Trnavská univerzita v Trnave

Pedagogická fakulta Katedra matematiky a informatiky

Veronika Gabaľová

Detské programovacie jazyky a ich využitie v pedagogickej praxi

TRNAVA 2022

Recenzenti: RNDr. Júlia Tomanová, PhD.
Mgr. Peter Pšenák, PhD.

Jazyková korektúra: Mgr. Zuzana Kupcová
Grafická úprava: PaedDr. Veronika Gabaľová, PhD.

ISBN 978-80-568-0510-7

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

V dnešnej dobe je extrémne rýchly vývoj v oblasti informačno-komunikačných technológií už prakticky nezastaviteľný, a taktiež je nezastaviteľný aj prienik informačno-komunikačných technológií do úplne všetkých oblastí bežného života. Tento fakt udáva aj smer, kam je potrebné zamerať pozornosť aj vo výchovno-vzdelávacom procese tak, aby si aj najmladšia generácia osvojovala zručnosti a návyky, ktoré sú nevyhnutné pre ich efektívne zvládnutie. Jednou z ponúkaných možností kam sa pri edukácii orientovať, je zamerať sa na rozvoj schopnosti algoritmického myslenia u detí a mládeže. Avšak aj toto je proces prakticky nekonečný, pretože tieto schopnosti je potrebné neustále zdokonaľovať a poskytnúť im priestor progresívne napredovať.

Aj vďaka rozvoju informačno-komunikačných techník sa mnohí ľudia každého veku a sociálneho postavenia neustále učia a zdokonaľujú vo využívaní výpočtovej techniky. Čím ďalej, tým viac mladšia generácia detí sa s ňou a s jej vymoženosťami dostáva do každodenného kontaktu. Dnes je už považované za samozrejmosť, že generácia detí, ktoré opúšťajú lavice základnej školy, vedia používať bežné aplikačné programy, akými sú grafické, textové, tabuľkové či prezentačné softvéry a dokonca vedia vytvoriť jednoduchú webovú stránku. Komunikácia prostredníctvom technológií je bežná aj medzi mladšími deťmi a práca s internetom tiež. Je preto považované za samozrejmosť, že moderné techniky sú nasadzované a využívané aj v školstve, a tak dochádza aj k jeho modernizácii (Pšenáková, 2021). Veď už aj samotné učebné osnovy z predmetu informatika pre žiakov základných škôl si dávajú za cieľ sprístupniť základné pojmy a techniky používané pri práci s údajmi či tvorbe algoritmov a výpočtových procesov. Jedným zo základných modulov týchto učebných osnov je možnosť sprostredkovať vzdelanie v oblasti algoritmického riešenia problémov a zdokonaľovať algoritmické myslenie prostredníctvom tvorby programov a práce v detských programovacích jazykoch.

Pri spracovaní publikácie sa autorka riadila svojimi dlhoročnými skúsenosťami z oblasti vyučovania základov programovania v detských programovacích jazykoch a tiež využila vedomosti týkajúce sa tejto problematiky. Počas pedagogickej praxe sa autorka často stretávala s otázkou, či je potrebné a dôležité deti na základných školách učiť algoritmizovať a programovať. Na základe vlastných skúseností je možné konštatovať, že takto sformulovanú otázku kladú zvyčajne len tí, ktorí sa v danej problematike orientujú len veľmi málo, alebo vôbec. Nasledujúce strany sú určené aj pre takýchto čitateľov.

1 Algoritmizácia verzus programovanie

Pojem algoritmus Slovník cudzích slov definuje ako „presný a logicky jednoznačný predpis na vykonanie určitej sústavy operácii, pričom je určené aj poradie riešenia úloh daného typu“ (Ivanová-Šalingová, 1993). Jedna z ďalších definícií algoritmu je spojená s menami a prácou prvých didaktikov informatiky na Slovensku: Drlíkom a Hvoreckým (1992), ktorí algoritmus chápu ako predpis, prípadne návod, ktorého realizovaním získame zo zadaných vstupných údajov po vykonaní konečného počtu krokov očakávané výsledky. Môžeme teda skonštatovať, že sa jedná o určitý, konkrétny postup, ktorého vykonaním sa zabezpečí jednoznačná realizácia deja alebo činnosti. Aby sme vedeli navrhnúť správny postup, musíme definovať, kde sa práve nachádzame a kam sa chceme dostať. Rovnako je potrebné vedieť, aj aké prostriedky môžeme využiť a vedieť posúdiť, ktoré kroky nás vedú k naplneniu cieľov a ktoré nevedú. Prepisom riešenia zadaní do algoritmického jazyka sa dieťa učí novému spôsobu myslenia. Môžeme skonštatovať, že aj v zápise riešenia problémov pomocou algoritmov sa jedná o takú metódu riešenia, ktorá sa dá naučiť a pomáha v existencii každému človeku bez ohľadu na to, či sa zaoberá návrhom kódov programov alebo nie. Drlík a Hvorecký (1992) ponímajú algoritmizáciu ako propedeutiku programovania, teda nejaký základný moment v tvorbe programov ako takých. Vo všeobecnosti môžeme predpokladať, že programovať sa veľmi ťažko naučí, či skôr sa vôbec nenaučí ten, kto nemá zvládnuté základné logické elementy algoritmizácie, ako sú základy analýzy, syntézy, indukcie, dedukcie a komparácie. Aj kvôli tomuto faktu je potrebné zaviesť pre formuláciu algoritmov charakteristický jazyk.

Algoritmický jazyk pozostáva z operačnej a riadiacej zložky. Operačnú zložku môžeme charakterizovať ako prostriedok, ktorý umožňuje spracovať údaje. Riadiacou zložkou sú prostriedky pre riadenie postupnosti vykonávania jednotlivých činností algoritmu. Základné riadiace konštrukcie z hľadiska postupu vykonania jednotlivých krokov môžeme rozdeliť do troch kategórií. Prvou kategóriu je sekvencia, čiže postupnosť príkazov, druhou kategóriou je podmienený príkaz, nazývaný aj vetvenie, ktorého realizácia závisí od splnenia či nesplnenia podmienky a treťou kategóriu je cyklus, čo je vlastne viacnásobné opakovanie istej činnosti, opakovanie ktorej je ukončované vopred známou podmienkou. (Stoffová & Czakoová, 2016)

Podľa Drlíka (1995) v tom okamihu, keď je algoritmus prevedený do zrozumiteľného tvaru pre počítač, stáva sa z neho program. Programovanie je podľa neho schopnosť naformulovať postupy riešenia zadaní problémov tak, aby boli riešiteľné pomocou počítačov.

Aj napriek tomu, že informatika je pomerne nová vedná disciplína, jej vznik siaha do 80. rokov 20. storočia. Rovnako aj história vývoja programovacích jazykov siaha do druhej polovice dvadsiateho storočia, čiže tiež do pomerne nedávnej histórie. Za toto obdobie však vznikli stovky programovacích jazykov, z ktorých niektoré zanikli, niektoré sa neustále rozvíjajú a ich verzie sa neustále zlepšujú, zjednodušujú, ale najmä prispôsobujú dnešnej dobe. Zároveň počas celej doby vznikajú nové programovacie jazyky v závislosti od špecifických požiadaviek techniky a vzrastajúcich poznatkov a skúseností programátorov. Špeciálnu skupinu tvoria edukačné programovacie jazyky, ktoré sú určené na výučbu programovania a algoritmického myslenia. V tejto súvislosti sa často hovorí o detských programovacích jazykoch, mini-jazykoch, či mikrosvetoch (Stoffová & Czakoová, 2016). Hedvigová (2006) uvádza, že

mikrosvety sú programovacie prostredia poskytujúce prostriedky na skúmanie konkrétnej témy alebo oblasti. Používateľ je ten, ktorý v nich samostatne objavuje existujúce súvislosti a zákonitosti, čo mu umožňuje si tieto ľahšie pamätať. „*Ak chceme zovšeobecniť, čo pod mikrosvetom, či detským programovacím jazykom rozumieme, môžeme povedať, že: učia základné postupy pri programovaní využiteľné aj pri práci s vyššími jazykmi; sú jednoduché, ľahko ovládateľné, názorné, zaujímavé; deti ovládajú nejaký objekt (postavičku čarodejníka, korytnačky, robota, zajačika...) pomocou príkazov, ktoré sú často podobné bežnému jazyku; umožňujú spájanie jednoduchých príkazov do zložitejších celkov*.“ (Gabaľová, 2008)

2 Detské programovacie jazyky

Detský programovací jazyk, malý programovací jazyk, či mikrosvet sú pojmy, ktorými je označovaný taký programovací jazyk, ktorý má v porovnaní s programovacími jazykmi používanými profesionálnymi programátormi menšiu množinu možných príkazov, jednoduchšiu syntax a často aj grafický (ikonický) spôsob zápisu príkazov, a preto je detský programovací jazyk vhodným jazykom na vyučovanie základov programovania.

Detské programovacie jazyky však nemôžeme považovať za univerzálne, pretože sa sústreďujú na sprístupnenie len určitého programátorského princípu a zvyčajne sú tematicky zamerané na tvorbu programov zo špecifickej oblasti.

Prvýkrát sa s detskými programovacími jazykmi stretávame už v 80-tych rokoch minulého storočia. Prostredníctvom prídavného modulu k osembitovým počítačom PMD 85 sa mohli žiaci stretnúť s detským programovacím jazykom Karel. Následne s nástupom osobných počítačov a operačného systému MS DOS sa mohli stretnúť s programovacími jazykmi Logo, SGP.

Programovací jazyk Logo bol predchodcom dnes ešte stále pomerne rozšíreného programovacieho jazyka Imagine a programovací jazyk SGP bol predchodcom dnes používaného programovacieho jazyka Baltík 3.0 či Baltie 4C#.

Detské programovacie jazyky môžeme zaradiť medzi vizuálne intuitívne jazyky. Na jednej strane sú to veľmi jednoduché, ale súčasne silné nástroje na uvedenie žiakov do spoznania programovacích jazykov. Sú dobrým základom pre vyučovanie algoritmizácie a programovania vo vyšších programovacích jazykoch, ako sú napríklad Delphi/Lazarus, C a Python. Detské programovacie jazyky poskytujú pre žiaka primárne prostredie pre prvé kroky tvorby programov a učia ho algoritmickému mysleniu.

Detské programovacie jazyky tiež poskytujú pevný základ pre systematické riešenie problému aj pre tých záujemcov, ktorí chcú tvoriť algoritmy len na úrovni aplikačných programov.

Od začiatku zavádzania počítačov do edukácie bolo vynaložené množstvo úsilia na vytvorenie špeciálnych jazykov, ktoré by slúžili najmä pre žiakov základných a stredných škôl na získanie prvotných programátorských zručností. Už z obdobia osembitových mikropočítačov si môžeme pamätať Robota Karla, ktorého modul bol pripojiteľný napríklad k počítačom PMD 85. Avšak pravé riešenie podstaty vyučovania základov programovania môžeme datovať do momentu vzniku programovacieho jazyka Logo, ktorý je do dnešných čias aj s jeho inovovanými verziami stále spájaný s pojmom korytnačia grafika. Úspech Loga vo všeobecnosti, ale aj korytnačej grafiky ako takej, bol počiatočným stimulom rozvoja mini-jazykov pre výučbu princípov programovania (Papert, 1980).

Prvotnou ideou vzniku detských programovacích jazykov bolo podľa Paperta (1980) poskytnúť jednoduchý jazyk, ktorý by bol vhodný pre položenie základov výučby programovania u detí. Detské programovacie jazyky majú svoju ústrednú postavičku, objekt, ktorý žiak pomocou jednoduchých príkazov ovláda. Ústrednou postavičkou, takzvaným objektom je napríklad korytnačka, robot, čarodejník alebo iná, v prevažnej miere, živá bytosť. Objektom môže, byť aj personifikovaný stroj, ktorý sa pohybuje vo svojom obmedzenom priestore, mikrosvete.

Detské programovacie jazyky najčastejšie pracujú v dvoch režimoch:

1. interaktívnom, v ktorom žiak priamo príkazmi ovláda ústrednú postavičku detského programovacieho jazyka,
2. programovacom, v ktorom žiak po návrhu a zapísaní kódu na riešenie problému program spustí. Vďaka vizualizácii príkazov na obrazovke počítača môže žiak priamo pozorovať správanie sa objektu vo svojom prostredí, a zistiť, či ním vytvorený algoritmus vykonáva správne zadanú úlohu.

Detský programovací jazyk obsahuje príkazy a operácie na riadenie objektu, teda svojej ústrednej postavičky. Väčšina detských programovacích jazykov obsahuje základné riadiace štruktúry sekvenciu, vetvenie a cyklus ako aj mechanizmus na vytváranie nových zložených príkazov a podprogramov. Termínom detské programovacie prostredia označujeme kombináciu objektu a jazyka, prvkami ktorého je tento objekt ovládaný.

2.1 Didaktické aspekty použitia detských programovacích jazykov

Základom zápisov algoritmov v prostredí vhodných a veku primeraných programovacích jazykov je téma, s ktorou by sa žiak mal stretávať už od prvého kontaktu so školou. Algoritmizácia a programovanie predstavujú základ pre logické i abstraktné myslenie, čo je dôležitý moment v primárnej edukácii.

Výhodou detských programovacích jazykov, ako to vyplýva už zo samotného názvu, je, že sú vhodné aj pre najnižšiu vekovú kategóriu, teda pre deti. Tieto jazyky majú ohraničenú syntax a zvyčajne aj jednoduchú sémantiku. Zápis algoritmov v detských programovacích jazykoch je vizualizovaný. Ich základnou vlastnosťou je jednoduchosť a zrozumiteľnosť. Deti, žiaci 1. stupňa základnej školy si ich dokážu ľahko a za pomerne krátky čas osvojiť a následne použiť na riešenie rôznych úloh. Osvoja si tak tajomstvá známych algoritmických princípov a zásad programovania.

Ďalšou pomerne dôležitou výhodou detských programovacích jazykov je, že sú postavené na metaforách, čím umožňujú vytvárať bohaté množiny rôznych problémov súvisiacich so skúsenosťami z bežného života žiakov.

Vizualizácia operácií vykonávaných ústrednou postavičkou v tom-ktorom detskom programovacom jazyku pomáha odhaľovať sémantiku jazyka i jeho jazykových konštrukcií. Práve vizualizácia môže žiakom pomôcť pochopiť sémantiku predkladaných a ústrednou postavičkou realizovaných konštrukcií, zároveň môže byť nápomocná v objasňovaní princípoch realizácie programových štruktúr a tiež zabraňovať vzniku a množeniu chýb. Vizualizácia objektov podporuje aj bádateľsky orientované vyučovanie, ktoré patrí k aktivizujúcim vyučovacím metódam.

3 Detské programovacie jazyky

Vývoj detských programovacích jazykov a mikrosvetov bol v osemdesiatych rokoch minulého storočia silne ovplyvnený korytnačou grafikou Loga. V istom zmysle môžeme programovací jazyk Logo považovať za prvý príklad detského programovacieho jazyka aj napriek tomu, že tento jazyk nebol vyvinutý špeciálne za účelom vyučovať programovanie, ale autori sa skôr sústredili na rozvíjanie myslenia. Práve toto obdobie je možné považovať za obdobie prudkého rozvoja vizualizovaných programovacích jazykov, akými mikrosvety a detské programovacie jazyky boli.

Rovnako ako vyššie programovacie jazyky aj detské programovacie jazyky prešli historickým vývojom. Spočiatku to boli jednoduché jazyky obsahujúce niekoľko príkazov, ktoré boli často zapisované jednoduchými skratkovými slovami. Neskôr sa tvorcovia snažili jazyk zasadiť do grafického prostredia, ktoré by uľahčilo deťom ovládanie programu, upútalo by ich fantáziu a programovanie by sa stalo pre nich zaujímavejším. Príkazy tu boli zapisované a znázornené obrázkami/ikonami alebo jednoduchými slovami. Tak vznikli mikrosvety.

V minulosti vzniklo niekoľkom mikrosvetov, ktoré slúžili žiakom na pochopenie základov programovania. Prehľad historického vývoja mikrosvetov a detských programovacích jazykov je uvedený v tabuľke 1.

Z tabuľky 1 je viditeľné, že počiatky vývoja detských programovacích jazykov sa tradujú už do šesťdesiatych rokov dvadsiateho storočia a do dnešných čias ich vzniklo niekoľko desiatok. Menili sa nielen ich verzie, ale vznikali rôzne nové návrhy, ktoré prinášali stále lepšie a zaujímavejšie prostredia pre pohodlnejšiu a jednoduchšiu prácu detí a obohacovali sa aj možnosti programovania, ktoré dovoľovali vytvárať zložitejšie programy.

Na Slovensku sa, ešte počas éry osembitových mikropočítačov, ako prvý na školách objavil programovací jazyk Robot Karel, ktorý bol neskôr využívaný aj na prvých šestnásťbitových mikropočítačoch. S rozvojom informačných a komunikačných technológií začiatkom deväťdesiatych rokov dvadsiateho storočia sa na školách v rámci projektu INFOVEK na vyučovanie základov programovania začal využívať detský programovací jazyk Logo.

Postupne sa prechádzalo na využívanie detských programovacích jazykov Logo, Imagine Logo, SGP, Baltík 3.0, Baltie 4C#, Scratch a Robot Emil. Tieto boli využívané na vyučovanie základov programovania už od prvého stupňa základnej školy počas povinných hodín informatiky, ktorých súčasťou je aj vyučovanie základov algoritmického myslenia.

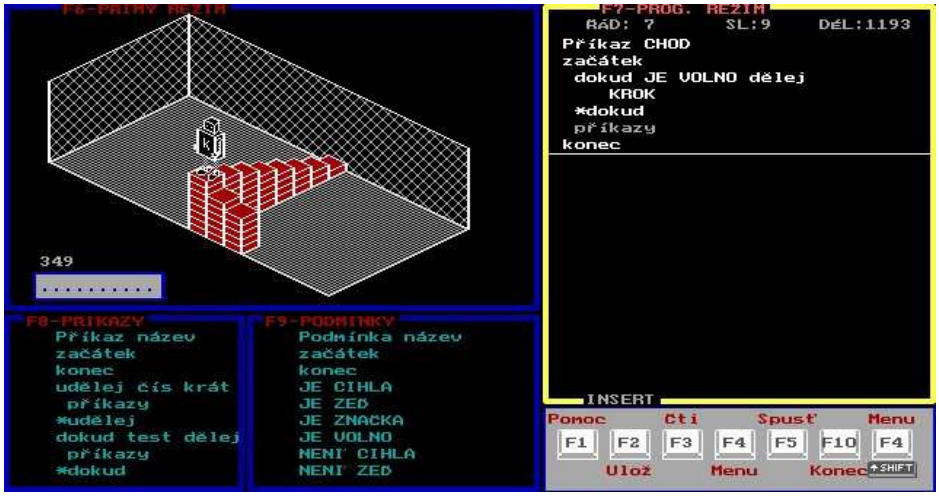
Existujú rôzne typy mikrosvetov, ktoré slúžia na rôzne účely. Sady riadiacich príkazov a typ ovládaného objektu primárne závisia od veku, záujmov a vyučovacích cieľov žiaka. Prístup mikrosvetov sa však dá aplikovať aj na iné paradigmy než na vyučovanie procedurálnych jazykov. Napríklad pre paralelné programovanie, kde je vhodný projekt Robot Brothers (Olimpio, 1998), detský programovací jazyk Imagine alebo Baltie 4C#. Príkladom objektovo-orientovaného prostredia okrem posledných dvoch spomenutých sú napríklad aj Playground (Fenton & Beck, 1989), Gravitas (Sellman, 1992) a KidSim (Smith a kol., 1994) alebo Alice. Niektorí autori považujú objektovo - orientované detské programovacie jazyky za najlepšiu možnosť ako vyučovať mladších žiakov základy programovania.

Tabuľka 1: Príklady mikrosvetov

Názov	*Tvorca*	*Rok vydania*
Logo	Feurzeig & Papert	1967
ALspace	Dionne & Mackworth	1978
SGP Baltazár	Soukup	1978
Petr	Gemtree	1978
Karel the robot	Pattis	1981
Josef the robot	Tomek	1982
Martino	Olimpio et al.	1985
PMS	Tomek et al.	1985
BALSA	Brown & Sedgewick	1985
SEE	Baecker & Marcus	1986
Robot Brothers	Olimpo	1988
ANIMUS	Duisberg	1988
BALSAII	Brown	1988
VIP	Mendes & Mendes	1988
ALLADIN	Helttula et al.	1989
Marta	Calabrese	1989
Pascal Genie	Miller & Chandhok	1989
GAIGS	Naps	1990
Cirkus šaša Tomáša – Poštárik na bicykli	Blaho & Kalaš& Tóth & Lupták	1990
TANGO	Stasko	1990
ANIM	Bently & Kerninghan	1991
ZEUS	Brown	1991
Turingal	Brusilovsky	1991
Karel-3D	Hvorecky	1992
Pavane	Roman et al.	1992
VCC	Baeza-Yates et al.	1992
XTANGO	Stasko	1992
POLKA	Stasko & Kraemer	1993
Darel	Kay & Tyler	1993
Tortoise	Brusilovsky	1993
Turtle-Graph	Jehng et al.	1994
DRUIDS	Whale	1994
FLAIR	Ingargiola et al.	1994
POLKA-RC	Stasko & McCrickard	1995
Baltík 2	Soukup	1997
Alice	Randy Pauschem.	1997
JAWAA	Pierson & Rodger	1998
ALMA	Varanda & Henriques	1999
Baltík 3.0	Soukup	1999
Imagine Logo	Blaho & Kalaš & Tomčányi	2001
Cirkus šaša Tomáša	Blaho & Kalaš & Tóth	2002
Baltík 4C#	Soukup	2003
Scratch	Resnick	2003
Robot Emil	Kalaš,	2017
Baltík Web	Soukup	2020

3.1 Robot Karel

Za prvý detský programovací jazyk je považovaný detský programovací jazyk Karel. Programovací jazyk Karel bol navrhnutý a využívaný Richardom Pattisom v roku 1981. Pattis v tom období pôsobil na Univerzite v Standorde a vyvinul tento jazyk za účelom zjednodušiť svojim študentom pochopenie štruktúrovaného programovania v jazyku Pascal. Aj to je dôvod toho, že detský programovací jazyk Karel (obr. 1) obsahuje všetky dôležité štruktúry programovacieho jazyka Pascal, učí základným predstavám o procedurálnej abstrakcii, sekvenčnému a podmienenému vykonávaniu príkazov. Objektom detského programovacieho jazyka je od jeho prvej verzie robot Karel, ktorý vykonáva rôzne úlohy vo svojom svete stien, tehličiek, kvádrov a značiek. Jeho hlavné aktivity sú "UROB KROK", "OTOČ SA", "POLOŽ TEHLU", "ZDVIHNI TEHLU", "POLOŽ KVÁDER" a "ZDVIHNI KVÁDER". Pomocou týchto základných príkazov a podmienok môže tento robot budovať a kontrolovať priestor svojho mikrosveta. Detský programovací jazyk Karel bol po prvýkrát implementovaný na osembitových počítačoch PMD-85, ktoré boli v druhej polovici osemdesiatych rokov minulého storočia využívané na školách a tiež pre počítače Sinclair ZX Spectrum. Od tej doby vzniklo množstvo ďalších verzií a implementácií, ako sú napríklad Karel 96, Karel 3D alebo internetová aplikácia detského programovacieho jazyka Karel.



Obrázok 1: Ukážka zápisu programu v detskom programovacom jazyku Robot Karel
(Zdroj: Autorka, 1997)

Podľa Karela Alana Kaya (1979), autor jednej z verzií, detský programovací jazyk „*by mal mať nízky prah a vysoký strop*“. Podľa Kaya (1979) by sa dieťa malo ľahko oboznámiť s programovacím prostredím a jazykom, pomerne jednoducho sa v ňom naučiť pracovať, ale súčasne by nemal príliš skoro naraziť na jeho limity.

Verzia Karel 3D bola na rozdiel od predchádzajúcej verzie objektovo orientovaným softvérom, ktorý dokázal pracovať aj v trojrozmernom priestore.

V osemdesiatych rokoch minulého storočia vzniklo niekoľko verzií odvodených od detského programovacieho jazyka robot Karel. Medzi ne patrili napríklad Marta alebo Joseph the Robot.

3.2 Marta

Ústrednou postavičkou detského programovacieho jazyka Marta je robot Marta. Základom tohto programovacieho jazyka je, na rozdiel od Robota Karela, Logo. Rovnako ako ostatné detské programovacie jazyky sa môže Marta použiť ako propedeutika do programovania pre všetkých záujemcov o základy programovania. Medzi jej vlastnosti, ktoré nekorešpondujú s vyššie spomenutým programovacím jazykom Karel, patrí napríklad, že Marta môže byť ovládaná klávesmi, čo je vhodné najmä pre najmenších žiakov. Marta môže stavať svoj svet, postaviť a zrušiť stenu, pracovať a byť riadená aj potme, keď nevidno prekážky a značky. V detskom programovacom jazyku Marta sa dajú ľahko definovať nové príkazy a operácie tým, že programátor napíše procedúru korešpondujúcu s jazykom Logo. Dá sa v ňom programovať na niekoľkých úrovniach. Autor vie vytvárať v interaktívnom, priamom režime jednoriadkové aj viacriadkové programy a v editovacom režime Logovské procedúry. Svet objektu Marta je ohraničený mriežkou rozmeru 7x15. Na jeden priesečník mriežky je možné položiť najviac 4 značky.

3.3 Josef the Robot

Detský programovací jazyk Josef the Robot bol vytvorený v rovnakom období ako jazyk Karel. Josef the Robot používa konštrukcie detských programovacích jazykov Karel a Logo. Jeho filozofia spočíva v príprave „malých“ programátorov na vytváranie programov vo „veľkom“ jazyku.

3.4 Turingal

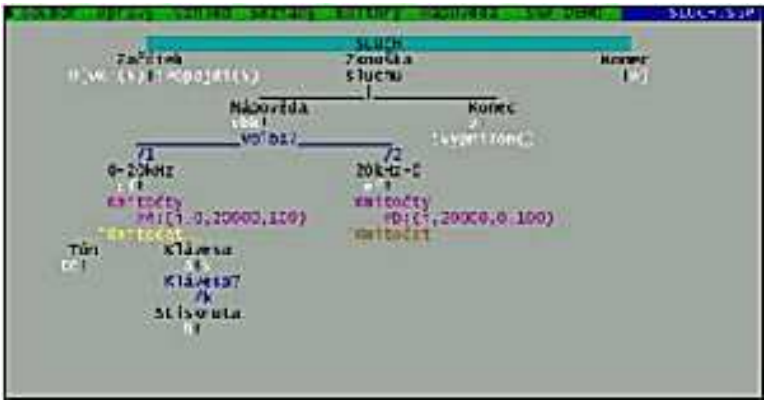
Turingal bol vytvorený pre študentov Katedry počítačov Moskovskej Štátnej univerzity. Jazyk poskytoval kontrolu známej algoritmickej teórie Turingovho stroja, ktorý pracoval s diernou páskou. Základné operácie jazyka Turingal boli veľmi jednoduché, avšak veľmi názorné. Boli to inštrukcie ovládajúce pohyb vľavo a vpravo po páske a príkazy na písanie symbolov na pásku. Tento jazyk ponúkal aj riadiace štruktúry, podmienené vykonávanie príkazov a príkazy opakovania, rovnako sa v ňom dali tvoriť aj podprogramy, ktoré mali podobnú syntax a sémantiku ako mal programovací jazyk Pascal.

3.5 Alice

Alice je programovací jazyk primárne vytvorený na vyučovanie objektového programovania žiakov a študentov stredných a vysokých škôl. Výučba objektového programovania v programovacom jazyku Alice prebieha prostredníctvom vytvárania príbehov, videí a interaktívnych hier a ich zverejňovaním prostredníctvom internetu. Bol navrhnutý pre študentov, ktorý sa s objektovo orientovaným programovaním stretávajú prvýkrát vo svojej programátorskej praxi. V programovacom jazyku Alice majú programátori k dispozícii trojrozmerné objekty, napríklad zvieratá a ľudí, ktorí sú obyvateľmi virtuálneho sveta, v ktorom študenti vytvárajú animácie týchto objektov. Zdrojový kód programu v programovacom jazyku Alice je podobný jazyku Java, C# alebo C++. Je to interaktívny jazyk. Je v ňom prepojený

priestor, v ktorom je zdrojový kód a zobrazená jeho realizácia. Je to miesto, kde žiaci a študenti majú možnosť okamžitej interakcie medzi zdrojovým kódom a realizáciou algoritmu.

3.6 SGP Baltazár



Obrázok 2: Ukážka zápisu programu v detskom programovacom jazyku SGP Baltazár
(Zdroj: Autorka, 1997)

Objektom v programovacom jazyku SPG Baltazár je čarodejník. Algoritmy sú v tomto jazyku zapisované prostredníctvom dvojrozmerných štruktúr (obr. 2) v Jacksonových štruktúrogramoch, ktoré zaviedol v roku 1975 M. A. Jackson, ktoré tvorcovi algoritmu umožňujú nelineárny zápis programu. Na tieto štruktúry je potom v prostredí programovacieho jazyka SGP Baltazár aplikovaný preprocesor, ktorý generuje výsledný program v jazyku C.

3.7 Baltík

Objektom vo všetkých verziách detského programovacieho jazyka Baltík či Baltie je čarodejník Baltazár. Nakoľko tento programovací jazyk je vo všetkých verziách na rozdiel od programovacieho jazyka SGP Baltazár ikonickým jazykom, už aj najmladší žiaci ľahko a rýchlo dokážu zvládnuť prostredie detského programovacieho jazyka a vytvárať programy. Algoritmy sú zostavované z prvkov – ikon, ktoré obsahujú jednotlivé príkazy a pomocou presunu ikony na pracovnú plochu prostredníctvom primárneho tlačidla myši je skladaný program. Problémy so syntaxou sú v týchto prostrediach minimálne, nakoľko žiaci majú pred očami kompletnú sadu príkazov vo forme ikoniek.

Česká spoločnosť SGP Systems vyvinula v roku 1993 programovací a kresliaci systém SGP Baltazár pre výučbu štruktúrovaného programovania v jazyku C. V tomto systéme bol v roku 1996 v tomto programe kompletne naprogramovaný jazyk SGP Baltík 2, prvá ikonická verzia programu. Nasledovníkmi týchto jazykov sú Baltík 3.0 a jeho objektovo orientovaná verzia Baltie 4C#. Autorom všetkých verzií Baltíka bol tím okolo pána Soukupa, ktorí sa dodnes venujú detskému programovaciemu prostrediu Baltík, ale aj žiakom, pre ktorých organizujú súťaže nielen na národnej, ale aj na medzinárodnej úrovni.

3.8 SGP Baltík 3.0

Baltík 3.0 je detský programovací jazyk, ktorý jednoducho a názorne dokáže vniesť do tajov a pravidiel programovania deti v predškolskom veku, ale aj dospelých záujemcov

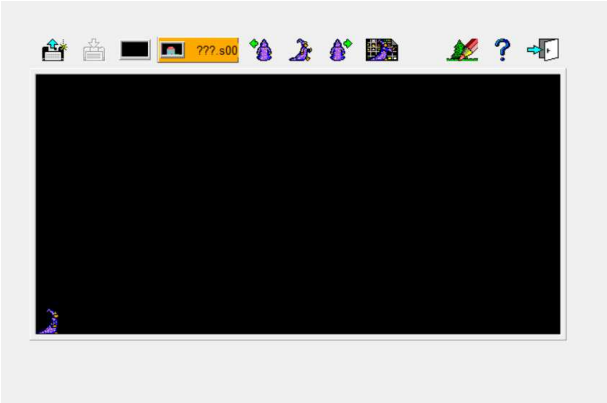
o programovanie. Je pokračovaním a ďalšou verziou detských programovacích jazykov z dielne programátorov okolo Soukupa. Je primárne určený na výučbu programovania detí vo veku od 6 do 15 rokov. Objekt programovacieho jazyka čarodejník Baltík, ako dobrý kamarát pomáha deťom zvládnuť postupne všetky programátorské úrovne od tej najjednoduchšej až po najťažšiu.

Vzhľadom na to, že všetky príkazy sú zadávané pomocou ikoniek, pričom pri nesprávnom použití je okamžite vidieť chybu, je Baltík vhodný skutočne aj pre tých najmenších, ale i pre deti s rôznymi poruchami učenia a inými hendikepmi.

Veľkým prínosom detského programovacieho jazyka Baltík 3.0 je zadávanie inštrukcií pomocou ikon a zároveň možnosť vybrať si z viacerých režimov podľa veku a skúsenosti používateľa.

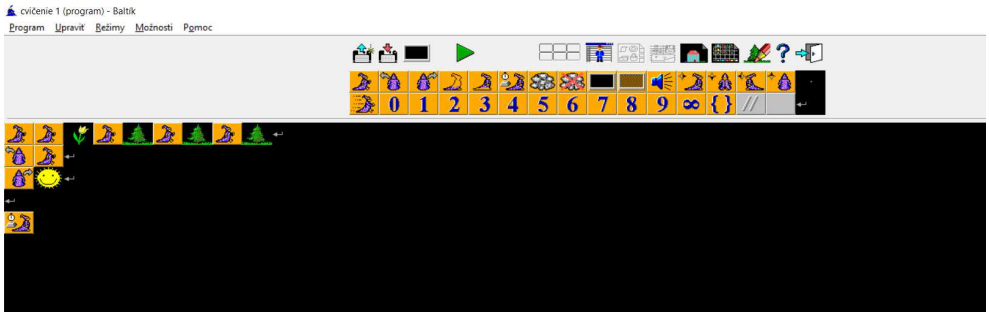
Baltík 3 pracuje v niekoľkých režimoch:

- *Skladať scénu* (obr. 3) je režim vhodný pre najmenšie deti, nejedná sa ešte o programovane, skôr o oboznámenie sa s prostredím, s prácou s myškou. V tomto režime má používateľ možnosť poskladať si obrázky pomocou predmetov umiestnených v bankách, ako i dotvoriť vlastné predmety podľa potreby a hlavne vlastnej fantázie.
- *Čarovať scénu* je interaktívny režim. Pripravuje dieťa na pochopenie princípov algoritmizácie a programovania. Pomocou príkazov ovládajúcich pohyb Baltíka a jeho čarovania si môžu deti natrénovať ako správne skladať postupnosti príkazov, ktoré využijú pri samotnej tvorbe programov.
- Posledný režim, *Programovať* sa skladá z dvoch častí, prvou je *Programovať začiatočník* (obr. 4), ktorý ma veľmi obmedzenú množinu príkazov. Je určený najmä pre deti na prvom stupni a nižšie ročníky druhého stupňa. Deti si môžu vytvoriť jednoduché programy, urobiť pre nich vhodné grafické prostredie, vymyslieť jednoduché animácie, rozprávky, prezentácie a pod. Prostredie ponúka obrovské možnosti pre rozvoj tvorivosti a realizáciu vlastných detských nápadov, rovnako rozvíja medzipredmetové vzťahy. Druhá časť režimu, *Programovať pokročilý* (obr. 5), obsahuje všetky príkazy, s ktorými sa stretávame aj v iných programovacích jazykoch. Využíva premenné, procedúry, vkladanie zvukových i obrázkových záznamov, vytváranie automatických i zložitých ručných animácií. Je plnohodnotnou prípravou pre neskoršie zvládnutie vyšších programovacích jazykov a je vhodný najmä pre druhý stupeň. Obsahuje prakticky všetky príkazy potrebné pre zvládnutie nielen základov programovania.



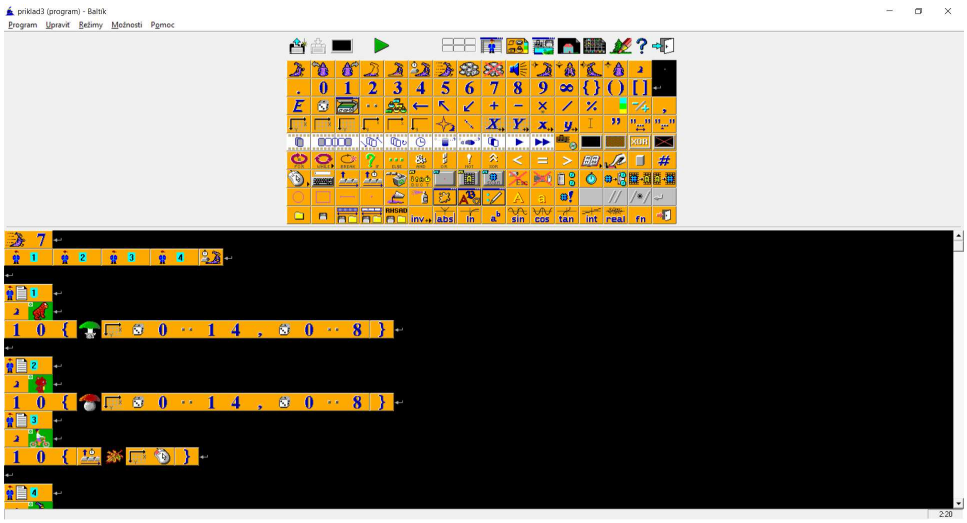
Obrázok 3: Režim Skladať scénu

(Zdroj: Autorka)



Obrázok 4: Režim Programovať začiatočník

(Zdroj: Autorka)



Obrázok 5: Režim Programovať pokročilý - zápis programu

(Zdroj: Autorka)

3.9 SGP Baltie 4C#

SGP Baltie 4C# je ďalšou verziou z „rodiny“ detských programovacích jazykov Baltík. Ide o moderný objektovo orientovaný výučbový programovací nástroj založený na C#, DirectX a .NET. Umožňuje jednoduché programovanie 3D aplikácií pre rôzne verzie Windows (2000, XP, Vista 7, 8), pri verzii Windows 10 je jeho používanie problematické. Je určený na

programovanie a programátorské experimentovanie pre deti, mládež vo veku od 13 do 19 rokov i dospelých. Poskytuje ďalšie varianty režimov, vytvárať multimediálne prezentácie, používať paralelné procesy a ich synchronizáciu, zmeniť vlastnosti a počet Baltíkov a iné. Pracovný priestor, v ktorom sa objekt Baltíka pohybuje je neohraničený, trojrozmerný. Verzia Pro (Professional) obsahuje aj editor C# kódu, generuje samostatne spustiteľné aplikácie (.exe) a vie exportovať projekt pre Visual Studio 2003, 2005 (C#). V tejto verzii bol integrovaný aj dvojrozmerný Baltíkov priestor, avšak objekt v ňom bol prevzatý z 3D priestoru.

3.10 Baltík WEB

Baltík web je novou webovou aplikáciou založenou na verzii Baltík 3.0. Na spustenie je potrebný iba internetový prehliadač na akomkoľvek zariadení s akýmkoľvek operačným systémom. Táto verzia je vhodná nielen pre počítače a notebooky, ale na rozdiel od predchádzajúcich programov, aj pre tablety a smartfóny. Je to ideálny detský programovací jazyk pre výučbu programovania aj v rámci online hodín.

Hlavnou zmenou verzie Baltík WEB v porovnaní s verziou Baltík 3.0 je to, že sa programuje už vo všetkých troch úrovniach:

- Úroveň **1** programovanie pohybom (myšou, prstom) je vhodné pre najmladšie deti (od 4 rokov), samotný kód v tejto úrovni nie je vidieť. Úroveň je vhodná pre tvorbu vlastných príbehov, ktoré môžu obsahovať animácie.
- Úroveň **2** –interaktívne programovanie pomocou niekoľkých príkazov. Programový kód je viditeľný, táto úroveň je vhodná pre učenie základných príkazov Baltíka vrátane jeho pohybu po scéne. V interaktívnom programovacom režime je vhodné vysvetliť žiakom rozdiel zdrojového kódu od programu a jeho realizácie.
- Úroveň **3** - klasické programovanie rovnako ako v predchádzajúcich verziách obsahuje dva režimy. Režim Začiatočník a režim Pokročilý, ktoré sú rozdelené podľa náročnosti práce,

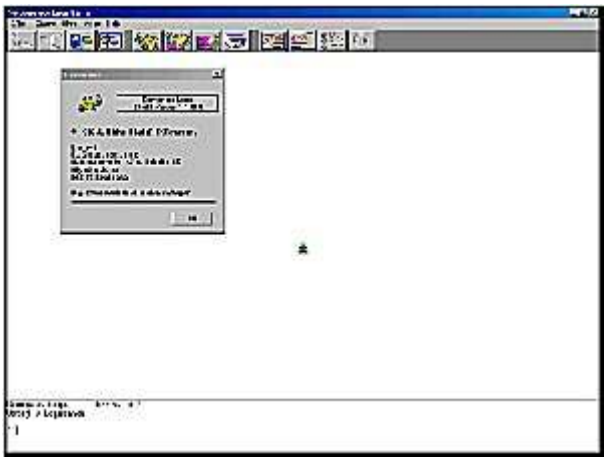
Rozdielom Baltík WEBu od predchádzajúcich verzií je automatické vyhodnocovanie úloh serverom pre všetky úrovne. Učiteľ si môže vytvárať vlastné úlohy. V čase tvorby tejto publikácie (2022) prebiehala testovacia verzia detského programovacieho jazyka Baltík WEB. V testovacej verzii je možné vytvoriť len 5 vlastných úloh.

3.11 Petr

Obrázok 6: Ukážka zápisu programu v detskom programovacom jazyku Petr
(Zdroj: Autorka, 1997)

V programovacom mini-jazyku Petr je ústrednou postavičkou - objektom zajac. Pohyb Petra nie je obmedzený štyrmi svetovými stranami. Dokáže sa, ako napríklad šachová figúrka kráľa, priamo presúvať na ktorékoľvek z ôsmich políčok, ktoré susedia s tým, na ktorom Petr stojí. Štruktúra zápisu programu vychádza zo stromovej štruktúry, aká je známa z usporiadania adresárov a súborov na disku. Prostredie neobsahuje ladiace prostriedky.

3.12 Comenius Logo



Obrázok 7: Prostredie detského programovacieho jazyka Logo
(Zdroj: Autorka, 1997)

V roku 1967 vyvinuli v USA na Masatchusetskom technologickom inštitúte jazyk Logo naprogramovaný v programovacom jazyku Lisp, z dôvodu „*aby vedci mohli pozorovať spôsob tvorivého myslenia, experimentovania a učenia sa žiakov*.“ (Blaho, 2001). Postupne jazyk prešiel rôznymi úpravami, od pridania špeciálneho grafického pera – korytnačky až po obohatenie rôznymi programovými a údajovými štruktúrami. Vyvinuli sa z neho rôzne verzie, ktoré prenikli aj na slovenské školy a to ComeniusLogo (obr. 7) a jeho nepriamy nasledovník

Imagine Logo. Imagine bol uvedený na trh vo februári 2001 vydavateľstvom Logotron v anglickej verzii. Od roku 2005 existuje aj jeho slovenská verzia. Asi najväčším prínosom Comenia Loga bola pravdepodobne myšlienka obrázkových tvarov korytnačiek, ktoré sa dali dosť jednoducho animovať a tiež práca s myšou a jednoduchá manipulovateľnosť s multimédiami. Pri získavaní základov programovania v jazyku Imagine sa jedná o osvojenie objektovo orientovaného prístupu a programovania riadeného udalosťami. Podporuje paralelné programovanie a tiež má prepracovanú myšlienku obrázkových tvarov korytnačiek.

Detský programovací jazyk Comenius Logo predstavuje typickú korytnačiu grafiku. Jeho verzia Imagine Logo, do ktorej je integrované aj národné jazykové prostredie, je stále využívaná v edukačnom procese v slovenskom školstve. Viac o detskom programovacom jazyku Imagine sa dočítate v kapitole 3.13.

3.13 Imagine Logo

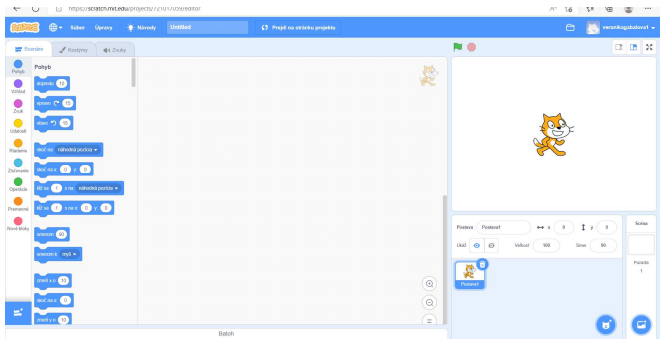
Imagine Logo (skrátene Imagine) je detský programovací jazyk, v ktorom programátor učí korytnačku príkazmi vykresľovať rôzne zaujímavé útvary, prípadne vykonať nejakú zadefinovanú akciu alebo interakciu. Detský programovací jazyk Imagine je pokračovateľom programovacieho jazyka Logo. Objektom detského programovacieho jazyka je korytnačka Žofka. Napriek tomu, že Imagine Logo je radený medzi detské programovacie jazyky, možno v ňom písať aj zložitejšie a komplexnejšie aplikácie. Motivačným základom je ovládanie objektu, korytnačky Žofky, ktorá je ovládaná niekoľkými základnými príkazmi zapísanými v materinskom jazyku. Projekt vytvorený v prostredí Imagine v IDE je možné skompilovať aj do .exe súboru a je spustiteľný aj na počítači, na ktorom nie je inštalovaný Imagine.

3.14 Scratch

Scratch je detský programovací jazyk, v ktorom programovaním môže žiak realizovať svoje vlastné interaktívne príbehy, hry a animácie - a zdieľať svoje výtvory s ostatnými aj v online komunite. Umožňuje žiakom nielen skúmať, ale aj experimentovať s ideou počítačového programovania pomocou „blokov“, ktoré sú zostavované na vytvorenie jednoduchého kódu. V online komunite majú žiaci k dispozícii diskusnú skupinu, ktorú používajú hlavne na komunikáciu a prípadnú pomoc s kódovaním. Scratch je preložený do viac ako 70 jazykov a je to jazyk, ktorý umožňuje mladým ľuďom naučiť sa kreatívne myslieť a systematicky uvažovať či spolupracovať. Scratch je primárne navrhnutý najmä pre deti a mládež vo veku 8 až 16 rokov, ale samozrejmosťou je, že ho používajú ľudia všetkých vekových kategórií.

Scratch je program, ktorý môžu žiaci využívať v podobe samostatného programu, ktorý funguje aj bez pripojenia na internet. Prostredie detského programovacieho jazyka s ukážkou zápisu algoritmu je na obrázku 8. Samozrejmosťou je aj jeho online verzia, ktorej výhodou je, že dokument, ktorý žiaci vytvoria je uložený na serveroch, a tak je im k dispozícií kdekoľvek sa do online verzie prihlásia a je možné ho zdieľať so svojimi priateľmi. Objektom v tomto detskom programovacom jazyku je mačka. Pracovná plocha, po ktorej sa môže pohybovať objekt v mikrosvete Sratch, má rozmer 480x360 bodov, pričom bod so súradnicami [0,0] sa

nachádza v prostriedku pracovnej plochy a tak x-ová súradnica nadobúda hodnotu -240 až 240 a y-ová nadobúda hodnotu od -180 do 180.



Obrázok 8: Prostredie detského programovacieho jazyka Scratch

(Zdroj: Autorka)

3.15 Robot Emil

Detský programovací jazyk Emil je považovaný za jedinečnú vyučovaciu metódu vhodnú na vyučovanie informatiky na základných školách. Emil podľa autorov prináša radosť z objavov a zábavu v jednom. Je primárne orientovaný na školu. Využíva formu bádania, riešenia problémov a spolupráce medzi predmetmi. Jeho tvorcovia využívajú svoje medzinárodné skúsenosti z vyučovania informatiky. Robot Emil učí žiakov k zodpovednej existencii v digitálnom prostredí, učí ich ako môžu zodpovedne poznávať a meniť svet.

Robot Emil obsahuje aj dve prostredia a to na programovanie Emil v Cirkuse a Emil na cestách, ktoré sú určené pre materské školy.

4 Základné charakteristiky detského programovacieho jazyka

Detský programovací jazyk býva často označovaný aj pojmom mini-jazyk. Skupina mini-jazykov je však širším pojmom a detské programovacie jazyky by sme mohli označiť ako podmnožinu tejto skupiny jazykov. Tieto jazyky sú charakteristické jednoduchosťou v syntaxe aj sémantike. Medzi ich základné charakteristiky patria:

- **názornosť** - pri väčšine operácií s objektom sú zmeny viditeľné v mikrosvete reprezentovanom na obrazovke počítača;
- **atraktívnosť a zmysluplnosť** - zameraný pre danú kategóriu žiakov (napr. v Japonsku s motívom Sumo zápasenia – Algo -Arena (Kato & Ide, 1993), alebo iný systém s metaforou nakupovania v supermarkete (Comr & Pintelas, 1989));
- **dialógový režim** – „zhovorčivosť" - t.j. každý príkaz sa vykonáva v riadiacom alebo programovacom móde;
- **modularita** - mal by obsahovať mechanizmus na vytváranie abstraktných inštrukcií (procedúr), tieto procedúry by mali tvoriť nezávislé jednotky, ktoré možno využiť pri riešení čiastkových problémov.

Keď učíme žiakov princípy programovania, špeciálny dôraz sa kladie na atraktívnosť objektu mikrosveta. Spôsobov ako ho zatraktívniť je viacero. Dôležité je však, aby sme sa priblížili reálnemu svetu. Vhodným námetom, predovšetkým u menších žiakov, môže byť robot, ako je to napr. v Karlovi. Iným príkladom môže byť použitie robota v paralelnom programovaní, s ktorým sa stretávame v MultiLogu (Resnick, 1990). Nahradením objektov mikrosveta reálnymi sa programovacie jazyky st-ávajú nielen atraktívnejšími, ale taktiež podporujú spoluprácu pri riešení problému. V ďalšej časti textu sa porovnávajú detské programovacie jazyky Imagine, Baltík 3.0 prípadne Baltie 4C#.

4.1 Úloha prostredia detského programovacieho jazyka

Dobré programovacie prostredie, ktoré využívame pri vyučovaní algoritmizácie a programovania, by malo umožniť žiakom, aby bol na obrazovke počítača súčasne viditeľný nielen mikrosvet, po ktorom sa ústredná postavička pohybuje, ale súčasne aj žiakov program. Ideálne je, ak v interaktívnom programovacom režime môže žiak realizovať opravu v príkaze, a opäť nechať ústrednú postavičku zapísaný algoritmus vykonať. Je tiež veľmi dôležité, aby sa v ňom znázorňoval priebeh vykonávania programu. Program by mal vykonať naraz jednu inštrukciu, zatiaľ čo interpreter ju v programe vysvieti a efekt sa zobrazí v mikrosvete. S týmto sa stretávame napríklad v programovacom jazyku Baltie 4.C#. Dôležité je tiež vizualizovať premenné a zásobník volaní podprogramov.

Prostredie by malo ponúkať štruktúrovaný editor na zvýšenie žiakovej produktivity. Štruktúrovaný editor pomáha žiakovi vyvarovať sa syntaktických chýb a poskytuje bezprostredné určenie ostatných chýb. Tiež napomáha orientovať sa v názvoch konštrukcií. Dobrý príklad mini-jazykového programovacieho prostredia s požadovanými funkciami predstavuje Karel Genie (Miller a kol., 1994). Poskytuje množinu špeciálne vytvorených nástrojov - štruktúrovaný editor, pohľad dekompozície programu a run-time systém. Karel Genie je integrované programovacie prostredie. Zdrojový kód sa počas vykonávania vysvieti.

Prostredie detského programovacieho jazyka by malo byť rozšírené o inteligentnú tútorovú a hypermediálnu súčasť. Inteligentný tútor môže zmenšiť množstvo nekreatívnej práce učiteľa a ušetriť jeho čas pre prácu so žiakmi, ktorí majú špeciálne požiadavky. Poskytuje potrebné množstvo vedenia, žiakovi navrhuje ďalšiu koncepciu pri riešení problému s ohľadom na jeho aktuálne vedomosti. Hypermediálna súčasť rozširuje priestor bádateľskému vyučovaniu, poskytuje žiakmi poháňaný prístup ku konceptuálnym vedomostiam a príkladom.

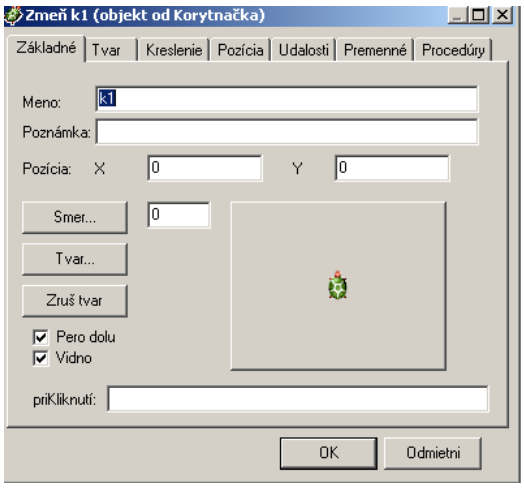
Dobrý detský programovací jazyk by mal byť doplnený o dobrú databázu atraktívnych a zmysluplných problémov určených pre žiakov na vyriešenie. Riešenie problémov je najefektívnejšia metóda na osvojenie si jazyka a na podporu dôležitých koncepcií. Databáza musí obsahovať problémy rôznej zložitosti a zahŕňať všetky dôležité koncepcie. Úlohy musia byť pre žiakov zaujímavé z hľadiska dosiahnutia cieľa i samotným procesom vyvíjania riešenia.

Veľmi úspešný je nasledujúci prístup: žiakovi predostrieme nový zmysluplný problém, potom mu predstavíme novú programovú konštrukciu, ktorá problém vyrieši.

Doteraz sme spomínali situáciu „jeden mikrosvet - mnoho problémov“. Existuje však aj iný prístup „jeden mikrosvet - jeden problém“, kde mikrosvet je orientovaný na riešenie jedného, avšak stále viac zložitejšieho problému (napr. nájsť východ z labyrintu, prípadne riešenie úloh Turingovho stroja).

4.2 Hlavný objekt programovacieho jazyka

V programovacom jazyku Imagine Logo je hlavným objektom korytnačka, ktorá je na začiatku umiestnená v strede stránky. Ide vlastne o grafický kresliaci robot, s ktorým sa spája mnoho procedúr na vytváranie zaujímavých kresieb či animácií. Korytnačka sa pohybuje po grafickej ploche na základe príkazov, pričom za sebou zanecháva stopu (kreslí čiaru). Základné nastavenie korytnačky ako jej meno, pozíciu, smer, farbu a hrúbku pera, tvar, udalosti, zábery a fázy pri animovanom tvare a iné je možné meniť podľa potreby. Na obrázku 9 je okno mikrosveta Imagine, v ktorom je možné zmeniť nastavenia korytnačky. Okno sa vyvolá stlačením sekundárneho tlačidla myši na ikone korytnačky.



Obrázok 9: Okno, ktoré umožňuje zmeniť základné nastavenia korytnačky
(Zdroj: Autorka)

Na obrázku 10 je ukážka ôsmich fáz jedného záberu novo zvoleného tvaru korytnačky, v tomto prípade bol vybratý kráčajúci kohútik. Záber predstavuje smer pohybu, ktorých býva spravidla viac (základný tvar korytnačky má 24 rôznych záberov, ktoré pomáhajú vnímať smer otočenia, v ktorom sa bude korytnačka posúvať (obr. 10). Prostredníctvom fáz sa vytvára efekt kráčania kohútika.

Obrázok 10: Ukážka fáz pohybu kohútika

(Zdroj: Autorka)

Obrázok 11: Základný tvar korytnačky - 24 záberov

(Zdroj: Autorka)

V každej stránke môže žiť aj niekoľko korytnačiek, ktoré musia byť jednoznačne pomenované, aby bolo možné korytnačky správne oslovovať a následne s nimi pracovať. Na obrázku 12 sú na štarte zobrazené štyri rôzne „korytnačky“, ktoré majú tvar potvoriek, V programe súťažia o prvenstvo, ktorá skôr príde do cieľa.

Obrázok 12: Ukážka tvarov objektu

(Zdroj: Žiaková, 2009)

V programovacom jazyku Baltie 4C# je ústredným sprievodcom mikrosveta Baltík, postavička znázorňujúca kúzelníka, ktorý je pánom nad pracovnou plochou okna – scénou. Jeho pozícia v dvojrozmernom svete je v ľavom dolnom rohu obrazovky. V trojrozmernom sa nachádza v kocke, ktorej ľavý dolný roh (vľavo za chrbtom Baltíka) je presne v strede Baltíkového priestoru, takže Baltík stojí na súradnici [X=1, Y=0, Z=1] a je otočený v smere osi Z.

Obrázok 13: Východzia pozícia Baltíka v trojrozmernom priestore

(Zdroj: Autorka)

Základným princípom jeho práce je „čarovanie“ predmetov pred seba, a tým vytváranie želaných obrazcov a programov. Baltík je schopný meniť nielen farbu svojho plášťa, ale aj čiapky, očí, pleti a pod. Je možné vybrať si jedného z desiatich ponúkaných Baltíkov, ktorého môžeme podľa potreby a vlastnej preferencie dotvoriť.

Obrázok 14: Desať rôznofarebných možností výberu Baltíka

(Zdroj: Autorka)

Obrázok 15: Možnosti farebného odlíšenia rôznych častí Baltíka

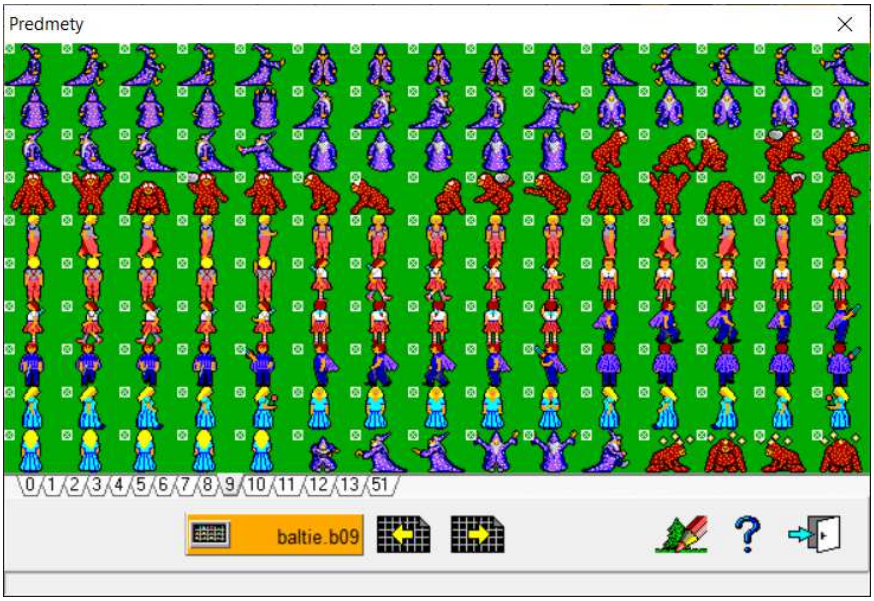
(Zdroj: Autorka)

Za zmienku určite stojí desiaty Baltík z ponúknutého výberu, ktorý predstavuje zástupného Baltíka a môže byť vymenený za ktoréhokoľvek iného Baltíka. Je vhodné ho používať ako parameter metódy, v ktorej bude zástupný Baltík niečo robiť a pri volaní metódy zvoliť Baltíka inej farby, aby sa príkazy v metóde naozaj vykonali. Tento spôsob je výhodný, ak je treba, aby viacerí Baltíci robili tú istú vec. Aj z vyššie uvedeného vyplýva, že aj v programovacom jazyku Baltie 4C# možno pracovať s viacerými Baltíkmi.

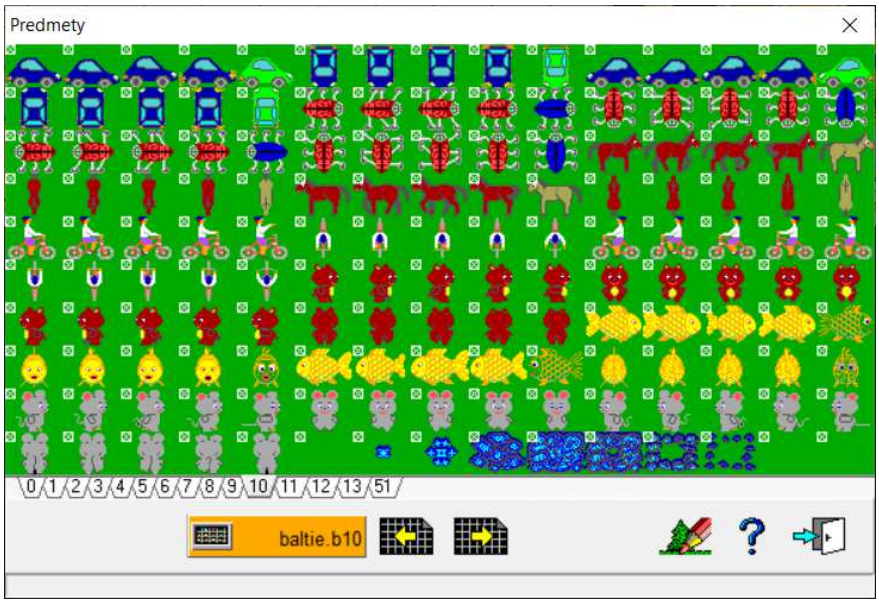
4.3 Baltík vo verzii Baltík 3.0

Verzia programovacieho jazyka Baltík 3.0 je verzia, podporujúca štruktúrované programovanie v dvojrozmernom prostredí. Objektom v tejto verzii je dvojrozmerná postavička Baltíka, ktorá sa pohybuje po obmedzenej pracovnej ploche pozostávajúcej so 150 štvorcov (rozmer

pracovnej plochy je 15x10 štvorcov). Po pracovnej ploche sa môže pohybovať najviac jeden objekt, ktorý však dokáže zmeniť svoj tvar. Namiesto čarodejníka sa môže po pracovnej ploche pohybovať napríklad chlapec, dievča, ale aj ryba, medveď, kôň či autíčko (viď obrázky 16 a 17).



Obrázok 16: Preddefinované obrázky použiteľné namiesto objektu Baltík – karta 9
(Zdroj: Autorka)



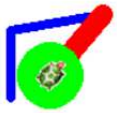
Obrázok 17: Preddefinované obrázky použiteľné namiesto objektu Baltík – karta 10
(Zdroj: Autorka)

4.4 Spôsob zadávania príkazov

Pri komunikácii s korytnačkou v programovacom jazyku Imagine sa využíva dolná časť okna Imagine (označená ako plocha výpisov), do ktorej zadávame inštrukcie a dostávame spätne textové správy a reakcie. Plocha výpisov sa delí na históriu (čiže úplný záznam inštrukcií a textových reakcií zo strany programovacieho jazyka Imagine) a príkazový riadok. Ak do

príkazového riadku napíšeme príkaz **dopredu 80** posunie sa korytnačka o 80 korytnačích krokov dopredu, **vpravo 37** otočí sa o 37 stupňov doprava apod. Dá sa meniť i farba a hrúbka pera. V detskom programovacom jazyku Imagine sa dajú príkazy okrem základnej verzie príkazov zadávať aj v skrátenej forme ich zápisu.

Príkladom je ukážka zápisu zdrojového kódu v základnej aj skrátenej verzii zápisu spoločne s realizáciou zapísaného algoritmu:

Zápis v základnej verzii príkazov	*Zápis v skrátenej verzii príkazov*	*Realizácia*
? nechhrubkapera 7 nechfarbapera "modra dopredu 50 vpravo 80 dopredu 70 nechfarbapera "cervena nechhrubkapera 15 vlavo 40 vzad 60 nechfarbapera "zelena bod 50	? nechhp 7 nechfp "modra do 50 vp 80 do 70 nechfp "cervena nechhp 15 vl 40 vz 60 nechfp "zelena bod 50	

Programovací jazyk Baltík je špecifický v tom, že sú textové príkazy nahradené ikonkovými príkazmi, ktoré sa vkladajú do zdrojového kódu pomocou myši, metódou potiahni a pusti. Priestor, do ktorého Baltík čaruje predmety alebo modely je, rozdelený na štvorce alebo kocky podľa toho či ide o dvoj- alebo trojrozmerný priestor. Pôvodná databáza predmetov, ktorá je k dispozícii po inštalácii softvéru, má 14 oddelení po 150 predmetov (obr. 18). Predmety možno dotvoriť kreslením pomocou grafického editora alebo podľa potreby nakresliť si úplne nové.

Obrázok 18: Ukážka banky predmetov v dvojrozmernom režime

(Zdroj: Autorka)

Každý predmet, ktorý Baltík vyčaruje v 2D priestore má určenú veľkosť 39 x 29 bodov, ktorú nie je možné meniť. Predmet nie je možné ani otočiť a pod. V trojrozmernom priestore sa dá namiesto už pripravených modelov použiť sprity, ktorý tieto nedostatky odstraňuje, ale tento spôsob je náročnejší a zároveň si vyžaduje väčšie množstvo vedomostí. Sprity sú 3D objekty vytvorené z modelov. Samotné modely sa do priestoru síce dajú jednoducho pridať pomocou

Baltíka, ale už sa s nimi nedá nijak pracovať. Ak však vytvoríte z modelu sprite, môžete ho potom pomocou spritu presúvať, otáčať, zväčšovať atď.

Baltík čaruje vždy pred seba a pohybuje sa o celú dĺžku štvorca alebo kocky. Baltík sa dokáže otočiť len do 2 základných strán – doprava a doľava. Dokáže ísť dopredu a dozadu vzhľadom na smer otočenia a ak je v trojrozmernom priestore vie sa presúvať aj hore a dole. Ukážka práce Baltíka - zdrojový kód, ktorý zabezpečí presun Baltíka a vyčarovanie dvoch predmetov je na obrázku 19 a jeho realizácia je na obrázku 20.

Obrázok 19: Príklad zdrojového kódu v 2D programovacom režime

(Zdroj: Autorka)

Obrázok 20: Realizácia zdrojového kódu algoritmu

(Zdroj: Autorka)

4.5 Režimy v mikrosvetoch

Mikrosvet Imagine nie je rozdelený na viaceré režimy alebo úrovne a nie je prispôsobený na „celý životný cyklus“ žiaka. Toto programovacie prostredie obsahuje okrem základných a pri výučbe najčastejšie používaných štruktúr aj náročnejšie prvky objektovo orientovaného programovacieho jazyka a prostredia ako prácu s objektmi, nekorytnačie kreslenie, tvorbu multimediálnych aplikácii, umožňuje používanie hlasového vstupu a výstupu, podporuje hierarchiu objektov a správaní a paralelné procesy.

Mikrosvet Baltie 4C# a aj jeho verzia Baltík 3.0 umožňujú v závislosti na veku a znalostí dieťaťa pracovať v rôznych režimoch, od úplného začiatočníka až po profesionála.

Úplne bazálny režim je režim skladania scény vhodný od 4 rokov[1], ktorý slúži pre deti na naučenie sa základného ovládania programovacieho jazyka, myši, metódy potiahni a pusti a používania menu. V tejto úrovni sa ešte nevyskytuje Baltík. Dieťa na scéne vytvára, skladá obrázky pomocou vopred pripravených bánk predmetov. Predmet na scénu vloží pomocou ťuknutia na vybraný predmet v banke a následným vrátením sa späť do scény, kde na požadovanej pozícii stlačí tlačidlo myšky. Presúvanie, kopírovanie a odstránenie predmetu sa deje tak isto pomocou myšky. V tomto režime sa vlastne ani nedá hovoriť o programovaní.

[1] Údaje o odporúčanom veku začatia používania režimov Baltíka sú uvedené podľa odporúčania autorov detského programovacieho jazyka Baltík 3.0 a Baltie 4C#

Ďalšie režimy môžeme rozdeliť na interaktívne režimy, ktoré slúžia na pochopenie ovládania Baltíka a programovacie režimy, pomocou ktorých sa dá naprogramovať akákoľvek aplikácia.

Do interaktívneho režimu (od 6 rokov) patrí:

1. 2D interaktívny režim– na obrazovke je postavička Baltíka, ktorú môže používateľ riadiť pomocou základných príkazov (obr. 21), a tak vytvárať scénu. Vo verzii Baltík 3.0 používateľ nevidí zapísaný kód iba realizáciu, kód vidí zapísaný v 2D režime vo verzii Baltie 4C# (obr. 22).

Obrázok 21: Základné ikonky, pomocou ktorých sa Baltík ovláda v interaktívnom režime

(Zdroj: Autorka)

Na tejto úrovni si treba osvojiť, že predmety sa čarujú pred Baltíka, ktorého je potrebné na vybrané miesto najprv doviesť pomocou príkazov vľavo bok, vpravo bok a posuň. Všetky príkazy, ktoré sa zadajú Baltíkovi, sa zobrazujú v dolnej časti ikonkového editora, ktorý sa dá upravovať podľa nastavenia, tak isto je možné meniť veľkosť plochy, v ktorej sa Baltík pohybuje. Štandardne je plocha rozdelená na 15 x 10 políčok. Na obrázku 22 je ukážka zápisu algoritmu s riešením v interaktívnom 2D režime.

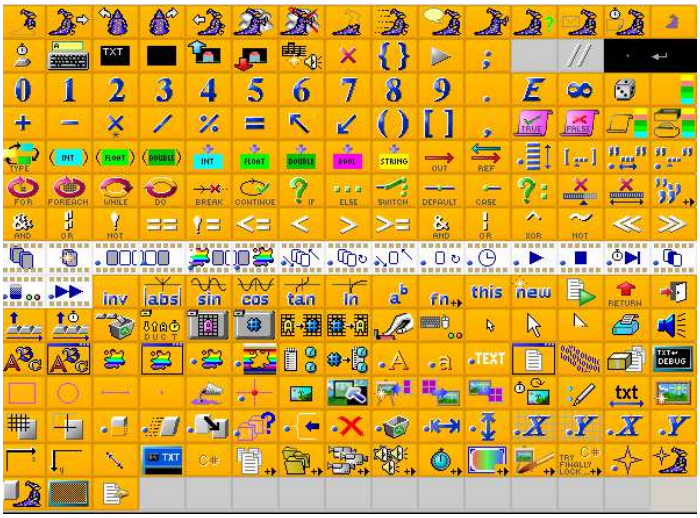
Obrázok 22: Interaktívny režim verzie Baltie 4C#

(Zdroj: Autorka)

2. 3D interaktívny režim - je vhodný na zoznámenie sa s trojrozmerným prostredím. Baltík sa pohybuje v priestore, ktorý nemá hranice, čo znamená, že ak sa Baltík vydá do ktoréhokoľvek smeru, nikdy nepríde na koniec. Celý priestor je rozdelený na kocky s veľkosťou 2x2x2m. Na scéne sa zobrazuje len tá časť priestoru, ktorú sníma 3D kamera.

Programovacie režimy sú vo verzii Baltie 4C# rozdelené nasledovne:

2D programovací režim s Baltíkom (od 7 rokov) a bez Baltíka (od 12 rokov) – vhodné na tie typy aplikácií, ktoré nevyžadujú 3D prostredie, napr. Tetris. V panely nástrojov sú k dispozícii ikonky príkazov pre vetvenie programu, prácu so súbormi, reťazcami, kreslením. Sú tu tiež ikonky pre matematické operácie, metódy, cykly, podmienky a iné (obr. 23).

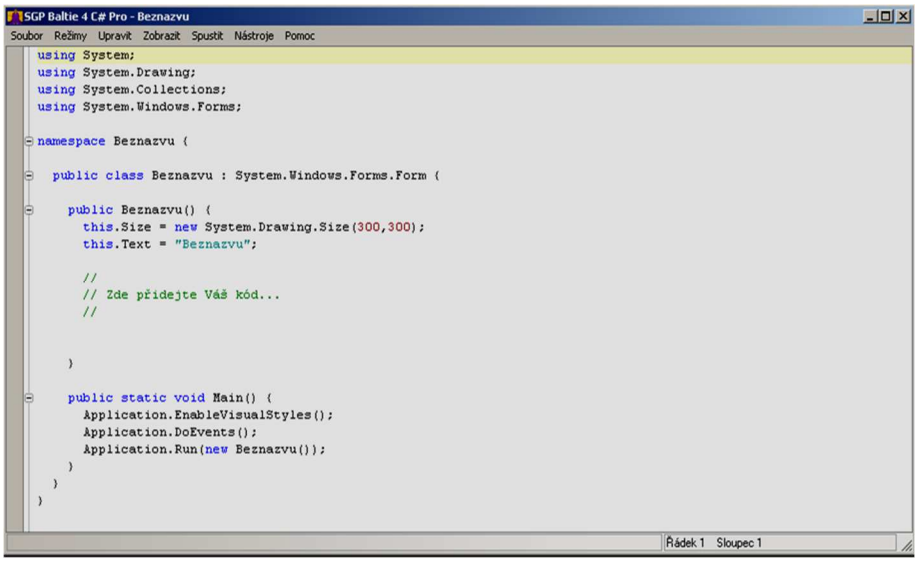


Obrázok 23: Ikonické príkazy používané v dvojrozmernom programovacom režime

(Zdroj: Autorka)

1. 3D programovací režim s Baltíkom (od 7 rokov) a bez Baltíka (od 12 rokov) - obsahuje všetky príkazy z 2D programovacích režimov plus mnoho príkazov pre prácu s 3D objektmi a 3D priestorom (vytváranie hmly, svetiel a animácií). Za zmienku stoja funkcie pre prehrávanie videa, hudby, práca s viacerými vláknami (umožňuje, aby sa viacero príkazov vykonávalo súčasne), posielanie správ medzi Baltíkmi – synchronizácia ich práce.
2. C# režim (od 16 rokov) –v ktorom sa dá vytvoriť prakticky všetko, je určený pre skúsených programátorov. Je dostupný iba vo verzii Baltie 4C#Pro. Dá sa v ňom písať len priamo C# kód bez možnosti vkladať ikony. Je možné priamo v ňom vytvárať vlastné triedy, štruktúry, rozhrania, menné priestory. Programátor si má možnosť zvoliť medzi piatimi rôznymi vopred pripravenými vizualizáciami C# režimov, a to:
    a) prázdna aplikácia, ktorá neobsahuje prednastavený kód, všetko musí programátor urobiť sám,
    b) konzolová aplikácia, v ktorej je pripravený jednoduchý kód so základnou štruktúrou programu,
    c) „okenní“[2] aplikácia je spustená v klasickom okne operačného systému Windows (obr. 24), do ktorého sa môžu pridávať rôzne ovládacie tlačidlá, editačné políčka, menu a pod.

[2] Termín prevzatý z originálu



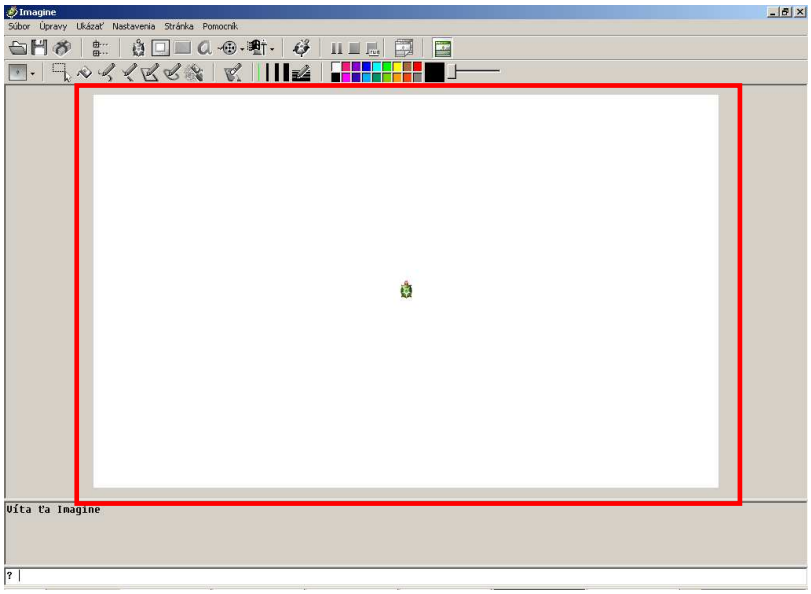
Obrázok 24: Ukážka aplikácie spustenej v okne operačného systému Windows
(Zdroj: Autorka)

d) 2D aplikácia obdoba 2D programovacieho režimu,
e) 3D aplikácia obdoba 3D programovacieho režimu.

3. Konzola (od 10 rokov) - programovací režim vhodný pre textové aplikácie nevyužívajúce grafické rozhranie.

4.6 Grafická plocha - Stránka

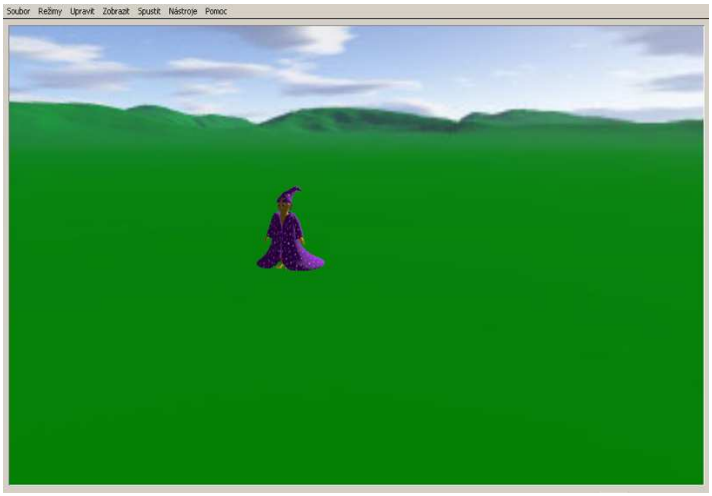
Každý projekt v prostredí Imagine môže obsahovať jednu alebo niekoľko stránok. Stránka je základný pracovný priestor pre všetky prvky projektu (obr. 25). V každom projekte môžu byť použité viaceré korytnačky, rôzne riadiace prvky, ako sú napríklad tlačidlá, posúvače, nástroje, texty, prehrávače médií, ďalšie papiere, atď.. Na pozadie stránky je možné vložiť vopred pripravený obrázok uložený v súbore, alebo vytvoriť obrázok v kresliacom editore LogoMotion, ktorý je určený na editáciu obrázkov a animácií. Stránka môže mať ľubovoľnú farbu pozadia. Základná farba pozadia stránky je biela a jej preddefinované rozmery sú: šírka 796 bodov a výška 499 bodov, ktoré sa dajú v prípade potreby zmeniť.



Obrázok 25: Pracovná plocha programovacieho jazyka Imagine

(Zdroj: Autorka)

Pri opise 2D a 3D režimov v mikrosvete Baltie 4C# existujú dva rôzne priestory: dvojrozmerný a trojrozmerný. Dvojrozmerný je rozdelený na 15 x 10 dielov, ktoré sa dajú meniť, štandardná farba plochy je čierna a opäť nastaviteľná podľa potreby programátora. Do priestoru sa dajú vkladať scény. Scéna je obrázok, ktorý sa dá zložiť z 2D dielikov a potom vložiť do programu. Ušetrí sa tým mnoho práce pri čarovaní scény cez príkazy. Trojrozmerný priestor je nekonečný a rozdelený na kocky, v rámci ktorých sa Baltík pohybuje. Baltík vždy stojí vnútri kocky a presúva sa do vnútra ďalšej kocky na ktorúkoľvek stranu. Nikdy sa nemôže nachádzať na rozhraní dvoch kociek. Z obrázka 25 je vidieť, že v 3D priestore je možné načítať aj pozadie. Ale aj toto pozadie je v nekonečne a nedá sa k nemu s Baltíkom prísť.



Obrázok 26: 3D prostredie Baltie 4C# so základným pozadím

(Zdroj: Autorka)

5 Príklady programových štruktúr

V tejto kapitole charakterizujeme vybrané štruktúry využívané pri výučbe programovania v detských programovacích jazykoch Imagine a Baltík, ako napríklad cyklus so známym počtom opakovaní, podprogramy – procedúry/metódy/pomocníci a rekurziu.

5.1 Cyklus so známym počtom opakovaní

Po prvom oboznámení sa s prostredím programovacieho jazyka a so základnými príkazmi, pri tvorbe jednoduchého algoritmu, bez ohľadu na to, o aký jazyk sa jedná, vždy vznikne potreba viacnásobného opakovania určitej konkrétnej (rovnakej) postupnosti príkazov. Opakovanie v detskom programovacom jazyku Imagine sa objaví pri kreslení jednoduchých geometrických útvarov, ako sú napríklad štvorec alebo obdĺžnik, a tiež v programovacom jazyku Baltie pri vyčarovaní radu kvetov - tulipánov.

Na opakovanie tých istých príkazov môžeme pri tvorbe algoritmu využiť príkaz cyklu so známym počtom opakovaní.

Na jeho zápis v detskom programovacom jazyku Imagine existuje príkaz:

opakuj n [p_1 ... p_k]

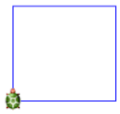
kde **n** je počet opakovaní a **p_1 ... p_k** je postupnosť príkazov, ktoré sa majú n-krát vykonať.

Príklad: Kreslenie štvorca v detskom programovacom jazyku Imagine.

Príkazy, ktoré sa majú vykonať zabezpečia vykreslenie čiary (strany štvorca) a nastavenie sa na kreslenie ďalšej strany štvorca. Štvorec má štyri strany, preto n bude mať hodnotu štyri.

Ukážka zápisu zdrojového kódu spoločne s realizáciou zapísaného algoritmu na vykreslenie štvorca:

opakuj 4 [do 100 vp 90]



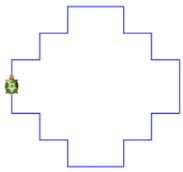
Príkaz **opakuj** sa môže nachádzať aj v postupnosti príkazov **p_1 ... p_k**.

opakuj n1 […opakuj n2 [p_1 ... p_m] …].

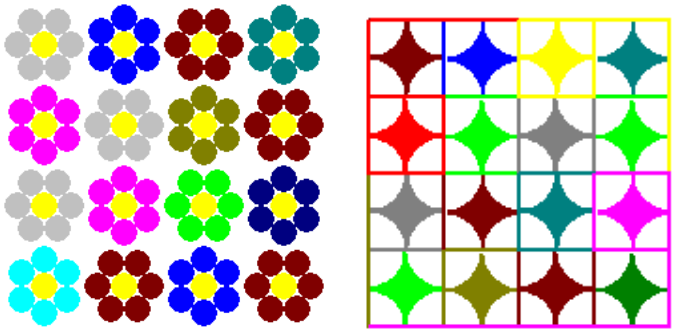
Príklad: Vnorenie príkazu opakuj do príkazov **p_1 ... p_k**.

Ukážka zápisu zdrojového kódu spoločne s realizáciou zapísaného algoritmu s vnoreným príkazom opakuj:

opakuj 4 [opakuj 3 [do 30 vp 90 do 30 vl 90] vp 90]



Takto je možné do seba vnárať viaceré príkazy opakuj, počet vnorení príkazu opakuj nie je obmedzený počtom vnorení, ale je ich možné realizovať podľa potreby algoritmu. Už so základnými príkazmi **dopredu, vzad, vpravo, vľavo** a **opakuj** je možné kresliť rôzne obrazce. Ak tieto navyše doplníme o ďalšie príkazy, ktoré napríklad menia nastavenie farby pera (príkaz **nechfarbapera - nechhp**) a nastavenie hrúbky pera (**nechhrubkapera - nechfp**) môžeme nakresliť komplikovanejšie farebné a graficky pre deti zaujímavé vzory (obr. 27). Pomocou príkazov **ph** - pero hore**, pd –** pero dole, zabezpečíme pre objekt – korytnačky, aby pri presune zanechal, alebo nezanechal stopu pera.



Obrázok 27: Príklad vzorov vytvorených pomocou vnoreného príkazu opakuj
(Zdroj: Žiaková, 2009)

V detskom programovacom jazyku Baltie 4C# na vyjadrenie najjednoduchšieho príkazu na opakovanie postupnosti príkazov slúži príkaz opakuj, ktorý má nasledovný zápis:

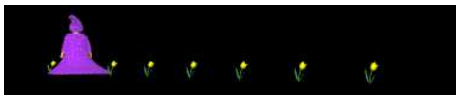
V detskom programovacom jazyku Baltík 3.0 najjednoduchší zápis príkazu na opakovanie činností má tvar:

Zápis pre príkaz opakuj je takmer identický v oboch verziách Baltíka s jediným rozdielom červeného znamienka „x“.

Na ľahšie pochopenie a demonštráciu uvádzame zápis algoritmu pre konkrétne príklady v oboch verziách detského programovacieho jazyka Baltík. Na obrázkoch 28 až 31 je uvedený príklad zadania, v ktorom má čarodejník Baltík vyčarovať záhon 8 tulipánov.

Obrázok 28: Zdrojový kód programu v obidvoch verziách Baltíka
(Zdroj: Autorka)

Obrázok 29: Postup realizácie algoritmu v dvojrozmernom režime v Baltie 4C#
(Zdroj: Autorka)



Obrázok 30: Postup realizácie algoritmu v trojrozmernom režime v Baltie 4C#

(Zdroj: Autorka)

Obrázok 31: Postup realizácie algoritmu v Baltík 3.0

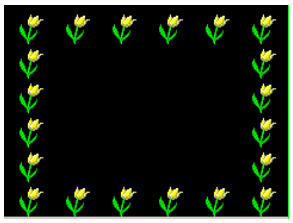
(Zdroj: Autorka)

Rozdiel vo verziách Baltíka je zrejmý aj zo zobrazenia realizácie predchádzajúcej realizácie úlohy. Viditeľný rozdiel je medzi realizáciami algoritmu v 2D a 3D verzii (obr. 29, 30 a 31). Tento rozdiel prináša pri vyučovaní aj určité výhody, ale aj nevýhody. Kým v 2D režime je presne určený priestor pozostávajúci z 10x15 štvorcov, v ktorom sa Baltík ako objekt pohybuje, v 3D režime Baltík nie je vôbec ohraničený. Fakt, že sa objekt pohybuje v neohraničenom priestore síce nekladie limity na pohyb objektu po pracovnej ploche, ale je veľmi náročný na predstavivosť žiakov. Stáva sa, že objekt pri nesprávnom algoritme „utečie“ z pracovnej plochy a „stratí sa v nekonečne“. Pre začiatočníkov v programovaní je preto jednoduchšie predstaviť si a sledovať pohyb Baltíka (vykonávanie algoritmu) v ohraničenom priestore. Nevýhodou 2D režimu je, že veľkosť kúzelníkovej postavičky je neproporcionálna vzhľadom ku čarovaným predmetom, kým v 3D režime je prispôsobená reálnym podmienkam. Pomer medzi výškou Baltíka a veľkosťou tulipánu je skutočná, a tak isto je zachovaná aj proporcionalita zvyšných predpripravených predmetov v banke predmetov, ktoré je možné čarovať navzájom.

Ďalším zaujímavým momentom je otázka perspektívy v 3D priestore, ktorú v žiadnom 2D priestore, a to nielen pre objekt Baltík, ale aj v iných detských programovacích jazykoch nemáme. Predmety, ktoré sú v zábere v 3D bližšie sú väčšie ako tie, ktoré sú umiestnené ďalej, čiže 3D režim zachováva perspektívu. Vzhľadom na trojrozmernosť tohto prostredia je možné nastaviť rôzne uhly pohľadu, čo však u žiakov vyžaduje tiež určitú zručnosť a priestorovú predstavivosť. Perspektíva ako taká predstavuje značné výhody a určitú dávku elegancie pri programovaní väčších projektov alebo hier, ale pre začínajúcich programátorov je oveľa vhodnejšie pracovať v ohraničenom priestore a využívať dvojrozmerný priestor a 2D režim. Aj to je dôvodom prečo sa v práci používajú názorné ukážky z dvojrozmerného režimu Baltie 4C#, prípadne Baltík 3.0. Na obrázku 32 je uvedený kód algoritmu, ktorý rieši zadanie úlohy, kde je v cykle opakuj vnorený ďalší cyklus opakuj. Riešenie úlohy je na obrázku 33.

Obrázok 32: Zdrojový kód algoritmu s využitím vnoreného príkazu opakuj

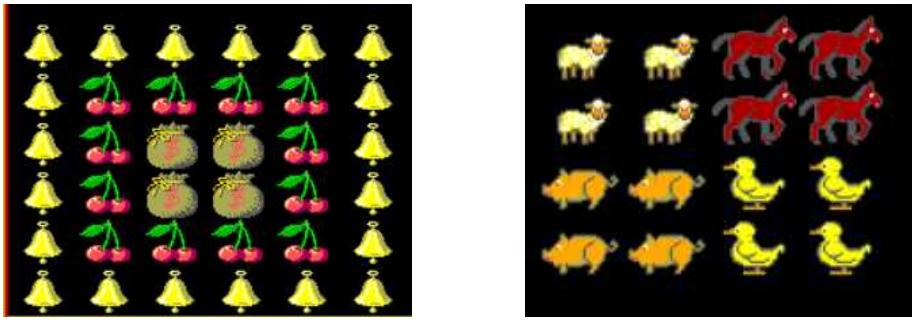
(Zdroj: Autorka)



Obrázok 33: Realizácia algoritmu z obrázka 35

(Zdroj: Autorka)

Rovnako ako v mikrosvete Imagine aj v detskom programovacom jazyku Baltík sa dajú príkazy na opakovanie vnárať do seba a podľa potreby tvoriť veľmi jednoduché obrazce.



Obrázok 34: Obrázky, ktoré sa dajú použiť pri precvičovaní príkazu opakuj

(Zdroj: Autorka)

5.2 Podprogramy – procedúry/metódy/pomocníci

Už prvé skúsenosti s programovaním v detských programovacích jazykoch, kedy sa žiak oboznámi so základnými príkazmi je pre neho zrejmé, že na zápis jednotlivých algoritmov potrebuje pomerne veľké množstvo príkazov. Práve tu nastáva priestor pre pedagóga na zavedenie členenia programov do podprogramov, a to aj kvôli sprehľadneniu zdrojového kódu.

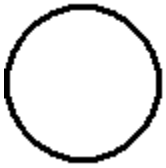
S nutnosťou členenia programov do menších častí sa žiaci stretávajú aj pri kreslení rôznych vzorov. Faktom je, že sa určité časti v zápise algoritmu neustále opakujú. Tieto opakujúce sa časti môžu vytvárať neprehľadný zdrojový kód. Neprehľadnosť zdrojového kódu môže byť príčinou logických chýb a chýb z nepozornosti, preto je potrebné, aby sa žiaci učili zapisovať zdrojový kód tak, aby bol prehľadnejší a skladal sa z menších logicky súvisiacich celkov. Na rozdelenie kódu do menších logicky súvisiacich častí sa v programovaní používajú ***procedúry*** (tzv. podprogramy). Procedúry môžu obsahovať aj opakujúce sa časti zdrojového kódu. Procedúry sa používajú ako ktorékoľvek iné príkazy a po ich zavolaní menom v hlavnom programe, prípadne inom podprograme (procedúre) sa vykonajú.

V rôznych programovacích prostrediach sú časti programu (prípadne podprogramu) označované rôznymi názvami. Napríklad v programovacom jazyku Baltík 3.0 slúži pre podprogram označenie *Pomocník*, v mikrosvete Baltie 4C# *metóda*, v programovacom jazyku Imagine, ako vo väčšine ostatných programovacích jazykoch, pre označenie podprogramov sa používa výraz *procedúra*.

Definíciu novej procedúry v detskom programovacom jazyku Imagine je možné realizovať v príkazovom riadku pomocou príkazu VIEM. Príkaz VIEM sa skladá z troch častí, v prvej

časti príkazu sa nachádza rezervované slovo VIEM a meno vytváranej procedúry. V prípade, že definovaná procedúra obsahuje parametre, udávajú sa v tejto časti procedúry. Druhá časť príkazu tvorí samotné telo procedúry. V tele procedúry sa nachádzajú príkazy, ktoré sa majú po zavolaní procedúry vykonať. Poslednou, treťou časťou príkazu VIEM je rezervované slovo KONIEC, ktoré ukončuje definíciu procedúry. Zoznam vytvorených procedúr je možné sledovať a upravovať v pamäti, kde je možné vytvárať aj nové procedúry. Korekciu príkazov už definovanej procedúry môžeme vykonať zadaním príkazu UPRAV meno procedúry v príkazovom riadku, prípadne upravením procedúry v pamäti.

Ukážka zdrojového kódu procedúry s realizáciou:

Zdrojový kód procedúry	*Realizácia*
viem *kruznica* opakuj 24 [do 10 vp 15] koniec	

Po zavolaní procedúry *kruznica* a po vykonaní algoritmu objekt sa na pracovnej ploche vykreslí kružnica konštantnej, v procedúre zadanej veľkosti. Ak by mal byť napísaný algoritmus všeobecný, ktorý pri realizácii vykreslí kružnicu ľubovoľnej veľkosti, je nutné použiť pri programovaní vykreslenia kružnice namiesto konkrétnej číselnej hodnoty za príkaz dopredu (v procedúre *kruznica* je použitý príkaz *do 10*) parameter, ktorý predstavuje zmenu veľkosti posunu dĺžky (*do : dlzka*).

Ukážka zdrojového kódu procedúry *kruznica* s parametrom *dlzka*:

Zdrojový kód procedúry	*Realizácia podľa hodnoty parametra dlzka*
viem *kruznica* **: dlzka** opakuj 24 [do **: dlzka** vp 15] koniec	

Ak by do definície procedúry *kruznica* bol pridaný ešte jeden parameter, ktorý bude predstavovať počet opakovaní príkazov v postupnosti príkazov a tento parameter by bol zároveň využitý na výpočet uhla otočenia, zmení sa realizácia procedúry nasledovne:

Zdrojový kód procedúry	*Realizácia podľa hodnoty parametrov dlzka a n*
viem *kruznica* : dlzka **: n** opakuj **: n** [do : dlzka vp **360 / : n**] koniec	

Z realizácie algoritmu zapísaného v procedúre *kruznica* s parametrami *dlzka* a *n* vidieť, že pridaním parametra, pomocou ktorého je prepočítavaný uhol otočenia, sa vykreslený útvar v závislosti od zadanej hodnoty parametra *n* môže zmeniť na konkrétny n-uholník.

Pri zadávaní mena procedúry je potrebné, aby pri voľbe názvov programov, procedúr, ale aj parametrov, bola dodržiavaná zrozumiteľnosť. Práve pri tomto príklade je veľmi vhodné realizovať zmenu názvu procedúry z *kruznica* na ***nuholnik*****.** Zmenený kód má nasledovný tvar:

viem ***nuholnik*** : dlzka : n
opakuj : n [do : dlzka vp 360 / : n]
koniec

Vo všeobecnosti zadefinovaný príkaz môže mať ľubovoľný počet parametrov. Je však potrebné klásť dôraz na to, aby názvy parametrov volili žiaci podľa ich významu. Ak by v procedúre ***nuholnik : dlzka : n*** miesto parametra „dlzka“ bol zadaný napr. parameter „*x*“, mohlo by sa stať, že si žiak neuvedomí čo-ktorý parameter udáva. A môže sa stať, že pri volaní procedúry, pri zadaní hodnôt za parametre príde pri realizácii algoritmu k vykresleniu iného útvaru, ako sa od programu očakáva.

Na obrázku 35 je zobrazená realizácia pri zamenení hodnôt parametrov v príkaze nuholnik.

nuholnik 5 50	*nuholnik 50 5*
	

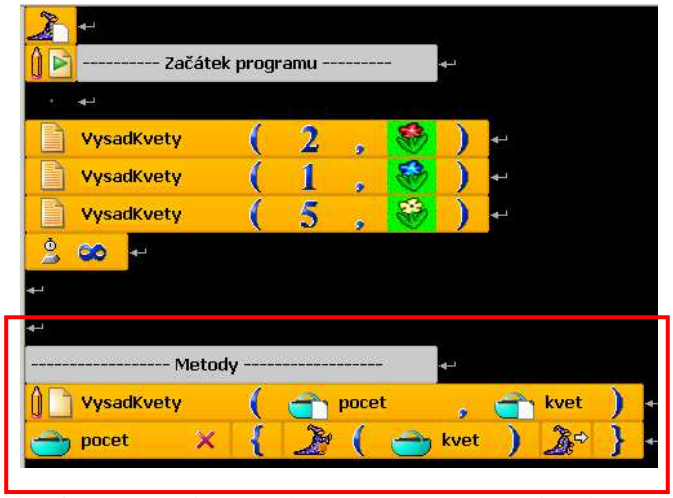
Obrázok 35: Realizácie procedúry nuholnik pri zamenených vstupných parametroch
(Zdroj: Autorka)

Kým volanie procedúry *„nuholnik 5 50*“ vykreslí kruh, ktorého dĺžka hrany je 5 korytnačích krokov a 50 udáva počet otočení korytnačky, volanie „*nuholnik 50 5“* vykreslí päťuholník s hranou dĺžky 50 korytnačích krokov.

V detskom programovacom jazyku Baltie 4C#, rovnako ako v iných programovacích jazykoch, existuje možnosť využívať a zapisovať časti algoritmu ako podprogramy – metódy.

Pri vysvetľovaní žiakom ukážeme, ako vytvoríme novú metódu stlačením ikonky *Metódy* na paneli nástrojov.

Otvorí sa okno, v ktorom je treba stlačiť ikonku vytvorenia novej metódy  . Na koniec hlavnej časti programu sa vloží hlavička novej metódy, ktorú je možné pomenovať a určiť jej typ. Na obrázku 36 je príklad zdrojového kódu obsahujúceho definíciu a volanie metódy.



Obrázok 36: Vytvorenie metódy v programovacom jazyku Baltie 4C#

(Zdroj: Autorka)

Pri pomenovávaní metódy je potrebné opäť si pripomenúť zmysluplnosť a výstižnosť jej mena, aby pri volaní metódy už z názvu vyplynulo, čo metóda robí. Vhodným príkladom môže byť napríklad pomenovanie metódy VysadTulipany, PostavDom a pod.

Obrázok 37: Ukážka metódy na vysadenie stanoveného počtu (štyri) kvetov

(Zdroj: Autorka)

Ak by sme chceli ovplyvniť, napríklad to, akej farby bude kvet, ktorý má čarodejník vyčarovať, musí byť pri tvorbe metódy použitý aj parameter. Parameter predstavuje lokálnu premennú danej metódy. Premenná je takým objektom, ktorý obsahuje hodnoty len presne určeného údajového typu. V programovacom jazyku Baltík sa môžu žiaci po prvýkrát stretnúť s údajovými typmi už pri prvku Literál. Literál je používaný v programovaní pri priamom zápise čísla alebo reťazca. Na prvku Literál sa mení podkladová farba poľa, do ktorého píšeme v závislosti od písanej hodnoty. V detskom programovacom jazyku Baltík majú údajové typy charakteristickú farbu pozadia, ktorá je totožná s farbou premenných.

Tabuľka 2: Údajové typy v Baltíku – prvok literál

Údajový typ	*Reprezentácia*	*Farba pozadia*	*Príklad*
string	jeden znak, skupina znakov	žltá	Ahoj
double, float	desatinné číslo, reálne číslo	zelená	3.14
int	celé číslo	tyrkysová	123

Rozlíšenie základných údajových typov a ktoré je potrebné určiť pri definovaní premenných. Takéto farebné rozlíšenie môže poslúžiť aj ako vítaná pomôcka, pre tých, ktorí sa učia programovať. Rýchlejšie a ľahšie si osvoja a budú používať správne priraďovanie hodnôt daného dátového typu do predtým definovaných premenných a následne budú vedieť realizovať korektné numerické operácie, ako napríklad násobiť, sčítavať, deliť, ale aj vypisovať text alebo obsah premennej na obrazovku a pod.

V Baltíku sú dva druhy premenných a to *globálne*, ktoré možno používať v triedach, v ktorých boli deklarované a *lokálne*, ktoré sa dajú použiť len v blokoch príkazov, pre ktoré boli deklarované.

Globálne premenné sú v Baltíku znázorňované ako zásuvky a lokálne premenné ako košíky. Aj toto rozlíšenie je pre výučbu veľmi názorné, lebo žiaci si uvedomia, kde ktorý typ premennej treba použiť.

Obrázok 38: Metóda s parametrom

(Zdroj: Žiaková, 2009)

Obrázok 39: Realizácia metódy s parametrom - z obrázku 38

(Zdroj: Žiaková, 2009)

V prípade ukážky zdrojového kódu metódy pre vyčarovanie kvetov prostredníctvom parametra, je vidieť, že premenná je vizualizovaná prostredníctvom košíka modrej farby, pretože sa jedná o premennú pre blok príkazov a predmet je reprezentovaný celým číslom.

V programovacom jazyku Baltík je možné vidieť iba v rámci okna pamäte, pri neobjektových premenných, globálne premenné ako zásuvky hnedej farby.

Vo výpise kódu v Baltíku sa pri deklarácii parametra na košíku objavil malý biely papierik, ktorý vizuálne odlišuje volanie parametra v tele metódy, ktorý je už bez papierika. Je to veľmi vhodné odlíšenie deklarácie od volania premennej, ktoré autor programovacieho jazyka neopomenul pri tvorbe jazyka.

Aj v uvedenom príklade s čarovaním kvetiniek je možné v zápise algoritmu definovať viac parametrov v jednej metóde (obr. 40). Druhý parameter bude nositeľom počtu kvetín

konkrétnej farby. A pri volaní metódy je možné podľa potreby vyčarovať rôzne množstvo kvetov odlišných farieb.

Obrázok 40: Metóda *VysadKvety* s parametrami *pocet* a *kvet*

(Zdroj: Žiaková, 2009)

Obrázok 41: Realizácia algoritmu pri využití metódy *VysadKvety*

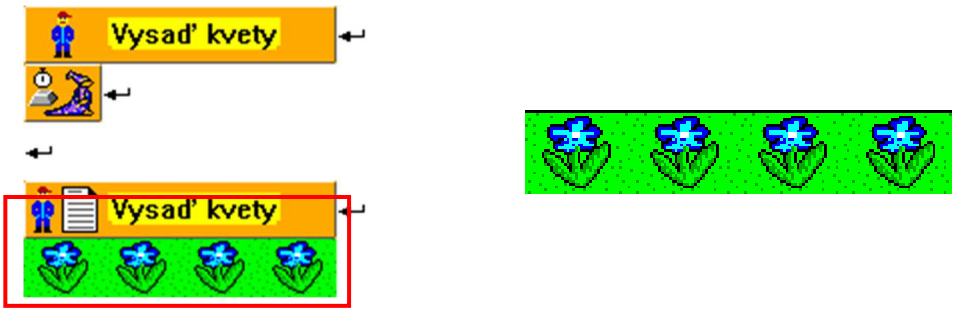
(Zdroj: Žiaková, 2009)

Vyššie opísaný algoritmus je možné následne zapísať vo verzii Baltík 3.0 pomocou Pomocníka a postup vysvetliť žiakom nasledovne: . Stlačiť tlačidlo Tabuľka pomocníkov . Otvorí sa  okno, v ktorom sa primárnym tlačidlom myši stlačí ikonka vytvorenia Nového pomocníka  . Na koniec hlavnej časti programu sa vloží hlavička nového pomocníka, ktorú je vhodné pomenovať.

Obrázok 42: Vytvorenie pomocníka v programovacom jazyku Baltík 3.0

(Zdroj: Autorka)

Pri pomenovávaní pomocníka, rovnako ako pri pomenovaní vyššie spomínanej procedúry v Imagine alebo metódy v Baltie 4C#, je treba mať na zreteli zmysluplnosť a výstižnosť mena, aby pri jeho volaní nebol problém určiť, čo pomocník robí. Vhodným príkladom pomenovania pomocníka (obr. 43) môže byť ako pri pomenovaní metódy, napríklad *VysadTulipany*, *PostavDom* a podobne.



Obrázok 43: Zdrojový kód a realizácia algoritmu s pomocníkom

(Zdroj: Autorka)

Ak by bolo potrebné ovplyvniť farbu čarovaného kvetu, je nutné pri tvorbe pomocníka použiť parameter. Parameter predstavuje lokálnu premennú daného pomocníka rovnako, ako to bolo pri metóde.

Obrázok 44: Pomocník pre vyčarovanie kvetov s parametrom v prostredí Baltik 3.0

(Zdroj: Autorka)

Obrázok 45: Realizácia algoritmu pomocníka z obrázku 44

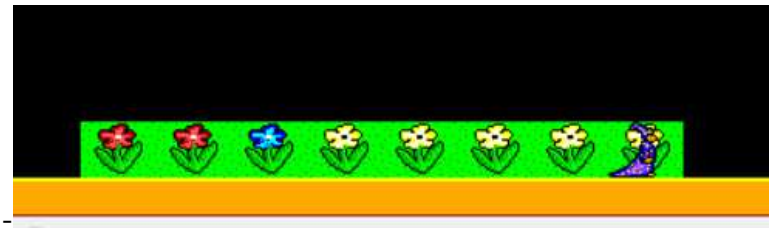
(Zdroj: Autorka)

Ako je vidieť na ukážke kódu pomocníka pre vyčarovanie kvetov prostredníctvom parametra na obrázku 44, premenná je vizualizovaná prostredníctvom košíka modrej farby, keďže ide o premennú pre blok príkazov a predmet z banky je reprezentovaný konkrétnym celým číslom.

Aj v uvedenom príklade, ktorého zdrojový kód je na obrázku 44 a realizácia na obrázku 45, je možné v zápise algoritmu definovať viaceré parametre v jednom pomocníkovi rovnako, ako to bolo metóde vo verzii Baltie 4C#. Druhý parameter aj v tomto prípade bude umožňovať zadať počet kvetín danej farby. Pri volaní metódy by sa potom mohli podľa potreby vyčarovať rôzne množstvá kvetov odlišných farieb.

Obrázok 46: Pomocník *Vysaď kvety* s parametrami udávajúcimi druh kvetu a počet

(Zdroj: Autorka)



Obrázok 47: Realizácia algoritmu pri využití pomocníka Vysaď kvety
(Zdroj: Autorka)

5.3 Rekurzia

Veľmi silným nástrojom nielen v programovaní, ale aj v matematike či biológii je rekurzia. Pre talentovaných žiakov ju môže učiteľ vhodnými príkladmi zaviesť už vo veku 12 - 14 rokov. Avšak zaviesť pojem rekurzia a žiakom vysvetliť princíp, na ktorom je založená, je dosť ťažké, pretože si vyžaduje istú dávku abstrakcie a predstavivosti u žiakov. Nutnosťou pre zavádzanie rekurzie je zvoliť veľmi vhodné slová a príklady blízke detskému mysleniu.

V tejto kapitole opisujeme možné návody pre pedagógov, ako vysvetliť princíp rekurzie a rekurzívny algoritmus pre tento vek. V detskom programovacom prostredí Imagine je rekurzia pomerne ľahko zvládnuteľná pre žiakov. Vhodným príkladom pre pochopenie významu slova rekurzia je predstava televíznej obrazovky, na ktorej je zobrazená televízna obrazovka a na tej ďalšia televízna obrazovka....

Na vysvetľovanie algoritmu rekurzie v detskom programovacom jazyku sa môže použiť napríklad aj nasledovná motivačná úloha:

Vytvorte lúku plnú kvetov, na ktorej budú iba jednofarebné kvety so žltým stredom v počte 10.

Úroveň vedomostí žiakov dovoľuje bez väčších problémov zadefinovať procedúru *kvietok* na vykreslenie kvetu a potom ju použiť v kombinácii s presunutím pozície korytnačky v príkaze **opakuj** desaťkrát.

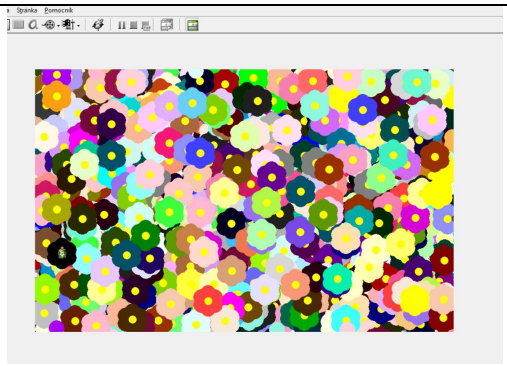
Jedno z riešení, ako je možné zadať parameter, ktorý bude určovať veľkosť kvetu a po každom vykreslení posunie korytnačku na náhodnú pozíciu je:

Príklad kódu procedúry *kvietok*:

```
viem kvietok : n
nechfp ?
opakuj 6 [ph do : n/2 pd bod : n ph vz : n/2 vp 60 ]
nechfp "žltá
bod : n/2
koniec
opakuj 10 [kvietok 20 nechpoz [? ?] cakaj 100]
```

Po úspešnom riešení môže byť zadanie úlohy zmenené tak, aby žiaci pri programovaní nepoužili na vykreslenie n kvetov príkaz **opakuj**.

Riešiť úlohu je možné tak, že žiak doprogramuje presunutie pozície korytnačky do procedúry na vykreslenie kvetu, aby nemusel 10 krát vypisovať jej presunutie po pracovnej ploche.
Ako by potom mohol vyzerať výpis? Desaťkrát volanie procedúry *kvietok 20*? A keby ju bolo potrebné volať viackrát, nebolo by to náročné na písanie? Za najvhodnejšie možno považovať zmenu kódu tak, aby na konci po vykreslení kvetinky a po presunutí korytnačky bola opäť volaná procedúra na vytvorenie ďalšej kvetinky. A táto nová kvetinka by volala na vytvorenie ďalšej kvetinky a tak ďalej. Je známe, že v procedúre je možné volať inú procedúru, ktorá bola predtým vytvorená v programe. Toto je moment, keď je možné zmeniť kód algoritmu tak, aby na konci procedúry *kvietok* bola zavolaná znova procedúru *kvietok*. Zdrojový kód takto zapísanej rekurzívnej procedúry a jeho realizácia by mohli vyzerať napríklad nasledovne:

Zdrojový kód rekurzívnej procedúry	*Realizácia*
viem *kvietok* : *n* nechfp ? opakuj 6 [ph do : n/2 pd bod : n ph vz : n/2 vp 60] nechfp "žltá bod : n/2 nechpoz [? ?] *kvietok* : *n* koniec	

Procedúra využívajúca rekurziu v svojom tele volá samu seba. Takéto volanie sa nazýva rekurzívne. Nakoľko je volanie samej seba posledným príkazom v procedúre hovorí sa o chvostovej rekurzii.

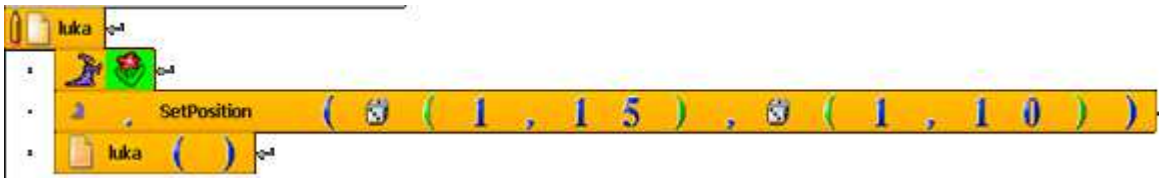
Aký môže byť počet vykreslených kvetov, keď každá ďalšia kvetinka sa vykresľuje vtedy, keď ju predchádzajúca zavolá? V podstate nekonečný, preto treba rekurziu vo vhodnom momente ukončiť pomocou naprogramovania podmienky, alebo ukončiť rekurzívne volanú procedúru manuálne prostredníctvom ikonky na nástrojovom paneli.

Rovnaké zadanie je možné riešiť aj v detských programovacích jazykoch Baltie 4C# a Baltík 3.0. V oboch prípadoch bude úloha riešená v rovine, aby boli algoritmy porovnateľné.

Metóda na vytvorenie lúky by mohla vyzerať napríklad takto:

Zmenu pozície Baltíka na nové miesto na ploche, kde bude opäť čarovať novú kvetinku je možné zabezpečiť pomocou príkazu **SetPosition** (nastav pozíciu), v ktorej je potrebné zadať súradnicu x a y, ktoré presne určujú, o ktorú pozíciu na pracovnej ploche sa jedná. V tomto prípade na generovanie náhodnej pozície je vhodné použiť príkaz (ikonu) generovania náhodného čísla pomocou príkazu  , ktorého parametre sú určené rozmermi pracovnej plochy scény.

Aj tomto prípade bude úloha realizovaná postupne, ako to bolo v prípade detského programovacieho jazyka Imagine. Ako prvý krok bude vytvorenie metódy prípadne pomocníka, v závislosti od verzie detského programovacieho jazyka Baltík, bez použitia cyklu. Tu je namieste ukázať aj spôsob, ako by bolo možné efektívne vytvoriť metódu a pomocníka „luka“, ak by Baltík po vyčarovaní kvetu a zmene svojej pozície znova zavolal tú istú metódu na vyčarovanie ďalšieho kvetu a následnú zmenu pozície. Ukážka zdrojového kódu algoritmu algoritmu v 2D režime programovacieho jazyka Baltie 4C#:



Aj v tomto prípade je potrebné vo vhodnom momente zastaviť čarovanie kvetov pomocou podmieneného príkazu pred rekurzívne volane metódy prípadne pomocníka.

V nasledujúcom kroku je potrebné preveriť správne pochopenie základného princípu jednoduchej chvostovej rekurzie. Na preverenie toho, ako žiaci pochopili rekurziu, možno zadať iný príklad: .

V noci sa vám sníval zvláštny sen, zmenili ste sa v ňom na módnych návrhárov. Vašou úlohou je tvoriť návrhy rôznych látok so vzormi. Zostavte algoritmus na tvorbu rôznych motívov látok, v ktorých sa využíva chvostová rekurzia.

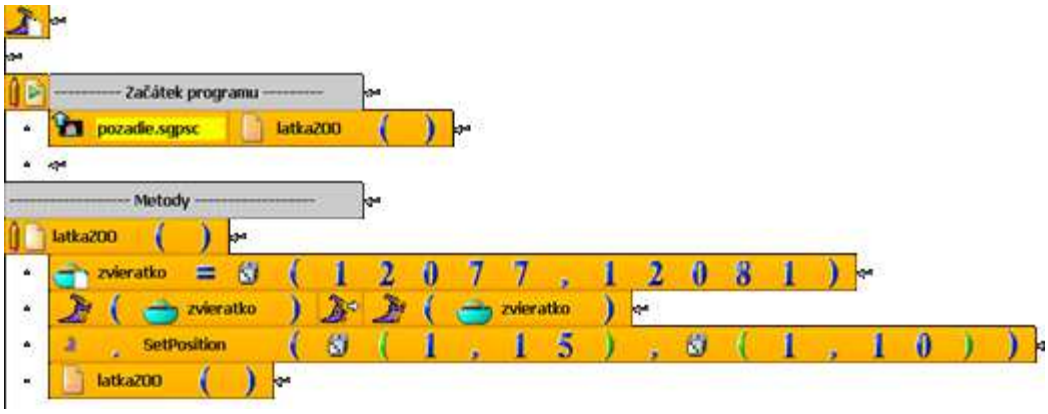
V závislosti od detského programovacieho prostredia je však nutné špecifikovať požiadavky vzoru. V detskom programovacom prostredí Imagine, ktorého rekurzia je oveľa silnejším nástrojom ako v prípade detského programovacieho jazyka Batlík, môže byť učiteľom úlohu bližšie špecifikovaná napríklad zadávaním požiadaviek na zmenu farebnej škály alebo veľkostí vzoru v rámci jednej látky. Inšpiráciou by mohla byť napríklad riešenie algoritmu vytvorenie látky z listov, ktoré majú rôzne odtiene napríklad zelenej farby, keďže sa jedná o listy a rôznu veľkosť. Procedúra látka môže vyzerať napríklad nasledovne pričom uvádzame zdroh zdrojový kód ale aj jeho realizáciu (Žiaková, 2009)

.Zdrojový kód rekurzívnej procedúry	*Realizácia*
viem *latka* list 15 +*nahodne* 15 ph vp *nahodne* 360 do 100 + *nahodne* 150 pd nechfp ?prvok [olivova3 olivova4 olivova5 olivova6 olivova7 zelena3 zelena4 zelena5 zelena6 zelena7] cakaj 1000 *latka* koniec	

Obrázok 48: Zdrojový kód a riešenie algoritmu -listová látka -Imagine
(Zdroj: Žiaková, 2009)

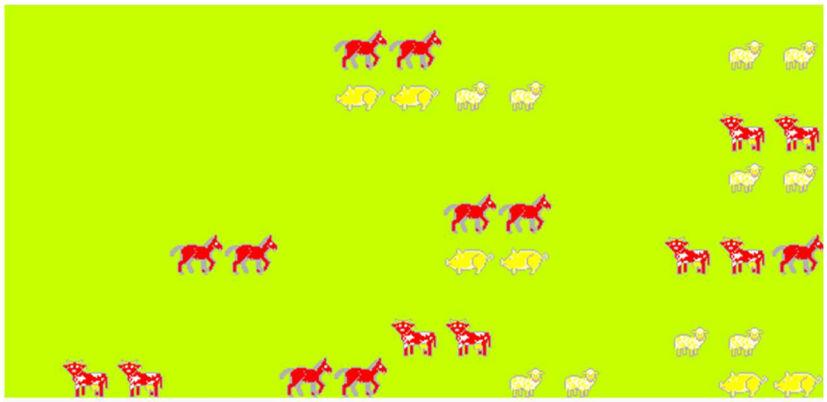
Analýzou kódu je možné vidieť, že zmenené veľkosti listov boli dosiahnuté tým tým, že v procedúre *list* je veľkosť parametra dĺžka generovaná pomocou procedúry *nahodne*, ktorá sa používa pri vykresľovaní listov. Pomocou volania *list 15 + nahodne 15* bolo zabezpečené, že dĺžka, ktorá je vstupnou hodnotou pre procedúru *list*, je vždy prirodzené číslo v našom prípade z intervalu <15,29>. Procedúru *nahodne* bola využitá aj pri generovaní počiatočnej pozície, kde bude vykresľovaný list, teda náhodnej pozície. Pre vykreslenie odtieňa zelenej farby bol využitý príkaz *nechfp ?prvok*, v ktorom je vymenovaný zoznam možných farieb, z ktorých počítač vygeneruje ľubovoľnú alternatívu, tým sa docielilo generovanie odtieňov zelenej farby.

V 2D režime v programovacom jazyku Baltie 4C# a ani Baltík 3.0 nie je možné meniť veľkosť a farbu predmetov priamo v zápise algoritmu programu, preto je potrebné zadať iné požiadavky na štruktúru, ktorými bude zabezpečený vzor na látke. Jedna z možností je využiť obrázky zvierat z banky predmetov, vzor bude tvorený rôznymi obrázkami zvierat, napríklad vždy v páre to isté zviera vedľa seba. Zdrojový kód tohto návrhu je na obrázku 49 a 50, ukážka realizácie zdrojového kódu navrhnutej látky je na obrázku 51.



Obrázok 49: Algoritmus na vytvorenie látky v programovacom jazyku Baltie 4C#
(Zdroj: Autorka)

Obrázok 50: Algoritmus na vytvorenie v programovacom jazyku Baltík 3.0
(Zdroj: Autorka)



Obrázok 51: Ukážka vytvorenej látky v 2D verzii oboch jazykov

(Zdroj: Autorka)

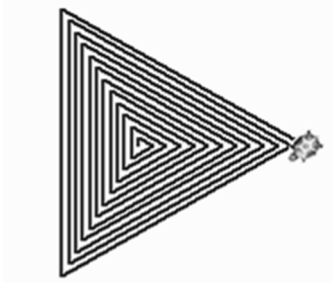
Pri generovaní rôznych zvierat bola využitá skutočnosť, že každý obrázok v banke obrázkov má svoje jedinečné číslo. Bolo potrebné vyčarovať dve identické zvieratá vedľa seba, využila sa premenná s názvom **zvieratko**, do ktorej bola priradená číselná hodnota vygenerovaného predmetu, aby mohol byt opätovne vyčarovaný.

Chvostová rekurzia môže byť použitá nielen pri nekonečnom vykresľovaní už hotových obrazcov na plochu, ale práve pri kreslení samotných obrazcov. Takáto definícia procedúr je veľmi jednoducho použiteľná práve v detskom programovacom jazyku Imagine.

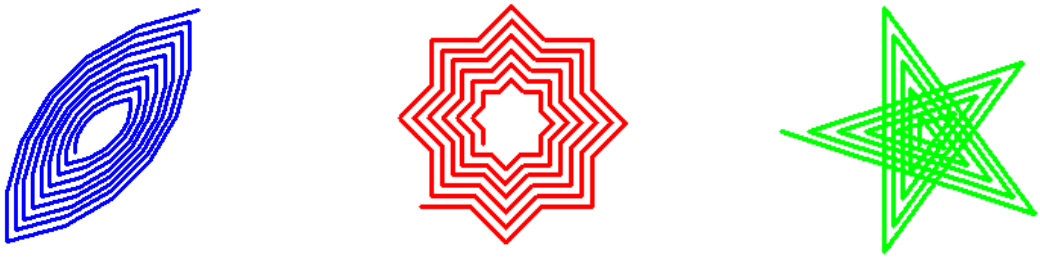
Pri definícii takýchto procedúr je vhodné začať ukážkou kreslenia jednoduchých geometrických útvarov ako sú napríklad kružnica, štvorec či trojuholník. Príklad procedúry na vykresľovanie trojuholníka je uvedený nižšie. V procedúre je však rekurzívne volanie s nezmenenou pôvodnou dĺžkou strany a teda, žiak neuvidí zmenu pri vykresľovaní obrázka.

```
viem trojuholnik : dlzka
do : dlzka vp 120
trojuholnik : dlzka
koniec
```

Je potrebné upraviť rekurzívne volanie tak, aby bolo vykresľovanie viditeľné. Procedúra s parametrom v prvom kroku vykreslí trojuholník so zadanou dĺžkou strany a následne bude o zadanú hodnotu zväčšovať ďalší vykreslený trojuholník, rekurzívny krok, kde je potrebné modifikovať pôvodnú hodnotu vstupného parametra tak, aby pri ďalšom rekurzívnom volaní vykreslená časť sa zväčšovala. Uvádzame ukážku možného zápisu prvého špirálovitého útvaru spoločne s kódom:

Zdrojový kód	*Realizácia*
viem *trojuholnik* : *dlzka* do : *dlzka* vp 120 cakaj 200 *trojuholnik : dlzka* + 5 koniec	

Námety na ďalšie jednoduché chvostové rekurzie využiteľné v detskom programovacom jazyku Imagine sú uvedené na obrázku 52.



Obrázok 52: Ukážka obrázkov na precvičenie jednoduchej chvostovej rekurzie
(Zdroj: Žiaková, 2009)

Vyvstáva otázka, ako by asi teraz vyzerala látka, keby bola použitá jednoduchá chvostová rekurzia aj na nakreslenie obrazcov? Dá sa predpokladať, že vďaka použitiu obrazcov by bola veľmi efektná.

Po zvládnutí jednoduchej chvostovej rekurzie je na mieste pokus o vysvetlenie princípu fungovania rekurzie kdekoľvek v programe. V tomto momente je však potrebné, aby bola zrejmá práca s podmienenými príkazmi.

Motiváciu však nie je potrebné hľadať iba v informatike, tu je možnosť nájsť ju v biológii, ale aj napríklad v literárnych prameňoch a využiť tak pri vyučovaní medzipredmetové vzťahy. Jednou z možností je teda využiť ako motiváciu klasickú slovenskú rozprávku, konkrétne *Rozprávku o kohútikovi a sliepo*čke. Celý dej sa v tejto rozprávke „odohráva rekurzívne“. Sliepočka nájde zrnko, chce ho prehltnúť, ale zabehne jej v gágore. Kohútik sa jej snaží pomôcť a uteká ku studničke zobrať jej trochu vody, keby studnička dala vodu kohútikovi, ktorý by ju odniesol sliepočke by bolo všetko v poriadku. Lenže studnička odpovedala, že mu nedá vody, kým jej nedonesie od panej kľúče. Keby pani dala kľúče, studnička by dala vodu a sliepočka by sa napila, ale nie je to také nenáročné. Pani nechce dať kľúče, kým nedostane od pána list. Ten chce najprv od husi pero. A tak kohútik beží k husi, ktorá sa tiež nezľutovala nad sliepočkou a žiada od kosca trávu. Ten chce najprv od pekára chleba a pekár zasa od horára dreva. Takto by mohol chudák kohútik behať neustále k niekomu ďalšiemu, keby neprišlo k tej šťastnej udalosti, že horár nič nechcel a daroval kohútikovi pre pekára dreva, pekár dal koscovi chleba, kosec dal pre hus trávu, húska dala pánovi pero, pán dal panej list a pani dala studni kľúče a studňa dala kohútikovi vody, sliepočka sa napila a zrnko jej vyskočilo z gágora. Treba si uvedomiť, že podstatným momentom v tejto rozprávke je chvíľa, kedy horár dal kohútikovi drevo a týmto okamihom sa už rozprávka prestane ďalej rozvíjať a začne sa proces postupného návratu cez jednotlivé osoby späť až ku sliepočke, kde sa to celé vlastne začalo.

Práve na odvíjaní deja rozprávky je možné vysvetliť aj známy algoritmus na výpočet faktoriálu prirodzeného čísla. Tento príklad je však vhodné použiť až pre žiakov strednej školy alebo prípadne pre žiakov končiaceho ročníka základnej školy so záujmom o matematiku. Najskôr je však potrebné názorne vysvetliť, čo je faktoriál.

Jeden z možných postupov:

Zapísať na tabuľu nasledovné riadky:

1
1.2 = 2
1.2.3 = 6
1.2.3.4 = 24
1.2.3.4.5 = 120
1.2.3.4.5.6 = 720

Pri výpočte faktoriálu je potrebné upozorniť na to, že vždy sa jedná o súčin všetkých celých čísel od jednotky až po určené číslo. Podľa hodnoty určeného čísla sa hovorí o faktoriále daného čísla. Samotný faktoriál je označovaný znakom výkričník napísaný za číselnú hodnotu. Teda napríklad 3! = 6, lebo je známe, že 1.2.3=6. Je však potrebné definovať, že 1! = 1 a môže byť definovaný aj 0! = 1.

Následne je vhodné pokúsiť sa tieto poznatky využiť na prepis predchádzajúceho takto:

1! = 1
2! = 1.2 = 1!.2 = 2
3! = 1.2.3 = 2!.3 = 6
4! = 1.2.3.4 = 3!.4 = 24
5! = 1.2.3.4.5 = 4!.5 = 120
6! = 1.2.3.4.5.6 = 5!.6 = 720

A čo má spoločné rozprávka s faktoriálom? Je zrejmé, že vypočítať napr. 6 faktoriál je relatívne jednoduché. Postačí vynásobiť 5 faktoriál číslom šesť. A práve v tomto okamihu sa postup začne nápadne podobať spomínanej rozprávke. Bez problémov vieme odpovedať, akú hodnotu nadobúda faktoriál šestky, len predtým je potrebné zistiť akú hodnotu nadobúda faktoriál päťky, a tak „utekáme“ spolu s kohútikom zistiť akej hodnote sa rovná faktoriál päťky. A tak „utekáme“ spolu s kohútikom zisťovať faktoriál päťky, len je predtým potrebné zistiť, akej hodnote sa rovná faktoriál štvorky. A tak „utekáme“ spolu s kohútikom zisťovať, koľko je faktoriál štvorky. Niet nič jednoduchšie, veď stačí vynásobiť faktoriál 3 štvorkou. Ale koľko je faktoriál trojky? Treba to ísť zistiť. Ale faktoriál trojky povie: „poviem akú mám ja hodnotu, ale potrebujem vedieť, koľko je faktoriál dvojky.“ Faktoriál dvojky sa tiež bude vykrúcať, že potrebuje vedieť hodnotu faktoriálu jednotky. Až faktoriál jednotky sa nad počítajúcimi zľutuje a dá výsledok presne tak, ako sa v rozprávke horárovi uľútostilo kohútika. A môže už bežať späť, lebo má pre faktoriál dvojky hodnotu faktoriálu jedna a to jednotku. Vypočítať faktoriál dvojky, čo predstavuje hodnotu dva a teda sa môžeme vrátiť k faktoriálu trojky. Ten už, samozrejme, teraz bude ochotný dať výsledok šesť a teda máme výsledok pre faktoriál štvorky a ten dá výsledok dvadsaťštyri pre faktoriál päťky, ktorý konečne vieme vypočítať. Faktoriál čísla päť je konečne známy, lebo mal naporúdzi všetky potrebné hodnoty a dá výsledok sto dvadsať.

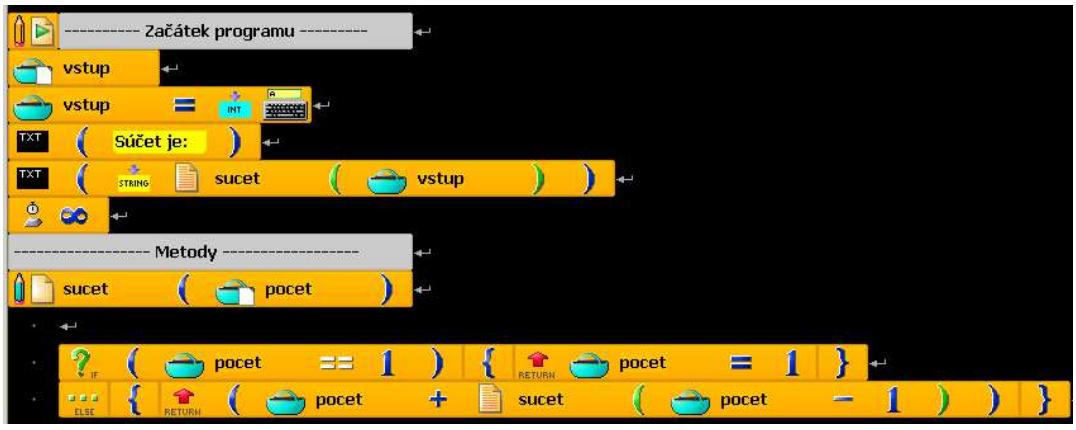
Túto situáciu je možné naprogramovať v detskom programovacom jazyku Imagine. Podľa Belana (2004) môže zápis zdrojového kódu na výpočet faktoriálu zadaného prirodzeného čísla v detskom programovacom jazyku Imagine vyzerať nasledovne:

viem *faktorial* :*n*
ak2 :*n* = 1
[výsledok 1]
[výsledok : *n* * *faktorial (: n - 1)*]
koniec

Postupne sa zistí, ako vyzerá realizácia zapísaného algoritmu, napríklad pre hodnotu 3, po zadaní príkazu *pis faktorial 3*? Je vhodné simulovať realizáciu algoritmu slovným opisom celého postupu a výpočtu procedúry *faktorial* so vstupným parametrom *n=3*. Je zrejmé, že výpočet faktoriálu nie je jediná vhodná matematická úloha, v ktorej je na riešenie úlohy využitá rekurzia. Pomocou rekurzie je možné riešiť aj nasledovné úlohy:

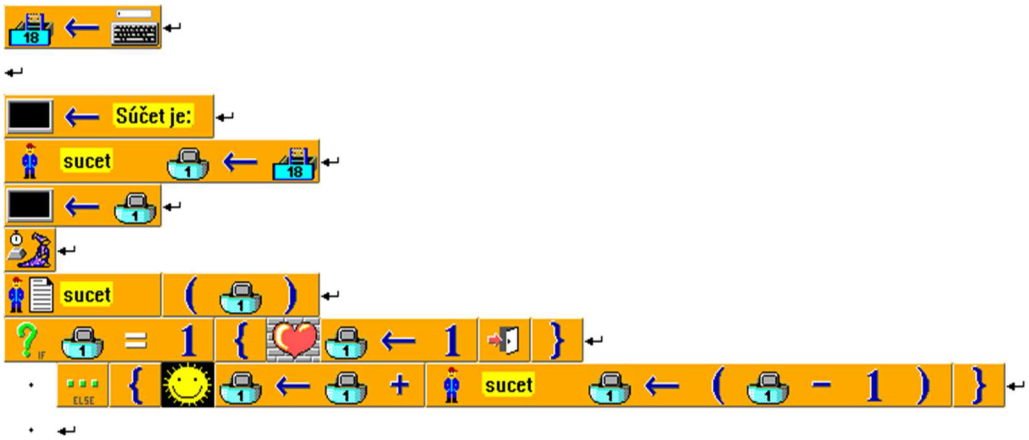
- Napíšte procedúru *sucet : n*, ktorá spočíta všetky čísla od 1 do n. Napríklad *sucet 10* bude 1+2+3+4+5+6+7+8+9+10= 55
- Napíšte procedúru *ParnySucet : n*, ktorá spočíta všetky neparne čísla od 1 do n. Napríklad *ParnySucet 9* bude 2+4+6+8= 20

Rekurzia ako taká, je samozrejme využiteľná aj v detskom programovacom jazyku Baltík 3.0 či Baltie 4C#, ako sme už vyššie uviedli. Zdrojový kód algoritmu súčtu všetkých prvých čísiel od jedna po *n* v obidvoch verziách Baltíka sa nachádza na obrázkoch 53 a 54.



Obrázok 53 Zdrojový kód v programovacom jazyku Baltie 4C#

(Zdroj: Žiaková, 2009)

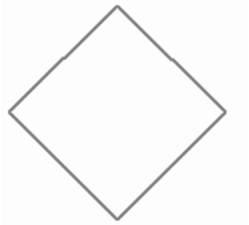
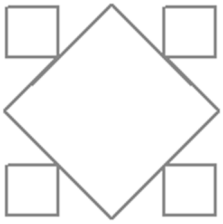
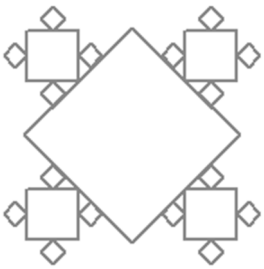


Obrázok 54: Zdrojový kód v programovacom jazyku Baltík 3.0

(Zdroj: Autorka)

V mikrosvete Imagine existuje aj iná možnosť, ako zaujímavým spôsobom priblížiť žiakom algoritmy využívajúce rekurziu. Je to vytváranie tzv. fraktálov. Fraktály sú geometrické krivky, ktoré sú na prvý pohľad zložité obrazce, ale spravidla sú generované opakovaným volaním jednoduchej matematickej štruktúry, krivky.

Pri tomto spôsobe je potrebné, aby sa najskôr napísal zdrojový kód na kreslenie základných obrázkov ako napríklad (štvorec, kruh, bodka). Následne pomocou základných procedúr postupne programovať ďalšie procedúry slúžiace na vykreslenie predkladaných obrazcov. Tieto obrazce vždy vychádzajú zo základného útvaru naprogramovaného v pôvodnej procedúre, ktorý sa v ďalších využíva. Napríklad u základného obrazca štvorca je možné úlohu modifikovať spôsobom jeho vykreslenia. Ukážka takýchto obrazcov, s postupnou tvorbou a s príslušnými procedúrami bez využitia rekurzie je na obrázku 55.

Predložený obrázok:	*Procedúra*
	viem *stvorec :dlzka* nechfp "modra vl 45 opakuj 4 [do : *dlzka* vp 90] vp 45 koniec
	viem *inystvorec :dlzka* vl 45 opakuj 4 [do :*dlzka/2* vl 90 *stvorec :dlzka/3* vp 90 *do : dlzka/2* vp 90] vp 45 koniec
	viem *dalsistvorec : dlzka* vl 45 opakuj 4 [do : *dlzka/2* vl 90 *inystvorec : dlzka/3* vp 90 do : *dlzka/2* vp 90] vp 45 koniec

Obrázok 55: Ukážky jednoduchých fraktálov a kódu bez využitia rekurzie

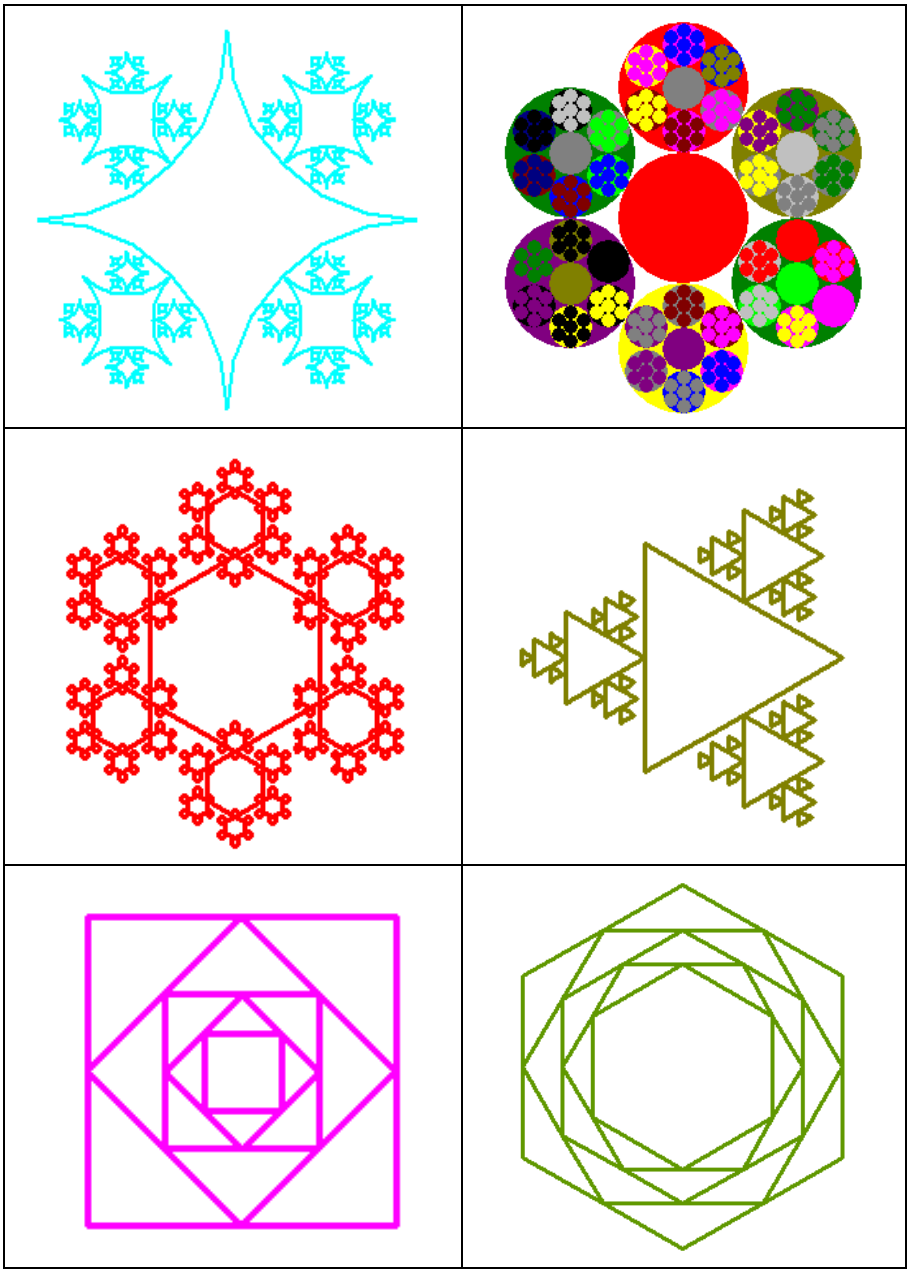
(Zdroj: Autorka)

Ako by mohla vyzerať procedúra, ktorá by nakreslila ešte menšie štvorčeky na hranách najmenších štvorčekov z tretieho predloženého obrázka? Je vidieť, že procedúry *inystvorec* a *dalsistvorec* sa líšia iba jediným riadkom. A práve o tento riadok sa líši aj nová procedúra.

Inou možnosťou riešenia je, procedúra, v ktorej sa zavedie parameter na udávanie počtu rôznych veľkostí štvorca.

```
viem inystvorec : dlzka : u
 Ak : u = 0[ukonci]
 vl 45 opakuj 4 [do : dlzka/2 vl 90
                 inystvorec : dlzka/3 : u-1 vp 90
                 do : dlzka/2 vp 90] vp 45
koniec
```

Pomocou uvedených procedúr sa dajú vytvárať skutočne krásne geometrické vzory, ktoré pozývajú experimentovať ďalej. Na obrázku 56 je niekoľko námetov, ktoré umožňujú precvičiť rekurziu pomocou vykresľovania obrázkov. Náročnosť fraktálov je nutné voliť v závislosti od vedomostnej úrovne skupiny detí, s ktorou pracujeme.



Obrázok 56: Námety na precvičenie rekurzie v programovacom jazyku Imagine
(Zdroj: Žiaková, 2009)

Exituje množstvo námetov na vyučovanie programovania v detských programovacích jazykoch, ktoré v závislosti od vyučovaného jazyka či úrovne vedomostí žiakov je možné zaradiť do vyučovania. Vhodnou motiváciou môže programovanie hry. Gamifikácia, či vyučovanie programovania prostredníctvom programovania hier je ďalšou veľmi atraktívnou oblasťou vyučovania programovania na základných a stredných školách.

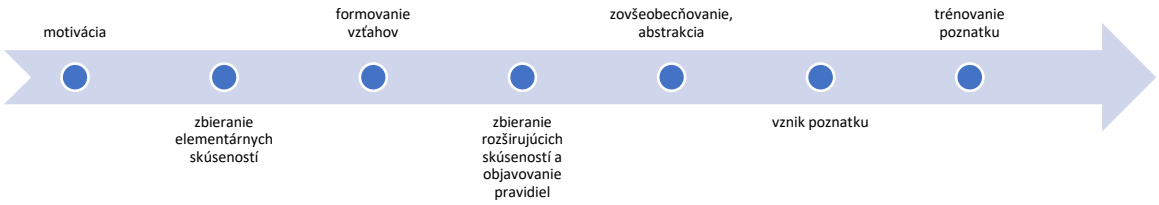
6 Námety na programovanie v detských programovacích jazykoch

Je veľmi dôležité, aby vyučovanie programovania bolo pre žiaka dostatočne motivujúce. Exituje niekoľko pohľadov na to, čo je motivácia:

- Je to vnútorný stav dodávajúci človeku energiu, aby dosiahol stanovený cieľ (Kassin, 2012).
- Je to zámerné nasadzovanie energie zabezpečujúce dosiahnutie stanoveného cieľa. K jeho dosiahnutiu je nevyhnutná disciplinovanosť, vytrvalosť, ale aj trpezlivosť, nikdy sa však nemôže nasadzovanie energie chápať ako povinnosť (Petty, 2008).

Motiváciu možno považovať za stimul, s ktorým je potrebné, aby sa žiak stotožnil. Pozitívna, ale aj negatívna motivácia v škole môže veľmi silne interferovať školskú úspešnosť žiaka, jeho výkony a rozvoj žiackej osobnosti. Pozitívna motivácia je jednou z podmienok efektívneho učenia sa, ovplyvňuje koncentráciu, pamäťové pochody, ale aj výdrž v učení. Žiaci, ktorí zažívajú potešenie z učenia sa a cítia sa pri ňom šťastní, vnímajú svoj život v škole pozitívne. Ak však pocítia kontrolu a usmerňovanie, či až riadenie zvonku ako súčasť vzdelávacieho procesu, tak nebudú motivovaní. Na druhej strane existuje aj veľa situácií, ktoré naopak spôsobujú znižovanie žiackeho výkonu. Tieto zvyčajne spôsobujú frustráciu žiaka sprevádzanú nudou, strachom, úzkosťou, ale aj nadmernou motiváciou. Pri akejkoľvek činnosti je nutná správna a skorá spätná väzba zameriavajúca sa na jej priebeh prípadne na jej výsledok. Nikdy však nie je vhodné posudzovať kvality žiaka, ktorý danú činnosť robí.

Poznatky sa u žiakov formujú po určitých etapách, ktoré je možné názorne zobraziť nasledovne:



(Zdroj: Autorka)

Znázornené etapy formovania poznatkov vychádzajú z didaktiky matematiky, ktorej základy sú ukotvené v teórii profesora Hejného (Hejný & a kol., 1990, Hejný, 2003). Autori tvrdia, že nový (matematický) poznatok môže žiak získať dostatočnými skúsenosťami a ich preusporiadaním.

Žiaka bavia také úlohy, ktorých stupeň náročnosti riešenia zodpovedá jeho najvyšším schopnostiam. Okrem prijateľného stupňa náročnosti je potrebné, aby bol pre žiaka zaujímavý aj kontext riešenej úlohy.

Zadanie úloh využívaných na vyučovanie programovania je potrebné orientovať nielen na návrh nových algoritmov, ale žiaci musia preukázať, že napísaným algoritmom aj rozumejú, teda chápu zadanie úlohy.

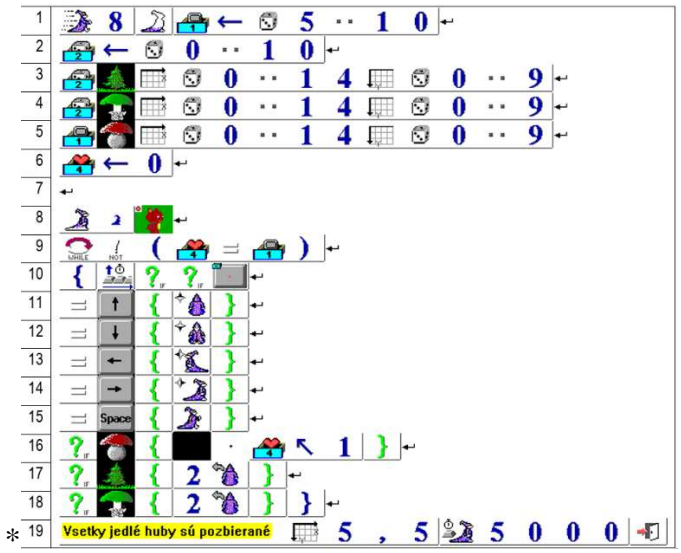
Na základe údajov rôznych autorov (Pečiovský, 2001; Blaho, Kalaš, 2005) a tiež vlastných poznatkov a skúseností autorky (pôsobí ako učiteľka informatiky), z vyučovania detských programovacích jazykov na základnej, strednej, ale aj vysokej škole uvádzame niekoľko riešených i neriešených námetov na zadania v detskom programovacom jazyku Baltík 3.0 a Imagine využiteľných v pedagogickej praxi učiteľa.

V nasledujúcej časti uvádzame niekoľko príkladov s grafickými výstupmi a zdrojovým kódom tak, ako boli využívané v pedagogickej praxi[3].

Príklad 1: Zber jedlých hríbov

Na obrázku 57 sa nachádza zdrojový kód k príkladu. Správne odpovede na nasledovné otázky, pomôžu zistiť jeho realizáciu:

1. Aká je podmienka vykonávania príkazov v cykle?
2. Kedy Baltík urobí krok?
3. Čo sa stane, keď sa Baltík ocitne pred stromčekom?
4. Čo sa stane po vykonaní príkazov v riadku 1 a kde a na čo sa použije v nasledujúcej časti programu premenná 1?
5. Čo je úlohou premennej 4?
6. Čo sa stane po vykonaní príkazov v riadku 8?
7. Čo sa stane po vykonaní príkazov v riadku 16?
8. V ktorých riadkoch sa nachádza celý príkaz, ktorého výsledkom je pohyb Baltíka (pomôcka: Pohyb Baltíka je riadený šípkami).
9. Čo je výsledkom realizácie príkazu v riadku 19?
10. V ktorých riadkoch sa nachádzajú príkazy vykonávané v tele cyklu riadeného podmienkou?



Obrázok 57: Zdrojový kód k príkladu 1

(Zdroj: Autorka)

[3] Poznámka autorky: Zadania príkladov nachádzajúcich sa v kapitole 6 boli vypracované v spoluautorstve s Ing. Beatou Janíčkovou a Ing. Ľubicou Pechanovou. Autorka so spoluautorkami overovali vhodnosť a účinnosť vybraných úloh počas spoločného pôsobenia na Gymnáziu Jána Hollého v Trnave v rokoch 2009 až 2019. Obe spoluautorky vyjadrili súhlas s uverejnením týchto úloh v predkladanej publikácii.

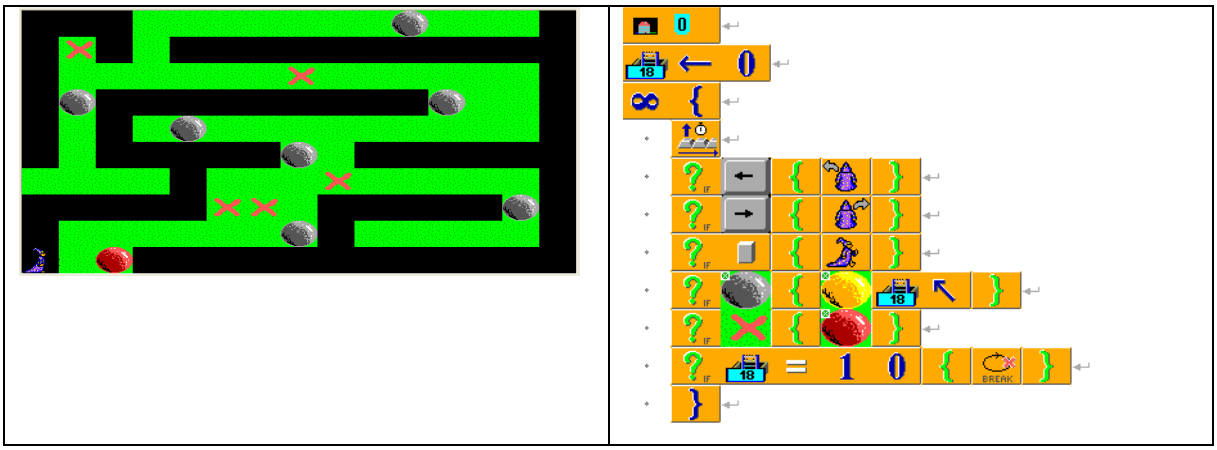
Na základe zistených údajov je možné vymyslieť a naformulovať alternatívne znenie úlohy.

Nasledujúce alternatívne príklady Bludisko A a Bludisko B môžu byť použité vo vyučovacom procese ako zadanie pre dve verzie písomnej práce, pri overovaní vedomostí z vetvenia a jeho využitia v príkladoch.

Príklad 2: Bludisko A

Na obrázku 58 sa nachádza scéna 0 a zdrojový kód úlohy, v ktorej je v scéna použitá. Správne odpovede na nasledujúce otázky pomôžu zistiť realizáciu príkladu na základe analýzy a porozumenia zdrojového kódu príkladu:

1. Cez ktoré predmety môže Baltík prechádzať?
2. Čo urobí Baltík, keď príde pred červený krížik?
3. Čo urobí Baltík, keď sa stlačí šípka vľavo
4. Kedy sa zmení hodnota premennej a ako?
5. Kedy sa cyklus skončí?



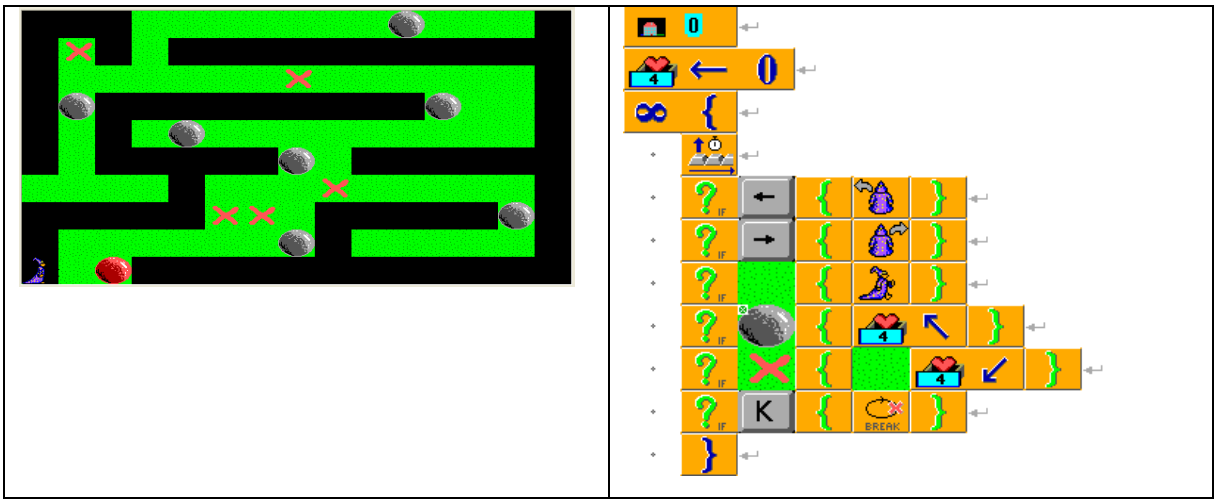
Obrázok 58: Kód a scéna k príkladu 2

(Zdroj: Autorka)

Príklad 3: Bludisko B

Na obrázku 59 sa nachádza scéna 0 a zdrojový kód úlohy, v ktorej je v scéna použitá. Správne odpovede na nasledujúce otázky pomôžu zistiť realizáciu príkladu na základe analýzy a porozumenia zdrojového kódu príkladu:

1. Cez ktoré predmety môže Baltík prechádzať?
2. Kedy sa Baltík otočí doprava?
3. Kedy sa zväčší hodnota premennej?
4. Čo urobí Baltík, keď príde ku krížiku?
5. Čo sa stane, keď sa stlačí kláves K?



Obrázok 59: Kód a scéna k príkladu 3

(Zdroj: Autorka)

Didaktická reflexia:

Uvedené príklady je možné využiť vo vyučovacom procese v prípade, ak chceme, aby žiak na základe vyhodnotenia zapísaného zdrojového kódu identifikoval zadanie úlohy, následne sformuloval jej možné zadanie. Je to jedna z možností vyučovať programovanie prostredníctvom čítania a porozumenia zapísaného zdrojového kódu programu. Tento postup, v ktorom žiak na základe riešenia úlohy hľadá jeho zadanie napomáha nielen v rozvoji jeho programátorských zručností, ale aj jeho predstavivosti a algoritmického myslenia.

Zadanie príkladov môže byť zasadené do zaujímavého príbehu.

Príklad 4: Pozbieraj tulipány

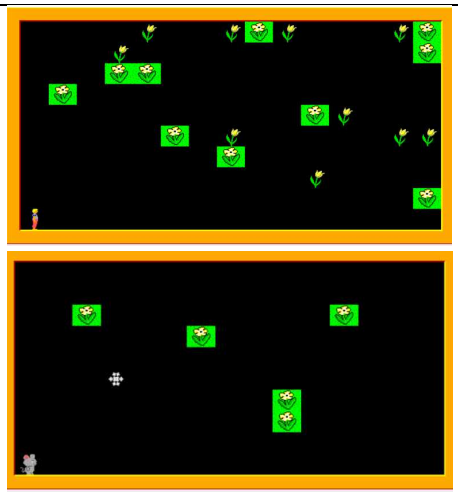
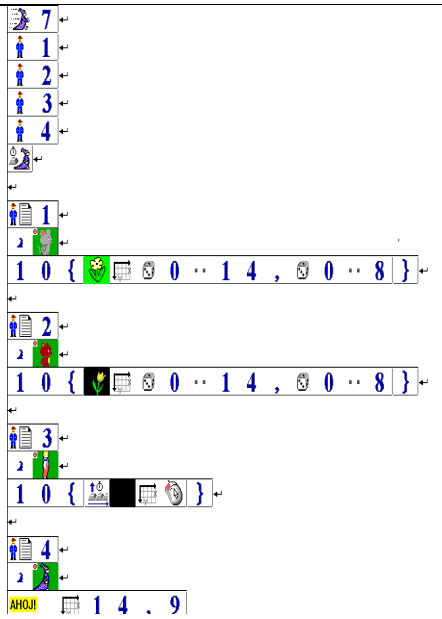
Baltík pripravil pre deti hru. Musia pozbierať všetky posadené žlté tulipány. K realizácii úlohy majú po ruke dvoch pomocníkov, medvedíka – predmet 10081 (banka predmetov – karta 10 predmet 081) a myšku – predmet 10121 (banka predmetov – karta 10 predmet 121). Malý pomocníci medvedík a myška dostali za úlohu posadiť na ľubovoľné miesto na poli (pole sa nachádza na pracovnej ploche objektu a je vymedzené nasledujúcou definíciou v zadaní), okrem riadku pri ceste, 20 žltých tulipánov (každý po 10 kusov). Myška je ale neposlušná predbehla medvedíka a posadila na ľubovoľné miesta namiesto žltých tulipánov iné žlté kvietky. Deti však musia pozbierať iba všetky žlté tulipány.

Postupným riešením nasledujúcich bodov príkladu dostanete riešenie príkladu. Jedno z možných riešení príkladu Pozbieraj tulipány spoločne s možným zdrojovým kódom sa nachádza na obrázku 60.

Naprogramuj štyroch Baltíkových pomocníkov a pomenuj ich Pomocník 1 až 4. Pomocníci budú robiť nasledovné úlohy:

1. Pomocník **1**
   - Zmení Baltíka počas svojej práce na myšku – predmet 10121
   - Posadí 10 žltých kvetov na náhodné súradnice okrem posledného (spodného) riadku (definícia cesty a poľa)
2. Pomocník **2**
   - Zmení Baltíka počas svojej práce na medvedíka – predmet 10081
   - Posadí 10 žltých tulipánov na náhodné súradnice okrem posledného (spodného) riadku (definícia cesty a poľa)
3. Pomocník **3**
   - Zmení Baltíka počas svojej práce na chlapca – predmet 9061 (banka predmetov – karta 9 predmet 061)
   - Pomocou myši pozbiera všetky žlté tulipány.
4. Pomocník **4**
   - Vráti Baltíkovi jeho pôvodnú podobu
   - Do pravého dolného políčka vypíše prostredníctvom prvku Literál AHOJ!

Súbor ulož do svojej zložky, pod názvom **Tvoje priezvisko_tulipány_A.bpr** .

Riešenie	*Zdrojový kód*
	

Obrázok 60: Kód a riešenie k príkladu 4

(Zdroj: Autorka)

Didaktická reflexia:

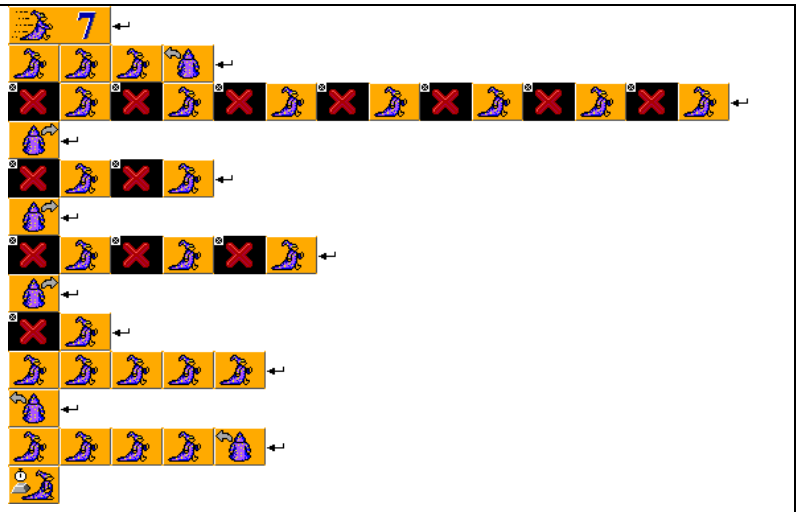
Príklad Pozbieraj tulipány môže byť využitý vo vyučovacom procese v prípade, ak by mal žiak na základe kontextového, zvyčajne problémového, zadania príkladu najskôr identifikovať podstatu príkladu, následne ju vyselektovať a navrhnúť možný zdrojový kód. Príklad môže byť použitý napríklad k preverovaniu vedomostí žiakov, pri práci s pomocníkom a zmene tvaru Baltíka.

V rámci propedeutiky programovania je žiadúce, aby pri vyučovacom procese boli využívané aj vizualizácia a krokovanie realizácie algoritmu. Krokovanie algoritmu a simulácia činnosti objektu detského programovacieho jazyka je veľmi silný prostriedok pochopenia realizácie programu.

Príklad 5 – Poskladaj obrázok zo znakov X

Na obrázku 61 je zapísaný zdrojový kód príkladu dvomi spôsobmi. Vľavo jednoduchými kódovanými príkazmi, ktoré boli vopred dohodnuté so žiakmi. Vpravo pomocou zdrojového kódu v detskom programovacom jazyku Baltík 3.0.

K K K VĽ
X K **X** K **X** K **X** K **X** K **X**
K **X** K
VP
X K **X** K
VP
X K **X** K **X** K
VP
X K
K K K K K
VĽ
K K K K VĽ



Obrázok 61: Zadanie úlohy zápis kódovaného programu a zdrojového kódu

(Zdroj: Autorka)

Príklad Poskladaj obrázok zo znaku X simuluje pohyb Baltíka po pracovnej ploche a činnosti, ktoré vykonáva.

K samotnej realizácii je možné pristupovať dvomi spôsobmi.

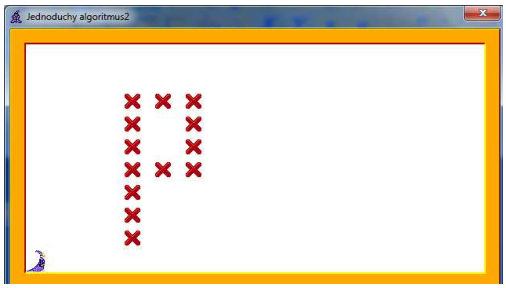
Prvý spôsob je podmienený vymedzením pracovnej plochy na vopred pripravenom papieri so štvorcovou sieťou, ktorá je zodpovedajúca pracovnej ploche Baltíka. Pri zakreslení krokovania do pripravenej štvorcovej siete na papieri je potrebné postupovať podľa nasledovných pokynov.

Zoberte si pripravený štvorčekový papier.

1. Skontrolujte si, či pracovná plocha, ktorú ste Baltíkovi pripravili má potrebný počet políčok. Aký má mať rozmer? (Žiak by mal správne odpovedať 150 políčok, rozložených do 10 radov a 15 stĺpcov.)

2. Simulujte realizáciu algoritmu objektom – postup zakreslite na pripravenú Baltíkovu plochu (štvorčekový papier):

Na obrázku 62 uvádzame riešenie získané z realizácie kódu



Obrázok 62: Riešenie príkladu 5

(Zdroj: Autorka)

Druhý spôsob spočíva v tom, že stanovíme žiaka za objekt programu (stane sa z neho čarodejník Baltík). Následne je vyzvaný k tomu, aby sa po pripravenej ploche v triede (vykreslená kriedou na podlahu, vyhradená stoličkami, vyskladaná papiera položeného na zemi) pohyboval.

Program, ktorý má žiak (objekt) realizovať dostane v papierovej forme v podobe obrázka 61.

Didaktická reflexia:

Je veľmi dôležité, aby žiak dokázal simulovať realizáciu algoritmu buď pohybovaním sa po triede, alebo jeho zakreslením na papier. Ak má žiak problém s čítaním zakódovaných príkazov môže mu byť poskytnutý zdrojový kód. Krokovanie algoritmu je využívané pri ladení programov aj vo vyšších programovacích jazykoch.

Príklad 6: Obrazce zo zvočekov

Na obrázku 63 je zobrazené riešenie zadania. Pri riešení postupujte podľa nasledujúcich pokynov.

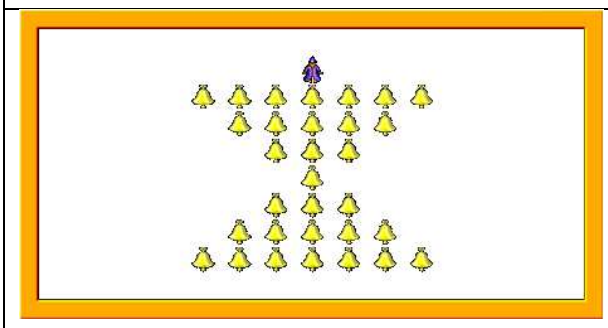
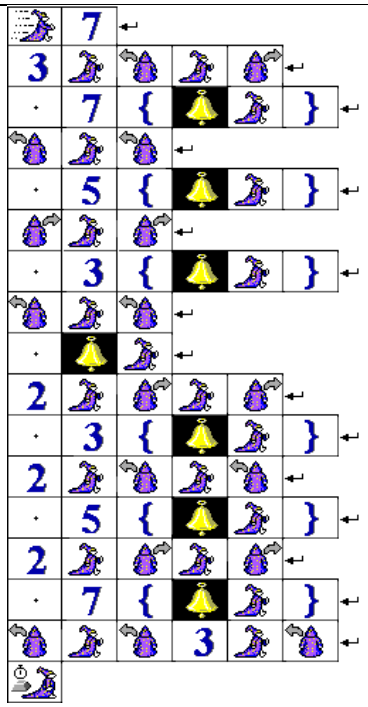
1. Zostavte program, pomocou ktorého objekt pri realizácii algoritmu zobrazí na pracovnej ploche obrazec podľa predlohy.
2. Pri tvorbe algoritmu a jeho zápise v zdrojovom kóde využite príkaz opakuj.

3. Dodržte:

   a) vzhľad objektov, z ktorých je zostavený obrazec
   b) umiestnenie obrazca na obrazovke
   c) pozíciu a orientáciu Baltíka pri ukončení realizácie programu.

4. Súbor je potrebné uložiť do zložky pod názvom **PresypHod-A-priezvisko.bpr** .

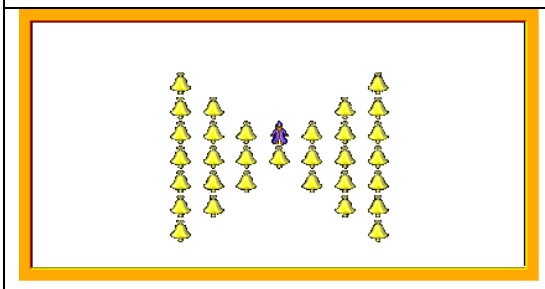
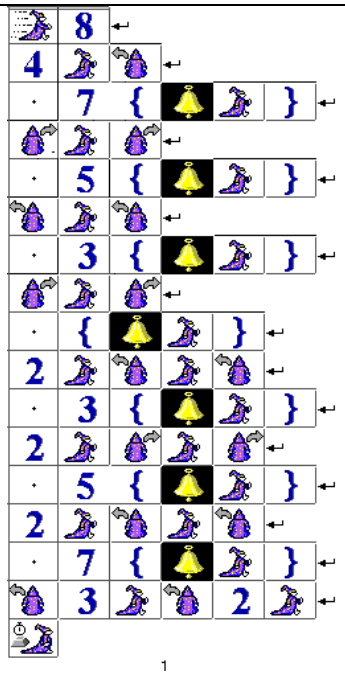
Okrem predlohy k príkladu 6 sa na obrázku 63 nachádza aj zdrojový kód riešenia.

Predloha	*Zdrojový kód*
	

Obrázok 63: Predloha a zdrojový kód príkladu 6 verzia A

(Zdroj: Autorka)

Na obrázku 64 sa nachádza jedna z možných alternatívnych predlôh a zdrojový kód k príkladu 6. Túto altenatívnu predlohu je vhodné kvôli rozlíšeniu označiť napríklad písmenom B:

Predloha	*Zdrojový kód*
	

Obrázok 64: Predloha a zdrojový kód k zadaniu príkladu 6 verzia B

(Zdroj: Autorka)

Didaktická reflexia:

Je veľmi dôležité, aby dokázal žiak zostaviť algoritmus na základe predlohy. Tieto predlohy sú námetmi na niekoľko alternatívnych predlôh, ktoré sú využiteľné pri precvičovaní príkazu opakuj.

Príklad 7 – Jednoduché geometrické útvary

V detskom programovacom jazyku Baltík je primárne pracovná plocha rozdelená na 15x10 štvorcov. Detský programovací jazyk Baltík medzi svojimi príkazmi k dispozícii aj grafické príkazy ako sú kružnica, čiara, obdĺžnik, preto je nutné oboznámiť sa aj s bodovým rozdelením Baltíkovej grafickej plochy. Nakoľko každý štvorec sa skladá z 39x29 bodov, rozmer Baltíkovej plochy v bodoch je rovný 585 x 290. V ľavom hornom rohu scény má bod súradnice x=0 y=0, bod v pravom dolnom rohu scény má súradnice x=584 y=289.

Postupne je potrebné zostaviť algoritmy na vykreslenie základných geometrických útvarov. Pre riešenie príkladu postupujte podľa nasledujúceho postupu:

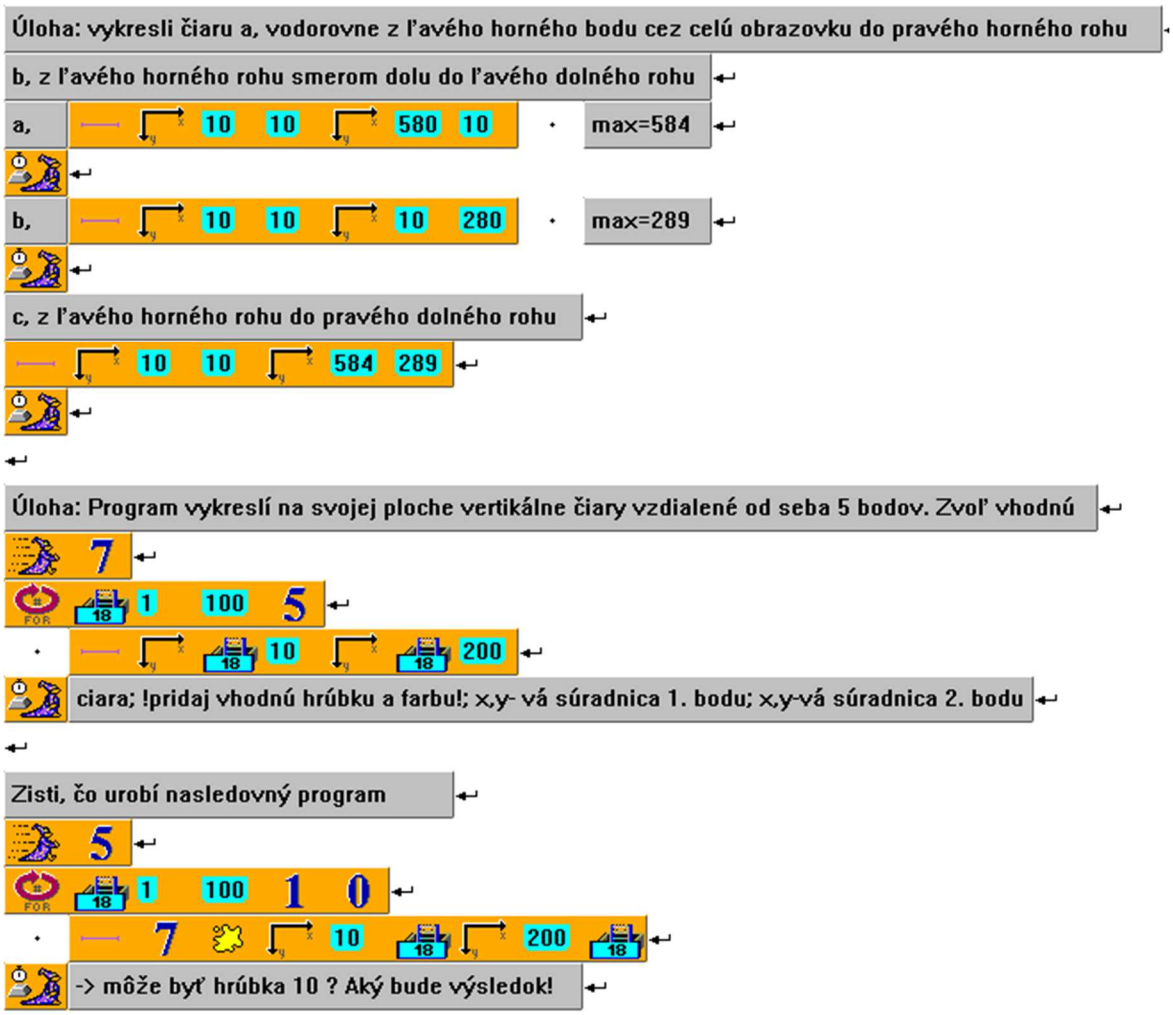
a) Baltík vykreslí čiaru vodorovne z ľavého horného horu cez celú obrazovku do pravého horného horu
b) Batlík vykreslí čiaru zvisle z ľavého horného horu cez celú obrazovku do ľavého dolného horu
c) Baltík vykreslí čiaru z ľavého horného horu cez celú obrazovku do pravého dolného horu ľubovoľnej hrúbky a farby
d) Baltík vykreslí vertikálne čiary vzdialené od seba 5 bodov
e) Baltík vykreslí vertikálne čiary vzdialené od seba 10 bodov

Obrázok 65: Riešenie príkladu 7 z bodov d a e.

(Zdroj: Autorka)

Didaktická reflexia:

Uvedený príklad slúži okrem precvičovania grafických príkazov aj na precvičovanie príkazu cyklus so známym počtom opakovaní (for), kde je pri vykresľovaní možné využiť aktuálnu hodnotu riadiacej premennej cyklu pri nastavení súradníc. Na obrázku 65 sa nachádza možný zdrojový kód k riešeniu príkladu 7.



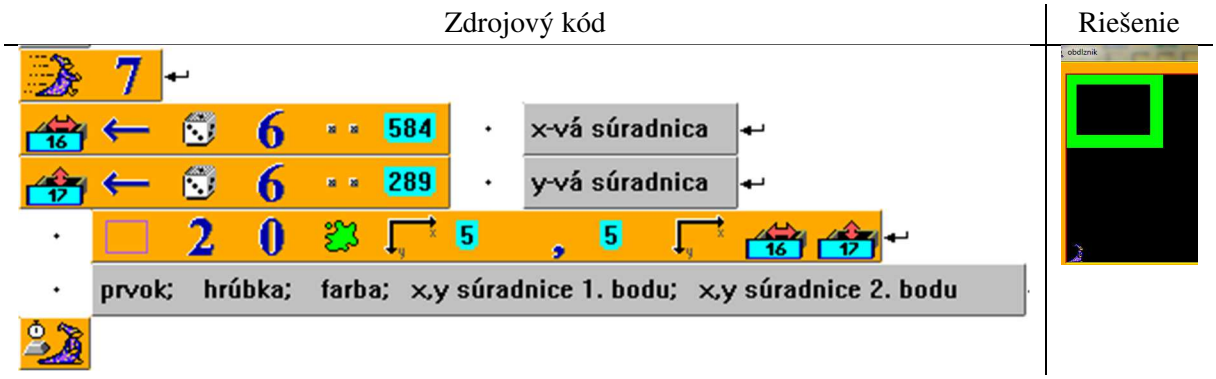
Obrázok 66: Zdrojový kód príkladu 7

(Zdroj: Autorka)

Príklad 8: Obdĺžnik

Na obrázku 67 je zobrazený zdrojový kód a realizácia tohto zdrojového kódu zobrazená na Baltíkovej ploche. Je potrebné navrhnúť algoritmus, ktorý zabezpečí, aby objekt detského programovacieho jazyka nakreslil obdĺžnik podľa zadaných pokynov:

1. Ľavý horný vrchol obdĺžnika má zadané súradnice x=5; y=5,
2. Súradnice pravého dolného vrcholu sú generované náhodne vždy tak, aby bol obdĺžnik zobraziteľný na Baltíkovej grafickej ploche. (tzn. x≤584 a y ≤289)

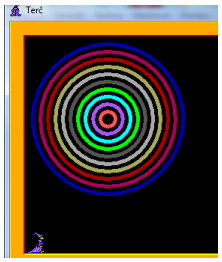


Obrázok 67: Zdrojový kód a riešenie príkladu 8.

(Zdroj: Autorka)

Príklad 9: Baltíkov obraz

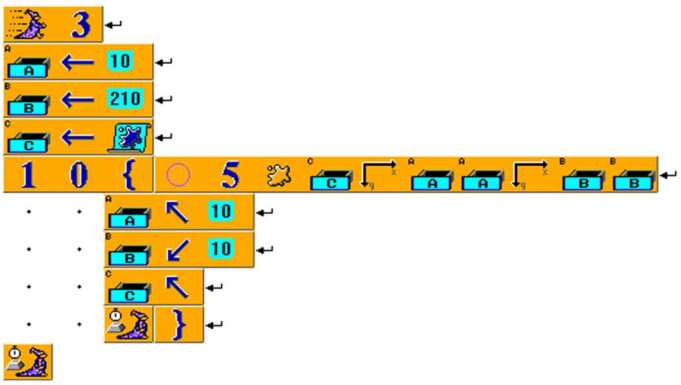
Na obrázku číslo 82 je znázornený obraz, ktorý výmyslom fantázie Baltíka. Pomôžte mu zostaviť algoritmus, pomocou ktorého bude vedieť obraz vyčarovať.



Obrázok 68: Predloha k príkladu 9

(Zdroj: Autorka)

Riešenie algoritmu, ktorý čaruje je na obrázku číslo 69



Obrázok 69: Zdrojový kód príkladu 9.

(Zdroj: Autorka)

Didaktická reflexia:

Vyžitie jednoduchých geometrických útvarov na tvorbu algoritmov je vhodné voliť bez ohľadu na programovací jazyk, ktorý je na zápis algoritmu využiteľný. Jednoduché geometrické útvary ako čiara, obdĺžnik a kružnica sú známymi pojmami v edukácii už na prvom stupni základných škôl, dokonca aj v predškolskej príprave, a preto rôzne alternatívy úloh, kde sú tieto obrazce námetmi je možné používať aj vo vyučovaní algoritmizácie a programovania.

Príklad 10: Obrazce

Na základe uvedeného zdrojového kódu v detskom programovacom jazyku Imagine je potrebné na papier nakresliť obrazec. Po vykreslení obrazca zakreslite aj výslednú pozíciu objektu–korytnačky.

Poznámka autorky: Na obrázku 70 sú uvedené zdrojové kódy a riešenia štyroch rôznych zadaní obrazcov zdrojových kódov.

Zdrojový kód	*Riešenie zadania*
Zadanie A nechFp ? nechHp 10 opakuj 8 [do 50 bod 35 vz 50 vl 45]	
Zadanie B nechFp ? nechHp ? opakuj 4 [do 45 bod 50 ph do 40 pd vp 90]	
Zadanie C nechHp 11 nechFp ? opakuj 8 [vl 90 do 40 vp 90 do 40 vl 45 nechFp ?]	
Zadanie D nechHp 11 opakuj 8 [do 50 bod 30 vp 45 nechFp ?]	

Obrázok 70: Zdrojové kódy a riešenia prikladu 10

(Zdroj: Autorka)

Didaktická reflexia:

Príklad 10 môže slúžiť podobne ako príklady 1 a 2 na učenie sa programovania na základe čítania zdrojového kódu algoritmu.

Príklad 11: Kolotoče

Na základe predlohy zapíšte postupnosť príkazov (algoritmus) procedúry *stvorec*. Pri zápise algoritmu procedúry *stvorec* je nutné dodržať nasledovné údaje:

1. hrúbka pera 5,
2. dĺžka strany štvorca 25,
3. po jeho vykreslení skončí v ľavom dolnom rohu štvorca, otočená nahor,
4. využite príkaz *opakuj*.

Obrazce sú známe z dostupnej literatúry a sú vo všeobecnosti využívané pri vyučovaní programovania v detskom programovacom jazyku Imagine.

Predloha	*Zdrojový kód*
	**Predloha A** viem *obrazok* nechFp ? nechHp 5 opakuj 8 [do 40 vl 45 *stvorec* vp 45 vz 40 vp 45 nechFp ?] koniec
	**Predloha B** viem *obrazok* nechFp ? nechHp 5 opakuj 8 [do 40 *stvorec* vz 40 vp 45 nechFp ?] koniec
	**Predloha C** viem *obrazok* nechFp ? nechHp 5 opakuj 8 [do 50 vp 45 *stvorec* vl 45 vz 50 vp 45 nechFp ?] koniec
	**Predloha D** viem *obrazok* nechFp ? nechHp 5 opakuj 8 [*stvorec* vp 45 do 40 nechFp ?] koniec

Obrázok 71: Predlohy a zdrojové kódy príkladu 11

(Zdroj: Autorka)

Pre všetky predlohy je spoločná procedúra *stvorec*, ktorej zdrojový kód je nasledovný

```
viem stvorec
opakuj 4 [do 25 vp 90]
koniec
```

Didaktická reflexia:

Procedúra *stvorec* je rovnaká pre všetky obrazce. V príklade je ukázaná, nutnosť voliť vhodne algoritmus procedúry, aby bola využiteľná vo všetkých procedúrach, ktorých zdrojový kód je uvedený v rámci riešení k jednotlivým predlohám.

Z uvedených riešených príkladov môže čitateľ čerpať inšpiráciu pri tvorbe vlastných zadaní v príslušnom detskom programovacom jazyku.

7 Didaktické aspekty vyučovania programovania v mikrosvetoch

Didaktika programovania je neoddeliteľnou súčasťou didaktiky informatiky, vyučuje sa na pedagogických fakultách v príprave budúcich učiteľov informatiky. Obsah didaktiky programovania je primárne zameraný na prípravu budúcich učiteľov na vyučovanie tematického celku Algoritmické riešenie problémov. V príprave budúcich učiteľov informatiky je preto nevyhnutné venovať sa vyučovaniu propedeutiky programovania v detských programovacích jazykoch a didaktike programovania vo vyšších programovacích jazykoch, ktoré sú odporúčané na vyučovanie základov programovania na stredných školách.

Názvom didaktika programovania je označovaná taká disciplína, ktorá je zameraná na nájdenie optimálneho spôsobu pri vyučovaní nielen tvorby algoritmov, ale aj ich prepisu do programovacích jazykov. Didaktika programovania sa zaoberá spoločnými problémami vyučovania algoritmizácie a programovania. Jej obsahom je zároveň objasniť príčiny ťažkostí žiakov s porozumením algoritmov, s ich tvorbou alebo so samotným programovaním. Cieľom didaktiky programovania je snaha o objavovanie zákonitostí a pravidiel, ktoré by prispeli k jednoduchšiemu, zrozumiteľnejšiemu a efektívnejšiemu vyučovaniu informatiky a programovania. (Drlík & Hvorecký, 1992).

Pre vyučovanie algoritmizácie a programovania je dôležitý konštruktivistický prístup. Pri aplikácii konštruktivistického prístupu je asi najdôležitejšie, že námety a nápady je možné takmer okamžite vyskúšať a vizualizovať, nakoľko pre žiakov je nesmierne dôležitá predstavivosť. V prípade detských programovacích jazykov je vizualizácia rešpektovaná. V interaktívnom režime žiak okamžite vie, čo objekt ako vykonávateľ algoritmu zapísaného zdrojovom kóde robí, a v programovacích režimoch po dopísaní kódu a po jeho spustení objekt celkovú realizáciu zapísaného algoritmu vizualizuje.

V detskom programovacom jazyku Imagine Logo pri tvorbe algoritmu sú príkazy zapisované do príkazového riadku a výsledok realizácie algoritmu je vidieť na pracovnej ploche ihneď po odoslaní príkazov po stlačení klávesu Enter V detskom programovacom jazyku Baltík (ak neberieme do úvahy interaktívny režim Čaruj scénu) v programovacom režime po zložení zdrojového kódu algoritmu z ikonických príkazov na učenej pracovnej ploche a po stlačení klávesu *spusti,* vidieť na obrazovke zobrazenú Baltíkovu pracovnú plochu pozostávajúcu zo 150 štvorčekov, na ktorej čarodejník vykonáva príkazy zapísané v zdrojovom kóde.

Vďaka týmto možnostiam získavajú žiaci spätnú väzbu a, samozrejme, cenné skúsenosti vizualizáciou fungovania jednotlivých príkazov. Na vyššie spomenutých príkladoch môžeme vidieť, že v detských programovacích jazykoch sa stráca hranica medzi tvorbou či zápisom algoritmov, ktoré riešia zadané úlohy, a samotným programovaním (Stoffová & Czakoová, 2016).

V súčasnosti pri vyučovaní programovania sa kladie dôraz na to, aby žiaci mali možnosť čo najskôr zapisovať zdrojový kód v samotnom programovacom prostredí a týmto si overovali správnosť ich uvažovania pri zapisovaní algoritmov. V minulosti sa programovanie či skôr návrh algoritmov začínal vždy grafickým zápisom algoritmov pomocou vývojových diagramov a neskôr pomocou štruktúrogramov či akéhosi pseudokódu špeciálneho algoritmického jazyka zapisovaného v rodnom jazyku, ktorý využíval jazykové konštrukcie programovacieho jazyka.

Dnes považujeme za potrebné naučiť začiatočníkov najskôr pracovať v samotnom prostredí a následne ich nechať zapisovať prvé kódy v interaktívnom režime (ak ho detský programovací jazyk má) a potom prejsť do samotného programovacieho režimu. Toto je spôsob, akým čo najlepšie pomôcť žiakom pochopiť programovanie.

Z pohľadu informatiky ako vednej disciplíny je možné považovať vyučovanie programovania na školách bez ohľadu na zvolený programovací alebo detský programovací jazyk za veľmi potrebné. Jedným z dôvodov je, že pri riešení algoritmických problémov a programovaní môžu žiaci získať veľké množstvo poznatkov a veľa cenných skúseností s rozmanitými úlohami, s ktorými sa táto vedná disciplína zaoberá a ktoré sú využiteľné v medzipredmetových vzťahoch.

Vyučovanie algoritmizácie a programovania je zakotvené v Štátnom vzdelávacom programe predmetu informatika, ktorý je súčasťou vzdelávacej oblasti Matematika a práca s informáciami. V predmete informatika je pre všetky stupne škôl patriace do regionálneho školstva tematický celok Algoritmické riešenie problémov. Významné postavenie algoritmizácie a programovania je zakotvené aj v cieľových požiadavkách na maturitnú skúšku z predmetu informatika: Dôležitosť odpovedí na otázky z tematického celku Algoritmické riešenie problémov je zdôraznená aj tým, že tvoria až 70 percent hodnotenia odpovede žiaka z celkovej odpovede z predmetu informatika. Na základe vlastných skúseností je možné konštatovať, že programovanie patrí medzi náročné tematické celky.

Nakoľko dnes sú počítače bežným technickým vybavením už v materských školách a stávajú sa samozrejmou súčasťou života detí. Už na prvom stupni základných škôl sa podľa Štátneho vzdelávacieho programu vyučuje predmet informatika. A práve tu majú svoje miesto detské programovacie jazyky, kde sa žiak učí základom programovania. Detské programovacie jazyky musia rešpektovať určité schopnosti a osobitosti detí v danom veku, napríklad na vyučovanie algoritmizácie a programovania je potrebná znalosť písania. Nielen zo skúseností, ale podľa Piageta (1997) aj z vývoja dieťaťa vieme, že deti do 12 rokov sú schopné rozmýšľať a vykonávať operácie s konkrétnymi predmetmi, jedná sa o tzv. obdobie konkrétnych operácií. Abstraktné myslenie, tzv. obdobie formálnych operácií, vo vývoji dieťaťa nastupuje a rozvíja sa až v neskoršom veku. Aj to je dôvod toho, že prostredia detských programovacích jazykov vyzerajú veľmi odlišne od prostredí vyšších programovacích jazykov, ktoré sú určené pre dospelých a profesionálnych programátorov.

Pre vyučovanie základov programovania je dôležité poznať výhody a nevýhody rôznych programovacích jazykov a vybrať ten najvhodnejší pre vyučovanie základov programovania. Výber vhodného detského programovacieho jazyka je veľmi dôležitý najmä, ak sa jedná o žiakov, ktorí získavajú prvé skúsenosti s programovaním. Nemenej dôležitý je aj nadhľad na využívané detské programovacie jazyky pre učiteľa informatiky. Rovnako je potrebné klásť veľký dôraz na to, aby žiak postupne a najmä po malých krôčikoch získaval veľa skúseností. Je preto nutné poskytnúť dostatočný čas a voliť vhodnú následnosť úloh od jednoduchých zadaní po zložitejšie. Postupné zvyšovanie náročnosti zadaní a problémov a pozitívna motivácia by mali byť považované za samozrejmosť. Jednou z možností pozitívnej motivácie u žiakov je využiť ich prirodzenú súťaživosť. Najmä u žiakov, ktorí sú mimoriadne nadaní a majú enormný záujem o programovanie ako také. Veľkým benefitom je aj to, že si môžu porovnávať svoje zručnosti a vedomosti na rôznych súťažiach.

8 Súťaže v programovaní

Predmetovo orientované súťaže vo všeobecnosti sú považované za veľmi potrebnú súčasť vzdelávania. Súťaže bez ohľadu na vyučovací predmet a vek žiakov sú pre žiakov nepovinné. Autori súťaží sa v prevažnej miere pri tvorbe zadaní či úloh zameriavajú na zvyčajne oveľa zložitejšie a zvyčajne aj oveľa komplexnejšie úlohy ako tie, ktoré žiaci riešia na hodinách a to bez ohľadu na to o aké súťažné kolo sa jedná. Väčšinou sú tieto úlohy náročnejšie na vedomosti žiakov.

Informatické súťaže považujeme nielen za možnosť komparácie vedomostí žiakov, ale do značnej miery aj za spôsob popularizácie informatiky a informatizácie ako takej. Nakoľko sa súťaží zúčastňujú žiaci, ktorí na svojich školách spravidla nemajú konkurenciu, nielen medzi spolužiakmi, ale častokrát ani vo svojom vyučujúcom. Títo žiaci práve na súťažiach môžu zažiť veľmi odlišnú atmosféru ako na hodinách informatiky. Na súťažiach, najmä na vyšších postupových stupňoch ako je školské kolo, sa stretávajú so svojimi rovesníkmi, s ktorými môžu porovnávať svoje zručnosti a vedomosti, a práve toto môže byť pre žiakov nesmierne motivujúce. Samozrejme, víťazi a úspešní riešitelia zažívajú aj zadosťučinenie v poznaní pocitu úspechu alebo víťazstva. Pri riešení zadaní na súťažiach získajú zvyčajne žiaci aj množstvo nových poznatkov a skúseností, ktoré len veľmi zriedka majú šancu získať u svojich učiteľov na školách. Zadania na súťažiach sú zvyčajne orientované na talentovaných a samostatne sa vzdelávajúcich žiakov. Predmetové súťaže v informatike sa delia na dve kategórie, a to na súťaže zamerané na informatické zručnosti a na súťaže zamerané na programovanie.

Zoznam súťaží zameraných na programovanie prípadne programovanie robotov s vekovou kategóriou žiakov je v tabuľke číslo 3.

Tabuľka 3: Zoznam súťaží so zameraním na programovanie

Názov súťaže	
Základné školy	*Stredné školy*
Palma junior	Liaheň
RoboCup Junior	Olympiáda v informatike
Cologobežka	Korešpondenčný Seminár z Programovania
ImagineCup	Palma
Miniprog	Zenit
First Lego League	Istrobot
Baltík[4]	ProFIIT
ScratchCup	USACO
Python Cup	Topcoder
Prásk	Haluz
	RoboCup Junior
	Baltík

[4] Súťaže v detských programovacích jazykoch Baltík a Baltie sú vhodné pre žiakov základných aj stredných škôl.

Olympiáda v informatike (OI)

Za najprestížnejšiu a zároveň aj za najnáročnejšiu súťaž v programovaní je považovaná Olympiáda v informatike. Je to súťaž s dlhoročnou tradíciou. Najvyššia úroveň súťaže je ukončená medzinárodným kolom, ktoré sa každoročne koná v inej krajine sveta.

Vyhlasovateľom OI pre daný ročník je Ministerstvo školstva Slovenskej republiky v garancii a v spolupráci so Slovenskou informatickou spoločnosťou a Slovenskou komisiou olympiády v informatike. OI je individuálnou súťažou žiakov stredných škôl riešiacich úlohy z informatiky, ktoré svojou formuláciou motivujú žiakov k tvorivej činnosti. Prvý ročník súťaže bol organizovaný v školskom roku 1985/86. Olympiáda v informatike sa každoročne usporadúva v dvoch kategóriách:

- **kategória A:** je určená pre žiakov tretieho a štvrtého ročníka stredných škôl príslušných ročníkov viacročných gymnázií a má tri kolá: domáce, krajské a celoštátne;
- **kategória B**: je určená pre žiakov prvého a druhého ročníka stredných škôl a príslušného ročníka viacročných gymnázií a má dve kolá: domáce a krajské.

V každej kategórii žiaci riešia dva typy úloh. Prvý typ úloh je teoretický, tieto úlohy zvyčajne pozostávajú zo zápisu programu, opisu algoritmu a aj zdôvodnenia správnosti riešenia. V druhom type úloh, v praktických úlohách súťažiaci odovzdávajú program v elektronickej podobe.

PALMA junior

Palma junior je súťaž v programovaní, ktorej prvé kolá boli realizované v detskom programovacom jazyku Imagine, z ktorého sa postupne v posledných rokoch prešlo do programovacieho jazyka Python. Palma Junior je určená pre žiakov navštevujúcich ôsme a deviate ročníky základných škôl v kategórii EXPERT a prvé až tretie ročníky stredných škôl a k nim príslušné ročníky osemročných gymnázií v kategórii GURU. Súťaž je určená pre jednotlivcov alebo dvojčlenné tímy. Aktuálne je realizovaná priebežnými kolami organizovanými online a ukončovaná finálovým kolom organizovaným prezenčne.

V detskom programovacom jazyku Baltík je organizovaných niekoľko súťaží. Typovo sú rozdelené na

- Prípravné, v ktorých nie je ohraničený čas pre riešenie úloh. Do tohto typu súťaží patrí Junior B3;
- Tvorivé, v ktorých je daná téma ako napríklad Vianoce, Veľká noc, prípadne bez tematického zamerania Creative Balie,
- Predmetové - postupová súťaž s ohraničeným časom pre riešenie je ohraničený (*Baltie xx*[5]).

[5] xx - zodpovedá aktuálnemu kalendárnemu roku, v ktorom je súťaž organizovaná

Súťaže Baltie xx sú organizované v postupových kolách, a súťažný server umožňuje zobraziť výsledky pre zaregistrovaných žiakov:

- školské
- okresné
- krajské
- národné
- medzinárodné.

Predmetová súťaž Baltie xx je organizovaná v piatich kategóriách, kde sú žiaci radení podľa ročníka, ktorí navštevujú:

Súťažné kategórie sú rozdeľované podľa triedy, ktorú súťažiaci navštevuje. V súťaži sú triedy označované ako grade **0 – 18**, kde grade 1 zodpovedá prvej triede základnej školy, grade 10 prvému ročníku strednej školy a 14 prvému ročníku vysokej školy. Kategórie sú rozdelené nasledovne:

A: grade 0-3 **B**: grade 4-6 **C**: grade 7-9 **D**: grade 10-13 **E**: grade 14-18

Vo všetkých súťažiach súťažiaci môžu programovať vo všetkých aktuálnych verziách detských programovacích jazykov Baltík a Baltie. Čas vymedzený na riešenie úloh začína v školskom kole s 90 minútovou dotáciou na celé kolo a končí v medzinárodnom kole 150 minútovou dotáciou spoločne na všetky riešené úlohy kola. V každom kole sú úlohy spoločné pre všetky vekové kategórie a ich náročnosť sa stupňuje. V každej kategórii sú stanovení postupujúci či víťazi. Vzhľadom na stupňujúcu sa náročnosť zadaní je možné očakávať, že súťažiaci kategórie A nedokážu vyriešiť všetky úlohy súťaže.

Záver

V publikácii bola upriamená pozornosť na kvalitatívnu komparáciu uvedených programovacích jazykov. Do komparácie sme postavili šesť oblastí v detských programovacích jazykoch Imagine, Baltík 3.0 a Baltie 4C# a to hlavný objekt programovacieho jazyka, spôsob zadávania príkazov, režimy programovacieho jazyka a ich vhodnosť využitia v závislosti od veku a znalostí žiaka, grafickú plochu v detských programovacích jazykoch a vybrané programovacie štruktúry. Z programovacích štruktúr boli v publikácii opísané cyklus so známym počtom opakovaní, podprogramy a rekurzia.

Podstatné rozdiely v týchto detských programovacích jazykoch je možné vidieť v spôsobe zadávania príkazov v mikrosvete Baltie 4C# a Baltík 3.0, kde sa príkazy zadávajú pomocou ikon. Tieto programovacie jazyky majú medzi ostatnými programovacími jazykmi výnimočné postavenie a zároveň v oboch existuje viacero režimov, ktoré sú prispôsobené veku a znalostiam žiaka.

Porovnaním programových štruktúr sme prišli k záveru, že výučbové programovacie jazyky jednak obidve verzie detského programovacieho jazyka Baltík, ale aj detského programovacieho jazyka Imagine, poskytujú podobné možnosti rozvoja algoritmického myslenia u žiakov. Máme za to, že efektívnosť výučby závisí od schopnosti učiteľa vzbudiť žiakov záujem o danú problematiku, od použitia vhodných metód a tvorivého prístupu učiteľa, ktorý uplatní pri výklade a precvičovaní učiva.

Na základe skúseností autorky získaných počas pedagogickej praxe v regionálnom školstve môže konštatovať, že je veľmi ťažké tvrdiť, ktorý detský programovací jazyk je lepší pre rozvoj algoritmického myslenia. V pedagogickej praxi je veľmi dôležité oceniť prácu učiteľa, pretože je významným a neopomenuteľných činiteľom vo vyučovacom procese. Aj preto je veľmi dôležité učiteľov, a to najmä informatiky, motivovať k ďalšiemu vzdelávaniu, nakoľko v informatike vývoj neustále napreduje. K zefektívneniu práce učiteľa by okrem jeho ďalšieho vzdelávania mohlo viesť vytvorenie dostupných metodických materiálov, ktoré by mohol využívať pri vyučovaní a svojej príprave na neho, dostupnosť zbierok úloh či námetov s možnými riešeniami a pod.

Je veľmi dôležité, aby motivovaný učiteľ dokázal prilákať svojich žiakov, aby sa venovali algoritmizácii a programovaniu, nakoľko sú veľmi nápomocné k rozvoju logického myslenia, naučia dieťa trpezlivosti a podporujú aj jeho ctižiadosť.

Zvoliť si vhodný detský programovací jazyk tak, aby bol pútavý pre žiakov je azda najdôležitejším momentom. Spôsob zápisu algoritmov sa mení v závislosti od jazyka, ktorý si zvolíme. Aj v prípade práce v multijazykovom prostredí existujú detské programovacie jazyky, ako napríklad Scratch, s ktorými žiaci dokážu bez väčších problémov pracovať vo viacerých jazykoch, nakoľko podľa zvolenej jazykovej mutácie je kód zobrazený vo zvolenom jazyku bez ohľadu na to, v ktorom národnom jazyku bol zapísaný. V prostredí detských programovacích jazykov je vhodné používať ikonický jazyk, kde sú algoritmy zapisované príkazmi znázornenými pomocou ikoniek. Význam týchto ikonami zapísaných príkazov korešponduje s príkazmi v programovacom jazyku, v prípade programovacieho jazyka Baltík je to programovací jazyk C#.

Pri vyučovaní programovania rovnako ako aj pri výučbe iných predmetov je dôležité dbať aj na to, aby pri vysvetľovaní bol žiakovi dostatočne vysvetlený problém a bolo vhodne použité prirovnanie k problému v skutočnom svete. Až keď je viditeľné, že žiak pochopil čo má robiť môže pristúpiť sa samotnej tvorbe algoritmu v detskom programovacom prostredí a následne môže tento personifikovať a preniesť sa tak do virtuálneho sveta.

Pri vyučovaní základov programovania je veľmi dôležité, aby učiteľ využíval rovnaké predmety, či symboly, ktoré predtým zadefinoval v skutočnom svete. V prípade, že pristúpi k samotnému programovaniu na počítači, je potrebné, aby bol počítač používaný ako trpezlivý pomocník. Trpezlivý pomocník nikdy nevybuchne hnevom ani v prípade, ak žiak zadáva opakovane chybné príkazy. Rovnako je veľmi dôležité vybrať vhodný detský programovací jazyk, aby boli základy programovania zrozumiteľné, ak nie pre všetkých, tak aspoň pre väčšinu zúčastnených. Je potrebné mať na zreteli, že detský programovací jazyk je intuitívny a skladá sa z pomerne jednoduchých príkazov. Samozrejme, je vhodné, ak má učiteľ pri voľbe detského programovacieho jazyka dbá nato, v akom programovacom jazyku budú žiaci programovať následne. Aj vzhľadom na tento fakt je vhodné sa rozhodnúť, samozrejme za predpokladu, že učiteľ nemá problém pracovať v akomkoľvek na školách využívanom detskom programovacom jazyku. A ak jeho voľba padne na taký, ktorý má príbuznú syntax s vyšším programovacím jazykom, uľahčí žiakovi prechod do programovania v ňom. Veľmi dôležité je aj to, aby detský programovací jazyk dokázal udržať záujem žiaka o programovanie aj v prípade, že žiak ešte nemá dobre rozvinuté abstraktné myslenie, a aj v prípade, ak sa jedná o slabšie prospievajúceho žiaka. Práve toto je jedna z najťažších úloh akéhokoľvek detského programovacieho jazyka vhodného pre vyučovanie základov programovania. Rovnako je výhodou detského programovacieho jazyka aj to, ak dokáže pracovať na všetkých -zariadeniach či už na počítači, tablete či smartfóne...

Literatúra

ADAMS, J. C. – WEBSTER, A. R. 2012. *What do students learn about programming from game, music video, and storytelling projects*? In *SIGCSE '12: Proceedings of the 43rd ACM technical symposium on Computer Science Education,* p. 643-648. Dostupné na internete: < https://doi.org/10.1145/2157136.2157319>.

BAJTOŠ, J. 2003. *Teória a prax didaktiky.* 1. vyd. Žilina : Žilinská univerzita, 2003. 384 s. ISBN 80-8070-130-X.

BAJÚSZOVÁ, Z. 2015. *Metody pro výuku programování na základních školách* : bakalárska práca. Zlín : Univerzita Tomáše Bati ve Zlíne, 2015. 79 s.

BELAN, A. 2004. *Imagine učebný text pre terciu osemročného gymnázia.* [online]. Bratislava: Škola pre mimoriadne nadané deti a gymnázium, 2004. [cit. 20.02.2021]. Dostupné na internete: <http: //www.smnd.sk/anino/moje/I.pdf>.

BLAHO, A. 2001. *Imagine – nové Logo.* [online]. Bratislava, 2001. [cit. 23.03.2022]. Dostupné na internete: <virtual.fpv.umb.sk/zbornik/zb2001/Blaho.pdf>.

BLAHO, A. 2002. *Imagine – charakteristika.* [online]. Bratislava, 2002. [cit. 24.04.2022]. Dostupné na internete: <http://www.imagine.input.sk/popis.htm>.

BLAHO, A. – KALAŠ, I. 2005. *Tvorivá informatika. 1. zošit z programovania*. Bratislava: SPN, 2005. 48 s. ISBN 80-10-00019-1.

BLAHO, A. – SALANCI, Ľ. – ŠIMANDL, V. 2018. *Základy programování v jazyce Python pro střední školy.* [online]. [cit. 18.05.2022]. Dostupné na internete: <https: //imysleni.cz/ucebnice/zaklady-programovani-v-jazycepython-pro-stredni-skoly>.

CÁPAY, M. 2017. *Objavovanie na hodinách informatiky.* In: *Blog.vzdelavameprebuducnost.sk* [online]. [cit. 27. 02. 2022]. Dostupné z: http: //blog.vzdelavameprebuducnost.sk/komunita/miee/objavovanie-na-hodinach-informatiky/.

CZAKÓOVÁ, K. *Felfedezésen alapuló aktív tanulás mikrovilág környezetben. In STOFFOVÁ, V. (eds.) - ZSAKÓ, L. (eds.) - SZLÁVI, P. (eds.) XXIXth DIDMATTECH 2016 : new methods and technologies in education and practice : [international scientific and professional conference, Budapest 25th - 26th August 2016]. 1st part. - 1. vyd. - Budapest: Eötvös Loránd University in Budapest - Faculty of Informatics, 2016. - ISBN 978-963-284-799-3, s. 168-173.*

CZAKÓOVÁ, K. –STOFFOVÁ, V. 2020. *Hravá forma rozvíjania algoritmického myslenia na základnej škole = A Playful Form of Developing Algorithmic Thinking in Primary School* In: Didinfo 2020 [electronic] : sborník konference : [medzinárodní konference o vyučování informatiky] / [bez zostavovateľa]. - Liberec : Technická univerzita v Liberci, 2020. - ISBN 978-80-7494-532-8. - ISSN 2454-051X. - online, s. 104-111.

DRLÍK, P. 1995. *Informatika a výpočtová technika*. Bratislava : ENIGMA Jr., 1995. 204 s. ISBN 80-967190-4-1.

DRLÍK, P. – HVORECKÝ, J. 1992. *Informatika – náčrt didaktiky*. Nitra: VŠPg v Nitre,1992. 165 s. ISBN 80-85183-81-1.

GABAĽOVÁ, V. 2008. *Mikrosvet – vhodný prostriedok na vyučovanie základov programovania*. In: *Zborník DidInfo2008*. Banská Bystrica : Fakulta prírodných vied UMB, 2008, s. 16. ISBN 978-80-8083-367-1.

GABAĽOVÁ, V. 1997. *Problémové vyučovanie programovania* : dizertačná práca. Nitra: FPV UKF Nitra, 2010.

GUNIŠ J. a kol. 2009. *Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika.* 1. vyd. Bratislava : Štátny pedagogický ústav, 2009. 40 s. ISBN 978-80-89225-96-5.

HEJNÝ, M. A kol. 1990. *Teória vyučovania matematiky*. 2. vydanie. Bratislava : SPN, 1990.

HEJNÝ, M. 2003. Understanding and structure. In E*uropean Research in Mathematics Education III.* [online]. 2003, č. 3 [cit. 04.05.2022]. Dostupné na internete: < http://www.mathematik.tu-dortmund.de/~erme/CERME3/Groups/TG3/TG3_Hejny_cerme3.pdf>.

HLAVATÁ, E. et al. 2008. *Základy programovania pre Baltie 4C #.* [online]. 2008 [cit. 19.03.2021]. Dostupné na internete: <http: //www.sgpsys.com/CRM/(S(tfepvo551ncvnk45kldcxp45))/CIB/methodology_detail.aspx?methodologyID=303>.

IVANOVÁ-ŠALINGOVÁ, M. 1993. *Vreckový slovník cudzích slov*. Bratislava : Kniha – spoločník, 1993. 848 s. ISBN 80-901160-2-7.

KOTRBA, T. – LACINA, L. 2007. *Praktické využití aktivizačních metod ve výuce.* Brno : Společnost pro odbornou literaturu, 2007. 188 s. ISBN 978-80-87029-12-1.

KRAUS, J. 2011. Nástroj Petr: programování i pro začátečníky In *Zive.cz* [online] 21.04.2022. [cit. 24.5.2022]. Dostupné na internete : <https://www.zive.cz/clanky/nastroj-petr-programovani-i-pro-zacatecniky/sc-3-a-158822/default.aspx>.

MARJI, M. 2014. *Learn to Program with Scratch: A Visual Introduction to Programming with Games, Art, Science and Math*. San Francisco, California : No Starch Press, 2014. 255 s. ISBN 978-159327-543-3.

MAĆKOWIAK, M. – NAWROCKI, J. – OCHODEK, M. 2018. *On some end-user programming constructs and their understandability* In *Journal of Systems and Software*. [online] 2018, č. 142. [cit. 06.06.2022]. Dostupné na internete : <DOI: 10.1016/j.jss.2018.03.064>.

ONDRIŠKOVÁ, J. 2005. *Rekurzia – metodický list pre gymnázium* in *Infovek.* [online]. 12.07.2005 [cit. 12.08.2022]. Dostupné na internete : <http: //www.infovek.sk/~ondriskova/pisomno/metodika.html>.

PAPERT, S. 1980. *Mindstorms: Children, computers, and powerful ideas*. New York : Basic Books, Inc., 1980. 230 s. ISBN 0-465-04617-4.

PAUCHLY, K. 2002. *Programovacie jazyky* [online]. 2002 [cit. 27.01.2022]. Dostupné na: <http: //www.flatulent.szm.sk/karel/programovanie.html>

PIAGET, J. 1999. *Psychologie inteligence.* Praha: SNP Praha, 1999. 166 s. ISBN 80-7178-309-9.

PIAGET, J. 1997. *The moral judgement of the child*. New York: Simon & Schuster Inc., 1997. ISBN 0-684-83330-1.

PECINOVSKÝ, R. 2001. *Proč učiť programování na základní škole* In *Ceskaskola.cz* [online]. 10. 9. 2001 [cit. 29. 2. 2021]. Dostupné na internete : <http: //www.ceskaskola.cz/2001/09/rudolf-pecinovsky-proc-ucit.html cit 29.2.2020>.

PECINOVSKÝ, R., VÁCHA, J. 2001. *Baltík. Učebnica programovania nielen pre deti.* SGP Systems, 2001. 216 s.

PETLÁK, E. 1997. *Všeobecná didaktika*. Bratislava : Iris, 1997. 270 s. ISBN 80-88778-49-2.

PERRY, N. E. 1998. Young children‘s self-regulated learning and contexts that support it. In: *Journal of Educational Psychology* [online] 1998 č. 90. [cit. 01.08.2022] Dostupné na internete: <DOI: 10.1037/0022-0663.90.4.715>.

PATTIS, R. E. 1994. *Karel the Robot, A Gentle Introduction to the Art of Programming*. 2 vyd. New Jersey : Wiley, 1994. ISBN 0-47159-725-2.

PŠENÁKOVÁ, I: 2021. *Tvorba didaktických interaktívnych materiálov a kritériá hodnotenia ich kvality*. Trnava : Pedagogická fakulta Trnavská Univerzita, 2021. ISBN 978-80-568-0425-4.

SALANCI, Ľ. – TOMCSÁNYIOVÁ, M. – BLAHO, A. 2010. *Didaktika programovania.* Bratislava : ŠPÚ, 2010. 36 s. ISBN 978-80-8118-065.

SALANCI, Ľ. – TOMCSÁNYIOVÁ, M. – BLAHO, A. 2011. *Didaktika programovania pre SŠ 2* . Bratislava : ŠPÚ, 2011. 40 s. ISBN 978–80–8118–090-3.

SALANCI, Ľ. – TOMCSÁNYIOVÁ, M. – BLAHO, A. 2011. *Didaktika programovania pre SŠ 1*. Bratislava : ŠPÚ, 2011. 36 s. ISBN 978–80–8118–079-8.

SALANCI, Ľ. – TOMCSÁNYIOVÁ, M. – BLAHO, A. 2015. *Ďalšie vzdelávanie učiteľov základných škôl a stredných škôl v predmete informatika: Didaktika programovania.* Bratislava: Štátny pedagogický ústav. 36 s. ISBN 978–80–8118–065–1.

STOFFOVÁ, V. – CZAKÓOVÁ, K.: Animačné modely v didaktických aplikáciách vytvorených v LogoMotion . In: *XXX. International Colloquium on the Management of Educational Process* = XXX. Mezinárodní kolokvium o řízení vzdělávacího procesu : aimed at current issues in science, education and creative thinking development : zaměřené k aktuálním problémům vědy, výchovy, vzdělávání a rozvoje tvůrčího myšlení : [proceedings of electronic

version of reviewed contributions]. - Brno : Univerzita obrany, 2012. - ISBN 978-80-7231-865-0. - P.

STOFFOVÁ, V. – CZAKÓOVÁ, K.: *Propedeutika programovania a nová školská reforma = Propedeutic of programming and new school reform* .In: Trendy ve vzdělávání 2012 = TVV 2012 : mezinárodí vědecko-odborná konference - Pedagogická fakulta UP v Olomouci 20. a 21.6.2012 Olomouc : sborník příspěvků z mezinárodní konference / [editoři: Miroslav Chráska a kol.]. - Olomouc : Agentura Gevak, 2012. - ISBN 978-80-86768-36-6. - CD ROM, S. 588-594.

STOFFOVÁ, V. – CZAKÓOVÁ, K.: *Prostredie mikrosveta v práci učiteľa 1. stupňa základnej školy* / Veronika Stoffová, Krisztina Czakóová. In: Edukacja - technika - informatyka = Education - technology - computer science. - ISSN 2080-9069. - Roč. 11, č. 1(2015), s. 281-286.

STOFFOVÁ, V. – CZAKÓOVÁ, K.: *Úvod do programovania v prostredí mikrosvetov* : (Vysokoškolská učebnica). 1. vyd. - Komárno : Univerzita J. Selyeho, Ekonomická fakulta, 2016. - 114 s. - ISBN 978-80-8122-170-5. – Spôsob prístupu: http://ukftp.truni.sk/epc/13632.pdf

STOFFOVÁ, V. – CZAKÓOVÁ, K.: A playful form of teaching and learning using micro-world applications. In: *eLSE 2019 : eLearning and Software for Education Conference*, Bucharest, April 11 - 12, 2019 / [Editors Ion Roceanu et al.]. - ISSN 2066-026X. - S. 110-115.

ŠIŠKOVÁ, J. *Informatické sútaže na Slovensku* in People.ksp.sk [online]. [cit. 26.6.2022]. Dostupné na internete: <http: //people.ksp.sk/~julka/sutaze>.

ŠNAJDER Ľ. – GUNIŠ, J. – GUNIŠOVÁ, V. 2008. *Aktivizujúce metódy v školskej informatike.* [online]. [cit. 23.06.2022]. Dostupné na internete: <https://di.ics.upjs.sk/gunis/praca/prezentacie/didinfo_2008_snajder_gunis_gunisova_aktivizujuce_metody.pdf>.

TUREK, I. 1998. *Zvyšovanie efektívnosti vyučovania*. Bratislava : Edukácia, 1998. 328 s. ISBN 80-88796-89-X.

TWIEHAUS, J. 1988. *Kľúč k počítaču. Programové vybavenie.* Bratislava : Alfa, 1988. 198 s.

VANÍČKOVÁ, V. 1997. *Propedeutika programovania* : diplomová práca. Nitra: FPV UKF Nitra, 1997.

VARGA, M. 1999. *Informatika pre stredné školy.. Algoritmy s Logom*. Bratislava: SPN – Mladé letá, 1999. 48 s. ISBN 80-08-02965-X.

WANDA, D. – COOPER, S. – PAUSCH, R. 2011. *Learning to Program with Alice.* Pearson Learning Solutions, 2009. 355 s. ISBN 0-13-212247-2

ŽIAKOVÁ M. 2009. *Porovnanie mikrosvetov a skúsenosť s nimi vo vyučovaní informatiky* : diplomová práca. Trnava : PdF TU, 2009.

Internetové zdroje:

ROBOT KAREL: vývojové prostředí [online]. 30.7.2013. [cit. 24.6.2022]. Dostupné na internete : <*http: //karel.oldium.net/*>.

PALMA junior, Programovanie, Algoritmy, Matematika, súťaž pre mladých programátorov v Imagine in *Upjs.sk* [online]. [cit. 24.05.2022]. Dostupné na internete: <https://di.ics.upjs.sk/palmaj/>.

Olympiáda v informatike in *Oi.sk* [online]. [cit. 29.6.2022]. Dostupné na internete: http://oi.sk

Scratch – Imagine, Program, Share in *Scratch.mit.edu*. [online] 10.8.2022. [cit. 15.8.2022]. Dostupné na internete : < https://scratch.mit.edu/statistics/>.

Súťaže Baltie in *Baltie.net* [online]. [cit. 30.6.2022]. Dostupné na internete: <https://baltie.net/(S(vfe1wr33qsemmb4ghhkn4roy))/default.aspx>.

O kohútikovi a sliepočke in People.ksp.sk [online]. [cit. 17.08.2022]. Dostupné na internete: https://people.ksp.sk/~kuko/texty/kas.pdf.

Autor: Veronika Gabaľová

Názov: Detské programovacie jazyky a ich využitie v pedagogickej praxi

Pedagogická fakulta, Trnavská univerzita v Trnave

Vydanie prvé

Počet strán 79

Rok vydania: 2022

Monografia vznikla s podporou KEGA MŠVVaŠ SR v rámci riešenia projektu 013TTU4/2021 "Interaktívne animačno-simulačné modely pre deep learning".

ISBN 978-80-568-0510-7