Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Online nástroje pro výuku algoritmizace

Pavel Procházka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Pavel Procházka

Informatika

Název práce

Online nástroje pro výuku algoritmizace

Název anglicky

Online tools for algorithms

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku nástrojů pro podporu výuky algoritmizace. Hlavním cílem práce je analyzovat a zhodnotit dostupné online nástroje pro výuku algoritmizace.
Dílčí cíle práce jsou:
- charakterizovat dostupné softwarové nástroje pro podporu výuky algoritmizace,
- analyzovat požadavky na online nástroje pro výuku algoritmizace,
- zhodnotit vybrané online nástroje pro výuku algoritmizace.

Metodika

Teoretická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů.

V praktické části práce budou na základě poznatků zjištěných v teoretické části zhodnoceny

požadavky na online nástroje pro výuku algoritmizace. Pomocí metody vícekriteriální analýzy variant budou zhodnoceny vybrané online nástroje pro výuku algoritmizace. Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry bakalářské práce.

Doporučený rozsah práce

35

Klíčová slova

programování, Snap!, Scratch, Petr, analýza, porovnání

Doporučené zdroje informací

BENTLEY, Jon Louis. Programming pearls. 2nd ed. Boston, Mass.: Addison-Wesley, c2000. ISBN 0-201-65788-0.

BORY, Pavel. C# bez předchozích znalostí. Brno: Computer Press, 2016. ISBN 9788025146866.

HYLMAR, Radek. Programování pro úplné začátečníky. Brno: Computer Press, 2009. ISBN 9788025121290.

WANG, Wally. Beginning programming all-in-one desk reference for dummies. Hoboken, NJ: Wiley Pub., c2008. ISBN 9780470108543.

Předběžný termín obhajoby

2018/19 LS – PEF

Vedoucí práce

Ing. Michal Stočes, Ph.D.

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 15. 10. 2018

Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 10. 2018

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 04. 03. 2019

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Online nástroje pro výuku algoritmizace" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Michalu Stočesovi, Ph.D. z katedry informačních technologií, za odborné vedení mé bakalářské práce za cenné rady, vstřícnost a čas, který mi věnoval při konzultacích.

Online nástroje pro výuku algoritmizace

Abstrakt

Předmětem této bakalářské práce na téma „Online nástroje pro výuku algoritmizace“ je analýza, zhodnocení a porovnání vybraných nástrojů. Těmi jsou Scratch, Snap, Petr a robot Karel. V teoretické části je nejprve vysvětlena algoritmizace a význam vývoje výukových prostředí, jako základ pro programování. Následně jsou pak z odborných zdrojů analyzovány nástroje, zejména z nápověd a manuálů. Je zde uvedené, jaké má uživatel při práci s nástroji možnosti. Důraz je pak dávám na funkce, které potřebují noví uživatelé, tedy ti, co s programováním začínají a na které jsou tyto nástroje cíleny. Praktická část obsahuje vlastní výzkum a hodnocení nástrojů pomocí vědeckých metod. Nejprve je uvedeno sedm kritérií, která jsou pro nástroj pro výuku algoritmizace nejdůležitější. Metodami vícekriteriální analýzy variant jsou nástroje zhodnoceny. Přesněji, váhy kritérii jsou vypočteny párovou metodou přes Fullerův trojúhelník. Vybrání kompromisní varianty je provedeno metodou pořadí. Jsou zde i doporučení, který výukový nástroj použít při výuce, plynoucí z výsledků výzkumu.

Klíčová slova: algoritmizace, programování, Scratch, Snap, Petr, Karel, analýza, porovnání

Online tools for algorithms

Abstract

The subject of this bachelor thesis on “Online tools for algorithms” is the analysis, evaluation and comparison of selected tools. These are Scratch, Snap, Peter and robot Charles. In the theoretical part, the algorithm is first explained and why it is necessary to develop the learning environment as a basis for programming. After that, tools are analysed from expert sources. Especially from help and manuals. There are introduced what options users have, when they are working with tools. The emphasis is on the functions that new users need, those that just begin with programming and on which these tools are targeted. In the practical part there is a research and evaluation of tools with mathematical methods. First, there are 7 criteria, that are the most important for the learning tools. There are evaluated with methods of multi-criteria analysis of variants. More precisely, the weights of the criterion are calculated by a pairwise method with Fuller’s triangle. Choosing a compromise variation is done with method of order. There are also recommendations, which learning tool use, resulting from the research results.

Keywords: algorithm, programming, Scratch, Snap, Petr, Karel, analysis, comparison

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Ve 21. století jsou lidé obklopeni počítači ze všech stran. Často se pak ani nemohou bez počítačů obejít ať už v pracovním nebo i osobním životě. Tato technologie jde kupředu mílovými kroky a má velkou budoucnost. Proto je důležité, aby lidé měli základní porozumění, jak fungují. Zejména děti by se měly s počítači a jejich logikou seznamovat už od brzkých let, tedy na základní a střední škole.

Jak už bylo řečeno, počítače mají velkou budoucnost. Už v dnešní době usnadňují spoustu práce a šetří lidem čas. Je snahou, aby počítače zastaly repetitivní úlohy a odlehčily tak práci, lidé se pak mohou více soustředit na činnost, kterou počítače zatím zastat nemohou.

Pokud se má daný problém vyřešit přes počítač, je nutné problém analyzovat a popsat postup. Tento postup zapíšeme pomocí algoritmu. Samotný algoritmus počítač nedokáže vypočítat. Nejprve je nutné algoritmus přepsat do programovacího jazyka, kterému počítač rozumí a zvládne ho zpracovat. Zápis algoritmu ve zvoleném programovacím jazyce se nazývá program.

V současné době existuje velké množství programovacích jazyků. Každý jazyk má svá pravidla pro zápis algoritmů, která je nutné se naučit. A právě tato pravidla jsou problémem. Bývají často složitá a náročná na výuku, proto se hodně lidí rozhodne s programováním skončit hned na začátku. Problém neřeší ani změna jednoho programovacího jazyka na druhý. Opět se lidé musí učit nová pravidla, ač jsou některá podobná nebo shodná. Dalším problémem při zápisu algoritmu je, že člověk ani neví, s čím vším počítač dokáže pracovat. Tento problém řeší výuková prostředí, kterými se lze vyhnou složitostem při učení programovacího jazyka. V této práci jsou rozebrány hned čtyři nástroje na výuku – Scratch, Snap, Petr a robot Karel. Každý z těchto nástrojů má odlišný způsob vyřešení této problematiky. Scratch a Snap uživateli ihned ukážou, co je možné udělat. Příkazy jsou rozdělené podle funkcí do bloků a barevně rozlišeny pro lepší orientaci. Tyto bloky uživatel skládá do sebe jako puzzle a nemusí si dělat starosti se složitými pravidly. U nástroje Petr je usnadnění pomocí obrázků. Všechny tyto výukové nástroje jsou zaměřeny na výuku pro děti. Používají je však pro všechny věkové kategorie. Zhodnocení nástrojů a problém s výběrem řeší tato bakalářská práce.

2 Cíl práce a metodika

2.1 Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku nástrojů pro podporu výuky algoritmizace.

Hlavním cílem práce je analyzovat a zhodnotit dostupné online nástroje pro výuku algoritmizace.

Dílčí cíle práce jsou:

- charakterizovat dostupné softwarové nástroje pro podporu výuky algoritmizace
- analyzovat požadavky na online nástroje pro výuku algoritmizace
- zhodnotit vybrané online nástroje pro výuku algoritmizace

2.2 Metodika

Teoretická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů.

V praktické části práce budou na základě poznatků zjištěných v teoretické části zhodnoceny požadavky na online nástroje pro výuku algoritmizace. Pomocí metody vícekriteriální analýzy variant budou zhodnoceny vybrané online nástroje pro výuku algoritmizace. Na základě syntézy teoretických a praktických poznatků budou zpracovány závěry bakalářské práce.

3 Teoretická východiska

3.1 Algoritmus a algoritmizace

V současnosti si málokdo dokáže představit pracovat nebo žít bez počítače. Počítače postupně pronikly jak do většiny pracovních odvětví, tak do osobních životů (minimálně pro komunikaci s ostatními, jako je email apod.). Spoustu lidí počítač používá na řešení různých problémů a úkolů, vykonávaným programem v počítači. Postup v programu se nazývá algoritmus. Tvorba algoritmu se pak nazývá algoritmizace. (Evropský sociální fond v ČR, 2008)

Algoritmus

Algoritmus je přesný popis, jak vyřešit problém. Tedy popis, který definuje proces, který vede od zadání vstupních dat až po výsledek. Aby se jednalo o algoritmus, musí splňovat 3 základní vlastnosti: determinovanost, hromadnost a resultativnost.

- Determinovanost znamená, že algoritmus musí být přesný, srozumitelný a jednoznačný. V každém místě je jasně určeno, jaký krok bude následovat. Pro stejná vstupní data musí vycházet stále stejné výsledky.
- Algoritmus neslouží k vyřešení jen jednoho problému, ale řeší skupinu problémů, které se od sebe liší jen vstupními daty. To se nazývá hromadnost.
- Resultativnost algoritmu je, že výsledky daného problému získáme po konečném počtu kroků, tj. algoritmus nesmí být nekonečný a musí někdy skončit.

Mezi další vlastnosti patří správnost, která se těžko dokazuje. Musí se dokázat, že pro všechna data vede postup ke správnému výsledku. Problémy lze samozřejmě řešit různými způsoby, to znamená i různými algoritmu. Naší snahou je problém vyřešit co nejefektivněji. Algoritmy se mohou zapisovat slovně i graficky (např. vývojové diagramy nebo strukturogramy). (Prokop, 2012)

Algoritmizace

Algoritmizace, díky které vytvoříme algoritmus, lze rozdělit na několik částí. V první řadě je důležité správně definovat problém. Přes požadavky, výchozí hodnoty, až po požadované výsledky. Další součástí je analýza problému, při které zjistíme, zda je vůbec řešitelný a získá se tím první představa o řešení. Až teprve po definici a analýze se postoupí

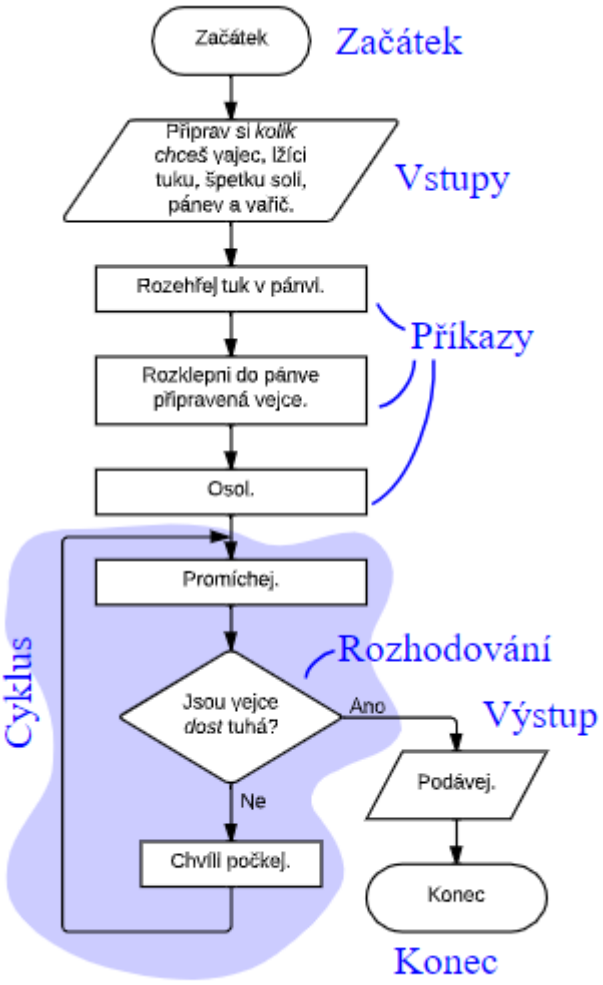
k vytvoření samotného algoritmu, který musí splňovat výše uvedená pravidla. Na základě algoritmu se sestaví program v konkrétním programovacím jazyce, který daný problém vyřeší. V poslední kroku je třeba odladit program, tedy odstranit chyby programu. Nejčastější chyby jsou v zápise, tzv. syntaktické. Závažnější jsou logické chyby, které plynou z nesprávně navrženého algoritmu. Po odstranění chyb se program konečně může využít k praktickému řešení problému.

Počítač rozumí jen algoritmům přepsaným do určitého programovacího jazyka. To se nazývá počítačový program. Sám algoritmus ale není závislý na programovacím jazyku. (Evropský sociální fond v ČR, 2008)

3.1.1 Grafický zápis algoritmů

Algoritmus můžeme zapsat jak slovně, tak graficky. Zjednodušeně je slovní zápis algoritmu klasický popis práce, nebo kuchařka. Mnohem zajímavější je grafický zápis. Jedním ze způsobů jsou vývojové diagramy. Je to grafické znázornění logické struktury řešeného úkolu. V diagramech se používá několik typů značek, z nichž každá má svůj přesný význam. Vývojové diagramy mají stejný tvar značek na začátek a konec, dále speciální tvar na vstupy a výstupy. Klasický obdélník na jednoduché příkazy. Kosočtverec jako značku rozhodování. Vše je uvedeno a popsáno na *Obrázek 1 - Vývojový diagram*. (Bayer, 2018)

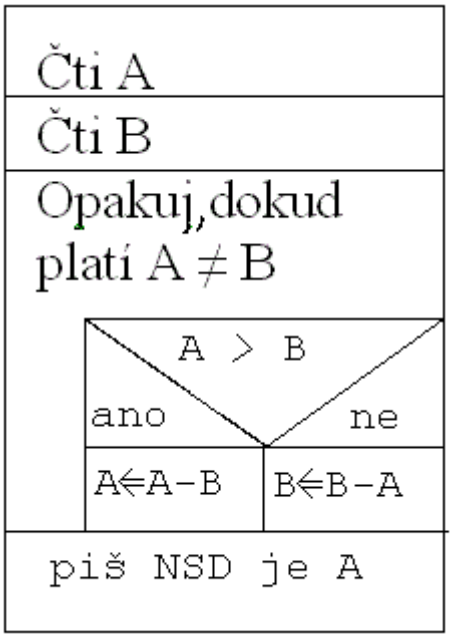
Obrázek 1 - Vývojový diagram



Zdroj: popelka.ms.mff.cuni.cz, 2019

Dalším ze způsobu znázornění algoritmů, a pro tuto práci důležitější, jsou strukturogramy. Je to kombinace grafického a textového popisu. Je tvořen obdélníkovou tabulkou, do řádku se zapisuje postup kroků symbolickou či slovní formou v pořadí, v jakém budou prováděny. První řádek tabulky obsahuje název algoritmu. (Bayer, 2018)

Obrázek 2 – Strukturogram



Zdroj: http://uzlabina2.aspone.cz/, 2019

3.2 Online nástroje pro výuku algoritmizace

3.2.1 LOGO

MIT[1] má dlouhou historii s propojením světů počítačů a dětí. Už od 70. let, kdy Seymour Papert vytvořil programovacího jazyk LOGO. Papert je matematik, zároveň dětský psycholog a díky této výhodě dokázal sestavit jazyk LOGO, který dovoluje dítěti ovládat pohyb robota pomocí příkazů pro posun a otočení (funguje na základě želví grafiky). Tyto příkazy jsou užitečné pro pochopení geometrie dítětem. Při použití v kombinaci s perem, při kreslení tvarů na obrazovce nebo na podlaze při použití pera připojeného k počítači ještě o něco lépe. Papert v roce 1980 identifikoval konstruktivistické učení, kde se dítě učí při experimentování, pozorování a účastí v procesu. V tomto případě se dítě naučí zásady geometrie tím, že si hraje s želví grafikou a vytváří různé tvary než pomocí vzorce pro výpočet úhlu potřebného k nakreslení mnohoúhelníku. LOGO se snadno naučí a Papert dokumentuje jeho použití na dětských hřištích, kde jsou příkazy LOGO vydávány slovně

[1] Massachusetts Institute of Technology

během hry, zatímco děti se pohybují kolem dětského hřiště ve variantě dětské hry "Simon Says". (Redware Research Limited, 2015)

LOGO se v posledních třech desetiletích používalo v mnoha školách, ale nemělo takový úspěch, který původně Papert a jeho kolegové předpokládali. Želví grafika je všudypřítomná ve všech školách ve Velké Británii, protože tvoří součást učebních osnov, ale často se omezuje na použití jednoduchých příkazů k přesunu jednoduchého robota a ztratila zástavu správného počítačového programovacího jazyka a sílu jazyka LOGO.

Skupina MIT pro celoživotní vzdělávání pokračovala ve vývoji LOGO, a to zejména zavedením blokového programování (ve společnosti StarLogo), která využívá grafické bloky k zastupování programových příkazů. To eliminuje chyby při psaní a syntaktické chyby. Blokové programování výrazně zvyšuje produktivitu programátorů a odstraňuje běžný zdroj frustrace pro začátečníky, což umožňuje programátorům již od 7 let začít programovat. Blokové programování se také komerčně využívá v soupravách LEGO Mindstorms Robotics, které používají děti a vzdělávací instituce po celém světě k sestavování a programování hraček robotů vyrobených z cihel LEGO. (Redware Research Limited, 2015)

Skupina MIT se již mnoho let zapojuje do provozování počítačových klubů pro mládež a zjistila, že multimediální aplikace zahrnující obrazy a zvuky jsou velmi oblíbené pro děti, které si mohou vzpomenout na své skvělé výtvory, které vytvářejí. Kombinací této zkušenosti spolu s třicetiletou účastí v počítačovém programování ve vzdělávání jsou výuková prostředí jako je Scratch, který vychází z jazyka LOGO, uchazečem o radikální změnu ICT (Informační a komunikační technologie) ve školách. (Redware Research Limited, 2015)

3.2.2 Scratch

Scratch je bezplatné vzdělávací vývojové prostředí, které vyvinula Lifelong Kindergarten Group v MIT, s více než 35 miliony registrovaných uživatelů a 37 milionů sdílených projektů. Zaměřuje se na děti ve věku 8-16 let a střední školy. I přes to ho používají všechny věkové kategorie. (Lifelong Kindergarten Group, 2019)

Scratch je navržen tak, aby byl zábavný, vzdělávací a snadno se učil. Disponuje nástroji pro vytváření interaktivních příběhů, her, umění, simulací a dalších prostřednictvím programování na základě bloků. Scratch má navíc vlastní editor barev a vestavěný editor zvuku. (Lifelong Kindergarten Group, 2019)

Uživatelé programují v aplikaci Scratch tažením bloků z palety bloků a jejich

připojením k jiným blokům, jako je skládačka. Struktury více bloků se nazývají skripty. Tato metoda programování je označována jako programování “drag-and-drop"[2]. (Lifelong Kindergarten Group, 2019)

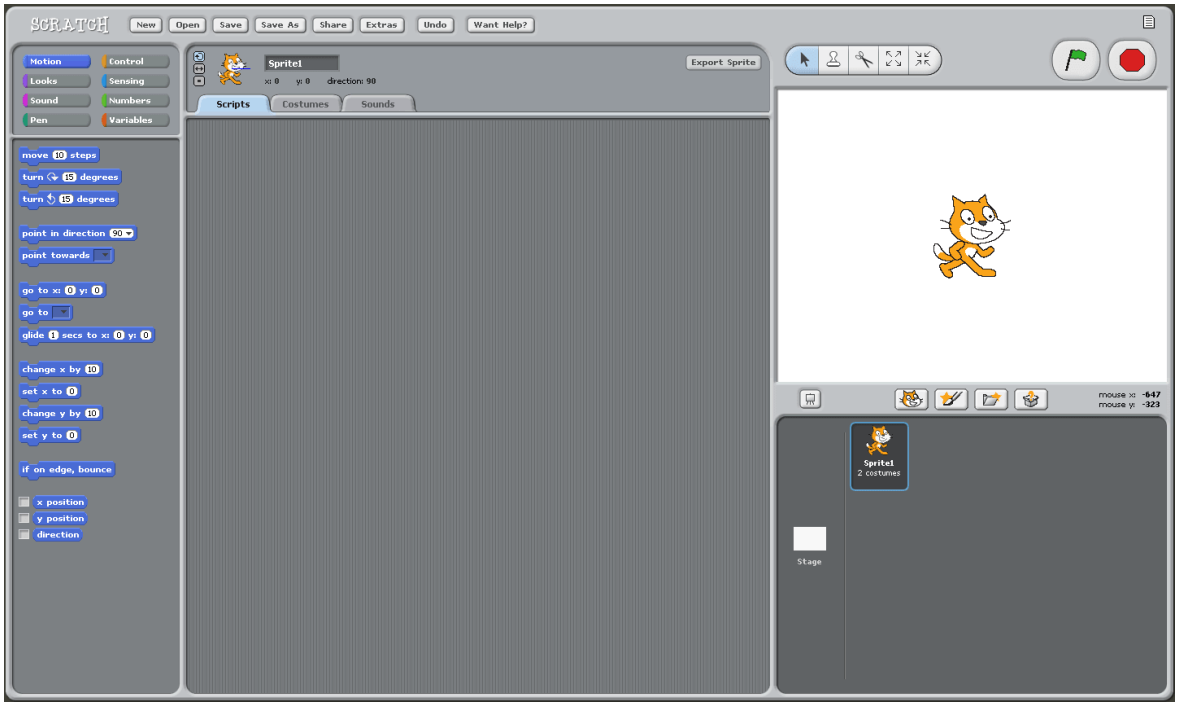
Scratch se ukázal být populární iniciativou s více než 25 000 000 studentskými projekty. Program, který se používá ve více než 150 různých zemích, je k dispozici ve více než 50 světových jazycích a může pracovat v operačních systémech Windows, Macintosh a Linus, nahradil aplikaci PowerPoint jako prezentační software v mnoha školách. Kromě podpory kódovací gramotnosti je program známý jako zábavný způsob, jak pomoci studentům řešit problémy logicky, pracovat společně a zlepšovat komunikační dovednosti. (Rouse, 2017) (Stroud, 2019)

Scratch 1.x

Scratch 1.0, vydaný 8. ledna 2007, byla první verze aplikace Scratch pro veřejnost. Prostředí Scratch 1.0 můžeme rozdělit do několika částí. Vlevo se nachází paleta bloků a jejich kategorie. Uprostřed je oblast pro scénáře postavy, kde se odehrává skládání bloků a tvoření skriptů. Nad touto oblastí je menší část pro další nastavení postavy a přepínání mezi jejími scénáři, kostýmy a zvuky. A v pravé části je scéna projektu (bílá část), kde probíhá sestavený scénář/skript, a pod ní je přehled postav projektu. (Scratchers, 2018)

[2] Táhni a pusť

Obrázek 3 - Scratch 1.0



Zdroj: en.scratch-wiki.info, 2019

Ve verzi 1.0 bylo celkem 95 bloků v 5 různých tvarech rozdělených do 8 barevně odlišených palet podle funkce. Bloky mohou být aktivovány dvojklikem. Komentáře v této verzi ještě neexistovaly. (Scratchers, 2018)

Programovacím jazykem byl Squeak, což je objektově orientovaný programovací jazyk. Squeak je implementace Smalltalku a je odvozen přímo od verze Smalltalk-80. To je první vydaná verze (v roce 1972) Smalltalku vůbec. Squeak byl používán od verze 1.0 až po verzi 1.4. (Scratchers, 2019)

Verze 1.1 vydaná 5 měsíců po první verzi nepřinesla hodně vylepšení, ale pár přeci. Nový blok *Repeat Until ()*, který je velmi často používaný v projektech. Byl představen tvar *Cap Block*[3], který se nachází pod paletou *Ovládání*. Přibyla nová funkce importovat *Postavy*. A další menší změny v uživatelském rozhraní. (Scratchers, 2018) (Scratchers, 2018)

Verze 1.2 přidala blok *hlasitost*, který spadá pod paletu *Zvuku*, a přidala možnost nastavit hlasitost zvuku, který lze zobrazit i na scéně. Pod *Zvuk* spadá také blok *Tempo*, kterým nastavujeme rychlost přehrávání zvuku a také lze zobrazit na scéně. Další změnou bylo přidání možnosti měnit vzhled postav pomocí *Kostýmů*. Dále *() of ()* (blok pod

[3] Konečný blok (např. konec skriptu)

Operátory) provádí zadanou funkci na daném čísle. Funkce lze vybrat z rozevírací nabídky. Několik bloků bylo přejmenováno. (Scratchers, 2018) (Scratchers, 2018)

Verze 1.3 přinesla *seznamy*, také bloky pro ukázání a skrytí proměnné. C bloky (tvar písmena C), do kterých se vnořují další bloky. Jsou to zejména podmínky a cykly. *Poznámky*, které byly ovšem následně odstraněny. Možnosti komentování buď u každého bloku nebo plovoucí ve scénáři. (Scratchers, 2018) (Scratchers, 2018)

Poslední verzí z 1.x série byla 1.4 a přinesla největší změnu uživatelského rozhraní oproti přechozím verzím. Dále dotazování a odpovědi, *spoj () ()*, *písmeno () z ()*, *délka ()*, *() obsahuje ()?*, vše z palety *Operátorů* (přejmenované z *Čísel*). Nově byla možnost stáhnout off-line verzi na Ubuntu/Debian. Verze 1.0 až 1.3 byly kompatibilní pouze s operačními systémy Windows a Mac OS X. (Scratchers, 2018)

Scratch 2.0

Scratch 2.0, také známý jako Scratch 2, byl druhou hlavní verzí Scratch, po Scratch 1.4. Zahrnuje redesignovaný editor a webové stránky a byla to první verze, která zahrnovala editor online i off-line. Online verze byla oficiálně vydána veřejnosti 9. května 2013. (Scratchers, 2019)

Scratch 2.0 oznámil uživatel „andresmh“ na Scratch fóru v lednu 2010. První uvolněný experiment týmu Scratch v rámci vývoje 2.0 byl Experimental Viewer v srpnu 2010. Později v roce 2011 byl uvolněn beta Flash Player, díky němu si přihlášení uživatelé mohli vybírat z různých projektů, na kterých chtějí pracovat. V roce 2012 byl nahrazen alfa verzí editoru. Tato verze byla pro uživatele stanovena jako výchozí v říjnu 2012.

V květnu 2011 byla na Scratch Day v MIT vydána první známá verze editoru projektu, prealpha, ale jen pro omezené publikum. Po chvíli začala společnost Scratch zveřejňovat na svém blogu aktualizace, nazvané Scratch 2.0 Progress Reports. Nová webová stránka a přepracovaný editor, byly dokonce přístupné několik dní veřejnosti na Scratch Day 2012. Po zbytek roku 2012 byli lidé vyzváni k testování této verze. Mezi testeři byli moderátoři komunity, vybraní pedagogové, bývalí kurátoři a současní ze Scratch Design Studio, moderátoři TBG (Text based games) a skupina 500 dobrovolníků. Někteří uživatelé byli také schopni infiltrovat a používat program kvůli závadě. (Scratchers, 2019)

V prosinci roku 2012 byla veřejná beta verze oznámena na začátek 28. ledna 2013. Tato verze byla k dispozici na beta.scratch.mit.edu od té doby až do vydání alfa verze.

Offline editor Scratch 2.0 byl vydán 26. srpna 2013. Měl málo rozdílů od editoru online. Zejména pak pozoruhodně málo místa v takzvaném batohu, který sloužil uživatelům k přetahování kostýmů, zvuků a části skriptů z jiných projektů. A také použití odlišných barev, když je text označený.

13. května 2014 byl uvolněn zdrojový kód pro aplikaci Scratch 2.0. Je k dispozici na GitHubu. (Scratchers, 2019)

Scratch 3.0

Novinkou roku 2019 se stal Scratch 3.0, vydaný 2. ledna 2019. Současně s online editorem byl vydán i nový off-line editor. Beta verze programovacího editoru Scratch 3.0 běžela na stránkách (beta.scratch.mit.edu) už od srpna 2018 na stránkách. Nová verze s sebou přináší spoustu změn. Má více způsobů, jak vytvářet a sdílet projekty, lepší podporu pro nově příchozí programátory. Zároveň lze pokračovat v projektech z minulých verzí. K již existujícím blokům se přidalo pár nových. Vývojové prostředí bylo přeloženo do dalších jazyků a stále zůstalo volně přístupné pro všechny. (The Scratch Team, 2018)

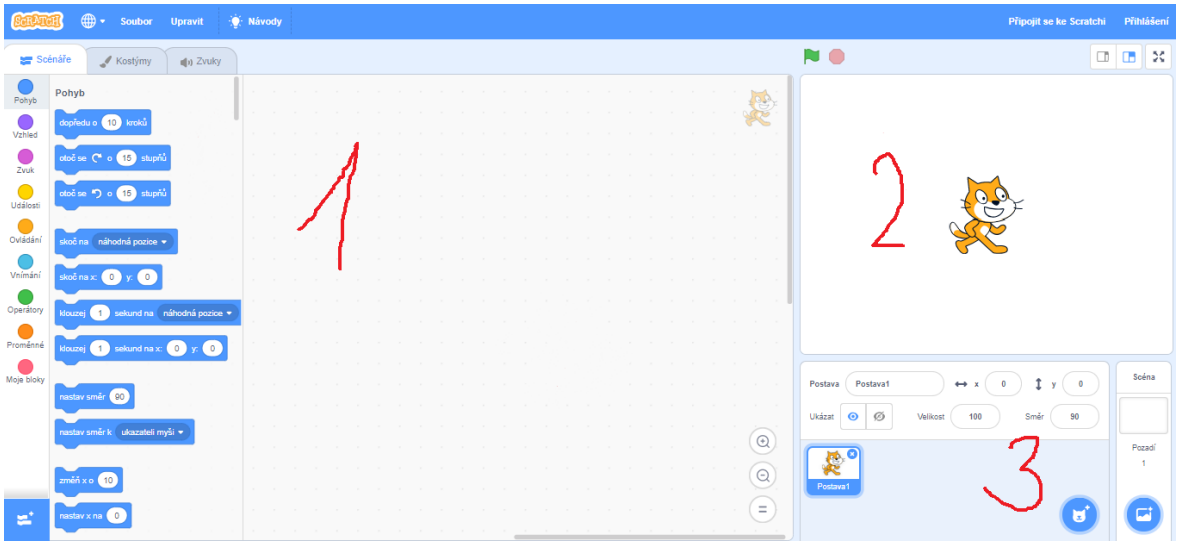
Vývojové prostředí

Vývojové prostředí Scratch 3.0 je rozděleno do 3 částí:

1. Scénáře – Bloky rozdělené podle kategorií.
   Kostýmy – Úprava kostýmů postav.
   Zvuky – Práce se zvukem.
2. Scéna projektu
3. Přehled postav projektu a pozadí scény

(Lifelong Kindergarten Group, 2019)

Obrázek 4 - Scratch 3.0 vývojové prostředí



Zdroj: scratch.mit.edu/projects/editor, 2019 dle autora

Bloky

Bloky jsou rozdělené do 9 kategorií: Pohyb, Vzhled, Zvuk, Události, Ovládání, Vnímání, Operátory, Proměnné a Moje bloky (pro nové vytvořené bloky programátorem). Pro další lepší orientaci jsou i barevně odlišeny. (Lifelong Kindergarten Group, 2019)

Obrázek 5 - Scratch kategorie bloků

Zdroj: scratch.mit.edu/projects/editor, 2019

Programovací jazyk

Celé vývojové prostředí je přepracované od verze 2.0 a napsané v HTML5 a JavaScriptu. (Scratchers, 2019)

Off-line verze

Pokud nemá někdo stálý přístup k internetu, může využít nový Scratch off-line editor 3.0, který byl vydán v lednu 2019. Lze stáhnout na oficiálních stránkách. Je podporován pro operační systémy Windows 10 a novější, macOS 10.13 a novější. V současné době není podporována verze na operační systém Linux. (Lifelong Kindergarten Group, 2019)

Jazyk rozhraní

V současné době je verze dostupná v 54 různých světových jazycích, včetně angličtiny a češtiny. (Lifelong Kindergarten Group, 2019)

Práce s projekty

Je několik cest, jak pracovat s projekty. Svůj projekt může programátor uložit do svého počítače, který se uloží souborem s koncovkou *.sb3*. Ze souboru v počítači lze projekt opětovně nahrát do editoru a pokračovat v práci. Další cestou, jak uložit projekt, je vytvořit si účet na Scratch. Projekty se budou automaticky ukládat pod účtem. Vytvořené projekty se mohou ukládat jako nesdílené a nikdo další je neuvidí, nebo naopak se dají sdílet pro celou komunitu Scratch. Další programátoři mohou projekt komentovat, nebo využít ve své práci, tím vznikají takzvané remixy projektů. Pro ukládání a sdílení svých projektů je nutné vytvořit si účet. Pro práce se sdílenými projekty od jiných programátorů účet není potřeba. (Lifelong Kindergarten Group, 2019)

Selekce a iterace

Scratch 3.0 nabízí programátorům 3 cykly (iterace) a 2 selekce: repeat počet-krát, forever, if (), if () else a repeat until ()[4]. (Lifelong Kindergarten Group, 2019)

[4] () představuje podmínku

Obrázek 6 - Scratch 3.0 podmínky a cykly

Zdroj: scratch.mit.edu/projects/editor, 2019

Proměnné

Nástroj rozlišuje globální a lokální proměnné (pro všechny postavy / pro aktuální postavu). Programátor může vytvářet proměnné dle potřeby. Proměnné lze nechat vypsat na scéně a sledovat jejich aktuální stav. (Lifelong Kindergarten Group, 2019)

Datové typy

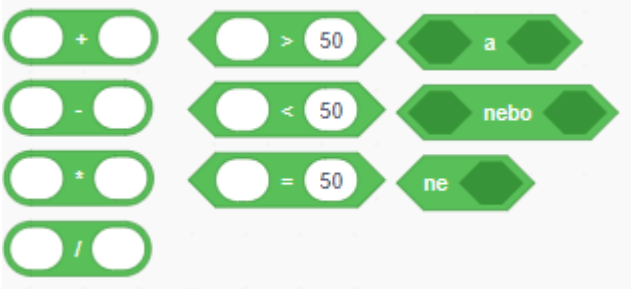
Kromě základních datových typů jako jsou číslo a text umožňuje práce se seznamy jako s poli (array). Obsah seznamů lze nechat vypsat na scéně, a rozdělují se opět na globální a lokální (pro všechny postavy / pro aktuální postavu). (Lifelong Kindergarten Group, 2019)

Logika a operátory

Nástroj umožňuje do podmínek skládat složitější výrazy. Rozumí logickým spojkám AND a OR. Podporuje také negaci, větší, menší nebo rovná se. Tyto logické spojky lze skládat do větších a složitějších podmínek.

Existují funkce pro součet, rozdíl, násobení a dělení. (Lifelong Kindergarten Group, 2019)

Obrázek 7 - Scratch 3.0 operátory

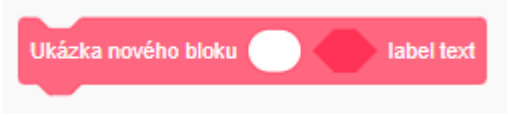


Zdroj: scratch.mit.edu/projects/editor, 2019

Tvorba nových bloků

Pro tvořivé programátory je tu funkce vytvoření nových bloků podle potřeby. Blok má svůj název a je možné přidat potřebné množství polí pro funkce, podmínek nebo textu. (Lifelong Kindergarten Group, 2019)

Obrázek 8 - Scratch 3.0 nový blok



Zdroj: scratch.mit.edu/projects/editor, 2019

3.2.3 Snap!

Snap je vizuální, drag-and-drop programovací jazyk. Jde o rozsáhlou reimplementaci projektu Scratch s postupy první třídy, seznamy první třídy a postavy první třídy s dědictvím. Tyto přidané schopnosti dělají nástroj vhodný pro úvod do informatiky pro střední nebo vysoké školy. Starší verze BYOB (Build Your Own Blocks) byla pouze modifikací zdrojového kódu Scratch 1.4, ale od verze Snap 4.0 je to zcela samostatný nástroj, ikdyž jeho uživatelské rozhraní vychází z nástroje Scratch a má tedy mnohé podobnosti.

První verze BYOB 1.0 byla vydána 21. října 2008 a vycházela ze základů Scratch 1.3 a navíc měla další funkce. Hlavní záměr vytvořit modifikaci, bylo omezeným množství funkcí v nástroji Scratch. Jako hlavní odlišnost byla možnost vytváření vlastních bloků v BYOB, která se odráží v názvu. (LKG, 2019)

BYOB 2.0 byl vydán 30. srpna 2009, aktualizoval Scratch na verzi 1.4, přidal několik vylepšení UI, včetně možnosti *Zpět*. Hlavní rys přidaný v BYOB 2.0 byl možnost dělat sbírky

postav, které se mohou pohybovat spolu. Příkladem může být kostra, ve které jsou kosti propojené, celá kostra se může otáčet společně, nebo se může otáčet určitá kost vzhledem k celku. 25. dubna 2010 byla vydána verze 3.0, která zavedla postupy a seznamy první třídy. Poslední z verzí modifikací byla verze 3.1 vydána 18. května 2011 s dalšími změnami jako byly prvotřídní postavy a dědictví. (LKG, 2019)

Snap! 4.0

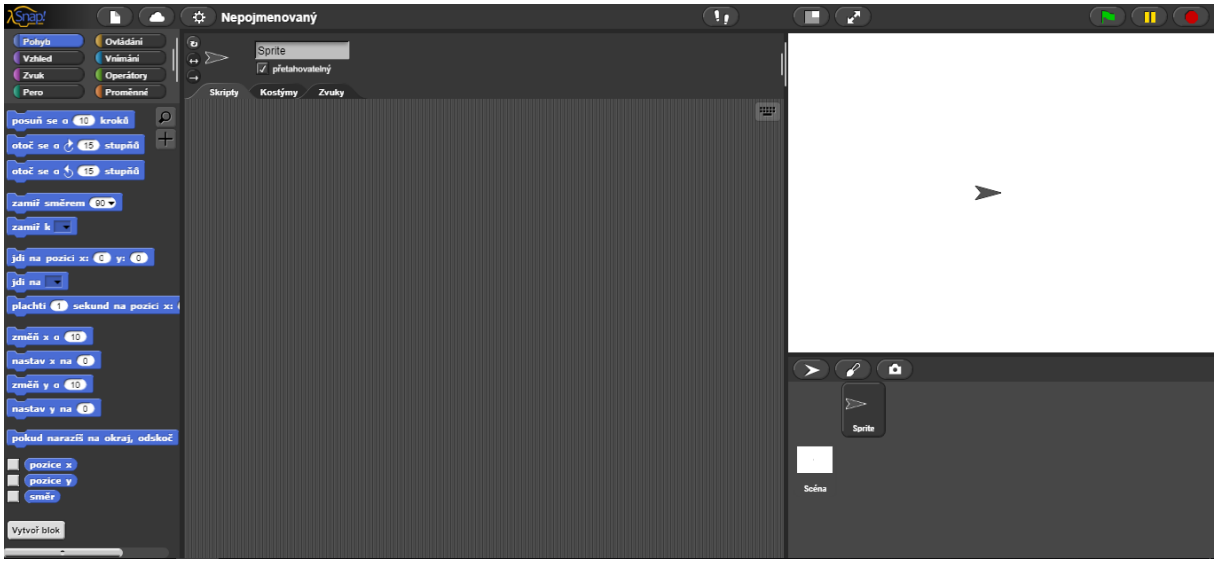
Důležitým milníkem byl datum 1. května 2015, kdy byl vydaný Snap 4.0 (nástupce BYOB). Byla to první verze, která byla od nástroje Scratch zcela oddělena. Výukové prostředí bylo kompletně přepsání v JavaScriptu. Vývoj začal už v roce 2011 (zhruba ve stejnou dobu jako vývoj Scratch 2.0). Snap začínal s nulovým rozpočtem a pouze jedním vývojářem. Byla to také první verze, která běžela na webovém prohlížeči, a dokonce měla české uživatelské rozhraní. Název byl změněn z BYOB na Snap hlavně kvůli stížnostem a nevhodnost názvu od učitelů programování. Nutno podotknout, že lidé mimo USA nechápou, proč by tato hříčka mohla být považována za nevhodnou. Tým Snap neochotně hledal náhradní název, který ještě nebyl názvem existujícího programovacího jazyka. Název Snap! se původně nikomu nelíbil, ale byla to menší nápověda k funkčnosti (Snap – bloky secvaknou k sobě). (Monig, a další, 2018)

Další vývoj nástroje byl v roce 2017 s verzí 4.1. Byly přidány postavy první třídy s dědictvím, také prvotřídní zvuky a kostýmy. V roce 2018 4.2 pak novinka automatického ukládání projektů v cloudu (registrovaným uživatelům). (Monig, a další, 2018)

Vývojové prostředí

Vývojové prostředí má velmi podobné rozložení jako prostředí Scratch. Nalevo se nachází kategorie bloků, mezi kterými lze přepínat. Uprostřed prostor pro psaní algoritmů. Programátor si zde pak může pomocí záložek přepínat mezi scénářem skriptům, úpravou kostýmů nebo úpravou zvuků. Napravo je nachází samotná scéna, na které probíhá vytvoření algoritmus. Pod scénou lze pracovat s postavami (přidávat nové, upravovat apod.) a se scénou. (Monig, a další, 2018)

Obrázek 9 - Snap! vývojové prostředí



Zdroj: snap.berkeley.edu/snapsource/snap.html, 2019

Programovací jazyk

Od verze 4.0 je Snap kompletně přepsané do JavaScriptu. Pro scénu a vše s ní spojené používá canvas (plátno) z HTML5. (LKG, 2019)

Off-line verze

Snap nemá svou off-line desktop verzi. Pro programátory bez stálého připojení k internetu je tu však možnost stáhnout si zdrojové kódy (nabídka přímo v prostředí). Není třeba žádná instalace. Prostředí běží stále v prohlížeči, ale nemá přístup k oficiálním Snap serverům a nejsou přístupné některé knihovny. (LKG, 2019)

Bloky

Bloky jsou rozdělené do 8 kategorií: Pohyb, Vzhled, Zvuk, Pero, Ovládání, Vnímání, Operátory, Proměnné. Pro lepší orientaci jsou i barevně odlišeny. Při vytváření nového bloku jsou vidět další 2 kategorie, kam nový blok zařadit a to jsou: Seznamy a Ostatní. Celkem tedy 10 kategorií. (Monig, a další, 2018)

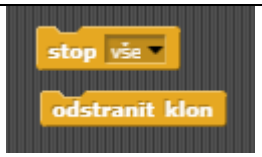
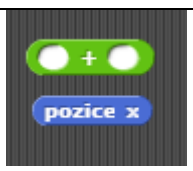
Obrázek 10 - Snap rozdělení bloků

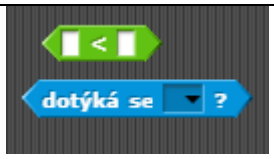
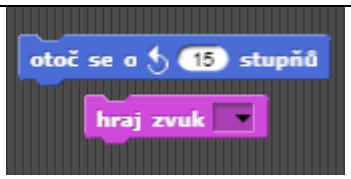
Zdroj: snap.berkeley.edu/snapsource/snap.html, 2019

Tvary a typy bloků

Kromě odlišných barev si lze všimnout odlišností ve tvarech bloků. Na první pohled lze rozlišit 5 různých tvarů. Počáteční blok, který má pacičku na skládání jen pod sebou, a naopak konečný blok, do kterého bude vést pacička seshora a dále nebude algoritmus pokračovat. Kromě těchto dvou základních pak lze rozlišit bloky funkcí, které se vyznačují kulatými kraji. Funkce slouží k výpočtu, nebo přijmutí různých vstupů a vrácení jedné hodnoty. D lze rozlišit blok podmínky se špičatými kraji. Podmínky zásadně vrací hodnotu *true* nebo *false*. Tyto bloky se používají hlavně v selekci a iteraci, kdy se vkládají do podmínky. A posledním a nejčastějšími bloky jsou příkazové bloky. Specifické jsou svým tvarem, kdy pacička směřuje dovnitř i dále ven. Při vstup do tohoto bloku se příkaz vždy vykoná a poté algoritmus pokračuje dále. Pro ilustraci je několik příkladů uvedeno v následující tabulce. (Monig, a další, 2018)

Tabulka 1 - Snap tvary bloků

Blok začátku	
Blok konce	
Blok funkce	

Bloku podmínky	
Blok příkazu	

Zdroj: snap.berkeley.edu/snapsource/snap.html, 2019

Selekce a iterace

Snap 4.0 nabízí programátorům možnost vybrání ze 3 cyklů a 2 selekcí. (Monig, a další, 2018)

Obrázek 11 - Snap selekce a iterace

Zdroj: snap.berkeley.edu/snapsource/snap.html, 2019

Datové typy

Snap rozlišuje celkem 7 datových typů: číslo, text, pravdivostní hodnotu, seznam, postavu, kostým, zvuk. Další 3 jsou spíše typy bloků, tedy: blok příkazů, blok funkcí, podmínky. Snap umožnuje v podmínce, která zpátky vrátí hodnotu *true* nebo *false*, porovnat, jestli je zadaný výraz (případně blok) vybraného datového typu. (LKG, 2019)

Obrázek 12 - Snap datové typy

Zdroj: snap.berkeley.edu/snapsource/snap.html, 2019

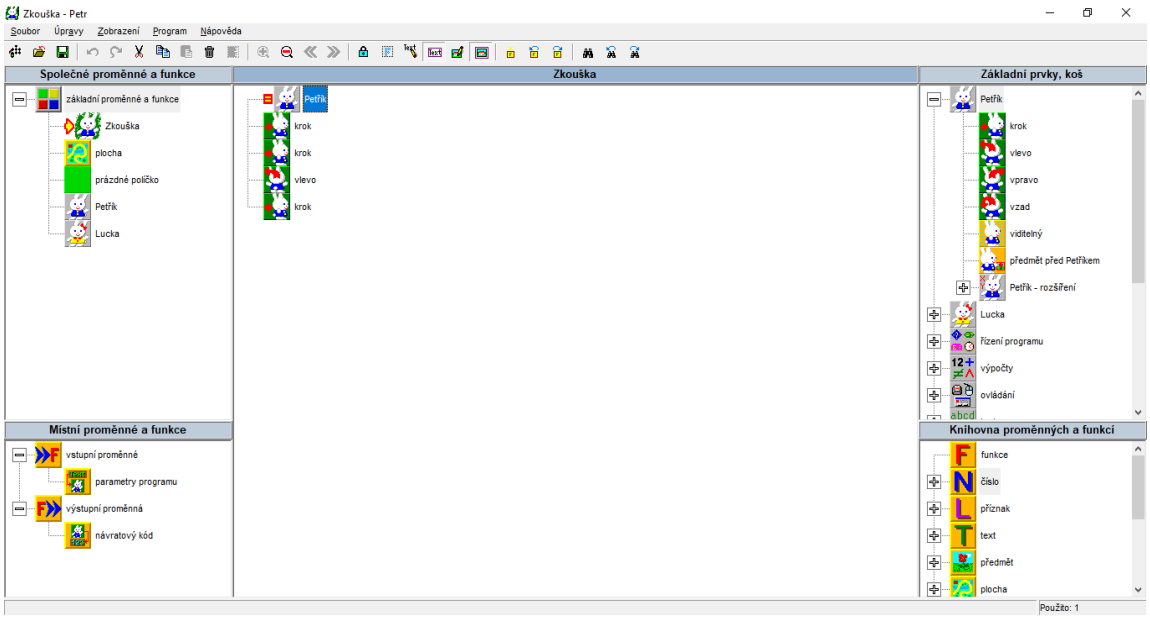
3.2.4 Petr

Gemtree Petr je vizuální programovací nástroj určený k snadné a rychlé tvorbě programů pro Windows 95/98/NT/ME/2000/XP/Vista. Jeho hlavní charakteristikou je grafické vyjádření struktury programu. Části programu se skládají pomocí myši k sobě jako skládačka. Díky kontrole smysluplnosti kombinací prvků již při vytváření programu odpadá jakákoliv možnost vzniku syntaktické chyby. Vyjádření programu stromovou strukturou přináší radikální zvýšení přehlednosti programu. Tvorba programu se stává nebývale snadnou, přehlednou a pružnou. (Němeček, 2013)

Gemtree Petr je určen pro nejširší okruh uživatelů. Již děti předškolního věku si v něm kreslí obrázky a učí chodit svého králíčka. Pro zkušené programátory je tu naopak připraveno velké množství bohatých funkcí. Petr si neklade za cíl vychovávat ze všech uživatelů programátory. Snaží se zpřístupnit tvorbu programů každému, kdo chce tvořit. Nejsou zapotřebí žádné znalosti z oblasti programování ani z oblasti počítačů. Vše je zajištěno nepřeberným množstvím funkcí. Je to účinný prostředek k procvičování logického myšlení, představivosti a estetického cítění. (Němeček, 2013)

Náhled do vývojového prostředí Petr. (Výuková prostředí běžící online na webu začínají být standardem. Bohužel tento výukový nástroj neběží na webu. Byl vybrán autorem práce jako ukázka výukových prostředí, které jsou pouze off-line. Proto nástroj Petr nebude hodnocen ve vlastní práci této práce, nesplňuje kritérium „online nástroj“. pozn. autora)

Obrázek 13 - Petr vývojové prostředí



Zdroj: nástroj Petr dle autora, 2018

3.2.5 Robot Karel

Vývoj

V sedmdesátých letech minulého století se student Stanfordské univerzity Richard Pattis rozhodl, že by bylo mnohem jednodušší učit základy programování, pokud by se mohli studenti nějakým způsobem naučit základní myšlenky v jednoduchém prostředí bez složitostí vycházejících z programovacích jazyků. Nechal inspirovat z úspěchu Seymora Paperta z MIT a jeho jazyku LOGO. Pattis vytvořil základní programovací prostředí, ve kterém se studenti učí robota řešit jednoduché úkoly. Robot byl pojmenován Karel, po českém dramatikovi Karlu Čapkovi, jehož hra R.U.R., vydaná v roce 1923, dala světu slovo Robot. (Roberts, 2005) (Jedlička, 2013)

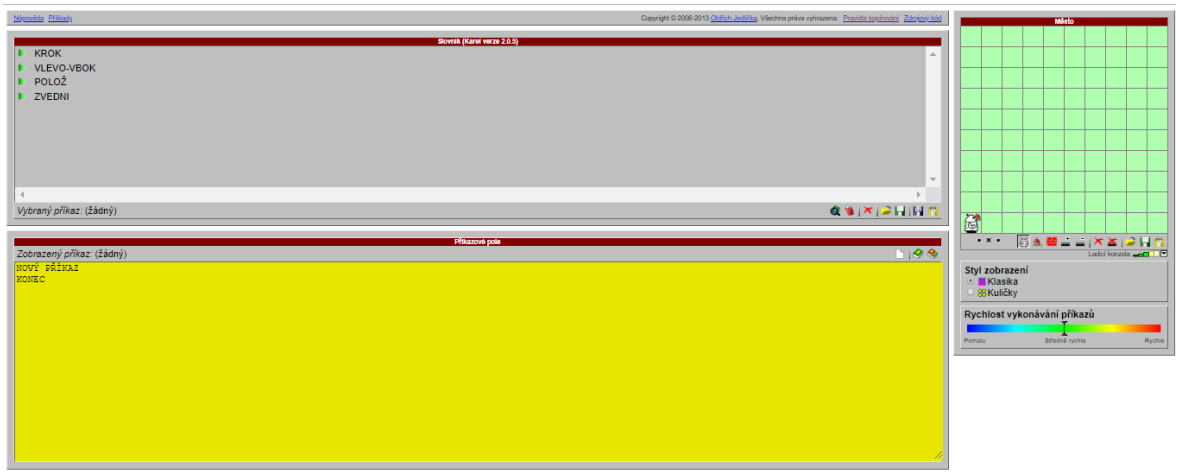
Robot Karel měl velký úspěch. Byl používán v kurzech úvodní informatiky v celém USA. Učebnicí od Riche se prodalo více než 100 000 kopií. Mnoho studentů se naučilo základy programování právě od Karla. Nic ale netrvá věčně. V polovině devadesátých let simulátor, který byl používán pro Robota Karla, přestal pracovat. Brzy se podařilo Karla opět zprovoznit a uvést v nové verzi. Ovšem hned rok poté byla vydána Java, programování se začalo vyučovat právě v ní a Karel opět zmizel ze scény. Poslední implementace Karla byla navržena tak, aby byla kompatibilní jak s Javou, tak s programovacím prostředí Eclipse. (Roberts, 2005)

V České republice se poprvé objevil zásluhou docenta Hvoreckého, Csc. Na Karlovi pracovalo spousta lidí, zejména pak Ing. Tomáš Bartovský, Csc. a Ing. Rudolf Pecinovský, CSc., kteří zjednodušili pravidla používání a přiblížili tak Robota Karla dětem. Na rozdíl od původního amerického Karla, byla také přidána rekurze. (Jedlička, 2013)

O současný stav Robota Karla se zasloužil tehdy vysokoškolský student Oldřich Jedlička, který projekt v roce 1999 opět oživil, v roce 2001 připravil pro internet a o rok později poprvé dostal na webové stránky. Nové zpracování, které známe do dnes, bylo vydáno 2006. (Jedlička, 2013)

Vývojové prostředí

Obrázek 14 - Vývojové prostředí robota Karla



Zdroj: karel.oldium.net, 2019

Vývojové prostředí je rozděleno do 3 částí: Slovník, Příkazové pole a Město. Prostředí je současně ve verzi 2.0.5. (Jedlička, 2013)

Město

Město robota Karla v pravé části prostředí je pole, kde se vykonávají příkazy. Skládá se ze šachovnice 10x10. Robot začíná v levém spodním rohu se souřadnicí 1x1, kde má i svůj domov, označený obrázkem domu, tj. startovné políčko. Robot se může pohybovat pouze po hranách a nemůže vyjít mimo město. Do města lze postavit zdi, které robot nedokáže přelézt. Pod městem je řádek s tlačítky na úpravu města a dalšího nastavení. (Jedlička, 2013)

Obrázek 15 - Město – funkce

Zdroj: karel.oldium.net, 2019

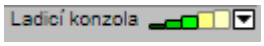
První funkce s obrázkem robota je Robot Karel, s kterou lze Karla přemístit na libovolné místo ve městě (pokud tam není zeď). Dvojklikem je Karla možné otočit od 90

stupňů vlevo. Obrázek domečku představuje změnu Karlova domova. Opět lze domov umístit libovolně s výjimkou zdi. Tlačítkem cihliček lze stavět ve městě zdi. Zeď nelze postavit na Karla ani na jeho domov. Následují 2 podobná tlačítka. Tlačítko Polož značku (s plusem) položí na políčko značku a Zvedni značku (s mínusem) zvedne značku z políčka. Maximální počet značek na políčko je 8. Z prvních 5 funkcí je možné mít aktivní pouze jedno, které je pak od ostatních odlišeno. Z obrázku je vidět, že je právě aktivní čtvrté tlačítko, tedy Polož.

Další 2 tlačítka interagují s prvními pěti. Červené křížek značí Vyčisti město, což znamená, že vyčistí celé město od značek a zdí, a zároveň umístí Karla a jeho Domov na počáteční pole, tedy 1x1. Červený křížek se značkami (Posbírej značka) odstraní pouze značky z celého města.

Poslední 3 funkce jsou pak na pracování s celými hotovými městy. Načti město načte město skrze řádek textu, pod kterým lze město uložit. A to lze právě udělat tlačítkem Ulož město, který vygeneruje řádek, který si musí uživatel uložit a dále neměnit, aby ho mohl opět načíst. Poslední funkcí z řádku je Exportuj město, která umí město změnit do textu pro použití v programu Visual Karel 99. Jsou zde napsané souřadnice Karla a jeho domovu, a kam je otočený. Město je definované symboly X, které označují zeď, čísly, které značí počet položených značek a tečky značí prázdná pole. (Jedlička, 2013)

Obrázek 16 - Město – Ladící konzole

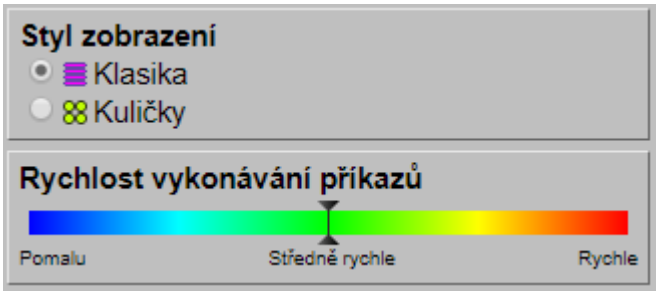


Zdroj: karel.oldium.net, 2019

Další funkcí robota Karla je ladící konzole, která vypisuje informace o průběhu programu. Má celkem 5 stupňů:

- Úroveň chyb – vypisují se pouze chyby
- Úroveň varování – vypisují se chyby a varování
- Úroveň informací – vypisují se chyby, varování a informativní zprávy
- Úroveň detailů – vypisují se chyby, varování, informativní a detailní zprávy
- Úroveň pro vývojáře – vypisuje všechny informace pro vývojáře
  (Jedlička, 2013)

Obrázek 17 - Město – Styl zobrazení a rychlost příkazů



Zdroj: karel.oldium.net, 2019

Mezi poslední nástroje města patří styl zobrazení značek. Uživatel si může vybrat mezi zobrazením značek klasicky nebo jako kuličky. Následně pak může upravit rychlost, jakou se budou vykonávat příkazy. (Jedlička, 2013)

Slovník

V slovníku jsou uloženy všechny příkazy, které Karel umí. Zobrazeny nejsou všechny, pouze ty, které Karel může vykonat okamžitě. Při prvním spuštění jsou zobrazeny pouze příkazy: krok, vlevo-vbok, polož, zvedni. Tyto příkazy se nazývají systémové. Slovník se bude rozšiřovat, podle toho, jak Karla bude uživatel učit novým a novým příkazům, které bude přidávat přes příkazové pole. Vedle od příkazů se pak nachází zelený trojúhelníček, který umožňuje příkaz spustit, nebo červený křížek, který zastaví právě běžící příkaz.

I pod slovníkem, tak jako pod městem, máme řadu tlačítek. Kromě tlačítek na ukládání a načítání slovníku, které pracují na stejný způsob, jako tlačítka pod městem, jsou zde 3 nová tlačítka. (Jedlička, 2013)

Obrázek 18 – Slovník – funkce

Zdroj: karel.oldium.net, 2019

První tlačítko Ukaž příkaz, zobrazí příkaz ze slovníku v příkazovém poli. Další tlačítkem Smaž příkaz, můžeme příkazy mazat z paměti. A červené tlačítko vymaže všechny naučené příkazy, to znamená, že ve slovníku zůstanou jen systémové příkazy. (Jedlička, 2013)

Příkazové pole

V příkazovém poli uživatel píše nové příkazy. První řádek je vyhrazen pro název příkazu, pod kterým bude uložen ve slovníku. Každý příkaz musí někdy končit, proto na poslední řádku musí být slovo „konec“. Příkazový řádek umí pracovat jak s velkými, tak malými písmeny, ovšem při ukládání převede vše na velká písmena. S háčky a čárky si také umí poradit. (Jedlička, 2013)

Obrázek 19 - Příkazové pole – funkce

Zdroj: karel.oldium.net, 2019

Nový příkaz se vytvoří tlačítkem nový příkaz, následně se zapíše požadovaný algoritmus do příkazového pole a uložení do slovníku se provede tlačítkem ulož příkaz. Při ukládání je důležité mít unikátní název příkazy. Pokud už je ve slovníku příkaz se stejným názvem, nové uložený příkaz přepíše původní. Pokud se prochází otevřeným příkazem ze slovníku, při změně názvu se i ve slovníku změní jen název a je uložen na stejném řádku. Pokud je nutné příkaz uložit jako nový, lze použít tlačítko ulož příkaz nově, nesmí se však zapomenout na změnu názvu. (Jedlička, 2013)

Funkce

Systémové příkazy

Karel zná 4 základní systémové příkazy, kterými ho uživatel může vést městem. KROK – Karel udělá jeden krok vpřed, podle toho, kam je otočený. Pokud krok nelze provést, Karel na to upozorní (například pokud je před ním zeď). VLEVO-VBOK – Karla otočí o 90 stupňů proti směru hodinových ručiček. Příkaz vpravo-vbok není v systémových příkazech, z toho plyne, že pokud uživatel chce, aby se Karel otočil o 90 stupňů vpravo, musí se použít příkaz vlevo-vbok třikrát. Aby si uživatel ušetřil čas, může si tak jednoduše uložit nový příkaz vpravo-vbok do slovníku, ve kterém bude právě třikrát vlevo-vbok. Karel se může otočit pouze do 4 směrů: sever, jih, východ, západ. Dalšími příkazy jsou POLOŽ – Karel položí ze svého kouzelného batohu, do kterého se vejde nekonečně značek a z kterého se může brát nekonečně značit (v reálu jen 10x10x8), značku na políčko. A ZVEDNI, Karel uloží značku z políčka do svého batohu. (Jedlička, 2013)

Komentáře

Do příkazů je možné psát komentáře. Zapisují se za středník a ukládají se společně s příkazy. Komentář se většinou píše na samostatnou řádku. Pokud se nachází za příkazem, je při ukládání umístěn také na samostatný řádek. (Jedlička, 2013)

Podmínky a cykly

Mimo základních příkazů, zná Karel i podmínky, které jsou jedním ze základních kamenů algoritmizace a programování. Jsou to:

- KDYŽ (podmínka)
  KONEC, JINAK[5]
- DOKUD (podmínka)[6]
- OPAKUJ počet[7]

Za podmínku DOKUD a za cyklus OPAKUJ, lze také použít podmínku AŽ (podmínka), která dokáže cyklus předčasně ukončit. Cyklus se vždy provede alespoň jednou.

Samotná podmínka je složená z jednoho, případně dvou slov. Může to být spojení slova JE, které je nepovinné, nebo NENÍ, a jednoho dalšího slova:

- ZEĎ – jestli JE/NENÍ před Karlem zeď, podmínka je splněna
- ZNAČKA – jestli JE/NENÍ na políčku, kde je Karel, značka, podmínka je splněna
- DOMOV – jestli JE/NENÍ na políčku, kde je Karel, domov, podmínka je splněna
- VÝCHOD – jestli JE/NENÍ Karel natočen na východ, podmínka je splněna
- SEVER – jestli JE/NENÍ Karel natočen na sever, podmínka je splněna
- ZÁPAD – jestli JE/NENÍ Karel natočen na západ, podmínka je splněna
- JIH – jestli JE/NENÍ Karel natočen na jih, podmínka je splněna

Karel také zvládá rekurzi, což je mocný nástroj. Je to použití příkazu, který právě probíhá, v těle tohoto příkazu. Tedy pokud je příkaz použit sám v sobě. Důležité je, aby rekurze měla podmínku, která cyklus ukončí. (Jedlička, 2013)

[5] Podmínka IF
[6] Cyklus WHILE
[7] Cyklus FOR

Ostatní

Při psaní příkazů lze také použít slovo RYCHLE. Karel pak následující příkaz provede co nejrychleji. Pokusí se neudělat žádnou viditelnou pauzu. Ukončuje se pak slovem KONEC, nebo POMALU, kdy následující příkazy poběží opět nastavenou rychlostí.

Slovem STOP Karel zastaví svou činnost a čeká na nové příkazy. (Jedlička, 2013)

4 Vlastní práce

Pro porovnání byla vybrána vývojová prostředí Scratch, Snap a robot Karel. Vývojové prostředí Petr je nedílnou součásti této tématiky, kriteriálně ale nesplňuje požadavky (hodnotí se pouze nástroje, které jsou přístupné online) a výsledky by byly zkreslené.

Po analýze vybraných výukových nástrojů z odborných zdrojů v teoretické části se může předpokládat, že vývojová prostředí Scratch a Snap mohou ve srovnání s robotem Karlem vyjít lépe. Mají nesrovnatelně více možností, jak v nástroji pracovat.

Praktická část proběhla následovně:

- Definování verzí nástrojů.
- Vybrání vhodných kritérií.
- Určené preference kritérií.
- Vytvoření tabulky na základě poznatků z teoretické části.
- Převedení tabulky podle metod na číselná ohodnocení.
- Výpočet vah kritérii pomocí VAV (Fullerův trojúhelník).
- Výpočet kompromisní varianty pomocí VAV (metoda pořadí).

Kritéria byla vybrána na základě shodných požadavků odborné literatury pro výuku algoritmizace a programování: *C# bez předchozích znalostí* od Pavla Boryho, *Programming pearls* od Jona Bentleyho, *Programování pro úplné začátečníky* od Radka Hylmara a *Beginning programming all-in-one desk reference for dummies* od Wallace Wanga. Výukovým prostředí se tak docílí zjednodušení složitostí, které vystupují s učením syntaxí programovacího jazyka.

4.1 Obecné informace

K porovnání nástrojů byly vybrány aktuální verze od každého vývojového prostředí. Pro lepší přehlednost jsou uvedené porovnávané verze nástrojů a jejich dostupnost na webu.

- **Scratch 3.0** dostupné na: *https://scratch.mit.edu/projects/editor*
- **Snap 4.2** dostupné na: *https://snap.berkeley.edu/snapsource/snap.html*
- **Karel 2.0.5** dostupné na: *http://karel.oldium.net*

4.2 Zhodnocení výukových prostředí

Pro porovnání byla vybrána kritéria, která umožní vývojová prostředí objektivně zhodnotit. Kritéria byla stanovena na základě shodných požadavků odborné literatury pro výuku algoritmizace a programování. Jedná se o:

- ***Datové typy*** – Pro algoritmizaci a programování je důležité pochopit, s jakým typem (číslo, text apod.) dat se pracuje.
- ***Podpora*** – Myšleno podpora od tvůrců nástroje, zda udržují nástroj aktuální, nebo naopak už nevydávají žádné aktualizace.
- ***Jazyk rozhraní*** – Nejdůležitějším kritériem je, zda nástroj
- ***Selekce a iterace*** – Základním kamenem algoritmizace jsou selekce (if) a iterace/cykly (for, while), které umožnují programu rozhodovat o dalších krocích.
- ***Tvorba nových bloků*** – Aby uživatel nebyl omezený pouze na předpřipravené příkazy/funkce/podmínky je flexibilita nástroje důležitá. Nástroj pak bude mnohem přívětivější uživateli.
- ***Funkce*** – Složitější možnosti nástrojů, jako jsou funkce, lépe připraví uživatele na další postup.
- ***Uložení/Nahrávání*** – Možnost uložit vytvořený projekt a jiný den opět nahrát, pro další práci je dalším důležitým faktorem, který by měl nástroj splňovat

Hodnoty doplněné do tabulky u nástrojů vychází z teoretické části bakalářské práce.

Tabulka 2 – Porovnání nástrojů

	Scratch	**Snap**	**Karel**
Datové typy	Číslo, Text, Pravd. hodnota, Seznam	Číslo, Text, Pravd. hodnota, Seznam Kostým, Zvuk, Sprite	žádné
Podpora	Ano	Ano	Ne
Jazyk rozhraní	Čeština, Angličtina + další (52)	Čeština, Angličtina + další (40)	Čeština
Selekce a iterace	If While For	If While For	If While For
Tvorba nových bloků	Příkazy, Funkce, Podmínky	Příkazy, Funkce, Podmínky	Příkazy
Funkce	Ano	Ano	Ne
Uložení/Nahrávání	Ano	Ano	Ano

Slovně ohodnocená kritéria se kvantifikují podle metod na číselná. Pokud dané kritérium splňuje, pak se ohodnotí číslem 1, pokud ne tak 0.

U datových typů je důležité rozlišení mezi číslem a textem, další typy jsou pro začátečníky zatím irelevantní. Rozhraní musí být přístupné v českém a anglickém jazyce. Nástroj musí umět pracovat se selekcí *if* a iteracemi *while* a *for*. Dále nechat volnější ruku programátorovi při tvorbě nových i složitějších bloků podle potřeby.

Tabulka 3 - Číselné ohodnocení

	Scratch	Snap	Karel
Datové typy	1	1	0
Podpora	1	1	0
Jazyk rozhraní	1	1	0
Selekce a iterace	1	1	1
Funkce	1	1	0
Tvorba nových bloků	1	1	0
Uložení/Nahrávání	1	1	1

Stanovení váhy jednotlivých kritérií byly spočítány metodou párových porovnání, přesněji přes Fullerův trojúhelníkem. Důležitost kritérií byla určena následovně: jazyk rozhraní, selekce a iterace, datové typy, uložení/nahrávání, funkce, podpora, tvorba nových bloků. Všechna kritéria jsou maximalizační.

Tabulka 4 - Vypočítání vah kritérií

Kritérium	A	B	C	D	E	F	G	bj	vj
Datové typy (A)	1	1	0	0	1	1	1	5	0,18
Podpora (B)	0	1	0	0	0	1	0	2	0,07
Jazyk rozhraní (C)	1	1	1	1	1	1	1	7	0,25
Selekce a iterace (D)	1	1	0	1	1	1	1	6	0,21
Funkce (E)	0	1	0	0	1	1	0	3	0,11
Tvorba nových bloků (F)	0	0	0	0	0	1	0	1	0,04
Uložení/Nahrávání (G)	0	1	0	0	1	1	1	4	0,14

vj – vypočtení váha kritérií

Po stanovení vah se metodou vícekriteriální analýzy variant, přesněji metodou pořadí, vypočítala kompromisní varianta.

Tabulka 5 - Hodnocení nástrojů

	Váha	**Scratch**	**Snap**	**Karel**
Datové typy	0,18	1,5	1,5	3
Podpora	0,07	1,5	1,5	3
Jazyk rozhraní	0,25	1,5	1,5	3
Selekce a iterace	0,21	2	2	2
Funkce	0,11	1,5	1,5	3
Tvorba nových bloků	0,04	1,5	1,5	3
Uložení a nahrávání	0,14	2	2	2
Součet		1,675	1,675	2,65
Pořadí		1.	1.	3.

5 Výsledky a diskuse

Z výsledků praktické části bakalářské práce jasně vychází dvě výuková prostředí, která by měla být doporučena pro výuku algoritmizace ve školách, nebo jen pro začínající programátory, které si neví rady se složitostí programovacích jazyků a chtějí se snadno a zábavnou cestou naučit základy.

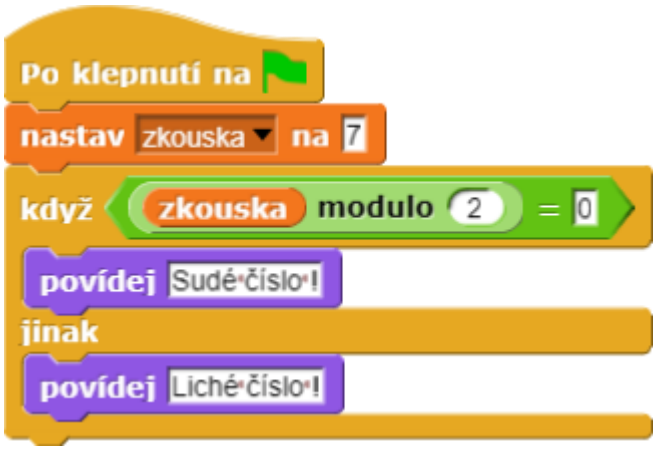
V Tabulka 2 – Porovnání nástrojů, se vycházelo z teoretické části a byly doplněny hodnoty k jednotlivým nástrojům.

Z číselného hodnocení online nástrojů pro výuku algoritmizace v Tabulka 3 - Číselné ohodnocení, vychází dva nástroje ideálně. Jsou to Scratch a Snap, které splnily všechny požadované podmínky na výukový nástroj. Oproti tomu robot Karel neobstál hned u 5 ze 7 kritérií. Jediná dvě kritéria, která Karel splnil bylo možnost mít v algoritmu selekci a iteraci, a snadné uložení a zpětné nahrávání projektů.

Po výpočtu vah kritérií a následného výpočtu nejlepší varianty v Tabulka 5 - Hodnocení nástrojů, vyšly nejlépe nástroje Scratch a Snap, které skvěle zastanou úvod do programování, tedy mají všechny potřebné možnosti a funkce do začátku pro nové programátory. Touto prací jsou doporučeny pro výuku algoritmizace. Oproti tomu robot Karel neměl přívětivé výsledky, mohl by být doporučen na základní školy. Na výuku algoritmizace na středních a vysokých školách se doporučují Scratch nebo Snap.

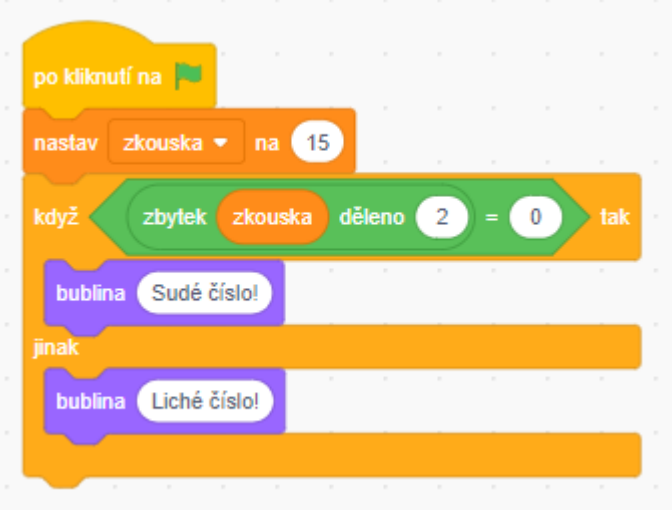
Pro ilustraci byl vytvořen autorem jednoduchý algoritmus, na výběr, zda je dané číslo liché nebo sudé, v hodnocených prostředích. Algoritmus nastavuje vytvořenou proměnnou *zkouska* (datový typ – číslo) na zadané číslo, poté selekcí *if () else* podle ne/splnění podmínky vybere, co vypíše, zda „Sudé číslo!“ nebo „Liché číslo!“ (datový typ – text). V podmínce je operátor porovnání a funkce modulo. Nejprve je vypočítána funkce modulo (zbytek po dělení), pokud je zbytek 0, pak je podmínka splněna a algoritmus pokračuje na další krok. Pokud není zbytek po dělení 0, podmínka není splněná a algoritmus přeskočí do kolonky *else* (jinak). V obou uvedených případech scénář vypíše „Liché číslo!“.

Obrázek 20 - Snap sudé/liché číslo



Zdroj: snap.berkeley.edu/snapsource/snap, 2019

Obrázek 21 - Scratch sudé/liché číslo



Zdroj: scratch.mit.edu/projects/editor, 2019

Z uvedených příkladů lze vidět menší odlišnosti i mezi nástroji Scratch a Snap. Ve výukovém prostředí robot Karel nelze vytvořit ani tento jednoduchý příklad, nemá dostatek funkcí. I proto se doporučují výuková prostředí Scratch a Snap.

6 Závěr

Výuková prostředí velmi přispívají k překonání překážek souvisejících se složitostí při výuce nových programovacích jazyků. Pomáhají učit a zlepšovat logické myšlení dětí zábavnou cestou. Zároveň je to výborný vstupní bod do programování jak pro děti, tak pro dospělé. S tím souvisí problém s výběrem mezi různými výukovými prostředími, která mají odlišné funkce.

V této bakalářské práci byla charakterizována čtyři výuková prostředí a to Scratch, Snap, Petr a robot Karel v jejich aktuálních verzích. Byly ukázané funkce každého prostředí. Možnosti, které jsou společné a zároveň odlišnosti, podle kterých si nástroj začínající programátoři mohou vybírat.

Výběr požadavků na nástroj proběhl na základě poznatků ze studia odborné literatury zabývající se problematikou výuky algoritmizace. Z analýzy vyplynulo sedm kritérií, která by měla výuková prostředí splňovat a v jakém rozsahu. Jsou to: datové typy, podpora, jazyk rozhraní, podmínky a cykly, tvorba nových bloků, funkce, uložení a nahrávání. Důležité při výuce je pochopení, že program může pracovat s různými datovými typy, jako je text nebo číslo. Podpora daného výukového prostředí je nutná pro aktuálnost a využití výuky v dnešní době. Nástroj také musí být lehce pochopitelný pro české programátory, proto je nutné, aby rozhraní bylo v českém jazyce, a jelikož jsou programovací jazyky v anglickém jazyce, tak i v něm. Základním kamenem programování jsou selekce a iterace, proto je výuka podmínky (selekce) *if* a cyklů *for* a *while* dalším požadavkem. Programátor by měl rozumět i možnosti podprogramů neboli funkcím. Volnost v programování si pak každý může vyzkoušet pomocí vytváření nových bloků ve výukových prostředí. Aby programátor mohl pokračovat v práci i jiný den, je potřebná možnost uložení projektu a následné opětovné nahrání.

Z výsledků zhodnocení online nástrojů pro výuku algoritmizace vyšly nejlépe dva nástroje: Scratch a Snap. Oba splňují všechna kritéria. Proto by bylo vhodné doporučit jeden z těchto nástrojů začínajícím programátorům. Oproti tomu robot Karel nesplnil všechna kritéria, a proto by se měla dát přednost při výuce algoritmizace jiným nástrojům.

7 Seznam použitých zdrojů

Bayer, Tomáš. 2018. Algoritmy a jejich znázorňování. *PF UK v Praze.* [Online] 2018. [Citace: 15. Prosinec 2019.] https://web.natur.cuni.cz/~bayertom/images/courses/Prog2/programovani2.pdf.

Bentley, Jon Louis. 2000. *Programming pearls.* New York : ACM Press, 2000. ISBN 0-201-65788-0..

Bory, Pavel. 2016. *C# bez předchozích znalostí.* Brno : Computer Press, 2016. ISBN 978-80-251-4686-6.

Evropský sociální fond v ČR. 2008. Algoritmizace. *coptkm.cz.* [Online] 1. Leden 2008. [Citace: 9. Březen 2019.] https://coptkm.cz/portal/reposit.php?action=0&id=19081&revision=-1&instance=5.

Hylmar, Radek. 2009. *Programování pro úplné začátečníky.* Brno : Computer Press, 2009. ISBN 9788025121290.

Jedlička, Oldřich. 2013. Robot Karel: nápověda. *karel.oldium.net.* [Online] 2013. [Citace: 10. Prosinec 2018.] http://karel.oldium.net/napoveda.html#zapis.

Lifelong Kindergarten Group. 2019. Často kladené otázky. *Scratch.* [Online] 2019. [Citace: 10. Březen 2019.] https://scratch.mit.edu/info/faq#scratch3.

LKG. 2019. Snap! *Scratch Wiki.* [Online] 3. Leden 2019. [Citace: 28. Leden 2019.] https://en.scratch-wiki.info/wiki/Snap!_(programming_language).

Monig, Jens a Harvey, Brian. 2018. Snap! Reference Manual. *Snap!* [Online] 2018. [Citace: 1. Únor 2019.] https://snap.berkeley.edu/SnapManual.pdf.

Němeček, Miroslav. 2013. O Petrovi. *Gemtree Petr.* [Online] 2013. [Citace: 12. Prosinec 2018.] http://www.breatharian.eu/Petr/about.htm.

Prokop, Jiří. 2012. *Algoritmy v jazyku C a C++.* Praha : Grada Publishing, 2012. ISBN 978-80-247-3929-8.

Redware Research Limited. 2015. History of Scratch. *Redware.* [Online] 2015. [Citace: 6. Březen 2019.] http://scratch.redware.com/content/history-scratch.

Roberts, Eric. 2005. Karel the Robot. *Laboratory of Advanced Research on Computer Science.* [Online] September 2005. [Citace: 10. Prosinec 2018.] http://lia.deis.unibo.it/Courses/FondA0809-AUT/materiale/karellearnsjava.pdf.

Rouse, Margaret. 2017. What is Scratch? *WhatIs.* [Online] Prosinec 2017. [Citace: 3. Únor 2019.] https://whatis.techtarget.com/definition/Scratch.

Scratchers. 2018. Scratch 1.x. *Scratch Wiki.* [Online] 28. Listopad 2018. [Citace: 11. Leden 2019.] https://en.scratch-wiki.info/wiki/Scratch_1.x.

Stroud, Forrest. 2019. Scratch programming language. *webopedia.* [Online] 2019. [Citace: 2. Únor 2019.] https://www.webopedia.com/TERM/S/scratch_programming_language.html.

The Scratch Team. 2018. 3 Things To Know About Scratch 3.0. *Medium.* [Online] 5. Duben 2018. [Citace: 10. Březen 2019.] https://medium.com/scratchteam-blog/3-things-to-know-about-scratch-3-0-18ee2f564278.

Wang, Wallace. 2008. *Beginning programming all-in-one desk reference for dummies.* New Jersey : Wiley Publishing, Inc, 2008. ISBN 978-0-470-10854-3 .