Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra Informačních technologí

Bakalářská práce

Online nástroje pro výuku algoritmizace

Matěj Brnka

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Matěj Brnka

Informatika

Název práce

Online nástroje pro výuku algoritmizace

Název anglicky

Online tools for teaching algorithm

Cíle práce

Bakalářská práce je tematicky zaměřena na problematiku online nástrojů pro podporu výuky algoritmizace a programování. Hlavním cílem práce je zhodnocení softwarových nástrojů pro výuku struktorovaného programování.
Dílčí cíle práce jsou:
- charakterizovat dostupné online softwarové nástroje pro výuku algoritmizace a programování
- analýza požadavků na online nástroje pro výuku algoritmizace a programování
- zhodnocení vybraných online nástrojů pro výuku algoritmizace a programování

Metodika

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

V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny požadavky na online nástroje pro výuku algoritmizace a programování. Pomocí metody vícekriteriální analýzy variant budou vybrané nástroje zhodnoceny.

Následně budou formulovány závěry a doporučení.

Doporučený rozsah práce

35 stran

Klíčová slova

algoritmizace, teaching algorithm, výuka algoritmizace, nástroje pro výuku algoritmizace

---

Doporučené zdroje informací

Brian Harvey Jens Mönig, SNAP! Reference Manual
KOLESNÁ, Dana. Základy algoritmizace a programové vybavení. 2. vyd. Praha: Tesla Eltos, 1986.
MILKOVÁ, Eva. Multimediální pomùcky pro výuku algoritmizace / Multimedia tools for teaching algorithmization. Technologia Vzdelavania. 2011, vol. 19, no. 2, s. 1. ISSN 1335-003X.
PŠENČÍKOVÁ, Jana. Algoritmizace. Vyd. 2. Kralice na Hané: Computer Media, 2009. ISBN 9788074020346;8074020347
VIRIUS, Miroslav. Základy algoritmizace. Vyd. 2., přeprac. Praha: Česká technika – nakladatelství ČVUT, 2008. ISBN 978-80-01-04003-4.

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 12. 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 15.3.2019 _____

Poděkování

Rád bych touto cestou poděkoval svému vedoucímu práce Ing. Michalu Stočesi, Ph.D za jeho za odborné vedení, za pomoc a rady při zpracování této práce.

Online nástroje pro výuku algoritmizace

Abstrakt

Práce se v teoretické části zabývá především základními pojmy, které jsou důležité pro výuku algoritmizace. Dále je v práci zmíněno, jak probíhá tvorba školních vzdělávacích programů, podle kterých je pak výuka algoritmizace na jednotlivých typech škol vedena. Na základě poznatků z teoretické části byly analyzovány vybrané školní vzdělávací programy, z nichž vyplynula kritéria pro online nástroje, aby mohly být použity ve výuce algoritmizace. Ve vlastní práci byly vybrané nástroje prozkoumány z hlediska funkčnosti. Na závěr jsou nástroje zhodnoceny pomocí vícekriteriální analýzy variant.

Klíčová slova: algoritmizace, výuka algoritmizace, nástroje pro výuku algoritmizace a programování, Snap!, Scratch, Code.org, Želví grafika

Online tools for teaching algorithm

Abstract

The theoretical part deals mainly with basic terms, which are important for the teaching of algorithmization and programming. It also mentions how school education programs are being developed, according to which the teaching of algorithmization and programming at particular types of schools is performed. Based on the findings from the theoretical part, selected school educational programs were analyzed, which outlined the criteria for online tools to be used in algorithmic teaching. In addition, the selected tools were explored in terms of functionality. Finally, tools are evaluated using multi-criteria analysis of variants.

Keywords: algorithm, teaching algorithm, tools for teaching algoritm and programming, Snap!, Scratch, Code.org, Želví grafika

Obsah

1 Úvod

Informační technologie jsou stále častěji součástí každodenního života. Pracovní trh je v dnešní době v oblasti vývoje aplikací nenasycen, a vzniká tak čím dál tím větší poptávka po programátorech.

Asistovat při výuce mohou právě online nástroje pro výuku algoritmizace, které budou v bakalářské práci hodnoceny. Aby mohly být nástroje zhodnoceny, je nutné určit si kritéria hodnocení. V práci budou analyzovány školní vzdělávací programy, podle kterých se algoritmizace na určitém typu škol vyučují. Z nich vyplynou kritéria, podle kterých bude možné jednotlivé nástroje zhodnotit a určit, které jsou pro výuku vhodné.

Nástrojů a přístupů k programování existuje celá řada. Tato práce bude však pojednávat o nástrojích, které jsou online a dají se využít k efektivní výuce strukturovaného programování.

2 Cíl práce a metodika

Bakalářská práce je tematicky zaměřena na problematiku online nástrojů pro podporu výuky algoritmizace. Hlavním cílem práce je zhodnocení softwarových nástrojů pro výuku strukturovaného programování.

Dílčí cíle práce jsou:

- charakterizovat dostupné online softwarové nástroje pro výuku algoritmizace a programování
- analýza požadavků na online nástroje pro výuku algoritmizace a programování
- zhodnocení vybraných online nástrojů pro výuku algoritmizace a programování

Metodika

Teoretická část bakalářské práce se bude zakládat na analýze a rešerši odborných zdrojů. V praktické části práce budou na základě poznatků zjištěných v analytické části zhodnoceny požadavky na online nástroje pro výuku algoritmizace a programování. Pomocí metody vícekriteriální analýzy variant budou vybrané nástroje zhodnoceny.
Následně budou formulovány závěry a doporučení.

3 Teoretická východiska

3.1 Algoritmizace

Pojmem algoritmizace se rozumí přesně daný postup, který počítač používá při tvorbě programů, jehož prostřednictvím lze řešit nějaký konkrétní problém. Postup řešení algoritmizační úlohy lze rozdělit do několika etap (Bechyňová, 2012):

1. *Definice problému* – V tomto kroku je třeba přesně formulovat požadavky, určit vstupní a výstupní hodnoty.
2. *Analýza úlohy* – Ověříme si, zda je úloha řešitelná a uděláme si první představu o jejím řešení.
3. *Vytvoření algoritmu úlohy* – Vytvoření jednoznačného návodu (postupu), který je třeba provést.
4. *Sestavení a odladění programu* – Na základě algoritmu sestavíme program již v konktérním programovacím jazyce. Odladěním se snažíme eliminovat chyby v zápise, které jsou rozděleny na syntaktické nebo logické.

3.2 Algoritmus a jeho vlastnosti a vyjádření

Pojmem algoritmus se rozumí přesný návod nebo postup, kterým se řeší daný typ úlohy. Tento pojem se nejčastěji vyskytuje v oblasti programování. Slovo algoritmus lze definovat mnoha způsoby. Definice pojmu algoritmus může znít například takto: *„Algoritmus je přesně definovaná konečná posloupnost kroků, která umožní, pro přípustné hodnoty vstupních dat, nalézt v konečném počtu kroků korektní výstupní data“.* (Mikláš, 2018). Kratší definice podle (Pšenčíková, 2009) zní *„Algoritmus je přesný postup, který je potřeba k vykonávání určité činnosti.“* Nejpřesnější definice je však podle (Staňková, a další, 1998) *„Algoritmus je soubor přesně definovaných pravidel určující pořadí vykonávání konečného počtu elementárních operací, který zapezpečuje řešení všech úloh daného typu.“*

Algoritmus má několik vlastností, které musí splňovat. Jsou jimi: konečnost, determinovanost, univerzálnost, rezultativnost, obecnost a efektivita. Ty jsou popsány v následující kapitole.

3.2.1 **Vlastnosti algoritmů**

Podle (Staňková, a další, 1998) má algoritmus tyto základní vlastnosti:

- *Elementárnost* – skládá se z konečného počtu jednoduchých kroků
- *Determinovanost* – po skončení každého kroku lze říct, zda má algoritmus skončit a jestli že ne, tak který krok je třeba vykonat jako další
- *Konečnost* – každý krok algoritmu se vykoná pouze konečný počet krát. Algoritmus tedy v určeném počtu kroků skončí
- *Rezultativnost* – algoritmus vede postup řešení od vstupních údajů k výsledku
- *Hromadnost* – je určen pro všechny problémy stejného typu

3.2.2 **Zápis algoritmů**

Existuje několik způsobů zápisu algoritmů. Bechyňová (2012) uvádí jako tři nejčastější: vývojový diagram, strukturogram a slovní zápis. Další možnosti vyjádření algoritmu jsou dále například: matematický zápis, rozhodovací tabulky a programovací jazyky. (Pšenčíková, 2009)

Vývojový diagram:

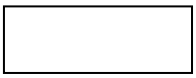
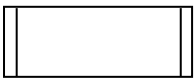
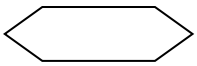
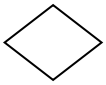
Definice vývojového diagramu je podle Bechyňové (2012) „*Vývojový diagram je druh diagramu, který slouží ke grafickému znázornění jednotlivých kroků algoritmu nebo obecného procesu. Je to grafické znázornění logické struktury řešeného úkolu. Vývojový diagram používá pro znázornění jednotlivých kroků algoritmu symboly, které jsou navzájem propojeny pomocí orientovaných šipek.*“

Odbornější definici uvádí Vaníček a další (2007). Vývojový diagram je pak definován jako prostý orientovaný graf $(\{p\} \cup V \cup R \cup S \cup \{k\}, H)$ s vrcholy pěti typů. Graf musí být souvislý a každý vrchol musí být na jedné cestě od *p* do *k*. Typy vrcholů jsou:

- Jeden vrchol *p*, proces v něm začíná. Má vstupní stupeň 0 a výstupní stupeň 1.
- Jeden vrchol *i*, proces v něm končí. Má vstupní stupeň 1 a výstupní stupeň 0.
- Vrcholy *v* množině *V* mají vstupní a výstupní stupně 1. Provádějí se v nich příkazy.
- Vrcholy *v* množině *R* mají vstupní stupeň 1 a výstupní stupeň 2. Po jejich provedení lze pokračovat dvěma různými směry. Jakým směrem se má pokračovat rozhoduje podmínka.
- Vrcholy v množině *S* mají vstupní stupeň 2 a výstupní stupeň 1. Neprovádí se v nich však žádná akce.

Značky vývojových diagramů jsou definovány normou ČSN ISO 5807. Vybrané značky jsou znázorněny a popsány v Tabulka 1.

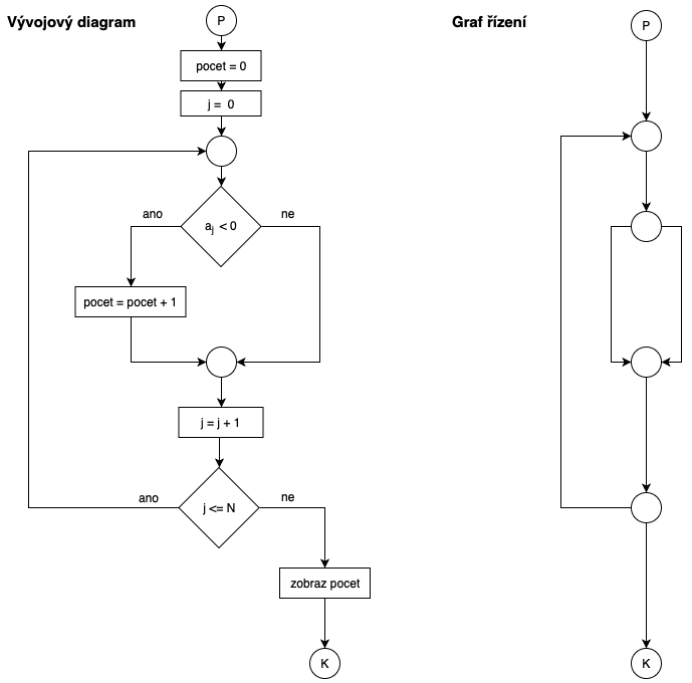
Tabulka vybraných značek:

Značka:	Význam:	Popis:
	Spojnice	Slouží ke spojování jednotlivých bloků vývojového diagramu.
	Zpracování	Provádí definované operace nebo skupiny operací, jejichž výsledkem je změna hodnoty.
	Předdefinované zpracování	Představuje pojmenované zpracování, které je specifikováno v jiné části vývojového diagramu.
	Příprava	Určuje úpravu instrukce nebo instrukcí pro ovlivnění následné činnosti, např. nastavení spínače.
	Rozhodování	Rozhodovací nebo alternativní funkce, která obsahuje jeden vstup a více výstupů, z nichž je jeden aktivován po vyhodnocení podmínky definovaných uvnitř symbolu.
	Spojka	Používá se k přerušení spojnice a k jejímu pokračování na jiném místě.

Tabulka 1 - Vybrané značky vývojových diagramů (Český normalizační institut, 1995)

Graf řízení algoritmu

Pro teoretické účely jsou jednodušší kroky, které mají jeden vstup a jeden výstup a jsou reprezentovány jako hrana grafu. Na každé hraně pak lze doplnit jeden nebo několik vrcholů z množiny V s jedinou výstupní hranou. Tím vznikne orientovaný graf $(\{p\} \cup V \cup R \cup S \cup \{k\}, H)$, který nazýváme jako **graf řízení algoritmu.** Vizuální zobrazení ilustruje Obrázek 1.



Obrázek 1 - Vývojový diagram a graf řízení, počet záporných čísel v poli (Vaníček, a další, 2007)

Strukturogram:

Jedná se o úspornější znázornění algoritmu než u vývojového diagramu. Tvoří je obdélníková tabulka, kam do řádků zapisujeme postup kroků. (Bayer)

Odborněji podle (Vaníček, a další, 2007) je strukturogram hierarchický rozklad jako orientovaný kořenový strom, kde kořenem je celá úloha. Jednotlivé listy stromu jsou dílčí úkoly v hierarchickém rozkladu. Shodují se také s jednotlivými podgrafy grafů řízení. Pokud se jedná o iteraci, využívá se značky * v případě že o selekci, používá se značka *O*. Značení je však nejednotné. Výhodou v rámci strukturovaného programování je, nemožnost návrhu nestrukturovnaného programu, naopak jejich nevýhodou je že na nich není na první pohled vidět časový sled.

Slovní zápis:
Podle Pšenčíkové (2009) se o slovní vyjádření algoritmu jedná v případě, že popis určité činnosti splňuje všechny podmínky algoritmu, tzn. je dostatečně přesný, věcný, správný atd. Mohou to být například návody k používání výrobků, technologické postupy atd.

3.3 Programování

„Programování je proces přepisu algoritmu do zdrojového kódu určitého programovacího jazyka za pomoci rozhodovacích struktur.“ (Pšenčíková, 2009)

3.3.1 Struktura programovacího jazyka, syntaxe a sémantika

Abeceda – je libovolná, neprázdná, konečná množina znaků (symbolů). Abeceda by měla mít alespoň dva prvky. Příklady abecedy *{A, B, C, D, …, Z}* nebo *{1, 2, 3, …9, +, -, *, /}* nebo {*if, else, while, for*.. a další klíčová slova programovacího jazyka}.
Slovo – nad danou abecedou je libovolný, konečný a případně i prázdný řetězec znaků abecedy. Například, máme abecedu *{0,1}* a slova mohou být *{00, 01, 11, 10, 000, 111… }*.
Jazyk – v oblastní informatiky podle Vaníčka a dalších (2007) je definován jako „*Je-li dána abeceda V, potom libovolná podmnožina množiny všech slov nad touto abecedou se nazývá formální jazyk“.* Například „*Množina všech slov zadané délky. Třeba byte (abeceda {0,1}, délka 8 – jazyk má 256 slov)*.“
Syntaxe – formální stránka jazyka – pravidla, jimiž se řídí formální zápis programu a jednotlivých prvků jazyka (Staňková, a další, 1998).
Sémantika – významová stránka jazyka – souhrn pravidel, podle kterých se každé správné konstrukci přiřazuje význam (Staňková, a další, 1998)

3.3.2 Datové prvky, struktury a proměnné

Základními kódy pro elementární informace jsou datové prvky neboli elementární datové typy. Jak uvádí Vaníček a další (2007) se za ně nejčastěji považují:

- *Logická hodnota* – Pravda / Nepravda, nejčastěji reprezentována jako jeden bit.
- *Znak* **–** Nejčastěji reprezentovaný jako jeden byte (ASCII), nebo dvoubitové slovo (UNICODE).
- *Číslo*, které může být dvou typů:
    - *Celé číslo* (*integer*) - různě omezeného rozsahu v pevné desetinné čárce

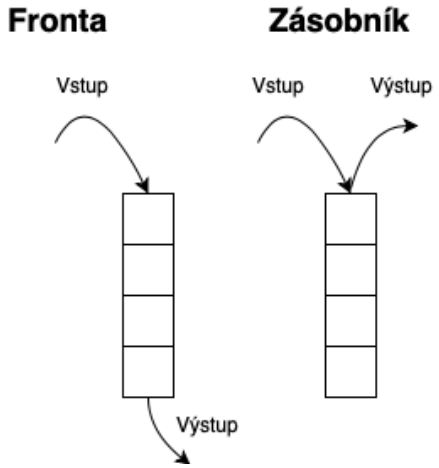
- *Reálné číslo* (*real*) – s různou přesností danou rozsahem mantisy.

Ze základních datových typů pak můžeme skládat složené datové typy, též strukturované datové typy. Ty jsou definovány takto: *„Každý elementární datový typ je strukturovaný datový typ. Homogenním a nehomogenním složením strukturovaných datových typů vznikne opět strukturovaný datový typ. Nic jiného než to, co vznikne opakovaným využitím předchozích pravidel, již strukturovaný datový typ není.“* (Vaníček, a další, 2007)

Ty mohou vzniknout:

- Ze struktur téhož datového typu (homogenní skládání) vytvoříme posloupnost, která může být:
  - *Pole* – předem určená délka a položky jsou přístupné přímo pomocí indexu
  - *Proud* – délka není předem určená, přístup je možný pomocí sekvence
- *Záznam* – skládání položek různých datových typů

Některé strukturované typy mohou mít také speciální pojmenování, např. pole znaků se nazývá **řetězec**. Některé typy mohou mít jen omezené možnosti přístupu k jednotlivým položkám, např. **zásobník** (LIFO – Last In First Out) umožňuje přístup jen k poslední položce. Naopak **fronta** (FIFO – First In First Out), umožňuje pouze přístup k první uložené položce. (Vaníček, a další, 2007). Schéma zásobníku a fronty je znázorněno na Obrázek 2.



Obrázek 2 - Schéma struktur zásobník a fronta

Proměnné jsou podle Virius (1995) již pojmenovaná místa v paměti počítače. Jejich vytvoření probíhá pomocí deklarace, kdy určíme jméno a datový typ. Proměnné mohou být tří typů, globální, lokální a dynamické.

- *Globální proměnné* – k jejich vytvoření dojde při spuštění programu a existují po celou dobu běhu programu. Ve většině programovacích jazycích jsou proměnné deklarované mimo těla podprogramů.
- *Lokální proměnné* – tento typ proměnných je deklarován v procedurách nebo funkcích. K jejich vzniku dochází při volání podprogramu a při ukončení podprogramu zanikají. Při rekurzních voláních se vytváří instance lokálních proměnných pro každé volání podprogramu.

3.3.3 **Podprogramy**

Podprogramy se využívají v případě, pokud potřebujeme opakovat určitou a do jisté míry logicky podobnou část programu.

Výhodou využití podprogramů je, že kód funkce je naprogramován jen jednou, čímž se šetří práce programátora. V případě, že je využít podprogram, je realizován skok na jeho začátek. Po provedení programu následuje návrat do hlavního programu k dalšímu příkazu.

Činnost podprogramu nebývá realizována se stejnými vstupními údaji, tzv. parametry. Ty je možno podprogramu předat při jeho volání v hlavním programu. Každý podprogram je definován jménem.

Podprogramy se pak dělí na dva typy, funkce a procedury. Funkce má na rozdíl od procedury po provedení přiřazenou konkrétní hodnotu, kterou vrací hlavnímu programu. (Kolesná, 1986)

3.3.4 **Programovací konstrukce rekurze**

„*Jako rekurzivní označujeme takové volání procedury nebo funkce, při kterém dojde k aktivaci těla podprogramu dříve, než se ukončí aktivace předchozí*“. (Virius, 1995)

Pro příklad rekurzivního myšlení lze úvést jako příklad (Wróblewski, 2004) sbírání kostek do krabice. Zadání pro dítě „*Seber všechno dohromady a ulož zpátky do krabice. Zadání je takto pro dítě složité, kostek je celá hromada a neví jak má postupovat*“. Pokud bude úkol zadaný rekurzivně zněl by takto „*Vezmi jednu kostku, vlož ji do krabice a*

následně posbírej ostatní“. Autor použil tento příklad, aby dokázal že je rekurzivní myšlení pro člověka přirozené.

Ačkoliv je použití rekurze dnes většinou programovacích jazyků podporováno, Wróblewski (2004) doporučuje použít rekurzi, avšak do konečné verze programu využít iterační tvar a globální proměnné. Rekurze lze odstranit například s použitím složené datového typu zásobník.

Podle Pelánka (2018) je rekurze v programování a v informatice obecně důležitá. „*Její použití je velmi jednoduché, na druhou stranu jde o myšlenkově hodně náročný koncept“*, uvádí autor ve své knize.

3.3.5 **Programovací jazyky a paradigmata**

Podle Vaníčka a dalších (2007) nejstarším paradigmatem je **imperativní paradigma**, někdy nazývané též jako paradigma příkazové. Vychází z představy, že počítač je stroj RASP (Random Access Stored Program), jehož činnost spočívá ve změnách obsahu paměti rozdělené do adresovatelných míst. V paměti jsou uložena data i instrukce. Původní představa imperativního programování odpovídala programování ve strojovém kódu nebo Assembleru. Odpovídají ji však i jiné imperativní programovací jazyky, rozdíl je však v tom, že jeden příkaz provede celý blok instrukcí. Tyto jazyky také poskytují možnost změny pořadí příkazů oproti pořadí, ve kterém byly napsány pomocí příkazů typu *goto.*

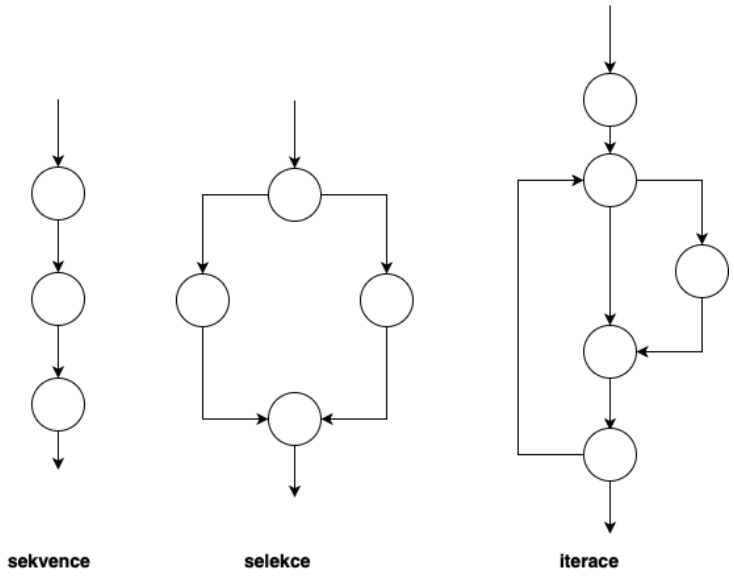
Složité konstrukce a zápisy algoritmů imperativního programování vedly již v šedesátých letech ke vzniku **strukturálního paradigmatu.** Strukturovaný návrh programů spočívá v postupu zvaném **návrh shora-dolů**. Myšlenka strukturovaného programování spočívá v redukci grafů řízení pouze na zcela nejjednodušší řídící struktury. Těmi jsou sekvence, selekce a iterace. **Strukturovaný algoritmus** je pak podle Vaníčka a dalších (2007) definován jako „*Každý jednoduchý krok je strukturovaným algoritmem. Sekvence, selekce a iterace strukturovaných algoritmů je také strukturovaný algoritmus. Nic jiného než to, co získáme opakovaným užitím přechozích bodů strukturovaný algoritmus není. P*odle Polanského (2012) je *„Základní myšlenka strukturovaného programování je zdánlivě velmi přirozená a jednoduchá: při tvorbě algoritmu a programu postupuj shora dolů, maximálně se drž abstraktního přístupu, používej jenom tři základní řídicí struktury, sekvence, selekce a opakování a detailní definici datových struktur odkládej až do té doby, dokud ji nepoužiješ při zápisu programového kódu.*“.

3.3.6 **Řídící struktury strukturovaného programování**

Podle Vaníčka a dalších (2007) se jedná o grafy, které mají žádný nebo jeden rozhodující blok.

- *Sekvence* – posloupnost – předepisuje postupné provedení dílčích příkazů, jediná konstrukce bez rozhodovacího bloku
- *Selekce* – větvení – předepisuje provedení dílčích příkazů na základě splnění či nesplnění určité podmínky
- *Iterace* – opakování – předepisuje opakované provádění dílčích příkazů na základě splnění či nesplnění určité podmínky. Cykly mohou být tří typů: s podmínkou na začátku, na konci a s předem určeným počtem opakování. (Balík, 2014)

Schémata řídících struktur jsou znázorněna na Obrázek 3.



Obrázek 3 - Schéma základních konstrukcí strukturovaného programování (Vaníček, a další, 2007)

3.4 Tvorba školních vzdělávacích programů

Od roku 2004 jsou kurtikulární (podle slovníku „*smysluplné uspořádání na sebe navazujících výchovných a vzdělávacích prvků"*) dokumenty na dvou úrovních. První z úrovní je státní úroveň představují, kterou představuje Národní program vzdělávání (NVP neboli Bílá kniha). Bílá kniha podle Ministertsva školství, mládeže a tělovýchovy (2015) „*Formuluje vládní strategii v oblasti vzdělávání. Strategie odráží celospolečenské zájmy a dává konkrétní podněty pro práci škol"*. Na první úrovni jsou zařazeny také Rámcové vzdělávací programy, které byly do systému vzdělávání zavedeny zákonem č. 561/ 2004 Sb. o předškolním, základním, středním, vyšším odborném a jiném vzdělávání (školský zákon). Ten byl v roce 2015 novelizován. *„Rámcové vzdělávací programy (RVP)*

tvoří obecně závazný rámec pro tvorbu školních vzdělávacích programů škol všech oborů vzdělání v předškolním, základním, základním uměleckém, jazykovém a středním vzdělávání.“ Rámcové vzdělávací programy vydává Ministerstvo školství, mládeže a tělovýchovy po projednání s příslušnými ministerstvy. (Národní ústav pro vzdělávání, nedatováno).

Druhá úroveň je školní a tu představují školní vzdělávací programy (ŠVP), podle nichž se na jednotlivých školách vyučuje. Školní vzdělávací program musí být souladu s Rámcovým vzdělávacím programem, pokud je pro daný obor vydán. Obsah vzdělávání může být ve školním vzdělávacím programu uspořádán do předmětů nebo do jiných ucelených částí učiva. Školní vzdělávací program vydává ředitel školy. (Národní ústav pro vzdělávání, nedatováno).

3.4.1 **Začátky v programování**

Děti schopny pochopit základní struktury již v první třídě, vytvářet pak jednoduché programy. Musí však mít nástroj, který je dostatečně jednoduchý a intuitivní. Navíc musí mít jazyk s jednoduchou syntaxí a prostředím, které se jim dostatečně líbí, aby v něm měly chuť řešit problémy. Dle zkušeností autora však není vhodné učit děti programovat od první třídy. Doporučuje začít mezi třetí a šestou třídou základní školy. (Pecinovský, 2001)

V zemích jako je například Velká Británie, Estonsko nebo Finsko (Kubátová, 2015) začínají s výukou programování již v první třídě.

3.5 **Online nástroje**

Nástroje z rodiny BYOB (Build Your Own Blocks) jsou nástroje postavené na principu „postav si svůj blok“. Práce bude zaměřena na zhodnocení nejpoužívanějších online nástrojů: Scratch, Snap!, Code.org a Želví grafika.

Programování probíhá přetahováním již předdefinovaných bloků na pracovní plochu. Bloky do sebe zapadají jako puzzle. V některých nástrojích si bloky může definovat sám uživatel, pak se jedná o funkce, které jsou popsané v předchozích kapitolách.

Všechny nástroje mají zobrazení odpovídající strukturogramu, z tohoto důvodu lze na těchto nástrojích velmi dobře demonstrovat posloupnost příkazů shora dolů. Ve všech nástrojích lze programovat například pohyb objektu (spritu) po souřadnicích X a Y, vzhled, zvuk a další. Následně pak lze využívat složitějších struktur, jako jsou například cykly, podmínky, či složitější proměnné. Jelikož se jedná o online nástroje, závisí na výkonu počítače jen minimálně.

4 Vlastní práce

4.1 Výuka algoritmizace podle RVP a ŠVP

Byly vybrány dva rámcové vzdělávací programy pro odborné středoškolské obory, konkrétně Ekonomika a podnikání a Informační technologie. Třetím je rámcový vzdělávací program pro gymnázia. Základní školy budou zmíněny jen okrajově. Záměrně byly vybrány tyto školy, aby bylo možné tyto typy škol porovnat mezi sebou z hlediska výuky algoritmizace.

4.1.1 Základní školy

V rámcovém vzdělávacím programu pro základní školy není zmínka o výuce algoritmizace ani programování. Některé základní školy však nabízí od druhého stupně možnost rozšířené výuky informatiky. (Jeřábek, a další, 2017)

Pro výzkum byly vyhodnoceny následujících školy z Prahy, které rozšířenou výuku informatiky nabízí. Ne všechny však zařazují do hodin základy algoritmizace, například ZŠ Glowackého na Praze 8 (ZŠ Glowackého). Naopak ZŠ Petřiny – jih (ZŠ Petřiny - jih) nebo ZŠ Máchovka (ZŠ Máchovka) učí žáky algoritmizovat, programově myslet, a dokonce i základy programování.

4.1.2 Rámcový vzdělávací program pro gymnázia

Dle očekávaných výstupů RVP pro gymnázia by měli žáci uplatňovat algoritmický způsob myšlení při řešení problémových úloh. V rámci učiva by měli probírat algoritmizaci úloh, co je to algoritmus, jak jej zapsat a úvod do programování. (Jeřábek, a další, 2013).

Některá gymnázia do výuky informatiky začleňují algoritmizaci a programování, např. Gymnázium J. Vrchlického v Klatovech, konkrétně dle ŠVP žák *„Je schopen porozumět nejzákladnějším algoritmickým konstrukcím zapsaným ve formě programu ve vyšším programovacím jazyku.“* (Gymnázium J. Vrchlického v Klatovech) nebo gymnázium Českolipská na Praze 9 ve svém ŠVP uvádí jako výstupy žáka ze septimy „*aplikuje algoritmický přístup k řešení problému s využitím získaných znalostí a vytváří algoritmicky správné vývojové diagramy“*. V rámci předmětu se pak vyučuje *„základní informace o tvorbě algoritmů a programů, úvodní pojmy a znalosti – algoritmus a jeho vlastnosti, zápis algoritmů, program, principy vývoje algoritmů, příkazy, data.“*. (Gymnázium Českolipská)

4.1.3 **Rámcový vzdělávací program Obchodní akademie**

Podle tohoto RVP v kapitole vzdělávání v informačních a komunikačních technologiích by měli žáci kromě základní činnosti práce na PC probírat také algoritmizaci. Konkrétně „*ovládá principy algoritmizace úloh a sestavuje algoritmy řešení konkrétních úloh (dekompozice úlohy na jednotlivé elementárnější činnosti za použití přiměřené míry abstrakce“*. (Národní ústav odborného vzdělávání, 2007)

Podle ŠVP Obchodní akademie Tomáše Bati ve Zlíně žák *„chápe pojem algoritmus, zná základní vlastnosti algoritmu, umí provést dekompozici úlohy na jednotlivé elementární činnosti“.* (Obchodní akademie Tomáše Bati, 2009)

Naopak soukromá střední škola Obchodní akademie a institut Praha, s.r.o. na Praze 9 výuku algoritmizace ve svém ŠVP neuvádí. (Obchodní akademie Praha, s. r. o.)

4.1.4 **Rámcový vzdělávací program Informační technologie**

Stejně jako u předchozích škol by zde měli žáci probírat algoritmizaci podle kapitoly vzdělávání v informačních a komunikačních technologiích. Jelikož se jedná o obor přímo zaměřený na informatiku, absolvent by měl mít také odborné kompetence v této oblasti, mimo jiných „*Programovat a vyvíjet uživatelská, databázová a webová řešení, tzn. aby absolventi algoritmizovali úlohy a tvořili aplikace v některém vývojovém prostředí; realizovali databázová řešení; tvořili webové stránky.“*

Aby absolventi mohli mít tyto kompetence tak „*Cílem obsahového okruhu je naučit žáka vytvářet algoritmy a pomocí programovacího jazyka zapsat zdrojový kód programu. Žák porozumí vlastnostem algoritmů, dále se naučí používat zápis algoritmu, datové typy, řídící struktury programu, jednoduché objekty a základní příkazy jazyka SQL*.“ (Národní ústav odborného vzdělávání, 2008)

Výsledky vzdělávání žáka:	Učivo:
• zná vlastnosti algoritmu • zanalyzuje úlohu a algoritmizuje ji • zapíše algoritmus vhodným způsobem	**Algoritmizace** • význam • prvky algoritmu
• použije základní datové typy • použije řídící struktury programu; • vytvoří jednoduché strukturované programy	**Strukturované programovaní** • datové typy • řídící struktury

Tabulka 2 - Cíle obsahového okruhu programováni a vývoj aplikací, (Národní ústav odborného vzdělávání, 2008)

Podle (SPŠE V Úžlabině, 2016) v předmětu Úvod do programování je žák veden k tomu, aby:

- *„používal správnou terminologii algoritmizace a příkazových struktur*
- *analyzoval text úlohy, postihnul podstatu problému a hledal nejjednodušší cestu k jeho řešení, odhadl a zdůvodnil výsledky*
- *sestavil algoritmus, na jeho základě odladil funkční program a ověřil jeho správnost*
- *prováděl základní analýzu problému, navrhoval strukturu dat vhodnou ke zpracování úlohy“*

Tematické celky jsou pak rozděleny do dvou ročníků. Na algoritmizaci a strukturované programování je kladen důraz v prvním ročníku. Jde především o tyto tematické celky:

- *„Vývojové prostředí*
- *Algoritmizace a vývojové diagramy*
- *Příkazy, proměnné jednoduchých datových typů, operátory*
- *Řízení běhu programu*
- *Funkce*
- *Strukturované datové typy*
- *Pokročilejší koncepty v programování“*

Dle SPŠ Prosek se algoritmizace vyučuje v předmětu Databáze a programování od druhého ročníku. *„Obsahem předmětu je nejprve seznámení s programováním obecně, následuje úvod do algoritmizace, dále programování s využitím základních i pokročilých technik, nástrojů a funkcí“.*

4.2 Požadavky na online nástroje

Průzkumem jednotlivých rámcových vzdělávacích programů a školní vzdělávacích programů bylo zjištěno, co žáci musí ovládat. Na základě toho vznikly základní požadavky na online nástroje, aby v nich mohlo být dané téma vyučováno. Těmto funkcionalitám je ve výsledku pak přidělena větší váha. Existují také další požadavky, které například jen usnadňují práci učitelům, nebo mohou dělat výuku zábavnější. Tyto další požadavky jsou uvedeny až v Tabulka 4.

Požadavek:	**Řešená problematika v teoretické části**
Datové typy	Podle kapitoly 3.3.2 je výsledkem vzdělávání použití základních datových typů
Složené datové typy	Podle kapitoly 3.3.2 „Podle (Pelánek, 2018) je rekurze v programování a v informatice obecně důležitá.“ Avšak „do konečné verze programu využít integrační tvar a globální proměnné. Rekurze lze odstranit například s použitím zásobníku“ a zásobník je složitější proměnná“
Funkce	Podle kapitoly 3.3.3. „Výhodou využití podprogramů je, že kód funkce je naprogramován jen jednou, čímž se šetří práce programátora.“
Procedury	
Selekce	Podle kapitoly 3.3.6 „mezi základní řídící struktury patří selekce“
Iterace	Podle kapitoly 3.3.6. „mezi základní řídící struktury patří selekce“

Tabulka 3 - Seznam požadavků na online nástroje, zdroj autor

4.3 Online nástroje

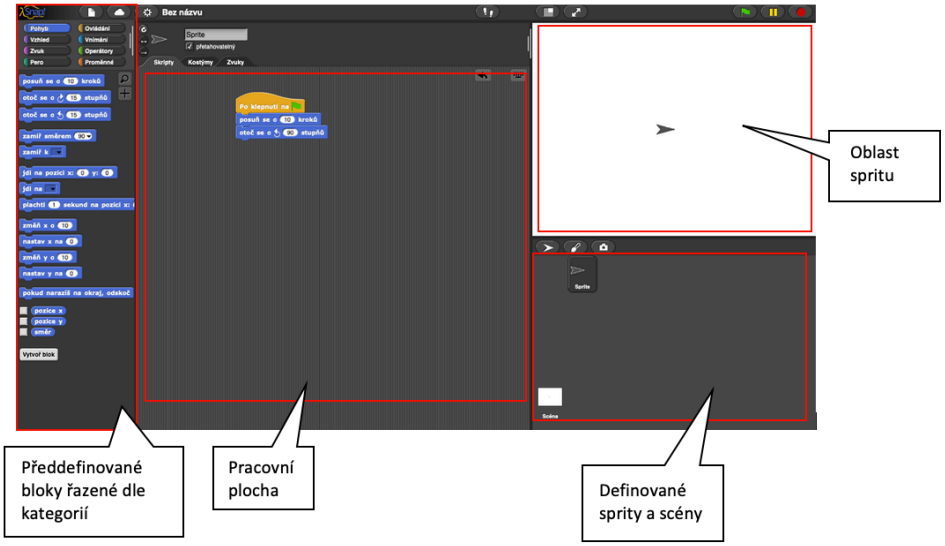
4.3.1 Snap!

Snap! (dříve BYOB) je vizuální drag-and-drop programovací nástroj. Jedná se o rozšířenou verzi jazyka Scratch s novými funkcionalitami, jako je například vytváření a volání podprogramů s návratovou hodnotou. Poprvé se objevil v roce 2011. Díky tomu je Snap! vhodný pro úvod do počítačů pro vysoké školy a univerzity. Autorem programu je Jens Mönig a o design se stará Brian Harvey jak uvádí (Snap!). Nástroj je dotupný na webové stránce https://snap.berkeley.edu.

Prostředí

Základní prostředí Snap! je lokalizováno do 42 jazyků, včetně češtiny. Některé z rozšiřujících knihoven však češtinu vůbec nepodporují. Prostředí lze rozdělit na čtyři logické části, viz Obrázek 4.

- Část s předdefinovanými bloky, které jsou rozděleny podle následujících kategorií – pohyb, vzhled, zvuk, pero, ovládání, vnímání, operátory a proměnné
- Pracovní plocha – na ni se právě předdefinované bloky přetahují a skládají
- Oblast spritu – po spuštění vytvořeného programu (např. kliknutím na zelenou vlajku) provádí sprite program.
- Definované sprity a scény – je možné si definovat více scén a spritů. Ty pak mohou mít odlišný vzhled a v programu si lze sprite nebo scénu změnit. Sprity dokáží mezi sebou komunikovat pomocí zpráv.



Obrázek 4 - Prostředí nástroje Snap!, zdroj autor

Funkce

V prostředí lze vytvářet libovolné programy skládáním libovolných bloků. Do prostředí si lze také importovat externí knihovny, které umožní například použití cyklu for či práci s textovými řetězci. Rozšířit prostředí je také možné dalšími kostýmy (vzhled spritu), které jsou vytvořené tvůrci programu. Lze také vytvářet vlastní bloky, které nazýváme podprogramy. Po přihlášení do uživatelského účtu je pak možné ukládat si projekty na integrované cloudové úložiště, který Snap! poskytuje registrovaným uživatelům zdarma.

Podpora pro výuku

Snap! má zpracovaný manuál, který je dostupný zdarma. Manuál je pouze v anglickém jazyce. Je zde také zveřejněno několik projektů, včetně zdrojového kódu pro inspiraci.

Technologie

Snap je naprogramován v JavaScriptu s pomocí HTML5. Výhodou je možnost spuštění bez ohledu na operační systém. (Wikipedia, 2019)

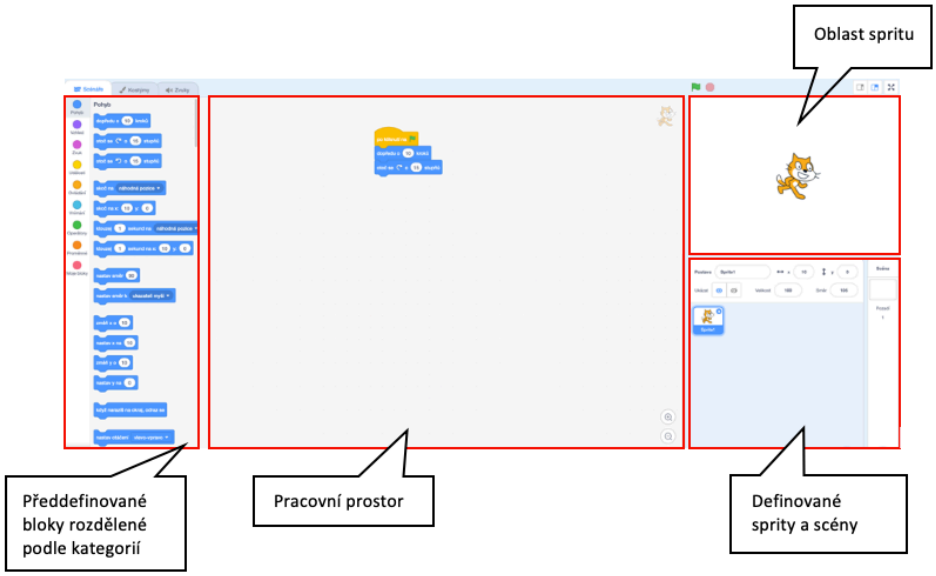
4.3.2 Scratch

Scratch je stejně jako ostatní nástroje drag-and-drop vizuální programovací nástroj. Jeho první prototyp se objevil v roce 2002 a v roce 2007 byla spuštěna první veřejná verze. Autorem nástroje je Mitchel Resnick. (Wikipedia, 2019). Podle (Scratch) je nástroj určen zejména pro děti od 8 do 16 let. Má registrovaných přes 35 milionů uživatelů a je dostupný ve více než 150 zemích světa. Převládá především USA s více než 40% uživatelů. (Scratch). Nástroj je dostupný na stránce https://scratch.mit.edu/.

Prostředí

Prostředí Scratch je dostupné ve více než 40 jazycích, včetně češtiny. Uživatelské prostředí je složené podobně jako u Snap! ze 4 oblastí, viz. Obrázek 5.

- Předdefinované bloky rozdělené podle kategorií
- Pracovní plocha – na ni se předdefinované bloky přetahují a skládají do sebe
- Oblast spritu – po spuštění vytvořeného programu (např. kliknutím na zelenou vlajku) provádí sprite příkazy.
- Definované sprity a scény – stejně jako u Snap! může být definováno více spritů a scén. Sprity si mezi sebou dokáží předávat zprávy a případně na ně reagovat.



Obrázek 5 - Prostředí Scratch, zdroj autor

Funkce

Podobně jako u Snap! lze prostředí Scratch rozšiřovat o externí knihovny. Možnosti knihoven ve Sratch jsou mnohem zajímavější. Přidávají například možnost programování „*hraček*“ třetích stran, jako je například *micro:bit*, nebo vnímat video z hardwarové kamery.

Pro ukládání lze využít stejně jako ve Snap! integrované online úložiště, které je dostupné po registraci a přihlášení zdarma.

Podpora pro výuku a komunita

Kde však Scratch značně dominuje nad Snap! je komunita a podpora výuky. Díky možnosti sdílení projektů vzniká velký prostor k inspiraci. Komunita si pak může sdílené projekty prohlížet, spouštět, *„komentovat či lajkovat*“.

Lektoři pak mohou využívat svůj komunitní portál ke sdílení nápadů, vyměňování zdrojů a pokládání otázek svým kolegům z celého světa. Jsou zde také zpracované video návody. Dále je možnost stáhnout si kartičky s úkoly pro studenty. Oboje je však dostupné jen v angličtině.

Učitelé pak mohou svůj účet upgradovat na účet učitele. Tím dostane možnost zakládat účty pro studenty bez použití emailů, spravovat projekty a komentáře studentů.

Zkušenosti se Scratch

Podle (Humpolcová, 2012) je Scratch vhodný pro použití nejen doma, ale také do zájmových kroužků či školní výuky informatiky. Autorka článku jej zařadila do výuky pro první ročníky SŠ Obchodní akademie SOVA, o.p.s. Neratovice a dle jejích slov z něj žáci byli nadšení.

4.3.3 Code.org

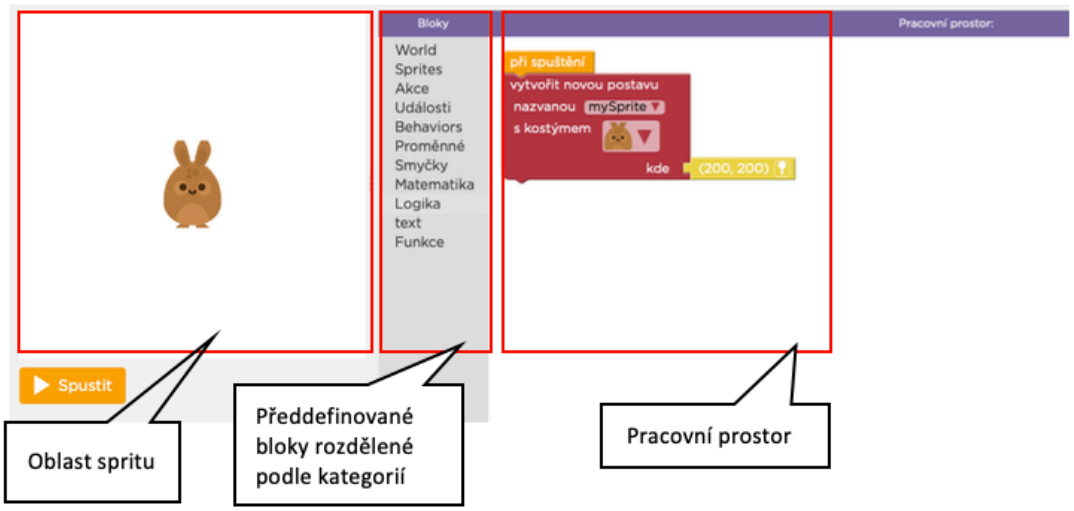
Code.org je nezisková organizace, která vznikla v roce 2013. Byla založena bratry Hadi Partovi a Ali Partovi. Organizace na svých webových stránkách provozuje online nástroje pro výuku algoritmizace. Mezi ně patří například SpriteLab, kterému bude práce dále věnována. Podobá se nástrojům Snap! a Scratch. Dalšími nástroji které Code.org poskytuje je GameLab, ve kterém je možné vytvořit si vlastní hru včetně animací, nebo několik typů Minecraft světa, kde pomocí skládání bloků můžete prozkoumávat svět prostřednictvím postav.

Prostředí SpriteLab

Prostředí nástroje Code.org je lokalizováno do 40 jazyků včetně češtiny. Ta však není zcela podporována, a tak překlady některých bloků zcela chybí. Prostředí se skládá ze tří částí, viz. Obrázek 6. Těmi jsou:

- oblast spritu, kde vykonává své příkazy,
- bloky rozdělené podle kategorií,
- prostor pro skládání bloků.

Na rozdíl od Snap! a Scratch zde není speciální prostor pro vytváření scén a spritů, ačkoliv nástroj podporuje vytvoření více spritů.



Obrázek 6 - Prostředí SpriteLab, zdroj autor

Funkce SpriteLab

V nástroji SpriteLab od Code.org není možnost vytvářet vlastní scény. Je zde definována pouze jedna scéna. Lze však vytvořit více spritů, ty však ale nedokáží na sebe reagovat pomocí zpráv, jak je tomu u předchozích nástrojů.

Počet předdefinovaných bloků je také podstatně menší, než je tomu u předchozích nástrojů. Je zde možný pohyb spritu, matematické a logické výrazy, práce s jednoduchými proměnnými či změna vzhledu spritu.

Po registraci je zde také možnost uložit své projekty na cloud a případně je sdílet s komunitou. Na rozdíl od předchozích nástrojů však SpriteLab umí pracovat s historií verzí a exportovat kód do JavaScriptu.

Podpora pro výuku a komunita

K dispozici jsou čtyři kurzy po přibližně dvaceti lekcích. Jsou rozdělené podle věku od 4 do 6 let, od 6 let a více, kde je předpokladem znalost čtení, od 8 do 18 let a od 10 do 13 let. Jednotlivé kurzy na sebe navazují. Některé části kurzu probíhají i bez využití

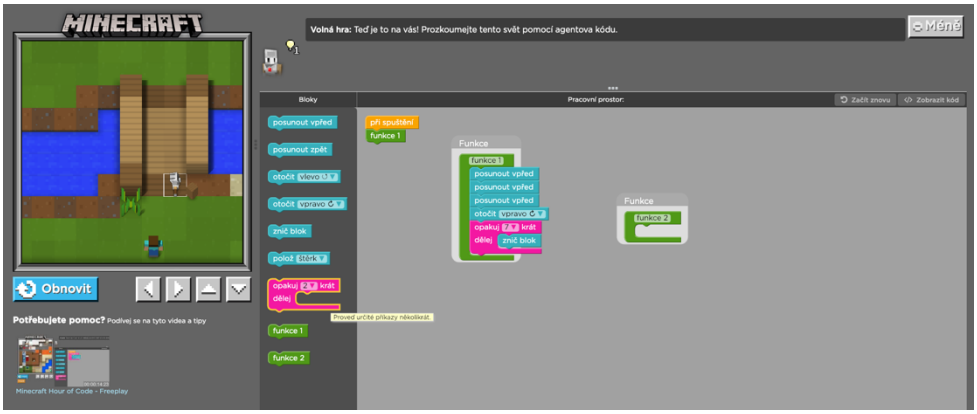
počítače. Prostředí kurzu je doprovázeno výukovými videi, které jsou dostupné v angličtině.

Je zde například řešení bludiště, kde jsou k dispozici bloky pro posun a otočení vlevo nebo vpravo. Díky podpoře velkých firem, jako jsou například Microsoft, Disney, Rovio jsou v kurzech použity náměty ze slavných filmů a počítačových her.

Podobně jako u Scratch je možné si uživatelský účet zapsat do určité třídy, kterou učitel vede. Učiteli to může pomoci ve správě projektů, uvidí postup žáků v kurzech a může obnovit heslo v případě jeho ztráty.

Prostředí Minecraft Hero

V tomto prostředí je k dispozici devět předdefinovaných bloků: posun vpřed, posun vzad, otočení vlevo a vpravo, zničení a položení bloku, iterace, a funkce jedna a dvě. Úkolem je postavit mosty přes řeku, aby se mohla postavička dostat přes vodu. Prostředí je rozloženo stejně jako u SpriteLab, jak demonstruje Obrázek 7.



Obrázek 7 - Ukázka Minecraft Hero, zdroj autor

4.3.4 Želví grafika

Želví grafika je součástí českého projektu umímeto.org, jehož autory jsou Petr Jarušek a Radek Pelánek. Projekt vzniká ve spolupráci s výzkumnou skupinou Adaptive Learning na Fakultě informatiky Masarykovy univerzity v Brně.

V oblasti programování je k dispozici celkem 22 oblastní s celkem 537 příklady, které jsou určené pro první a druhý stupeň základních škol a střední školy.

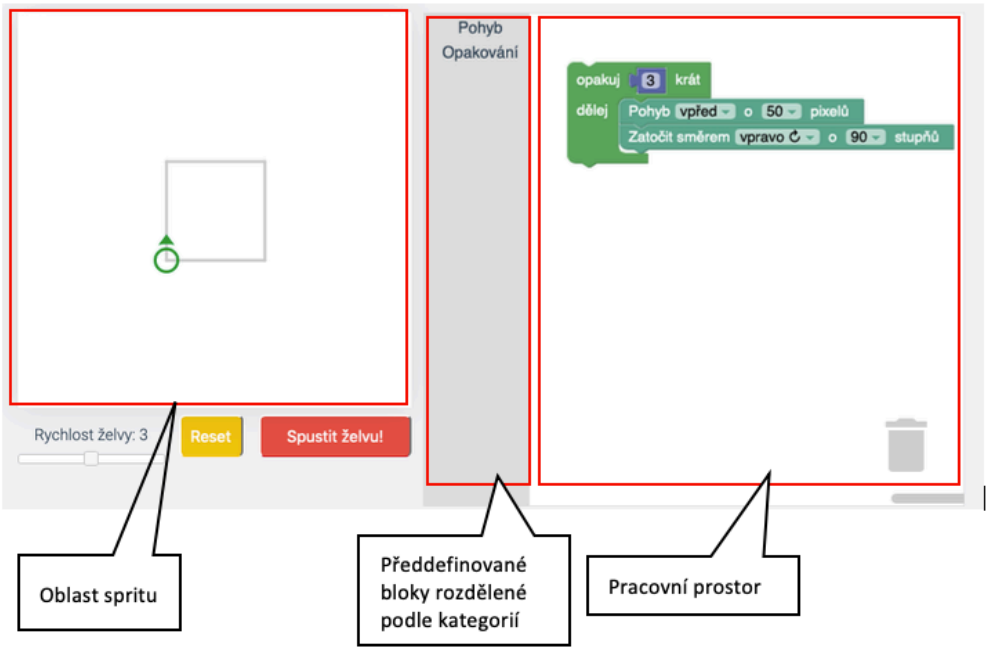
Tento nástroj jako jediný, z již zmíněných patří mezi placené. Jeho cena je pro středně velkou školu, tj. pro max. 200 žáků 3 199 Kč na rok. Pokud si škola nástroj zaplatí, může využívat portál pro učitele a funkce pro správu tříd.

Prostředí

Želví grafika je podobná předchozím nástrojům, viz Obrázek 8. Podstatným rozdílem je didaktika využití nástroje. Uživatel plní zadání od vývojářů nástroje. Ty se dle zadané úlohy liší.

Prostředí je v českém jazyce a lze rozložit na tři části, kterými jsou:

- Oblast spritu (želvy)
- Bloky k použití jsou v některých příkladech rozděleny podle kategorií
- Pracovní prostor, kde se bloky skládají



Obrázek 8 - Prostředí želví grafiky, zdroj autor

Funkce

Principem želví grafiky je tvoření obrazců pomocí želvy a předdefinovaných bloků. Úkolem je například dodělání čtverce, písmena M, dokreslení pětiúhelníku, spirál a dalších.

Principem nástroje je však také používání matematiky při řešení úloh, například práci s úhly, počítání vzdáleností, Pythagorovu větu, goniometrické funkce nebo fraktály.
V případě že by si žák nevěděl rady, může si zobrazit nápovědu. V pokročilejších příkladech je pak již nutná práce s proměnnými, cykly nebo podmínkami.

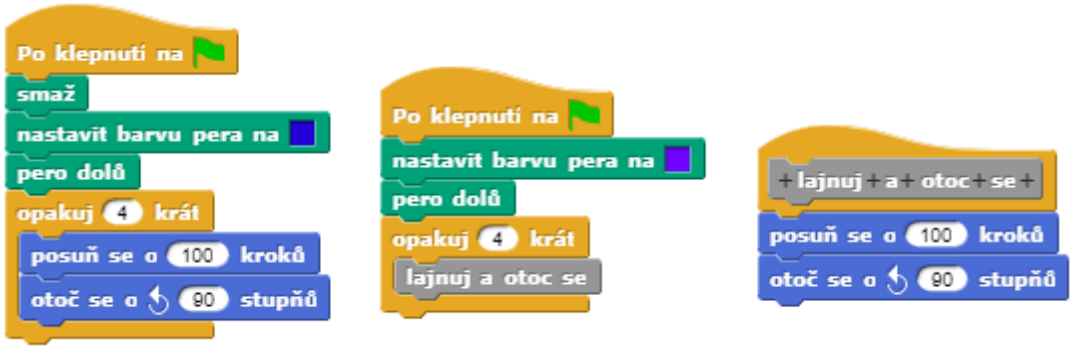
Podpora pro výuku a komunita

Podobně jako u ostatních nástrojů si může učitel vytvářet své třídy při zaplacení licence. Nástroj umožňuje učiteli možnost zadávání domácích úloh žákům, které systém automaticky zkontroluje. Nástroj také učitelům slibuje širokou paletu cvičení. Je možné si prohlížet výsledky žáků.

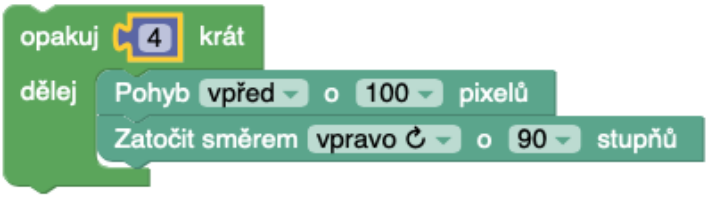
4.4 Tvorba vybraných algoritmů v online nástrojích

4.4.1 Kreslení čtverců

Nejjednodušším příkladem, který je v této práci uveden je kreslení jednoduchého čtverce. Kreslení realizováno pomocí iterace a procedury, která zajišťuje posun spritu a otočení o 90 stupňů. Šlo by také opakovat příkaz „opakuj“ a „otoč se“ několikrát po sobě. Obě varianty jsou demonstrovány na Obrázek 9. Pro poslední zmíněný postup však Želví grafika nenabízí vhodné bloky. Je třeba opravit hodnoty v blocích, které jsou sestavené.



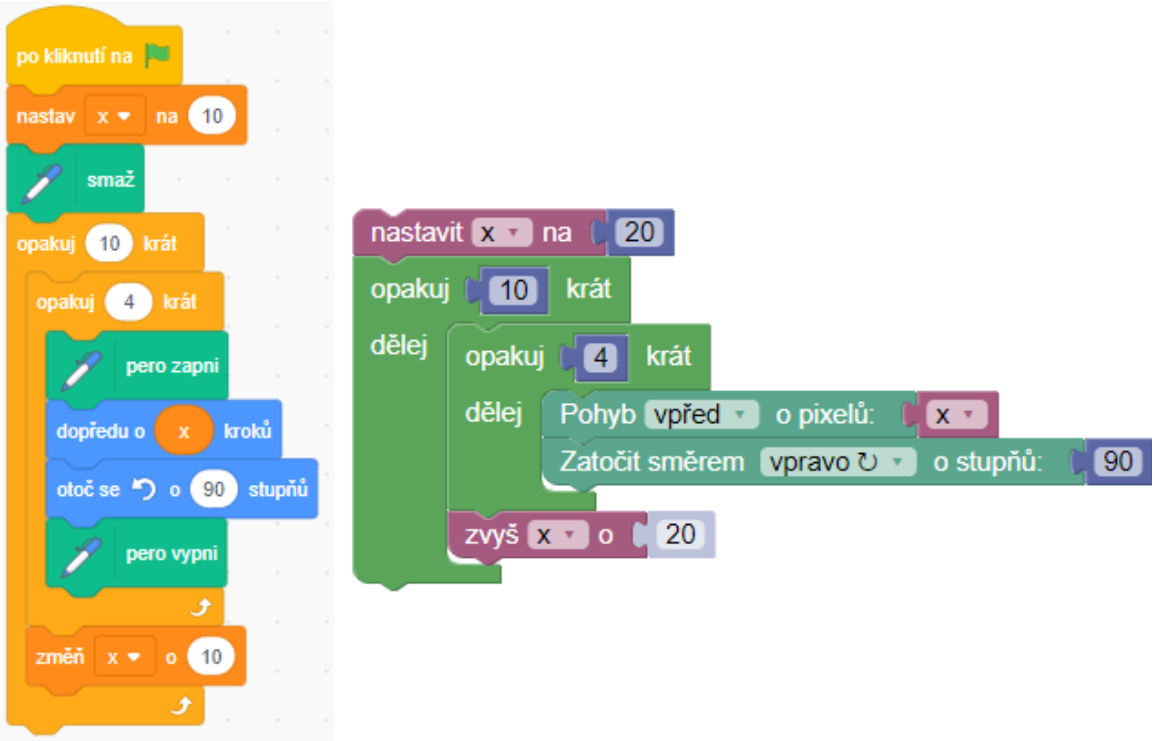
Obrázek 9 - Kreslení čtverců pomocí Snap! pomocí iterace a pomocí procedury, zdroj autor



Obrázek 10 - Kreslení čtverce v Želví grafice, zdroj autor

4.4.2 Keslení vnořených čtverců

Následující příklad již vyžaduje využití práce s vnořenými iteracemi a proměnnými. Pro ukázku je uvedeno řešení ve Scratch a Želví grafice. Vnořená iterace řeší kreslení jednoho čtverce, vnější zajišťuje počet vykreslených čtverců. Čtverce jsou zvětšovány pomocí jednoduché číselné proměnné x. V Želví grafice je po otevření zadání již program sestaven a stačí jen opravit velikost kreslené čáry a vzdálenost čtverců od sebe.



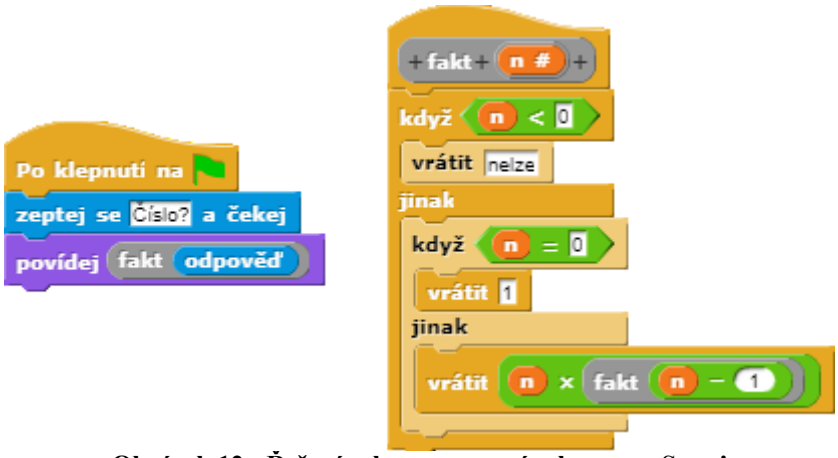
Obrázek 11 - Řešení vnořených čtverců ve Scratch a Želví grafice, zdroj autor

4.4.3 Výpočet faktoriálu čísla

Faktoriál čísla n je číslo, které je rovné součtu všech kladných celých čísel menších nebo rovných n, pokud je n kladné (Wikipedia, 2019). Matematicky vyjádřeno jako $n! = 1*2...n$. V rámci algoritmizace lze využít dvě možnosti, jak získat výslednou hodnotu. První možností je využití iterace a uložení posledního výpočtu do složeného datového typu pole, nebo elegantněji pomocí rekurze. V obou případech je však nutné zajistit, aby zadané číslo nebylo záporné a výsledek 0! je 1.

Rekurze je opakované volání funkce ve funkci, jak je uvedeno v analytické části. Podprogram vždy vrátí hodnotu čísla faktoriálu, které je o jedna menší. To se děje, než skončí u čísla 1, kdy je vrácen jako výsledek 1. Pak se funkce vrací zpět a vždy vynásobí vyšší hodnotu čísla s menší. Výhodou jazyka Snap! je možnost vrácení hodnoty jako datový typ číslo. Řešení v jazyce Snap! je uvedeno na Obrázek 12. Scratch však tuto

možnost nemá, a je tak nutné si vždy přechozí hodnotu rekurze uložit například do složené datové struktury – zásobník. Řešení je uvedeno na Obrázek 13.



Obrázek 12 - Řešení rekurze pomocí rekurze ve Snap!

Obrázek 13 - Řešení rekurze ve Scratch pomocí iterace a zásobníku

5 Výsledky a diskuse

5.1 Charakteristika online nástrojů

Jelikož všechny nástroje běží v online prostředí, na hardwarových specifikacích zařízení koncového uživatele záleží minimálně. Naopak velmi záleží na síle připojení i internetu.

V dnešní době podíl mobilních zařízeních roste, proto byly nástroje zběžně vyzkoušeny na mobilním telefonu a tabletu. Všechny nástroje otevřely svá uživatelská rozhraní kromě Snap!, kde optimalizace pro ovládání dotykem chybí a přesunout blok bylo téměř možné. Na ostatních nástrojích se s nimi pracovat dalo, ačkoliv těžko a práce není tak efektivní jako na PC či notebooku.

Snap! je bezesporu funkcionálně nejvybavenější z testovaných nástrojů. Jako jediný podporuje práci s funkcemi s návratovou hodnotou a umí pracovat i s cyklem s předem známým počtem opakování. Obsahuje také český překlad. Ten však chybí u některých externích knihoven. Jako u všech ostatních nástrojů v něm lze demonstrovat pohyb spritu, na kterém lze krásně ukázat posloupnost jednotlivých příkazů.

Code.org je naopak z hlediska funkcionality nejchudším. Proto ani nemohl být v kapitole 4.4 využit. Chybí zde například možnost zaznamenání pohybu čárou, na kterém se většina z demonstrovaných příkladů zakládala. Ačkoliv je zde možnost výběru českého jazyka, velká část bloků není přeložena vůbec.

Scratch je velmi podobný Snap!, jelikož Snap! z jazyka Scratch vychází. Funkcionálně jsou si spolu také velmi podobné, oba umí například pracovat se složitějšími datovými strukturami. Výhodou Scratch je obrovská komunita, která jej využívá. Je dostupných mnoho příkladů, ze kterých se mohou učitelé inspirovat.

Jediný nástroj, který se od principu předchozích mírně odlišuje je Želví grafika. U ostatních si uživatel může „*dělat co chce*“. Principem Želví grafiky je plnit zadaná cvičení. Některá cvičení jsou velmi jednoduchá a jsou vhodná i pro mladší žáky. Lze nalézt i složitá zadání, která vyžadují znalosti matematiky a geometrie.

5.2 Analýza požadavků na online nástroje

Z průzkumu Rámcových vzdělávacích programů byly zjištěny funkce, které obsahují online nástroje pro výuku algoritmizace, viz. Tabulka 4.

	Snap!	**Scratch**	**Code.org**	**Želví grafika**
Selekce (if, když)	Ano	Ano	Ano	Ano
Iterace (do, while)	Ano	Ano	Ano	Ano
Iterace s předem známým počtem opakování	Ano	Ne	Ne	Ne
Jednoduché datové typy	Ano	Ano	Ano	Ano
Složené datové typy (pole)	Ano	Ano	Ne	Ne
Procedury	Ano	Ano	Ano	Ne
Funkce	Ano	Ne	Ne	Ne
Pohyb objektu	Ano	Ano	Ano	Ano
Import rozšiřujících knihoven	Ano	Ano	Ne	Ne
Podpora zvukových výstupů	Ano	Ano	Ne	Ne
Podpora pro výuku	Ne	Ano	Ano	Ano
Předklad do češtiny	75%	100%	50%	100%
Cena	Zdarma	Zdarma	Zdarma	Tisíce korun
Učitelský účet pro správu studentů	Ne	Ano	Ano	Ano

Tabulka 4 - Přehled funkcionalit jednotlivých nástrojů, zdroj autor

5.3 Zhodnocení online nástrojů

Výsledky byly zhodnoceny pomocí vícekriteriální analýzy variant bodovací metodou s váhami, viz. Tabulka 5.

Funckionalitám byly přiděleny váhy kritérií. Základní funkcionality mají větší váhu než funkcionality, které výuku jen zpestřují nebo pomáhají učitelům. Kritéria byla obodována a to následovně: pokud nástroj funkcionalitu obsahuje – 10 bodů, funkcionalitu neobsahuje – 1 bod, obsahuje ji jen z části – poměrný počet bodů.

Z výsledků lze říci, že je Snap! nejlepším nástrojem pro výuku algoritmizace. Obsahuje všechny základní funkcionality popsané v kapitole 4.2. Scratch některé základní funkcionality nepodporuje, a proto skončil se na druhém místě. Zbývající dva nástroje obsahují potřebných funkcionalit velmi málo. Ačkoliv se Želví grafika vydává jiným směrem, a tak ve stanovených kritériích neobstála, nelze však říct že je nejhorší.

Bodovací metoda s váhami	Snap!	Scratch	Code.org	Želví grafika	Váha kritéria
Selekce (if, když)	10	10	10	10	0,12
Iterace (do, while)	10	10	10	10	0,12
Iterace s předem známým počtem opakování	10	0	0	0	0,1
Jednoduché datové typy	10	10	10	10	0,12
Složené datové typy (pole)	10	10	0	0	0,1
Procedury	10	10	10	0	0,1
Funkce	10	0	0	0	0,1
Pohyb objektu	10	10	10	10	0,04
Import externích knihoven	10	10	0	0	0,04
Podpora zvukových výstupů	10	10	0	0	0,02
Podpora pro výuku	0	10	10	10	0,04
Překlad do češtiny	7	10	5	10	0,04
Zpoplatnění	10	10	10	0	0,02
Učitelský účet pro správu studentů	0	10	10	10	0,04
Skalární součin	9,08	8	6,2	5,2	1
Pořadí	1	2	3	4	

Tabulka 5 - Výsledek vícekriteriální analýzy variant, zdroj autor

6 Závěr

V práci byly vyhledány a hodnoceny nejpoužívanější online nástroje pro výuku algoritmizace. Jsou jimi Snap, Scratch, Code.org a Želví grafika.

Byly analyzovány školní vzdělávací programy vybraných škol. Školy byly vybrány tak, aby bylo možné jejich programy porovnat z hlediska přístupu k výuce algoritmizace. Byly vybrány následující typy škol: základní, střední odborné a střední všeobecné.

Analýzou školních vzdělávacích programů byla zjištěna hlavní kritéria, podle kterých byly nástroje následně zhodnoceny. Důležitými kritérii jsou práce s jednoduchými a složenými datovými typy, funkce, procedury a základní programové struktury (selekce a iterace).

Nástroj Snap! je na základě hodnocení pro výuku nejvhodnější, jelikož obsahuje nejvíce funkcionalit z vytvořených kritérií. Scratch sice neobsahuje některé funkcionality, ale ty je možné nahradit jiným postupem řešení problému. Scratch výuku podporuje svou rozsáhlou komunitou a množstvím sdílených projektů. Code.org spolu s Želví grafikou obsahují funkcionalit nejméně. Želví grafika se na zadaných kritériích umístila na posledním místě, jelikož se didaktika využití nástroje zcela liší od ostatních.

Práce byla zaměřena na nástroje online, ale existuje mnoho dalších nástrojů dostupných ke stažení. Pouze v rámcovém vzdělávacím programu pro obor Informační technologie bylo zjištěno, že se žáci měli zabývat objektovým modelováním. Přístup Object First, využívající se v cizích zemích, začíná právě objektovým modelováním, a to se stalo celosvětově nejpoužívanějším přístupem k programování. (Pecinovský, 2018). Námětem na diplomovou práci může být právě analýza nástrojů, které umožňují výuku objektově orientovaného programování.

7 Seznam použitých zdrojů

Balík, Miroslav. 2014. [Online] 2014. [Citace: 14. 11 2018.] https://edux.fit.cvut.cz/courses/BI-PA1/_media/lectures/04/pa1-04-ifacykly.pdf.

Bayer, Tomáš. [Online] [Citace: 06. 11 2018.] https://web.natur.cuni.cz/~bayertom/images/courses/Prog2/programovani2.pdf.

Běhálek, Ing. Marek. [Online] [Citace: 14. 11 2018.] http://wiki.cs.vsb.cz/images/4/4c/Progjaz.pdf.

Bechyňová, Marta . 2012. STRÁNKY K VÝUCE INFORMATIKY. *STRÁNKY K VÝUCE INFORMATIKY.* [Online] 03. 10 2012. [Citace: 06. 11 2018.] http://www.ivt.mzf.cz/algoritmizace-a-programovani/uvod-do-algoritmu/4-vyvojove-diagramy/.

Český normalizační institut. 1995. ČSN ISO 5807 Zpracování informací: dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy a diagramy zdrojů systému. 1995.

Dvořáková, Helena. 2015. Výhody a nevýhody ukládání dat do cloudu. *coolzine.cz.* [Online] 05. 04 2015. http://coolzine.cz/vyhody-a-nevyhody-ukladani-dat-do-cloudu/.

Gymnázium Českolipská. Gymnázium Českolipská. *Školní vzdělávací program gymnázia Českolipská.* [Online] [Citace: 21. 01 2019.] http://www.ceskolipska.cz/data/ke-stazeni/svp.pdf.

Gymnázium J. Vrchlického v Klatovech. Gymnázium J. Vrchlického v Klatovech. *Školní vzdělávací program gymnázia J. Vrchlického v Klatovech.* [Online] [Citace: 21. 01 2019.] https://klatovynet.cz/gymkt/user/skola/svp-vyssi.pdf.

Humpolcová, Tereza Čechová. 2012. Metodický portál RVP. *Metodický portál RVP.* [Online] 26. 01 2012. [Citace: 2019. 01 01.] https://spomocnik.rvp.cz/clanek/14905/SCRATCH-VE-%20VYUCE.html.

Jeřábek, Jaroslav a Tupý, Jan. 2017. Rámcový vzdělávací program pro základní vzdělání. *Ministarstvo školství, mládaže a tělovýchovy.* [Online] 15. 09 2017. [Citace: 22. 01 2019.] http://www.msmt.cz/file/43792/.

Jeřábek, Jaroslav, Krčková, Stanislava a Hučnínková, Lucie. 2013. *Rámcový vzdělávací program pro gymnázia.* Praha : Výzkumný ústav pedagogický v Praze, 2013. 978-80-87000-11-3.

Kolesná, Dana. 1986. *Základy algoritmizace a programové vybavení.* místo neznámé : Tesla Eltos, 1986.

Kubátová, Eliška. 2015. E15. [Online] 19. 10 2015. [Citace: 2019. 01 01.] https://www.e15.cz/magazin/deti-se-maji-ucit-programovat-jinak-tezko-najdou-praci-tvrdi-experti-1237679.

Mikláš, Michal. 2018. Informatika a grafika. *Informatika a grafika.* [Online] 06. 11 2018. http://www.gjszlin.cz/ivt/esf/algoritmizace/algoritmus.php.

Ministertsvo školství, mládeže a tělovýchovy. 2015. MŠMT. *MŠMT.* [Online] 22. 07 2015. http://www.msmt.cz/dokumenty/bila-kniha-narodni-program-rozvoje-vzdelavani-v-ceske-republice-formuje-vladni-strategii-v-oblasti-vzdelavani-strategie-odrazi-celospolecenske-zajmy-a-dava-konkretni-podnety-k-praci-skol.

Národní ústav odborného vzdělávání. 2008. Rámcový vzdělávací program pro obor vzdělání informační technologie. *Národní ústav odborného vzdělávání.* [Online] 29. 05 2008. [Citace: 21. 01 2019.] http://zpd.nuov.cz/RVP/ML/RVP%201820M01%20Informacni%20technologie.pdf.

—. 2007. Rámcový vzdělávací program pro obor vzdělání obchodní akademie. *Národní ústav odborného vzdělávání.* [Online] 28. 06 2007. [Citace: 21. 01 2019.] http://zpd.nuov.cz/RVP/ML/RVP%206341M02%20Obchodni%20akademie.pdf.

Národní ústav pro vzdělávání. Národní ústav pro vzdělávání. [Online] http://www.nuv.cz/t/rvp.

Obchodní akademie Praha, s. r. o. Školní vzdělávací program Obchodní akademie Praha, s. r. o. *OA Praha.* [Online] [Citace: 21. 01 2019.] https://www.oapraha.cz/sites/default/files/OA_SVP_2017-18.pdf.

Obchodní akademie Tomáše Bati. 2009. Školní vzdělávací program Obchodní akademie Tomáše Bati. *OA Zlín.* [Online] 25. 08 2009. [Citace: 21. 01 2019.] http://www.oazlin.cz/wcd/docs/svp/svp_oa.pdf.

Pecinovský, Rudolf. 2001. Česká škola. [Online] 10. 09 2001. [Citace: 2019. 01 01.] http://www.ceskaskola.cz/2001/09/rudolf-pecinovsky-proc-ucit.html.

—. 2018. Současné trendy v metodice výuky programování. *Researchgate.* [Online] 2018. https://www.researchgate.net/publication/228650522_Soucasne_trendy_v_metodice_vyuky_programovani.

Pelánek, Radek. 2018. *Želví grafika Exkurze do programování, geometrie a umění.* Brno : Computer Press, 2018. 978-80-251-4905-8.

Polanský, Dušan. 2012. Dušan Polanský. *Dušan Polanský.* [Online] 11. 03 2012. [Citace: 2019. 01 01.] http://dusanpolansky.cz.

Pšenčíková, Ing. Jana. 2009. *Algoritmizace.* místo neznámé : Computer Media s.r.o., 2009. 978-80-7402-034-6.

RVP. 2015. Metodický portál RVP. *Metodický portál RVP.* [Online] 02. 11 2015. [Citace: 2019. 01 21.] https://digifolio.rvp.cz/view/artefact.php?artefact=70545&view=10429&block=57827.

Scratch. Scratch. [Online] https://scratch.mit.edu/about.

—. Scratch statistiky. *Scratch.* [Online] https://scratch.mit.edu/statistics/.

Snap! Snap! [Online] https://snap.berkeley.edu/about.html.

SPŠ Prosek. Školní vzdělávací program SPŠ Prosek. *SPŠ Prosek.* [Online] [Citace: 21. 01 2019.] http://www.sps-prosek.cz/soubory/obory/SVP_2017/WEB/4_SVP_IT_2017.pdf.

SPŠE V Úžlabině. 2016. Školní vzdělávací program SPŠE V Úžlabině. *SPŠE V Úžlabině.* [Online] 17. 06 2016. [Citace: 21. 01 2019.] https://www.uzlabina.cz/uploads/file/SVP_ITE_2016_minimalizace.pdf.

Staňková, Jana a Staněk, František. 1998. *Vytváření a realizace algoritmů.* Ostrava : VŠB - Technická univerzita Ostrava, 1998. 80-7078-506-3.

Vaníček, Jiří, a další. 2007. *Teoretické základy informatiky.* Praha : Kernberg Publishing, 2007. 978-80-903962-4-1.

Virius, Miroslav. 1995. *Základy algoritmizace.* 1995. 978-80-01-04003-4.

Wikipedia. 2019. Faktoriál. *Wikipedia.* [Online] 01. 01 2019. (https://cs.wikipedia.org/wiki/Faktoriál.

—. 2019. Scratch (programming language). *Wikipedia.* [Online] 24. 01 2019. https://en.wikipedia.org/wiki/Scratch_(programming_language).

—. 2019. Snap! (programming language) from Wikipedia, the free encyclopedia. *Wikipedia.* [Online] 20. 01 2019. https://en.wikipedia.org/wiki/Snap!_(programming_language).

Wróblewski, Piotr. 2004. *Algoritmy Datové struktury a programovací techniky.* Brno : Computer Press, 2004. 80-251-0343-9.

ZŠ Glowackého. ZŠ Glowackého. *Škokní vzdělávací program ZŠ Glowackého.* [Online] [Citace: 21. 01 2019.] https://www.glowackeho.cz/vyuka/.

ZŠ Máchovka. ZŠ Máchovka. *Škokní vzdělávací porogram ZŠ Máchovka.* [Online] [Citace: 21. 01 2019.] http://www.machovka.cz/RV_IT/.

ZŠ Petřiny - jih. ZŠ Petřiny - jih. *Škokní vzdělávací program ZŠ Petřiny - jih.* [Online] [Citace: 21. 01 2019.] http://www.petrinyjih.cz/stranka-rozsirena-vyuka-informatiky-66.

8 Přílohy

Tabulka verzí zmíněných nástrojů

Snap!	4.2.2.9
Scratch	3.0
Code.org	nezjištěno
Želví grafika	nezjištěno

Tabulka 6 - Tabulka verzí

9 Seznam obrázků

10 Seznam tabulek