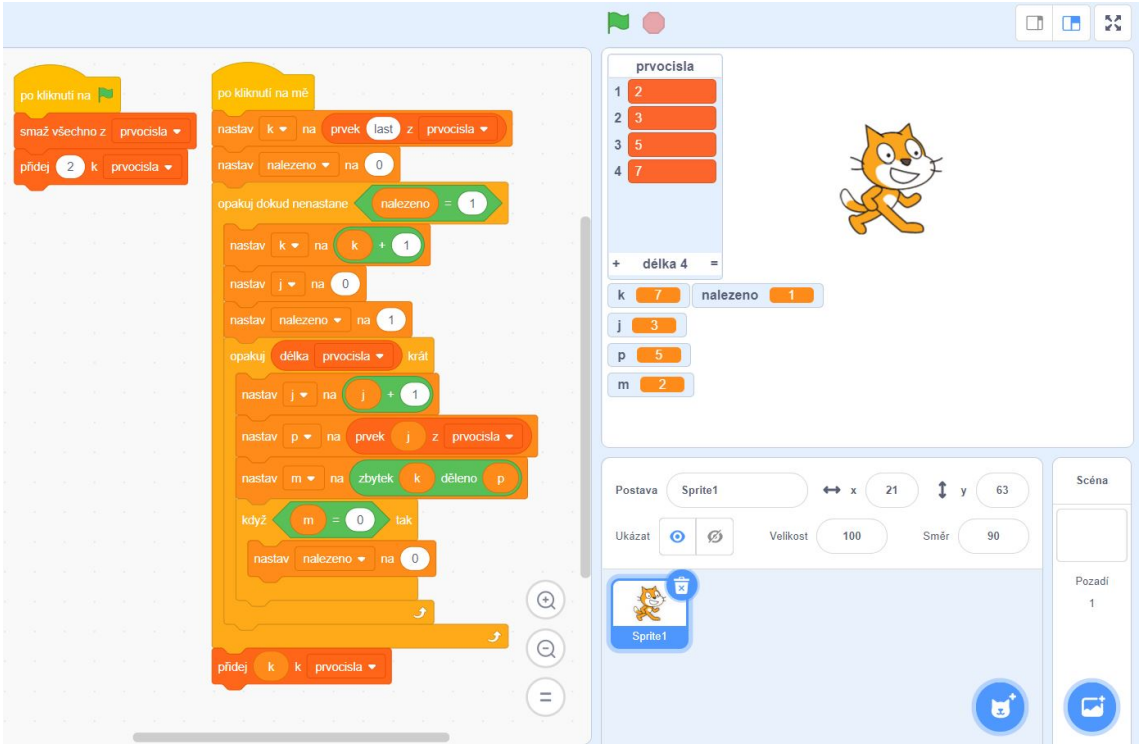


Obrázek 35: Výpočet největšího společného dělitele dvou čísel (pro lepší zobrazení využit vlastní blok pro zadání)

Ukažme si práci se seznamy na hledání vytváření seznamu prvočísel. Potřebujeme k tomu dva skripty, z nichž první, který nastaví počáteční stav seznamu (seznam obsahující pouze první prvočíslo 2), spustíme kliknutím na startovní praporek a druhý, který připíše k seznamu jedno další prvočíslo (pomocí něj můžeme seznam rozšířit na libovolný počet prvočísel), spustíme opakovaně kliknutím na sprite kocoura. Kromě seznamu nazvaného ***prvocisla*** vytvoříme proměnné *j, k, p, m* a *nalezeno*. Použitý algoritmus, hledání dalšího prvočísla vycházející z definice prvočísla, je vidět ze scénáře na obázku 36 a s analogickým postupem se setkáte v proceduře **dalsiprvocislo** v kapitole věnované práci se seznamy v jazyce LOGO. Všimněme si, že prvky seznamu jsou při zobrazení seznamu na scéně prostředí Scratch očíslovány, takže se na ně můžeme v situaci, kdy je to z oborově didaktického pohledu vhodné, dívat také jako na prvky jednorozměrného pole.

V dosud uvedených příkladech jsme si mohli povšimnout, že Scratch převzal z jazyka LOGO jen některé základní typy cyklů. Programovací jazyk LOGO zná sedm různých příkazů cyklu. Jsou to příkazy **repeat**, **for**, **foreach**, **forever**, **while**, **repeatwhile** a **repeatuntil**. Odpovídající řídící struktury v jazyce Scratch jsou tři: **opakuj [n] krát** (repeat), **opakuj stále** (forever) a **opakuj dokud nenastane**

(while not). Cyklus **opakuj stále** může být umístěn pouze na konci skriptu, tj. za ním již nemůže být žádná další kostka. Poslední cyklus **opakuj dokud nenastane**, odpovídají cyklu while not, tj. za určitých podmínek nemusí být proveden ani jednou. v první verzi Scratche bylo také možné vytvořit speciálním případ cyklu **opakuj dokud nenastane**, ovšem s prázdným tělem cyklu (tedy jen s cyklickým ověřováním podmínky) a získat tak příkaz **čekej dokud nenastane**. Takováto modifikace již není potřeba, protože příkaz **čekej dokud nenastane** je již vytvořen rovnou jako samostatný nový příkaz dostupný z běžné nabídky *Ovládání.*



Obrázek 36: Scénář generující seznam prvočísel

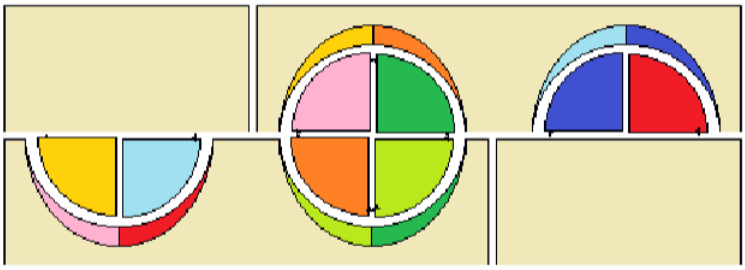
2.2.1 Permutační hlavolam jako ukázka možností Scratch

Možnosti vývojového prostředí Scratch si ukážeme na projektu, jehož výsledkem bude permutační hlavolam. Celá tato podkapitola je považována za rozšiřující učivo a je určena k samostudiu. Výsledný plně funkční program je veřejně k dispozici na adrese https://scratch.mit.edu/projects/2645670 přičemž *mim3* je uživatelská přezdívka pana doktora Musílka na webu Scratch.

Kromě kocoura se v něm objeví celkem dvanáct dalších spritů, z toho osm barevných čtvrtkruhů na průhledném pozadí, dva sprity připomínající mosty s dvěma oblouky, tj. půlkruhovými výřezy, v nichž se mohou pohybovat barevné čtvrtkruhy, a nakonec dva pomocné obdélníky, které se objevují při posunu „mostů“ a opticky dotvářejí celistvý vzhled hlavolamu (viz obrázky 37 a 38).

Obrázek 37: Spuštění permutačního hlavolamu

Na začátku, tj. po kliknutí na startovní praporek, nastavíme jednotlivé dílky hlavolamu do výchozích pozic. Střed levého horního čtvrtkruhu bude na souřadnicích [-80, 2], středy dalších čtvrtkruhů umístíme na pozice [-76, 2], [-80, -2], [-76, 2], [76, 2], [80, 2], [76, -2] a [80, -2]. Okolo čtvrtkruhů jsou pak „mosty“ na pozicích [0, 41] a [0, -41]. Přesné určení polohy jednotlivých dílků hlavolamu je nutné, protože z polohy dílku se určuje, jakým způsobem proběhne jeho přesun v daném tahu. Řízení veškerých pohybů je koordinováno rozesíláním zpráv, protože pak je nejen možné hlavolam řešit, ale také jej náhodně promíchat před započetím vlastního řešení. Výhodou tohoto způsobu řízení je možnost použít pro všechny čtvrtkruhy stejné skripty a ty potom rozkopírovat z prostoru skriptů jednoho sprite do všech zbývajících.



Obrázek 38: Posunutá pozice při míchání permutačního hlavolamu

Kopírování skriptu provedeme následujícím postupem. Klikneme pravým tlačítkem myši na záhlaví skriptu a objeví se kontextové menu. z něj vybereme možnost „kopírovat“, čímž se u kurzoru myši objeví kopie skriptu. Tu pak přetáhneme myší nad ikonu cílového sprite. Když je kurzor myši i s kopií skriptu nad cílem kopírování, klikneme levým tlačítkem myši a skript se vloží na požadované místo. Ještě jednodušší je kopírovat celé sprity. Stačí kliknout na ikonu sprite pravým tlačítkem myši a vybrat „kopírovat“. Pro každý ze čtvrtkruhů vložíme následujících osm skriptů:

Tabulka 2: Strukturogram rotace levého kruhu v kladném směru

Po obdržení zprávy 1		
pokud –85 < pozice x a pozice x < -70		
	otoč se o ↶ 90 stupňů	
	pokud (pozice x + 78) * pozice y > 0	
		jdi na pozici x: –156 – pozice x y: pozice y
	jinak	
		jdi na pozici x: pozice x y: 0 – pozice y
zastav scénář		

Tabulka 3: Strukturogram rotace kruhu uprostřed v kladném směru

Po obdržení zprávy 2		
pokud –10 < pozice x a pozice x < 10		
	otoč se o ↶ 90 stupňů	
	pokud pozice x * pozice y > 0	
		jdi na pozici x: 0 – pozice x y: pozice y
	jinak	
		jdi na pozici x: pozice x y: 0 – pozice y
zastav scénář		

Tabulka 4: Strukturogram rotace pravého kruhu v kladném směru

Po obdržení zprávy 3		
pokud 70 < pozice x a pozice x < 85		
	otoč se o ↶ 90 stupňů	
	pokud (pozice x – 78) * pozice y > 0	
		jdi na pozici x: 156 – pozice x y: pozice y
	jinak	
		jdi na pozici x: pozice x y: 0 – pozice y
zastav scénář		

Tabulka 5: Strukturogram posunu horní poloviny hlavolamu směrem vlevo

Po obdržení zprávy 4	
pokud pozice y > 0	
	nastav ps na směr
	zamiř směrem 90
	posuň se o –78 kroků
	zamiř směrem ps
jinak	
	nastav ps na směr
	zamiř směrem 90
	posuň se o 78 kroků
	zamiř směrem ps
zastav scénář	

Tabulka 6: Strukturogram posunu horní poloviny hlavolamu vpravo

Po obdržení zprávy 6	
pokud pozice y > 0	
	nastav ps na směr
	zamiř směrem 90
	posuň se o 78 kroků
	zamiř směrem ps
Jinak	
	nastav ps na směr
	zamiř směrem 90
	posuň se o –78 kroků
	zamiř směrem ps
zastav scénář	

Tabulka 7: Strukturogram rotace levého kruhu v záporném směru

Po obdržení zprávy 7		
pokud –85 < pozice x a pozice x < -70		
	otoč se o ↷ 90 stupňů	
	pokud (pozice x + 78) * pozice y > 0	
		jdi na pozici x: pozice x y: 0 – pozice y
	Jinak	
		jdi na pozici x: –156 – pozice x y: pozice y
zastav scénář		

Tabulka 8: Strukturogram rotace kruhu uprostřed v záporném směru

Po obdržení zprávy 8		
pokud –10 < pozice x a pozice x < 10		
	otoč se o ↷ 90 stupňů	
	pokud pozice x * pozice y > 0	
		jdi na pozici x: pozice x y: 0 – pozice y
	Jinak	
		jdi na pozici x: 0 – pozice x y: pozice y
zastav scénář		

Tabulka 9: Strukturogram rotace pravého kruhu v záporném směru

Po obdržení zprávy 9		
pokud 70 < pozice x a pozice x < 85		
	otoč se o ↷ 90 stupňů	
	pokud (pozice x – 78) * pozice y > 0	
		jdi na pozici x: pozice x y: 0 – pozice y
	jinak	
		jdi na pozici x: 156 – pozice x y: pozice y
zastav scénář		

Podobně, ale ještě o něco jednodušeji, se pohybují sprity, které uzavírají čtvrtkruhy v jejich pozicích. Horní „most s dvěma oblouky“ se posunuje při obdržení zprávy 4 o –78 kroků, při obdržení zprávy 6 o +78 kroků, na jiné zprávy nereaguje. Dolní „most s dvěma oblouky“ se posunuje vždy opačným směrem, tj. při obdržení zprávy 4 o +78 kroků, při obdržení zprávy 6 o –78 kroků. Polohu sprite 10, čili horního „mostu s dvěma oblouky“, využijeme při kontrole pohybů hlavolamu, abychom zamezili vysunutí dílků mimo scénu. Ovládání, které můžeme umístit jako skripty scény, bude velmi jednoduché (v tabulce 10 je kvůli úspoře místa místo názvu Sprite10 použita zkratka S_10):

Tabulka 10: Strukturogramy rozeslání řídících povelů všem spritům

Poslední skripty připíšeme kocourovi, který po kliknutí na startovní praporek připlachtí na pozici [–78, –118] a vyzve uživatele: „Kliknutím na mě zamíchej hlavolam.“ Kliknutím na sprite 1 – kocoura se spustí skript, který zamíchá dílky hlavolamu a předá uživateli informaci, jak ovládat pohyby hlavolamu. v programu je využita proměnná *mix*, podle jejíž hodnoty se provede jednotlivý pohyb. Hodnota proměnné *mix* se náhodně generuje pomocí funkce **zvol náhodné číslo od 0 do 9**.

Tabulka 11: Strukturogram zamíchání čtvrtkruhů hlavolamu

```
Po kliknutí na Sprite 1
    plachti 0,5 vteřin na pozici x: –78 y: –148
    opakuj 50 krát
        čekej 0,1 vteřiny
        nastav mix na zvol náhodné číslo od 1 do 9
        pokud pozice x z Sprite10 < –70 a mix = 4
            nastav mix na 5
        pokud pozice x z Sprite10 > 70 a mix = 6
            nastav mix na 5
        rozešli všem mix
    čekej 0,1 vteřiny
    pokud pozice x z Sprite10 < –70
        rozešli všem 6
    pokud pozice x z Sprite10 > 70
        rozešli všem 4
    povídej „Můžeš začít řešit hlavolam.“ příští 3 vteřiny
    povídej „Program ovládej numerickými klávesami.“ příští 3 vteřiny
    povídej „tedy klávesami 1, 2, 3, 4, 6, 7, 8 a 9.“ příští 3 vteřiny
zastav scénář
```

Zajímavým problémem, nepatřícím ovšem tematicky do informatiky, ale do matematiky, je také analýza řešení tohoto kombinatorického hlavolamu.

Samostatná práce (část 3)

Abyste si upevnili získané vědomosti, vypracujte následující cvičení:

a) **Naprogramujte ve Scratchi hru Tic-tac-toe (tj. piškvorky** v síti 3 x 3 čtverce) tak, aby kocour (tj. program) byl uživateli počítače (alespoň relativně) inteligentním soupeřem.

b) Vytvořte **hru pexeso ve zmenšené podobě (např. 6 x 4)**. Líc a rub kartičky budou dva kostýmy téhož sprite. Po kliknutí na sprite se otočí lícem, po kliknutí na druhý sprite budou otočené dva různé sprity po dobu 3 vteřin a pak se otočí zpět rubem nahoru. Dva stejné sprity však již zůstanou otočeny lícem trvale.

c) Naprogramujte **hlavolam Faust, resp. hlavolam Osel a stáj**, který je složený ze čtverců různých velikostí a obdélníků. Inspiraci najdete na webových stránkách jednoho z autorů těchto skript (http://www.musilek.eu/michal/), kde jsou uvedené hlavolamy vytvořené pomocí HTML a JavaScriptu.

d) Napište v jazyce Scratch **dvě procedury**, z nichž první vytvoří výchozí seznam s dvěma prvky (čísly 1 a 2) a druhá po každém kliknutí na sprite přidá k danému seznamu s *k* **prvky Fibonacciho posloupnosti** připojí další prvek *ak+1*.

e) Napište proceduru, ve které **sprite – auto jede od středu scény po spirálové trajektorii a zároveň kreslí** tuto spirálu perem. Pohyb sprite se zastaví v okamžiku, kdy se dotkne okraje scény.

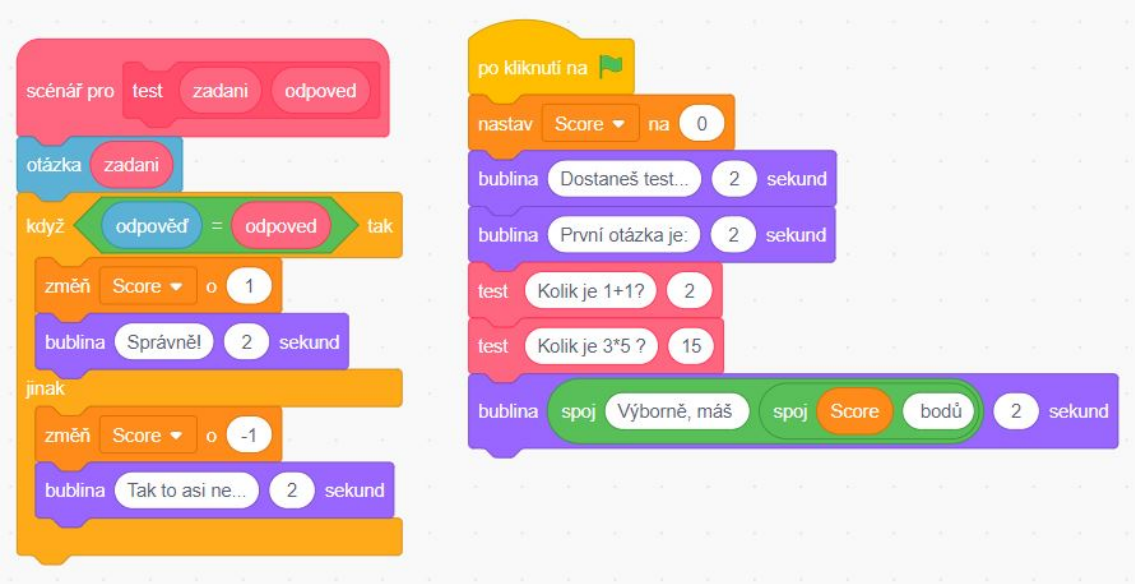
f) Naprogramujte **šifrovací program, který realizuje transpoziční šifru Podle plotu** (angl. Rail Fence Cipher) a zároveň vhodným názorným způsobem vizualizuje postup šifrování a dešifrování.

g) Vytvořte v prostředí Scratch p**omocí dvou kruhových sprite o různé velikosti Albertiho šifrovací disk**. Vnější disk ať má 26 písmen řazeno abecedně, vnitřní podle hesla ALBERTI. Stisk klávesy s písmenem ať otočí oba spojené disky písmenem nahoru. Stisk číslice ať pootočí oba disky proti sobě navzájem.

2.3 Tvorba vlastních jednoduchých her

V tuto chvíli jsme již schopni tvořit poměrně rozsáhlé programy, ve kterých již může být problém se orientovat. Některá řešení by navíc mohla být podstatně elegantnějšího za použití vlastních funkcí, což v prostředí Scratch naštěstí není problém. **My si práci s vlastními funkcemi ilustrujeme na jednoduché hře, ve které nás bude kocour testovat sadou otázek**. Tohoto výsledku lze docílit i bez využití vlastních bloků, avšak každá otázka znamená rozšíření kódu o sedm řádků, které se navíc až na zadání otázky a její správnou odpověď vůbec nemění. Jedná se tedy o ideální případ pro vytvoření vlastního bloku s parametry.

V rámci samotného hlavního těla kódu pak při správně vytvořeném vlastním bloku **stačí pro každou otázku přidat pouhý jeden řádek kódu** se dvěma parametry (zadání otázky a očekávaná správná odpověď).



Obrázek 39: Ukázkový minitest vytvořený za pomoci vlastního bloku

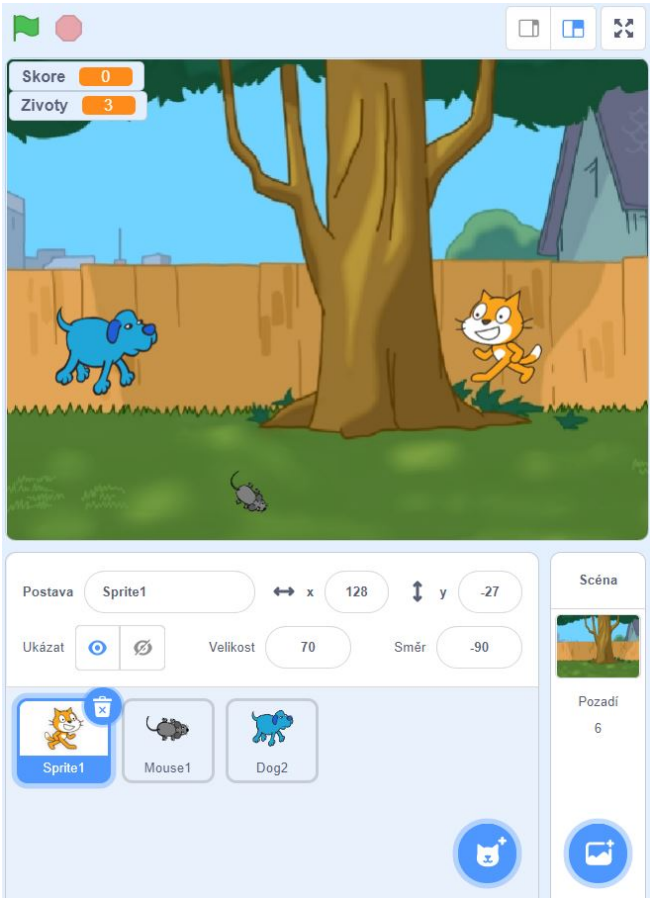
Tato úloha žákům nejlépe prokáže užitečnost vlastních bloků v případě, že žáky necháte vytvořit delší test o alespoň pěti otázkách a poté po nich budete chtít nějakou změnu ve vyhodnocování otázek (například v bodování). Orientace v takovémto dlouhém a duplicitním kódu je obtížná, což perfektně kontrastuje s lehkostí modifikace jednoho vlastního bloku.

Konkrétně tato **úloha byla pro zařazení do těchto skript vybrána s úmyslem přesahu do mezipředmětových souvislostí**. Takto žáky vytvořený „test“ může být na jakékoliv téma, do jakéhokoliv předmětu. Žáci tak vidí přímou spojitost a dopady světa informatiky na reálný svět kolem nich. Navíc v určité míře dochází k synergickému efektu, kdy si žáci při jedné činnosti osvojují dva a více předmětů zároveň.

2.3.1 Ukázka jednoduché, avšak plnohodnotné hry

Hry, které obsahují veškeré typické herní náležitosti (úvodní obrazovka, cíl hry, zvyšování obtížnosti hry a její zakončení, atp.) nám umožňují propojit veškeré algoritmické konstrukce a datové typy v jednom velkém projektu. Podstata a zadání takové hry však nemusí být nijak zastrašujícím způsobem složitá. Například můžeme chtít **vytvořit hru, ve které bude hráč ovládat kocoura, který se bude snažit chytit náhodně utíkající myš a zároveň bude prchat před psem, který bude kocoura pronásledovat**.

Takovéto zadání bez jakýchkoliv dalších rozšíření vede k plně funkční hře, kterou by měli být schopni do určité míry stvořit i slabší žáci, a zároveň umožňuje **téměř nekonečná rozšíření pro žáky rychlé**. Tito žáci mohou přidat skóre a životy, power-upy, úvodní a závěrečnou obrazovku, různé herní módy (třeba pro dva hráče, kdy druhý hráč ovládá právě psa), různé úrovně obtížnosti, náhodně generované překážky, složitější pohybové vzorce pro prchající myš, speciální schopnosti pro kocoura, ovládání kocoura pomocí myši namísto klávesnice, atp.

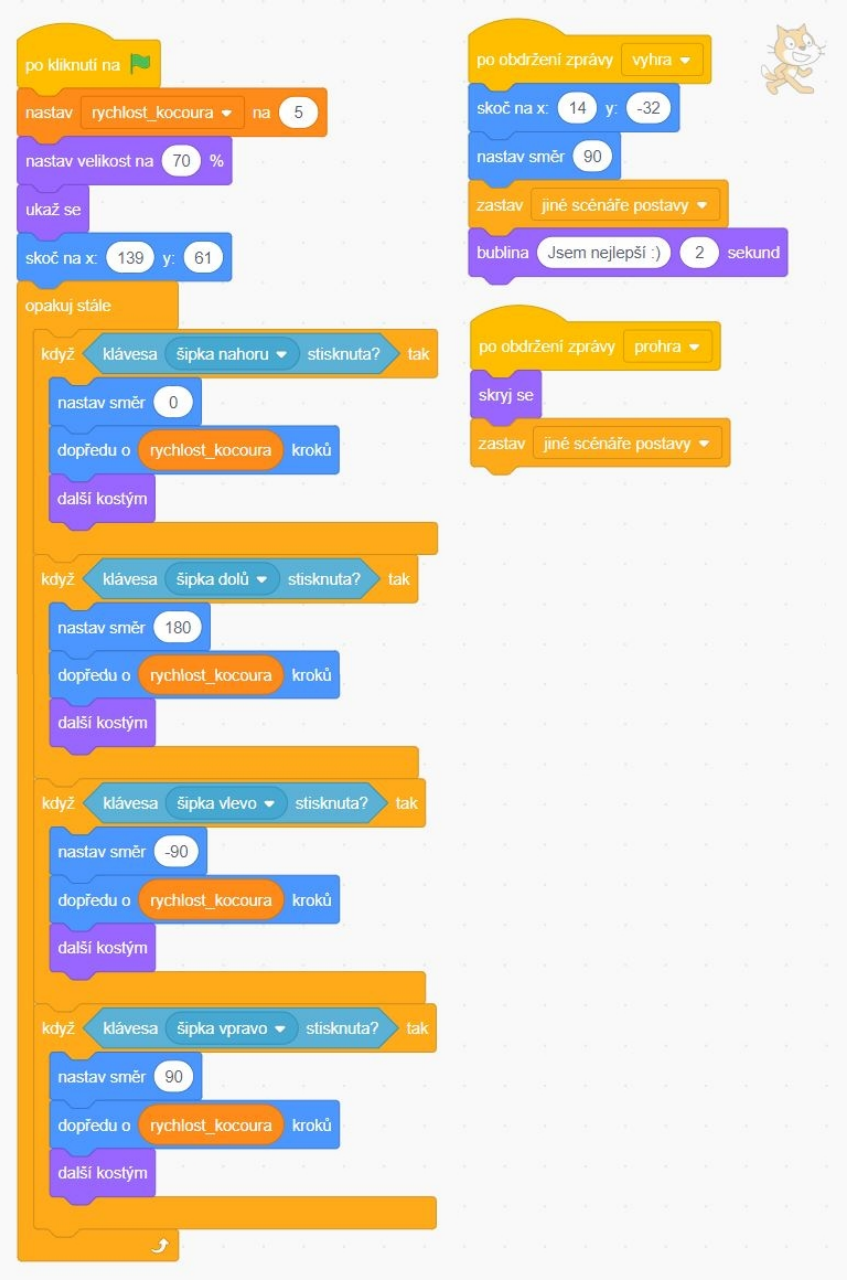


Obrázek 40: Finální herní scéna a použité postavy

Obrázek 41: Scéna kontrolující běh hry na základě skóre a počtu životů

Scénářem kontrolujícím postup hrou (tedy výhru či prohru) je scéna. Ta restartuje, počítá a kontroluje skóre a životy, se kterými manipulují ostatní postavy. Na základě získání určitého počtu bodů či ztráty životů **scéna vysílá zprávu *výhra* nebo *prohra***, na kterou má každá z postav naprogramovanou odpovídající reakci.

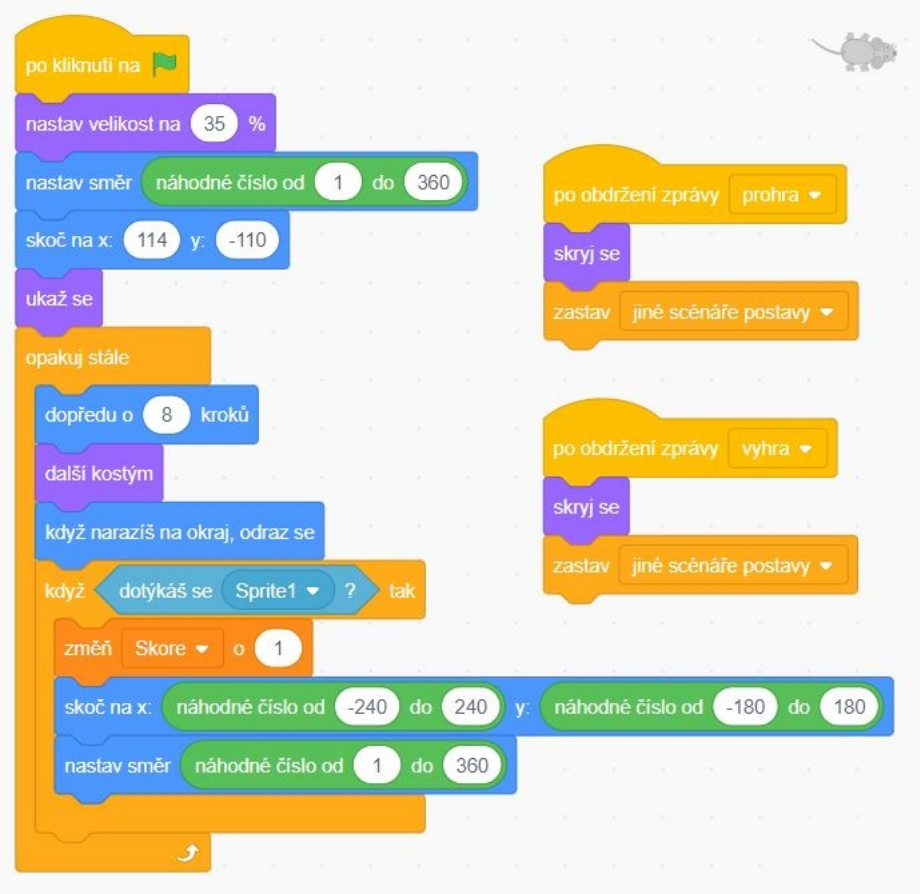
Pořadí postav je dále na uživateli, z důvodu kontroly funkčnosti kódu je ale vhodné vytvořit a programovat postavu psa až po postavě kocoura. v našem příkladu tedy **začneme postavou kocoura, kterého má ovládat hráč**. Zde se nabízí vícero možností – pohyb kocoura lze ovládat pomocí čtyř (a více) událostí ***Po kliknutí na klávesu…*** avšak výsledkem je trhavý pohyb, který hru kazí. Lepším řešením je vytvořit nekonečnou smyčku, která běží od začátku hry a umožňuje plynulou kontrolu kocoura ošetřenou v rámci jedné procedury (viz obrázek 42).



Obrázek 42: Ovládání hráčem a reakce na výhru či prohru

Máme-li hotovou plynulou a funkční kontrolu pohybu kocoura, můžeme se přesunout buď ke psovi nebo k myši. v našem případě si zvolíme myš. Stejně jako v případě kocoura Scratche, mají **myš i pes již připravené sprity ve výchozí knihovně postav Scratche**, stačí si tedy tuto postavu v knihovně jen najít a přidat.

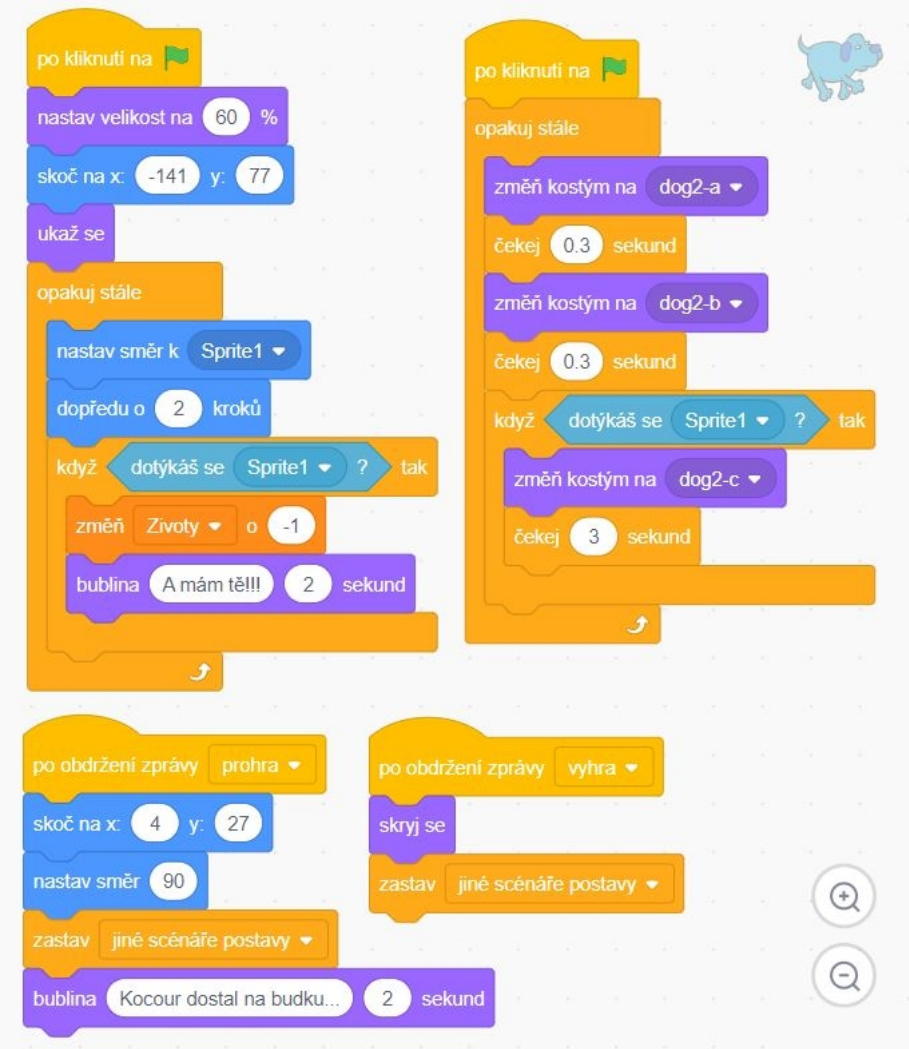
Pohyb myši má být nepřetržitý a dostatečně náhodný, aby bylo chycení myši pro hráče alespoň trochu výzvou. Využijeme tedy velice mocného bloku *Když narazíš na okraj, odraz se*, díky kterému můžeme vytvořit velice jednoduchou smyčku pohybu myši. Stejně jako u každé jiné postavy, **nesmíme zapomenout začít nastavením výchozí pozice**, která je v případě myši lehce zkomplikovaná faktem, že je-li myš natočena kolmo k jakémukoliv okraji scény, její pohyb bude buď pouze horizontální nebo vertikální. Využijeme tedy matematické funkce, mezi nimiž se nachází i generátor náhodných (respektive pseudonáhodných) čísel. Pomocí takto generovaného čísla je tak na začátku každé hry myš natočena jinak a hra není úplně jednotvárná. Stejnou funkci generující náhodná čísla využijeme i pro skok na novou, náhodnou pozici v případě, že je myš chycena kocourem.



Obrázek 43: Scénáře kontrolující prchající myš

Pohyb psa je jednodušší, **stačí psa jen průběžně natáčet směrem ke kocourovi a nechat ho běžet vpřed**. Pro iluzi pohybu, kterou získáváme střídáním spritů je zde nutné vytvořit samostatné vlákno, čili v kódu psa jsou dvě procedury

začínající stisknutím zelené vlaječky. **Rychlost psa a tím i obtížnost celé hry lze jednoduše zvýšit zvýšením rychlosti psa**, které dosáhneme zvětšením hodnoty jeho kroku (z 1 na 2, případně i výše, přičemž na obrázku 44 je rychlost psa nastavena právě na 2).



Obrázek 44: Scénáře kontrolující pronásledujícího psa

U všech postav je nakonec potřeba přidat reakce na výhru či prohru hráče. U myši je situace nejjednodušší, protože v obou případech potřebujeme jen ukončit pohyb myši po scéně a skrýt ji. v případě zprávy *výhra* je nutné skrýt psa a kocoura postavit doprostřed scény na pódium (k této změně dochází v rámci kódu scény, viz obrázek 41). v případě zprávy *prohra* je situace přesně opačná. Pro větší atraktivitu je vhodné nechat žáky vymyslet nějaký vtipný komentář pro výherní postavu. v našem případě vyhraje-li pes (tj. rozešle se zpráva *prohra*), pes stojí na pódiu a říká *„Kocour dostal na budku...“*

2.3.2 Klonování aneb tvorba nových instancí třídy

S největší pravděpodobností jste již také narazili na situaci, kdy velké množství postav mělo prakticky totožný kód (někteří z vás si možná vzpomenou na rozšiřující permutační hlavolam, který je bez využití klonování vytvořený poněkud neohrabaně). Navážeme-li na právě dokončenou hru s kočkou a psem, co kdybyste chtěli například zvýšit úroveň obtížnosti přidáním dalšího psa, nebo rovnou dvou? Přidat nové postavy se stejným kódem je sice možné, ale co když budete chtít tento kód nějak modifikovat?

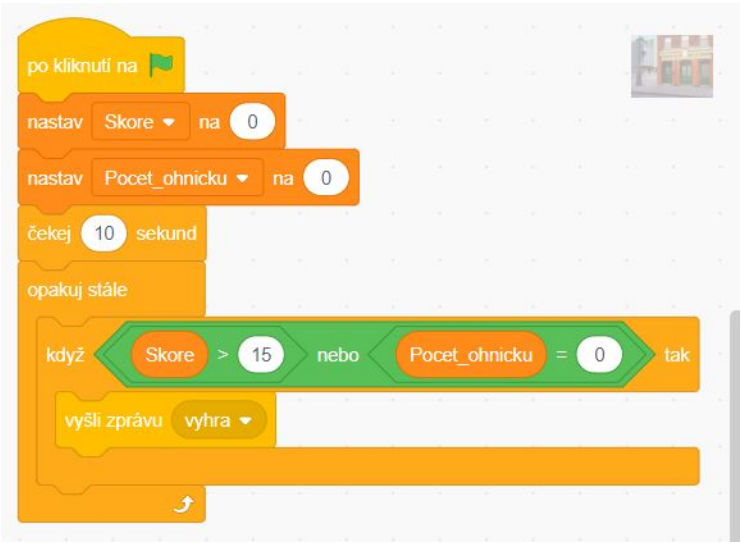
Příkladem, ve kterém **se bez klonování neobejdeme** je hra, jejíž centrálním principem je zvyšování počtu kopií (neboli klonů) jedné postavy. Úkolem tedy je vytvořit hru, ve které bude **hasič ovládaný hráčem (pro změnu tentokrát pomocí myši) hasit neustále se objevující ohníčky**, které se mají po svém objevení pomalu pohybovat po scéně. Hra vypadá například jako na obrázku 45.

Obrázek 45: Ukázka spuštěné hry s hasičem hasicím ohníčky

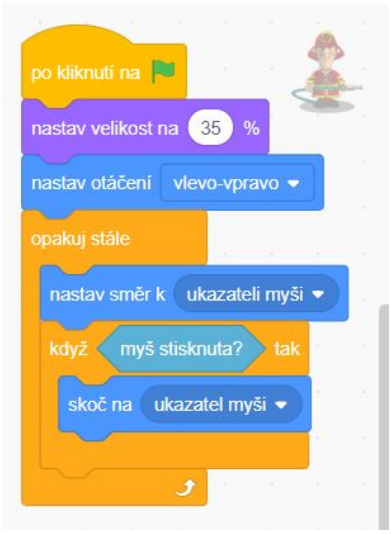
V této úloze **lze opět využít pomocné proměnné obsahující například počet ohníčků**, které jsou aktuálně na scéně. Ačkoliv lze indexovat jednotlivé klony (viz dále v této kapitole), zde není počítaný skutečný počet na obrazovce, ale proměnná *Pocet_ohnicku* je inkrementována při vytvoření nového klonu a naopak snížena při uhašení ohníčku. Jako ve většině našich her si kontrolu této proměnné hlídá kód přiřazený ke scéně (viz obrázek 46). **Kód ovládající hasiče je velice jednoduchý** a v podstatě se v něm jen neustále kontroluje kde se nachází myš, což je pozice na kterou má hasič při kliknutí myši skočit (obrázek 47).

Podstatně zajímavější je kód pracující s ohníčky. **Kód klonování má vždy minimálně dvě části. v první části se nastaví výchozí vlastnosti původního objektu** (originální postavy, která se má klonovat) a ve druhé akce konkrétní nově

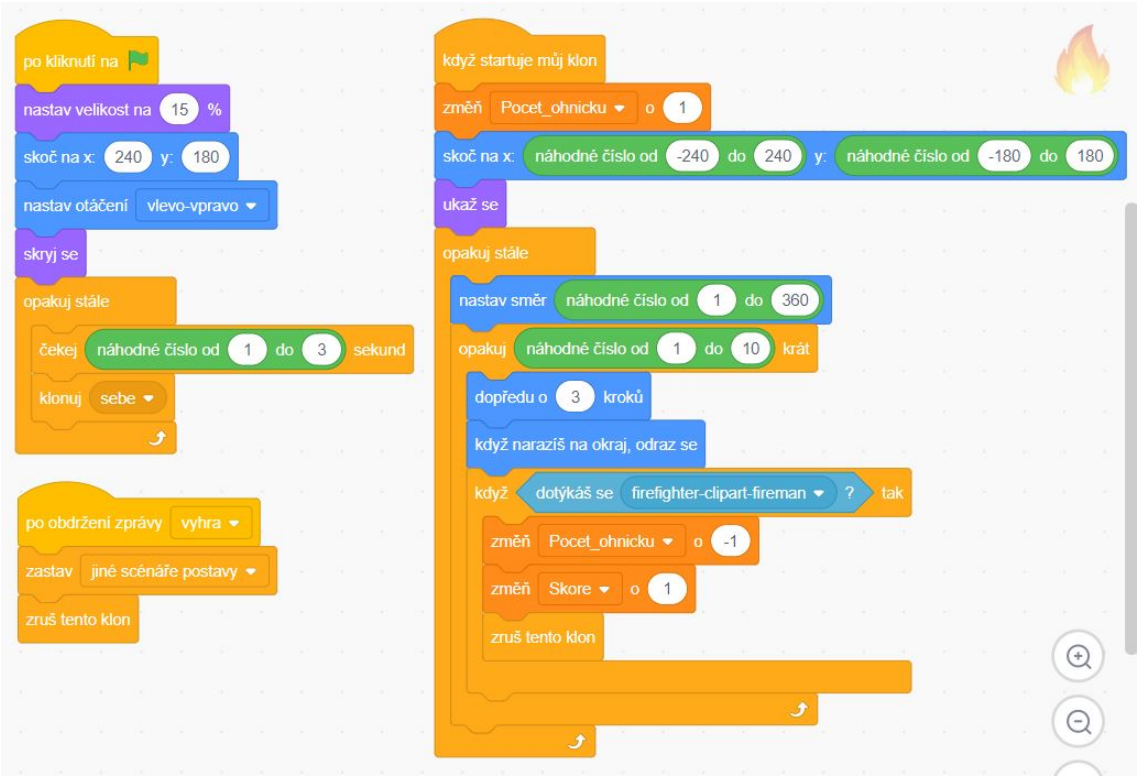
vytvořené instance objektu (tedy klonu). Ačkoliv to není úplně přesné, **lze o první části kódu uvažovat jako o definici třídy**, tedy společného výchozího nastavení všech objektů. **Tento princip je základem objektově orientovaného programování (OOP)** a v určité omezené míře tak lze učit již ve Scratchi. Všechny ohníčky mají například mít nějakou standardní velikost a způsob otáčení. Tyto **vlastnosti tedy nastavíme původní postavě**, a to ještě předtím, než se začneme zabývat samotným procesem klonování (viz po kliknutí na vlaječku na obrázku 48).



Obrázek 46: Kód scény kontrolující proměnné



Obrázek 47: Scénář pro ovládání hasiče pomocí myši

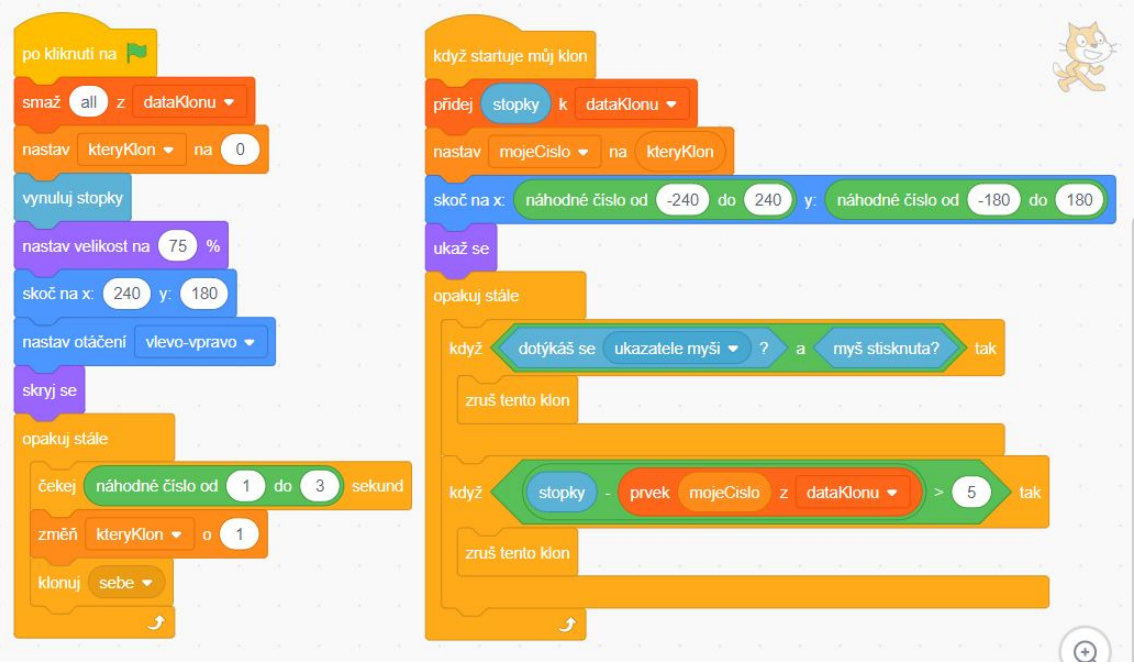


Obrázek 48: Scénář pro klonování ohníčků

Původní postava klonu nám ale v další práci překáží, je tedy vhodné ji „odklidit“ někam stranou a hlavně ji skrýt. Jedním z prvních příkazů u každého klonu tak je znovu ukázání daného klonu, protože s postavami a s klony, které jsou skryté nelze provádět žádné interakce. Každý klon má poté nastavené instrukce pro pohyb a zároveň si také hlídá, jestli se náhodou nedotýká hasiče. Jestliže ano, klon se zruší. **Bez nutnosti duplikování kódu můžeme tímto způsobem vytvořit téměř libovolné množství klonů**. Modifikace klonů v takovém případě není problém a případné změny není nutné opakovat na více místech v kódu.

Tato jednoduchá hříčka je překvapivě zábavná a umožňuje takřka stejné možnosti rozšíření jako předcházející hra s kočkou a psem. Stejně tak je možné ponechat veškeré herní principy a nechat na žácích, aby si vybrali grafický obsah hry, kterým může být třeba pavouk lovící mouchy, kocour lovící myši, miska či kbelík chytající kapky, atp. Jakkoliv je změna vizuální stránky hry minimálním zásahem do hry jako takové, pro žáky se jedná o velice silně motivující prvek a jejich představivost se většinou ukáže jako bezbřehá.

Každý klon si při takovéto práci hlídá vlastní kód a vlastní proměnné. **Je tedy možné, aby všechny klony obsahovaly stejnou proměnnou, jejíž obsah je pro každý klon individuální** (opět se jedná o princip tříd a objektů z OOP). Tento fakt můžeme demonstrovat například na úloze, kde se mají v náhodných intervalech objevovat klony kocoura Scratche. Každý klon si má pamatovat přesný čas svého vytvoření a neklikneme-li na něj (čímž bychom ho zrušili), sám se vždy přesně za pět vteřin vymaže.



Obrázek 49: Časované mizení klonů ve Scratchi

2.3.3 Tematická návaznost v hodinách

Z důvodu plně rozvinutého abstraktního myšlení a určité míry zkušeností s algoritmy a jejich principy, které jste získali v odpovídajících předmětech, jsou v těchto skriptech jednotlivé algoritmizační konstrukce a programovací principy namíchány velice živelně. Velké množství nových konceptů je tak často představeno jako součást jediného příkladu, což je postup, který je v rámci vaší výuky na základní či střední škole nanejvýš nevhodný.

Žáci jsou ve velké míře typičtí určitou „těkavostí“ myšlení, nesoustředěností na osvojení konkrétního principu a snahou vyřešit každou úlohu pokud možno jen za použití již známých konstrukcí. **Problematiku je tedy třeba rozdělit na lehce stravitelná dílčí témata, která na sebe logicky navazují**. Posloupnost těchto témat se liší v závislosti na konkrétní učebnici, avšak základní **struktura je však vždy velice podobná**.

Během tříletého testování v zájmovém útvaru na základní škole byly využity různé sekvence a postupy práce. Vybrané úlohy zaměřené na specifické algoritmické konstrukce byly průběžně modifikovány a jejich pořadí upravováno. Nejvíce se doposud osvědčila následující posloupnost témat (kde základní témata vycházejí zhruba na jednu vyučovací hodinu, komplexnější témata si však mohou vyžádat i dvě až tři hodiny):

- ✔ **Ukázkové projekty a základní seznámení s prostředím** – Pro tuto část se výborně hodí již několikrát opakované nasazení ukázkové Hour of Code před samotnou prací v prostředí Scratch. Práci ve Scratchi začínáme ukázkami již hotových projektů (přičemž ideální jsou přímo vámi vytvořené a navzájem nasdílené projekty, které vznikly v rámci samostatných prací podle úloh na konci každé kapitoly v rámci tohoto předmětu). Žáci se dále seznámí s celým webem Scratch a vyzkouší si veřejně sdílené projekty ostatních uživatelů z celého světa (sekce *Explore*).
- ✔ **Stažení projektů do PC a registrace na webu Scratch** – Žáci se naučí ukládat a zálohovat své projekty, a to včetně stažení projektu přímo do svého počítače. Na osvojení tohoto postupu stačí využít jen ten naprosto nejelementárnější kód (např. klouzání kocoura z jedné strany na druhou).
- ✔ **Základní bloky příkazů a jejich návaznost** – Žáci se učí pracovat s příkazy v kategoriích události, pohyb a vzhled. Zjistí, že každá postava v rámci jedné procedury umí zpracovávat jen jeden příkaz.

- ✔ **Kreslení pomocí pera a základní cykly** – Tzv. želví grafika je velice vhodná pro práci s těmi nejelementárnějšími algoritmickými konstrukcemi a dá se perfektně využít pro osvojení práce s cykly.
- ✔ **Přidání nové postavy a práce více postav** – Každá postava má vlastní nezávislý kód a každá stejná sekvence příkazů se vždy vykoná za stejný čas. Těchto poznatků lze využít k implementaci zdánlivé interaktivity jednotlivých postav, která je založená na navzájem plně nezávislých kódech obsahujících přesně načasované příkazy čekání.
- ✔ **Zasílání zpráv mezi objekty** – Časově závislá interaktivita je velice neohrabaná a stačí sebemenší zásah do existujícího kódu, aby se vyladěnost celého programu úplně rozpadla. Jednotlivé postavy však mezi sebou mohou komunikovat podstatně pružněji. Ačkoliv je zasílání zpráv mezi objekty významným prvkem objektově orientovaného programování, žáci s pochopením tohoto tématu zpravidla nemívají větší problémy.
- ✔ **Rozhodování a vnímání** – Ty nejjednodušší scénáře fungují kaskádově, tedy postupují z vrchu dolů a každý příkaz se před posunem k dalšímu vždy vykoná. Takový program však postrádá flexibilitu a je v podstatě v rozporu se základními vlastnostmi algoritmů (konkrétně hromadnosti a obecnosti). Kategorie rozhodování a vnímání signifikantně rozšíří škálu úloh, které jsou žáci schopni řešit.
- ✔ **Vlastní komplexní skriptovaná scénka** – Teprve v tuto chvíli je vhodné zařadit na základní škole tvorbu vlastní scénky, kterou jsme my v rámci těchto skript v podstatě začínali.
- ✔ **Testování projektů a bug hunting** – v této fázi jsou již žáci schopni vytvářet poměrně rozsáhlé a komplexní programy, které ale často bývají plné chyb a nezamýšlených událostí. Jedná se tedy o ideální fázi pro téma takzvaného bug huntingu, neboli hledání chyb v existujícím kódu.
- ✔ **Klonování** – Tvorba většího množství postav s takřka stejným kódem je vysoce neefektivní a vede k těžce (až nemožně) udržitelnému kódu. Klonování svým způsobem simuluje tvorbu nových instancí tříd v objektově orientovaném programování a je tak velice užitečným nástrojem, který by měl být využit před přesunem k vyšším jazykům.
- ✔ **Konstanty a proměnné** – Toto téma lze zařadit i před klonování a teoreticky i ještě dříve, v této etapě však již žáci začínají tvořit vlastní hry, jejichž přirozenou součástí je udržování skóre a velice často i „životů."
- ✔ **Složené podmínky a logické operátory** – Zatímco u tvorby skriptovaných scének si žáci vystačí s jednoduchými podmínkami, v hrách jsou situace výrazně komplexnější a přirozeně se tak nabízí využít i komplexnějších podmínek složených z vyhodnocení dvou a více dílčích podmínek.

- ✔ **Vlastní bloky a vlastní bloky s parametry** – Obdobně jako u konstant a proměnných se jedná o téma, které lze zařadit kamkoliv. V současné době zkoušíme toto téma přesunout již za rozhodování a vnímání, a to zejména z důvodu tendence žáků v případě problému v kódu „opravovat" i sekce, které jsou ve vztahu k danému problému naprosto irelevantní. To zpravidla vede k úplné degradaci kódu a následné demotivaci žáků.
- ✔ **Seznamy** – Zatímco princi práce s proměnnými chápou velice rychle i matematicky nenadaní žáci, práce se seznamy je nesrovnatelně komplikovanější a pro žáky také náročnější téma.
- ✔ **Vlastní komplexní hra** – v tuto chvíli by již žáci měli chápat a dokázat používat veškeré základní algoritmické konstrukce a měli by být schopni samostatně tvořit i značně komplexní projekty typu her.

Ideálním **signálem pro posun k dalšímu tématu je osvojení základních principů práce v současného tématu do takové míry, že žáci sami přirozeně narazí na nové limity** doposud osvojených konstrukcí a jejich možných kombinací. Zde však zejména u rychlejších žáků hrozí nebezpečí příliš povrchních a nedostatečně osvojených znalostí. Vše navíc dále komplikuje intelektuálně nehomogenní skupina, ve které často ti nejpomalejší žáci potřebují k osvojení povrchních základů probírané problematiky více času, než rychlí žáci k relativně hlubokému osvojení celého probíraného tématu.

Na průchod všemi výše vypsanými tématy je nutný rozsah odpovídající alespoň dvaceti vyučovacím hodinám. Výuka se dá urychlit až na téměř polovinu, témata však potom nejsou probrána dostatečně hluboko a žáci nemají prakticky žádný prostor pro tvorbu vlastních scének a her. Je-li časová dotace na téma algoritmizace a programování na dané škole příliš omezená a nejedná-li se o specializovanou školu, je na zvážení nestačí-li využít například jen vybrané lekce ze zkráceného kurzu Hour of Code. Naopak máte-li k dispozici zhruba dvanáct a více hodin, Scratch je velice mocný vzdělávací jazyk.

Na trhu se nachází velké množství (zejména anglických) knih pro výuku programování ve Scratchi a stejně tak internet je plný i volně dostupných publikací (jako jsou například knihy od autora jménem Al Sweigart, viz doporučená literatura na konci skript). Kvalita těchto publikací je kolísavá, téměř všude však můžete načerpat minimálně nějakou inspiraci. Velká část úloh testovaných v rámci výše popsaného zájmového útvaru byla inspirována velice povedenou diplomovou prací Jana Krejsy,[43] ve které také naleznete další analýzu návaznosti jednotlivých dílčích témat v různých učebnicích a dalších publikacích.

43 KREJSA, Jan. (2014). *Výuka základů programování v prostředí Scratch* [online]. České Budějovice, 2014 [cit. 2019-09-20]. Jihočeská univerzita v Českých Budějovicích, Pedagogická fakulta. Vedoucí práce doc. PaedDr. Jiří Vaníček, Ph.D. Dostupné z: https://theses.cz/id/b5f11x/. Diplomová práce.

Závěrečné shrnutí kapitoly 2

Po přečtení této kapitoly byste měli:

- ✔ znát historii vývoje programovacího jazyka a prostředí Scratch a jeho spojitost s Google Blockly a Snap!;
- ✔ dokázat využít plného potenciálu celého portálu Scratch;
- ✔ prohlížet a upravovat projekty ostatních uživatelů;
- ✔ vytvářet a sdílet vlastní scénky a hry;
- ✔ realizovat všechny základní algoritmické konstrukce v prostředí Scratch a vytvářet pro ně odpovídající úlohy;
- ✔ využít ve Scratchi tzv. želví grafiku a její typické úlohy;
- ✔ rozšířit funkcionalitu Scratche použitím skrytých kategorií a propojit tak například Scratch s LEGO Mindstorms;
- ✔ naplánovat delší výuku ať již do hodin informatiky, nebo do extra-kurikulárního zájmového útvaru.

Samostatná práce (část 4)

Soutěž ScratchCup je v Čechách i na Slovensku velice povedená. Budete-li chtít zapojit i vaše žáky (což je vřele doporučeno), měli byste alespoň tušit, jak obtížné úlohy v této soutěži jsou a na co žáky připravovat. Obdobně komplexní je i tvorba vlastní hry, která vyžaduje velice důkladné promyšlení konceptů, postupů i vlastní tvorby. Tyto projekty doporučujeme sdílet s vašimi kolegy pro obohacení vaší učitelské praxe (stejně jako scénky ze 3. části samostatných prací).

a) Vyberte si **jednu úlohu ze sbírky soutěžních úloh soutěže ScratchCup**, jejichž ukázkové zadání je veřejně dostupné na adrese http://www.scratchcup.cz/ulohy/ dole po kliknutí na odkaz *úlohy*. z nejjednodušších úloh si nevybírejte úlohy 1 a 4, ze středních úloh si nevybírejte úlohu 1 a z komplexních úloh si nevybírejte úlohu 1.

b) Alternativní úlohou je **vytvoření nějaké vlastní hry** ve Scratchi. Může se jednat o hru závodní, postřehovou, logickou, adventuru, střílečku, RPGčko, atp. Hru „dotáhněte“ až do konce, tj. nějaký jasně daný cíl (možnost výhry či prohry) by hra mít měla a nezapomeňte na instrukce pro hráče na začátku.

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

3 Roboti a robotické stavebnice

Jací roboti a jaké robotické stavebnice jsou na trhu dostupné?

Porovnejte základní výhody a nevýhody vybraných robotů.

Jaké jsou možnosti a nějaké konkrétní příklady propojení výuky s roboty a robotickými stavebnicemi s mezipředmětovými vztahy?

Co je potřeba připravit a zkontrolovat před hodinou s Ozokódy?

Jaké jsou tři základní způsoby ovládání Ozobota EVO?

Jak lze rozšířit schopnosti Ozobota?

Jaké jsou možné úlohy rozvíjející mezipředmětové vztahy za použití Ozobota (ať již s Ozokódy nebo OzoBlockly)?

3.1 Situace na trhu s robotickými stavebnicemi

Stejně jako se objevují stálé nové ryze softwarové projekty pro výuku programování a algoritmizace na počítačích a tabletech, vznikají také nové nástroje a zařízení pro podporu výuky pomocí robotů a robotických stavebnic. Zatímco pohyb kreslené postavičky po monitoru je pro děti určitě zábavný, rozhýbání skutečně fyzicky přítomného robota je **obrovský motivační faktor**.

Stále populárnější a běžně dostupné robotické výukové pomůcky jsou odraženy i v projektu PRIM, který se pro rozvoj informatického myšlení žáků zaměřil i na širokou škálu takovýchto robotů. **Informatické myšlení žáků lze rozvíjet již v mateřské školce** a na prvním stupni základní školy, k čemuž jsou na webu imysleni.cz **dostupné vzdělávací materiály pro jednoduchého robota jménem Bee-bot**.[44]

44 MANĚNOVÁ, Martina a Simona PEKÁRKOVÁ. (2019). *Rozvoj IM s využitím robotických hraček v MŠ a na 1. stupni ZŠ* [online]. ©2019 [cit. 2019-08-22]. Dostupné z: https://www.imysleni.cz/ucebnice/rozvoj-informatickeho-mysleni-s-vyuzitim-robotickych-hracek-v-materske-skole-a-na-1-stupni-zs

Pro **čtvrtou a pátou třídu jsou zde materiály pro LEGO WeDo**,[45] které je takovým značně zjednodušeným předstupněm pro LEGO Mindstorms. z hlediska výuky podporované robotickými pomůckami se na webu imysleni.cz dále nachází **vzdělávací materiály právě pro LEGO Mindstorms**.[46]

Přesuneme-li se ze vzdělávacího sektoru do nabídky robotických „hraček," nalezneme i zde širokou škálu použitelných projektů. Zajímavým je inteligentní **programovatelný robot Sphero**, který je oficiálně licencován a vytvořen podle slavného **BB-8 nebo BB-9E ze Star Wars**. i populární výrobce dronů nabídne (kromě drona DJI Tello, který je pro vzdělávání také použitelný a cenově dostupný) svou variantu vzdělávacího robota jménem RoboMaster S1, který sice není modulární/přestavitelný, ale zato disponuje kanónem, velice zajímavým hardwarovým vybavením a nepříliš příznivou cenovkou až 15 000 Kč.[47]

Poslední kategorií produktů, které z hlediska výuky programování stojí za zmínění, jsou **vzdělávací jednodeskové počítače typu Raspberry Pi nebo Arduino**. Tyto miniaturní počítače byly původně vyvinuty pro výuku informatiky (i když jsou v současné době hojně využívány i pro různé DIY projekty) a kromě programování na nich také lze demonstrovat práci s hardwarovou stránkou počítače, a to bez rizika poškození školních počítačů žáky. Konkrétně pro platformu Arduino v rámci imysleni.cz vznikla učebnice také a je doporučená studentům druhých až třetích ročníků středních škol s doporučením pro školy technicky odborné a gymnázia.[48] Poněkud jednodušší variantou k Arduinu je velice podobný počítač **Micro:bit,**[49] přičemž samotné programování Micro:bitů je v odpovídajícím vzdělávacím materiálu založeno na reálně běžně používaném programovacím jazyce Python a tyto podklady jsou určeny pro střední školy s doporučením pro školy *ne*technicky zaměřené a gymnázia.

Kromě webu imysleni.cz je dalším významným českým zdrojem informací, nápadů a materiálů web http://robotika.sandofky.cz/, sourozenecký portál k webu o Ozobotech (o kterém se dočtete mnohem více v kapitole Ozobot), přičemž oba tyto weby jsou od stejné autorky, Hanky Šandové.

---

45 PROCHÁZKA, Josef, Jakub LAPEŠ a Daniel TOCHÁČEK. (2019). *Edukační robotika s LEGO® WeDo 2.0 pro 1. stupeň základní školy* [online]. ©2019 [cit. 2019-08-22]. Dostupné z: https://www.imysleni.cz/ucebnice/edukacni-robotika-s-lego-wedo-2-0-pro-1-stupen-zakladni-skoly

46 JAKEŠ, Tomáš, Jan BAŤKO a Petr SIMBARTL. (2019). *Robotika na 2. stupni základní školy s LEGO® Mindstorms*[online]. ©2019 [cit. 2019-08-22]. Dostupné z: https://www.imysleni.cz/ucebnice/robotika-na-2-stupni-zakladni-skoly-s-lego-mindstorms

47 SEDLÁČEK, Vojtěch. (2019). Výrobce populárních dronů DJI představil robota s kanónem, který naučí děti programovat. In: *Czechcrunch* [online]. Praha, ©2019, 13. 6. 2019 [cit. 2019-08-22]. Dostupné z: https://www.czechcrunch.cz/2019/06/vyrobce-popularnich-dronu-dji-predstavil-robota-s-kanonem-ktery-nauci-deti-programovat/

48 NOVÁK, Milan a Jiří PECH. (2019). *Robotika: učebnice pro střední školy - Arduino* [online]. ©2019 [cit. 2019-08-18]. Dostupné z: https://www.imysleni.cz/ucebnice/robotika-ucebnice-pro-stredni-skoly

49 NOVÁK, Milan a Jiří PECH. (2019). *Robotika: učebnice pro střední školy - Micro:Bit* [online]. ©2019 [cit. 2019-08-18]. Dostupné z: https://www.imysleni.cz/ucebnice/robotika-ucebnice-pro-stredni-skoly-micro-bit

3.2 LEGO Mindstorms

LEGO Mindstorms je pravděpodobně jednou z nejznámějších a nejkomplexnějších stavebnic běžně dostupných na českém trhu. v současné době můžete narazit na **druhou verzi NXT, i nejnovější třetí verzi EV3**, přičemž v obou případech se jedná o v podstatě tradiční **LEGO Technic, rozšířené o centrální řídící jednotku, širokou škálu senzorů pro vnímání svého okolí a s dvěma typy motorů pro pohyb** (viz obrázek 50 níže).

Mezi základní senzory patří ultrazvukový senzor, dotykové, optické a zvukové čidlo. **Ultrazvukový senzor** snímá vzdálenost od nejbližší překážky před senzorem, je však relativně nepřesný. **Dotykové čidlo** funguje na principu tlačítka a snímá se jeho zmáčknutí. **Zvukové čidlo** reaguje na určitou intenzitu zvuku, nerozlišuje však žádné detaily. Tento senzor nelze naprogramovat na hlasové příkazy, nastavuje se jen hraniční hodnota hluku, při které se něco stane. Tímto „hlukem“ může být tlesknutí, křiknutí, bouchnutí, či jakýkoliv jiný zvuk. **Optické čidlo** snímá intenzitu světla, které na něj dopadá. Hodnota je udávána v procentech, přičemž 0 je tma a 100 je jasné světlo. Toto čidlo si může samo přisvěcovat a lze tak například snímat barvu povrchu (čidlo rozezná bílý podklad od černého a lze tak robotovi nastavit například pohyb po čáře nebo v určité vymezené oblasti). Motorky zároveň fungují jako **rotační senzory**.

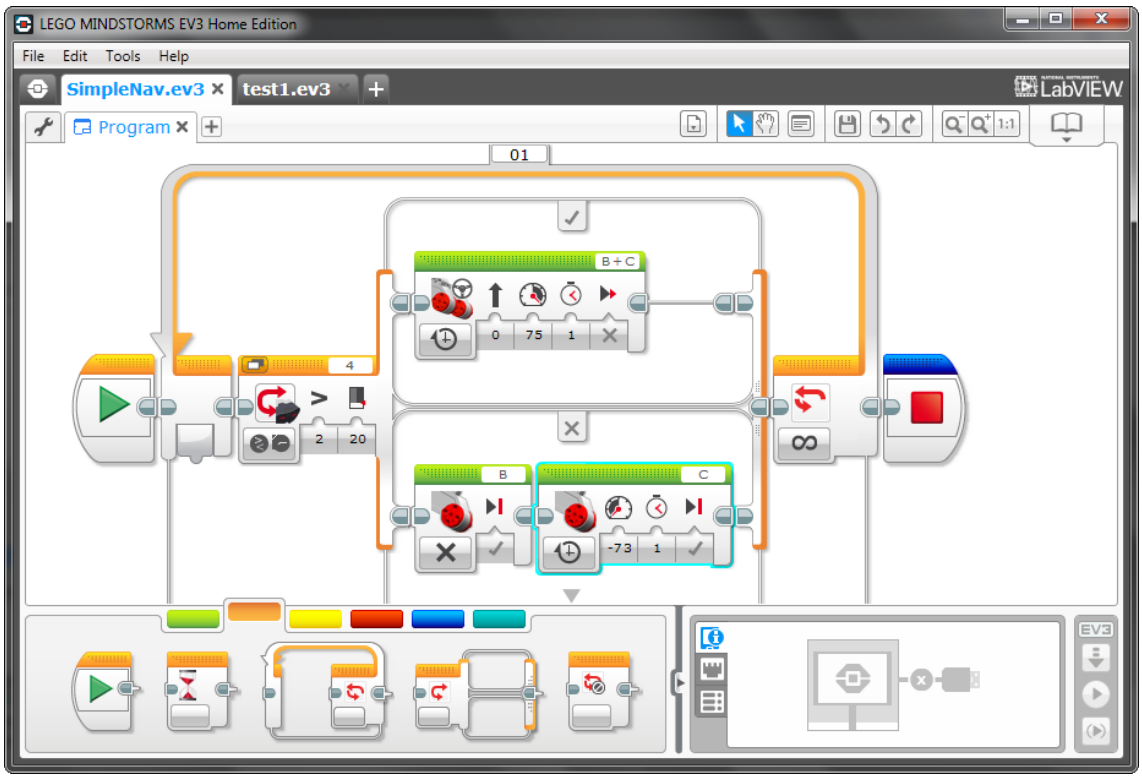
Obrázek 50: Řídící jednotka s motory a senzory (vlevo starší verze LEGO Mindstorms NXT, vpravo nové EV3)

Výše zmíněné senzory lze libovolně kombinovat (omezeni jsme pouze počtem vstupů do řídící jednotky, přičemž ale i toto omezení lze obejít dokoupením extra součástek) a vytvořit cokoliv **dle vlastní představivosti nebo podle návodu**, jako například inteligentní autíčko, robopejska se simulací základního chování, nebo programovatelného krokodýla (vzpomínáte si na obrázek v úvodu?)

Samotná konstrukce, jakkoliv je důležitá a pro žáky bez návodu mnohdy až nepřekonatelně obtížná, však ještě robota nerozhýbe. LEGO Mindstorms lze ovládat větším počtem různých jazyků i pomocí různých prostředí (e.g. C++, Java,

Python, Scratch a další). Základním je ale vizuální programovací prostředí vytvořené přímo firmou LEGO, nazývané ve své starší verzi **LEGO Mindstorms NXT 2.0** a v nové již jen jednoduše **EV3 Software**. Princip práce je v obou těchto prostředích také založený na zapojování předem připravených grafických bloků (kostiček) a pro žáky, kteří již jsou s nějakým dětským programovacím jazykem seznámeni, nečiní zpravidla žádné potíže (nejtěžší je pravděpodobně práce s daty).

Zdarma je navíc k dispozici i několik mobilních aplikací (pro Google Android i Apple iOS), každá s trochu jiným účelem. LEGO Mindstorms Commander je určen pro přímé ovládání robota na mobilu připojeném přes Bluetooth. LEGO Mindstorms Education EV3 a LEGO Mindstorms Programmer jsou zjednodušené verze počítačového programovacího prostředí EV3 Software a poslední aplikace, LEGO Mindstorms Fix the Factory, je vzdělávací logická hra na stejném principu jako nám již známý Lightbot.



Obrázek 51: Nové programovací prostředí EV3 Software (zdroj: https://www.lego.com/en-us/themes/mindstorms/learntoprogram)

Pravděpodobně největší komplikací pro nasazení na školách je **vysoká pořizovací cena** stavebnic, přičemž nejvyšší ideální počet jsou dva žáci na stavebnici. Základní souprava LEGO Mindstorms 45544 EV3 Education Edition vychází na zhruba 10 000 Kč a rozšiřující sada dílků na další cca 3 000 Kč. Pro třídu o velikosti 16 až 18 žáků (standardní počet žáků na hodinách informatiky, limitovaný počtem počítačů v učebně) se tedy pohybujeme na zhruba 104 až 117000 Kč, přičemž je ještě nutné dokoupit alespoň čtyři nabíječky a ideálně by měl mít jednu stavebnici i učitel. Budete-li zařizovat takovýto nákup, určitě nechte

školu vypracovat cenovou nabídku od několika prodejců, protože slevy u takovýchto částek mohou být signifikantní. Alternativou je také potenciální klon stavebnice jménem MITU od firmy Xiaomi,[50] který (jakmile/jestli tato firma doladí programovací prostředí a jeho překlad) je podstatně dostupnější.

LEGO Mindstorms je **vhodné i pro projektovou výuku a propojení s dalšími vzdělávacími oblastmi** (ostatně jako veškeré jazyky a prostředí v těchto skriptech). Mezipředmětové propojení zde lze ilustrovat na možnosti vytvářet roboty, demonstrující například různé fyzikální principy.[51, 52] Pro další informace, týkající se výuky za pomoci LEGO Mindstorms, využijte výše zmiňovanou učebnici publikovanou na webu imysleni.cz.

Ačkoliv se jedná o další finanční náklad (konkrétně cca 6 000 Kč), v praxi se autorovi osvědčila rozšiřující sada LEGO Mindstorms EV3 Space Challenge,[53] která byla použita v rámci zájmového útvaru v počtu dvou kusů, kde s každou sadou pracovali dva žáci. v této sadě žáci plní celkem sedm předem připravených misí, napodobujících reálné situace řešené v rámci vesmírného výzkumu a možnosti mezipředmětových vztahů se samy nabízejí.

3.3 Ozobot Bit 2.0 a Ozobot EVO

Jedním z projektů, který v úvodu této kapitoly zmíněn nebyl, je Ozobot. **Pro tohoto robota ucelený vzdělávací materiál v rámci projektu PRIM nevznikl** a blíže se s ním seznámíte v těchto skriptech v celkovém rozsahu dvou lekcí.

Ne všichni roboti musí být stavebnicového charakteru a umožňovat vysoce komplexní konstrukce, a přesto být využitelní pro širokou škálu věkových skupin. **Ozobot je v podstatě malá kulička o průměru 2,5 cm s kolečky a senzory**, přičemž druh a množství senzorů závisí na zakoupené verzi. v současné době jsou na trhu dostupné verze dvě, a to Ozobot Bit 2.0 (dále již bez čísla) a Ozobot EVO.

Cenově dostupnější je Ozobot Bit, jehož cena se běžně pohybuje kolem 1 500 Kč , zatímco verze EVO vyjde na zhruba 3 500 Kč, tedy více jak dvojnásobek

50 LIANG, Dong. (2017). Mitu Robot - is this an alternative to Lego EV3? In: *Medium* [online]. Feb 2, 2017 [cit. 2019-08-15]. Dostupné z: https://medium.com/@dongliang/mitu-robot-is-this-an-alternative-to-lego-ev3-7b2b1dc2a91c

51 COUFAL, Petr. (2010). *Využití stavebnice LEGO ve výuce fyziky*. Bakalářská práce. Univerzita Palackého, Přírodovědecká fakulta, Katedra experimentální fyziky, 54 s. Vedoucí práce RNDr. Pavel Krchňák, Ph.D.

52 COUFAL, Petr. (2014). *Využití stavebnice LEGO Mindstorms ve výuce*. Bakalářská práce. Univerzita Hradec Králové, Přírodovědecká fakulta, Katedra informatiky, 54 s. Vedoucí práce Doc. RNDr. Štěpán Hubálovský, Ph.D.

53 LEGO® MINDSTORMS® Education EV3 Space Challenge. In: *LEGO Education* [online]. [cit. 2019-08-10]. Dostupné z: https://education.lego.com/en-gb/support/mindstorms-ev3/space-challenge

(ceny k datu 08/2019). Obě verze mohou být ovládány pomocí čar a barevných kódů na papíře i pomocí online prostředí OzoBlockly, schopnosti verze Bit jsou však výrazně limitovanější. Základní rozdíly naleznete v tabulce 12.

Tabulka 12: Porovnání vybraných vlastností verzí Ozobot Bit a Ozobot EVO

Vybrané parametry	Ozobot Bit	Ozobot EVO
Cena	Cca 1 500 Kč	Cca 3 500 Kč
Optické senzory na podvozku	ANO (5x)	ANO (7x)
Počet LED světel	1	7
Doprovodné aplikace	ANO (2x)	ANO (3x)
Bluetooth	NE	ANO
Zabudovaný repráček	NE	ANO
Senzory vzdálenosti	NE	ANO (4x)
Aktualizace firmware	NE	ANO

Ačkoliv se může zdát, že verze Bit nic neumí, rozhodně tomu tak není. Velké množství aktivit ukázaných v následujících dvou kapitolách je použitelné pro obě verze Ozobota. Je-li však ve finančních možnostech školy zakoupit pokročilejší roboty EVO, určitě se to vyplatí, protože **cokoliv je možné provést s Ozobot Bit je vždy automaticky možné i s verzí EVO**.

Obrázek 52: Ozobot Bit (vlevo) vs. Ozobot EVO (vpravo)

Z hlediska aplikací jsou pro Google Android i Apple iOS zdarma dostupné celkem tři různé verze (čtyři, počítáme-li i aplikaci pro hromadnou aktualizaci firmwaru). **Aplikace *Ozobot Bit*** vám umožní vytvářet mapy, po kterých může na tabletu Ozobot jezdit a oproti papíru je zde výrazně zjednodušená práce v případě nějaké chyby v trase nebo barevném kódu. Kromě kreslení vlastních tras je zde i možnost využít interaktivity tabletu a splnit několik připravených výzev, bludišť a her jak

pro jednoho, tak pro dva hráče. Navzdory lehce zavádějícímu pojmenování této aplikace ji můžete bez jakýchkoliv komplikací využít i v kombinaci s Ozobot EVO. Stejně tak další **aplikace *Ozobot Bit Groove***[54] je použitelná i pro EVO. Zde je očividný **mezipředmětový vztah s hudební výchovou**, protože smyslem této aplikace je vytvářet taneční choreografie pro vašeho robota (i více robotů zároveň).

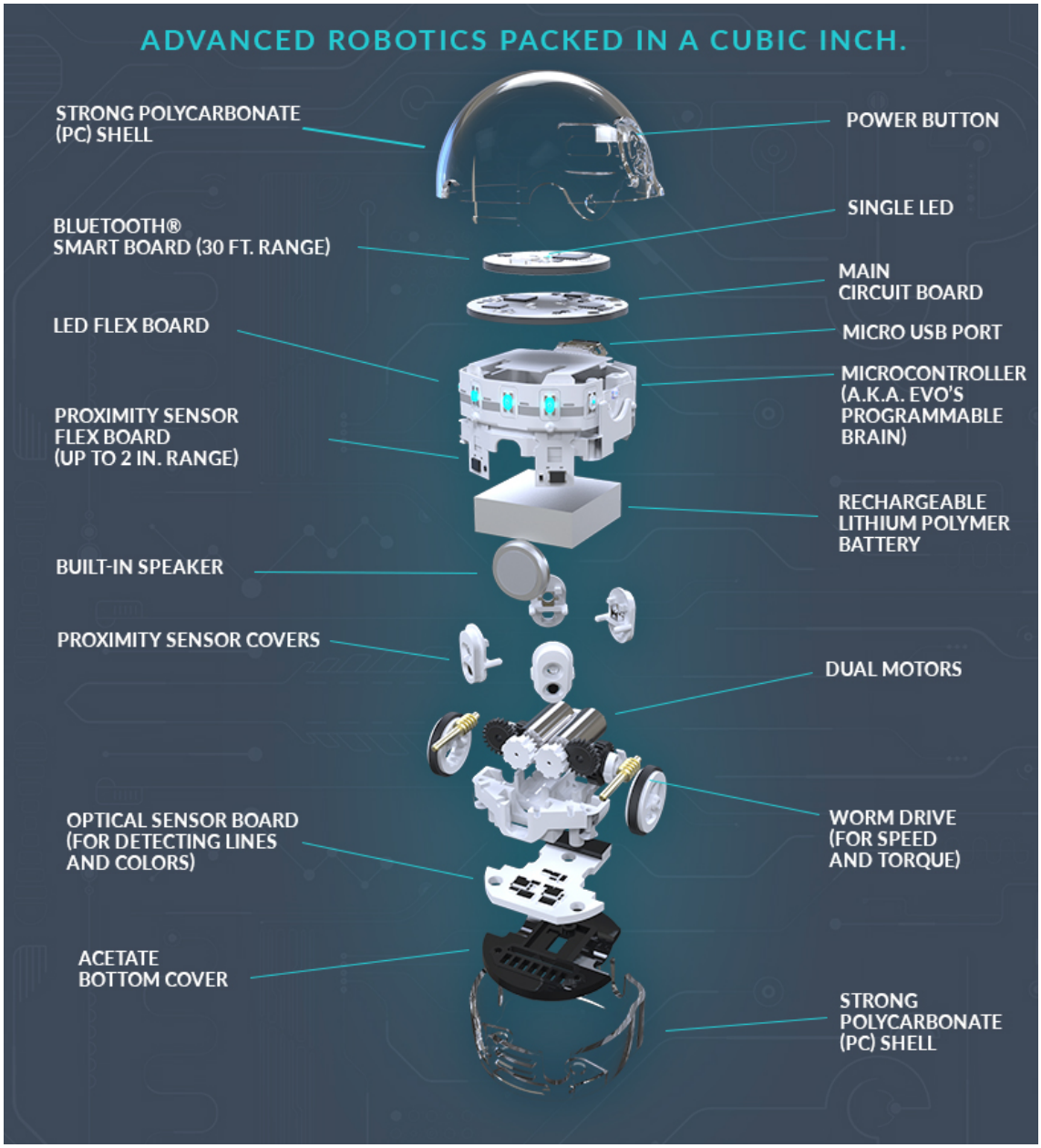
Zatímco obě verze Ozobota můžete řídit jak pomocí barevných čar, tak kódování v OzoBlockly, připojení Ozobota EVO přes Bluetooth k mobilnímu telefonu či tabletu výrazně rozšiřuje možnosti ovládání vašeho robota. Za tímto účelem je nutná **aplikace Evo by Ozobot**, která pro levnější verzi Bit právě z důvodu absence Bluetooth není určena. Tato aplikace vám umožňuje přímé ovládání Ozobota EVO pomocí virtuálního joysticku, obdobně jako například autíčko na dálkové ovládání. Můžete si také vyzkoušet nastavování jednotlivých LED světel, jejich barev a efektů a je zde i možnost přehrávání zvuků z knihovny všech zvuků, které váš Ozobot umí vyprodukovat. Jestliže je vaše mobilní zařízení zároveň připojeno na internet, aplikace podstatně zjednodušuje nahrávání a spouštění vámi připravených programů z prostředí OzoBlockly (viz dále v kapitole OzoBlockly).

Kromě několika jednoduchých her je zde v neposlední řadě ještě možnost **nahrávání nového firmwaru**, který neustále rozšiřuje schopnosti těchto robotů, a to například přidáním nových zvuků nebo skrytých funkcí. Tyto skryté funkce, nazývané jejich tvůrci *triky*, byly přidány v updatu firmwaru 1.7 v listopadu 2017 a spouštěly se a přepínaly přikrytím všech čtyř senzorů vzdálenosti. Ozobot potom buď pronásledoval předmět (třeba prst) v dosahu jeho senzorů, nebo naopak před překážkou (třeba rukou) utíkal a nebo stál na místě a hrály různé tóny, podle toho, u kterého vzdálenostního senzoru se objevila překážka (opět třeba prst). Vývoj této aplikace a Ozobota EVO se ale nezastavil a v současné verzi 2.2.268 z prosince 2018 byly tyto skryté funkce upraveny a hlavně konečně přibyla možnost vytvářet OzoBlockly kód přímo v rámci aplikace (přičemž do této doby se muselo programovat výhradně přes prohlížeč, odkud se přes synchronizaci účtu výsledný kód poslal do aplikace a odtud se teprve mohl spouštět).

Obrázek 53: Obsah starého (vlevo) a nového (vpravo) balení Ozobot EVO

54 Ukázkové video dostupné na: https://www.youtube.com/watch?v=JrY2AyzI3mc

Součástí původního modrého balení Ozobota EVO býval nabíjecí USB kabel (ale bez koncové nabíječky do zásuvky), ochranný obal, sada čtyř fixek (černá, zelená, červená a modrá), oboustranný hrací plán pro testování základních schopností a funkcí Ozobota, ochranný transportní sáček a samozřejmě sám Ozobot. Podle textu přímo na oficiálním obchodu výrobce ale došlo ke změně v balení. Krabice už není modrá, ale oranžová, a s transportním sáčkem a hracím plánem již počítat nelze. Nejednalo se o nijak důležité součásti celého setu, ale jejich absence zejména při počátečním objevování a pokusování zamrzí.



Obrázek 54: Konstrukce verze Ozobot EVO (zdroj: https://shop.ozobot.com/products/evo-educator-kit)

Ozobot nemá žádné ručičky a ani žádné jiné pohyblivé části, vyjma dvou koleček. Jeho možnosti se tak mohou zdát omezené, ale **principiálně je takovýto typ robota použitelný i v reálném světě** a v praxi je skutečně možné najít roboty, kteří jsou takřka klonem Ozobotů. Konkrétně se jedná o skladištní roboty využívané například v obrovských komplexech firmy Amazon[55] nebo Alibaba.[56] Specificky tato dvě citovaná **ukázková videa jsou velice užitečná pro propojení teorie s praxí** a zdůraznění faktu, že Ozobot není jen na hraní, ale i na učení a pro pochopení toho, jak svět kolem nás funguje. Odmyslíme-li si pohyb po čáře, potom jsou ekvivalentem z běžného každodenního života chytré vysavače, jejichž pohyb také můžeme nasimulovat v Ozoblockly.

Základní premisou Ozobota je však pouze jeho schopnost pohybu. Ve verzi EVO se nabízejí možnosti využití zvukové a světelné signalizace a reakce na překážky. Stále se ale může zdát, že to je málo. Pro zatraktivnění této „hračky" pro širší veřejnost tak vznikla všelijaká rozšíření, jako například spojení se světoznámou komiksovou firmou a vydání **edice Marvel's The Avengers**.[57]

Obrázek 55: Ozobot s oblečky Marvel's The Avengers

Bohužel se jednalo o velice limitovanou edici, která v současné době již v běžné nabídce není ani v zahraničí (a v ČR nikdy nebyla). Inspirace a motivace je to však stále použitelná. **Na Ozobota si totiž můžete (mimo oblast senzorů) nalepit**

55 TECH INSIDER. (2016). Inside An Amazon Warehouse On Cyber Monday. In: *YouTube* [online]. 28. 11. 2016 [cit. 2019-08-18]. Dostupné z: https://www.youtube.com/watch?v=qRQwkJLRfWw

56 BUSINESS INSIDER. (2017). Inside Alibaba's smart warehouse staffed by robots. In: *YouTube* [online]. 20. 9. 2017 [cit. 2019-08-18]. Dostupné z: https://www.youtube.com/watch?v=FBl4Y55V2Z4

57 JR TOY COMPANY. (2017). Marvel's Avengers and Ozobot Evo From JR Toy Company. In: *YouTube* [online]. 22. 3. 2017 [cit. 2019-08-19]. Dostupné z: https://www.youtube.com/watch?v=3mzIX4IutEU

všelijaké nálepky a očička, vlasy a rohy, apod., což také sama firma aktivně podporuje. Zde se nabízí další možný přesah do předmětu výtvarná výchova.

Ukázky práce s Ozoboty a možnosti kreativního využití tohoto robota spolu s velkým množství návodů naleznete například na YouTube.[58] Přímo tvůrci Ozobota prodávají kromě všelijakých nálepek také například ochranné obaly s poutkem třeba na klíčenku a další drobnosti. Pravděpodobně nejzajímavější však je sada jménem **Ozobot Bit Construction Kit**, která je v ČR dostupná v ceně kolem 400 Kč. Pro tuto sadu také tvůrci Ozobota vytvořili jednoduché začátečnické výzvy, které jsou včetně tisknutelných podkladů dostupné na oficiálním webu.[59]

Obrázek 56: Rozšiřující sada Ozobot Construction Kit

Tato rozšiřující sada **přidává Ozobotovi manipulátor připomínající vidle vysokozdvižného vozíku** a tím také možnost interakce Ozobota se svým okolím. Zatímco samotný kulatý Ozobot může něco přesunout, posunout, zatlačit nebo odvézt jen těžko, tyto vidle všechny zmíněné akce umožňují a signifikantně tak rozšiřují repertoár úloh, které žákům můžete zadat. Jelikož se jedná jen o jednoduchý nástavec, mohou si děti podobné rozšíření vytvořit sami ze čtvrtky nebo kartonu (mezipředmětový vztah geometrie a rýsování).

Stejně jako může Ozobot něco tlačit, může něco i táhnout. Nabízí se proto tvorba vlastního přívěsu/návěsu, který může být pomocí šikmé plošinky a dalšího Ozobota s vidlemi rovnou i naložen. k dalšímu rozšíření schopností Ozobota je možné přidat nástavec, který by držel tužku a umožnil tak robotovi kreslit na povrch, po kterém jezdí. Takovýto nástavec si můžete vytvořit jak z kartonu, tak pomocí 3D tiskárny, pro kterou jsou podklady na internetu zdarma k dispozici.[60]

58 OZOBOT. (2015). Ozobot Bit Adventures. In: *YouTube* [online]. 14. 12. 2015 [cit. 2019-08-19]. Dostupné z: https://www.youtube.com/watch?v=KX1wuSiwViA

59 Ke stažení na adrese: https://ozobot.com/play/construction-kit-games

60 MRBENBRITTON. (2018). Ozobot Bit, side slung pen holder. In: *Thingiverse* [online]. Dec 28, 2018 [cit. 2019-08-20]. Dostupné z: https://www.thingiverse.com/thing:3317899

3.4 Ozokódy, aneb práce s tužkou a papírem

Jak již bylo v minulé kapitole řečeno, Ozobota je možné ovládat více způsoby. Základními způsoby jsou čáry a barevné kódy na papíře nebo tabletu a nebo tvořením programů v prostředí OzoBlockly. Ozobot EVO navíc může být ovládán aplikací přes Bluetooth z mobilního zařízení. v této kapitole se seznámíte s první možností, tedy „programováním bez počítače," jen za pomoci robota, fixek a papíru.

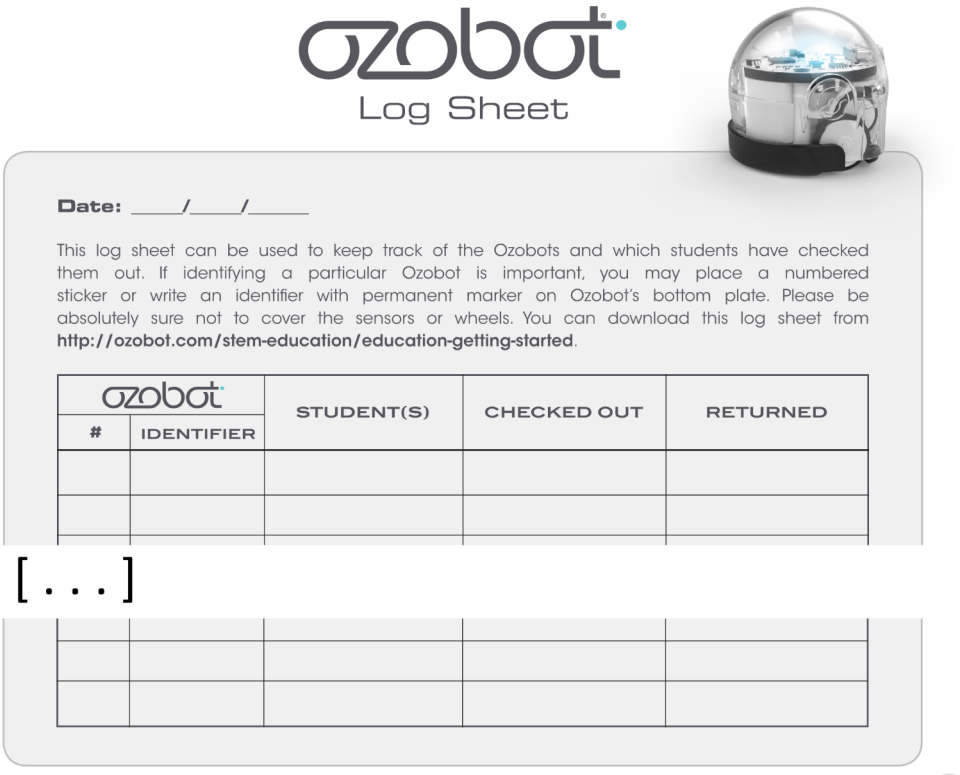
Práce s Ozoboty je relativně jednoduchá a přímočará, přesto se setkáte s větším množstvím problémů a komplikovaných situací, které jako učitel musíte vzít v potaz a být na ně připraveni. v následující podkapitole vás čeká téma přípravy robotů na téměř jakoukoliv hodinu, doprovázené řadou praktických rad.

3.4.1 Příprava na hodinu a možné problémy

Hodina, na kterou vám stačí přinést jen krabice s Ozoboty, papíry a fixy, je v podstatě jen jedna (a i to platí jen, jestliže jste již s vašimi Ozoboty pracovali a máte je připravené a ozkoušené). Úplně první seznámení žáků s Ozoboty si vystačí téměř bez pomůcek, i když i na tuto hodinu byste měli mít připravené alespoň nějaké úvodní povídání a motivační video. Na všechny ostatní hodiny potřebujete „něco extra." Následující rady byly nasbírány v rámci vedení zájmového útvaru a mnoha ukázkových hodin, určených pro dospělé i děti, a to včetně skupin s věkově heterogenním složením.

- ✔ **Přípravy nových Ozobotů EVO** – Jestliže se jedná o úplně nové roboty, je **nutné je plně nabít a u všech robotů zaktualizovat firmware!** V opačném případě nemusí fungovat ani některé základní rozeznávání Ozokódů. Máte-li robotů více, vyplatí se využít aplikace pro dávkovou instalaci. Velice často se aktualizace nepovede, i když je robot plně nabitý. Pomáhá aktualizovat robota ve chvíli, kdy je připojený na nabíječce. Dalším doporučením je **ztlumit hlasitost robotů**. Každý Ozobot si téměř neustále žbrblá, žbleptá a píská, což v plné třídě velice ztěžuje práci.
- ✔ **Bezpečnostní opatření** – Přestože je polykarbonátová schránka odolná, ve školním prostředí **se doporučuje využívat dodávaná silikonová čepička.** Robot pak třeba není tak pěkný, ale silná vrstva silikonu tlumí nárazy třeba při pádu z lavice. Další komplikací je podobnost robotů. Řešením je **každému robotovi přiřadit nějaké číslo a na robota toto číslo viditelně napsat nebo nalepit**. Čísla pomáhají udržovat přehled o odevzdaných robotech, ale chcete-li přidat ještě jednu vrstvu kontroly, můžete **využít originálního volně dostupného zápisového listu**, kde si každý žák podepisuje zapůjčení a vrácení konkrétního robota.[61]

61 Volně ke stažení zde: https://files.ozobot.com/stem-education/ozobot-log-sheet.pdf



ozobot
Log Sheet

Date: _____/_____/_____

This log sheet can be used to keep track of the Ozobots and which students have checked them out. If identifying a particular Ozobot is important, you may place a numbered sticker or write an identifier with permanent marker on Ozobot's bottom plate. Please be absolutely sure not to cover the sensors or wheels. You can download this log sheet from http://ozobot.com/stem-education/education-getting-started.

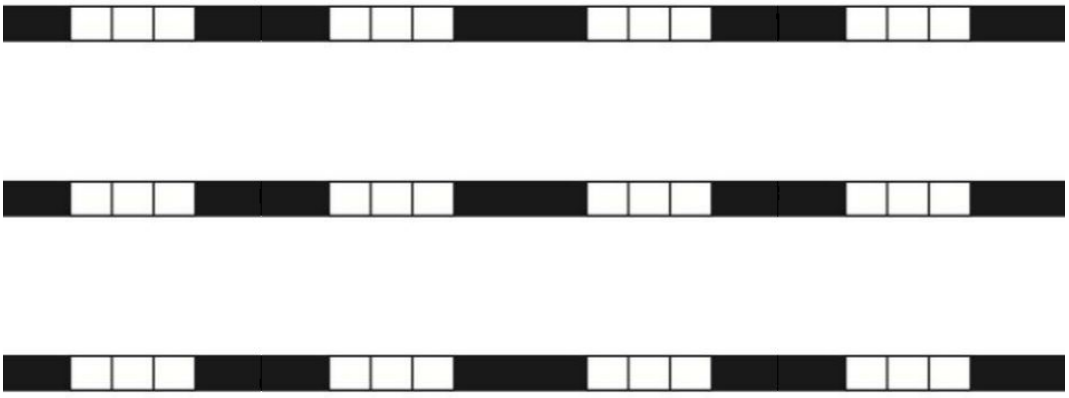
ozobot #	IDENTIFIER	STUDENT(S)	CHECKED OUT	RETURNED

[. . .]

Obrázek 57: Originální zápisový list (Log Sheet)

- ✔ **Přípravy a kontrola Ozobotů před hodinou** – Ideálně den předem je **nutné zkontrolovat, jak jsou nabité baterie**. Míru nabití indikuje barva LED, které se na Ozobotovi rozsvítí po připojení na nabíjecí USB kabel (plné svítí bez blikání sytě zelenou barvou). U robotů **také dochází k samovolnému vybíjení**, takže nepoužíváte-li roboty delší dobu, rozhodně počítejte s nutností jejich nabití.
- ✔ **Hromadné nabíjení** – Přestože se roboti nabíjejí jen zhruba hodinu, máte-li jich nabít více jak deset, jedná se o nepříjemný úkol, který vám **výrazně zjednoduší nějaký nabíjecí USB rozbočovač**. Počítejte s touto položkou už rovnou při nákupu Ozobotů.
- ✔ **Příprava tištěných podkladů a barevných fix** – Pro lekce bez použití PC (i některé výzvy na OzoBlockly) je nutné mít připravené tištěné podklady. Těchto podkladů je většinou větší množství a jejich **tištění těsně před hodinou není dobrý nápad**. Velice zákeřným problémem je také vypisování fix, kdy zejména černá barva dochází velice rychle a většinou všem najednou. Ozobot sice funguje s jakýmikoliv fixami, ideální jsou ale fixy se zešikmeným seříznutým hrotem, které žákům napomáhají ve správné a konstantní šířce čáry, kterou Ozobot vyžaduje.

- ✔ **Výdrž baterie a párová práce** – Ozobot se nabíjí zhruba hodinu a stejně tak dlouho i vydrží. **Nepočítejte s tím, že by byl použitelný pro celou dvouhodinovku**. Stejně jako u LEGO Mindstorms, je i zde vhodné využít párovou práci, přičemž jakýkoliv robot, který vám zbude navíc, se téměř vždy hodí při řešení problémů. Občas Ozobot jednoduše nefunguje, i když jste všechno poctivě zkontrolovali a žáci vše řeší naprosto korektně. v takovém případě je nejlepší dvojici žáků s problematickým Ozobotem dát nějakého záložního a podivné technické problémy řešit až po hodině.
- ✔ **Pomocné papírky na Ozokódy** – Při plnění úloh pomocí doplňování barevných Ozokódů například do bludiště počítejte s tím, že i **jeden špatně doplněný kód zpravidla vede k zacyklení Ozobota v bludišti** a nutnosti buď vypracovat celý papír úplně od začátku, nebo kód překrýt prázdným papírem. Tvůrci Ozobota prodávají překreslitelné **papírky, které lze na potřebná místa přiložit** a poté znovu využít. Takové papírky si však můžete vytisknout sami (a případně je zalaminovat). Žáci poté obdrží několik pomocných papírků spolu se zadáním úlohy a pomocí nich si nejprve mohou trasu vyzkoušet a teprve poté ji nakreslit „naostro." Tento postup navíc podporuje správný návyk nutnosti přemýšlet před vykonáním nějaké akce (nejprve algoritmizace a až poté programování). **Nezapomeňte si nechat dostatek prostoru pro podržení papírku** (jakmile EVO zaregistruje překážku, tak se zastaví).



Obrázek 58: Ukázkové pomocné papírky pro testování trasy řízené Ozokódy

- ✔ **Nesdělujte žákům řešení** – Jedná se o logickou záležitost, kterou je přesto nutné sem napsat. Žáci by na řešení měli přijít sami a jestliže někde udělají chybu, měli by ji také sami odhalit. Vaše pomoc má žáky jen nasměrovat správným směrem (třeba otázkou nebo zopakováním nějakého pravidla, které žáci porušili a které jim **někde** v rámci jejich řešení vše kazí).

✔ **Veškeré instrukce a teorii na začátku hodiny** – Všechny informace, které chcete žákům frontálně předat (tj. nějakou teorii k robotům a jejich využití v praxi, instrukce pro práci s robotem a průběh hodiny, apod.) předejte na začátku hodiny PŘED rozdáním Ozobotů. Po obdržení robota by již žáci měli pracovat sami a vlastním tempem. Strhnout na sebe pozornost všech žáků v celé třídě je v průběhu hodiny již velice obtížné a většinou kontraproduktivní (narušíte tím žákům koncentraci a žáci hluboce zabraní do řešení aktuálního problému vás stejně nebudou registrovat).

✔ **Spolehlivý pomocník** – Hodiny s Ozokódy i OzoBlockly jsou velice **vstřícné individuálnímu přístupu** k jednotlivým žákům. Učitel zpravidla jen kontroluje průběh hodiny a pomáhá řešit dílčí problémy (velice často technického charakteru). Často se vám ale (zejména z počátku vaší praxe) může stát, že vám **přebíhání od jednoho žáka ke druhému rozhodí koncentraci a nebudete se zvládat soustředit na vše najednou**. Umožňuje-li to vaše škola, tyto hodiny jsou **naprosto perfektní pro tandemovou výuku** (= spolupráce dvou učitelů v jedné třídě). Jestliže toto možné není, domluvte se předem s vybraným rychlejším žákem, který vám bude v průběhu hodiny pomáhat a **který hlavně bude na konci hodiny hlídat a pomáhat s kontrolou odevzdání všech robotů!** Roboti jsou malinkatí a úmyslná krádež či jen neškodné zapomenutí robota odevzdat se zpětně může řešit extrémně problematicky.

Ozokódy i Ozoblockly lze provozovat jen s fixami, papírem a počítačem, ale doprovodná aplikace Evo by Ozobot je velice užitečná a možnost využívat tvorbu tras elektronicky a nechat vaše roboty jezdit po tabletech nebo dotykových noteboocích práci s Ozobotem posouvá ještě dále. Takže přestože tablet není vůbec potřeba, platí, že máte-li možnost tablety nějakým způsobem získat, neváhejte a využijte ji. Tablet je kamarád.

3.4.2 Ukázková (dvou)hodina práce s Ozokódy

V této podkapitole se seznámíte s úvodní lekcí, jejíž účelem je osvojit si základy práce s Ozokódy. Celou lekci v těchto skriptech naleznete v **podobě okomentované bodové přípravy na hodinu. Využito je několik volně dostupných materiálů doplněných vlastními nápady**. Lekce byla testována při více jak deseti příležitostech, a to na žácích od třetí třídy až po studenty středních škol (v závislosti na konkrétní věkové skupině jsou nutné jen velice jednoduché modifikace zpravidla zahrnující vyřazení několika úloh). Současná podoba této úvodní lekce je výsledkem několika modifikací na základě získaných zkušeností.

Programování na papíře, aneb Ozokódy

Vhodné pro:	Žáky od 3. třídy základní školy až po (ne)odborné střední školy		
Předmět:	Informatika a výpočetní technika	**Rozsah:**	2 vyučovací hodiny (90 minut)
Učivo:	Úvod do problematiky algoritmizace a programování pomocí Ozokódů		
Cíle hodiny:	**1.)** Žáci popíší, z čeho se Ozobot skládá a vysvětlí, jaká jsou jeho omezení. **2.)** Žáci nejprve zformulují řešení úlohy a poté jej demonstrují (tedy rozlišují návaznost algoritmizace → programování). **3.)** Žáci identifikují různé možnosti řešení a vyhodnotí, které je nejvhodnější.		

Seznam pomůcek a podkladů (obrázky za plánem lekce):

0. Ozoboti, prázdné papíry, barevné fixy, elektronická zadání a vytištěná bludiště

1. YouTube video „Inside An Amazon Warehouse On Cyber Monday“ dostupné na https://www.youtube.com/watch?v=qRQwkJLRfWw

2. YouTube video „Inside Alibaba's smart warehouse staffed by robots“ dostupné na https://www.youtube.com/watch?v=FBl4Y55V2Z4

3. Konstrukce Ozobota (lze využít obrázek 54 z těchto skript, nebo z oficiální příručky pro učitele https://files.ozobot.com/stem-education/ozobot-educators-guide.pdf na straně 4)

4. Pravidla pro kreslení čar a Ozokódů (perfektní česká verze instrukcí dostupná na adrese https://www.easystore.cz/manualy/ozobot/Ozobot-Rychle-tipy-CZ.pdf)

5. Mini-test na Ozokódy k promítnutí (z oficiálního kurzu https://storage.googleapis.com/ozobot-lesson-library/6-8-basic-training-color-codes/6-8-Basic-Training-Student-Handouts-Color-Codes.pdf na straně 8)

6. „Ozořidičáky“ k vytištění (hned za mini-testem na stranách 9 a 10)

7. Podpisový arch zápůjček (zde https://files.ozobot.com/stem-education/ozobot-log-sheet.pdf)

8. Zadání prvních úloh k promítnutí (zadání vytvořené a používané přímo pro tyto lekce níže)

9. Seznam všech použitelných Ozokódů (dostupné v české i anglické verzi, novější AJ design zde https://files.ozobot.com/stem-education/ozobot-color-codes.pdf)

10. Podklady bludišť k tištění (jedná se o celkem tři úlohy, konkrétně:

10a. Winter Scavenger Hunt (https://portal.ozobot.com/lessons/detail/winter-scavenger-hunt)

10b. Průvod prezidentů (česká verze zde https://www.easystore.cz/manualy/ozobot/lekce/Ozobot%20aktivita%20-%20průvod%20prezident

10c. Cesta k obchodu (https://www.easystore.cz/manualy/ozobot/lekce/Ozobot%20z%C3%A1kladn%C3%AD%20tr%C3%A9nink%20-%20lekce%201.pdf strana 9)

11. Rozšiřující úlohy pro rychlé žáky (zadání vytvořené a používané přímo pro tyto lekce níže)

12. Motivační certifikát (https://files.ozobot.com/stem-education/ozobot-certificate.pdf)

Čas	Fáze	Činnost učitele	Činnost žáka
3	Expozice	Vyučující se zeptá žáků, **jestli znají nějaké příklady robotů** či jiných počítačem ovládaných zařízení ze svého života (ideální je, když zazní robotický vysavač). Navazuje dotaz, jestli **jsou roboti chytří a proč dělají to, co dělají** (očekávaná odpověď: nejsou a dělají výhradně to, co se jim „řekne,“ takže chytří musí být programátoři)	Žáci odpovídají na položené otázky, případně (jsou-li toho schopni) o odpovědích diskutují.
6	Motivace	Vyučující pustí **dvě motivační videa** o robotech z obchodů Amazon a Alibaba (**viz podklady 1 a 2 výše**) a položí otázku, proč je lidé využívají.	Žáci sledují videa a poté se snaží zformulovat odpověď, že roboti usnadňují a zrychlují lidem práci.

3	Expozice	Vyučující se zeptá, co je to „programování“ a případně navede žáky na odpověď, že se jedná o **tvorbu nových programů pomocí sledu instrukcí** od programátora pro počítač, který má podle nich něco přesně nějakým způsobem provést (viz poznámky 1 a 2 níže)	Žáci odpovídají a případně se snaží zrealizovat instrukci „jdi k tabuli.“
5	Expozice	Vyučující se zeptá, z **čeho se roboti skládají** (očekávaná výsledná odpověď řídící jednotka, senzory a motory). Vyučující dále promítne složení Ozobota (**viz podklad 3 výše**) a zeptá se žáků, **jaká součástka má jaky účel**.	Žáci společně odvodí funkce jednotlivých senzorů.
5	Expozice	Vyučující vysvětlí omezení optických senzorů na podvozku a **ukáže konkrétní pravidla pro jejich kalibraci a následné kreslení čar a barevných kódů**. (**viz podklad 4 výše**)	Žáci poslouchají a případně se doptávají.
6-8	Fixace / Diagnóza	Vyučující pro kontrolu pochopení pravidel **promítne test na platnost Ozokódů** (pro výrazné urychlení práce NEtisknout, ale jen promítnout a vyvolávat žáky, **(viz poznámka 4 níže a podklad 5 výše**)	Vyvolaní žáci odpovídají, jestli je daný kód platný nebo ne a vždy vysvětlují, proč tomu tak je.
2	Expozice	**Vyučující rozdělí žáky do dvojic** (téměř nikdy není dobrý nápad nechávat žáky vytvářet si dvojice podle sebe) a rozdá „řidičáky“ na Ozobota (**podklad 6**) spolu s Ozoboty, bílými papíry a fixami. Žáci při převzetí Ozobota podepisují zápisový arch (**podklad 7**).	Žáci si přesednou do určených dvojic (viz poznámka 3). Poté jeden ze dvojice převezme robota s pomůckami a podepíše arch.
15	Fixace	Vyučující promítne první sadu úloh (**podklad 8**), znovu projde s žáky proces kalibrace, a to předtím, než rozdá seznam všech Ozokódů, jinak tyto úlohy nemají smysl. **(viz poznámka 4 a 5 níže)**	Žáci na čistém bílém papíru dle instrukcí testují Ozokódy, zjišťují a zapisují si jejich účel.
20-30	Expozice / Fixace	Vyučující **nechá žáky rozdat seznam všech Ozokódů a první tištěná bludiště**. Další bludiště si žáci přebírají vždy až po kontrole splnění předcházejícího. (**podklady 9 a 10**)	Ve chvíli, kdy některá dvojice dokončí aktuální úlohu, přihlásí se a čekají na kontrolu a následné zadání dalšího bludiště.
0-12	Expozice / Fixace	Až první dvojice dokončí všechna tři bludiště, promítne vyučující rozšiřující úlohy (**podklad 11, poznámka 6 a 7**)	Žáci samostatně řeší úlohy a připravují si vysvětlení svého řešení pro spolužáky.
10	Fixace / Diagnóza	Patnáct minut před koncem dvouhodinovky vyučující ukončí práci všech žáků a společně zkontrolují správná řešení (v případě více řešení porovnají efektivitu těchto řešení (**viz poznámka 8**), nezapomenout na rozšiřujících úlohy (byly-li řešeny).	Vyvolaní žáci demonstrují jejich řešení (spuštěním Ozobota) a přidávají svůj komentář popisující postup práce. Na závěr shrnout obsah celé dvouhodinovky.
5	Motivace	Pět minut před koncem hodiny **vyučující rozdá certifikáty o absolvování robo-výuky výměnou za Ozoboty** (pomáhá kontrole vrácení robotů).	Žáci vracejí Ozoboty, podepisují navrácení na zápisovém archu a přebírají certifikáty.

Poznámky k realizaci:

1.) Pro ilustraci si mohou žáci představit robota a štěňátko někde na vysoké skále. Oba dva dostanou instrukci „*Jdi*“, co se stane? Žáci většinou správně uhodnou, že robot se rozbije a štěňátko na ně bude koukat jak na hlupáky. Vyučující musí dodat vysvětlení – roboti dělají jen a pouze to, co jim programátor řekne, a to včetně naprosté pitomosti. Je tedy úkolem programátora chování robota naprogramovat. Potom lze pokračovat příkladem, kdy vyučující vyvolá jednoho žáka či žákyni někde z prostředku třídy a řekne jí/jemu „*Představ si, že jsi robot a já programátor a dávám ti jedinou jednoduchou instrukci – dojdi k tabuli.“* Žák či žákyně tak zpravidla učiní, na čemž jde demonstrovat jak by to vypadalo ve skutečnosti. Jestliže by robot byl humanoidní a seděl, nejprve by musel dostat instrukci vstát, potom otočit se směrem do uličky a tak dál, až by se dostal k tabuli. Kdyby to byl velký silný robot a dostal jen instrukci jdi k tabuli, prostě by se rozjel a skrz počítače a lavice se k té tabuli proboural. Zde žáci zpravidla pochopí nutnost přesných instrukcí a jejich rozdrobení do dílčích kroků.

2.) Vyučující v rámci vysvětlování, co to vlastně to programování je a k čemu je to dobré, řekne, že *„informatiku použijete v případech, o které se zajímají chlapci i dívky, jako je zachraňovat životy, pomáhat lidem, spojovat lidi atd. Přemýšlejte o věcech každodenního života, které používají informatiku: mobil, mikrovlnná trouba, počítač, semafory ... všechna tato zařízení potřebují informatiky, aby je pomohli vymyslet a sestrojit. Informatika je umění smíchat lidské myšlenky s digitálními nástroji pro zlepšení našich schopností a sil. Informatici pracují v mnoha rozdílných oblastech: píšou aplikace pro smartphony, léčí nemocné, vytvářejí animované filmy, pracují v sociálních médiích, konstruují roboty, kteří zkoumají jiné planety a ještě daleko více.“* (převzato z: https://hourofcode.com/cz/cs/resources/how-to) Vyučující by ale měl zdůraznit, že podstatou programování jako takového je vytváření nových programů pomocí sledu instrukcí od programátora pro počítač, podle kterých dané

zařízení něco přesně nějakým způsobem provede.

3.) v případě Ozobotů je na místě doporučení párové práce, jedna dvojice na Ozobota je úplně ideální. Nutné ale je, aby žáci uměli ve dvojicích efektivně pracovat – mohou si navzájem radit a spolupracovat spolu, jen musí pracovat relativně potichu, aby nerušili ostatní. Často se stává, že dvojice jsou velice hlučné a v takovém případě je nutný zásah vyučujícího. Žáci by si navíc práci měli rozdělit takovým způsobem, aby oba byli neustále něčím zaměstnání. Dělba práce předpokládá nutnost naplánovat pracovní postup, což je další věc, kterou žáci často neumí a základy smysluplné spolupráce ve dvojicích jsou tak součástí této lekce. Neumí-li žáci takto plánovat, případně je-li dvojice nevyvážená, často je jeden ze žáků pasivní, což je stav, který nechceme.

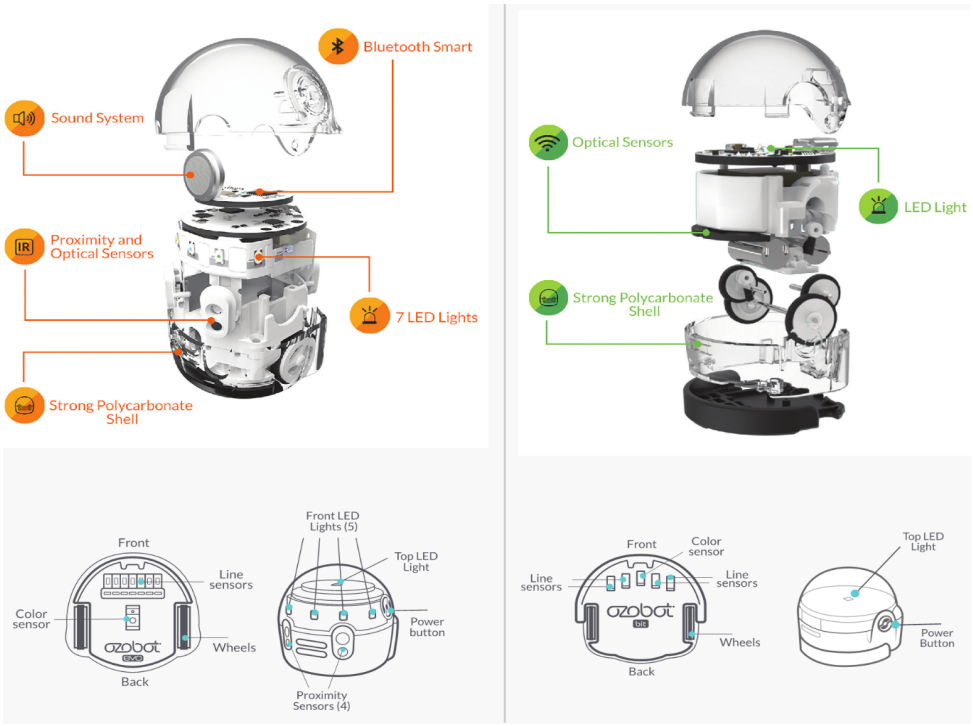
4.) Mini-test i první sadu úloh je na základě odkazovaných podkladů možné i vytisknout, ale výrazně se tak zvýší čas nutný na kontrolní mezikrok v podobě mini-testu a navýší se takzvaný přechodový čas před začátkem práce žáků na první sadě úloh. Promítnutí úloh také šetří velké množství papíru. Úvodní úlohy z přílohy číslo 8, vytvořené pro tuto lekci, v praxi efektivně nahrazují výchozí úlohy pro učení korektního zápisu čar a Ozokódů, které jsou k dispozici na oficiálním webu.

5.) Pozor na kalibraci – tu je nutné provést pro přizpůsobení Ozobota aktuálním světelným podmínkám ve třídě. Na tento důvod zpravidla žáci přijdou sami. v případě změny světelných podmínek (například když se zatáhne nebe nad prosluněnou učebnou) je nutné provést kalibraci znovu. Navzdory jasnému návodu je právě kalibrace „drobnost," která žákům způsobuje velké problémy. Nejspolehlivější postup je následující: Položit Ozobota na černé kolečko, které je zhruba o půl centimetru na každé straně větší než robot. Milimetr nebo dva ho přizvednout nad kolečko a podržet spouštěcí tlačítko po dobu zhruba pěti vteřin (než se mu rozbliká horní LED) a poté robota okamžitě pustit. Robot se zatočí a když mu hlavička blikne zeleně, vše je v pořádku. v opačném případě znovu.

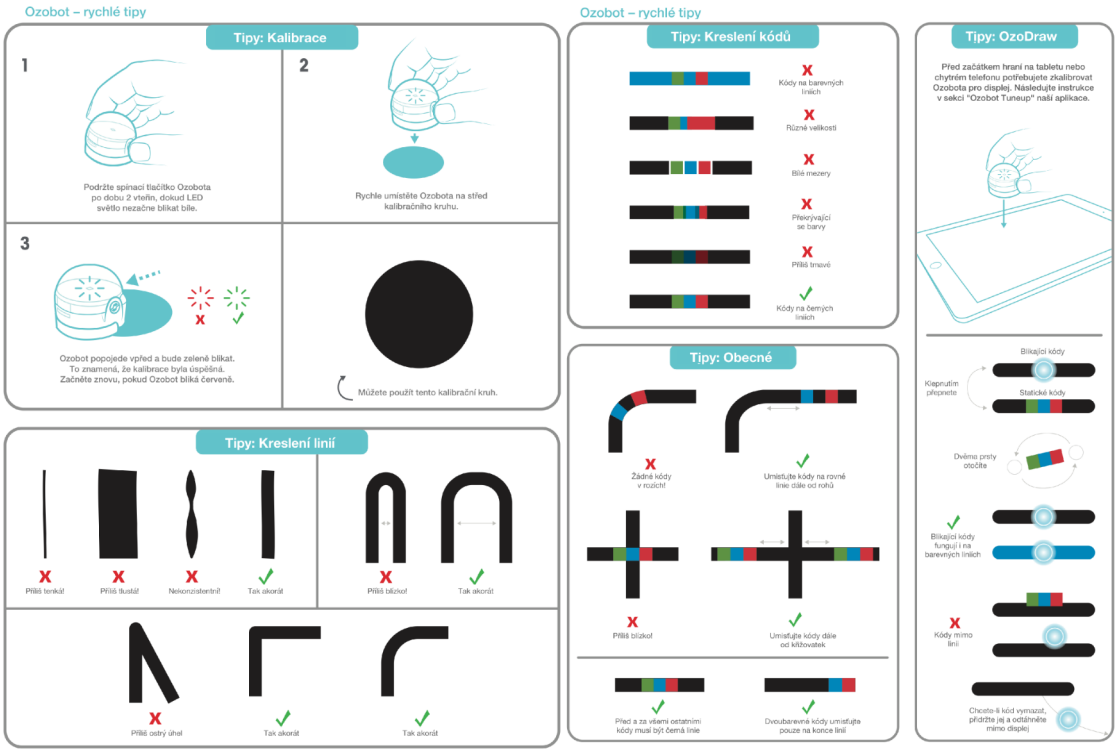
6.) v praxi se k rozšiřujícím úlohám dopracuje jen zlomek žáků a správně je vypracovat stihne málokdo. Protože tyto není nutné tisknout, jedná se o ideální doplněk, který bere v potaz nutnost individuálního přístupu v často velice rozmanitých skupinách. Alternativou je využít rychlé žáky jako vaše pomocníky.

7.) Jestliže před křižovatkou není Ozokód určující kam má robot zatočit, nový směr bude vybrán náhodně (ALE robot se nikdy nevrací zpátky). Tohoto náhodného rozhodování lze využít i při tvorbě aktivit – Trasa může obsahovat jen náhodná zatáčení a například čísla pro odečítání a sčítání, kdy žáci musí k aktuálnímu číslu přidat novou operaci předtím, než Ozobot dojede na další křižovatku s dalším číslem.

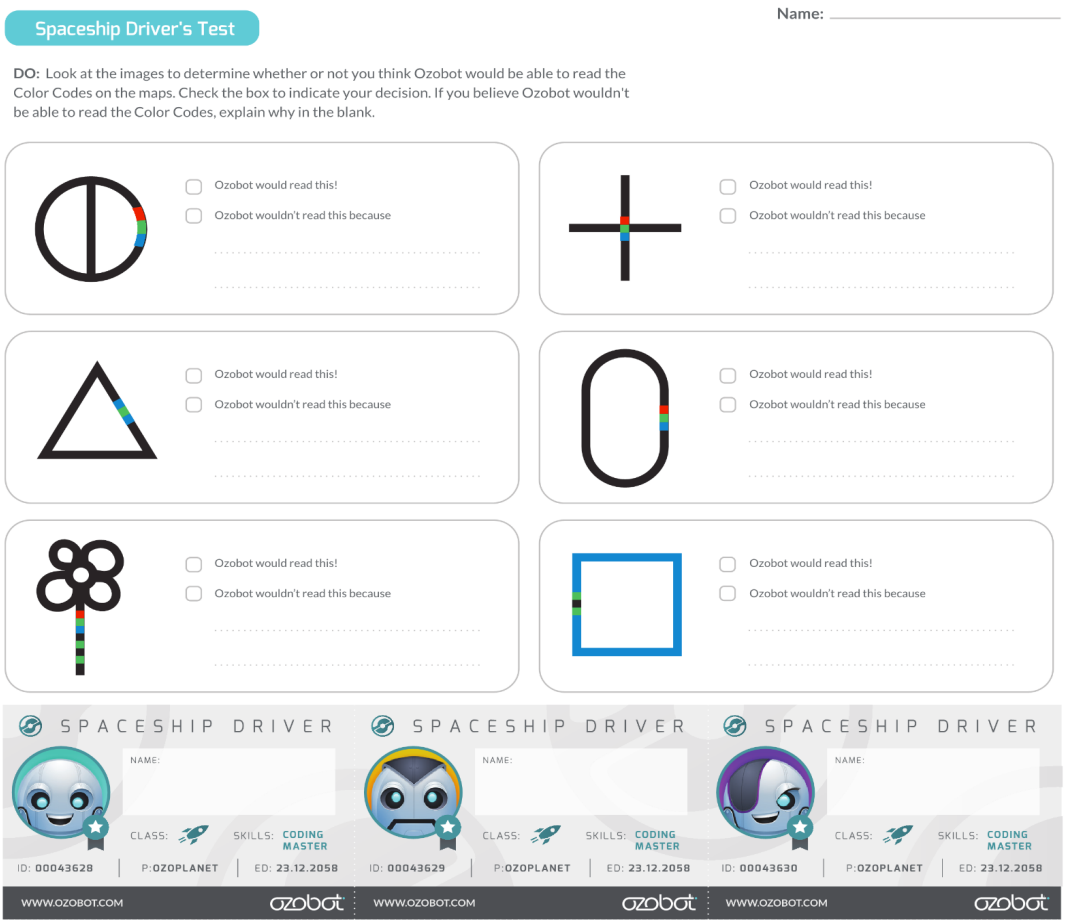
8.) Téměř všechny problémy (ve smyslu úlohy k vyřešení) jdou v programování vypracovat více než jedním způsobem (zpravidla velkým množstvím různých způsobů). Ne všechna řešení jsou ale stejně dobrá. Zde se nabízí možnost porovnat efektivitu dvou či více různých řešení jednoduchým spuštěním robotů ve stejnou chvíli. Ozobot, který dojede do cíle jako první, má pravděpodobně lepší řešení. v takovém případě následuje analýza žáků, na jejímž základě by měli dojít k důvodu, v čem je jedno řešení lepší než druhé.



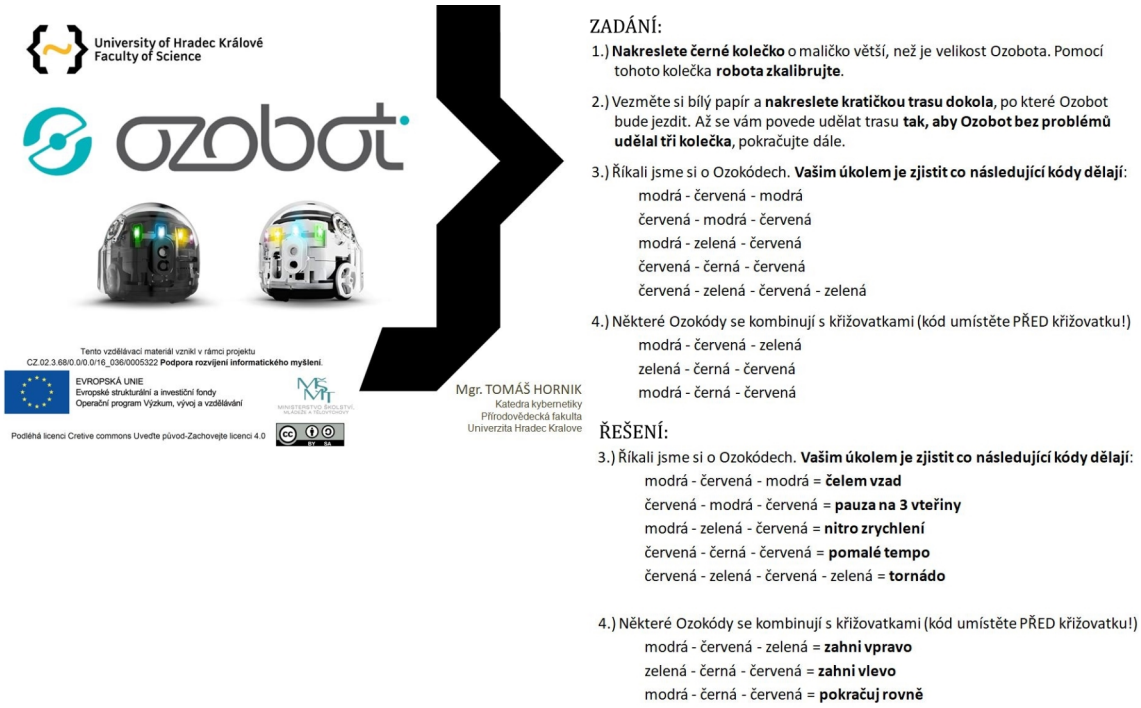
Obrázek 59: ***Podklad 3*** *- Konstrukce Ozobotů Bit a EVO z oficiální příručky*



Obrázek 60: ***Podklad 4*** *- České instrukce pro kreslení čar a Ozokódů*

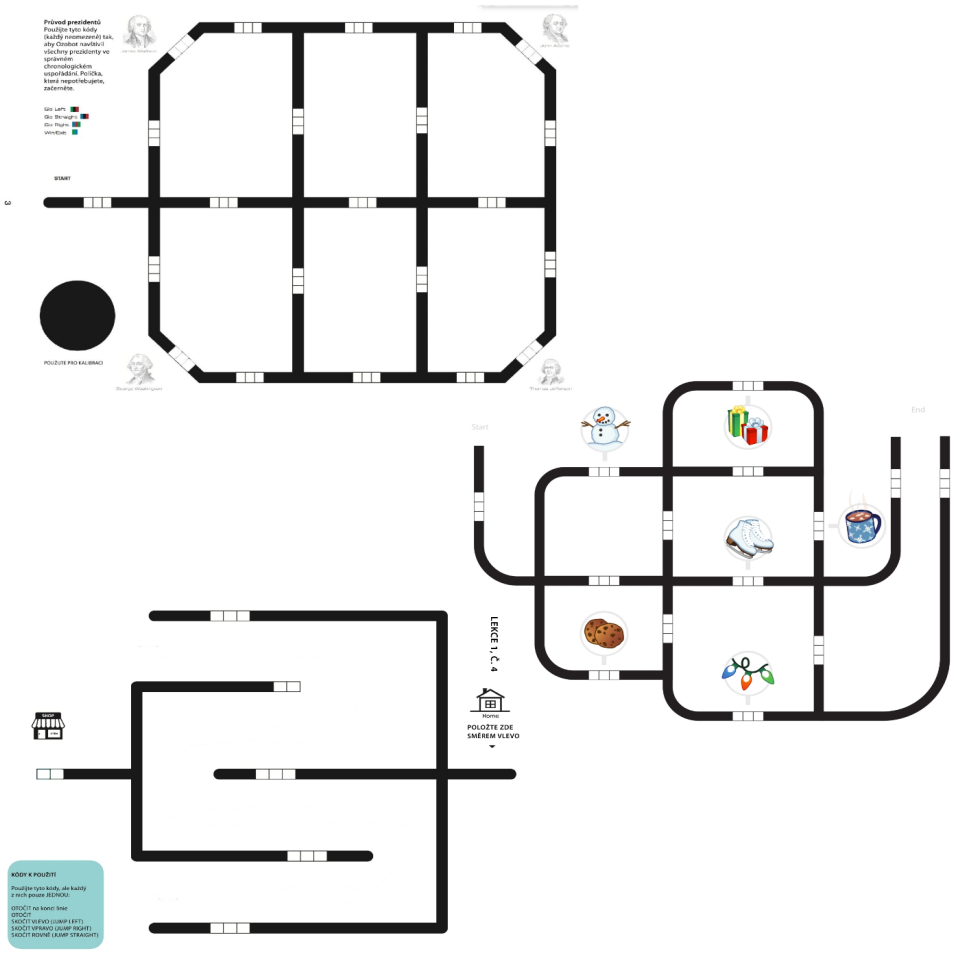


Obrázek 61: ***Podklady 5 a 6*** *- Mini-test a Ozořidičáky*

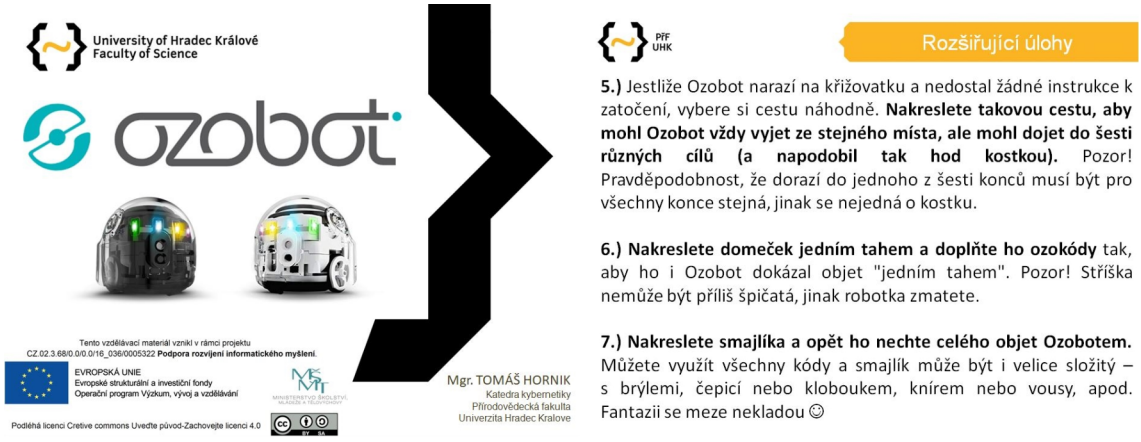


Obrázek 62: ***Podklad 8*** *- Zadání a řešení první sady úloh pro žáky*

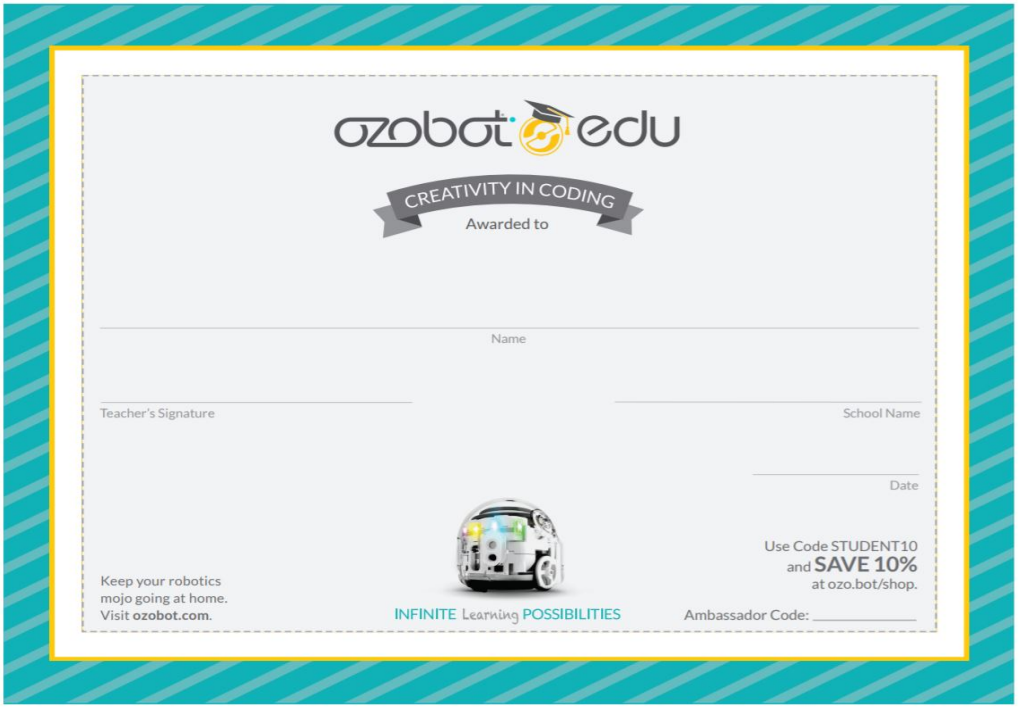
Obrázek 63: ***Podklad 9*** *- Karta všech barevných Ozokódů (v AJ)*



Obrázek 64: ***Podklad 10*** *- Tři úlohy ve stylu bludiště*



Obrázek 65: ***Podklad 11*** *- Rozšiřující úlohy pro rychlé žáky*



Obrázek 66: ***Podklad 12*** *- Finální certifikát pro žáky o absolvování kreativní hodiny kódování*

Samostatná práce (část 5)

Cílem tohoto předmětu je připravit vás, jakožto budoucí učitele, na problematiku výuky programování a algoritmizace za pomoci různých vzdělávacích nástrojů. V této lekci je takovým nástrojem Ozobot, pro kterého je sice velké množství zdrojů, které ale velice často potřebují překlady či doladění. Následující úlohy tedy zpracujte pečlivě a své výtvory doporučujeme následně nasdílet s vašimi spolužáky a vytvořit si tak základ vlastní knihovny materiálů pro vaši praxi:

a) **Vytvořte nějakou úlohu pro Ozoboty založenou na využití Ozokódů**. Úloha může být takřka jakákoliv, ale musí mít nějakou souvislost s Vaší druhou aprobací (čili něco, co byste využili v rámci hodin dějepisu, češtiny, matematiky, apod.). Úlohu si připravte včetně instrukcí pro učitele, zadání pro žáky či studenty a nezapomeňte na její řešení. Pro kreslení trasy můžete využít vynikající šablonu http://ozobot.sandofky.cz/kreslime-vlastni-mapy/#more-576

b) Alternativní možností (či rozšiřujícím úkolem) je **vymyslet nějaký nástavec pro Ozobota** (něco jako byly výše zmiňované vidle pro vysokozdvižný vozík, ale může se jednat například o přívěs, tažný vozík, hák, manipulátor, apod.) a **plán mini města/staveniště/bludiště/atp., ale včetně šablon pro modely** (čili aby stačilo jen vytisknout a slepit).

3.4.3 Volně dostupné zdroje tisknutelných materiálů

Příprava vlastních tisknutelných materiálů je zpravidla velice časově náročná a přinejmenším zpočátku vaší učitelské praxe je s **ohledem na ohromné množství práce, kterou jako učitel budete vykonávat** (přípravy na hodiny, přípravy a opravy testů, dozory, porady, jednání s rodiči, kontroly sešitů, organizace soutěží, zájezdů a exkurzí, atd. a navíc se jako informatici zpravidla setkáte s takřka neustálými žádostmi o radu, pomoc a opravu čehokoliv technického na škole) **je čas nedostatkové zboží**. Proto se hodí mít k dispozici nějaké již připravené volně dostupné materiály, které si s vaší rostoucí praxí upravíte podle svých potřeb. V ukázkové lekci bylo takových materiálů využito hodně, odkud se tedy vzaly?

Nejzákladnějším zdrojem je přímo web výrobce, který je neustále aktualizován novými nápady a vzdělávacími materiály. Základní a největší nevýhodou je ryze anglická verze a tedy občasná nutnost úpravy materiálů (většinou ale stačí jen mít připravené vlastní vysvětlení a zadání).

Ze základních dvou záložek **Education a Playground** vás jako učitele bude zajímat zejména ta první. v záložce Playground naleznete vysvětlení základů práce s Ozokódy (a krásné demonstrační video[62]), dále odkaz na webové programovací prostředí OzoBlockly, několik her pro vytištění i pro Ozoblockly a odkazy na doprovodné aplikace; tedy věci užitečné hlavně pro samostatné domácí uživatele.

Sekce Education je obsahově bohatější a užitečnější. Hned v první záložce *Getting Started* je **připravený anglický rychlokurz pro učitele**[63], ve kterém se během zhruba 40 minut seznámíte se základními možnostmi a ovládáním Ozobota, přičemž nejužitečnější je, že text obsahuje možné otázky vašich žáků, na které se tak předem můžete připravit. Dále jsou v této sekci dostupné základní „kurzy“ pro žáky, které jsou perfektní pro menší děti, ale u starších žáků je tisk většiny těchto materiálů plýtvání papírem (viz modifikace v ukázkové hodině).

Záložka **Lesson Library** vás přenese na starší verzi webu na adrese *portal.ozobot.com*, kde naleznete téměř 150 již připravených a otestovaných podkladů do vašich hodin. Na tomto portálu vám navíc přibyla možnost přesunout se na **oficiální OzoBlog**,[64] který kromě zajímavých nápadů pro samotnou výuku s Ozobotem obsahuje také zajímavosti a aktuality ze světa týkající se tématu vzdělávání v oblasti informatiky a v neposlední řadě kategorii **Educator of the Month**. Články v této kategorii obsahují rozhovory s vybranými učiteli, kteří dle tvůrců Ozobotů kvalitně využívají jejich roboty a jsou tak výborným zdrojem nápadů a postřehů do vašich lekcí. Jedním z takových nápadů je například realizace

62 OZOBOT. (2019). How to: Color Codes. In: *YouTube* [online]. Aug 7, 2019 [cit. 2019-08-20]. Dostupné z: https://www.youtube.com/watch?v=2wgBxElvInQ&feature=youtu.be

63 Dostupné ve formátu PDF na: https://files.ozobot.com/stem-education/educator-botcamp.pdf

64 OzoBlog je dostupný na adrese: https://blog.ozobot.com/

projektu Oregon Trail,[65] který spojuje předměty dějepis, zeměpis, výtvarnou výchovu a základy algoritmizace.

Sekce Classroom je založená na celých Classroom Kitech (které jsou v ČR také dostupné), ale obsahuje v podstatě jen možnost virtuální konzultace stran zapojení Ozobota do STEAM výuky (science, technology, engineering, arts, mathematics) školeným odborníkem. **Sekce Calendar** je ale opět podstatně přínosnější – obsahuje totiž nejen kalendář nadcházejících webinářů (semináře přes internet), ale hlavně **volně dostupné videozáznamy ze všech předcházejících webinářů od roku 2016**. v podstatě se tak pro vás jedná o zlatý důl nápadů do vaší praxe.

Poslední záložka, popisující proces získání titulu certifikovaného učitele Ozobotů, se týká jen učitelů, kteří se nacházejí na území USA. Tímto jsme kompletně vyčerpali oficiální web, dalších zdrojů materiálů je ale ještě mnoho.

Nezanedbatelným zdrojem nápadů jsou i portály **YouTube a Pinterest**, ačkoliv nejčastějším jazykem je stále angličtina. Naleznete zde například nápady pro zapojení Ozobota do hodin českého jazyka a literatury pomocí vyprávění pohádek,[66] případně aktivity pro všelijaké svátky a významné dny typu jako je Den Star Wars, 4. května (z anglického May the Fourth Be With You).[67]

Přesuneme-li se na česky lokalizované či přímo vytvořené materiály, **předním českým zdrojem je web *Ozobot ve výuce*,**[68] kde naleznete podklady rozsahem a kvalitou prakticky ekvivalentní s originálními anglickými zdroji. Nachází se zde přehledné shrnutí pro první seznámení s Ozoboty, několik málo českých materiálů rozdělených podle jednotlivých předmětů (na biologii, fyziku, matematiku a výtvarnou výchovu), dále vybrané a otestované 3D tisknutelné doplňky a možnost akreditovaného dalšího vzdělávání pedagogických pracovníků.

Z konkrétních pomůcek zde znovu zmíníme **šablonu pro kreslení tras a tvorbu vlastních herních plánů a bludišť pro Ozoboty v MS Excel**,[69] kterou jsme zmiňovali v zadání první sady úloh z Ozobota. Dále je zde velké množství lokalizovaných lekcí pro Ozokódy, z nichž **velice zajímavá je lekce 3**, která se zaměřuje na základy kombinatoriky (a odpovídá na častou otázku „kolik různých kombinací barevných kódů pro Ozobota existuje?“) Tyto lekce jsou v aktualizované a okomentované verzi dostupné také jako součást bakalářské práce.[70]

---

65 Educators of the Month: Third Graders Bring Back the Oregon Trail. In: *Ozobot* [online]. April 2018 [cit. s2019-08-16]. Dostupné z: https://blog.ozobot.com/educator-of-the-month/creators-of-the-month-third-graders-bring-back-the-oregon-trail/

66 EDTECHS Education Technology Specialists. (2017). Ozobot Bit - Little Red Riding Hood Video. In: *YouTube* [online]. 5.6.2017 [cit. 2019-08-26]. Dostupné z: https://www.youtube.com/watch?v=Xd686C5-Ds0

67 MISSTECHQUEEN. STEM Ideas with Miss Tech Queen: May the 4th be with you. In: *Pinterest*. [online]. Apr 28, 2018 [cit. 2019-08-26]. Dostupné z: https://cz.pinterest.com/pin/125749014582204162/

68 ŠANDOVÁ, Hanka. (2019). Ozobot ve výuce: robůtci s českým <3 [online]. [cit. 2019-08-26]. Dostupné z: http://ozobot.sandofky.cz/

69 Šablona ke stažení dostupná na http://ozobot.sandofky.cz/kreslime-vlastni-mapy/#more-576

3.5 Online prostředí Ozoblockly a mobilní aplikace

Tvůrci projektu Ozobot se snaží zpřístupnit programování pomocí Ozoblockly co nejširší veřejnosti a tak vytvořili i **celkem čtyři způsoby nahrávání programů** do vašeho robota (v případě pokročilejšího EVO). Základní způsob je poskládat váš program na webu Ozoblockly, zkalibrovat robota na bílém pozadí dle instrukcí a nahrát program **pomocí barevného blikání vyznačeného místa na monitoru**. Takto nahraný kód se vám uloží do vnitřní paměti Ozobota a **spouští se dvojím rychlým zmáčknutím zapínacího tlačítka** (tlačítko je multifunkční - jedno zmáčknutí je vypnout/zapnout a dlouhé podržení spouští kalibraci). Tento postup jako jediný funguje i s levnějším Ozobot Bit. Nevýhodou je, že blikání na projektoru i monitoru je velice nepříjemné, a ačkoliv prevalence fotosensitivní epilepsie je jen 1/4000,[71] riziko tu je. Druhou nevýhodou je, že komplexní programy se nahrávají i déle jak minutu a jestliže se nahrání nepovede, je nutné celý proces zopakovat.

Druhá možnost je **kombinace webového prostředí na počítači a mobilního telefonu**. Ta je také již podmíněna nutností registrace na webu Ozoblockly. Celkem dvanáct programů, které po přihlášení ve webovém prostředí vytvoříte si totiž můžete ukládat přímo na svůj účet a po přihlášení do aplikace Evo by Ozobot se vám v aplikaci objeví možnost vámi vybraný kód spustit. i velmi dlouhé kódy se spouští během vteřiny či dvou a práce tak není přerušovaná opakovaným zdlouhavým nahráváním, které žáci během testování kódu opakují mnohokrát. Ukládání do účtu vám navíc dává možnost mít rozpracovaných či hotových více kódu najednou. Nevýhodou je poněkud nadbytečný krok v podobě spouštění kódu z telefonu (je tedy nutné, aby ke každému Ozobotovi byl počítač i telefon).

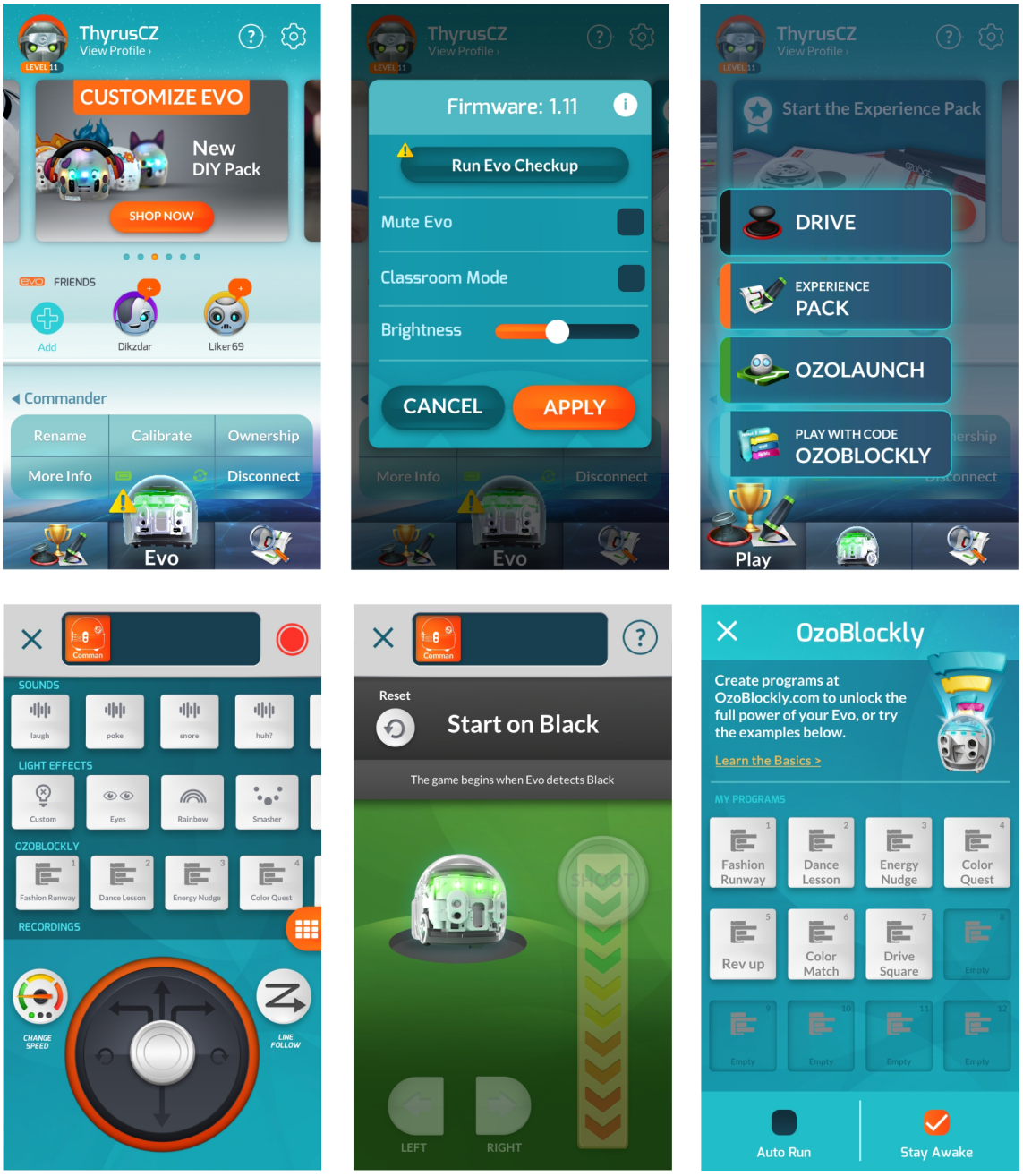
Třetí variantou je využít možnosti **Ozoblockly Editor, který byl nedávno zabudován přímo jako součást aplikace Evo by Ozobot** od verze 2.2.268 publikované v prosinci 2018.[72] Je zde ale háček – tlačítko pro spuštění Ozoblockly Editor je viditelné jen na tabletech a na mobilu se tato možnost vůbec nezobrazí.

Poslední, čtvrtou, variantou je **využít webové prostředí přímo ve webovém prohlížeči na telefonu a mezi prohlížečem a aplikací si přepínat**. Jelikož je Ozoblockly bez problémů použitelné i na mobilním telefonu (testováno na telefonu Samsung Galaxy Xcover 4 SM-390F s Androidem verze 9 v prohlížeči Google Chrome ve verzi 76.0.3809.111), jeví se absence přístupu do Ozoblockly Editoru přímo v aplikaci přinejmenším zvláštně.

70 YAGHOBOVÁ, Anna. (2019). *Zapojení Ozobotů do výuky algoritmizace a informatického myšlení* [online]. Praha, 2019 [cit. 2019-08-26]. Dostupné z: http://ozobot.sandofky.cz/nove-ulohy-pro-ozoboty-v-cestine/. Bakalářská práce. Univerzita Karlova. Vedoucí práce Mgr. Lenka Forstová.

71 OKUDAN, Zeynep Vildan a Özkara ÇIĞDEM. (2018). Reflex epilepsy: triggers and management strategies. *Neuropsychiatr Dis Treat* [online]. Vol. 14, 2018 Jan 18. 327–337. DOI: s10.2147/NDT.S107669. Dostupné z: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5779309/

72 Evo Updates. *Ozobot* [online]. December 24, 2018 [cit. 2019-08-20]. Dostupné z: https://blog.ozobot.com/ozobot/evo-updates/



Obrázek 67: Screenshoty ze starší verze Evo by Ozobot při zobrazení na mobilním zařízení

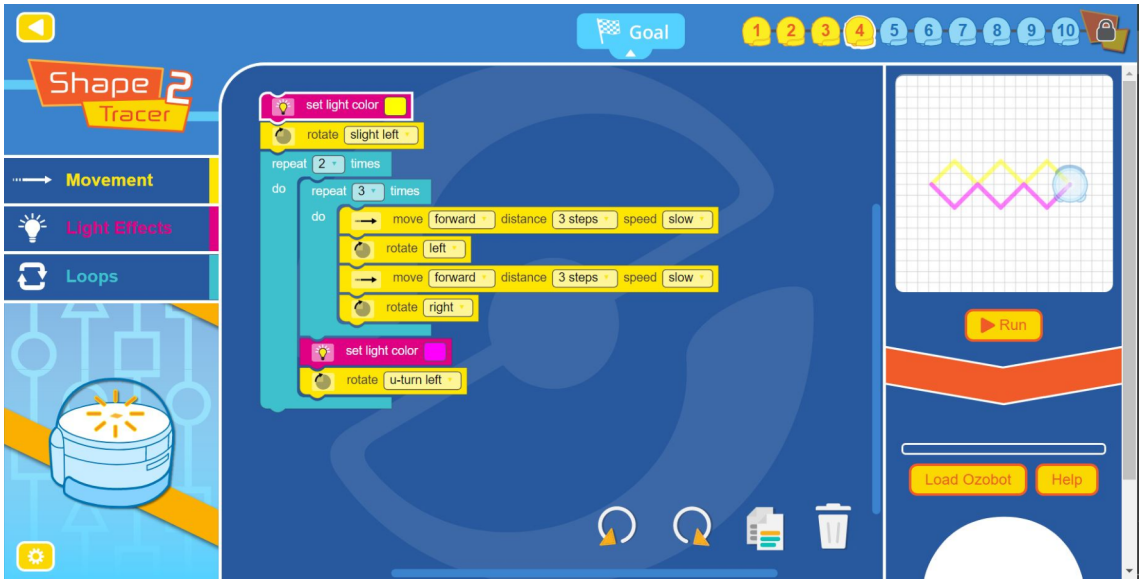
V úvodu kapitoly 3.3 byly zmiňovány základní možnosti práce s aplikací Evo by Ozobot. Na obrázku 67 výše je zobrazeno několik vybraných sekcí z této aplikace. Vlevo nahoře je základní obrazovka, kterou vidíte vždy jako první po zapnutí aplikace. Jestliže uprostřed dole u připojeného Ozobota (proces připojení je popisován krok za krokem) vidíte žlutý vykřičník, je dostupná nová aktualizace firmwaru a měli byste ji provést (a rovnou snížit hlasitost Ozobota a snížit jas LED). Ozoboty si můžete pojmenovat i v aplikaci, což usnadňuje orientaci v případě jejich velkého počtu. Zapnutí **Classroom mode** sníží množství zvuků vydávaných Ozoboty, což může pomoci při práci s většími skupinami (žvatlání robotů je ale samo o sobě motivujícím faktorem, takže se více osvědčilo jen snížení hlasitosti).

Ikonky v dolních rozích otevírají dodatečná menu, přičemž nás zajímá především levé dolní submenu (které je vidět na obrázku 67 vpravo nahoře). v tomto menu nás zajímají možnosti *Drive, OzoLaunch a Ozoblockly* (v dolní části obrázku). Možnost ***Drive* přepne robota do přímého ovládání** ve stylu autíčka na dálkové ovládání, čehož se dá využít pro jemné ovládání například v kombinaci se závodní dráhou. Robot reaguje ostře a navzdory své velikosti je schopen vyvinout značnou rychlost (přesné změření je možné jako fyzikální experiment se žáky).

***OzoLaunch* je jednoduchá hra**, ve které se na zemi rozloží barevná kolečka, program vygeneruje na jakou barvu má Ozobot ze svého místa dojet a cílem je dostat Ozobota do cíle s co nejmenším počtem pokusů. Ovládání Ozobota ale probíhá ve stylu například Angry Birds, kdy si nejprve nastavíme směr, jakým má Ozobot jet, a poté si pomocí posuvníku nastavíme počáteční rychlost, která se následně snižuje jako u autíčka na setrvačník. Poslední možnost je výše popisované spouštění programů připravených ve webovém prostředí OzoBlockly. v případě práce s tablety by pod touto možností bylo ještě tlačítko *OzoBlockly Editor*.

3.5.1 Tutoriály OzoBlockly Games a doprovodné hry

Ačkoliv můžete rovnou využít prostředí Ozoblockly, tvůrci Ozobotů vytvořili ještě **doprovodný portál Ozoblockly Games**.[73] Na tomto portálu se v současné době nacházejí **dva hodinové tutoriály Shape Tracer 1 a 2** (a dva složitější ještě bohužel nejsou dostupné a již dva roky jsou označeny jako „Coming soon“) Principiálně se jedná o podobný koncept jako v Hour of Code. v tomto případě žáci Ozobotem napodobují barevně vyznačené trasy a v případě prvního tutoriálu si osvojí základy skládání instrukcí, které ve druhém tutoriálu rozšíří o práci s cykly. Tutoriály žáky sice baví, ale jsou velice pomalé a tak vhodné spíše pro mladší žáky.



Obrázek 68: Prostředí hry Shape Tracer 2 z OzoBlockly Games (vyřešená 4 úloha)

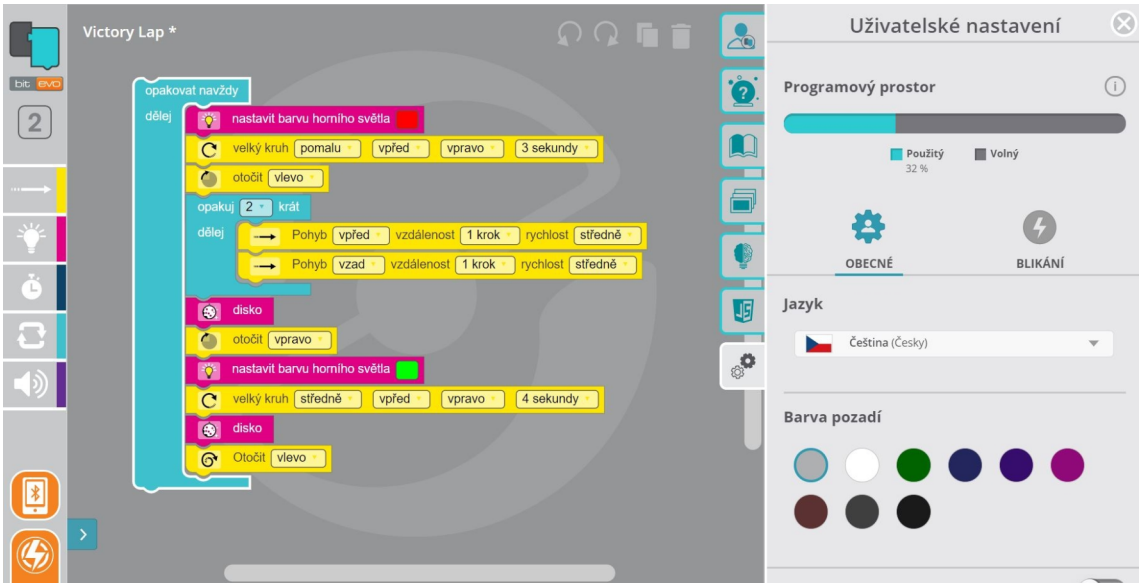
73 *OzoBlockly Games: Code to play* [online]. Evollve, ©2018. Dostupné z: https://games.ozoblockly.com/

Pro učitele je důležitá odměna, kterou je **přístup ke hře určené pro vytištění**, která se otevře vždy po dokončení daného tutoriálu (tyto hry jsou tedy celkem dvě, Shape Tracer Basic[74] a Shape Tracer Advanced[75]). Obě hry jsou opět na stejném principu, kdy si žáci z nabídky různých a různobarevných tras vylosují nějakou (či několik), tuto trasu naprogramují a nahrají do robota. Ostatní žáci poté v týmech hádají podle spuštěného programu o kterou trasu se jedná. Hra je vždy velice dobře přijatá od nejmenších žáků až po vysokoškolské studenty.

3.5.2 Ovládání OzoBlockly a vybrané výzvy

Setkali-li se žáci se Scratchem, Hour of Code nebo OzoBlockly Games, prostředí OzoBlockly pro ně zpravidla nečiní žádné problémy. Nejmarkantnějším rozdílem je absence jakékoliv herní scény, protože kód realizuje samotný Ozobot. Zatímco OzoBlockly Games jsou použitelné i bez robotů, zde jsou roboti podmínkou.

Na screenshotu na obrázku 69 je zapnutý příkladový program a rozkliknutá pravá spodní záložka *Nastavení*. Zde si **lze celé prostředí přepnout do českého jazyka**, který trpí stejnými nedostatky, jako všechny ostatní rapidně se rozvíjející projekty zmiňované v první kapitole. Místy se tedy setkáte s angličtinou namíchanou s českým jazykem, ale tyto kombinace zpravidla v práci nepřekáží. Paměť Ozobota je navíc limitovaná a příliš velký program do něj není možné nahrát, proto je také v této záložce grafické a procentuální znázornění velikosti programu, která nikdy nesmí překročit 100%.



Obrázek 69: Příkladová úloha v prostředí OzoBlockly přepnutém do češtiny (vpravo otevřená záložka Nastavení)

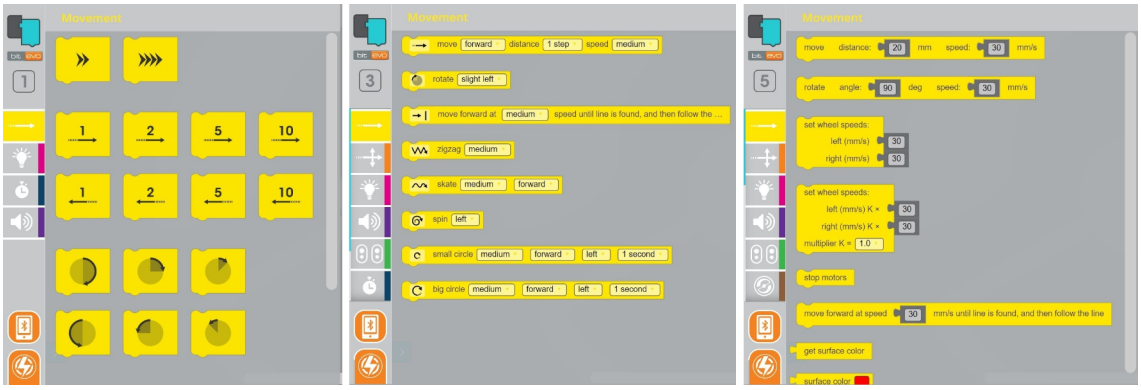
Z hlediska dalšího nutného nastavení je potřeba vlevo nahoře vybrat správnou verzi OzoBota, protože lepší verze EVO má výrazně více příkazů, které pracují se senzory a zvuky. Tyto možnosti Bit úplně postrádá a prostředí to reflektuje.

74 Jednodušší verze bez cyklů, dostupná na: https://games.ozoblockly.com/shapetracer/basic/game.pdf

75 Složitější verze s cykly, dostupná na: https://games.ozoblockly.com/shapetracer/advanced/game.pdf

Posledním nastavením před samotným započetím práce je **výběr úrovně složitosti** programovacího prostředí. Těchto **úrovní nabízí OzoBlockly celkem pět**, přičemž každá úroveň zvyšuje komplexnost a množství instrukcí a otevírá nové možnosti mimo jiné i přidáním základních algoritmických konstrukcí.

Nejnižší úroveň operuje jen s obrázky a čísly a umožňuje jen základní ovládání robota. Při použití této úrovně je možné Ozobota pohodlně využít i s menšími žáky již od první třídy (a určité úrovně jde využít už na mateřské škole). Na obrázku 70 níže vidíte progresivní zvyšování složitosti instrukcí v kategorii pohybu robota a zároveň vidíte zvyšující se množství kategorií v levém sloupečku. Jednotlivé úrovně přibývaly postupně a poslední pátá byla přidána na konci května 2018.



Obrázek 70: Přepínání verze a úrovně prostředí, porovnání instrukcí pro pohyb z úrovní 1, 3 a 5

Na první úrovni mají žáci možnost využít jen elementární obrázkové instrukce pro pohyb, rozsvěcení LED světýlek, čekání a základní zvuky. **Na druhé úrovni** přibudou smyčky v podobě nekonečného cyklu a cyklu s pevným počtem opakování. Největší změnou je ale posun od obrázkových instrukcí k textovému popisku jednotlivých grafických bloků.

Třetí úroveň je ideální začátek pro práci s žáky na druhém stupni. v této úrovni již jsou možnosti ovládání v uvedených kategoriích téměř bez omezení a navíc přibyly rozhodovací podmínky IF, vnímání pomocí senzorů vzdálenosti i barvy a možnosti zapojení práce s čárou. v takovém případě již lze kombinovat kreslené mapy a projekty s vlastními programy. **Čtvrtá úroveň** přidává možnosti ukončení programu, kompletně přepracovává kategorii podmínek, přidává možnost tvorby proměnné a s nimi spojené matematické operace. Posledním velkým rozšířením této úrovně je velice užitečná tvorba vlastních funkcí. **Nejvyšší úrovní je pátá „master“ úroveň**, která umožňuje využívat spouštěcí tlačítko Ozobota a přidává seznamy a jedno-rozměrové pole. Seznamy a pole jsou již velice komplexní konstrukce, které žákům činí ty největší problémy. Spolu s oznámením na OzoBlogu o vypuštění finální verze[76] proto tvůrci Ozobota rovnou dali k dispozici výukové materiály od profesora Richard Borna zaměřené přímo na výuku práce s poli za využití nových možností Ozobotů.[77]

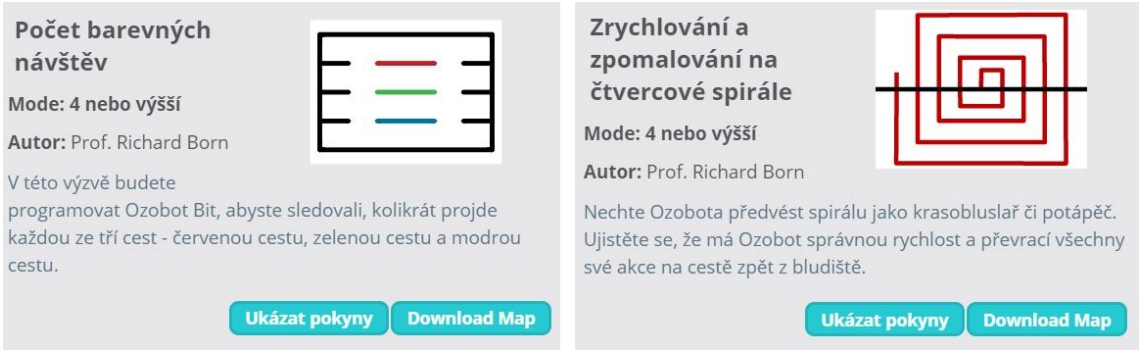
76 What's New in OzoBlockly Mode 5? In: *Ozobot* [online]. 2018-05-28 [cit. 2019-07-08]. Dostupné z: blog.ozobot.com/ozobot/whats-new-ozoblockly-mode-5/

V pravém postranním menu se nahoře nachází možnost přihlášení, po kterém lze programy ukládat na svůj účet a odtud odesílat do aplikace Evo by Ozobot. Bez příhlášení lze programy ukládat i do svého lokálního profilu na daném počítači. v obou případech je limitním počtem dvanáct programů. Po kliknutí na *Uložit jako* máte ale ještě **možnost si své programy stáhnout do počítače** a mít tak připravený archiv programů, které můžete využít pro ukázky nebo jako řešení úloh.

Následující dvě položky jsou pomocné – jedná se o sekce Nápověda a Slovníček. ***Nápověda* je dobře zpracovaná** a v případě nutnosti vysvětluje krok za krokem celý proces. ***Slovníček* je v podstatě dokumentace**. Ve slovníčku naleznete jednotlivé algoritmické konstrukce (bloky) a jejich vysvětlení. Většina těchto textů ve slovníčku je přeložená do češtiny a tato sekce je výborný úvod do problematiky práce s dokumentací. Některé bloky a jejich rozdíly tak např. není třeba vysvětlovat, ale naopak odkázat třídu na slovníček a nechat vysvětlení na žácích.

Přínosem pro učitele je následující **záložka *Příklady***. Zde se totiž nacházejí konkrétní zadání vybraných úloh vhodných pro danou úroveň, přičemž tyto úlohy se mění právě podle nastavené úrovně prostředí. Tyto úlohy mají krátké zadání (ale jednoznačné) a obsahují tlačítko, které na pracovní plochu rovnou vyskládá řešení. Takovou úlohu ale samozřejmě můžete s žáky vypracovat samostatně a od samého začátku tak máte k dispozici předem připravenou sadu úloh pro každou úroveň.

Nejkomplexnější úlohy jsou v **záložce *Výzvy*, jejíž obsah se podle nastavené úrovně NEmění**. Nacházejí se zde celkem čtyři úlohy, z nichž první dvě jsou jednoduššího charakteru a jsou určené pro druhou a třetí úroveň prostředí. Druhé dvě úlohy jsou komplexnější a jejich zadání naleznete na screenshotu na obrázku 71. Řešení těchto dvou úloh je popsáno a vysvětleno dále.



Obrázek 71: Zadání Challenge 3 a 4

Pořadově první (i když složitější) je úloha řešící počet průjezdů přes barevné čáry. k této úloze je nutné mít vytištěný podklad v podobě barevné mapy, která je jako miniatura viditelná na obrázku 71 výše. Pokyny (a případná mapa) jsou k dispozici ke stažení u všech výzev, ale bez ohledu na nastavený jazyk jen anglicky.

77 BORN, Richard. OzoBlockly Array Primer. In: *Ozobot* [online]. Ozobot & Evollve, ©2018, May 22, 2018 [cit. 2019-08-28]. Dostupné z: https://portal.ozobot.com/lessons/detail/array-primer?utm_source=blog&utm_medium=text&utm_campaign=blog&utm_term=brand&utm_content=mode5

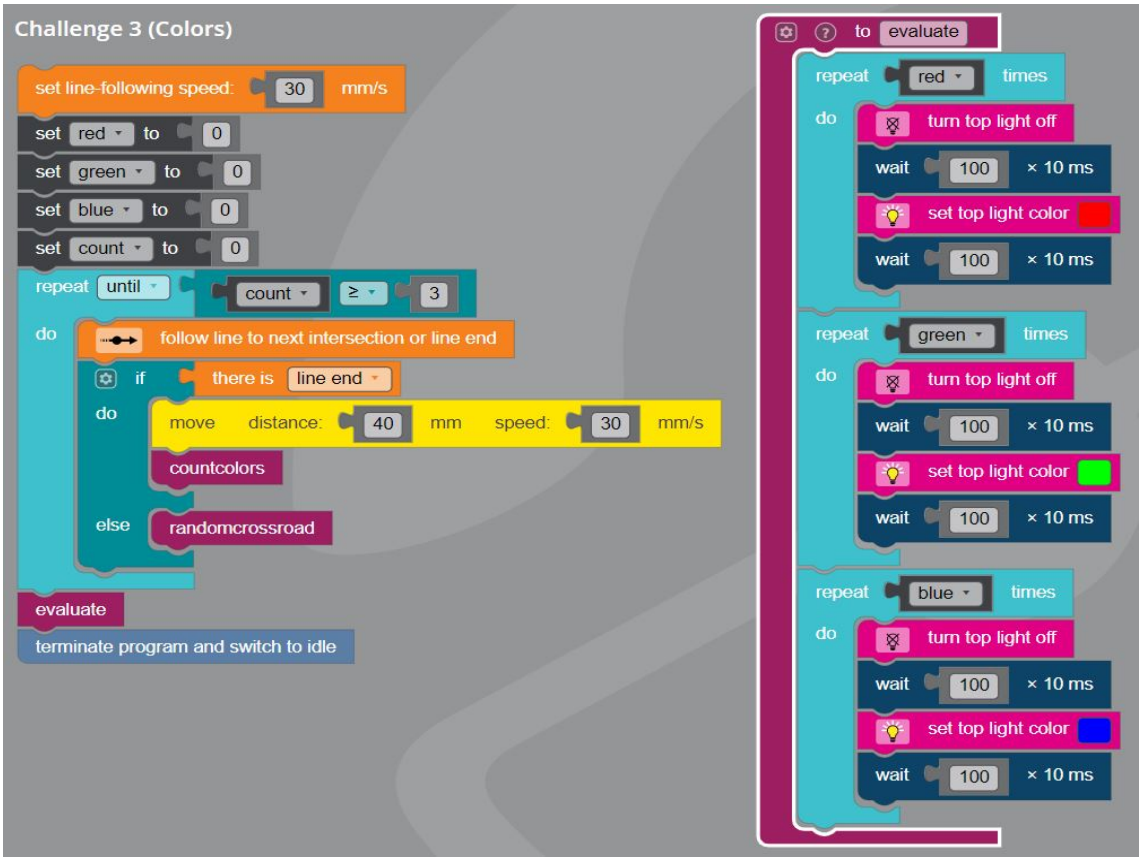
V úloze *„Počet barevných návštěv“* má Ozobot jezdit po plánku, na křižovatkách si vybírat náhodný směr, po černé čáře má svítit bíle a na barevných úsecích odpovídající barvou. Program má být zakončený po celkem dvanácti přejetí barev (doporučujeme snížit na tři až pět – dvanáct trvá velmi dlouho), načež má zablikat červeně tolikrát, kolikrát přejel červenou čáru, a poté stejně tak pro zelenou a modrou barvu. Nakonec se má Ozobot vypnout.

Ačkoliv je originální zadání v jasně definovaných bodech, jedná se o rozsáhlejší úlohu obsahující několik samostatných problémů k řešení. U žáků na druhém stupni ZŠ je velice **častým problémem míchaní více problémů dohromady a snaha vše vyřešit najednou**. Tento problém lze adresovat vypracováním postupu práce v bodech, ještě před započetím samotného programování. Kompartmentalizaci dílčích problémů během tvorby kódu ve velké míře napomáhá **vytváření vlastních funkcí**.

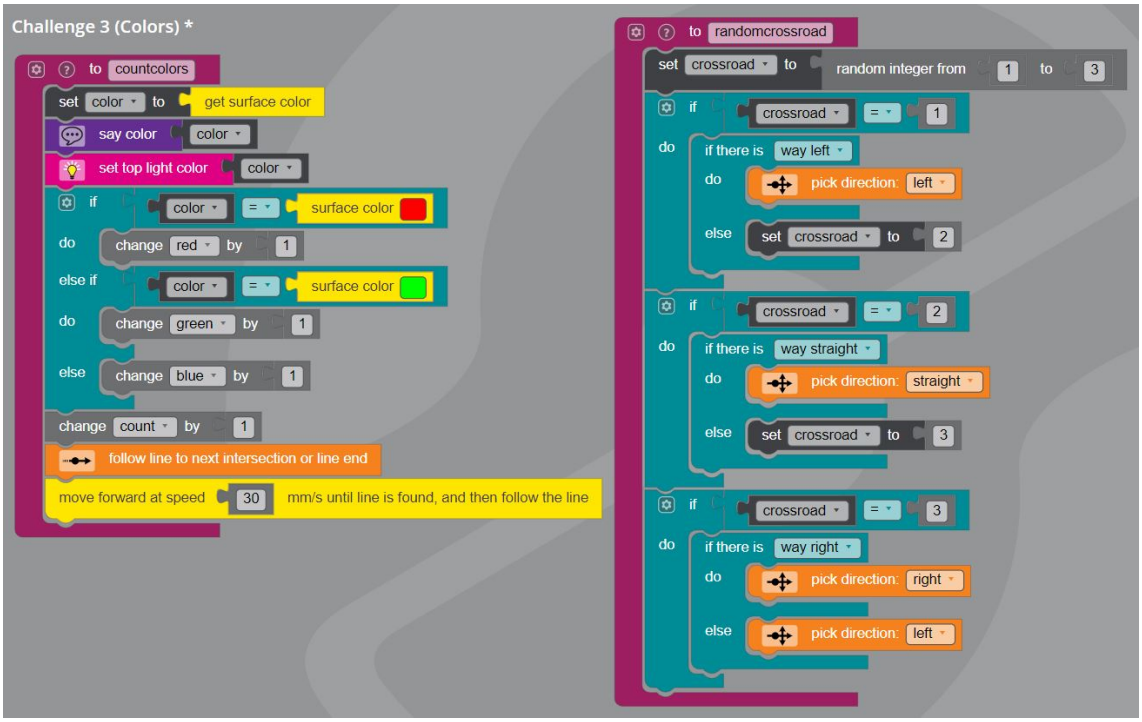
Prvním krokem při řešení složitějších úloh by vždy měla být analýza zadání a promyšlení dílčích kroků k řešení. Tento, či obdobný, postup by si měli žáci zafixovat a využívat i při plně samostatné práci. Výsledná příprava v bodech pak může vypadat následovně (přičemž první dva kroky jsou prakticky vždy stejné):

1. *Vytvořit potřebné proměnné*
2. *Připravit výchozí stav (nastavit defaultní hodnoty proměnných)*
3. *Zformulovat základní běh programu (tj. jeď do křižovatky nebo konce čáry a podle toho zareaguj)*
4. *Připravit podmínky pro ukončení programu (opakuj základ dokud…)*
5. *Vytvořit novou funkci pro náhodné zatáčení na křižovatkách*
6. *Vytvořit novou funkci pro úseky s barevnými čárami*
    - *6a. Jak má robot přejet přerušenou oblast bez čáry?*
    - *6b. Co má robot dělat když je na barevné čáře?*
    - *6c. Jak se má zase napojit zpátky na černou čáru?*
7. *Vyhodnotit výsledné barvy a ukončit program*

Je výhodné takovýto postup alespoň zpočátku tvořit skupinově a vést žáky k chybějícím krokům. Výše popsaný popis jednotlivých kroků řešení je tak jen jedna možnost z mnoha a v každé skupině se tak bude více či méně lišit. Na základě těchto kroků byl následně vytvořen program zachycený na obrázcích 72 a 73.



Obrázek 72: Řešení Challenge 3 (část 1 - základní program a závěrečné vyhodnocení)



Obrázek 73: Řešení Challenge 3 (část 2 - počítání barev a náhodné křižovatky)

Mezi problémy, které mohou nastat spadá **časté zapomínání aktualizace defaultního nastavení** v případě přidání nových proměnných či jiných změn. Druhým nejčastějším problémem je **určení, jaké části programu by měly mít vlastní funkci**. Poslední komplikace souvisí s **přeformulováním vlastních nápadů do podoby kódu**. Zde je nejplatnější rada nechat žáky nahlas co nejjednoduššími slovy popsat, co přesně má robot dělat. Pomáhá žáky navodit otázkami, jako např. *„A jak dlouho má ten robot jezdit?“ -> „No dokud nepřejede tři/pět/dvanáct barev.“* (čili opakuj dokud počet přejetých barev je roven tři/pět/dvanáct)

V závislosti na časové dotaci je možné práci rozdělit do skupinek. Nejslabší skupince nechat vypracovat finální blikání (na obrázku 72 je zobrazeno jako funkce *evaluate* neboli *vyhodnoť*) a silnějším skupinkám uložit náhodné zatáčení a přejíždění barev. v každém případě se ale hodí využívat vlastní funkce, které nejenže velice usnadní takovéto rozdělení do skupinek, ale zaměří pozornost žáků jen na jejich část kódu. Žáci mívají tendenci se do programů zamotat a opravovat chyby v sekcích programu, které s danou chybou nemají žádnou spojitost, čímž si program „rozsypou“ úplně.

Druhá výzva (v prostředí OzoBlockly čtvrtá a zároveň poslední) je paradoxně jednodušší než počítání barev a hodí se na **ukázku práce s parametry funkcí**. Součástí zadání úlohy je také **vysvětlení fyzikálního principu zákona zachování momentu hybnosti**, který má úloha velmi hrubě ilustrovat. Tento zákon říká, že je moment hybnosti soustavy stálý, když je výsledný moment vnějších sil roven nule. v běžném životě tento efekt vidíme například na krasobruslařích, jejichž rychlost rotace v piruetě se zvýší jestliže přitáhnou ruce blíže k vertikální ose otáčení.

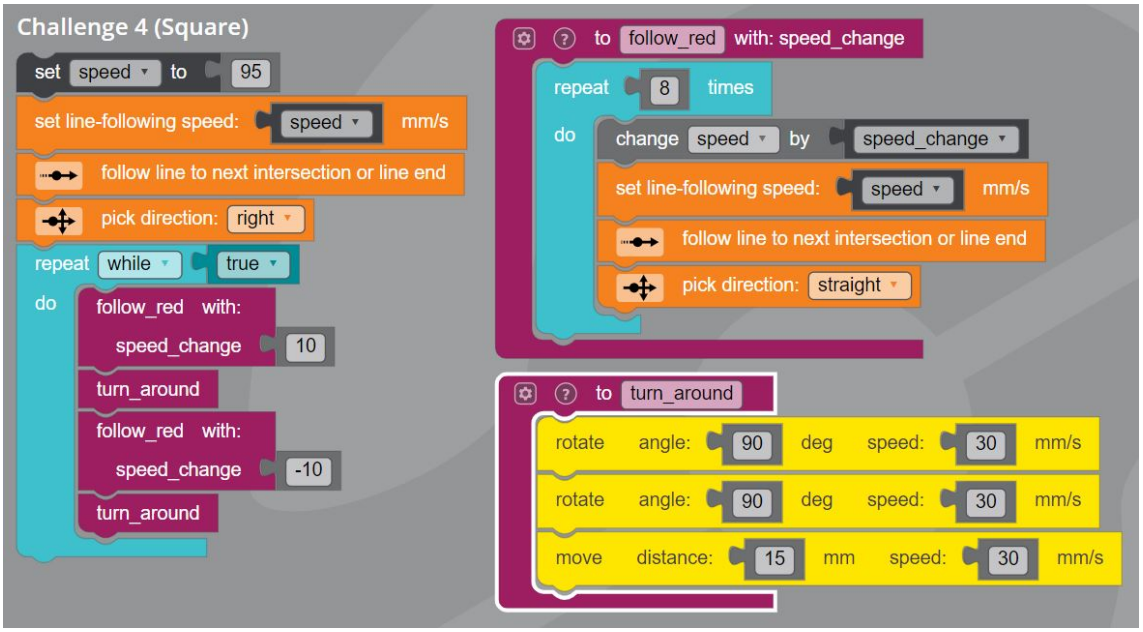
V úloze *„Zrychlování a zpomalování na čtvercové spirále“* má Ozobot jezdit po červené spirále (v náhledu na obrázku 71 vpravo nahoře), přičemž při každém přejetí černé čáry má Ozobot uzpůsobit rychlost od počátečních 15 mm/s až na maximum 85 mm/s a při cestě dovnitř tak zrychlovat a při cestě ven zpomalovat. Tento pohyb má Ozobot opakovat donekonečna.

I v této úloze je tak nutné kombinovat OzoBlockly s tištěným podkladem a je nutné dát pozor, aby nebyl podklad vytištěný příliš malý. v takovém případě totiž Ozobot navzdory své mrštnosti na vnitřním konci dráhy nestíhá reagovat na zatáčky a přejíždí na vnější čáry.

Prvním krokem je opět vypracování postupu řešení

1. *Vytvořit potřebné proměnné*
2. *Připravit výchozí stav (nastavit proměnné a nájezd na čáru)*
3. *Jet po červené čáře a při přejetí černé čáry změnit rychlost*
4. *Na konci čáry se otočit čelem vzad*

V této úloze není potřeba řešit zakončení programu a lze tedy jednoduše využít nekonečný cyklus (který je od úrovně čtyři pomocí bloku ***repeat while true***). Dílčích kroků také není mnoho a žáci, kteří již absolvovali výuku se Scratchem by s touto úlohou neměli mít problém. Zpravidla však úlohu řeší jednou dlouhou procedurou, kde se kód pro cestu dovnitř a pro cestu ven opakuje dvakrát, jen s rozdílnými znaménky u změny rychlosti. Tento stav je ideální pro ukázku práce s parametry funkce, které můžete v OzoBlockly přidat pomocí ozubeného kolečka vlevo nahoře u každé funkce. v tomto případě se nám hodí proměnná *zmena_rychlosti* (na obrázku anglicky *speed_change*), které bude při cestě dovnitř předána hodnota +10 a při cestě ven -10.



Obrázek 74: Řešení s využitím parametru funkce

Zdánlivě jednoduchý robot za využití veškerého svého potenciálu zvládne překvapivé množství různých úloh, od jednoduchého ježdění až po komplexní výpočty v prostředí OzoBlockly. Potenciál pro mezipředmětové vztahy je při jeho použití takřka bezbřehý a záleží jen na fantazii vaší a vašich žáků. Nezapomeňte, že také můžete všelijaké nástavce a oblečky kombinovat i s programy tvořenými v OzoBlockly a tímto způsobem značně rozšířit už tak rozsáhlé možnosti tohoto programovacího prostředí.

Závěrečné shrnutí kapitoly 3

Po přečtení této kapitoly byste měli:

- ✔ vyjmenovat různé robotické hračky vhodné pro vaši výuku;
- ✔ nalézt zdroje pro výuku za pomoci robotů a Ozobotů;
- ✔ porovnat možnosti LEGO Mindstorms a Ozobotů;
- ✔ znát rozdíly mezi jednotlivými verzemi Ozobota;
- ✔ popsat z čeho se Ozobot skládá a jaké možnosti vám konkrétní senzory dávají pro zapojení do vaší výuky;
- ✔ vědět, jakými způsoby lze rozšířit možnosti Ozobotů;
- ✔ ovládat Ozoboty všemi možnými způsoby;
- ✔ připravit vlastní úlohy a vzdělávací materiály pro výuku jak za pomoci Ozokódů tak s využitím OzoBlockly;
- ✔ a hlavně – využít masivní možnosti Ozobota z hlediska mezipředmětových vztahů (jazyky, fyzika, dějepis,...)

Samostatná práce (část 6)

V tuto chvíli byste měli být zběhlí ve všech oblastech práce s robotickou vzdělávací pomůckou jménem Ozobot. Pro tyto úlohy platí stejné doporučení, jako pro ty minulé – dejte si na jejich zpracování záležet a své pečlivě připravené výtvory si se spolužáky nezapomeňte vyměnit. Každý materiál se vám může (a bude) hodit:

a) **Vytvořte vlastní úlohu pro Ozoboty tentokrát s využitím prostředí Ozoblockly**. Úloha nemusí mít žádnou souvislost s Vaší druhou aprobací (ale může) a může se jednat ideálně o nějaký hlavolam (například ve stylu námi zkoušené Challenge 3). Úloha může obsahovat i případný hrací plán (ale logicky nemusí, jestliže pro její vyřešení není plán potřeba). Může se jednat i o nějakou složitější úlohu, kterou jsme dělali v kapitole Scratch, ale přepracovanou a upravenou pro Ozoboty. Nezapomeňte vždy zadání i řešení.

b) **Vytvořte vlastní hru pro Ozobota s využitím prostředí Ozoblockly**. Může se jednat o něco ve stylu hry, kterou jsme hráli po online tutoriálu za pomoci zalaminovaných hracích karet, ale může se jednat klidně o hru závodní, postřehovou, hru využívající zvuky, atp. Hru připravte včetně všech potřebných materiálů, pravidel, bodování, určení vítěze, atp.

4 Tradiční programovací jazyk KAREL

Proč se původem americký jazyk KAREL jmenuje Karel a ne Charles?

Jaká je podstata práce v programovacím jazyce Karel?

Co to jsou kopenogramy a kdo je vymyslel?

Jaké jsou rozdíly mezi prostředími Karel Oldium a KarelWin?

V jakých úlohách lze využít značky a zdi na dvorku? Uveďte příklad.

Jakým způsobem funguje rekurze a v jakých úlohách ji lze využít?

4.1 Historie jazyka a práce v online implementaci

S myšlenkou použít k výuce algoritmického myšlení a nejjednodušších základů programování světa robota, přišel v roce 1981 profesor **Richard E. Pattis** ve své knize *Karel the Robot: a gentle introduction to the art of programming*. (česky *Robot Karel: Jemný úvod do umění programovat.*) Robot byl americkým profesorem informatiky pojmenován Karel a ne Charles **na počest českého spisovatele Karla Čapka**, autora slavné divadelní hry RUR, ve které bylo slovo robot (jako novotvar vzniklý z českého slova robota, tj. těžká fyzická práce) poprvé použito.

Programovat robota Karla znamená ovládnout jeho mikrosvět. Karel se může **pohybovat na jakési šachovnici, čtvercové hrací desce s čtvercovými poli**. Deska, nazývaná také **dvorek či město**, má typický rozměr 10 x 10 polí, ale může být i větší (záleží na konkrétní implementaci). Na začátku je Karel „novorozeně“, které rozumí jen několika slovům. Umí **pouze čtyři základní pohyby**:

- ✔ krok směrem, do kterého je otočen (příkaz KROK),
- ✔ otočení o 90° proti směru hod. ručiček (příkaz VLEVO-VBOK),
- ✔ položit značku na pole, na kterém stojí (příkaz POLOŽ),
- ✔ zvednout značku z pole, na kterém stojí (příkaz ZVEDNI).

Neumí tedy např. udělat krok vzad, úkrok stranou, otočit se o 90° ve směru hodinových ručiček, nebo otočit se o 180°. Toto **všechno může udělat pomocí opakování a kombinování základních čtyř příkazů**, nebo, což je pochopitelně lepší, ho můžeme naučit vykonávat nové příkazy.

Kromě základních pohybů má Karel také smysly. Jsou velmi jednoduché, takže pomocí nich je schopen určit pouze:

- ✔ zda před ním **JE ZEĎ, nebo NENÍ ZEĎ**; zeď může být před Karlem ze dvou důvodů. **Buď stojí na kraji hrací desky** (někdy se jí také říká dvorek, což lépe vysvětluje představu zdi okolo desky) a je natočen tak, že dalším krokem by se ocitl mimo desku, **nebo jsme zeď Karlovi umístili na hrací desku**, a to na políčko, na které by Karel vstoupil dalším krokem;
- ✔ zda pod ním, tedy na políčku, na kterém právě stojí, **JE ZNAČKA, nebo NENÍ ZNAČKA**; není však schopen spočítat, kolik značek je zde umístěno;
- ✔ do třetice má Karel v sobě zabudován kompas, který **je schopen rozlišit, do které ze čtyř základních světových stran je natočen** (sever je nahoru, jih dolů, západ vlevo a východ vpravo); vyhodnotí tedy pravdivost výroků: JE SEVER, NENÍ SEVER, JE JIH, NENÍ JIH, JE ZÁPAD, NENÍ ZÁPAD, JE VÝCHOD, nebo NENÍ VÝCHOD;
- ✔ posledním smyslem je schopnost poznat, zda políčko, na kterém Karel stojí je jeho výchozí pole, jedno z polí na šachovnici (na dvorku, ve městě) můžeme označit jako domov; Karel tedy rozliší, zda pole, na kterém právě stojí, JE DOMOV, nebo NENÍ DOMOV.

Třetí oblastí, kterou Karel ovládá, jsou **řídící struktury programů**. Konkrétně podmíněný příkaz, čili větvení programu a několik různých příkazů cyklu. Tyto řídicí struktury si vysvětlíme na příkladech v dalších podkapitolách.

Příkazy Karlova jazyka jsme uváděli v češtině. Společným rysem všech dětských programovacích jazyků a jejich odlišností od běžných programovacích jazyků je lokalizace do mateřského jazyka dítěte (žáka základní školy), které chceme naučit algoritmickému myšlení a základům programování. **Lokalizace usnadní žákům pochopení** příslušného programovacího jazyka, **ale na druhou stranu vede k odlišnostem v překladu** mezi jednotlivými implementacemi.

4.1.1 Práce v online prostředí Karel OLDIUM

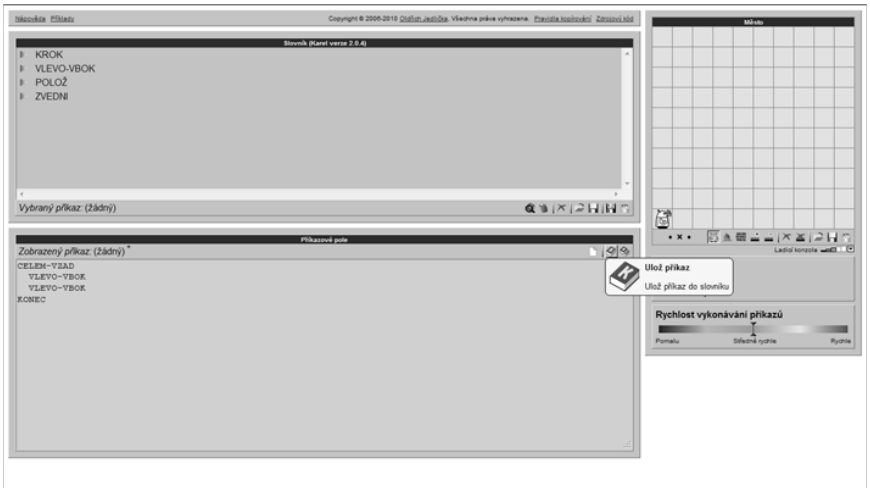
V této kapitole si ukážeme jak snadné je naučit Karla další prvky pořadové přípravy (vojenský dril není nijak intelektově náročný). Začněme nejprve tím, že **naučíme Karla dva nové obraty**, a to o 180°, tj. čelem vzad, a o 90° ve směru hodinových ručiček, tj. vpravo vbok. Vyzkoušíme si práci s Karlem a přidání těchto jednoduchých příkazů do jeho slovníku v interaktivním webovém prostředí bez nutnosti instalace. Vývojové prostředí robota Karla naprogramované Oldřichem Jedličkou jako skript na straně klienta vytvořený v jazyce JavaScript najdete na adrese http://karel.oldium.net/.

Program začíná vždy názvem nového příkazu a končí slovem KONEC. Mezi ně zapisujeme další příkazy v pořadí, v jakém se mají provádět, tedy například:

```
ČELEM VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

```
VPRAVO-VBOK
    ČELEM-VZAD
    VLEVO-VBOK
KONEC
```

Karla můžeme naučit, aby se choval rozumně, např. aby nenarážel do zdi, ale ve chvíli, kdy ji před sebou vidí, raději zatočil. k tomu nám slouží **podmíněný příkaz KDYŽ *podmínka* ‖ *příkaz(y) 1* ‖ KONEC, JINAK ‖ *příkaz(y) 2* ‖ KONEC** (symbol dvojité svislé čáry ‖ v tomto textu používáme pro oddělení částí příkazu, které musí začínat na novém řádku) . Podobně můžeme zařídit, aby se Karel nesnažil zvednout značku, když pod ním žádná značka není. v tomto případě použijeme **jednodušší verzi podmíněného příkazu KDYŽ *podmínka* ‖ *příkaz(y)* ‖ KONEC**.



Obrázek 75: Webové prostředí robota Karla

```
CHYTRÝ-KROK
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        VLEVO-VBOK
    KONEC
KONEC
```

```
CHYTŘE-ZVEDNI
    KDYŽ JE ZNAČKA
        ZVEDNI
    KONEC
KONEC
```

Ještě rozumnější chování by bylo, kdybychom Karlovi řekli: „Jdi ke zdi!“ a on by šel tak dlouho, dokud by ke zdi nedošel. Jindy bychom chtěli, aby se Karel, když už má v sobě zabudovaný kompas, natočil na určitou světovou stranu, třeba na sever.

```
JDI KE ZDI
    DOKUD NENÍ ZEĎ
        KROK
    KONEC
KONEC
```

```
NA SEVER
    DOKUD NENÍ SEVER
        VLEVO-VBOK
    KONEC
KONEC
```

K takovým úkolům nám poslouží příkaz **cyklu s podmínkou**. Provádí se tak dlouho, dokud je splněna podmínka. Pokud není podmínka splněna hned na začátku, příkazy uvnitř cyklu se neprovedou vůbec. Podmínka může být uvedena také na konci cyklu. Představme si, že chceme, aby Karel, který stojí na poli označeném jako „domov" v levém dolním rohu dvorku, obešel pomocí svého „chytrého kroku", který jsme ho před chvílí naučili, celý dvorek kolem dokola a po návratu domů se opět zastavil. Kdybychom použili podmínku na začátku cyklu, Karel by se vůbec nehnul. Hned na začátku by zjistil, že je doma a už by neudělal ani jeden krok. Proto **použijeme příkaz cyklu s podmínkou na konci, který má strukturu OPAKUJ ‖ *příkaz(y)* ‖ AŽ *podmínka***. Cyklus končí, až když je podmínka splněna, ale vždy se provede aspoň jednou.

```
OBCHŮZKA
    OPAKUJ
        CHYTRÝ KROK
    AŽ JE DOMOV
KONEC
```

Posledním typem cyklu je **cyklus s předem daným počtem opakování**. Příkaz má strukturu **OPAKUJ *n*-KRÁT ‖ *příkaz(y)* ‖ KONEC**. Ukažme si jeho použití, jestliže chceme, aby Karel udělal trojkrok, nebo aby obešel čtvercovou trasu. Během provádění těchto dvou příkazů nevyhodnocuje Karel situaci ve svém okolí. Když nechceme, aby narazil do zdi, musíme mu příkaz dát jen tam, kde vidíme dost místa pro jeho vykonání.

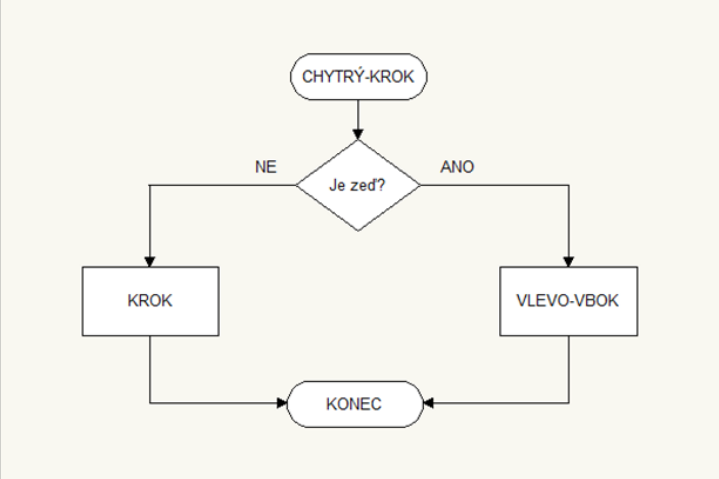
```
TROJKROK
    OPAKUJ 3-KRÁT
        KROK
    KONEC
KONEC
```

```
ČTVEREC
    OPAKUJ 4-KRÁT
        TROJKROK
        VLEVO-VBOK
    KONEC
KONEC
```

4.1.2 Grafické znázornění algoritmů a kopenogramy

Jak jsme již v úvodu stanovili, programy představují vlastně slovní zápis algoritmů v určitém formálním jazyce. Některým žákům bude bližší grafické znázornění struktury algoritmu. k tomuto účelu běžně užíváme jednak vývojové diagramy, jednak kopenogramy. Právě kopenogramy jsou spojeny s implementací jazyka Karel v českém prostředí. s kopenogramy pracuje důsledně např. metodický materiál *Programovací jazyk KAREL.*[78]

Kopenogramy rozlišují jednotlivé typy řídících struktur barevně. Záhlaví příkazu vybarvíme žlutě, výkonné příkazy růžově, podmíněný příkaz světle modře. v kopenogramu příkazu CHYTRÝ-KROK tedy vybarvíme pole CHYTRÝ-KROK žlutě, pole NENÍ ZEĎ a pole se šipkou světle modře a pole s výkonnými příkazy KROK a VLEVO-VBOK růžově. a začátek a konec cyklu světle zeleně (viz obrázek 72). Pro práci s kopenogramy je tak nutné si pořídit pastelky nebo zvýrazňovače a kopenogramy vybarvovat. **Kopenogramy jsou český vynález**, i když částečně inspirovaný Nassi-Schneidermanovými diagramy, jejich název je odvozen z prvních dvou písmen příjmení jejich autorů (Jiří Kofránek, Rudolf Pecinovský a Petr Novák).

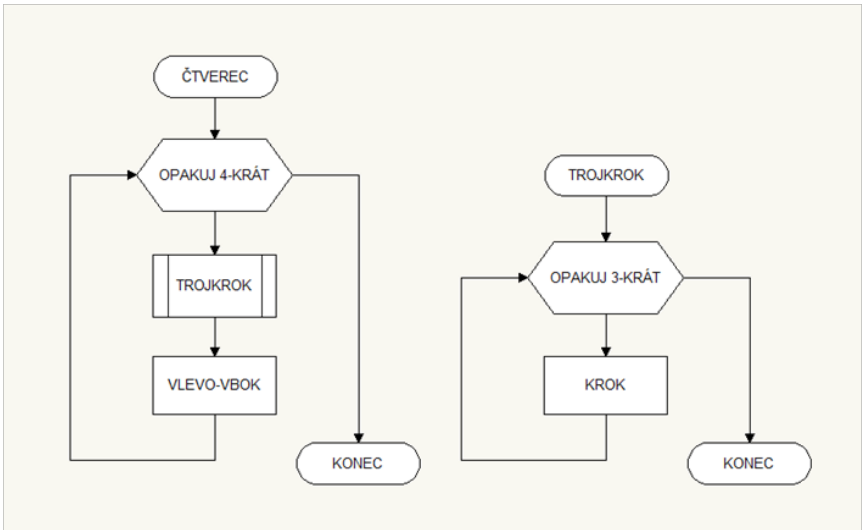


Obrázek 76: Vývojový diagram příkazu CHYTRÝ-KROK

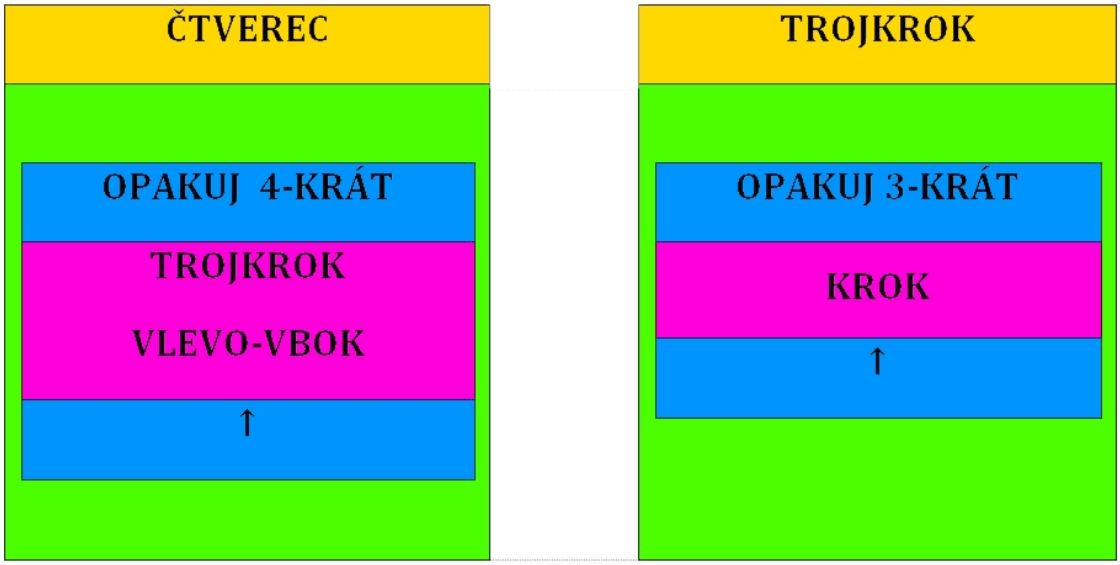
Obrázek 77: Barevný kopenogram příkazu CHYTRÝ-KROK

78 VEJVODA, M. & RYTÍŘ, M. (2001). Programovací jazyk KAREL. In: *Stanice Mladých Techniků při ZRP Větřní*, 2. vydání. Český Krumlov: 2001. 72 s. ISBN 99972-03-09-7

Kopenogramy jsou jedním z typů stukturogramů. Používáním sturkturogramů ve výuce vytváří učitel u svých žáků správné představy a správné návyky, pokud jde o strukturované programování.



Obrázek 78: Vývojový diagram příkazů ČTVEREC a TROJKROK



Obrázek 79: Barevné kopenogramy příkazů ČTVEREC a TROJKROK

4.1.3 KAREL staví ze značek schody a značí si jimi cestu

V této kapitole si pohrajeme s Karlovou **schopností pokládat a zvedat značky**. Začněme něčím jednoduchým, nechme Karla postavit schody. Protože Karlovy příkazy nemají žádné parametry, musíme pevně zvolit výšku schodu. Zvolíme výšku 2 značek (použijeme klasické značky, nikoliv kuličky). Nejprve navrhneme strukturu celé stavby a teprve pak dořešíme dílčí příkazy. Zadávat jednotlivé algoritmy musíme ovšem v opačném pořadí. Příkaz pro stavbu schodů ukazuje, že je možné v jazyce Karel psát i delší programy.

```
SCHODY
    NA-VÝCHOD
    KROK;  POLOŽ2
    KROK;  POLOŽ4
    KROK;  POLOŽ6
    KROK;  ZAPLŇ
    KROK;  ZAPLŇ
    ÚKROK-VLEVO;  POLOŽ2
    ÚKROK-VPRAVO
    KROK;  ZAPLŇ
    ÚKROK-VLEVO;  POLOŽ4
    ÚKROK-VPRAVO
    KROK;  ZAPLŇ
    ÚKROK-VLEVO;  POLOŽ6
    ÚKROK-VPRAVO
    KROK;  ZAPLŇ
    ÚKROK-VLEVO;  ZAPLŇ
    ÚKROK-VPRAVO
    KROK
KONEC
```

```
NA-VÝCHOD
    DOKUD NENÍ VÝCHOD
        VLEVO-VBOK
    KONEC
KONEC
```

```
POLOŽ2
    POLOŽ;  POLOŽ
KONEC
```

```
POLOŽ4
    POLOŽ2;  POLOŽ2
KONEC
```

```
POLOŽ6
    POLOŽ4;  POLOŽ2
KONEC
```

```
ZAPLŇ
    POLOŽ4;  POLOŽ4
KONEC
```

```
ÚKROK-VLEVO
    VLEVO-VBOK
    KROK
    VPRAVO-VBOK
KONEC
```

Značky může do města pokládat nejen Karel, ale **můžeme mu je do jeho města předem umístit a tím mu například vyznačit cestu**, po které se má pohybovat. Při pohybu po značkách se snaží Karel jít vždy nejprve rovně, když to nejde, zkusí zahnout doleva a když ani to nejde, pak zahnout doprava. Poslední možnost je obrat o 180° a cesta zpátky odkud přišel. Spouštěcím příkazem je PO-ZNAČKÁCH.

```
NA-ZNAČKU
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        VLEVO-VBOK
        NA-ZNAČKU2
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        VLEVO-VBOK
        NA-ZNAČKU2
    KONEC
KONEC
```

```
NA-ZNAČKU3
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        VPRAVO-VBOK
        KROK
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        VPRAVO-VBOK
        KROK
    KONEC
KONEC
```

```
NA-ZNAČKU2
    KDYŽ NENÍ ZEĎ
        KROK
    KONEC, JINAK
        ČELEM-VZAD
        NA-ZNAČKU3
    KONEC
    KDYŽ NENÍ ZNAČKA
        ZPĚT
        ČELEM-VZAD
        NA-ZNAČKU3
    KONEC
KONEC
```

```
ČELEM-VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

```
VPRAVO-VBOK
    ČELEM-VZAD
    VLEVO-VBOK
KONEC
```

```
PO-ZNAČKÁCH
    OPAKUJ 100-KRÁT
        NA-ZNAČKU
    KONEC
KONEC
```

4.1.4 Seznamujeme KARLA s rekurzí

Karla jsme naučili spoustu zajímavých příkazů, ale dosud jsme mu neprozradili jedno velké překvapení. Spočívá v možnosti **použít uvnitř definice příkazu, kterou teprve vytváříme (ale ještě jsme ji neukončili), tento nově vytvářený příkaz**, jako kdyby jej už Karel znal. Takové **volání sama sebe** nazýváme rekurze a je to velmi silný nástroj v řadě programovacích jazyků.

Skutečnost, že jazyk robota Karla umožňuje rekurzi, nám nejen pomůže zapsat jednodušším způsobem některé dříve definované příkazy, ale také zapsat zcela nové příkazy, které by bez rekurze vůbec nebylo možné realizovat. Například při stavbě schodů už nemusíme předem určovat počet položených značek, stačí dát Karlovi chytře najevo, aby na každé další pole dal vždy o jednu značku víc.

Karel neumí rozpoznat kolik značek má pod sebou, přitom ale pokus, položit více než osm značek na jedno pole, skončí chybným hlášením. Proto je tento program potřeba spouštět příkazem SCHODY-N, který vložením nadbytečného kroku zaručí nepřekročení maximálního počtu položených značek.

```
SCHODY-N
    KROK
    SCHODY-R
    ZPĚT
KONEC
```

```
ČELEM-VZAD
    VLEVO-VBOK
    VLEVO-VBOK
KONEC
```

```
SCHODY-R
    KDYŽ NENÍ ZEĎ
        KDYŽ JE ZNAČKA
            ZVEDNI
            KROK
            POLOŽ
            ZPĚT
            SCHODY-R
            POLOŽ
```

```
ZPĚT
    ČELEM-VZAD
    KROK
    ČELEM-VZAD
KONEC
```

```
        KONEC, JINAK
            KROK
            POLOŽ
            SCHODY-R
            ZPĚT
        KONEC
    KONEC
KONEC
```

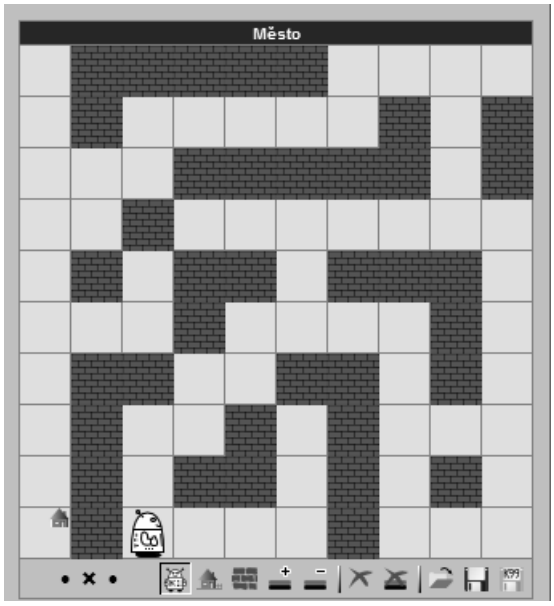
Příkladem příkazu, který by bez rekurze vůbec nešel naprogramovat, je příkaz JDI-DO-PŮLKY, který zajistí, že Karel dojde do poloviny vzdálenosti mezi svoji okamžitou pozicí a zdí.

```
    JDI-DO-PŮLKY
        KDYŽ JE ZEĎ
            ČELEM-VZAD
        KONEC, JINAK
            KROK
            KDYŽ NENÍ ZEĎ
                KROK
            KONEC
            JDI-DO-PŮLKY
            KROK
        KONEC
    KONEC
```

```
    ČELEM-VZAD
        VLEVO-VBOK
        VLEVO-VBOK
    KONEC
```

Pomocí rekurze můžeme robota Karla naučit hledat cestu z labyrintu. Postavíme labyrint ze zdí a umístíme Karla daleko od políčka označeného DOMOV a dáme mu pokyn, aby se vydal na cestu. Před každým krokem Karel položí značku a tím si vyznačí, na kterém políčku už byl. Používání značek zaručí, že Karel nakonec najde cestu domů.

```
LABYRINT
    KDYŽ NENÍ DOMOV
        KDYŽ NENÍ ZEĎ
            POLOŽ;  KROK
            KDYŽ NENÍ ZNAČKA
                LABYRINT
            KONEC, JINAK
                ZVEDNI;  ZPĚT
                ZVEDNI
                VLEVO-VBOK
                LABYRINT
            KONEC
        KONEC, JINAK
            VLEVO-VBOK
            LABYRINT
        KONEC
    KONEC
KONEC
```



Obrázek 80: KAREL bloudí labyrintem

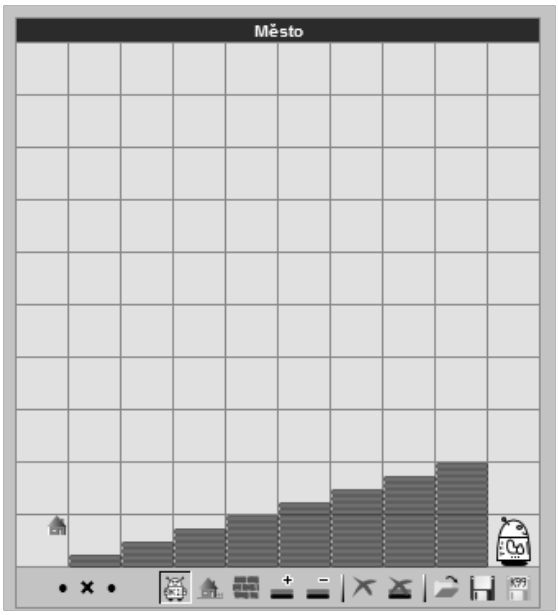
Poslední ukázkou rekurze, kdy jedna z nově definovaných procedur volá sama sebe, bude soubor příkazů pro vytvoření Pascalovo trojúhelníka ze značek. Jsme omezeni maximálním počtem značek na jednom poli a také velikostí Karlova dvorku. Přesto je příkaz zajímavý tím, že Karla musíme naučit sčítat počty značek v určených dvou polích a výsledek zobrazit ve třetím poli.

```
PASCAL
    NA-SEVER;  KE-ZDI
    VPRAVO-VBOK
    OPAKUJ 4-KRÁT
        KROK
    KONEC
    POLOŽ;  KE-ZDI;  CR-LF
    OPAKUJ 4-KRÁT
        PASCAL-RADEK
        CR-LF
    KONEC
    VLEVO-VBOK
    KROK;  POLOŽ
    ČELEM-VZAD;  KE-ZDI
KONEC
```

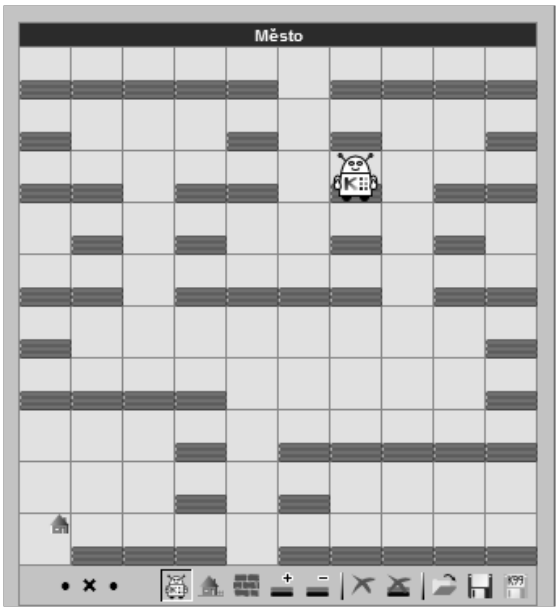
```
PRESUN
    L
    KDYŽ JE ZNAČKA
        ZVEDNI
        VLEVO-VBOK;  L
        POLOŽ
        VLEVO-VBOK
        PRESUN
        VLEVO-VBOK;  L
        POLOŽ
        VLEVO-VBOK;  L
    KONEC, JINAK
        VLEVO-VBOK;  L
    KONEC
KONEC
```

PASCAL-RADEK KROK OPAKUJ 8-KRÁT PRESUN ČELEM-VZAD PRESUN KROK KONEC KONEC	L KROK; VLEVO-VBOK KROK KONEC
	CR-LF VPRAVO-VBOK; KROK VPRAVO-VBOK; KE-ZDI ČELEM-VZAD KONEC

Pozn.: v tabulce příkazů již znovu neopakujeme dříve odladěné příkazy ČELEMVZAD, VPRAVO-VBOK a KE-ZDI.



Obrázek 81: KAREL staví schody (ilustrace k úloze e)



Obrázek 82: KAREL chodí po značkách (úloha f)

Samostatná práce (část 7)

a) **Znázorněte pomocí vývojového diagramu** algoritmus „chytrého úkroku“, při kterém se Karel nejprve otočí vlevo, podívá se, zda před ním není zeď a pokud ne, tak udělá krok a otočí se vpravo. Kdyby ovšem před sebou viděl zeď, otočí se o 180° a podívá se, zda je zeď směrem vpravo od jeho původního směru. Pokud ne, udělá krok a otočí se vlevo, pokud ano, otočí se na místě vlevo a zůstane stát.

b) **Znázorněte pomocí kopenogramu** algoritmus popsaný ve cvičení *a*.

c) **Naprogramujte a odlaďte příkaz CHYTRÝ-ÚKROK**, který realizuje postup popsaný ve cvičení *a*.

d) Navrhněte algoritmus, jehož realizací Karel **systematicky projde všechna pole svého dvorku a sesbírá všechny značky** na něm rozmístěné. Postup můžete popsat slovně, nebo znázornit buď vývojovými diagramy, nebo kopenogramy. Vyberte si jednu z možností.

e) Algoritmus navržený ve cvičení *d* **realizujte jako příkaz UKLIĎDVOREK**, podpříkazy potřebné k realizaci celého postupu volte podle potřeby a dle svého návrhu ve cvičení *d*.

f) **Naprogramujte a odlaďte příkaz PYRAMIDA**, pomocí kterého Karel postaví stupňovitou pyramidu. Můžete si ji představit také jako **osově souměrné schody**, které nejprve stoupají a pak zase klesají.

g) **Navrhněte algoritmus, ve kterém bude Karel chodit po značkách, ale** na rozdíl od postupu popsaného v této kapitole **bude značky zvedat** a značení tak během své cesty rušit. Ve chvíli, kdy dojde na konec trasy a nebude mít kolem sebe již žádné pole se značkou, se zastaví. Postup můžete popsat slovně, nebo znázornit buď vývojovými diagramy, nebo kopenogramy. Vyberte si jednu z možností.

h) Algoritmus navržený ve cvičení *f* **realizujte jako příkaz PO-STOPĚ**, podpříkazy volte podle potřeby a dle svého návrhu ve cvičení *d*.

4.2 Práce v offline prostředí KarelWin

Práce s Karlem ve webovém rozhraní plně vyhovuje pro úvod do algoritmizace a programování. Jeho výhodou je jednoduchost a dostupnost. Co se žáci naučí ve škole, mohou si hned doma zopakovat bez nutnosti cokoliv instalovat do svého domácího počítače. Prostředí i obrázek robota jsou navíc velmi zdařilé z hlediska grafického a svou jednoduchostí příliš nerozptylují a neodvádějí pozornost od tvorby instrukcí. **Pokud se však rozhodneme pro instalaci nějakého klasického spustitelného programu, můžeme získat některá zajímavá rozšíření**.

Tak tomu je např. v **implementaci KarelWin autora Petra Laštovičky**, která je dostupná na webové stránce http://petr.lastovicka.sweb.cz/ostatni.html#karel. v tomto případě se dokonce nejedná o klasickou instalaci, ale pouze o prosté překopírování souborů, z nichž nejdůležitější je spustitelný soubor (aplikace) KarelWin.exe. **Tento typ programů je anglicky označován jako Portable** a pro uživatele má širokou řadu výhod. s portable programy můžete například pracovat i v rámci kroužku, kde byste nemuseli mít administrátorská práva k plnohodnotné instalaci na počítač. Žákům pro domácí procvičování stačí si jen zkopírovat složku s aplikací a mohou ji využívat i doma, zatímco instalace je pro žáky zpravidla příliš komplikovaná záležitost (potažmo vám to budou tvrdit, budou-li v tomto prostředí hrozit domácí úkoly).

Jednotlivé implementace prostředí a jazyka robota Karla se také mohou navzájem lišit v některých detailech. My si rozdíly ukážeme na příkladu porovnání webové verze (autor Oldřich Jedlička) a spustitelné verze KarelWin (autor Petr Laštovička). Prvním rozdílem je definice nového příkazu. Zatímco ve webové verzi jsme uváděli pouze název příkazu, implementace KarelWin vyžaduje jedno ze tří klíčových slov:

- ✔ PŘÍKAZ = spustitelná procedura, na vyžádání se provede;
- ✔ FUNKCE = samostatně nelze spustit, ale lze ji volat z procedury, vrací číselnou hodnotu;
- ✔ PODMÍNKA … podobně jako FUNKCE, ale místo čísla vrací logickou hodnotu (pravda, nebo nepravda).

Za klíčovým slovem teprve následuje název příkazu, funkce, nebo podmínky, za kterým navíc mohou (ale nemusí) bezprostředně navazovat kulaté závorky, v nichž je uveden parametr (nebo několik parametrů) oddělený čárkami.

Další odlišnost je v základních příkazech:

- ✔ Web Karel … KROK, VLEVO-VBOK, POLOŽ, ZVEDNI;
- ✔ KarelWin … KROK, VLEVO, VPRAVO, POLOŽ, ZVEDNI, MALUJ ‘*znak*‘, DOMŮ.

Pokud bychom chtěli používat příkazy dříve odladěné ve webovém prostředí, není v tom velký problém, musíme však na začátek každé definice přidat slovo PŘÍKAZ, dodefinovat příkaz VLEVO_VBOK jako alias základního příkazu VLEVO a ve všech příkazech s pomlčkou použít místo pomlčky (-) podtržítko (_).

Nový příkaz MALUJ '*znak*' napíše na pole pod Karlem znak uvedený mezi apostrofy. Je také možné psát MALUJ *ascii*, kde *ascii* je číslený kód znaku podle ASCII (American Standard Code for Information Interchange). Příkaz DOMŮ umístí Karla do levého dolního rohu města a otočí jej směrem na východ. Kromě písmen a podtržítka můžeme v názvech příkazů používat znaky ? ! : ` @ # a číslice. Název však nesmí začínat číslicí! Příklad definic příkazů:

```
PŘÍKAZ KROKY(n)
    OPAKUJ n
        KDYŽ NENÍ ZEĎ
            KROK
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ VLEVO_VBOK
    VLEVO
KONEC
```

```
PŘÍKAZ KE_ZDI
    DOKUD NENÍ ZEĎ
        KROK
    KONEC
KONEC
```

Podmínky v obou implementacích musí předcházet buď klíčové slovo JE, nebo klíčové slovo NENÍ. Nabídka aplikace pro OS Windows je opět o něco širší:

- ✔ Web Karel … SEVER, JIH, VÝCHOD, ZÁPAD, ZEĎ, ZNAČKA,
- ✔ KarelWin … SEVER, JIH, VÝCHOD, ZÁPAD, ZEĎ, ZNAČKA, MÍSTO, ZNAK, ZNAK='*znak*'

Podmínka MÍSTO rozhodne, zda je na daném poli ještě místo k položení značky, podmínka ZNAK, zda je na poli napsán nějaký znak, a podmínka ZNAK='*znak*', zda je zde zapsán konkrétní znak, uvedený mezi apostrofy. **Další podmínky je možné definovat** pomocí programů označených klíčovým slovem PODMÍNKA.

Zajímavým rozšířením jsou příkazy pro řízení běhu programu RYCHLE, POMALU a ČEKEJ *ms* (kde *ms* je počet milisekund, po který má Karel čekat), resp. **ČEKEJ -1 (čekání na stisk klávesy)**. Pravděpodobně ještě zajímavější je **možnost pracovat s proměnnými (i když pouze s celočíselnými)** a využívat vestavěné a uživatelem definované funkce. Novou proměnnou deklarujeme příkazem ČÍSLO *p*, kde *p* je proměnná, ale ještě vhodnější je přiřadit proměnné zároveň nějakou hodnotu, příkaz pak má tvar ČÍSLO *p = číselný výraz*. s čísly pak můžeme provádět běžné početní operace +, –, *, / a % (% znamená zbytek po celočíselném dělení).

```
PŘÍKAZ ABECEDA
    DOMŮ
    ČÍSLO I=0
    OPAKUJ 26
        i = I+1
        KROK
        MALUJ 64 + I
    KONEC
    KROK
KONEC
```

```
PŘÍKAZ NAPIŠ
    DOMŮ
    ČÍSLO p = 32
    DOKUD NENÍ p = 13
        MALUJ p
        KROK
        ČEKEJ -1
        p = KLÁVESA
    KONEC
KONEC
```

Všimněte si, že abeceda s 26 znaky se nám do Karlova města pohodlně vešla. Rozměry plochy, po níž se pohybuje KarelWin, jsou totiž implicitně 40 x 20 polí. Šikovným využitím proměnných a funkcí **dokážeme zadávat Karlovi stiskem kláves znaky, které má vypsat**. Funkce KLÁVESA totiž vrací ASCII kód znaku. Dalšími novými funkcemi, kromě funkce KLÁVESA, jsou funkce NÁHODA(*n*), která vrací náhodné přirozené číslo z množiny {0, 1, 2, …, n-2, n-1}, a funkce ZNAK, která vrací hodnotu znaku, který se právě nachází pod Karlem.

```
PŘÍKAZ CHAOS
    OPAKUJ 300
        OPAKUJ NÁHODA(3)
            VLEVO
        KONEC
        OPAKUJ NÁHODA(20)
            KDYŽ NENÍ ZEĎ
                KROK
            JINAK
                ČELEM_VZAD
            KONEC
        KONEC
        OPAKUJ NÁHODA(10)
            KDYŽ JE MÍSTO
                POLOŽ
            KONEC
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ ZNAČKY_NA_ČÍSLICE
    DOMŮ
        OPAKUJ 20
        DOKUD NENÍ ZEĎ
            PŘEVOD
            KROK
        KONEC
        KDYŽ JE ZEĎ
            PŘEVOD
            VLEVO
            KDYŽ NENÍ ZEĎ
                KROK
            KONEC
            VLEVO
            KE_ZDI
            ČELEM_VZAD
        KONEC
    KONEC
KONEC
```

```
PŘÍKAZ PŘEVOD
    ČÍSLO k=0
    DOKUD JE ZNAČKA
        ZVEDNI
        k=k+1
    KONEC
    KDYŽ k>0
        MALUJ 48+k
    KONEC
KONEC
```

Příkazy cyklů se v implementaci KarelWin téměř neliší, pouze u cyklů s daným počtem opakování se uvede pouze číslo k (nikoliv *k*–KRÁT) a místo čísla lze použít také proměnnou. v podmíněných příkazech s dvěma bloky příkazů neukončujeme první blok slovy … KONEC, JINAK …, ale pouze slovem … JINAK … **Zcela novým nástrojem pro větvení programu je příkaz ZVOLIT**. Má následující strukturu:

```
ZVOLIT výraz
    PŘÍPAD výraz, n. seznam výrazů
        blok příkazů
    PŘÍPAD výraz, n. seznam výrazů
        blok příkazů
    JINAK
        blok příkazů
KONEC
```

Priorita operátorů:

*, /, %
+, -
=, <>, >, <, >=, <=
JE, NENÍ
A
NEBO

Nové jsou také **možnosti opustit okamžitě prováděnou proceduru** a vrátit se na místo, odkud byla volána, příkazem NÁVRAT, nebo možnost okamžitě ukončit provádění všech procedur příkazem STOP. Vedle struktury příkazu zvolit jsme uvedli prioritu operátorů používaných při práci s proměnnými a s podmínkami. Tím jsou odlišnosti a rozšíření implementace KarelWin proti základní verzi programovacího jazyka robota Karla vyčerpány. Zbývá už jen říci, že programy můžeme komentovat, a to buď po dvou limítkách *//komentář*, nebo mezi dvojznaky */* komentář */*. Ukažme si nakonec, jak je lze využít k zobrazování víceciferných čísel na příkladu příkazu SEČTI(x,y), který zobrazí dvě zadaná čísla, sečte je a pak zobrazí jejich součet. Neuvádíme znovu výpisy příkazů ZPĚT a KE_ZDI.

```
PŘÍKAZ SEČTI(x,y)
    //nastavení pozice
    DOMŮ
    VLEVO
    KE_ZDI
    ZPĚT
    VPRAVO
    KE_ZDI
    //první sčítanec
    PIŠ(x)
    VPRAVO
    KROK
    VLEVO
    //druhý sčítanec
    PIŠ(y)
    VPRAVO
    KROK
    VLEVO
    //podržení
    OPAKUJ 10
        ZPĚT
        MALUJ '-'
    KONEC
    KE_ZDI
    VPRAVO
    KROK
    VLEVO
    //součet
    PIŠ(x+y)
KONEC
```

```
PŘÍKAZ PIŠ(n)
    KDYŽ n > 9
        VYPIŠ(n)
    JINAK
        ZPĚT
        MALUJ n+48
        KROK
    KONEC
KONEC
```

```
PŘÍKAZ VYPIŠ(c)
    KDYŽ c > 9
        ČÍSLO z = c % 10
        ČÍSLO p = (c-z)/10
        ZPĚT
        KDYŽ p>9
            VYPIŠ(p)
        JINAK
            ZPĚT
            VYPIŠ(p)
            KROK
        KONEC
        VYPIŠ(z)
        KROK
    JINAK
        MALUJ c+48
    KONEC
KONEC
```

Závěrečné shrnutí kapitoly 4

Po přečtení této kapitoly byste měli:

- ✔ být schopni vysvětlit význam jazyka KAREL z hlediska historie vzdělávacích programovacích jazyků;
- ✔ ovládat dvě konkrétní implementace jazyka KAREL (Karel Oldium a KarelWin);
- ✔ vysvětlit co to je rekurze a využít ji ve vhodných úlohách;
- ✔ vytvářet různé šifry a hlavolamy za využití práce s ASCII.

Samostatná práce (část 8)

a) Naprogramujte příkaz ČÍSLICE_NA_ZNAČKY, po jehož zadání **Karel projde systematicky všechna pole a nalezené číslice zamění za příslušný počet značek**.

b) Naprogramujte příkaz ATBAŠ, po jehož zadání Karel projde systematicky všechna pole a **nalezená písmena velké či malé abecedy zamění podle převodové tabulky substituční šifry atbaš**.

c) Naprogramujte příkaz ALBAM, po jehož zadání Karel projde systematicky všechna pole a **nalezená písmena velké či malé abecedy zamění podle převodové tabulky substituční šifry albam**.

d) Naprogramujte příkaz PODLE_PLOTU, který vhodným způsobem **realizuje transpoziční šifru nazývanou „podle plotu“**.

e) Naprogramujte příkaz ODEČTI(x,y), po jehož spuštění a následném vložení dvou přirozených čísel **Karel zobrazí pod sebe čísla *x* a *–y* a pak vypočte a zobrazí jejich rozdíl**, včetně případného znaménka mínus.

f) Naprogramujte příkaz ROZKLAD(n), po jehož spuštění Karel **načte přirozené číslo zadané uživatelem a zobrazí jeho rozklad na prvočinitele**.

g) Naprogramujte příkaz NSD(x,y), po jehož spuštění a následném vložení dvou přirozených čísel Karel **zobrazí pod sebe tato čísla a potom vypočte a zobrazí jejich největší společný dělitel**.

h) Naprogramujte příkaz PRVOČÍSLA, po jehož zadání Karel pod sebe vypíše **posloupnost prvních dvaceti prvočísel**.

i) Naprogramujte příkaz PASCAL_TROJ, po jehož zadání Karel **zobrazí prvních devět řádků Pascalova trojúhelníka**. Poslední zobrazený řádek tedy bude: 1 8 28 56 70 56 28 8 1.

j) Naprogramujte příkaz MAGIC, po jehož zadání Karel **zobrazí magický čtverec o velikosti 4 x 4. Vyplní jej tedy čísly 1 až 16**. Neznáte-li hlavolam magický čtverec, najděte si ukázkové zadání s řešením na internetu.

k) Pro desky zaplněné jedno a dvojcifernými čísly (viz cvičení *j* a *k*), určete příkaz NAJDI(n), který **systematicky prochází všechna pole a zastaví se při nalezení zadaného čísla *n***.

l) Naprogramujte příkaz PATNACTKA, který vhodným způsobem realizuje **známý hlavolam Samuela Loyda s čísly 1 až 15 a jednou volnou pozicí uvnitř čtverce o velikosti 4 x 4**. Volnou pozici Karel posune vždy podle právě stiknuté kurzorové klávesy.

m) Naprogramujte příkaz MINCE, který vhodným způsobem **realizuje známý hlavolam Martina Gardnera s mincemi**. Možné tahy nechť jsou vyznačeny písmeny a program reaguje na jejich stisk.

n) Naprogramujte příkaz LABYRINT, který způsobí, že **Karel najde cestu ven z bludiště, přičemž si vhodným způsobem značí cestu pokládáním a zvedáním značek**.

o) Naprogramujte příkaz JEZDEC, při jehož provádění se Karel **pohybuje jako šachový jezdec, cílová políčka označuje** (např. písmeny abecedy) a na dříve označená pole již nevstupuje, ale přeskakuje je.

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

5 Želví grafika LOGO

V Scratch a LOGO vznikly na stejné univerzitě, jak se navzájem liší?

V čem spočívá podstata takzvané "želví grafiky"?

EXTRA - Co to jsou fraktály? Uveďte příklady známých fraktálů.

Práce se seznamy je v LOGO velice mocným prvkem, uveďte nějaké příklady úloh zaměřených na práci se seznamy.

Co je to Multi-turtle mode a proč se jedná o pseudo-vláknovost?)

5.1 Úvod do jazyka LOGO a prostředí xLOGO

Nejstarším programovacím jazykem určeným nejen pro výuku dětí, ale k rozvoji logického a tvůrčího myšlení vůbec (u dětí i dospělých studentů) je LOGO. Jeho vznik byl ovlivněn pracemi švýcarského vývojového psychologa a duchovního otce konstruktivistické školy **Jeana Piageta** (1896 – 1980). Navrhl jej Piagetův žák, jeden z průkopníků umělé inteligence, známý americký matematik, informatik a pedagog **Seymour Papert** (* 1928). Papert pracoval s Piagetem v letech 1958 – 63 na ženevské univerzitě. *„Nikdo nerozumí mým myšlenkám tak dobře jako Papert“*, řekl prý o svém žákovi Piaget. Po svém příjezdu na MIT (Massachusettský technologický institut) navrhl Papert v roce 1967 programovací jazyk LOGO, založený částečně na programovacím jazyce LISP (bývá nazýván jazykem „umělé inteligence“), z nějž převzal např. práci se seznamy. Zcela novou myšlenkou byla tzv. želví geometrie, využívající jako základní princip kreslení čar tažených za postavou.

Je zajímavé, že první implementace jazyka LOGO počítala s připojením robota podobného mechanické želvě k počítači. Tehdy šlo samozřejmě o sálový počítač, protože první domácí počítače (s osmibitovými procesory) se měly objevit zhruba o 10 let později. i pro domácí počítače **existovali ovšem „želví roboti“**, kteří se připojovali k počítači kabelem a reálně vykonávali příkazy předávávané želvě programem v jazyce LOGO. Např. Tasman Turtle Robot firmy Denning International z roku 1979 (viz obrázek 83). Běžně však vystačíme s obrázkem želvy na monitoru počítače, podobně jako tomu bylo u jazyka KAREL. Na rozdíl od Karla, želva při

svém pohybu kreslí stopu, kterou vytváří zajímavé geometrické konstrukce. Želví krůčky jsou daleko drobnější a želva se může natočit do libovolného směru (azimutu), nejen na čtyři hlavní světové strany.

Také v současnosti je LOGO velmi populární dětský programovací jazyk s celou řadou implementací velmi rozdílné úrovně. k výborným **komerčním produktům patří Imagine LOGO** autorů z Univerzity Komenského v Bratislavě. My budeme pracovat s **open source implementací XLOGO**, která je volně dostupná (včetně dokumentace) na adrese http://xlogo.tuxfamily.org/ .

Obrázek 83: Tasman Turtle Robot (zdroj http://www.theoldrobots.com/turtle1.html)

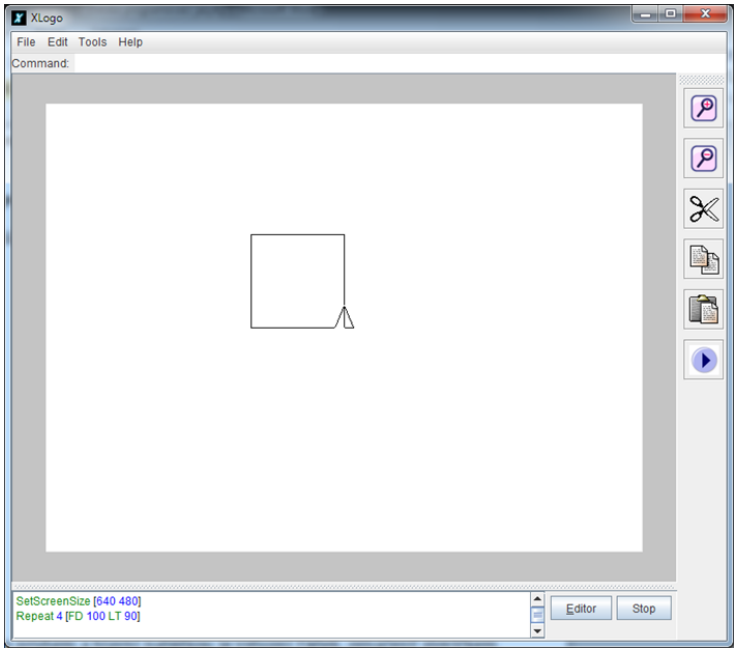
Stejně jako KarelWin se jedná o portable aplikaci, avšak XLOGO není úplně 100% portable, protože je pro jeho spuštění **nutné mít nainstalovanou Javu (JRE – Java Runtime Environment)**. Ta je sice ke stažení také zdarma, ale opět vyžaduje instalaci a není vždy jisté, že na daném PC již nainstalovaná bude. i této komplikaci se lze vyhnout použitím portable Javy, kterou lze například přidat do platformy PortableApps.com.

5.1.1 Základní pohyby želvy a možnosti kreslení

Aplikace XLOGO se spouští v okně, které můžeme vidět na obrázku 84. Pod titulkovým pruhem a hlavní nabídkou je vstupní řádek, označený slovíčkem Command, tedy příkaz. Sem budeme psát příkazy pro želvu, našeho nového robota. Příkaz pro pohyb vpřed ve směru natočení želvy je **Forward** a za něj napíšeme nejprve mezeru a pak číslo (později proměnnou, nebo výraz) určující o kolik kroků se má želva posunout. Např. **Forward 100** znamená posun želvy o 100 malých krůčků. Nebojte se, že by želva utekla někam daleko, a vyzkoušejte! Delší příkazy mívají navíc zkratky, takže místo **Forward** stačí psát **FD**. Podobně pro otočení proti směru hodinových ručiček o daný počet úhlových stupňů máme příkaz **Left**, který

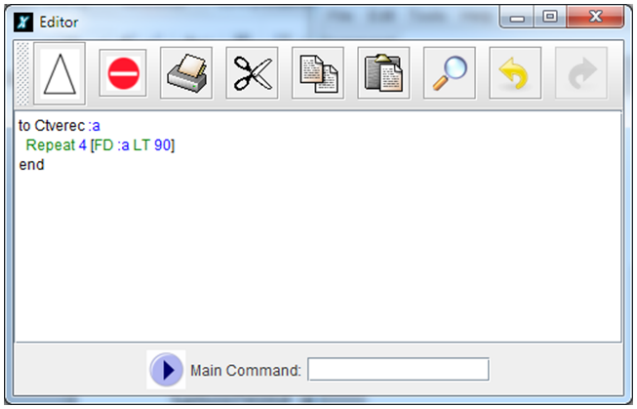
zkracujeme **LT**. Budeme-li chtít, aby želva nakreslila čtverec o straně 100 malých želvích kroků, stačí prostě napsat: **FD 100 LT 90 FD 100 LT 90 FD 100 LT 90 FD 100 LT 90**. Samozřejmě je výhodné použít příkaz pro opakování **Repeat** a napsat výrazně kratší příkaz: **Repeat 4 [FD 100 LT 90]**.

Zobecnění na čtverec o libovolné délce strany a vyžaduje práci s proměnnou a znalost postupu učení (tj. programování) nových příkazů. **Proměnnou označíme kombinací dvojtečky a písmene, tedy :a**. Pro naučení nového příkazu napíšeme **slovíčko to a za něj název procedury**, následovaný případně ještě proměnnou jako parametrem. Tedy například `to Ctverec :a`



Obrázek 84: Okno prostředí XLOGO

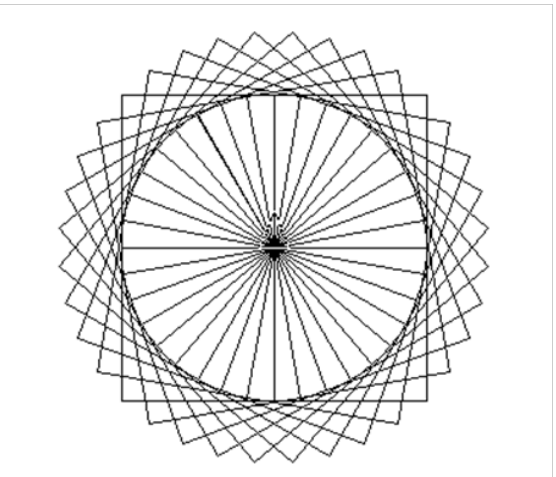
Po zadání této sekvence želva **neprovede žádný pohyb, ale místo toho se otevře editační okno, do něhož zapíšeme algoritmus nového příkazu** (viz obrázek 85). Do editačního okna se přepíše námi zadaný řádek, další řádek se nechá volný (prostor pro zápis algoritmu) a nakonec se doplní klíčové slovo **end**.



Obrázek 85: Editační okno pro tvorbu nových procedur

Kliknutím na první ikonu – obrázek želvy – příkaz uložíme. Od této chvíle už mu bude náš želví robot rozumět. Možnost opakování příkazů a natáčení želvy nejen o 90°, ale i o mnohem menší úhly lze využít ke kreslení esteticky zajímavých obrázků. Třeba i ze čtverců lze složit obrazec připomínající květ (viz obrázek 86).

```
to KvetC :a
  Repeat 36 [Ctverec :a LT 10]
end
```



Obrázek 86: Květ ze čtverců (výsledek procedury KvetC)

Jazyk LOGO umožňuje použití více parametrů u jednoho příkazu. Můžeme např. naprogramovat příkaz **Obdelnik :a :b**, kde **:a** a **:b** jsou délky stran obdélníka. z obdélníků o různých šířkách pak postavíme stupňovitou pyramidu. Na druhou stranu existují také příkazy bez parametrů. Příkladem jsou příkazy pro zvednutí pera **PenUp**, zkráceně **PU**, (po jehož provedení želva přestane při pohybu kreslit stopu) a jeho protějšek **PenDown**, zkráceně **PD** (po kterém želva znovu začne kreslit). Jiným důležitým příkazem je smazání celé kreslicí plochy **ClearScreen**, zkráceně **CS**, který nejen smaže „tabuli“, ale také vrátí želvu do výchozí pozice (želva je umístěna v počátku souřadnic, tj. v bodě [0, 0], a je natočena na sever.)

```
to Obdelnik :a :b
  Repeat 2 [FD :b LT 90 FD :a LT 90]
end

to Pyramida :s
  Obdelnik 5*:s :s PU FD :s LT 90 FD :s/2 RT 90 PD
  Obdelnik 4*:s :s PU FD :s LT 90 FD :s/2 RT 90 PD
  Obdelnik 3*:s :s PU FD :s LT 90 FD :s/2 RT 90 PD
  Obdelnik 2*:s :s PU FD :s LT 90 FD :s/2 RT 90 PD
  Obdelnik 1*:s :s PU FD :s LT 90 FD :s/2 RT 90 PD
  PU FD :s PD
end
```

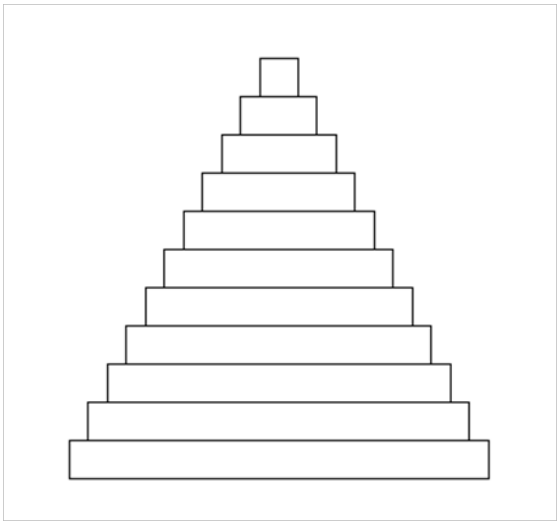
Tabulka 13: Základní příkazy pro pohyb želvy a základní příkazy ovlivňující kreslení

Příkaz	Zkratka	Význam příkazu
Forward :n	**FD :n**	Pohyb želvou vpřed o **:n** kroků
Back :n	**BK :n**	Pohyb želvou vzad o **:n** kroků
Left :a	**LT :a**	Otočení želvy proti směru hod. ručiček o úhel **:a** ve stupních
Right :a	**RT :a**	Otočení želvy ve směru hod. ručiček, tj. vpravo o úhel **:a** ve stupních
PenUp	**PU**	Zvedne pero, tj. nekreslí stopu
PenDown	**PD**	Položí pero, tj. kreslí stopu
HideTurtle	**HT**	Želva se zneviditelní (skryje)
ShowTurtle	**ST**	Želva se zviditelní (ukáže)
Wash		Smaže kreslicí plochu
Home		Umístí želvu do počátku, tj. na pozici [0, 0] a natočí ji na azimut 0°
ClearScreen	**CS**	Smaže kreslicí plochu a umístí želvu jako po příkazu **Home**
SetX	**:x**	Pohyb želvy vodorovně na pozici **:x**
SetY	**:y**	Pohyb želvy svisle na poz. **:y**
SetXY	**:x :y**	Pohyb želvy na pozici **:x :y**
SetHeading :a	**SetH :a**	Natočení želvy na azimut **:a** (90° … východ, 180° … jih, 270°… západ)
SetPenWidth :w	**SetPW :w**	Nastaví tloušťku pera na **:w** pixelů
SetPenShape :s	**SetPS :s**	Nastaví tvar hrotu pera na čtverec (0), nebo kruh (1)
SetPenColor :c	**SetPC :c**	Nastaví barvu, kterou bude želva kreslit. Lze použít číslo $c \in \{0, 1, 2, 3, \ldots 15, 16\}$, nebo klíčové slovo označující barvu, nebo seznam tří čísel od 0 do 255, která značí intenzitu složek ve schématu RGB
SetScreenColor :c	**SetSC :c**	Nastaví barvu pozadí kreslicí plochy

5.1.2 Proměnné, aritmetické operace, podmínky a rekurze

Procedura pro „želví“ kreslení pyramidy v minulé kapitole byla naprogramována dost těžkopádně. Několikrát se v ní opakovala stejná sekvence příkazů, měnila se pouze šířka jednotlivých stupňů, navíc byl počet stupňů předem dán. **Proceduru je možné výrazně vylepšit pomocí vhodné práce s proměnnými**. Už víme, že v okamžiku, kdy chceme použít hodnotu proměnné, napíšeme dvojtečku a za ni jméno proměnné. Pokud však chceme nějakou danou, či vypočtenou hodnotu uložit do proměnné, musíme její jméno uvést místo dvojtečky uvozovkami. Celý přiřazovací příkaz začíná klíčovým slovem **make**, za nímž následuje nejprve jméno proměnné, do níž uložíme výsledek (tj. jméno proměnné s uvozovkami) a potom číslo, nebo výraz, jehož hodnotu počítáme (proměnné s dvojtečkami).

```
to pyramida :n :s
  make "a :n*:s
  make "b :s
  Repeat :n [
    Obdelnik :a :b
    PU FD :s LT 90 FD :s/2 RT 90 PD
    make "a :a-:b
  ]
  HT
end
```



Obrázek 87: Pyramida s 11 stupni a výškou stupně 25

Podobnou procedurou, tentokrát ale bez parametrů, je postup pro vykreslení systému „soustředných“ čtverců. Obrázek, který navozuje představu postupného vkládání či vnořování čtverců do sebe, nám připomene rekurzi. Ale nejprve bez použití rekurze:

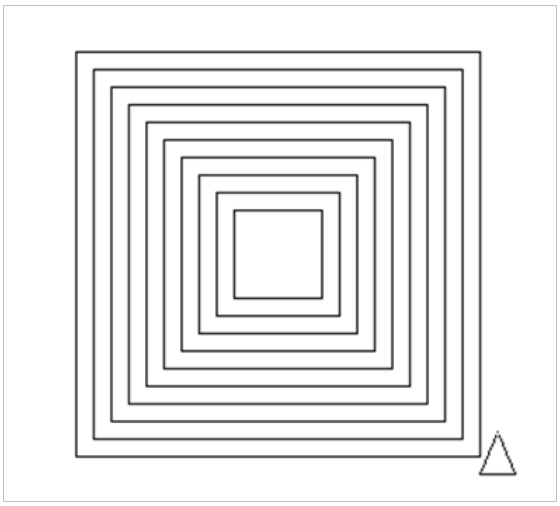
```
to SoustredneCtverce
  make "a 30
  Repeat 10 [
    make "a :a + 20
    Ctverec :a
    PU RT 90 FD 10 RT 90 FD 10 RT 180 PD]
end
```

Připomeňme si, že rekurze znamená volání sebe sama uvnitř těla procedury. Pro LOGO je rekurze naprosto přirozeným a velmi užitečným nástrojem. Abychom mohli rekurzivní volání procedury korektně ukončit, musíme se naučit používat podmíněný příkaz, který má tvar ***If*** *podmínka* ***[****seznam příkazů****]***. Podmínka je logický výraz, který může být buď pravdivý, nebo nepravdivý. Pokud je pravdivý, provedou se všechny příkazy uvedené v seznamu v hranatých závorkách. Pokud není pravdivý, celá sekvence v hranatých závorkách se přeskočí

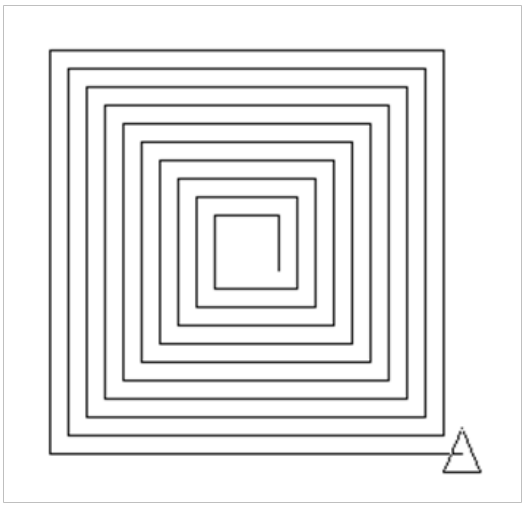
```
to sousctver :a :max
  ctverec :a
  make "b :a+20
  PU RT 90 FD 10 RT 90 FD 10 RT 180 PD
  if :b < :max [sousctver :b :max]
end
```

Podobně můžeme rekurzivně popsat kreslení spirály zhruba čtvercového tvaru:

```
to spiralactver :a:max
  FD :a LT 90
  make "b :a+5
  if :b < :max [spiralactver :b :max]
end
```



Obrázek 88: Soustředné čtverce



Obrázek 89: Čtvercová spirála

Pro čtyři základní početní operace můžeme užívat běžné aritmetické operátory, tedy +, –, * a /. U těchto čtyř operací můžeme, a u ostatních musíme použít zápis začínající názvem operace, za nímž jsou operátory uvedeny jako parametry. Tedy v proceduře **spiralactver** bychom místo **`make "b :a+5`** mohli psát pomocí infixové notace **`make "b sum :a 5`**. Pokud bychom chtěli např. zjistit, zda je číslo 654321 dělitelné devatenácti, zadáme příkaz **`print mod 654321 19`**. LOGO nám odpoví dole v textovém dialogu číslem 18. Zbytek po dělení 19ti je 18, takže číslo 654321 není dělitelné devatenácti beze zbytku.

Funkci modulo můžeme využít v upraveném Euklidově algoritmu pro výpočet největšího společného dělitele dvou čísel. Jde o rekurzivní algoritmus. Největší společný dělitel dvou čísel a a *b*, kde a *> b* je roven:

- ✔ číslu b, pokud je zbytek po dělení a mod *b* roven nule;
- ✔ největšímu společnému děliteli čísla *b* a zbytku z = a mod *b.*

Opět uplatníme rekurzi s ukončovací podmínkou.

```
to NSD :a :b
  if (:a > :b) [make "z mod :a :b
    if not (:z = 0) [NSD :b :z]
    if (:z = 0) [print :b make "z 1]]
  if (:a < :b) [NSD :b :a]
end
```

Jiným jednoduchým příkladem rekurze je výpočet faktoriálu, nebo výpočet kombinačního čísla. Chceme-li definovat funkci, tj. nový příkaz vracející hodnotu, musíme uvnitř její definice použít příkaz **output :x**, resp. **op :x**, který vrátí hodnotu proměnné (výrazu, či čísla) x. Jestliže chceme vyzkoušet, jak funguje příkaz funkce, předsuneme před její volání příkaz **print**, který způsobí zobrazení vypočtené hodnoty. Definujeme příkaz **faktorial :n** a následně jej vyzkoušíme, např. zadáním sekvence příkazů **print faktorial 5**, který vrátí hodnotu 120.

```
to faktorial :n
  if (:n > 1) [output product :n faktorial :n-1]
  if not (:n > 1) [output 1]
end
```

Tabulka 14: Seznam aritmetických operací a matematických funkcí

Příkaz	Zkratka	Význam příkazu
sum :x :y		Součet dvou čísel, totéž co **:x + :y**
difference :x :y		Rozdíl dvou čísel, totéž co **:x – :y**
minus :x		Číslo opačné k číslu **:x**
product :x :y		Součin dvou čísel, totéž co **:x * :y**
divide :x :y	**div :x :y**	Podíl dvou čísel, totéž co **:x / :y**
quotient :x :y		Celočíselný podíl dvou čísel
remainder :x :y	**rem :x :y**	Zbytek po celočíselném dělení dvou čísel
modulo :x :y	**mod :x :y**	Funkce modulo (*x* mod *y*)
round :x	**rnd :x**	Zaokrouhlení *x* na nejbližší celé číslo
integer :x	**int :x**	Celá část čísla *x*
power :x :y		Mocnina x na ypsilontou, tj. *x y*
squareroot :x	**sqrt :x**	Druhá odmocnina z čísla *x*
log :x		Přirozený logaritmus *x*
log10 :x		Dekadický logaritmus *x*
exp :x		Exponenciální funkce *e x*
sine :alfa	**sin :alfa**	Sinus úhlu *α (α* ve stupních)
cosine :alfa	**cos :alfa**	Kosinus úhlu *α (α* ve stupních)
tangent :alfa	**tan :alfa**	Tangens úhlu *α (α* ve stupních)
arcsine :x	**asin :x**	Arkus sinus *x* (výsl. ve stupních)
arccosine :x	**acos :x**	Arkus kosinus *x* (výsl. ve stupních)
arctangent :x	**atan :x**	Arkus tangens *x* (výsl. ve stupních)
pi		Číslo *π* (3,141592653589793)
random :n	**ran :n**	Náhodné celé číslo od 0 do *n*-1
alea		Náhodné reálné číslo od 0 do 1
absolute :x	**abs :x**	Absolutní hodnota z čísla *x*, tj. |*x*|
output :x	**op :x**	Uživatelem definovaná fce vrací hodnotu *x*
setdigits :n		Nastavuje počet číslic, tedy přesnost při provádění výpočtů; defaultně 16 číslic
digits		Vrací nastavený počet číslic, tedy přesnost při provádění výpočtů; defaultně vrací -1

Podobně můžeme definovat kombinační číslo *C(n,k)* pomocí vzorce, popisujícího známý vztah mezi prvky Pascalova trojúhelníka, tedy *C(n,k) = C(n–1,k) + C(n–1,k–1)*, a pomocí ukončovací podmínky pro *k=0*, resp. *k=n*, kdy platí *C(n,0) = C(n,n) = 1*. Logické operátory and a or používáme opět v infixové notaci, tj. píšeme je před oba argumenty (oba logické výrazy).

```
to kombinacnicislo :n :k
  if and (:n > :k) (:k > 0) [
    output sum kombinacnicislo :n-1 :k kombinacnicislo :n-1 :k-1
  ]
  if or (:n = :k) (:k = 0) [output 1]
end
```

Pokud bychom chtěli výslednou funkční hodnotu, nebo například nějaký nápis (řetězec znaků) napsat do grafické obrazovky, v níž se pohybuje želva, použijeme místo příkaz **print** příkaz **label**. Výstup se vypíše fontem velikosti 12 pt. Kdybychom chtěli psát větším písmem, např. velikosti 30 pt, uplatníme ještě před příkazem **label** příkaz **SetFontSize 30** (zkráceně **setFS :s**). Zarovnání nápisů, či výpisů čísel vůči poloze želvy nastavíme příkazem **SetFontJustify [:h :v]**, kde číslo *h* je horizontální zarovnání (0 … vlevo, 1 … na střed, 2 … vpravo) a číslo v vertikální zarovnání (0 … želva na lince, 1 … želva na středu písmen, 2 … želva nad písmeny).

Samostatná práce (část 9)

a) Využijte příkazy popsané v této podkapitole k zobrazení Pascalova trojúhelníka. Ať příkaz **Pascal :n** zobrazí Pascalův Δ, jehož poslední řádek tvoří binomické koeficienty, čili kombinační čísla *C(n, 0), C(n,1), C(n,2), …, C(n,n–1), C(n,n)*.

b) Naprogramujte příkaz **polygon :n :a**, který nakreslí pravidelný *n*-úhelník o straně délky *a*.

c) S využitím příkazu polygon a příkazů pro zvednutí a položení pera naprogramujte příkaz **plastev :a :b :r**, který vykreslí včelí plástev tvořenou pravidelnými šestiúhelníky o straně *r* (stranu šestiúhelníka značíme r proto, že v pravidelném šestiúhelníku se současně jedná o délku poloměru kružnice opsané), přičemž v jedné řadě bude a šestiúhelníků a plástev bude tvořit *b* řad buněk nad sebou.

d) S využitím příkazu polygon vytvořte nový příkaz **AZkviz**, který vykreslí hrací plán pro televizní soutěž AZ kvíz, do středů jednotlivých šestiúhelníků umístěte 28 písmen české abecedy.

e) Naprogramujte příkaz **domek1tahem :a**, který nakreslí známý domeček jedním tahem, přičemž délka strany čtverce, tvořícího dolní část domečku, je *a*.

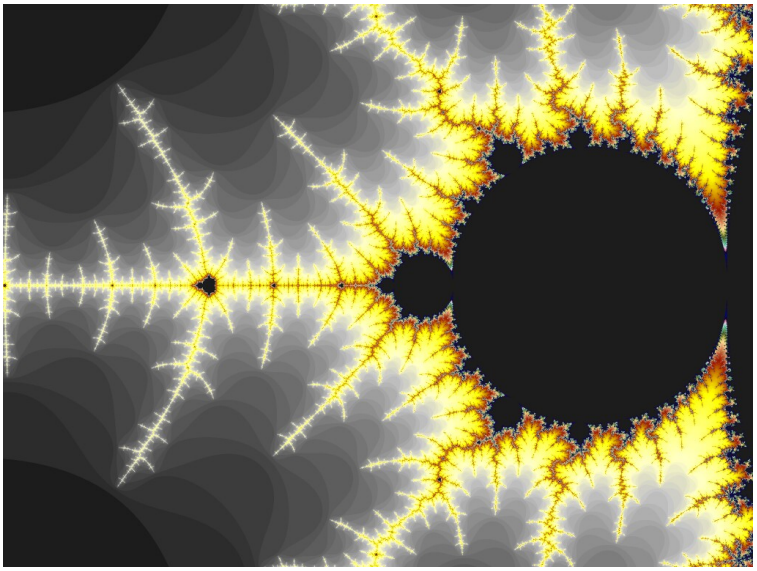
f) Naprogramujte příkaz **rozklad :k**, který rozloží číslo k na součin prvočinitelů a vypíše jej pomocí příkazu print. v případě, že k je prvočíslo, vypíše pouze toto číslo a skončí.

5.2 Fraktální útvary v xLOGO pomocí rekurze

Celá tato podkapitola je pojata jako rozšiřující učivo zaměřené na hlubší pochopení principů rekurze. Příklady zde prezentované se zaměřují na takzvanou fraktální geometrii a jsou určeny zejména pro zájemce z řad matematiků.

Programovací jazyk LOGO je ideálním nástrojem pro kreslení křivek či ploch, které představují prvky geometrických posloupností, jejichž limitou jsou objekty nazývané fraktály. Francouzský matematik **Benoît Mandelbrot** (1924 – 2010) o nich píše v úvodu své knihy (Mandelbrot, 2003): *„… fraktály jsou tvary, u nichž – nezávisle na smyslu, který těmto slovům dáme – detail reprodukuje část a část reprodukuje celek …“* a jen o několik odstavců dále cituje slova Eugèna Delacroixe: *„… větve stromů jsou samy malými úplnými stromy, úlomky skal se podobají skalním masivům, částečky půdy obrovským shlukům půdy. Jsem přesvědčen, že najdeme množství takových analogií. Ptačí péro je složeno z milionů per.“*

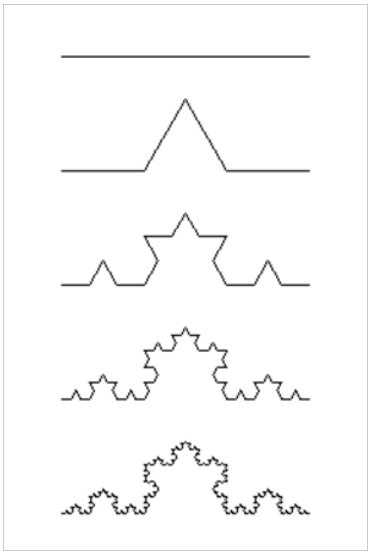
Obrázek 90: Fraktální kapradina



Obrázek 91: Mandelbrotova množina (zdroj: https://www.root.cz/clanky/fraktaly-v-pocitacove-grafice-xvi/)

Prvním zajímavým útvarem, který si ukážeme, je von Kochova křivka, kterou popsal v roce 1904 švédský matematik **Niels Fabian Helge von Koch** (1870 – 1924). Jeho záměrem bylo sestrojit křivku, která je spojitá a přitom nemá v žádném svém bodě tečnu. Posloupnost křivek, která vede k von Kochově křivce je určena rekurentně. Nultý člen je úsečka dané velikosti. Každý další člen získáme z předchozího tak, že všechny úsečky, z kterých se skládá, rozdělíme na třetiny, obě krajní části ponecháme, zatímco prostřední nahradíme dvěma úsečkami stejné

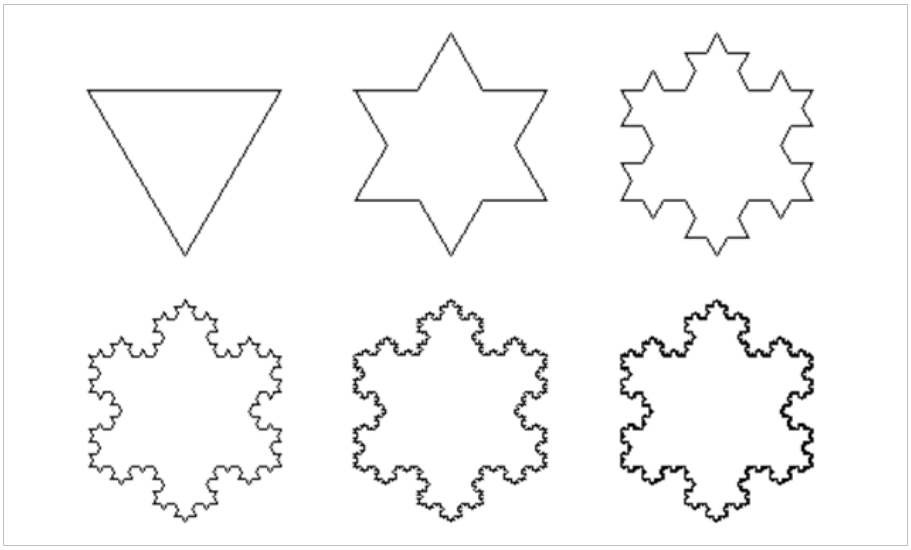
velikosti, jako je nahrazovaná úsečka (jako kdybychom sestrojili nad prostřední třetinou rovnostranný trojúhelník a jednu z jeho stran nahradili oběma zbývajícími stranami). Von Kochova křivka je pak limitou této posloupnosti.



Obrázek 92: Von Kochova křivka, prvních pět členů posloupnosti

```
to krivkaKoch :a :n
  if (:n > 0) [
    krivkaKoch :a/3 :n-1
    lt 60
    krivkaKoch :a/3 :n-1
    rt 120
    krivkaKoch :a/3 :n-1
    lt 60 krivkaKoch :a/3 :n-1
  ]
  if (:n = 0) [fd :a]
end
```

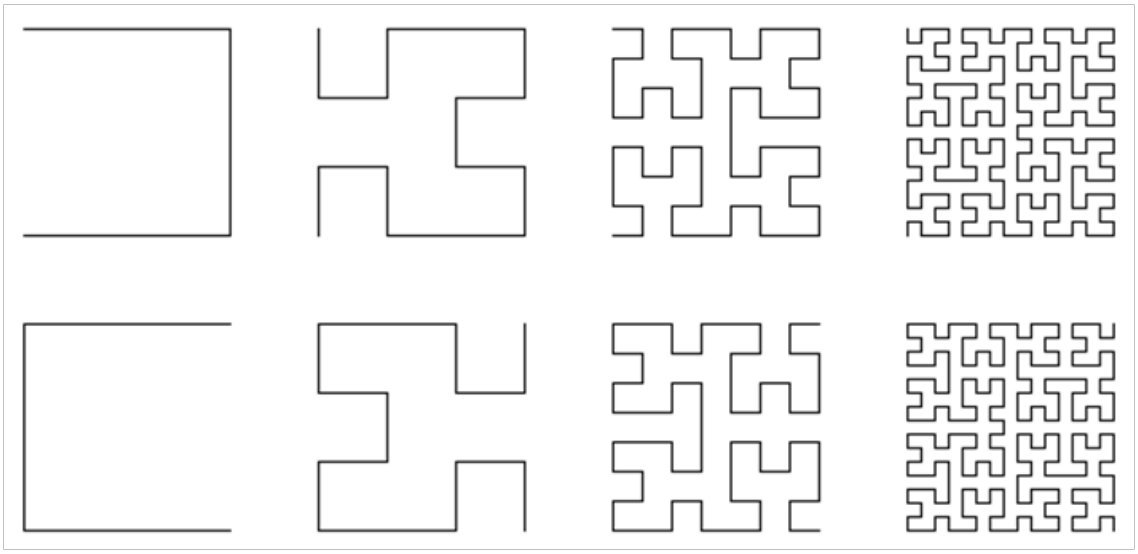
Spojíme-li tři von Kochovy křivky, vznikne von Kochova vločka. Zatímco délka hraniční křivky von Kochovy vločky roste nade všechny meze (stačí si uvědomit, že posloupnost délek jednotlivých křivek je geometrická posloupnost s kvocientem $q = 4/3$), obsah plochy obrazce ohraničeného touto křivkou je evidentně konečný.



Obrázek 93: Von Kochova vločka, prvních 6 členů posloupnosti

```
to vlockakoch :a :n
  Repeat 3 [krivkaKoch :a :n rt 120]
end
```

Křivku, která se snaží co nejlépe vyplnit danou čtvercovou plochu, sestrojil v roce 1891 německý matematik **David Hilbert** (1862 – 1943). Rekurze je tentokrát složitější, protože každá další Hilbertova křivka v posloupnosti se skládá ze dvou typů Hilbertových křivek (vyplňujících prostor vlevo, nebo vpravo od spojnice výchozí a cílové pozice želvy), ze spojovacích úseček a otočení o 90°. Celý program je tentokrát definován pomocí dvou vzájemně provázaných procedur **HilbertL** a **HilbertP**.

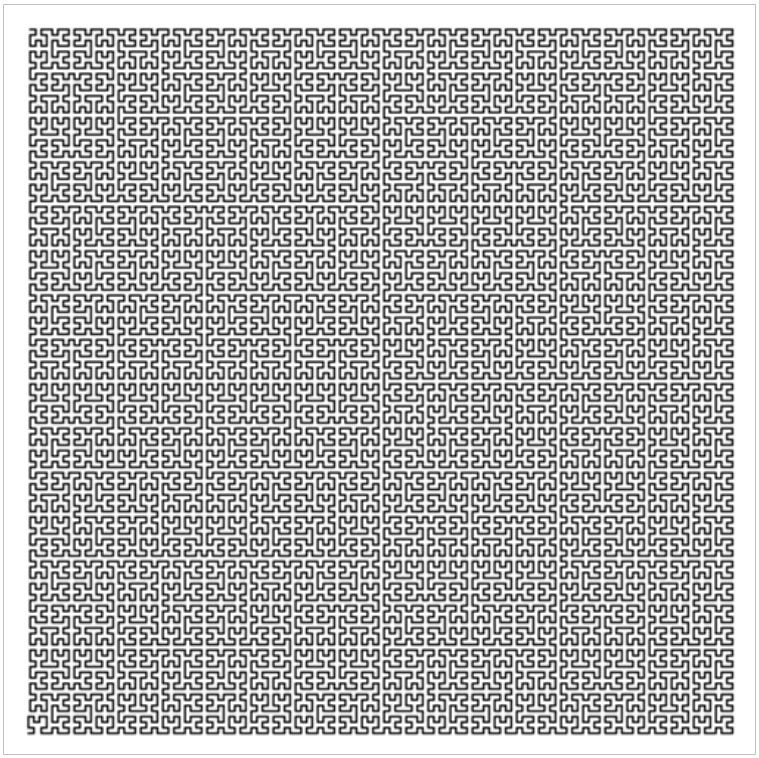


Obrázek 94: Hilbertovy křivky typu P (nahoře) a L (dole)

```
to HilbertL :a :n
  if (:n > 0) [
    lt 90
    HilbertP :a :n-1
    fd :a
    rt 90
    HilbertL :a :n-1
    fd :a
    HilbertL :a :n-1
    rt 90
    fd :a
    HilbertP :a :n-1
    lt 90
  ]
end
```

```
to HilbertP :a :n
  if (:n > 0) [
    rt 90
    HilbertL :a :n-1
    fd :a
    lt 90
    HilbertP :a :n-1
    fd :a
    HilbertP :a :n-1
    lt 90
    fd :a
    HilbertL :a :n-1
    rt 90
  ]
end
```

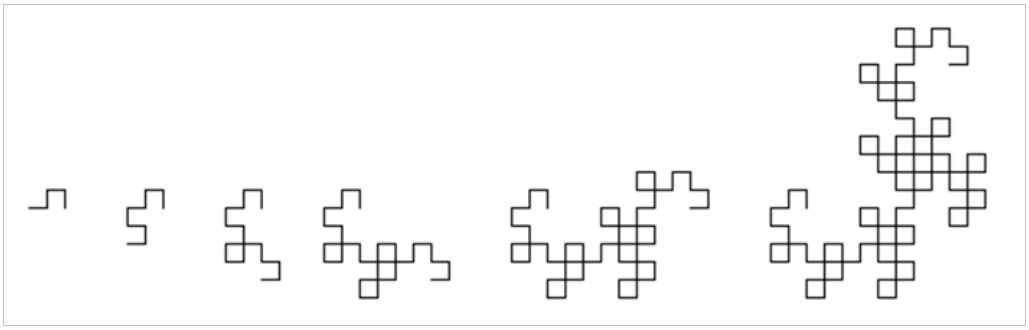
Posloupnost křivek na obrázku 95 byla nakreslena postupným voláním procedur s parametry 105 1, 35 2, 15 3 a 7 4.



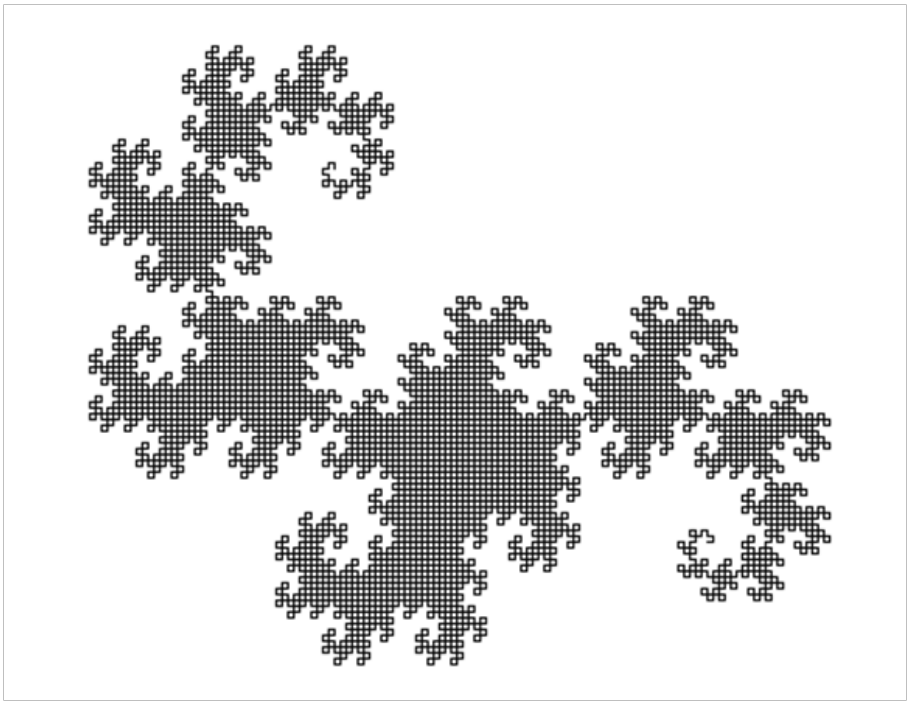
Obrázek 95: Hilbertova křivka (sedmý člen posloupnosti)

Třetí zajímavou fraktální křivkou je tzv. dračí křivka. Ve skutečnosti jsou dračí křivky, podobně jako Hilbertovy křivky, dvě různé křivky, ale liší se jen směrem otočení želvy mezi oběma rekurzivními voláními. Proto tentokrát nemusíme psát dvě různé procedury, ale přidáme jen jeden parametr navíc. Parametru pak přiřadíme řetězec **"L**, bude-li otočení směrem vlevo a **"R**, bude-li směrem vpravo.

```
to drak :o :a :n
  if (:n > 0) [drak "L :a :n-1]
  if (:n = 0) [fd :a]
  if (:o = "L) [lt 90]
  if (:o = "P) [rt 90]
  if (:n > 0) [drak "P :a :n-1]
  if (:n = 0) [fd :a]
end
```

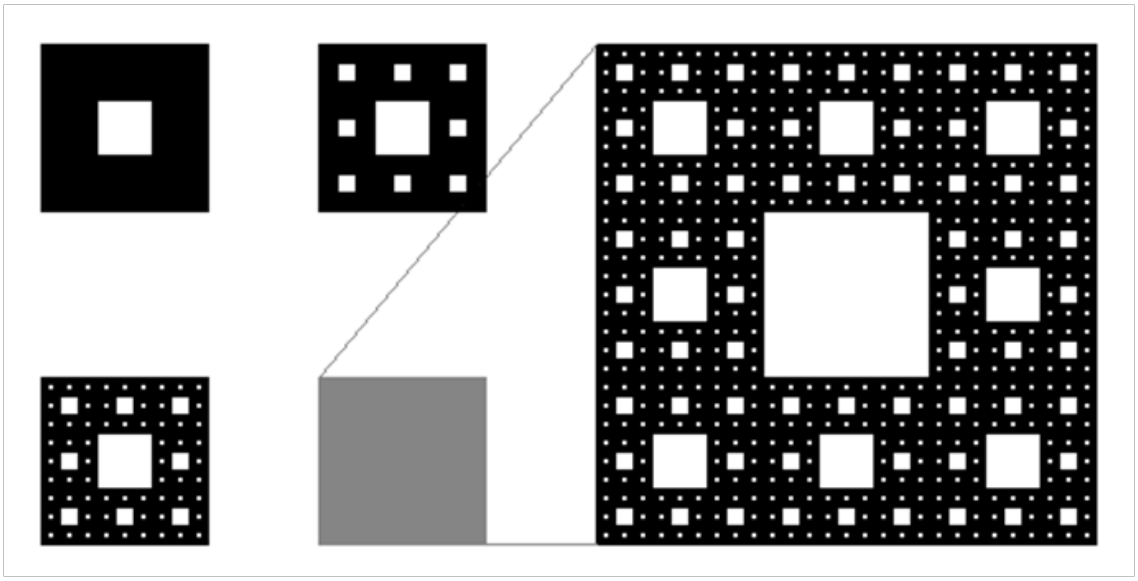


Obrázek 96: Dračí křivky, prvních 7 členů posloupnosti



Obrázek 97: Dračí křivka, dvanáctý člen posloupnosti

Další fraktální útvary nevychází z křivek, ale z plošných obrazců. Velmi známé jsou Sierpińského trojúhelník a Sierpińského koberec, které vzniknou rekurzivním odstraňováním rovnostranných trojúhelníků, resp. čtverců z dané trojúhelníkové, resp. čtvercové plochy. Oba tyto fraktály popsal polský matematik **Wacław Franciszek Sierpiński** (1882 – 1969) v letech 1915 a 1916. Ukažme si princip na koberci. Daný čtverec rozdělíme na 3 x 3 menších čtverců. Prostřední „vystřihneme“, tedy odstraníme a s ostatními osmi čtverci naložíme stejně jako s původním velkým čtvercem.



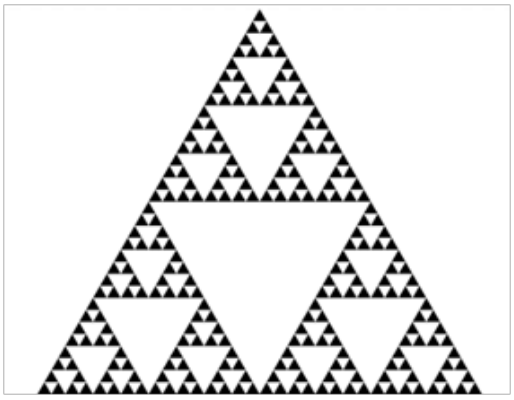
Obrázek 98: Sierpińského koberec, první čtyři členy posloupnosti

```
to serp :a :n
  if (:n > 0) [
    Repeat 4 [
      serp :a/3 :n-1 fd :a/3
      serp :a/3 :n-1 fd :a/3
      fd :a/3 rt 90
    ]
  ]
  if (:n = 0)    [
    Repeat 4 [fd :a rt 90]
    pu fd :a/3 rt 90 fd :a/3 lt 90 pd
    Repeat 4 [fd :a/3 rt 90]
    pu rt 180 fd 1 rt 90 fd 1 lt 90 pd fill
    pu fd :a/3-1 rt 90 fd :a/3-1 rt 90 pd
  ]
end
```

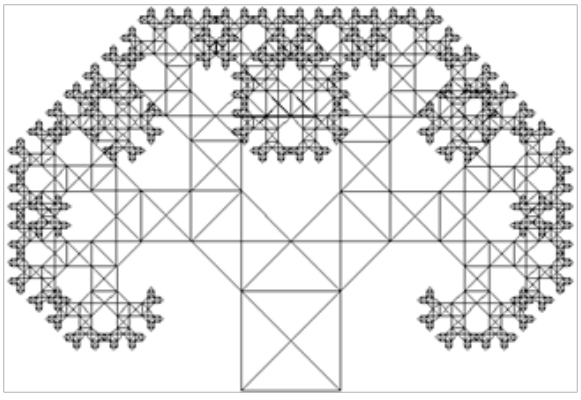
Samostatná práce (rozšiřující učivo)

a) S využitím rekurze naprogramujte **proceduru, která vykreslí *n*-tý člen posloupnosti Sierpińského trojúhelníka**. Výchozím obrazcem je rovnostranný rojúhelník o délce strany *a*.

b) Využijte rekurzi k **vykreslení stromu z domečků kreslených jedním tahem**. Celý takový strom je možné také kreslit jedním tahem, protože zmenšený strom budeme kreslit místo každé ze dvou úseček tvořících střechu domku.



Obrázek 99: Sierpińského trojúhelník



Obrázek 100: Strom z domečků 1 tahem

5.3 Práce se seznamy v jazyce LOGO

V úvodu této podkapitoly na chvíli opustíme grafické funkce a budeme se věnovat seznamům. **Seznam je uspořádaná množina hodnot, kterou zapíšeme do hranatých závorek a vzájemně oddělíme mezerami**. Prvky seznamu mohou být čísla, řetězce znaků, či další (vnořené) seznamy. Pro seznamy je definována v jazyce LOGO řada užitečných funkcí. Uvedeme si v tabulce jejich přehled a pak na příkladech i jejich použití.

Tabulka 15: Příkazy pro práci se seznamy a slovy

Příkaz	Význam příkazu
list a_1 a_2	Vrací dvouprvkový seznam, tvořený argumenty a_1 a a_2, např.: list [1 2] [3 4 5] → [[1 2] [3 4 5]]
sentence a_1 a_2	Vrací dvouprvkový seznam, tvořený argumenty a_1 a a_2, pokud je a_1 nebo a_2 seznam, spojí je do celku se [1 2] [3 4 5] → [1 2 3 4 5]
fput a_1 s_2	Vloží a_1 jako 1. prvek seznamu s_2
lput a_1 s_2	Vloží a_1 jako poslední prvek sez. s_2
reverse s	Invertuje řazení prvků seznamu *s*
remove a_1 s_2	Vyjme prvek a_1 ze seznamu s_2
pick s	Vrací náhodně vybraný prvek z s, s je seznam, nebo slovo.
item *n s*	Vrací *n*-tý prvek (písmeno) z *s*, s je buď seznam, nebo slovo
first s	Vrací první prvek (písmeno) z *s*
last s	Vrací poslední prvek (písmeno) z *s*
butfirst s	Vrací seznam nebo slovo kromě prvního prvku, nebo znaku
butlast *s*	Vrací seznam, nebo slovo kromě posledního prvku, nebo písmene
setitem s *n a* **replace s** *n a*	Nahradí *n*-tý prvek seznamu s novým prvkem s hodnotou *a*
additem s *n a*	Přidá *n*-tý prvek do seznamu s nový prvek s hodnotou *a*
count s	Vrací počet prvků seznamu, nebo počet písmen slova
unicode *p*	Vrací unicode hodnotu písmene *p* (tj. i číslice, či znaku)
character *n*	Vrací znak, jehož unicode hotnota je *n* (např. char 65 → A)
word w_1 w_2	Spojí dvě slova w_1 a w_2 do jednoho

„Slovem“ v této tabulace rozumíme buď řetězec znaků, nebo číslo, řetězec znaků označují uvozovky, zatímco číslo je složeno z číslic, slovo nikdy neobsahuje mezeru.

Prvním příkladem je **procedura generující Fibonacciovu posloupnost**. Tuto rekurentně definovanou posloupnost popsal ve 13. století italský matematik **Leonardo Pisano, zvaný Fibonacci** (cca. 1180 – 1250), zajímavá je její souvislost se zlatým řezem. První dva členy posloupnosti pevně určíme a každý další je pak součtem předchozích dvou členů. Nejprve definujeme funkci, která k seznamu prvních k členů této posloupnosti připojí člen s pořadovým číslem *k+1*, tedy při zadání seznamu [1 2 3 5 8 … a_{k-1} a_k] vrátí [1 2 3 5 8 … a_{k-1} a_k a_{k+1}], kde $a_{k+1} = a_k + a_{k-1}$.

```
to dalsifib :seznam
  output lput sum last :seznam last butlast :seznam :seznam
end
```

Tuto funkci pak opakovaně voláme v proceduře fibonacci :a, jejímž výstupem je seznam prvních n členů Fibonacciovy posloupnosti.

```
to fibonacci :n
  make "fibposl [1 2]
  Repeat :n-2 [make "fibposl dalsifib :fibposl]
  print :fibposl
end
```

Druhým, podobným příkladem, je **generování posloupnosti prvočísel**. Také v tomto případě si řešení rozdělíme na funkci, přidávající k dané posloupnosti prvočísel další nejbližší vyšší prvočíslo, kterou nazveme **dalsiprvocislo** (jejím argumentem i výstupem je opět seznam).

```
to dalsiprvocislo :seznam
  make "k last :seznam
  forever [
    make "k sum :k 1
    make "m 1
    foreach "j :seznam [make "m product :m modulo :k :j]
    if (:m > 0) [output lput :k :seznam]
  ]
end
```

Vlastní vypsání posloupnosti prvních *n* prvočísel zajistí procedura **prvocisla :n**, ve které na začátku explicitně uvedeme číslo 2 jako nejmenší prvočíslo a pak již opakovaně voláme funkci **dalsiprvocislo**.

```
to prvocisla :n
  make "seznam [2]
  write 2 write char 44 write char 32
  Repeat :n-1 [make "seznam dalsiprvocislo :seznam
    write last :seznam write char 44 write char 32]
  write "... print char 32
end
```

Obraťme na chvíli pozornost ke kódům a šifrám. Nejprve zakódujeme otevřený text pomocí jejich unicode hodnot. **Protože operace se slovy jsou často shodné jako operace se seznamy, můžeme volat následující proceduru se vstupním argumentem ve tvaru slova** (řetězce znaků, nebo čísla) stejně dobře jako ve tvaru seznamu znaků.

```
to koduj :vstup
  make "n 0
  make "vystup []
  Repeat count :vstup
    [make "n :n+1
    make "vystup lput unicode item :n :vstup :vystup]
  print :vystup
end
```

Pro opačný postup – dekódování – musíme pro vstup připravit seznam čísel. Ten se pak převede nejprve na seznam znaků a tyto znaky se pak spojí do řetězce znaků, čili slova.

```
to dekoduj :vstup
  make "n 0
  make "vystup1 []
  Repeat count :vstup [
    make "n :n+1
    make "vystup1 lput char item :n :vstup :vystup1
  ]
  make "vystup2 word first :vystup1 first butfirst :vystup1
  make "n 2
  Repeat difference count :vstup 2 [
    make "n :n+1
    make "vystup2 word :vystup2 item :n :vystup1
  ]
  print :vystup2
end
```

Jako ukázku jednoduché šifry můžeme zvolit např. šifru atbaš, ve které se první písmeno abecedy nahradí posledním, druhé předposledním atd. Tedy místo písmene a s kódem 65 napíšeme z s kódem 90, místo B s kódem 66 napíšeme Y s kódem 89 atd. Všimněme si, že vzájemně nahrazovaná písmena (použijeme-li pouze velká písmena) mají vždy součet kódů 155. Upravíme-li dříve naprogramované procedury **koduj** a **dekoduj** tak, že z nich vzniknou funkce (v obou stačí klíčové slovo **print** v předposledním řádku nahradit slovem **output**), napíšeme proceduru **atbas** poměrně snadno.

```
to atbas :vstup
  make "kody koduj :vstup
  make "n 0
  make "vystup1 []
  Repeat count :kody [
    make "n :n+1
    make "vystup1 lput difference 155 item :n :kody :vystup1
  ]
  make "vystup2 dekoduj :vystup1
  print :vystup2
end
```

V posledním příkladu práce se seznamy se vrátíme k rekurzi a k želví geometrii. Jedná se již o relativně komplexní úlohu, která je v kontextu těchto skript považována za rozšiřující. **Želva bude řešit známý hlavolam „Hanojské věže“**, přičemž aktuální složení jednotlivých tří věží bude uloženo ve třech seznamech a na základě obsahu těchto seznamů se budou zobrazovat jednotlivé kroky řešení. Připomeňme si, že hlavolam „Hanojské věže“ vymyslel francouzský matematik **Édouard Lucas** (1842 – 1891) v roce 1883. Hlavolam je složen ze tří kolíků, na které je možné navlékat kruhové disky různých poloměrů. Na začátku je všech n disků nasazeno na levém krajním kolíku, kde jsou seřazeny od největšího po nejmenší. Úkolem řešitele je přemístit všechny kotouče na prostřední kolík, přičemž pravý krajní je využíván jako pomocný odkládací prostor. Přitom je nutno dodržet následující pravidla:

- ✔ V jednom tahu lze přemístit jen jeden kotouč.
- ✔ Jeden tah znamená vzetí vrchního kotouče z některé věže a jeho položení na vrchol jiné věže.
- ✔ Je zakázáno položit větší kotouč na menší.

Program je realizován následujícími čtyřmi procedurami. Procedura **obdelnik** kreslí obdélník netradičně od středu „spodní“ strany. Procedura **kreslivez** nakreslí věž na základě aktuálního seznamu kotoučů. Jednotlivé kotouče jsou představovány obdélníky, které jsou navíc vyplněny barvami pro větší názornost. k vyplnění ohraničené plochy barvou se využívá příkaz **fill**, který vyplní plochu nastavenou barvou pera. Pro použití tohoto příkazu je nutné, aby želva stála uvnitř vyplňované plochy a měla pero „dole“. Zpravidla tedy musí želva, která dokreslila obrazec (v našem případě obdélník) zvednout pero, pak se posune dovnitř obrazce, spustí pero, nastaví barvu pera na zvolenou barvu výplně, použije funkci **fill** pro vyplnění plochy, zvedne pero, posune se na další výchozí pozici, spustí pero, změní barvu pera na barvu obrysů (zpravidla černou) a pak pokračuje kreslením dalšího obrazce v pořadí.

```
to obdelnik :a :b
  rt 90 fd :a/2 lt 90
  fd :b lt 90 fd :a
  lt 90 fd :b lt 90
  fd :a/2 lt 90
end
```

```
to kreslivez :a
  if not (:a = [])
  [make "prvni first :a
  make "dalsi butfirst :a
  make "sirka :prvni*20
  obdelnik :sirka 20
  pu fd 10 setpc :prvni fill
  fd 10 pd setpc black
  kreslivez :dalsi]
end
```

Další procedura je klíčová, slouží k provedení jednoho tahu a obsahuje dvě rekurzivní volání sebe sama. Poslední procedura **hanoi** pak už jen nastavuje výchozí stav seznamů i grafiky a spouští proces řešení.

```
to tah :n :z :do :pres
  if (:n = 0) [stop]
  tah :n-1 :z :pres :do
  if (:z = 1) [
    make "horni last :vlevo
    make "vlevo butlast :vlevo]
  if (:z = 2) [
    make "horni last :uprostred
    make "uprostred butlast :uprostred]
  if (:z = 3) [
    make "horni last :vpravo
    make "vpravo butlast :vpravo]
  if (:do = 1) [make "vlevo lput :horni :vlevo]
  if (:do = 2) [make "uprostred lput :horni :uprostred]
  if (:do = 3) [make "vpravo lput :horni :vpravo]
```

```
  cs
  pu setXY -150 0 pd kreslivez :vlevo
  pu setXY 0 0 pd kreslivez :uprostred
  pu setXY 150 0 pd kreslivez :vpravo
  ht wait 30 st
  tah :n-1 :přes :do :z
end

to hanoi :n
  make "vlevo []
  make "i 0
  Repeat :n
    [make "i :i+1
    make "vlevo fput :i :vlevo]
  make "uprostred []
  make "vpravo []
  cs pu setXY -150 0 pd kreslivez :vlevo
  ht wait 60 st
  tah :n 1 2 3
end
```

Samostatná práce (část 10)

a) Naprogramujte funkci **dvojcata :seznam**, která z daného seznamu prvočísel **vypíše všechna prvočíselná dvojčata**. Např. pro seznam [2 3 5 7 11 13 17 19 23 29 31 37 41 43 47] vypíše: 3 a 5, 5 a 7, 11 a 13, 17 a 19, 29 a 31, 41 a 43, ...

b) Naprogramujte procedury **realizující substituční šifru albam**, v níž se nahrazují odpovídající písmena z první poloviny abecedy písmeny z druhé poloviny a naopak. Tedy a ↔ N, B ↔ O, C ↔ P, ... L ↔ Y, M ↔ Z. Využijte přitom funkce **koduj** a **dekoduj**.

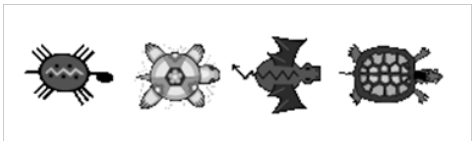
c) Naprogramujte procedury realizující transpoziční šifru **cikcak**, kdy do šifrového textu program vypíše nejprve všechna lichá písmena otevřeného textu a za ně pak všechna sudá písmena.

d) Naprogramujte dešifrovací program **decikcak**, který převede šifrový text získaný způsobem popsaným v předcházející úloze, zpět na otevřený text.

5.4 Multi-turtle mode v prostředí xLOGO

Pro rozhodování želvy v rámci našich programů můžeme používat různé funkce vyjadřující její aktuální stav. Když jsme se zabývali mikrosvětem robota Karla, měl Karel kompas, kterým určil své otočení na sever, jih, východ, či západ. Želva LOGO má podobný, ale přesnější kompas. Funkce **heading** určí její azimut, tj. úhel který svírá natočení želvy se severním směrem (měřeno ve směru hodinových ručiček), pak severu odpovídá azimut 0°, východu 90°, jihu 180° a západu 270°. Jiná funkce **position**, zkráceně **pos**, funguje jako želví GPS. Další funkce zjistí nastavení pera želvy, včetně jeho barvy a tloušťky.

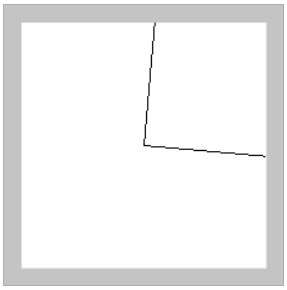
Využívat můžeme také některé booleovské funkce. Jejich **poznávacím znamením je otazník na konci jména funkce** a jejich základní vlastností, že vrací logickou hodnotu, tj. hodnotu **"true** (pravda) při splnění určité podmínky, vyjádřené názvem stručně v názvu funkce, a **"false** (nepravda), pokud podmínka splněna není. Booleovské funkce se mohou týkat jak stavu želvy (**pendown?**, **visible**?), tak typu a hodnot proměnných (**list?**, **word?**, **integer?**, **empty?**, …).



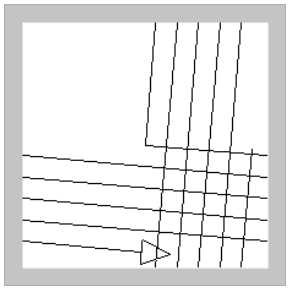
Obrázek 101: Různé tvary (obrázky) želvy

Mimo to, lze v prostředí programu také měnit tvar želvy ze strohého rovnoramenného trojúhelníka na různé obrázky želv, nebo **nastavovat či zjišťovat jednak rozměry kreslicí plochy, jednak chování plochy** ve chvíli, kdy želva dorazí na její okraj. Různé obrázky želvy se nám budou hodit také ve chvíli, kdy budeme chtít pracovat s několika želvami současně v režimu zvaném Multiturtle Mode. Ve skutečnosti však v daném okamžiku ovládáme vždy jen aktivní želvu.

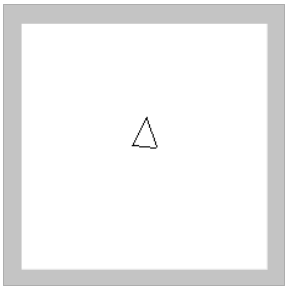
Vyzkoušejme, jak se projeví změna režimu kreslicí plochy z **window** na **wrap**, nebo **fence**. Přepneme příslušnými příkazy na jednotlivé režimy a pak zadáme sekvenci **cs rt 5 fd 1000 pu home pd rt 95 fd 1000**. Výsledek bude v každém z režimů jiný. Pokud předtím nastavíme rozměry kreslicí plochy 200 x 200 pixelů pomocí příkazu **SetScreenSize [200 200]**, bude takový jako na obr. 102 až 104.



Obrázek 102: Režim window



Obrázek 103: Režim wrap



Obrázek 104: Režim fence

Tabulka 16: Příkazy nastavení kreslicí plochy, Multiturtle Mode (MM) a přerušení

Příkaz	***Význam příkazu***
window	Kreslicí plocha je jako okno, želva může za jeho hranice
wrap	Kreslicí plocha je jakoby navinutá na válec, po překročení hranice se želva objeví na protější straně
fence	Hranice kreslicí plochy je jako plot, pokud by želva měla opustit svoji ohradu, systém ohlásí chybu
ScreenSize	Vrací rozměry kreslicí plochy jako 2- prvkový seznam [*width height*]
SetScreenSize [:width :height]	Nastaví velikost kreslicí plochy na šířku *width* a výšku *height*
shape	Vrací číslo označující tvar (obrázek) želvy → 0 až 6; 0 → trojúhelník
setshape :n	Nastavuje tvar (obrázek) želvy
turtlesmax	Vrací maximální počet želv v MM
setturtlesmax :n	Nastaví maximální počet želv
turtle	Vrací pořadové číslo aktivní želvy
setturtle :n	Vybere jako aktivní želvu číslo *n*
turtles	Vrací seznam, s čísly všech aktivních želv v MM
eraseturtle :n	Zruší – smaže želvu číslo n v MM
Stop	Uvnitř příkazu cyklu okamžitě ukončí cyklus a pokračuje příkazem za koncem cyklu, případně uvnitř procedury (ale mimo cyklus) ukončí okamžitě aktuálně prováděnou proceduru
StopAll	Okamžitě ukončí všechny užívané procedury a úplně zastaví program
output :a op :a	Ukončí okamžitě právě prováděnou proceduru a udělá z ní funkci; vrátí hodnotu proměnné či výrazu *a*

Tabulka 17: Příkazy zjišťující aktuální stav želvy a booleovské funkce

Příkaz	***Zkratka***	***Význam příkazu***
position	**pos**	Vrací dvouprvkový seznam, jehož prvky jsou souřadnice polohy želvy
x		Vrací vodorovnou souřadnici želvy
y		Vrací souřadnici želvy na svislé ose
heading		Vrací natočení = azimut želvy
towards [:x :y]		Vrací směr od želvy k bodu [*x, y*]

distance [:x :y]		Vrací vzdálenost želvy k bodu [*x, y*]
PenColor	**pc**	Vrací aktuální barvu pera [*r g b*]
ScreenColor	**sc**	Vrací aktuální barvu pozadí [*r g b]*
FindColor [:x :y]	**fc [:x :y]**	Vrací *[r g b]* barvu pixelu na pozici [*x y*]
PenWidth	**pw**	Vrací tloušťku pera v pixelech
PenShape	**ps**	Vrací tvar pera (0 → čtverec, 1 → kruh)
TRUE		Vrací hodnotu pravda → "true
FALSE		Vrací hodnotu nepravda → "false
pendown?	**pd?**	Vrací pravda → "true, když želva má pero dole (píše), jinak "false
visible?		Vrací pravda → "true, když je želva vidět (*show*), jinak (*hide*) "false
word? :arg		Vrací "true, když argument je typu slovo
integer? :arg		Vrací "true, když argument je celé číslo
list? :arg		Vrací "true, když arg. je typu seznam
empty? :arg		Vrací "true, když argument je prázdné slovo, nebo prázdný seznam, jinak "false
equal? :a1 :a2		Vrací "true, když a_1 a a_2 jsou si rovny
before? :a1 :a2		Vrací "true, když a_1 je abecedně před a_2
member? :w :s		Vrací "true, když *w* je prvkem seznamu s
member? :p :w		Podobně: je *p* písmenem slova *s*?

Abychom si mohli ukázat smysluplnou práci s více želvami (Multiturtle Mode), naučíme se nový příkaz **run :s**. Jeho argumentem je seznam příkazů, které se voláním příkazu **run** provedou. To se nám bude hodit, když budeme chtít předat stejné „pokyny“ postupně několika želvám.

```
to sextet :s
  make "n 0
  repeat 6
    [make "n :n+1
     setturtle :n
     run :s]
end
```

```
to startsextet
  make "n 0
  repeat 6
    [make "n :n+1
     setturtle :n
     home rt :n*60]
end
```

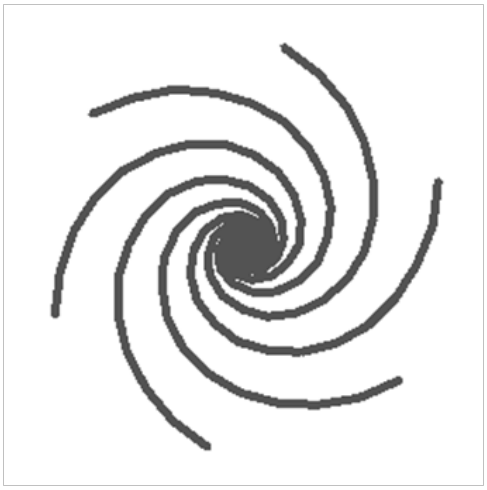
V tomto případě ovládáme šestici želv, které budou vykonávat stejné pohyby. k tomu poslouží procedura **sextet :s**, jejímž argumentem je seznam příkazů. Mimo to jsme si připravili druhou proceduru jménem **starsextet**, která umístí želvy do

středu kreslicí plochy a natočí jednotlivé želvy postupně na azimut 60°, 120°, 180°, 240°, 300° a 360°. Nyní můžeme například nechat růst (krystalizovat) sněhovou vločku, nebo modelovat vír:

```
to vlocka
  cs ht setsc blue
  startsextet
  sextet [setpc white setps 1 setpw 5 fd 75]
  wait 60
  sextet [lt 120 repeat 3 [rt 60 fd 30 bk 30]]
  wait 60
  sextet [lt 180 repeat 3
    [rt 60 fd 30 lt 120 repeat 3
      [rt 60 fd 12 bk 12]
    lt 60 bk 30]
  lt 60 bk 38]
  wait 60
  sextet [lt 60 fd 50 bk 50 rt 120 fd 50 bk 50 lt 60 fd 25]
end

to vir
  cs ht startsextet
  sextet [setpc red setps 1 setpw 5]
  make "k 0
  repeat 25
    [make "k :k+1
    sextet [fd :k lt 300/:k]
    wait 5]
end
```

Obrázek 105: Sněhová vločka



Obrázek 106: Vír

5.4.1 Různé typy cyklů v jazyce LOGO a kopretinová šifra

V jazyce LOGO nejčastěji využijeme cyklus s předem daným počtem n opakování, tedy **repeat :n [...]**. v jiných imperativních programovacích jazycích se často setkáváme s příkazem cyklu, který neurčuje jen počet opakování, ale stanoví výchozí a konečnou hodnotu, případně krok, o který se mění řídící proměnná po každém průchodu cyklem. Takový příkaz má LOGO také, ale jeho nevýhodou je, že počáteční i konečná hodnota musí být zadána jako číslo, nikoliv jako proměnná, či výraz. Zadáme-li např. **for [i 1 5] [print :i*:i]**, program pod sebe vypíše čísla 1, 4, 9, 16 a 25. Bezchybně funguje procedura, která vypíše prvních dvacet lichých čísel:

```
to lichacisla
  for [i 1 20]
    [write 2*:i-1 write char 44 write char 32]
  print "...
end
```

Jestliže však chceme proceduru zobecnit tak, aby vypsala prvních n lichých čísel:

```
to lichacisla :n
  for [i 1 :n]
    [write 2*:i-1 write char 44 write char 32]
  print "...
end
```

Upravená procedura vypíše po volání např. **lichacisla 15** chybové hlášení „**:n isn't a number!**“. Nezbyde nám tedy, než proceduru přepsat způsobem typickým pro jazyk LOGO:

```
to lichacisla :n
  make "i 0
  repeat :n
    [make "i :i+1 write 2*:i-1
    write char 44 write char 32]
  print "...
end
```

Příkaz **for** může mít v řídícím seznamu uvedeno ještě čtvrté číslo, které znamená krok, o nějž se zvyšuje řídící proměnná cyklu. Např. příkaz **for [i 1 9 2] [write :i*:i write char 44 write char 32]** vypíše posloupnost čísel 1, 9, 25, 49, 81.

Příkazem cyklu, který jsme již dříve použili ve funkci **dalsiprvocislo**, je příkaz **foreach "i :s [...]**, který vykoná příkazy uvnitř těla cyklu **[...]** postupně pro všechny prvky seznamu **:s**, nebo pro všechna písmena slova **:s**. Zadáme-li např. **foreach "i "NAZDAR [print :i]**, program pod sebe vypíše písmena N, A, Z, D, a a R. Podobně lehce můžeme vytvořit proceduru, která vypíše ciferný součet čísla *n*:

```
to cifernysoucet :n
  make "soucet 0
  foreach "c :n [make "soucet sum :c :soucet]
  print :soucet
end
```

Pokud nahradíme klíčové slovo **print** v předposledním řádku slovem **output**, získáme funkci vracející ciferný součet zadaného čísla. Použití cyklu foreach se seznamem si můžeme připomenout na funkci, která do seznamu prvočísel přidá další nejbližší vyšší prvočíslo:

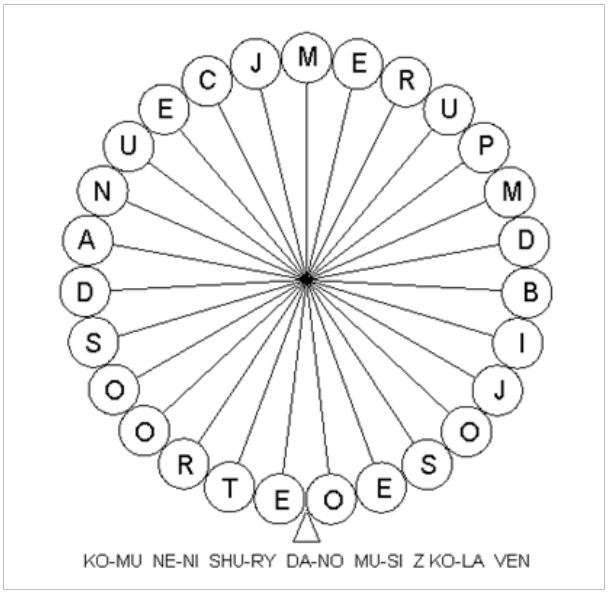
```
to dalsiprvocislo :seznam
  make "k last :seznam
  forever
    [make "k sum :k 1
    make "m 1
    foreach "j :seznam [make "m product :m modulo :k :j]
    if (:m > 0) [output lput :k :seznam]
    ]
end
```

Uvnitř této funkce je použit také další příkaz cyklu **forever**. Jak jeho název napovídá, jde o cyklus, který by probíhal navždy. Proto musí být uvnitř takového cyklu použit některý z příkazů přerušení, v tomto případě příkaz **output**.

Nakonec jsme si nechali tři podobné příkazy cyklu **while**, **repeatwhile** a **repeatuntil**, protože je potřeba vysvětlit rozdíly mezi nimi. Příkazy mají následující strukturu:

- ✔ **while [***podmínka***] [***tělo cyklu***],**
- ✔ **repeatwhile [***tělo cyklu***] [***podmínka***],**
- ✔ **repeatuntil [***tělo cyklu***] [***podmínka***].**

První rozdíl je vidět ze struktury. v případě cyklu **while** se nejprve vyhodnocuje podmínka, čili seznam příkazů, které jako celek vrací logickou hodnotu **"true**, nebo **"false**. Pokud se podmínka vyhodnotí hned na začátku jako nepravda → **"false**, neprovede se tělo cyklu ani jednou a program pokračuje ihned za koncem cyklu. Naproti tomu se v případě cyklů **repeatwhile** a **repeatuntil** vždy provede tělo cyklu alespoň jednou a teprve potom se vyhodnotí podmínka. v tuto chvíli se každý z těchto dvou příkazů chová jinak. U příkazu **repeatwhile** se cyklus opakuje, pokud je podmínka splněna → **"true**, jinak se cyklus ukončí. U příkazu repeatuntil se cyklus opakuje, pokud podmínka není splněna → **"false**, jakmile dosáhneme splnění podmínky, cyklus se ukončí.



Obrázek 107: Kopretinová šifra s rozpočítadlem

Ukažme si použití cyklu repeatuntil na příkladu procedury vytvářející kopretinovou šifru. Šifru dešifrujeme tak, že začneme písmenem úplně nahoře (azimut 0°) a pak počítáme další písmena ve směru hodinových ručiček pomocí rozpočítadla. Použitá písmena „škrtáme“ a opakovaně je už nepoužijeme. Proto musíme každé písmeno zkontrolovat, a pokud bylo použito, do rozpočítadla už jej nesmíme započítat. Řešení se skládá z funkce **preshranu :b :a a** vlastní procedury pro vytvoření kopretiny:

```
to preshranu :b :a
  if (:b <= :a) [output :b]
  if (:b > :a) [output sum mod :b :a 1]
end
```

```
to kopretinovasifra :n :k :text :rozpocitadlo
  make "vystup list 1 2
  make "i 2
  repeat :n-2
  [make "i :i+1 make "vystup lput :i :vystup]
  make "i 0 make "pozice 1 make "pocitadlo 0
  repeat :n
    [make "i :i+1
    make "vystup replace :vystup :pozice item :i :text
    make "j :pozice
    repeatuntil
      [make "j preshranu :j+1 :n
      if number? item :j :vystup [make "pocitadlo :pocitadlo+1]]
    [or :pocitadlo = :k :i =:n]
    make "pozice :j
    make "pocitadlo 0]
  print :vystup
  cs setfontsize 20 setfontjustify [1 1]
  make "i 0
  repeat :n
    [rt 360/:n*:i make "i :i+1
    fd 150 rt 90 repeat 120 [fd 1 lt 3]
    pu lt 90 fd 20 setheading 0
    label item :i :vystup home pd]
  setfontsize 14 pu bk 215
  label :rozpocitadlo fd 15
end
```

Pro zajímavost můžeme uvést také algoritmus pro zpětné dešifrování. U předchozí procedury jsou argumenty počet písmen textu, počet slabik rozpočítadla, otevřený text k zašifrování a text rozpočítadla. Tato procedura také požaduje zadat počet písmen textu, počet slabik rozpočítadla, pak ale následuje šifrový text (tj. řetězec písmen, který přečteme po obvodu kopretiny ve směru hodinových ručiček, počínaje písmenem úplně nahoře), který s textem rozpočítadla nepracuje vůbec.

```
to rozpocitej :n :k :text
  make "vystup list item 1 :text item :k+1 :text
  make "pouzite list 1 :k+1
  make "posledni :k+1
  repeat :n-2
    [make "pocitadlo 0
    make "j 0
```

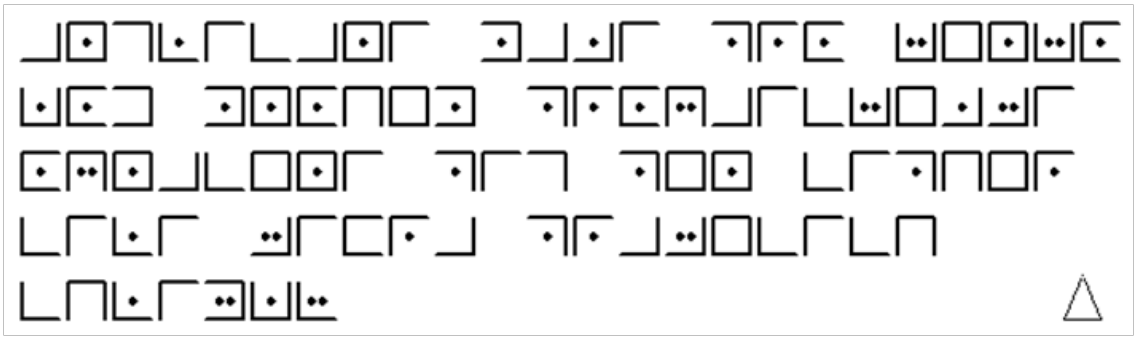
```
    repeatuntil
      [make "j :j+1
      if not member? preshranu :posledni+:j :n :pouzite
        [make "pocitadlo :pocitadlo+1]]
    [:pocitadlo = :k]
    make "vystup
    lput item preshranu :posledni+:j :n :text :vystup
    make "pouzite lput preshranu :posledni+:j :n :pouzite
    make "posledni preshranu :posledni+:j :n]
  print :vystup
  cs setfontsize 20 setfontjustify [1 1]
  make "i 0
  repeat :n
    [rt 360/:n*:i make "i :i+1
    fd 150 rt 90 repeat 120 [fd 1 lt 3]
    pu lt 90 fd 20 setheading 0
    label item :i :vystup home pd]
end
```

5.4.2 Komunikace s uživatelem, klávesnice, myš, GUI

Jazyk LOGO má řadu možností, jak komunikovat s uživatelem. Nejméně „akční“ je dialogové okno vyzývající k zapsání textu či čísla z klávesnice a jeho potvrzení (kliknutím na tlačítko OK či stiskem klávesy ENTER). k vyvolání dialogu slouží příkaz **read :s "w**, který vypíše text zadaný jako seznam slov s do záhlaví dialogu a řetězec zapsaný uživatelem uloží do proměnné *w*. Pomocí tří dialogů můžeme např. zpříjemnit uživateli ovládání procedury k vytvoření kopretinové šifry:

```
to dialogkopretina
  read [Zadejte text k sifrovaní bez mezer:] "text
  make "n count :text
  read [Zadejte pocet slabik rozpocitadla:] "k
  read [Zadejte text rozpocitadla v hranatych zavorkach:]
    "rozpocitadlo
  kopretinovasifra :n :k :text :rozpocitadlo
end
```

O stupínek více interaktivní je čtení hodnoty právě stisknuté klávesy. v příkladu ho využijeme v rámci programu, v němž želva po stisku klávesy zobrazí příslušné písmeno abecedy v tzv. zednářském kódu. Jak vypadá zpráva napsaná tímto „tajným“ písmem vidíme na obrázku 108.



Obrázek 108: Zpráva napsaná tzv. zednářským kódem

A	B	C
D	E	F
G	H	I

J	K	L
M	N	O
P	Q	R

•

S	T	U
V	W	X
Y	Z	?

••

Obrázek 109: Křížové tabulky pro dešifrování tzv. zednářského kódu

Následuje program, tvořený jedinou procedurou **zednarskykod**, který na stisk určité klávesy s písmenem reaguje vypsáním odpovídajícího znaku zednářského kódu. Stisk mezerníku, nebo klávesy ENTER udělá mezeru. Stisk klávesy ESC program ukončí.

```
to zednarskykod
  cs pu make "r screensize
  make "polosirka div first :r 2
  make "polovyska div last :r 2
  setxy 10-:polosirka :polovyska-30
  forever
    [make "kk readchar
    make "kod [3 0 0 0 0 mezera]
    if or :kk=65 :kk=97 [make "kod [0 0 0 1 1 A]]
    if or :kk=66 :kk=98 [make "kod [0 1 0 1 1 B]]
    if or :kk=67 :kk=99 [make "kod [0 1 0 0 1 C]]
    if or :kk=68 :kk=100 [make "kod [0 0 1 1 1 D]]
    if or :kk=69 :kk=101 [make "kod [0 1 1 1 1 E]]
    if or :kk=70 :kk=102 [make "kod [0 1 1 0 1 F]]
    if or :kk=71 :kk=103 [make "kod [0 0 1 1 0 G]]
    if or :kk=72 :kk=104 [make "kod [0 1 1 1 0 H]]
    if or :kk=73 :kk=105 [make "kod [0 1 1 0 0 I]]
    if or :kk=74 :kk=106 [make "kod [1 0 0 1 1 J]]
    if or :kk=75 :kk=107 [make "kod [1 1 0 1 1 K]]
    if or :kk=76 :kk=108 [make "kod [1 1 0 0 1 L]]
```

```
    if or :kk=77 :kk=109 [make "kod [1 0 1 1 1 M]]
    if or :kk=78 :kk=110 [make "kod [1 1 1 1 1 N]]
    if or :kk=79 :kk=111 [make "kod [1 1 1 0 1 O]]
    if or :kk=80 :kk=112 [make "kod [1 0 1 1 0 P]]
    if or :kk=81 :kk=113 [make "kod [1 1 1 1 0 Q]]
    if or :kk=82 :kk=114 [make "kod [1 1 1 0 0 R]]
    if or :kk=83 :kk=115 [make "kod [2 0 0 1 1 S]]
    if or :kk=84 :kk=116 [make "kod [2 1 0 1 1 T]]
    if or :kk=85 :kk=117 [make "kod [2 1 0 0 1 U]]
    if or :kk=86 :kk=118 [make "kod [2 0 1 1 1 V]]
    if or :kk=87 :kk=119 [make "kod [2 1 1 1 1 W]]
    if or :kk=88 :kk=120 [make "kod [2 1 1 0 1 X]]
    if or :kk=89 :kk=121 [make "kod [2 0 1 1 0 Y]]
    if or :kk=90 :kk=122 [make "kod [2 1 1 1 0 Z]]
    if or :kk=10 :kk=32 [make "kod [0 0 0 0 0 mezera]]
    if :kk=8 [pu lt 90 fd 25 rt 90]
    if :kk=27 [stop]
    make "c first :kod
    make "kod butfirst :kod
    if :c="1 [pu fd 10 rt 90 fd 10 rt 90
      pd setpw 5 fd 1 pu fd 9 rt 90 fd 10 rt 90]
    if :c="2 [pu fd 10 rt 90 fd 7 rt 90
      pd setpw 5 fd 1 bk 1 pu lt 90 fd 6 rt 90
      pd fd 1 pu fd 9 rt 90 fd 13 rt 90]
    if :c="3 [pu lt 90 fd 25 rt 90]
    setpw 2
    repeat 4
      [make "c first :kod
      make "kod butfirst :kod
      if :c=0 [pu] if :c=1 [pd]
      fd 20 rt 90]
      pu rt 90 fd 25 lt 90
    if x>:polosirka-30 [pu rt 180 fd 35 rt 90
      setx 10-:polosirka rt 90]]
end
```

Ještě bezprostřednější reakcí programu na požadavky uživatele, než je reakce na stisk kláves, nám zprostředkují funkce sledující počítačovou myš, konkrétně polohu kurzoru a stisk tlačítek myši. Polohu myši vrací příkaz **mouseposition**, zkráceně **mousepos** ve formě dvouprvkového seznamu, jehož první prvek reprezentuje vodorovnou souřadnici *x* a druhý prvek svislou souřadnici *y*. Příkaz **readmouse** identifikuje pohyb myši (při pohybech bez

stisknutého tlačítka vrací 0) a stisk tlačítek (při stisku a opětovném uvolnění levého tlačítka vrací 1; při stisku a opětovném uvolnění dalších tlačítek vrací 2, nebo 3). Příkaz **readmouse** můžeme jednoduše vyzkoušet v nekonečném cyklu napsáním **forever [write readmouse]**. Přitom si povšimneme, že myš „vrací nuly“ pouze při pohybu nad kreslicí plochou, mimo kreslicí plochu nereaguje ani na stisk tlačítek, a cyklus **forever** pak ukončíme stiskem tlačítka „Stop“ (které je mimo kreslicí plochu).

Sledování pozice kurzoru myši si ukážeme na proceduře, která navíc připomene Multiturtle Mode, protože „sledovat“ myš budou hned tři želvy. První se bude pohybovat pouze svisle po levém okraji kreslicí plochy, druhá vodorovně po dolním okraji a třetí bude stát na místě v pravém horním rohu, ale bude se natáčet směrem na kurzor. Navíc se naučíme nové příkazy související s tzv. Animation Mode. Animační mód je velmi užitečný při vytváření animací a také pro rychlejší vykreslování složitějších linií a ploch, protože bez nich by byl pohyb želv po kreslicí ploše trhaný. Animační mod spustíme příkazem **animation**, zkráceně **anim**. To způsobí, že želvy, jejich pohyby a stopy se nebudou vykreslovat okamžitě, ale vždy jen v okamžiku, kdy překreslení vyvoláme příkazem **repaint**. Animační mod je signalizován v levém dolním rohu okna obrázkem kamery a lze jej kdykoliv vypnout příkazem **stopanimation**, zkráceně **stopanim**. Vyzkoušejme následující program jak s použitím, tak s vynecháním příkazů **animation** a **repaint**:

```
to sledujtemys
  cs ht make "r screensize animation
  make "polosirka div first :r 2
  make "polovyska div last :r 2
  forever
    [make "p mousepos
    make "xm first :p
    make "ym last :p
    setturtle 1 setshape 6 st pu setheading 90
    setXY 40-:polosirka :ym
    setturtle 2 setshape 3 st pu
    setXY :xm 40-:polovyska
    setturtle 3 setshape 2 st pu
    setXY :polosirka-40 :polovyska-40
    make "hm towards :p
    setheading :hm
    repaint
    wait 1]
end
```

Tabulka 18: Základní barvy dostupné v jazyce XLOGO

Číslo barvy	*Název barvy*	*Český překlad*	*[R G B]*
0	**black**	černá	[0 0 0]
1	**red**	červená	[255 0 0]
2	**green**	zelená	[0 255 0]
3	**yellow**	žlutá	[255 255 0]
4	**blue**	modrá	[0 0 255]
5	**magenta**	purpurová	[255 0 255]
6	**cyan**	azurová	[0 255 255]
7	**white**	bílá	[255 255 255]
8	**gray**	šedá	[128 128 128]
9	**lightgrey**	světle šedá	[192 192 192]
10	**darkred**	tmavě červená	[128 0 0]
11	**darkgreen**	tmavě zelená	[0 128 0]
12	**darkblue**	tmavě modrá	[0 0 128]
13	**orange**	oranžová	[255 200 0]
14	**pink**	růžová	[255 175 175]
15	**purple**	nachová	[128 0 255]
16	**brown**	hnědá	[153 102 0]

Propojíme-li schopnosti příkazů **mousepos** a **readmouse**, můžeme vytvářet jednoduché GUI (grafické uživatelské rozhraní) s grafickými tlačítky na kreslicí ploše. Běžně užívaným tvarem tlačítek je obdélník. Současně si můžeme pohrát s barvami (viz tabulka 2.6) a vytvořenými tlačítky rozsvěcet světla na semaforu. Světla semaforu budou představovat tři kruhy nad sebou. Jejich středy budou mít souřadnice $S_1 = [0, 90]$, $S_2 = [0, 0]$ a $S_3 = [0, -90]$. Kružnice poloměru 40 vytvoříme příkazem **circle 40** a pak jimi ohraničenou plochu vyplníme zvolenou barvou pera příkazem **fill**. Zda je myš, v okamžiku kliknutí nad určitým obdélníkovým tlačítkem zjistíme porovnáním hodnot vrácených **mousepos** s hodnotami vodorovných a svislých hranic tlačítka.

```
to svetla :h :s :d
  pu setxy 0 90 pd setpc :h fill
  pu setxy 0 0 pd setpc :s fill
  pu setxy 0 minus 90 pd setpc :d fill
end
```

```
to semafor
  # Nejprve nakreslime zhasnuty semafor
  cs ht circle 40 pu fd 90 pd circle 40 pu
  back 180 pd circle 40 setpc [64 64 64]
  fill pu fd 90 pd fill pu fd 90 pd fill pu
  setxy 50 140 lt 90 setpc black pd fd 100
  lt 90 fd 280 lt 90 fd 100 lt 90 fd 280 pu
  bk 10 lt 90 fd 10 pd fill pu
  # a potom nakreslime ovladaci tlacitka
  setxy minus 280 140 setheading 180
  repeat 5
    [pd repeat 2 [fd 40 lt 90 fd 120 lt 90]
    pu fd 60]
  setxy minus 220 120 setfontjustify [1 1]
  pd setfs 20 setpc lightgray fill setpc black
  setheading 0 label "Stop! pu back 60
  make "popisky [Připravit Volno Pozor! Vypnuto]
  make "n 0
  repeat 4
    [make "n :n+1
    setpc lightgray fill setpc black
    label item :n :popisky pu back 60]
  # nyní je vše připraveno a semafor může fungovat
  forever
    [while [not readmouse=1]
    [make "p mousepos]
    make "xm difference 300 first :p
    make "ym sum 160 last :p
    make "dg [64 64 64]
    make "lo [255 220 0]
    # Je-li mys umistena nad tlacitkem, zmenime svetla
    if and :xm>20 140<:xm
    [if and :ym<300 260<:ym [svetla red :dg :dg]
    if and :ym<240 200<:ym [svetla red :lo :dg]
    if and :ym<180 140<:ym [svetla :dg :dg green]
    if and :ym<120 80<:ym [svetla :dg :lo :dg]
    if and :ym<60 20<:ym [svetla :dg :dg :dg]
    ]
  ]
end
```

5.4.3 Další možnosti jazyka XLOGO

Jazyk XLOGO má celou řadu dalších možností, na něž v tomto stručném učebním textu není prostor a v rámci předmětu *Dětské programovací jazyky* na ně nezbývá čas. Je však rozumné se o nich alespoň zmínit. Zájemce o danou problematiku si může najít potřebné informace např. v originální dokumentaci programu. Co jsme zcela vynechali:

- ✔ příkazy pro zobrazení mřížky a souřadných os;
- ✔ perspektivní zobrazení trojrozměrného obrazu – režim 3D;
- ✔ hraní hudby s využitím MIDI syntetizéru a přehrávání MP3;
- ✔ vkládání XLOGO aplikací do webových stránek;
- ✔ práci se soubory a adresáři a síťové funkce jazyka XLOGO.

Závěrečné shrnutí kapitoly 5

Po přečtení této kapitoly byste měli:

- ✔ popsat rozdíl mezi textovými jazyky KAREL a xLOGO;
- ✔ vědět, že LOGO vychází z jazyka LISP (důraz na seznamy);
- ✔ orientovat se v dokumentaci xLOGO a využívat ji k práci;
- ✔ vědět, že xLOGO zná ze všech probraných jazyků nejvíce různých příkazů cyklu (sedm). Jsou to příkazy repeat, for, foreach, forever, while, repeatwhile a repeatuntil;
- ✔ pracovat s více želvami najednou a efektivně využívat proceduru předávající jednotlivým želvám jejich instrukce;
- ✔ dokázat získat input od uživatele a to jak za pomoci klávesnice, tak myši;
- ✔ vytvořit vlastní GUI včetně tlačítek (viz dokumentace);
- ✔ vědět, že xLOGO má ještě mnoho dalších možností, ke kterým jsme se nedostali, ale naleznete je v dokumentaci.

Samostatná práce (část 11)

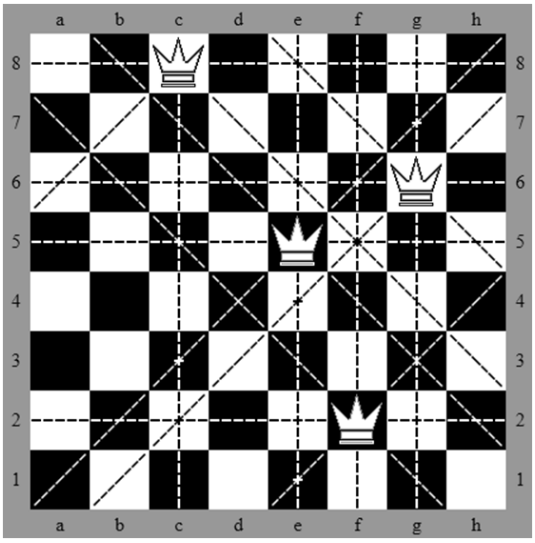
a) Naprogramujte proceduru, ve které se **želva natočí a posunuje na místo, na kterém je umístěn kurzor myši**. Při stisku levého tlačítka se cyklicky mění několik tvarů (obrázků) želvy.

b) **Nakreslete rovnoramenný trojúhelník, kruh a čtverec vhodných rozměrů**. Další kód procedury má ošetřit, že **po kliknutí dovnitř některého z těchto obrazců se cyklicky změní jeho barva** (červená → zelená → žlutá → modrá).

c) Naprogramujte šifrovací **program, který zednářský kód kombinuje s číselným heslem složeným pouze z číslic 0, 1, 2 a 3**, ve kterém 1 znamená otočení daného symbolu o 90°, číslice 2 otočení o 180° a číslice 3 otočení o 270° ve směru hodinových ručiček (nula = daný symbol se neotočí).

d) Naprogramujte kódovací program, který **přepíše zadaný text pomocí Morseovy abecedy**. Jako u zednářského kódu se želva na konci pravého okraje kreslicí plochy vrátí o řádek níž na levý okraj a pokračuje znovu směrem zleva doprava.

e) Naprogramujte v jazyce xLOGO **interaktivní hlavolam Pět dam na šachovnici**. Stejný hlavolam (ale v JavaScriptu) je na webu jednoho za autorů (http://www.musilek.eu/michal/ → Hry a hlavolamy → Pět dam na šachovnici). Úkolem řešitele je rozmístit na šachovnici 5 dam tak, aby kontrolovaly všechna pole šachovnice.



Obrázek 110: Hlavolam Pět dam na šachovnici

f) Naprogramujte v jazyce XLOGO **interaktivní hlavolam Deset sirek**. Stejný hlavolam, naprogramovaný v jazyce JavaScript najdete na webu na stejném webu, jako předchozích Pět dam. **Úkolem řešitele je pomocí skoků přes dvě sirky vytvořit z 10 volných sirek 5 křížků**.

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

6 Závěrečné slovo… a kam dál?

Jestliže jste se poctivě prokousali těmito skripty až do této části, nejenže vám náleží gratulace, ale zároveň byste měli být schopni orientovat se v problematice výuky základů programování pro žáky základních a středních škol.

Až budete stát v rámci své učitelské praxe před nelehkým úkolem výuky programování a algoritmického myšlení, nemělo by vás polévat horko a pokoušet se o vás panika a hysterie. Naopak byste měli být schopni vše zvládnout s naprostým klidem a chladnou hlavou. Vžijete-li se nyní do takovéto situace a máte-li tento pocit, ambice těchto skript jsou naplněny a autoři nemohou být šťastnější.

Vaše cesta v této vzdělávací oblasti však zdaleka nekončí. Vývoj jde ve světě informatiky a výpočetní techniky dopředu naprosto nevídanou rychlostí a zavřete-li na chvíli oči, zjistíte, že svět je zase o kus dál a vaši žáci také.

Navíc se tato skripta zaměřovala jen na úvod do problematiky výuky programování, který je pro základní školy a nespecializované střední školy s největší pravděpodobností dostatečný, ale zamíříte-li na střední školu specializovanou na problematiku kybernetiky, potřebujete mít znalosti mnohem, mnohem hlubší. Proto vám v následující podkapitole (a konečně již také poslední) nabízíme jakýsi odrazový můstek, kam můžete vaše vzdělávání dále směřovat.

Doporučená rozšiřující literatura

Dobrých učebnic a knih, podle kterých se můžete dále vzdělávat, je velmi mnoho, následující výčet tedy není v žádném případě absolutní a vyčerpávající. Jedná se však o ověřené tituly, podle kterých byste s trochou snahy (a občas také velkou mírou angličtiny) měli být schopni posunout se dále. Seznam titulů je abecední a nereflektuje míru doporučení dané knihy (například jakoukoliv knihu od autora jménem Al Sweigart lze vyvažovat zlatem).

V seznamu jsou také uvedeny vzdělávací materiály vzniklé v rámci projektu PRIM, které nebyly zmíněny v průběhu těchto skript. Materiály uvedené například v kapitole 3.1 při rozboru možností výuky za pomoci robotů a robotických stavebnic zde již znovu opakovány nejsou.

BLAHO, Andrej a Ivan KALAŠ. (2006). *Imagine Logo: učebnice programování pro děti*. Brno: Computer Press, 2006. Česká škola (Computer Press). ISBN 80-251-1015-X.

BLAHO, Andrej, Ľubomír SALANCI a Václav ŠIMANDL. (2019). *Základy programování v jazyce Python pro střední školy* [online]. ©2019 [cit. 2019-08-10]. Dostupné z: https://imysleni.cz/ucebnice/zaklady-programovani-v-jazyce-python-pro-stredni-skoly

ČERNOCHOVÁ, Miroslava, ŠTÍPEK, J. & VAŇKOVÁ, P. (2019). *Programování ve Scratch II: projekty pro 2. stupeň základní školy* [online]. ©2019 [cit. 2019-08-16]. Dostupné z: https://imysleni.cz/ucebnice/programovani-ve-scratchi-ii-projekty-pro-2-stupen-zakladni-skoly

Learn How to Think with Karel the Robot [online]. 2018 [cit. 2019-08-19]. Dostupné z: https://community.nclab.com/question/learn-how-to-think-with-karel-the-robot-about/

PILGRIM, Mark. (2011). *Dive Into Python 3: All you need to know to get off the ground with Python 3* [online]. apress [cit. 2019-08-22]. Dostupné z: https://www.diveinto.org/python3/table-of-contents.html

SWEIGART, Albert. (2015). *Invent Your Own Computer Games with Python* [online]. ©2008-2015 [cit. 2019-08-09]. Dostupné z: https://inventwithpython.com/inventwithpython_3rd.pdf

SWEIGART, Al. (2015). *Making Games with Python and Pygame: a guide to programming with graphics, animation, and sound*. 2015. ISBN 978-1469901732.

SWEIGART, Al. (2016). *Scratch programming playground: learn to program by making cool games*. San Francisco: No Starch Press, [2016]. ISBN 15-932-7762-8.

VANÍČEK, Jiří, Ingrid NAGYOVÁ a Monika TOMCSÁNYIOVÁ. (2019). *Programování ve Scratch pro 2. stupeň základní školy* [online]. ©2019 [cit. 2019-08-16]. Dostupné z: https://imysleni.cz/ucebnice/programovani-ve-scratchi-pro-2-stupen-zakladni-skoly

Poděkování

Na aktualizaci probíraných témat uvedených v této práci a jejich co nejsrozumitelnějším podání se kromě autorského kolektivu v průběhu let svými postřehy podílela též Mgr. Kristýna Horniková. Za tuto konstruktivní spolupráci a podporu jí patří náš dík.

Dále děkujeme recenzentu doc. Václavu Vrbíkovi za cenné připomínky a postřehy, díky kterým jsme opravili některé drobné nedostatky skript a zlepšili srozumitelnost některých pasáží.

Tomáš Hornik

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